EE 341 Summer 2018
Lab 1 Elementary Music Synthesis

Professor: Tai-Chang Chen

July 2nd, 2018

He Feng 1427841
Hanling Lei 1673094
Zikai Yang 1435493

Introduction

The purpose of this lab is to improve the quality of the music. We used a part of some music which contains several notes as an example of this lab, and we initially use Matlab to play the music directly, then we used two methods to improve the sound quality. The first method is about volume variation, we write a separate script as the window function, which introduce the variation of the volume over time. In window function, we divided the time into Attack, Decay, Sustain, and Release segments. We times the sinusoidal function, which is the function for the music notes, times the window function to get the new function for the music. The second method relates to overlapping tones. We start to play each note while the previous note is decaying, and we defined each note separately by using zeros function such that each note can have the longer signal with same length. We add them together to build a new signal. Both of these two methods can improve the sound quality, and the sound is more smooth and stable after the improvement.

Results and Discussion

Exercise #1:

In the first exercise, the lab asks us to synthesis the piece of music which contains eight notes show in the previous page. We also need to play it back by using the sound function at the end of the code. Initially we clear all variables and close all windows, and then we define the sampling frequency and its belong period. After the beginning steps, we defined three time vectors, which represent whole note, half note and fourth note:

```
t1 = 0:(1/Fs):(0.5-1/Fs);
t2 = 0:(1/Fs):(1-1/Fs);
t4 = 0:(1/Fs):(2-1/Fs);
```

We used 0.5-1/Fs instead of 0.5 because we want the number of samples to be 4000 such that it will be more convenient in the following exercises. There are 4000, 8000, and 16000 samples in each whole, half, and fourth note.

Later we set the frequency of based on the chart given in spec, and we build a function called “pause” which can bring an empty period after each movement. Then we build the sinusoidal waveform of each note by using sinusoidal function. In this lab, we choose to use sin function as the sinusoidal waveform. Based on the equation y = sin(wt), because $w = 2\pi f$, therefore the final Equation is $y = \sin(2\pi f t)$. Here is an example of the sinusoidal function of one of the note:

A2 = sin(2 * pi * fa * t2);

Finally, we combine the notes together define the piece of music, and we add pause between each note such that our ears can hear the sound clearly. We used sound function to play the music.

Exercise #2:

After synthesis the piece of music, we now are going to make the quality better. As we mentioned in the introduction, the method in exercise 2 relates to the volume variation. The variation over time contains four segments: Attack, Decay, Sustain, and Release (ADSR). We initially build a separate script to represent the window function. In the separate script we mainly use linspace function to demonstrate the volume changing, and we used ruler in the classroom to measure the length of each segments so that we can scale them later. For example, A = linspace(0, 9.2, n/10) represents the attack status. There are three inputs in it. The first input represent the initial y value, which is the leftmost point y value. The second point is the rightmost point y value. The third value is the difference between two x-values. In this lab we used rule to measure the data, and this cause the size of the sum of A, D, S, and R is not equal to the size of ADSR. Therefore we added one point to the third input of R to balance the size.

After we build the window script, we started to build the main script of the second exercise. The second exercise is similar to the first exercise, and we only need to use window function to define each note again and get the new version of each note. Then we defined the piece of music by using the new notes, and used sound function to play the music.

The music in this case is smoother and clearer. The sound quality has been improved. To make the problem easier to solve and clearer to view, we used size function to determine the size of each factor such that we can see the problem easier.

Exercise #3:

The last exercise if also aim at improving the sound quality of the piece of music in the first exercise. We write this script based on exercise 2, and we added some codes to make the sound even better. As the spec introduced, the variation of volume can be divided into four segments. Therefore if we can let a note start to attack while the previous note is decaying, the sound quality will be improved. In this case, the decaying I mentioned is not the second stage of the four segments, it is actually some point in R segment.

Based on the idea above, we defined each note separately and we defined them with same size. We put zeros before or after each note to make each individual signal has the equal length. Then we added them together and form a new signal. The new signal is the signal which we are going to play later. There are nine signals in this case, and the overlapping can make the overall sound quality better. After using sound function to play the new piece of music, we heard the sound smoother and better.

Conclusion:

The purpose of this lab is to learn the methods to improve the sound quality. In the first exercise, we used matlab to play a piece of music which contain several notes. In the second exercise, we multiply the notes by the window function such that the volume variation can improve the sound quality. We wrote another script as the window function, and put them into the same directory in the computer. In the final exercise, we add some code to overlap the notes based on the codes in exercise two. If we start each note when the previous notes is decaying, the sound quality can be improved as well.

We got the ideal result in these three exercises, and the method of volume variation and overlapping can both improve the sound quality. Even though we had some problems while we were coding such that Matlab always told us the size of two variables does not match. Then we used size function to measure the size of each variable and then figured out the way to solve the problems. We keep some comments in the script to show the logic we followed to do this lab. This lab is interesting and it is also a good start for the following lab sections.

