Lab 5

To: Ross Snider
From: Anthony Louis Rosenblum
Regarding: Lab 3
Date: January 27th, 2019

Summary

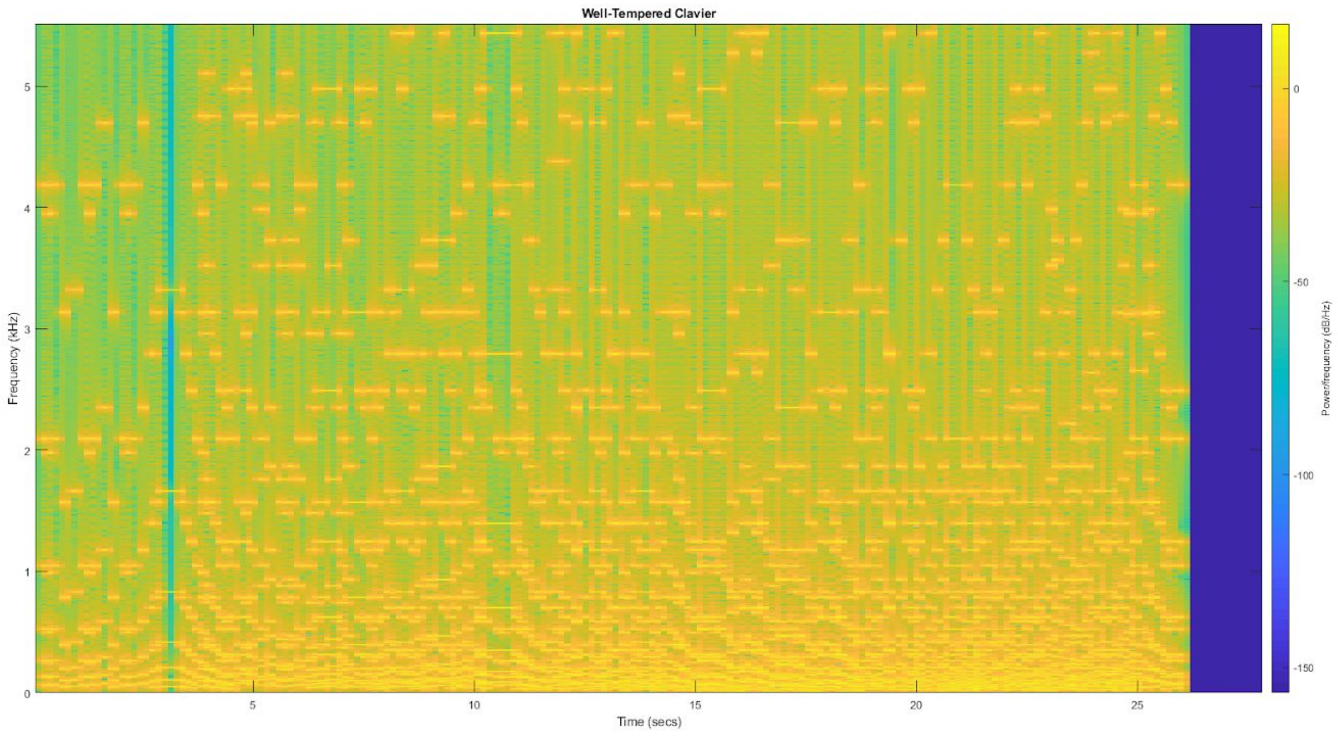
In this lab I examined using MATLAB to synthesize music based off of vector information describing the associated piano key, starting beat, and duration. By far the most difficult section to address was including the rests so the notes would begin on the right beats and that will be addressed in my conclusion.

Main Body

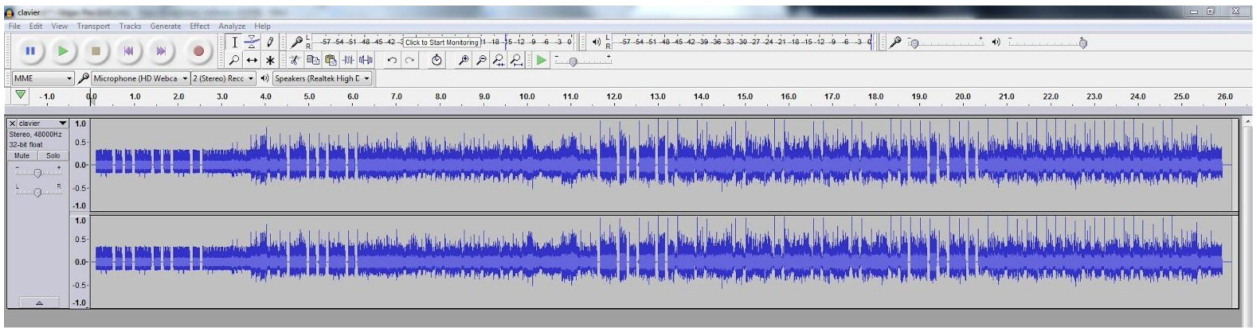
I chose to generate the song using all three instruments and also including a total of 7 harmonics for each instrument. These include -3 octaves, -2 octaves, -1 octave, +0 octaves, +1 octave, +2 octaves, and +3 octaves. Every harmonic had the same amplitude of 10.

