ECE ECE280L– Introduction to Signals and Systems
Lab 1: Music Synthesis

Michael Casio

February 16, 2018

Duke University

Instructor:	Professor M. Knox
TA:	Noah Johnson
Section:	ECE 280L-07L

I have adhered to the Duke Community Standard in completing this assignment.

Contents

1 Introduction

For this lab, I chose to do Redbone by Childish Gambino, from his 2016 album *Awaken, My Love!* I chose to do this song because not only am I a fan of Childish Gambino and thought that this song and this album were great, but also because the song has a catchy intro and melody that seemed interesting to code in Matlab. In this lab, different signal processing characteristics such as volume variation, instrument synthesis, and overlapping tones will be explored through quick attacks and slow decays of notes, variation of frequencies to generate different waveforms, and layering of vectors of notes.

2 Procedure

2.1 Approach

To generate the notes, a function called `myNote.m` was created, as shown in the Appendix. This function took in two parameters, the frequency of the note, which ranged from 55 Hz to 440Hz, and the duration of the note (i.e. to make it into an eighth note, quarter note, half note, etc). This note function generated a sinusoid that was in the form:

$$y(t) = (2^{-4t})\cos\left(2\pi f_n t + 50^{-t}\cos\left(\pi f_n t\right)\right)$$

where $t =$ time duration of the note, and f_n = the frequency of the note. In the main code, `awakenMyLove.m`, each note (a, b, b-flat, c, etc.) composed of an anonymous function that uses the `myNote` function to generate a sinusoid with the appropriate frequency and duration. These note anonymous functions are later called in the vector that contains the song to generate notes and chords.

2.2 Implementation

In this lab, instrument synthesis, layering of vectors, and volume variation were implemented. Instrument synthesis was implemented through the note generation function formula as shown above. The general formula is as follows:

$$y(t) = A(t)\cos\left(2\pi f_c t + I(t)\cos\left(2\pi f_m t\right)\right)$$

where $A(t)$ and $I(t)$ are amplitudes that both have an exponential decay. Through varying both amplitudes and frequencies f_c and f_m, a note that sounded similar to a piano was generated. Furthermore, layering of vectors, where chords and notes were placed on each other, was created by adding the vectors and divided by the amount of vectors being added. This can be seen in the Matlab code in the Appendix. Lastly, volume variation was implemented by varying the exponential decay for each note to produce a small sustain. To actually vary the volume (i.e. making the left hand of the song quieter than that of the right hand), the vector's amplitude was altered by multiplying each element in the array by a certain factor.

3 Results

At first, the song sounded very plain, and had a lot of points where there was silence. This is because of problems with finding the right factor for the decay of each note so that it has a sustain effect. The final output, however, was able to find a good balance to fill these points of silence and implemented chords and slurs. The final product and the initial product were also completely different in that the initial product utilized a composition from a YouTube tutorial of how to play Redbone on the piano, whereas the final product actually used sheet music of the song. The reason for the change was because when the YouTube tutorial of the song was coded, only 28 seconds worth of the song was shown the notes for. To fulfill the 30 second minimum of the assignment, it would have required blindly figuring out the notes for the verses, unlike with the sheet music, the intro/hook's notes as well as the verses' notes were provided.

Furthermore, earlier versions of the song didn't implement exponential decay functions in the note generator, resulting in very flat and abrupt notes. The inner cosine function was initially not implemented, resulting in notes sounding like computer beeps, unlike the final version, which generated notes that sounded more like piano notes.

