EE-UY 373 Guided Studies Project: Real-time Filtering & Modulation Using Python
Guided Studies in Electrical Engineering

Department of Electrical & Computer Engineering
New York University Tandon School of Engineering

Azcona, Emanuel
emanuelazcona@nyu.edu

May 16, 2016

Abstract

The objective of this semester-long project was to integrate all of the material from previous coursework in Digital Signal Processing and Signals & Systems to create an application that does any sort of real-time signal processing. My application was built primarily through keyboard-inputs, drawings, and basic user interface (UI) design using the `pygame`() module. The purpose of this application is to grant users access to playing multiple music tracks (`.wav` files) simultaneously, as well as access to filters/modulators with tunable cutoff/shift frequencies via mouse interaction. The application overall works as expected, however is definitely subject to improvements. The write-up for this project was written in LaTeX for practice.

Audio File Usage

The following audio application was designed to work using with **stereo** `.wav` audio files with 16-bits per sample. Any sampling rate will do. If you have any audio files that are not in the aforementioned format, I suggest you use any DAW software (i.e. Audacity) to convert your audio file to stereo 16-bits per sample, `.wav` files. The `.wav` format was used because of its simplicity in use with the `pyaudio` library as well as its advantages as a lossless audio format.

1 PyGame Aspects

1.1 GUI Design

This application heavily depends on the `pygame`() module for its UI design and keyboard/mouse interactivity. The entire UI was built using the different `draw`() methods in the `pygame`() module, such as the rectangle drawing method:

```
pygame.draw.rect(screen, red, pygame.Rect(160, 335, 20, 220))
```

The above command draws a rectangle centered at position x = 160, y = 335 inside the application window, and of dimensions $l = 20$ and $w = 220$. Other drawing functions and surface updating functions included, but not limited to:

- `polygon`()

- `blit()`
- `pygame.init()`
- `display.set_mode((dimensions of window))`
- `display.set_caption('Window Title')`

Use of the keyboard/mouse interactivity/monitoring methods in `pygame()` was also heavily practiced during the design of this UI. The user keyboard interactivity/monitoring dealt mostly with, but not limited to, the `event.get()`, key, KEYDOWN, `K+''char''` methods. Below is an example of a section in the project's code demonstrating logic statements dependent on keyboard input which perform certain actions when pressed.

```
################################################################################
# Channel 1 Controls

        elif event.type == pygame.KEYDOWN and event.key == pygame.K_s:
            Channel_1.on = not Channel_1.on
        elif event.type == pygame.KEYDOWN and event.key == pygame.K_w:
            if Channel_1.gain + 0.05 > 1.50:
                Channel_1.gain = 1.50
            else:
                Channel_1.gain += 0.05
        elif event.type == pygame.KEYDOWN and event.key == pygame.K_x:
            if Channel_1.gain - 0.05 < 0.0:
                Channel_1.gain = 0.0
            else:
                Channel_1.gain -= 0.05
        elif event.type == pygame.KEYDOWN and event.key == pygame.K_a:
            if Channel_1.left_gain + 0.1 > 2.0 and Channel_1.right_gain - 0.1 < 0.0:
                Channel_1.left_gain = 2.0
                Channel_1.right_gain = 0.0
            else:
                Channel_1.left_gain += 0.1
                Channel_1.right_gain -= 0.1
        elif event.type == pygame.KEYDOWN and event.key == pygame.K_d:
            if Channel_1.right_gain + 0.1 > 2.0 and Channel_1.left_gain - 0.1 < 0.0:
                Channel_1.right_gain = 2.0
                Channel_1.left_gain = 0.0
            else:
                Channel_1.right_gain += 0.1
                Channel_1.left_gain -= 0.1
```

Figure 1.1: Logic statements dependent on specific keyboard input

Mouse interactivity was also widely used in this project to control the cutoff/shift frequencies of specific filters or modulators. In this case, each channel has two filters and a complex modulator that can be used to modify the output sound. Each filter/modulator has a square "ON" button that controls whether or not the output sound is pushed through the filter/modulator before outputting. Within the program I designed a tracking algorithm using the `mouse()` module (located inside of the `pygame()` module) which keeps track of your mouse's position within a `pygame` screen. Four the square "ON" button shapes, keeping track of whether or not the mouse is inside was relatively simple. It just involved keeping track of whether or not the mouse's "x-location" is within the limits of the square's "x-limits", and keeping track of the same for the mouse's "y-location."

