AMATH 582 Homework 2: Classic Rock and Roll Shredding

Emma Nuss, github: emmashie

Abstract

Sweet Child O' Mine by Guns N' Roses and Comfortably Numb by Pink Floyd both have guitar solos that classic rock and roll fans love and attempt to replicate. We use the Gabor transform to perform time-frequency analysis on clips from each of these classic rock and roll songs and isolate the notes played in the guitar and bass lines in these songs. The Gabor transform proves to be an effective method for identifying the guitar and bass notes in these classic rock and roll songs; however, for audio clips with several instruments playing quick short notes concurrently, it can be hard to clearly isolate all of the notes played.

1. Introduction

We aim to identify the guitar and bass parts in two classic rock and roll songs with epic guitar solos. To identify both time and frequency information we use the Gabor transform to isolate the guitar solo in Sweet Child O' Mine by Guns N' Roses and both the bass and guitar parts in Comfortably Numb by Pink Floyd. This report outlines what the Gabor transform is, why it is useful for time-frequency analysis, and how it is applied to identifying these solos.

2. Theoretical Background

2.1. Gabor Transform

In our previous homework we explored the use of the Fourier transform to identify information about frequencies present in spatial or time-series data. Due to the uncertainty principle, while the Fourier transform can provide high resolution information in frequency space, it cannot provide high resolution time information at the same time. The Gabor transform is a method to maximize both frequency and time information. It is defined as

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

where $f(t)$ is the function of interest, τ is a fixed time-shift, and $g(t-\tau)$ is a filter function centered at τ (Kutz, 2013). The filter function, $g(t-\tau)$ must satisfy two properties: 1) the function must be real and symmetric and 2) the L_2-norm of the function must be 1. Many functions satisfy the properties and can be used as the filter function, but for the purposes of this report, the Gaussian filter is exclusively used.

The power of the Gabor transform lies in its ability to capture both frequency and time information. To optimize this balance you must choose a reasonable Gabor window and filter width. Like in our previous work with Gaussian filtering, the width parameter sets how much of the

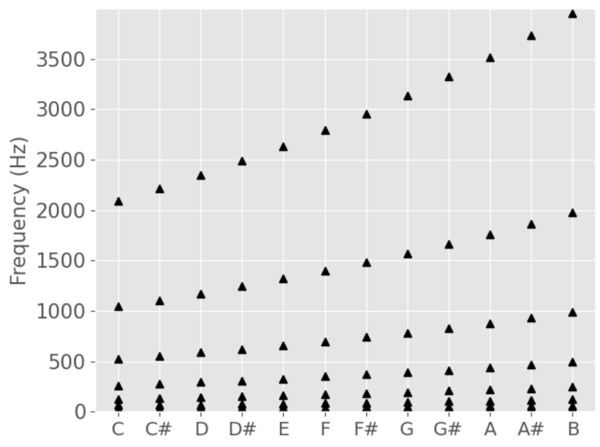

Figure 1: The frequencies associated with musical notes. Frequencies increase from C to B (left to right) and as you increase octaves.

data is maintained versus how much of the data is eliminated by being multiplied by a very small number close to zero. The Gabor window is set by the variable, τ, which determines what part of the time-series data is 'observed' at a time. For example, if a time record is 60 seconds long, you could use τ values of 0, 20, 40, and 60 and the time record would be divided into 20 second chunks. If the window is too large, the Gabor transform essentially returns the Fourier transform (i.e. high frequency resolution, no time resolution); however, if the window is too small, we lose the frequency information but have high time-resolution. For time-frequency analysis, a balance between these two extremes is necessary and relies on the data you are given and the type of signal you are identifying.

The frequency information from each windowed subset of the data can be displayed graphically in a spectrogram. The spectrogram is a graphical representation that shows frequency and time information. It is useful in visualizing time-frequency analysis.