4 Discussion

4.1 Effect on Sound

The main effects that affected the sound of the notes and song were the exponential decay amplitudes, as shown in the above equations, and the instrument synthesis. In the earlier stages of the project, the song sounded very high-pitched, and had very flat and abrupt notes. By implementing decay amplitudes, the note decayed slower, and thus added a sustain-like feature to the notes. This allowed for less awkward quiet moments in the song, and for a more well-rounded sound. Additionally, the instrument synthesis formula allowed for the song to sound more like a piano, unlike its earlier versions where it sounded very much like a wind instrument or a high-pitched beep. Multiple notes also added for richer sound and a more complete song. For example, when the verse starts to play, chords play as well. These chords allowed for a mimicking of the many instruments that are present in the actual song. Though my song did not include harmonics or reverb, I believe that it would have allowed for a sound that sounds like it was recorded in a certain type of room (reverb). Out of the effects that I implemented, the ones that had the greatest impact on sound would be instrument synthesis, as explained above. On the other hand, the effect that had the least impact would be the multiple notes because although it provided a richer sound, the song would have sounded fine since the left hand notes were present, even though the chords weren't.

4.2 Challenges

One of the biggest challenges I experienced in this lab is trying to understand and read sheet music. Because I don't have any background in music, it was a struggle to translate sheet music, which was one of the reasons why I initially chose to find the notes to the song through a step-by-step YouTube tutorial. Through this decision, I learned that the tutorial was only for the intro, an not for any of the verses, which made it difficult to fulfill the 30 second time minimum. To finally overcome the challenge of not understanding sheet music, I sought help from my peers who do know how to read music and to help me translate.

After finally getting the proper notes, another challenge that was experienced in this lab was making sure that the left hand notes and the right hand notes were synced. This required constantly checking the size of the two vectors and making sure that the right note played in the right time. The fact that there is a slight delay when two `soundsc` functions are called did not help with making sure that the notes lined up with each other.

4.3 Future Improvements

As for future improvements, assuming more time and resources, the possible steps that I would have taken to further improve the quality of my song would be to fully implement an ADSR envelope that varies with each note. This would provide for better sustains in the song and allow for more instruments (e.g. drums have faster decays than pianos, so more time could have allowed for experimentation with different instruments. I think an ADSR envelope would further lessen quiet pauses in between notes.

As for the remaining questions in the lab, the endpoint was set to $1 - \frac{1}{f_s}$ instead of 1 because ending a note at 1 would cause the loop to go out of bounds. Additionally, the use of cosine vs sine does not matter in this case. It all depends on how the sound is intended to sound, since a sine function is a cosine function, except shifted. Furthermore, the use of vectors of zero amplitude was useful in setting up rests and making sure that all vectors were aligned.

5 Extension

In Figure 1, the plot of the entire song is shown. The blue sinusoids with an amplitude of 1 are the right hand melody of the song, while the orange sinusoids are the left hand beats, and the yellow are the chords. This graph depicts that variations of sound as well as the amplitude modulation in the song, where there were pauses, and where there was music playing. One key feature of this graphs is the fact that overlapping tones allowed for a richer and more complete song. Though it is seen that there are many periods of silence in the song, the overlapping tones didn't start at the exact same time as the right hand's melody notes, which allowed the left hand to decay while the right hand is at rest.

In Figure 2, the plot of one b-flat eighth note with a frequency of 110 Hz is shown. This graph was included to demonstrate the exponential decay that each note experienced to produce a sustain of some sort. This made the song flow smoother. Additionally, in this figure, it can also be seen that each note had a significantly quick attack. This is very similar to a piano, as noted in the ADSR curves in De Leon's "Computer Music in Undergraduate Digital Processing."