The shape that proved to be a little tricky for this were the triangular shapes, which control the increase/decrease of the cutoff/shifting frequencies of the filters/modulators. In order to know if the mouse is within the "triangular-region," the locations of the triangles' base vertices were needed. Using the a right-side up equilateral triangle as an example, by knowing the location of the mouse at a given moment, we can check to see if the mouse's "x-location" is to the left of the right vertex's "x-coordinate' and to the left of the right vertex's "x-coordinate." If this condition is satisfied then the next step is to take the mouse's "y-location" and subtract it from the "y-position" of the triangle's base. Afterwards, the following

inequalities were evaluated to finally determine if the mouse is within a specific triangular-region:

$$0 \leq \text{atan2}\left(\Delta_{base} - m_y, \Delta_{x_{right}} - m_x\right) \leq \frac{\pi}{3} \tag{1}$$

$$0 \leq \text{atan2}\left(\Delta_{base} - m_y, m_x - \Delta_{x_{left}}\right) \leq \frac{\pi}{3} \tag{2}$$

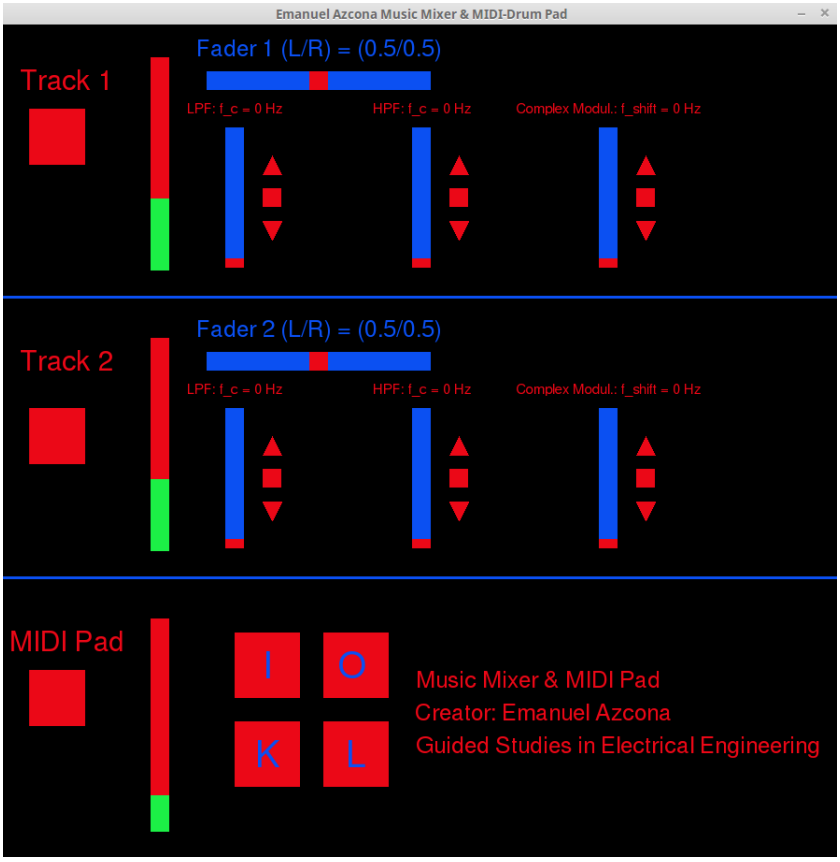

Figure 1.2: Front-end GUI of Music Mixer & MIDI Pad

Whether it's mouse or keyboard interaction, the color of the correspondingly linked "buttons" were changed between red (OFF) or green (ON) using the RGB values of red and green as a tuple respectively.

Blue bars in the program are used as references to the range of values for fading, cutoff/modulation frequencies. The small red boxes within these blue "meters" are reserved for "ticking" a position along its respective meter.

Below are the button layouts to their corresponding channels.

Figure 1.3: Channel 1 Buttons Layout

- <"S"> = channel on
- <"A"> = fade left
- <"D"> = fade right
- <"W"> = increase gain
- <"X"> = decrease gain
- <"1"> = mute track (continues playing with zero gain)

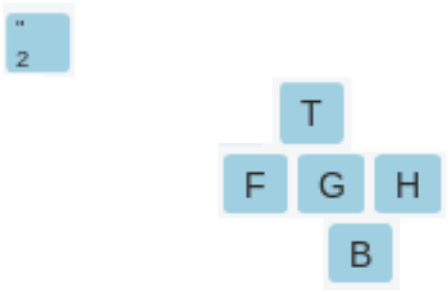

Figure 1.4: Channel 2 Buttons Layout

- <"G"> = channel on
- <"F"> = fade left
- <"H"> = fade right
- <"T"> = increase gain
- <"B"> = decrease gain
- <"2"> = mute track (continues playing with zero gain)