2.2. Music

In music, notes are associated with distinct harmonics. When a person sings or plays an instrument, each note emits a sound wave with a particular frequency where higher notes have higher frequencies and lower notes have lower frequencies. These notes are defined by names (i.e. A, B, G, etc.) associated with particular frequencies given in Hertz (Hz) which have units of s^{-1} (Fig. 1). In a musical composition, different instruments play unique notes that emit specific frequencies at specific times. Audio files, such as mp3 files, encode all the information about which frequencies are emitted at what time and for how long into something we can listen to. These files can also be loaded into a computer software and these frequencies can be analyzed using time-frequency analysis.

3. Algorithm Implementation and Development

The audio files of Sweet Child O' Mine and Comfortably Numb were loaded into Matlab using the audioread function (see Appendix A), saved as .mat files, then read and analyzed in

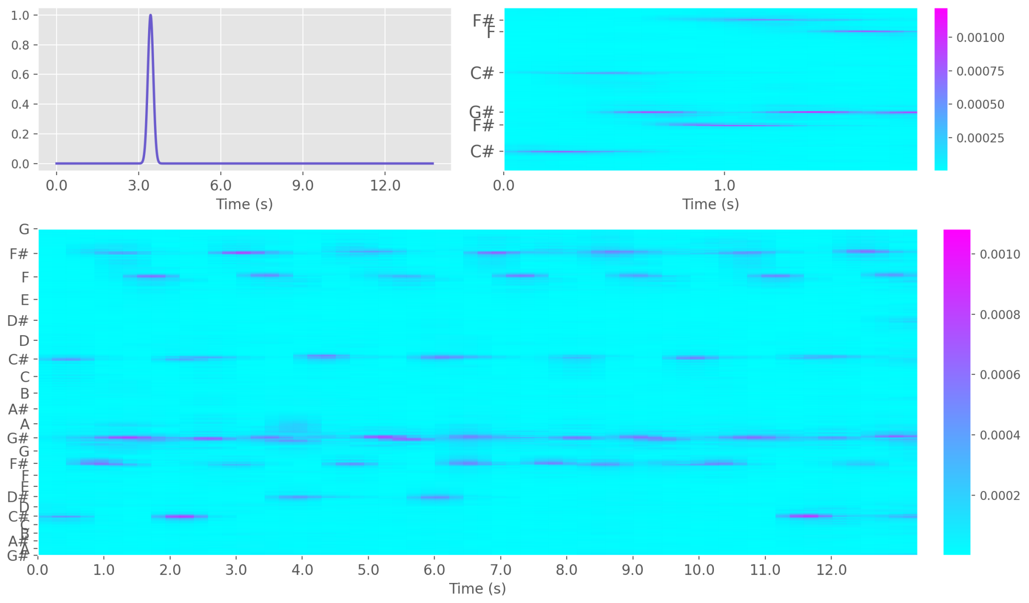

Figure 2: Gaussian filter, with width parameter of 50, used in the Gabor transform (upper left). Spectrograms of the first 2 seconds (upper right) and full 13 second clip (bottom) of Guns N' Roses Sweet Child O' Mine. Guitar notes are identified by higher amplitudes (purple).

Python. Sweet Child O' Mine is a 13.7 second clip that has a sample rate of 48000 s^{-1} and Comfortably Numb is a 59.8 second clip that has a sample rate of 44100 s^{-1}. For audio clips with an odd number of data points, the last time record was omitted to ensure frequency calculations would match. The associated frequencies with the time record were calculated and scaled by $\frac{1}{L}$, where L is the length of the audio clip in seconds, to put frequencies into Hz in order to match with musical notes.

Using the Gabor transform, spectrograms were built. Gaussian and Gabor window widths were chosen by visual inspection of the spectrograms to optimize maintaining both accurate frequency and time information. The specific widths chosen are noted in the next section along with discussion of the results.

To assist in isolating the guitar and bass in both audio clips, frequency filtering was utilized to filter out overtones and frequencies that were not of interest. Once the frequency range of the guitar or bass solos were determined, a boxcar filter centered around the middle of the solo range with a width that encompassed the entire range was applied to the Fourier transformed audio data to filter out extraneous frequencies. The Gabor transform was then reapplied and the spectrograms were reassembled. Lastly, for aid in visual inspection, the color limits on the spectrograms were restricted to the upper range of the amplitudes to allow the notes to contrast background noise in the signal.

