Digital Music Synthesis Using MATLAB

Yulin Liu*
*Southern University of Science and Technology, Shenzhen, China
Email: 12211351@mail.sustech.edu.cn

***Abstract*—The project explores the digital generation and playback of music using MATLAB. Using numbered musical notation, music is encoded into a matrix representation, enabling the creation of waveforms corresponding to musical notes. My report involves designing functions for note frequency calculation, waveform generation, and integrating these into complete musical compositions. Additionally, experiments aim to simulate realistic instrument sounds. The results demonstrate the feasibility of using computational tools for music generation.**

***Index Terms*—Numbered musical notation, Matlab.**

I. Introduction

With advancements in digital signal processing (DSP), it is possible to represent and music using computational methods. The project aims to bridge the gap between traditional music notation and modern audio processing by using MATLAB to synthesize music from numbered musical notations. My focus is on understanding how music theory elements translate into digital data and how this data can be processed to generate music by encoding musical notations into matrices, generating corresponding waveforms, and experimenting with sound attributes like decay and harmonic ratios.

Notations:

II. Numbered Musical Notation

A. Tone

To accurately interpret the numbered musical notation and extract precise musical information, a method to convert tones to their corresponding frequencies was developed. This method is implemented in the function tone2freq, which outputs the frequency of tones 1 to 7 while accounting for variations such as octave shifts (higher or lower) and tonal adjustments like sharps or flats. By using 440 Hz as the reference frequency for tone 1, this function allows for flexible and accurate frequency computation based on the input parameters.

```
function freq = tone2freq(tone,noctave,
    ↪ rising)
base = 440;
n = [0,2,4,5,7,9,11];
if (tone == 0)
    freq = 0;
else
    freq = base*(2.^(n(tone)/12))*(2.^
        ↪ noctave) * (2.^(rising/12));
end
```

The inputs of the function tone2freq are tone, scale, noctave, and rising. The parameter scale specifies the musical scale, noctave indicates whether the tone is in a higher or lower octave (values: 1 for higher, 0 for middle, and -1 for lower octave), and rising denotes the pitch adjustment (values: -1 for flat, 0 for no change, and 1 for sharp). The output freq represents the calculated frequency of the note.

For musical notes 1 to 7, each note has a predefined semitone interval relative to note 1. For instance, note 1 and note 1 have an interval of 0 semitones, while note 2 and note 1 have an interval of 2 semitones. These intervals are stored in the array n. Using these intervals and the provided inputs, the frequency is calculated based on a standard formula.

B. Scale

Compared with the method in the last part, a variable scale is implemented in the method tone2freq. When the scale changes, the base varies.

```
    function freq = tone2freq(tone,scale,
        ↪ noctave,rising)
switch(scale)
    case'C'
        base = 523;
    case'D'
        base = 587;
    case'E'
        base = 659;
    case'F'
        base = 698;
    case'G'
        base = 784;
    case'A'
        base = 880;
    case'B'
        base = 988;
end
n = [0,2,4,5,7,9,11];
if (tone == 0)
    freq = 0;
else
    freq = base*(2.^(n(tone)/12))*(2.^
        ↪ noctave) * (2.^(rising/12));
end
```

III. Generate sine wave

On the basis of function tone2freq shown in the last part, two input parameters rhythm and fs are implemented.

```
function waves = gen_wave(tone, scale,
    ↪ noctave, rising, rhythm , fs)
    f = tone2freq(tone, scale , noctave ,
        ↪  rising);
    t = linspace(0, rhythm, fs*rhythm);
    waves = sin(2*pi*f*t);
```

The function gen_wave is used to generate wave of a note, and based on this function, we can generate the wave of the whole song.

Besides the given music **The Castle of Sky**, I choose another Jay Chou's music, **Sunny Day**. The numbered musical notation is shown. In order to clearly display all information in the notation, I use an excel file to record four parameters, which are tone, noctave, rising, and rhythm. Then I use function readmatrix to import the information of each note. Finally, I use gen_wave method to generate the whole wave of this song.

Fig. 1. Numbered Musical Notation of 'Sunny Day'

```
data = readmatrix("qingtian.xlsx");
rhythm = 1;
fs = 8192;
scale = 'G';
N = length(data(:, 1));
y = [];
for n = 1:N
    y = [y gen_wave(data(n, 1), scale,
        ↪ data(n, 2), data(n, 3), data(n
        ↪ , 4)*rhythm, fs)];
end
sound(y, fs)
figure
plot(y)
audiowrite("qingtian.wav", y, fs,
    ↪ BitsPerSample=16);
```


qingtian.xlsx

	A	B	C	D
	qingtian			
	tone	noctave	rising	rhythm
	数值	数值	数值	数值
1	tone	noctave	rising	rhythm
2	6	-1	0	0.5000
3	1	0	0	0.5000
4	5	0	0	0.5000
5	1	0	0	0.5000
6	4	-1	0	0.5000
7	5	-1	0	0.2500
8	6	-1	0	0.2500
9	5	0	0	0.5000
10	1	0	0	0.5000
11	1	-1	0	0.5000
12	5	-1	0	0.5000
13	5	0	0	0.5000
14	1	0	0	0.5000
15	1	-1	0	0.5000

Fig. 2. Notes Information Record Example

The above procedures are also implemented for the piece **The Castle in the Sky**.

```
data = readmatrix("sky.xlsx");
rhythm = 0.5;
fs = 8192;
scale = 'D';
N = length(data(:, 1));
y = [];
for n = 1:N
    y = [y gen_wave(data(n, 1), scale,
        ↪ data(n, 2), data(n, 3), data(n
        ↪ , 4)*rhythm, fs)];
end
sound(y, fs)
figure
plot(y)
audiowrite("sky.wav", y, fs,
    ↪ BitsPerSample=16);
```

So in this part, I generate two wav files, one for **Sunny day**, another for **The Castle of Sky**.