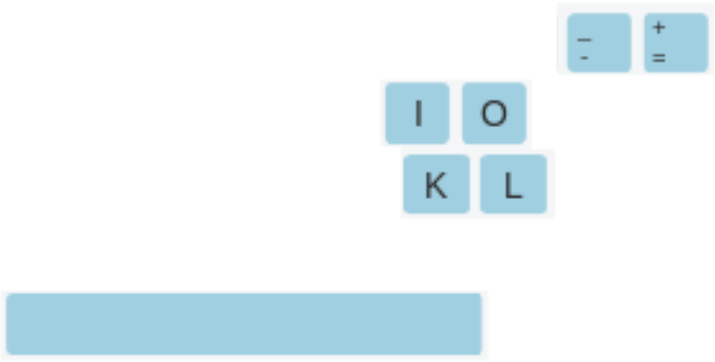

Figure 1.5: MIDI Buttons Layout

- <"[SPACE]"> = MIDI on
- <"I"> = key-1 sound
- <"O"> = key-2 sound
- <"K"> = key-3 sound
- <"L"> = key-4 sound
- <"−"> = decrease gain
- <"+"> = increase gain

All other triangular shapes and square in between these triangles are controlled by clicking within the region of the pygame window

1.2 MIDI Channel Module

Along with the "main" module, `Emanuel_Azcona_Semester_Long.py`, a second module, `MIDI_Channel.py`, was created in order to create a class known as `MIDI_Channel`. The `MIDI_Channel` class contained several initial parameters such as:

```
def __init__(self, channel):
        self.channel_num = channel
        self.gain = 0.5
        self.left_gain = 1.0
        self.right_gain = 1.0
        self.on = False
```

Other methods are included within the `MIDI_Channel` class such as methods to open,read, and manipulate `.wav` files, MIDI-key creation, and audio stream I/O.

1.3 Filters & Effects

The filters and effects used to manipulate the audio output of each MIDI Channel Track included:

- Lowpass Filter
- Highpass Filter

- Complex Modulator

Starting with the Lowpass Filter, the transfer function of this system was based on Eq. 3.24 from [1].

$$H_{LP}(z) = \frac{(z+1)\tan\left(\frac{\omega_c}{2}\right)}{z\left[\tan\left(\frac{\omega_c}{2}\right)+1\right]+\tan\left(\frac{\omega_c}{2}\right)-1}$$

Using this information, two lists were created on Python representing the coefficients of the polynomials in each the numerator and denominator of the transfer function. Using these lists, "b" and "a", they were manipulated alongside a library from the Python module `scipy`(), known as `signal`(). One of the many methods inside the `signal`() library of the `scipy`() module is the `lfilter`() method, which filters a data sequence, x, using a digital linear filter. The standard I/O form of the `lfilter`() method is:

```
from scipy import signal

y = signal.lfilter(b, a, x)
```

According to [2], the filter function is implemented as a direct II transposed structure. This means that the filter implements:

$$a_0 y[n] = (b_0 x[n] + b_1 x[n-1] + ... + b_i x[n-i]) - (a_1 y[n-1] + ... + a_i y[n-i])$$

using the following difference equations:

$$\begin{aligned}
&y[m] = b_0 x[m] + z[0, m-1] \\
&z[[0, m] = b_1 x[m] + z[1, m-1] - a_1 y[m] \\
&... \\
&z[n-3, m] = b_{n-2} x[m] + z[n-2, m-1] - a_{n-2} y[m] \\
&z[n-2, m] = b_{n-1} x[m] - a_{n-1} y[m]
\end{aligned}$$

where m is the output sample number and

$$n = \max\{a_{len}, b_{len}\}$$

is the model order. The rational transfer function describing this filter in the z-transform domain is:

$$Y(z) = \frac{b_0 + b_1 z^{-1} + ... + b_i z^{-i}}{a_0 + a_1 z^{-1} + ... + a_i z^{-i}} \cdot X(z)$$

Using the same idea, the Highpass Filter was also implemented, using Eq. 3.27 from [1].

$$H_{HP}(z) = \frac{z-1}{\left[\tan\left(\frac{\omega_c}{2}\right)+1\right]z+\tan\left(\frac{\omega_c}{2}\right)-1}$$

Lastly, complex modulation was also implemented through the multiple steps necessary to perform it.