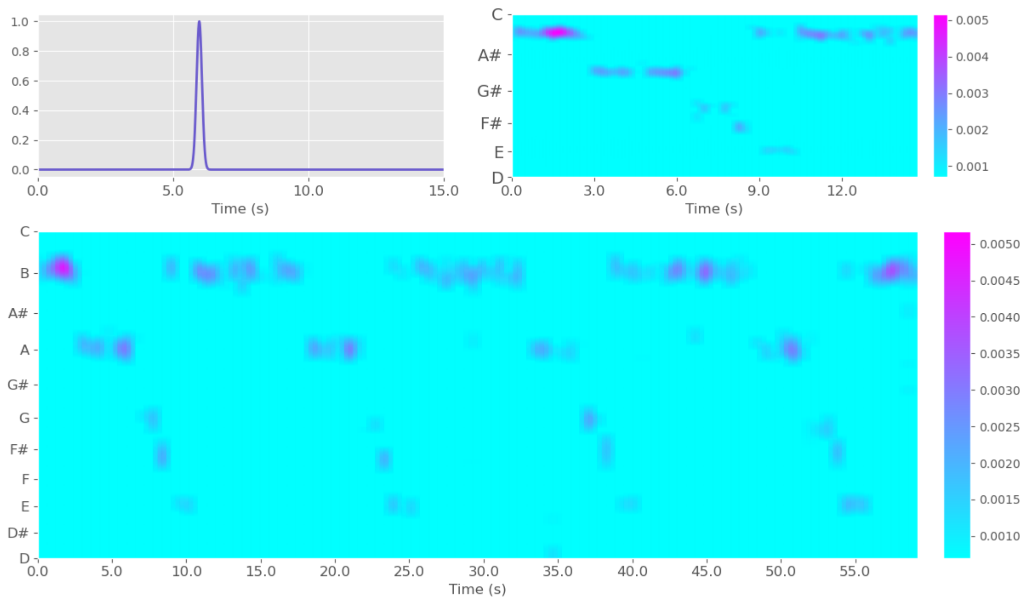

Figure 3: Gaussian filter, with width parameter of 50, used in the Gabor transform (upper left). Spectrograms of the first 15 seconds (upper right) and full 59 second clip (bottom) of Pink Floyd's Comfortably Numb. Spectrogram color limits are set at 0.0005 to the maximum value to visually isolate the notes being played by the bass guitar (purple).

4. Computational Results

4.1. Sweet Child O' Mine

Using a Gaussian filter with a width parameter of 50 and Gabor window widths of $\frac{1}{16}$ of the clip length (Fig. 2), the first measure (approximately 2 seconds) of the opening guitar riff was isolated. There are 8 notes being played in the opening measure that can be seen as elevated amplitudes (purple) in the spectrogram (Fig. 2). The musical notes associated with this opening riff is a sequence of C♯, C♯ (one octave up), G♯, F♯, F♯ (one octave up), G♯, F, and G♯.

This method was applied to the entire clip using a Gaussian filter with a width parameter of 50 and Gabor window widths of $\frac{1}{32}$ of the clip length. This opening sequence of 8 notes is mostly repeated throughout the 12 second clip; however, the sequence is modulated slightly after the second repetition of the sequence. At approximately 4 seconds into the clip, the third repeat of this sequence, the opening note is D♯ instead of C♯. This sequence is repeated twice, and then the sequence is modulated a third time where the first note is now an F♯. This third modulation is repeated twice and then the original sequence is repeated one last time. The repetition of these modulated note progressions can be seen in the spectrogram of the full clip (Fig. 2). Since the guitar is the only instrument being played, the Gabor transform is able to isolate each note clearly without much interference or noise from higher and lower frequencies and other instruments.