I generated the harmonics using the following code:

```
tone = key2note(10,key(kk),dur(kk))
tone = tone + key2note(10,key(kk)-12,dur(kk))
tone = tone + key2note(10,key(kk)+12,dur(kk))
tone = tone + key2note(10,key(kk)+24,dur(kk))
tone = tone + key2note(10,key(kk)-24,dur(kk))
tone = tone + key2note(10,key(kk)-36,dur(kk))
tone = tone + key2note(10,key(kk)+36,dur(kk))
```



Spectrogram of "Well Tempered Clavier"



Time Varying Representation in Audacity

Here is the code for generating the song in its entirety:

```
function xx = key2note(X, keynum, dur)
% KEY2NOTE Produce a sinusoidal waveform corresponding to a
% given piano key number
%
% usage: xx = key2note (X, keynum, dur)
%
% xx = the output sinusoidal waveform
% X = complex amplitude for the sinusoid, X = A*exp(j*phi).
% keynum = the piano keyboard number of the desired note
% dur = the duration (in seconds) of the output note
%
fs = 11025; %-- or use 8000 Hz
tt = 0:(1/fs):dur;
num = 49 - keynum;
freq = 440 / 2^(1/12*num)
xx = real( X*exp(j*2*pi*freq*tt) );
end

load bach_fugue.mat

key = theVoices(1).noteNumbers;
pulse = theVoices(1).startPulses;
dur = theVoices(1).durations;

dur = dur/8;

fs = 11025;
xx = zeros(1,28*fs);
n1 = 1;
n2 = 0;

for kk = 1:length(key)
n1 = (pulse(kk)-1)/9*fs;
tone = key2note(10,key(kk),dur(kk))
tone = tone + key2note(10,key(kk)-12,dur(kk))
tone = tone + key2note(10,key(kk)+12,dur(kk))
tone = tone + key2note(10,key(kk)+24,dur(kk))
tone = tone + key2note(10,key(kk)-24,dur(kk))
tone = tone + key2note(10,key(kk)-36,dur(kk))
tone = tone + key2note(10,key(kk)+36,dur(kk))
n2 = n1 + length(tone) - 1;
xx(n1:n2) = xx(n1:n2) + tone; %<=== Insert the note
end

key = theVoices(2).noteNumbers;
```

```
pulse = theVoices(2).startPulses;
dur = theVoices(2).durations;
dur = dur/8;

n1 = 1;
n2 = 0;

for kk = 1:length(key)
n1 = (pulse(kk)-1)/9*fs;
tone = key2note(10,key(kk),dur(kk))
tone = tone + key2note(10,key(kk)-12,dur(kk))
tone = tone + key2note(10,key(kk)+12,dur(kk))
tone = tone + key2note(10,key(kk)+24,dur(kk))
tone = tone + key2note(10,key(kk)-24,dur(kk))
tone = tone + key2note(10,key(kk)-36,dur(kk))
tone = tone + key2note(10,key(kk)+36,dur(kk))
n2 = n1 + length(tone) - 1;
xx(n1:n2) = xx(n1:n2) + tone; %<=== Insert the note
end

key = theVoices(3).noteNumbers;
pulse = theVoices(3).startPulses;
dur = theVoices(3).durations;
dur = dur/8;

n1 = 1;
n2 = 0;

for kk = 1:length(key)
n1 = (pulse(kk)-1)/9*fs;
tone = key2note(10,key(kk),dur(kk))
tone = tone + key2note(10,key(kk)-12,dur(kk))
tone = tone + key2note(10,key(kk)+12,dur(kk))
tone = tone + key2note(10,key(kk)+24,dur(kk))
tone = tone + key2note(10,key(kk)-24,dur(kk))
tone = tone + key2note(10,key(kk)-36,dur(kk))
tone = tone + key2note(10,key(kk)+36,dur(kk))
n2 = n1 + length(tone) - 1;
```

```
xx(n1:n2) = xx(n1:n2) + tone; %<=== Insert the note
end

soundsc( xx, fs )

spectrogram(xx,3000,[],3000,fs,'yaxis'),title("Well-Tempered
Clavier")
```

This could have been performed using another nested for loop but I just did each instrument separately for my own sanity.

Conclusion

It was relatively simple to implement the correct frequencies and durations using the previously generated KEY2NOTE function. However incorporating the rests became a challenge. I initially attempted to add the starting beat for each note as an input to the KEY2NOTE function but that proved difficult to account for in using the output. I tried to make all the note vectors the same length as the song with zero amplitude outside their duration, but that lead to memory issues. I then attempted to use the pulse information when indexing the note into the zeros vector. This was much easier and I eventually found a factor that made the rests sound appropriately long. If they were too long I would get large blocks of silence in between notes and if they were too short I would get notes stacking on top of each other. Because I adjusted the duration vector so a value of 1.00 was 0.125 seconds I ended up adjusting the starting beat index to become (starting_beat - 1) * sampling frequency / 9. Or about (starting_beat - 1) * 1225. This produced the desired output and the music sounded lovely.