- Obtaining b and a coefficient vectors for a Highpass Filter with $\omega_c \approx 0$ rad/sec. In order to do so, an Elliptic (Cauer) Filter was designed using the `scipy.signal`() library.
- After using the elliptic filter design method to create a Highpass Filter, the output of that system was multiplied by another signal (convolution in the frequency domain) in order to shift the frequency of the input signal by ω_{shift}. If you analyze such a signal that has shifting properties in the frequency domain, we realize that such a signal's Fourier transform must be $\delta(\omega - \omega_{shift})$. Therefore, we can analyze that this signal must be:

$$\mathcal{F}^{-1}\left[\delta(\omega - \omega_{shift})\right] = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{j(\omega-\omega_{shift})n} dn$$

- Lastly, the real part of the signal obtained in the previous step was taken to get re-obtain a symmetrical Fourier transform. To do this, one just has to make sure that the array who's real part we are taking, is a numpy array. Numpy arrays have a built-in `.real` method, that returns the real-part of each element in the numpy array in an array.

1.4 Files

1. `Emanuel_Azcona_Semester_Long.py` - main file
2. `MIDI_Channel.py` - file with MIDI-Channel class definition and methods
3. `EasyGame.py` - file called in main file for use of filegetter method
4. `Music` folder - folder containing Stereo `.wav` files
5. `lib` folder - folder containing important files for FileGetter module used in main file
6. `example` folder - folder containing important files for FileGetter module used in main file
7. `Emanuel_Azcona_Music_Mixer_eaa349.tex`
8. `Emanuel_Azcona_Music_Mixer_eaa349.pdf`

2 Possible Future Improvements

2.1 GUI Design Libraries

The overall design of the interactive GUI for the application was very basic only consisting of basic polynomial shapes, and basic color patterns. There must be a better way of designing a GUI without having to draw and re-draw lines and figures after a changes has occurred in the GUI environment.

After some extensive research in GUI design libraries available for Python, many tutorials and forums showed appreciation for the use of the following libraries for GUI design:

- `Tkinter`
- `Kivy`
- `gui2py`
- `PyJamas`

and many more. Many of these libraries eliminated the necessity of constant "drawing" and base their GUi design mainly using object-oriented programming techniques and functionalities.

2.2 Artifact Removal Using De-Noising Techniques

Since our application runs in real-time, sometimes due to the limit on the operating computer's performance, artifacts and other discontinuities in the output sound may occur after filtering/modulation is applied. When no effects are present however, these artifacts do not occur. A solution to the presence of these artifacts and discontinuities in our output waveform may be solved using different de-noising techniques.

2.2.1 Thresholding Techniques

Many thresholding techniques, but two simple thresholding techniques were extensively studied because of their applicable simplicity:

- Soft-Thresholding
- Hard-Thresholding

The soft-thresholding function can easily be determined from the following minimization problem:

$$\hat{x}(y,\lambda) = \arg\min_{x} \left\{ \frac{1}{2}(y-x)^2 + \lambda|x| \right\}, \quad \lambda > 0. \tag{3}$$

The following solution, known as the soft-thresholding function, can be determined and we can obtain the following solution where $\hat{x}$ is the noise-reduced output, λ is the threshold, and y is the noisy input signal:

$$\hat{x}(y,\lambda) = \begin{cases} y+\lambda, & y < -\lambda \\ 0, & |y| < \lambda \\ y-\lambda, & y > \lambda 1 \end{cases} \tag{4}$$

The hard-thresholding function can be determined from the following minimization problem as well:

$$\hat{x}(y,\lambda) = \arg\min_{x} \left\{ \frac{1}{2}(y-x)^2 + \lambda|x| \right\}, \quad \phi(x) = \begin{cases} 0, & x = 0 \\ 1, & x \neq 0 \end{cases}, \quad \lambda > 0. \tag{5}$$

where we obtain the solution to a hard-thresholding function as:

$$\hat{x}(y,\lambda) = \begin{cases} y, & |y| > \sqrt{2\lambda} \\ 0, & |y| < \sqrt{2\lambda} \end{cases} \tag{6}$$

We can apply, any of these and the many other variations of thresholding functions to our overall signal, windows of our signal, or wavelet representations of our signal which will be discussed in the next sub-sub-section.