4.2. Comfortably Numb

4.2.1. Bass

Using a Gaussian filter with a width parameter of 50 and Gabor window widths of $\frac{1}{60}$ of the clip length (Fig. 3), the first 15 seconds of the bass notes were isolated. To reduce noise and

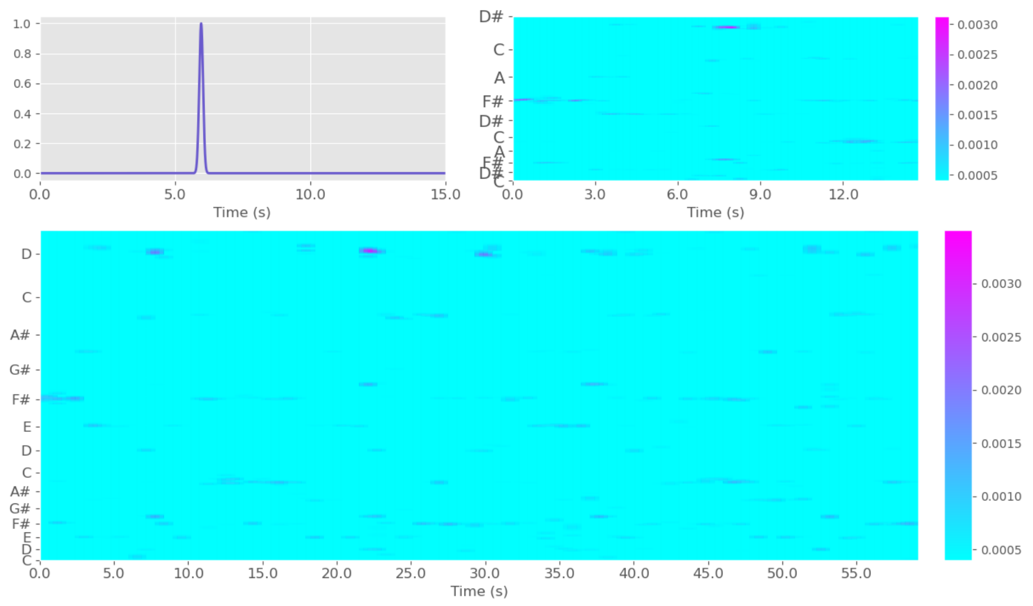

Figure 4: Gaussian filter, with width parameter of 100, used in the Gabor transform (upper left). Spectrograms of the first 15 seconds (upper right) and full 59 second clip (bottom) of Pink Floyd's Comfortably Numb. Spectrogram color limits are set at 0.0004 to the maximum value to visually isolate the notes being played by the bass guitar (purple).

overtones, a boxcar filter was applied in frequency space to isolate the frequency range where the bass notes are present (60 - 150 Hz). The main note progression that the bass plays is identified in the spectrogram (Fig. 3). The sequence of notes associated with the bass line are B, A, A, G, F♯, E, B. Now using a the same method, but with Gabor window widths of $\frac{1}{100}$ of the clip length, we identify the bass line and we can see that the opening note progression is repeated about 4 times throughout the clip (Fig. 3).

4.2.2. Guitar

Using a Gaussian filter with a width parameter of 100 and Gabor window widths of $\frac{1}{60}$ of the clip length (Fig. 3), the first 15 seconds of the guitar solo in Comfortably Numb were identified. To reduce noise and overtones, a boxcar filter was applied in frequency space to isolate the frequency range of a guitar (130 - 700 Hz). This frequency range was chosen focus on the guitar range while excluding the range where the bass notes are present.

The guitar solo in the first 15 seconds of Comfortably Numb is not as clear in the spectrogram as the bass line was; however, some of the notes are identifiable. Most notably, the long note at the beginning of the guitar solo can be seen as a F♯ and the high note hit approximately 8 seconds into the song can be seen as a high D. Unfortunately, the notes between these two notable notes are harder to identify, likely due to the fact that many of the notes in the guitar solo are sixteenth or thirty-second notes (very fast) and so identifying each note distinctly is difficult even by decreasing the Gabor window width. Examining the spectrogram of the entire clip using the same method, but with Gabor window widths of $\frac{1}{100}$ of the clip length, we see that a few notes in the guitar solo are clear, but again many of the notes do not appear clearly.

