Mini Project Report

SID: 12012505
NAME: 占陈郅

I. Introduction

Music files come in many formats, but the main part of the data, which is saved, is an M * N matrix (where N is the number of channels).
Thus, for a musical score, we can generate such a matrix with a computer program and play it out with the computer. The objectives of this project are.
1. to understand the basic elements of music, such as pitch, meter, timbre, fundamental frequency, chords, etc.
2. to understand how these musical elements are reflected in our data.
3. transforming a piece of sketch music into a piece of data that can be played as music, through a MATLAB program.
4. generate playable music based on the data.
5. generate music that imitates a certain instrument based on the data

II. Result & Analysis

1. Build function: *tone2freq*

First of all, a Matlab function should be built to generate frequency with input of *tone, scale, noctave, rising.* These four inputs represent notes, tune, number of high or low octaves, ascending or descending.

- tone2freq.m

```
function freq = tone2freq(tone, scale, noctave, rising)
% tone: 输入数字音符，数值范围 1 到 7
% scale: 调号，决定了 f0 的大小
% noctave: 高或低八度的数量，数值范围整数。0 表示中音，正数表示高 noctave 个八度，负数为低 noctave 个八度
% rising: 升或降调。1 为升，-1 为降，0 无升降调
% freq 为输出的频率
switch(scale)
    case'A'
        f0=440;
    case'B'
        f0=494;
    case'C'
        f0=261.5;
    case'D'
        f0=293.5;
    case'E'
        f0=329.5;
    case'F'
        f0=349;
    case'G'
        f0=391.5;
end
f0 = f0*(2.^noctave);%高低八度
```

```
dur = [0,2,3,5,7,8,10];
if tone == 0
    freq = 0;
else
    freq = f0*(2.^((dur(tone)+rising)/12));
end
end
```

- ANALYSIS

This function is the core of generating music, but it is not difficult. The focus is on understanding how to convert notes, key signatures, and elevations into intervals and then into frequencies. First determine the key number and thus f0, then create a new *dur* array to store the positions of the 7 notes in the mean meter through a table of 12 mean meters, and then calculate the final frequency through an exponential equation.

2. Build function: *gen_wave*

In the second step, the frequency generated by *tone2freq* is generated into a playable frequency waveform.

- gen_wave.m

```
function waves = gen_wave(tone, scale, noctave, rising, rhythm)
% tone 为数字音符，scale 为调号, noctave 为高低八度数量，rising 为升降调，rhythm 为节拍，即每个音符持续时长，fs 为采样频率，
fs = 44100; %采样频率默认为 8192Hz

freq = tone2freq(tone, scale, noctave, rising);%计算音的频率


x=linspace(0,2*pi*rhythm,rhythm*fs);%增加时值

y=0.7*sin(freq.*x)+0.2*sin(2.*freq.*x)+0.1*sin(5.*freq.*x);%输出包络 sin 波
waves = y.*exp(-x/rhythm);%音的自然消散

end
```

- ANALYSIS

What this function does is first call the *tone2freq* method to figure out the frequency of each note, and then create *linespace* from the input rhythm and sample frequency (here I set *fs* as a local variable) to envelope the output of the freq and do some processing on the sound.

1. Volume fluctuation

The method *waves = y.*exp(-x/rhythm);* of natural dissipation of each tone is given in Lab Manual, which I think greatly optimizes the listening experience, and I wrote a separate test to explore the natural dissipation of the visualized sound.

- test.m

```
%这个 test 用于探究无声音自然消散和有声音自然消散体现在包络上的区别
fs = 44100; %采样频率默认为 8192Hz
rhythm = 1;
freq = 440;

x=linspace(0,2*pi*rhythm,rhythm*fs);%增加时值

y=0.7*sin(freq.*x)+0.2*sin(2.*freq.*x)+0.1*sin(5.*freq.*x);%输出包络 sin 波
figure(1)
subplot(211)
plot(x,y);
```

```
axis([0,2*pi,-2,2]);

subplot(212)
plot(x,y);
axis([0,0.1*pi,-2,2]);

waves = y.*exp(-x/rhythm);%音的自然延长
figure(2)
subplot(211)
plot(x,waves);
axis([0,2*pi,-2,2]);

subplot(212)
plot(x,waves);
axis([0,0.1*pi,-2,2]);
```