2.2.2 Wavelet De-Noising

One of the researched de-noising techniques that came up involved the use of Wavelet Transforms and different thresholding techniques within Wavelet Transforms. The Wavelet Transform has become a very useful computational tool for a variety of signal processing applications such as

- Compression of digital image/audio/video files
- 'Cleaning' signals and images (reducing unwanted noise and blurring)

The most basic Wavelet Transform is the Haar Transform which is described by

$$\begin{aligned} c[n] &= \frac{1}{2}x[2n] + \frac{1}{2}x[2n+1] \\ d[n] &= \frac{1}{2}x[2n] - \frac{1}{2}x[2n+1] \end{aligned} \tag{7}$$

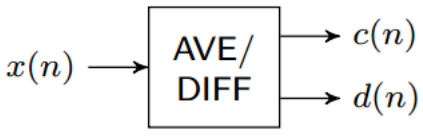

Figure 2.1: Wavelet Transform Black Box Systematic Diagram

As we can see, this transform can clearly be reversed and the original input can be reconstructed from the decomposition above.

$$\begin{aligned} y[2n] &= c[n] + d[n] \\ y[2n+1] &= c[n] - d[n] \end{aligned} \tag{8}$$

Figure 2.2: Inverse Wavelet Transform Black Box Systematic Diagram

The Haar Transform, along with other Wavelet Transforms can be repeated in the following representation (with N-"stages") in order to obtain many smaller-length "wavelet representations" of our original signal $x[n]$.

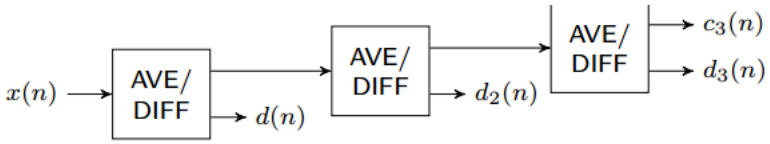

Figure 2.3: Three-stage Wavelet Transform

This multi-stage property and sparse-representation property of the wavelet transform is what makes it useful for compressing audio/images/video signals.

Using the wavelet representation of a signal we can apply a thresholding function to each of the wavelets by finding the standard deviation of the values for the corresponding wavelet and using the standard deviation towards that respective wavelet as the threshold for the thresholding function. Sometimes however, depending on the thresholding function, the thresholding function can remove artifacts while adding its own.

2.2.3 Short-Time Fourier Transform (STFT) De-Noising

Noise reduction is also possible with the Short-Time Fourier Transform (STFT). The STFT is a windowed version of the Fourier Transform such that the Fourier Transform of a specific window of our input signal is taken, one window at a time. The function to be transformed is first multiplied by a window function (preferably one that satisfies the perfect reconstruction property with respect to the overlap size from window to window), which is nonzero for only a short period of time. The result is a two-dimensional representation of the signal (Fourier Transform vs. time/sample). The STFT of a signal can be plotted, yielding the spectrogram of the function, which displays the magnitude squared of the STFT at the respective frequency and time that it occurs (see following figure).

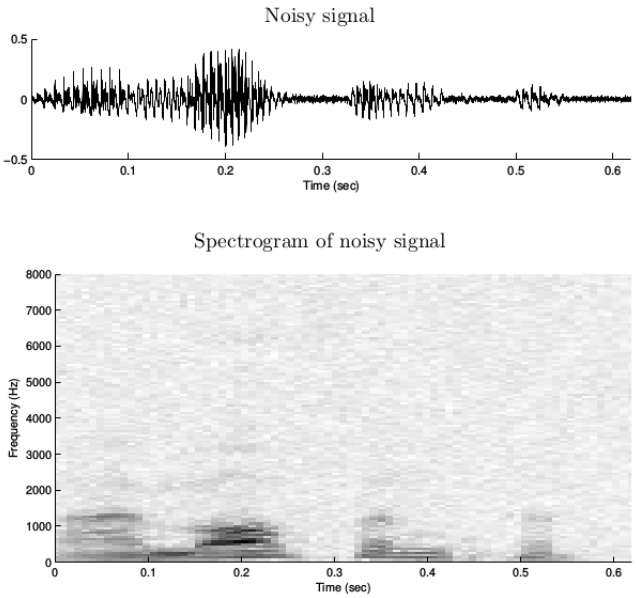

Figure 2.4: Noisy Input Signal (Top), Spectrogram of Noisy Input Signal (Bottom)

Similar to wavelet-based noise reduction, the same technique regarding the standard deviation of each "window" can be applied to reduce noise through the STFT.