5. Summary and Conclusions

The Gabor transform proves to be useful for performing time-frequency analysis on rock and roll songs by balancing the amount of time and frequency information retained. Musical notes can be identified by distinct frequencies in the spectrogram. In Sweet Child O' Mine and Comfortably Numb, the guitar and bass lines were identified visually using spectrograms. Identifying the notes being played was more difficult when several instruments were playing at the same time, like in Comfortably Numb, as opposed to a single instrument being played, like in Sweet Child O' Mine.

Appendix A Python Functions

Relevant Python functions:

- `data = pd.read_csv(filename, header=None)`: Reads in data from a csv file with no header using the Pandas python package
- `x = np.arange(0, n)`: Creates a 1D array of integers starting at 0 up to `n-1`
- `z = np.append(x, y)`: Appends array y to the end of array x and stores it in variable z
- `k = np.fft.fftshift(k)`: Shifts the zero-frequency component to the center of the spectrum
- `[X, Y] = np.meshgrid(x, y)`: Creates a 2D grid based on the coordinates contained in the vectors `x` and `y`.
- `Dspec = np.fft.fft(D)`: The 1-dimensional Fourier transform is taken
- `ind = np.argwhere(D == condition) or np.where(D == condition)`: Finds the indices of an array where a logical condition is satisfied
- `np.asarray(D)`: Takes a list or some non-array and converts it to an array type
- `L = [i for i in range(n)]`: (List comprehension) Creates a list of `0` to `n-1`, can be used for high dimensions of lists and with if conditional statements
- `os.path.join(path, filename)`: Creates a path object based on the `path` and `filename` given
- `dat = sio.loadmat(filename)`: Loads a Matlab .mat file into Python
- `audioread(filename)`: (Matlab function) Function that loads an audiofile and converts the data to floating point values and provides the sample rate of the data.