Attached Code:

Ex1.m

```
% He Feng; Zikai Yang; Hanling Lei

%This script will play a piece of music which contain several notes.

% Clear all variables and close all windows.
clear all;
close all;

% Define the sampling frequency and its belonging period.
Fs = 8000;
Ts = 1/8000;

% To define the time vectors.
t1 = 0:(1/Fs):(0.5-1/Fs);
t2 = 0:(1/Fs):(1-1/Fs);
t4 = 0:(1/Fs):(2-1/Fs);

% Set the frequency of each note.
fa = 220;
fb = 220 * 2^(2/12);
fc = 220 * 2^(3/12);
fe = 220 * 2^(7/12);

% Use a function to define pause, which means there
% is an empty period after each movement.
pause = zeros(1, 0.02 * Fs);
```

```
% Build the sinusoidal waveform of each note.
A2 = sin(2 * pi * fa * t2);
A1 = sin(2 * pi * fa * t1);
E1 = sin(2 * pi * fe * t1);
B1 = sin(2 * pi * fb * t1);
C1 = sin(2 * pi * fc * t1);
A4 = sin(2 * pi * fa * t4);

% Define the music with note one by one, and each of them has a 'pause'
% function between them.
music = [A2, pause, A1, pause, E1, pause, E1, pause, E1, pause, B1, pause, C1, pause, B1, pause, A4];

% Play the music.
sound(music, Fs);
```

Ex2.m

```
% He Feng; Zikai Yang; Hanling Lei

% This script based on Ex1, and we used the method of volume
% variation to improve the sound quality.

% Clear all variables and close all windows.
clear all;
close all;

% Define the sampling frequency and its belonging period.
Fs = 8000;
Ts = 1/8000;

% To define the time vectors.
t1 = 0:(1/Fs):(0.5-1/Fs);
t2 = 0:(1/Fs):(1-1/Fs);
t4 = 0:(1/Fs):(2-1/Fs);

% Set the frequency of each node.
fa = 220;
fb = 220 * 2^(2/12);
fc = 220 * 2^(3/12);
fe = 220 * 2^(7/12);

% Use a function to define pause, which means there
% is an empty period after each movement.
pause = zeros(1, 0.02 * Fs);

% Build the sinusodal waveform of each node.
A2 = sin(2 * pi * fa * t2);
A1 = sin(2 * pi * fa * t1);
E1 = sin(2 * pi * fe * t1);
B1 = sin(2 * pi * fb * t1);
C1 = sin(2 * pi * fc * t1);
A4 = sin(2 * pi * fa * t4);

% Apply ADSR to each node.
new_A2 = window(A2);
new_A1 = window(A1);
new_E1 = window(E1);
new_B1 = window(B1);
new_C1 = window(C1);
new_A4 = window(A4);

% Define the series of nodes one by one with a pause between each two
% nodes.
music = [new_A2, pause, new_A1, pause, new_E1, pause, new_E1, pause, new_E1, pause, new_B1, pause, new_C1, pause, new_B1, pause, new_A4];

% Play the music.
sound(music, Fs);
```

window.m

```
% He Feng; Zikai Yang; Hanling Lei

% This script multiply the sinusoid waveform with the window function
% which will be used as the new signal.

function mod = window(x)

% Length of the sinusoidal function.
n = length(x)

% Define each state of the window function
A = linspace(0,9.2,n/10);
D = linspace(9.2,6.9,n/10);
S = linspace(6.9,6.9,7*n/15);
R = linspace(6.9,0,n/3+1);

% Define the window function
ADSR = [A,D,S,R]

%size(A)
%size(D)
%size(S)
%size(E)
%size(x)
%size(ADSR)

% Times the sinusoidal function with the window function
mod = x.*ADSR

```

Ex3.m

```
% He Feng; Zikai Yang; Hanling Lei

% This script based on Ex2, and we used overlapping to make
% the sound quality even better.

% Clear all variables and close all windows.
clear all;
close all;

% Define the sampling frequency and its belonging period.
Fs = 8000;
Ts = 1/8000;

% To define the time vectors.
t1 = 0:(1/Fs):(0.5-1/Fs);
t2 = 0:(1/Fs):(1-1/Fs);
t4 = 0:(1/Fs):(2-1/Fs);

% Set the frequency of each node.
fa = 220;
fb = 220 * 2^(2/12);
fc = 220 * 2^(3/12);
fe = 220 * 2^(7/12);

% Use a function to define pause, which means there
% is an empty period after each movement.
pause = zeros(1, 0.02 * Fs);

% Build the sinusodal waveform of each node.
A2 = cos(2 * pi * fa * t2);
A1 = cos(2 * pi * fa * t1);
E1 = cos(2 * pi * fe * t1);
B1 = cos(2 * pi * fb * t1);

 C1 = cos(2 * pi * fc * t1);
 A4 = cos(2 * pi * fa * t4);

 % Apply ADSR to each node.
 new_A2 = window(A2);
 new_A1 = window(A1);
 new_E1 = window(E1);
 new_B1 = window(B1);
 new_C1 = window(C1);
 new_A4 = window(A4);

 % Define each node of the music.
 node_0 = [new_A2,zeros(1,44000)];
 node_1 = [zeros(1,7000),new_A1,zeros(1,41000)];
 node_2 = [zeros(1,7000+3500),new_E1,zeros(1,37500)];
 node_3 = [zeros(1,7000+3500*2),new_E1,zeros(1,34000)];
 node_4 = [zeros(1,7000+3500*3),new_E1,zeros(1,30500)];
 node_5 = [zeros(1,7000+3500*4),new_B1,zeros(1,27000)];
 node_6 = [zeros(1,7000+3500*5),new_C1,zeros(1,23500)];
 node_7 = [zeros(1,7000+3500*6),new_B1,zeros(1,20000)];
 node_8 = [zeros(1,7000+3500*7),new_A4,zeros(1,4500)];
```

```
    %size(ex_0)
    %size(ex_1)
    %size(ex_2)
    %size(ex_3)
    %size(ex_4)
    %size(ex_5)
    %size(ex_6)
    %size(ex_7)
    %size(ex_8)

    % Summation the overlap song signal by adding the defined nodes above.
    music = node_0+node_1+node_2+node_3+node_4+node_5+node_6+node_7+node_8;


    %size(music)

    % Play the final signal
    sound(music,Fs);

```