- Result

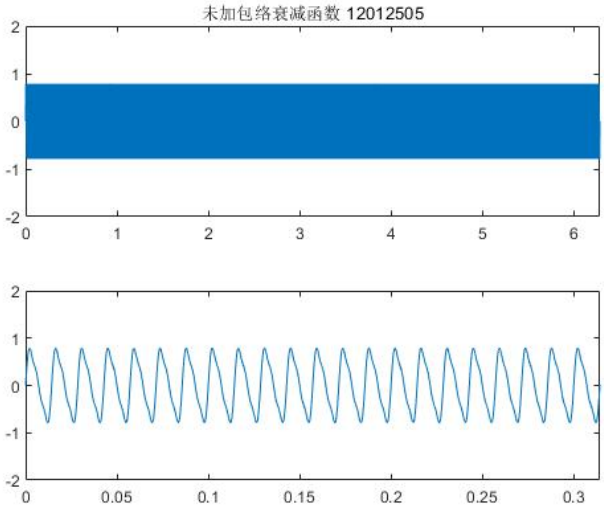

Fig.1 No envelope decay function added

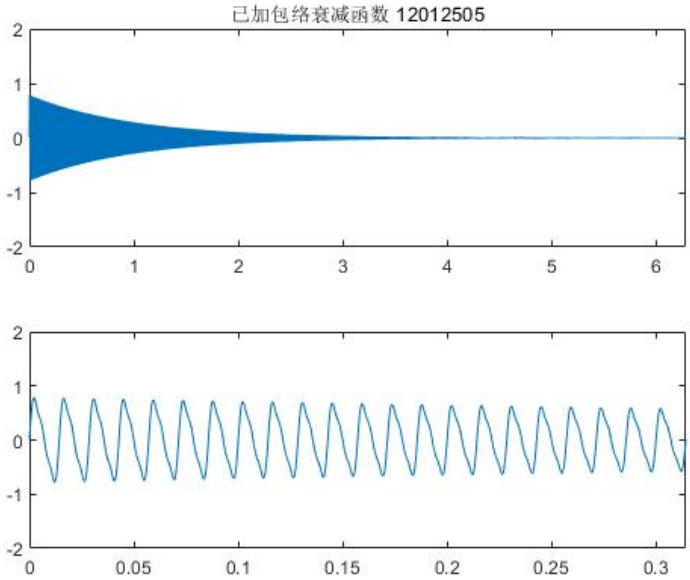

Fig.2 Envelope decay function added

With the addition of the envelope decay function, the sound becomes smoother and the decay feels more in tune with the sound of real instruments, in line with what we are used to hearing on a daily basis.

2. Differences in Overtones/timbre

The method *y=0.7*sin(freq.*x)+0.2*sin(2.*freq.*x)+0.1*sin(5.*freq.*x);* of differences in Overtones/timbre of different instruments is also mentioned in Lab. The instrument produces harmonic frequencies including the fundamental frequency and several integer multiples, with the main energy concentrated in the fundamental frequency. For harmonic frequencies of 2x, 3x, 4x, and 5x ....., the proportion of energy in these harmonic frequencies varies from instrument to instrument.

By browsing on the Internet, I learned that:

Tenor: fundamental 131-494, harmonic 1k-12k

Alto: fundamental 175-698, harmonic 2k-12k

Soprano: fundamental 247-1175, harmonic 12k-12k

Bass vocal: fundamental 28-4196, harmonic 1k-12k

Piano: fundamental 28-4196, harmonic 5k-8k

...

I tweaked it several times to my liking and determined the above ratio to approximate the sound of an electric piano.