Appendix B Python Code

Code can be found at: https://github.com/emmashie/amath-582

```
import numpy as np
import matplotlib.pyplot as plt
import os
import scipy.io as sio
import scipy.io.wavfile as wav
#plt.ion()
plt.style.use('ggplot')

# definte data path and filename
datapath = '../data/'
floydfile = 'floyd.mat'

floyd_fs = 44100.0

floyd = sio.loadmat(os.path.join(datapath, floydfile))['y']/floyd_fs
floyd = np.squeeze(floyd)[:-1] #remove last time-step to keep even time series
#first_bit = int(floyd_fs*15)
#floyd = floyd[:first_bit]
#start = int(floyd_fs*28)
#end = int(floyd_fs*44)
#floyd = floyd[start:end]
Tf = len(floyd)/floyd_fs
time_floyd = np.linspace(0, Tf, len(floyd))

# clip
start = int(floyd_fs*0)
end = int(floyd_fs*15)
floyd_c = floyd[start:end]
Tf_c = len(floyd_c)/floyd_fs
time_floyd_c = np.linspace(start/floyd_fs, end/floyd_fs, len(floyd_c))


# define time and frequency domains
n = len(floyd)
k_pos = np.arange(0, n/2)
k_neg = np.arange(-n/2, 0)
k = (1/(Tf))*np.append(k_pos, k_neg)
floyd_ks = np.fft.fftshift(k)

# clip
n = len(floyd_c)
k_pos = np.arange(0, n/2)
k_neg = np.arange(-n/2, 0)
k = (1/(Tf_c))*np.append(k_pos, k_neg)
floyd_ks_c = np.fft.fftshift(k)

def filter(a, t, tau):
        return np.exp(-a*(t-tau)**2)

def boxcar(k, ulim, blim):
        bx = np.zeros(len(k))
        bx[np.where((k>blim)&(k<ulim))[0]] = 1
        return bx

# filter data prior to building spectogram
#floyd = np.fft.ifft(np.fft.ifftshift(np.fft.fftshift(np.fft.fft(floyd))*boxcar(floyd_ks, 150, 60)))
```

```
#floyd_c = np.fft.ifft(np.fft.ifftshift(np.fft.fftshift(np.fft.fft(floyd_c))*boxcar(floyd_ks_c, 150, 60)))
floyd = np.fft.ifft(np.fft.ifftshift(np.fft.fftshift(np.fft.fft(floyd))*boxcar(floyd_ks, 700, 130)))
floyd_c = np.fft.ifft(np.fft.ifftshift(np.fft.fftshift(np.fft.fft(floyd_c))*boxcar(floyd_ks_c, 700, 130)))


a = 100
floyd_tau = np.arange(0, Tf, Tf/100)
floyd_spec = np.asarray([np.fft.fftshift(np.abs(np.fft.fft(filter(a, time_floyd, floyd_tau[i])*floyd))) for i in ran
# clip
floyd_tau_c = np.arange(0, Tf_c, Tf_c/60)
floyd_spec_c = np.asarray([np.fft.fftshift(np.abs(np.fft.fft(filter(a, time_floyd_c, floyd_tau_c[i])*floyd_c))) for

[floyd_Tau, floyd_Ks] = np.meshgrid(floyd_tau, floyd_ks)

#clip
[floyd_Tau_c, floyd_Ks_c] = np.meshgrid(floyd_tau_c, floyd_ks_c)


notes_octave = ['C','C#','D','D#','E','F','F#','G','G#','A','A#','B']
freq_octave = [16.35, 17.32, 18.35, 19.45, 20.60, 21.83, 23.12, 24.50, 25.96, 27.5, 29.14, 30.87]

notes = []
freq = []
for i in range(8):
        for j in range(len(notes_octave)):
                notes.append(notes_octave[j])
                dum = 2**i
                freq.append(freq_octave[j]*dum)

if 0:
        fig, ax = plt.subplots()
        ax.plot(notes, freq, '^', color='black')
        ax.set_ylabel('Frequency (Hz)', fontsize=14)
        ax.set_ylim((0,4000))
        ax.set_yticks(np.arange(0, 4000, 500))
        ax.set_xticklabels(notes, fontsize=14)
        ax.set_yticklabels(np.arange(0, 4000, 500), fontsize=14)
        fig.tight_layout()
        fig.savefig('notes2freq.png')

# bass
blim = 12*2 + 2
ulim = 12*3 + 4 - 3