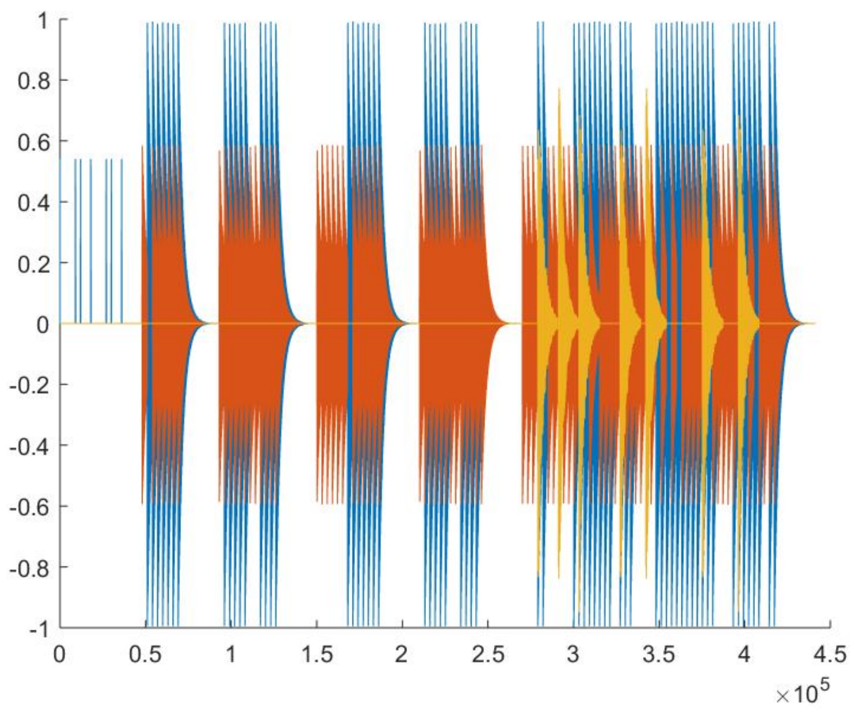

Figure 1: Graph of Redbone from `awakenMyLove.m`

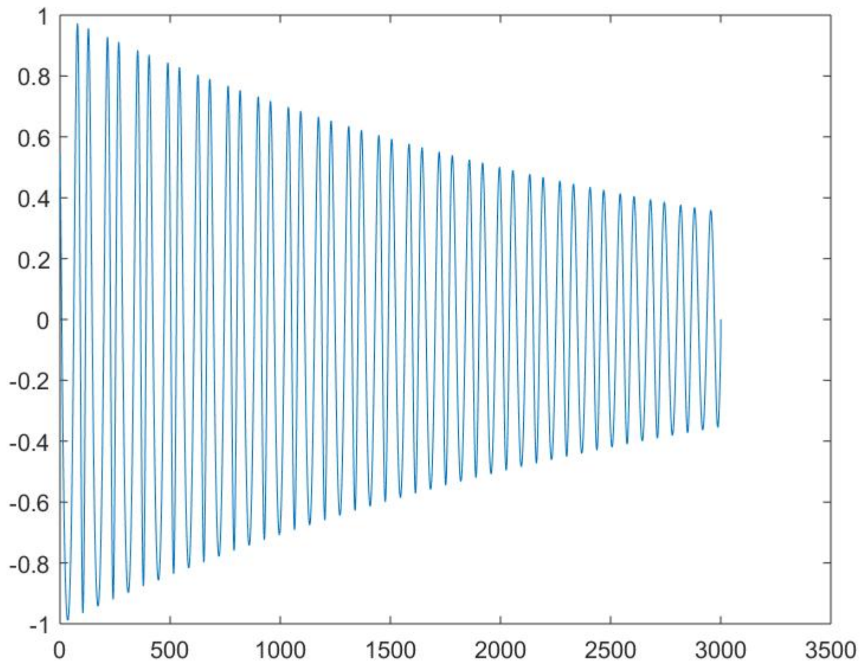

Figure 2: Graph of b-flat eighth note from `myNote.m`

References

De Leon, Phillip 'Computer Music in Undergraduate Digital Processing' Handout. New Mexico State University Klipsch School of Electrical and Computer Engineering. Las Cruces, N.M., n.d. Print.

A MATLAB Code for `myNote.m`

```
function [note] = myNote(noteFreq,time)
smallRest = zeros(1,3);
%amplitude and function
Amp = 2.^(-4*time);
I = @(time) 50.^(-time);

%% output
note = [(Amp.*cos((2*pi*noteFreq*time) + ...
    I(time).*cos(2*pi*(noteFreq/2)*time))) smallRest];
end
```