3. Build function: *gen_music*

- gen_music.m

```
fs = 44100;
rhythm = 0.5;
s = [];
a = gen_wave(6, 'D', 0,0, rhythm*0.5);
s = [s a];
a = gen_wave(7, 'D', 0,0, rhythm*0.5);
s = [s a];
a = gen_wave(1, 'D', 1,0, rhythm*1.5);
s = [s a];
a = gen_wave(7, 'D', 0,0, rhythm*0.5);
s = [s a];
a = gen_wave(1, 'D', 1,0, rhythm*1);
s = [s a];
a = gen_wave(3, 'D', 1,0, rhythm*1);
s = [s a];
a = gen_wave(7, 'D', 0,0, rhythm*3);
s = [s a];
a = gen_wave(3, 'D', 0,0, rhythm*0.5);
s = [s a];
a = gen_wave(3, 'D', 0,0, rhythm*0.5);
s = [s a];
a = gen_wave(6, 'D', 0,0, rhythm*1.5);
s = [s a];
a = gen_wave(5, 'D', 0,0, rhythm*0.5);
s = [s a];
a = gen_wave(6, 'D', 0,0, rhythm*1);
s = [s a];
a = gen_wave(1, 'D', 1,0, rhythm*1);
s = [s a];
a = gen_wave(5, 'D', 0,0, rhythm*2);
s = [s a];
a = gen_wave(0, 'D', 0,0, rhythm*1);
s = [s a];
a = gen_wave(3, 'D', 0,0, rhythm*0.5);
s = [s a];
```

```
a = gen_wave(3, 'D', 0,0, rhythm*0.5);
s = [s a];
a = gen_wave(4, 'D', 0,0, rhythm*1.5);
s = [s a];
a = gen_wave(3, 'D', 0,0, rhythm*0.5);
s = [s a];
a = gen_wave(4, 'D', 0,0, rhythm*0.5);
s = [s a];
a = gen_wave(1, 'D', 1,0, rhythm*1.5);
s = [s a];
a = gen_wave(3, 'D', 0,0, rhythm*2);
s = [s a];
a = gen_wave(0, 'D', 0,0, rhythm*0.5);
s = [s a];
a = gen_wave(1, 'D', 1,0, rhythm*1.5);
s = [s a];
a = gen_wave(7, 'D',0 ,0, rhythm*1.5);
s = [s a];
a = gen_wave(4, 'D',0 ,1, rhythm*0.5);
s = [s a];
a = gen_wave(4, 'D',0 ,1, rhythm*1);
s = [s a];
a = gen_wave(7, 'D',0 ,0, rhythm*1);
s = [s a];
a = gen_wave(7, 'D',0 ,0, rhythm*2);
s = [s a];
a = gen_wave(6, 'D',0 ,0, rhythm*0.5);
s = [s a];
a = gen_wave(7, 'D',0 ,0, rhythm*0.5);
s = [s a];
a = gen_wave(1, 'D', 1,0, rhythm*1.5);
s = [s a];
a = gen_wave(7, 'D', 0,0, rhythm*0.5);
s = [s a];
a = gen_wave(1, 'D', 1,0, rhythm*1);
s = [s a];
a = gen_wave(3, 'D', 1,0, rhythm*1);
s = [s a];
a = gen_wave(7, 'D', 0,0, rhythm*3);
s = [s a];
a = gen_wave(3, 'D', 0,0, rhythm*0.5);
s = [s a];
a = gen_wave(3, 'D', 0,0, rhythm*0.5);
s = [s a];
a = gen_wave(6, 'D', 0,0, rhythm*1.5);
s = [s a];
a = gen_wave(5, 'D', 0,0, rhythm*0.5);
s = [s a];
a = gen_wave(6, 'D', 0,0, rhythm*1);
s = [s a];
a = gen_wave(1, 'D', 1,0, rhythm*1);
s = [s a];
a = gen_wave(5, 'D', 0,0, rhythm*3);
s = [s a];
a = gen_wave(0, 'D', 0,0, rhythm*0.5);
s = [s a];
a = gen_wave(3, 'D', 0,0, rhythm*0.5);
s = [s a];
sound(s,fs);
audiowrite('q3_output.wav', s, fs);
```

In this file, you need to input the information of each note of the score into the gen_wave function one by one.

III. Conclusion

In this project I got an insight into how to use matlab to generate playable music from input information, and there are many details of the simulation that need attention. However, I was also confused about some issues during the experiment.

- The time value of each note in 4/4 time is 0.25 seconds, but the music I got by assigning rhythm to 0.25 is faster than the original song I heard, and rhythm is assigned to 0.5 to match my impression of the original song. The reason for this problem may be that Lab has given me a different sketch or one of my functions has a problem with rhythm.
- In the QQ group, the assistant teacher said that I need to adjust the fs from 8192 to 44100Hz, otherwise there will be noise. But I tried these two values in the experiment, the output music sounds exactly the same, there is no difference.
- for the overtone / tone adjustment I seem to have not been able to fully grasp the method, mainly to test the exhaustive list to try their luck.