# guitar solo
#blim = 12*2-2 #a sharp below the staff
blim = 12*3
ulim = 12*5 + 3 +1
ind = np.where((floyd_ks>=np.min(freq[blim:ulim]))&(floyd_ks<=np.max(freq[blim:ulim])))
Tau = np.squeeze(floyd_Tau[ind,:])
Ks = np.squeeze(floyd_Ks[ind,:])
spec = np.squeeze(floyd_spec.T[ind,:])

# clip
ind = np.where((floyd_ks_c>=np.min(freq[blim:ulim]))&(floyd_ks_c<=np.max(freq[blim:ulim])))
Tau_c = np.squeeze(floyd_Tau_c[ind,:])
Ks_c = np.squeeze(floyd_Ks_c[ind,:])
spec_c = np.squeeze(floyd_spec_c.T[ind,:])

#spec_rmean = np.asarray([np.convolve(spec[:,i], np.ones(int(len(Ks)/30))/int(len(Ks)/30), mode='same') for i in rang
## panel plot
import matplotlib.gridspec as gridspec

fig = plt.figure(figsize=(12,7), constrained_layout=True)
gs = fig.add_gridspec(3,2)
ax1 = fig.add_subplot(gs[0,0])
ax2 = fig.add_subplot(gs[0,-1])
ax3 = fig.add_subplot(gs[1:,:])

ax1.plot(time_floyd, filter(a, time_floyd, floyd_tau[10]), color='slateblue', linewidth=2)
ax1.set_xticks(np.arange(0, np.floor(Tf),5))
ax1.set_xticklabels(np.arange(0, np.floor(Tf),5), fontsize=12)
ax1.set_xlim((0,15))
ax1.set_xlabel('Time (s)')
print('plotted ax1')

#blim = 12*2 + 2
#ulim = 12*3 + 4 - 3
#ind = [5,10,12,17,21,22]
#freqlabels = [freq[blim:ulim][ind[i]] for i in range(len(ind))]
#labels = [notes[blim:ulim][ind[i]] for i in range(len(ind))]
freqlabels = freq[blim:ulim]
labels = notes[blim:ulim]
#p = ax2.pcolorfast(floyd_tau_c, floyd_ks_c, floyd_spec_c.T, cmap='cool')
p = ax2.pcolormesh(Tau_c, Ks_c, spec_c, shading='gouraud', cmap='cool')
p.set_clim((0.0003, np.max(spec_c)))
#p.set_clim((0.0007, np.max(spec_c)))
ax2.set_ylim((np.min(freq[blim:ulim]), np.max(freq[blim:ulim])))
ax2.set_yticks(freqlabels[::3])
ax2.set_yticklabels(labels[::3], fontsize=14)
ax2.set_xticks(np.arange(0,np.floor(Tf_c),3))
ax2.set_xticklabels(np.arange(0,np.floor(Tf_c),3), fontsize=12)
ax2.set_xlabel('Time (s)', fontsize=12)
fig.colorbar(p, ax=ax2)
print('plotted ax2')

p = ax3.pcolormesh(Tau, Ks, spec, shading='gouraud', cmap='cool')
p.set_clim((0.0003, np.max(spec)))
#p.set_clim((0.0007, np.max(spec_c)))
ax3.set_ylim((np.min(freq[blim:ulim]), np.max(freq[blim:ulim])))
ax3.set_yticks(freq[blim:ulim][::2])
ax3.set_yticklabels(notes[blim:ulim][::2], fontsize=12)
ax3.set_xticks(np.arange(0,np.floor(Tf),5))
ax3.set_xticklabels(np.arange(0,np.floor(Tf),5), fontsize=12)
ax3.set_xlabel('Time (s)', fontsize=12)
fig.colorbar(p, ax=ax3)
print('plotted ax3')

fig.savefig('floyd_filter_fb_full_guitar_bxfiltered_log.png')
print('saved figure')

import numpy as np
import matplotlib.pyplot as plt
import os
import scipy.io as sio
import scipy.io.wavfile as wav
plt.ion()
plt.style.use('ggplot')
```

```
# definte data path and filename
datapath = '../data/'
gnrfile = 'GNR.mat'

gnr_fs = 48000.0

gnr = sio.loadmat(os.path.join(datapath, gnrfile))['y']/gnr_fs
gnr = np.squeeze(gnr)
Tgnr = len(gnr)/gnr_fs
time_gnr = np.linspace(0, Tgnr, len(gnr))
### first bar only
first_bar = 48000*2
gnr_fb = np.squeeze(gnr)[:first_bar]
Tgnr_fb = len(gnr_fb)/gnr_fs
time_gnr_fb = np.linspace(0, Tgnr_fb, len(gnr_fb))


n = len(gnr)
k_pos = np.arange(0, n/2)
k_neg = np.arange(-n/2, 0)