B MATLAB code for `awakenMyLove.m`

```
% Michael Casio (mkc40)
% ECE 280-07L
% February 16, 2018
% I have adhered to the Duke Community Standard in completing this assingment.

% sets the sampling frequency to 8000 Hz
fs = 8000;

%% Notes
% anonymous functions for different notes
% each note takes in it's appropriate freq #, which
% is it's positioning in the piano and the duration of the note
a = @(freq,time) myNote(27.5*2^freq,time);
bf = @(freq,time) myNote((27.5*2^freq)*2^(1/12),time);
b = @(freq,time) myNote((27.5*2^freq)*2^(2/12),time);
c = @(freq,time) myNote((27.5*2^freq)*2^(3/12),time);
df = @(freq,time) myNote((27.5*2^freq)*2^(4/12),time);
d = @(freq,time) myNote((27.5*2^freq)*2^(5/12),time);
ef = @(freq,time) myNote((27.5*2^freq)*2^(6/12),time);
e = @(freq,time) myNote((27.5*2^freq)*2^(7/12),time);
f = @(freq,time) myNote((27.5*2^freq)*2^(8/12),time);
gf = @(freq,time) myNote((27.5*2^freq)*2^(9/12),time);
g = @(freq,time) myNote((27.5*2^freq)*2^(10/12),time);
```

```
af = @(freq,time) myNote((27.5*2^freq)*2^(11/12),time);
beat = @(time) myBeat(time);
rest = @(time) zeros(1,size(time,2));

%% Time vectors
% vectors for different times; this determines whether a note is an eighth
% note, quarter note, etc.
sixteenth = [0:1/fs:.2-1/fs];
eighth = [0:1/fs:.375-1/fs];
quarter = [0:1/fs:.75-1/fs];
half = [0:1/fs:1.5-1/fs];
threeQ = [0:1/fs:2.25-1/fs];
full = [0:1/fs:3-1/fs];

%% Song Vectors
% vectors for the song's right hand notes, left hand notes, and chords

stayWokeRight = [beat(eighth) rest(quarter) beat(eighth) beat(eighth) ...
    rest(eighth) beat(eighth) rest(quarter) beat(eighth) beat(eighth) ...
    rest(eighth) beat(eighth) rest(quarter) rest(quarter) ...
    rest(zeros(1,3)) bf(4,eighth) af(3,eighth) gf(3,eighth) f(3,eighth) ...
    df(3,eighth) bf(3,eighth) ef(3,full) rest(eighth) rest(zeros(1,6))...
    df(3,eighth) ef(3,eighth) gf(3,eighth) df(4,eighth) bf(4,quarter) ...
    rest(zeros(1,3006)) f(3,eighth) gf(3,eighth) af(3,eighth) gf(3,full)...
    rest(eighth) rest(zeros(1,15018)) bf(4,eighth) af(3,eighth) ...
    gf(3,eighth) f(3,eighth) df(3,eighth) bf(3,eighth) ef(3,full)...
    rest(eighth) rest(zeros(1,6)) df(3,eighth) ef(3,eighth) gf(3,eighth)...
    df(4,eighth) bf(4,quarter) rest(zeros(1,3006)) gf(3,eighth) ...
    af(3,eighth) gf(3,eighth) ef(3,full) rest(zeros(1,3003)) rest(eighth)...
    rest(zeros(1,6009)) af(3,eighth) af(3,eighth) rest(eighth) ...
    rest(zeros(1,12015)) gf(3,eighth) bf(4,eighth) b(4,eighth) ...
    bf(4,eighth) af(3,eighth) af(3,eighth) gf(3,eighth) gf(3,eighth) ...
    rest(eighth) rest(zeros(1,3)) af(3,eighth) af(3,eighth) bf(4,eighth)...
    rest(quarter) rest(zeros(1,6012)) gf(3,eighth) bf(4,eighth) ...
    b(4,eighth) bf(4,eighth) af(3,eighth) af(3,eighth) gf(3,eighth) ...