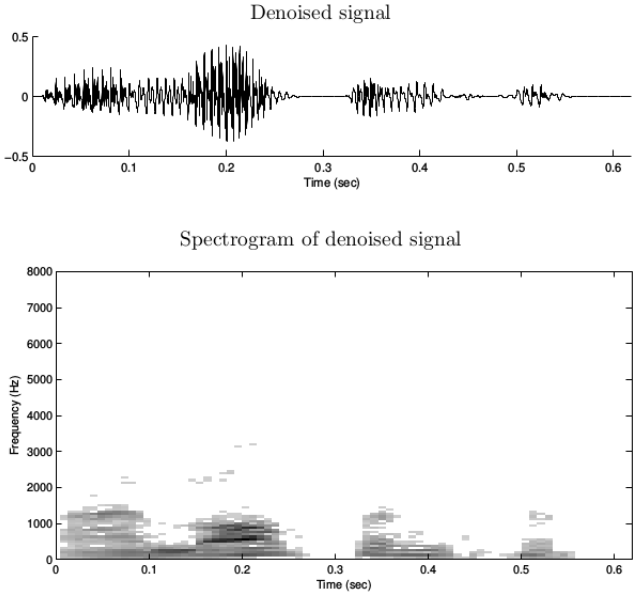

Figure 2.5: Denoised Signal (Top), Spectrogram of Denoised Signal (Bottom)

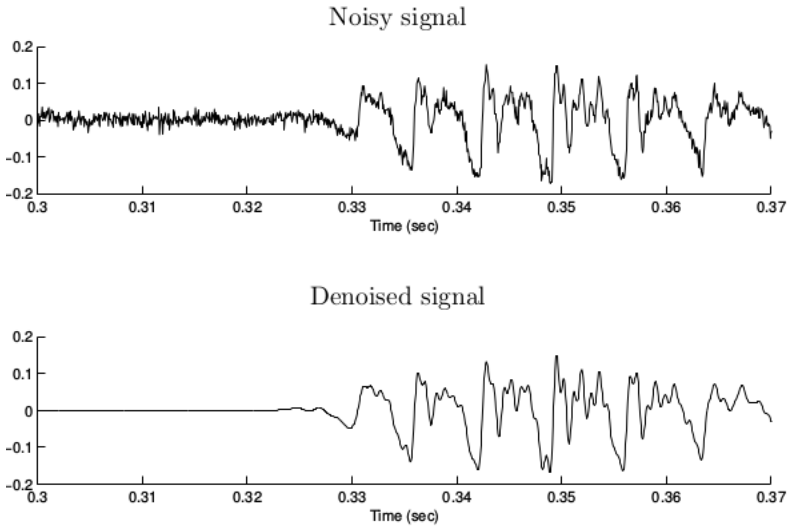

Figure 2.6: Another example of noise reduction using the STFT

References

[1] Reiss, Joshua D., and Andrew P. McPherson. *Audio Effects Theory, Implementation and Application.* Boca Raton, Fla.: CRC, Taylor & Francis Group, 2015. Print.

[2] "Scipy.signal.lfilter." *Scipy.signal.lfilter SciPy V0.16.1 Reference Guide.* The Scipy Community (Sponsored by Enthought), n.d. Web. 18 Dec. 2015.

[3] Morgan, John, and Frederick Fong. "Lecture 31." *University Lecture Series Ricci Flow and Geometrization of 3-Manifolds* (2010): 141-43. Web. <http://links.uwaterloo.ca/amath353docs/set11.pdf>

[4] "Pygame.draw." *Pygame V1.9.2 Documentation.* N.p., n.d. Web. 18 Dec. 2015. <https://www.pygame.org/docs/ref/draw.html>.

[5] "Pygame.mouse." *Pygame V1.9.2 Documentation.* N.p., n.d. Web. 18 Dec. 2015. <https://www.pygame.org/docs/ref/mouse.html>

[6] "PyAudio." *: PortAudio V19 Python Bindings.* N.p., n.d. Web. 18 Dec. 2015. <https://people.csail.mit.edu/hubert/pyaudio/>

[7] "Music21 Documentation." Music21 Documentation Music21 Documentation. N.p., n.d. Web. 18 Dec. 2015. <http://web.mit.edu/music21/doc/index.html>

[8] "Improve Your Python: Python Classes and Object Oriented Programming." *Improve Your Python: Python Classes and Object Oriented Programming.* N.p., n.d. Web. 18 Dec. 2015. https://www.jeffknupp.com/blog/2014/06/18/improve-your-python-python-classes-and-object-oriented-programming/