k = (1/(Tgnr))*np.append(k_pos, k_neg)
gnr_ks = np.fft.fftshift(k)
## frequencies for first bar only
n = len(gnr_fb)
k_pos = np.arange(0, n/2)
k_neg = np.arange(-n/2, 0)
k = (1/(Tgnr_fb))*np.append(k_pos, k_neg)
gnr_ks_fb = np.fft.fftshift(k)

def filter(a, t, tau):
        return np.exp(-a*(t-tau)**2)

a = 50
gnr_tau = np.arange(0, Tgnr, Tgnr/32)
gnr_spec = np.asarray([np.fft.fftshift(np.abs(np.fft.fft(filter(a, time_gnr, gnr_tau[i])*gnr))) for i in range(len(g
# for first bar only
gnr_tau_fb = np.arange(0, Tgnr_fb, Tgnr_fb/16)
gnr_spec_fb = np.asarray([np.fft.fftshift(np.abs(np.fft.fft(filter(a, time_gnr_fb, gnr_tau_fb[i])*gnr_fb))) for i in

[gnr_Tau, gnr_Ks] = np.meshgrid(gnr_tau, gnr_ks)
# for first bar only
[gnr_Tau_fb, gnr_Ks_fb] = np.meshgrid(gnr_tau_fb, gnr_ks_fb)


notes_octave = ['C','C#','D','D#','E','F','F#','G','G#','A','A#','B']
freq_octave = [16.35, 17.32, 18.35, 19.45, 20.60, 21.83, 23.12, 24.50, 25.96, 27.5, 29.14, 30.87]

notes = []
freq = []
for i in range(8):
        for j in range(len(notes_octave)):
                notes.append(notes_octave[j])
                dum = 2**i
                freq.append(freq_octave[j]*dum)

if 0:
        fig, ax = plt.subplots()
        ax.plot(notes, freq, '^', color='black')
        ax.set_ylabel('Frequency (Hz)', fontsize=15)
        ax.set_ylim((0,4000))
        ax.set_yticks(np.arange(0, 4000, 500))
        ax.set_xticklabels(notes, fontsize=14)
        ax.set_yticklabels(np.arange(0, 4000, 500), fontsize=15)
        fig.tight_layout()
        fig.savefig('notes2freq.png')


## panel plot
import matplotlib.gridspec as gridspec

fig = plt.figure(figsize=(12,7), constrained_layout=True)
gs = fig.add_gridspec(3,2)
ax1 = fig.add_subplot(gs[0,0])
ax2 = fig.add_subplot(gs[0,-1])
ax3 = fig.add_subplot(gs[1:,:])

ax1.plot(time_gnr, filter(a, time_gnr, gnr_tau[8]), color='slateblue', linewidth=2)
ax1.set_xticks(np.arange(0, np.floor(Tgnr),3))
ax1.set_xticklabels(np.arange(0, np.floor(Tgnr),3), fontsize=12)
ax1.set_xlabel('Time (s)')

blim = 12*3 + 8
ulim = 12*5 + 3 + 5
ind = [5,10,12,17,21,22]
freqlabels = [freq[blim:ulim][ind[i]] for i in range(len(ind))]
labels = [notes[blim:ulim][ind[i]] for i in range(len(ind))]
p = ax2.pcolormesh(gnr_Tau_fb, gnr_Ks_fb, gnr_spec_fb.T, shading='gouraud', cmap='cool')
p.set_clim((0.0003, np.max(gnr_spec_fb)))
ax2.set_ylim((np.min(freq[blim:ulim]), np.max(freq[blim:ulim])))
ax2.set_yticks(freqlabels)
ax2.set_yticklabels(labels, fontsize=14)
ax2.set_xticks(np.arange(0,np.floor(Tgnr_fb),1))
ax2.set_xticklabels(np.arange(0,np.floor(Tgnr_fb),1), fontsize=12)
ax2.set_xlabel('Time (s)', fontsize=12)
fig.colorbar(p, ax=ax2)

p = ax3.pcolormesh(gnr_Tau, gnr_Ks, gnr_spec.T, shading='gouraud', cmap='cool')
p.set_clim((0.0003, np.max(gnr_spec)))
ax3.set_ylim((np.min(freq[blim:ulim]), np.max(freq[blim:ulim])))
ax3.set_yticks(freq[blim:ulim])
ax3.set_yticklabels(notes[blim:ulim], fontsize=12)
ax3.set_xticks(np.arange(0,np.floor(Tgnr),1))
ax3.set_xticklabels(np.arange(0,np.floor(Tgnr),1), fontsize=12)
ax3.set_xlabel('Time (s)', fontsize=12)
fig.colorbar(p, ax=ax3)
fig.savefig('gnr_filter_fb_full_threshold.png')
```