    gf(3,eighth) gf(3,eighth) af(3,eighth) af(3,eighth) bf(4,quarter) ...
    rest(zeros(1,6009)) gf(3,eighth) bf(4,eighth) b(4,eighth) ...
    bf(4,eighth) af(3,eighth) af(3,quarter) rest(zeros(1,3)) ...
    ef(3,eighth) af(3,full)];

stayWokeLeft = .6.*[rest(zeros(1,48021)) b(2,eighth) ...
    rest(zeros(1,3003)) gf(1,eighth) b(2,eighth) df(2,eighth) ...
    af(1,eighth) df(2,eighth) ef(2,full) bf(1,eighth) rest(zeros(1,3)) ...
    ef(2,eighth) df(2,eighth) df(2,eighth) b(2,eighth) b(2,eighth) ...
    gf(1,eighth) b(2,eighth) df(2,eighth) af(1,eighth) df(2,eighth) ...
    ef(2,full) bf(1,eighth) ef(2,eighth) df(2,eighth) df(2,eighth) ...
    b(2,eighth) b(2,eighth) rest(zeros(1,3003)) gf(1,eighth) ...
    b(2,eighth) df(2,eighth) af(1,eighth) df(2,eighth) ef(2,full) ...
    bf(1,eighth) rest(zeros(1,3)) ef(2,eighth) df(2,eighth) ...
    df(2,eighth) b(2,eighth) b(2,eighth) gf(1,eighth) b(2,eighth) ...
    df(2,eighth) af(1,eighth) df(2,eighth) ef(2,eighth) ef(2,full) ...
    bf(2,eighth) bf(2,eighth) bf(2,eighth) rest(zeros(1,6006)) ...
    gf(1,eighth) b(2,eighth) df(2,eighth) af(1,eighth) df(2,eighth) ...
    ef(2,eighth) ef(2,quarter) rest(zeros(1,3)) ef(2,quarter) ...
    rest(zeros(1,3)) ef(2,eighth) df(2,eighth) df(2,eighth) b(2,eighth)...
    b(2,eighth) rest(zeros(1,3)) gf(1,eighth) b(2,eighth) df(2,eighth)...
    af(1,eighth) df(2,eighth) ef(2,eighth) rest(eighth) ef(2,eighth)...
    rest(zeros(1,3003)) ef(2,eighth) rest(zeros(1,3003)) ef(2,eighth) ...
    df(2,eighth) df(2,eighth) b(2,eighth) b(2,eighth) rest(zeros(1,3003)...
    gf(2,eighth) b(2,eighth) df(2,eighth) af(2,eighth) df(2,eighth) ...
    ef(2,eighth) rest(zeros(1,3003)) ef(2,eighth) rest(zeros(1,3003)) ...
    ef(2,eighth) df(2,eighth) df(2,eighth) b(2,full)];

stayWokeChords = [rest(zeros(1,279174)) ...
(gf(2,half)+ef(2,half)+c(1,half))./3 rest(zeros(1,9))...
(af(2,half)+f(2,half)+df(1,half))./3 rest(zeros(1,9)) ...
(bf(2,half)+gf(2,half)+ef(2,half))./3 rest(zeros(1,12021)) ...
(gf(2,half)+ef(2,half)+c(1,half))./3 rest(zeros(1,2991)) ...
(af(2,half)+f(2,half)+df(1,half))./3 rest(zeros(1,21051)) ...
(gf(2,half)+ef(2,half)+c(1,half))./3 rest(zeros(1,9018)) ...
(bf(2,half)+gf(2,half)+ef(2,half))./3 rest(zeros(1,33021))];

%% Final Vector and Plots
fVec = (stayWokeRight+stayWokeLeft+stayWokeChords)./3; % final song vector
% plots the left, right, and chord hands, as well as a graph of a single B
% flat eighth note
```

```
hold on
figure(1);
plot(stayWokeRight);
plot(stayWokeLeft);
plot(stayWokeChords);
figure(2);
plot(bf(2,eighth));

% plays the left, right, and chord hands of the song
soundsc(stayWokeLeft,fs);
soundsc(stayWokeRight,fs);
soundsc(stayWokeChords,fs);

%% Save to File
% saves the song to a .wav file
filename = 'Casio_Redbone.wav';
audiowrite(filename,fVec,fs);
clear fVec fs
```