IV. Choices of Decay Function

In order to better simulates the generation of music in real life, where the energy of a wave will decay. I will apply a decay function on the wave I have already generated. I compare exponential, linear and square decay with no decay conditions.

```
fs = 8192;
f = 440;
rhythm = 1;
```

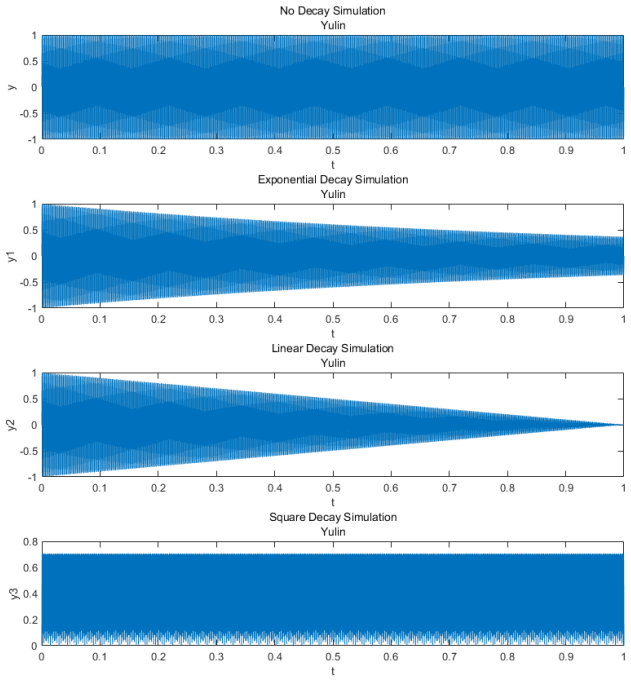

Fig. 3. Different Decay Functions Comparison

```
t = linspace(0, rhythm, fs*rhythm);
y = sin(2*pi*f*t);
figure
subplot(4,1,1)
plot(t,y)
title(["No Decay Simulation";"Yulin"])
xlabel('t')
ylabel('y')
y1 = y.*exp(-t/rhythm);
subplot(4,1,2)
plot(t,y1)
title(["Exponential Decay Simulation","
    ↪ Yulin"])
xlabel('t')
ylabel('y1')
subplot(4,1,3)
y2 = y.*(1-t/(rhythm));
plot(t,y2)
title(["Linear Decay Simulation";"Yulin
    ↪ "])
xlabel('t')
ylabel('y2')
y3 = (y.^2)*0.5;
y3 = sqrt(y3);
subplot(4,1,4)
plot(t,y3)
title(["Square Decay Simulation";"Yulin
    ↪ "])
xlabel('t')
ylabel('y3')
```

Without decay, the sound maintains a constant amplitude throughout its duration, lacking realism. Exponential decay is the best for me. However, the best decay model depends on the intended musical context. For classical pieces, exponential decay is preferred, while linear or square decay might suit modern electronic music.

V. Harmonic Waves and Simulation of Different Musical Instrument Timbres

The tones we talked about earlier all refer to the fundamental frequency of the music. When playing music with musical instruments, in addition to the fundamental frequency sound in the music scores, different numbers of standing waves are also generated.

Therefore, when a musical instrument is played, it will produce harmonic frequencies including the fundamental frequency and several integer multiple frequencies, and the main energy is concentrated on the fundamental frequency. For harmonic frequencies of 2 times, 3 times, 4 times, 5 times..., the energy ratio of these harmonic frequencies varies from instrument to instrument. If we adjust this ratio, we will produce sound waves with completely different timbres.

```
fs = 8192;
f = 440;
rhythm = 1;
t = linspace(0,rhythm,fs*rhythm);

y = zeros(1,length(t));
a = [0.8,0.1,0.05,0.05];
for n = 1:4
    y = y + a(n) * sin(2*pi*f*n.*t);
end
figure
subplot(2,1,1)
plot(t,y)
title(["The Wave Praph of Piano Timbres
    ↪ ";"Yulin"])
xlabel('t')
ylabel('y')
xlim([0,0.01])
subplot(2,1,2)
Y = fft(y);
N = length(Y);
w = (-N/2:N/2-1)*fs/N;
plot(w,abs(fftshift(Y)))
xticks([-1760,-1320,-880,-440,0,440,880,
1320,1760])
xticklabels({'-4f','-3f','-2f','-f','0','
    ↪ f','2f','3f','4f'})
title(["The Frequency Spectrum ";"Yulin
    ↪ "])
xlabel('f')
ylabel('Y')

y = zeros(1,length(t));
a = [0.5,0.3,0.15,0.05];
for n = 1:length(a)
    y = y + a(n) * sin(2*pi*f*n.*t);
end
figure
subplot(2,1,1)
plot(t,y)
title(["The Wave Graph of Cello Timbres
    ↪ ";"Yulin"])
xlabel('t')
ylabel('y')
```

```
xlim([0,0.01])
subplot(2,1,2)
Y = fft(y);
N = length(Y);
w = (-N/2:N/2-1)*fs/N;
plot(w,abs(fftshift(Y)))
xticks([-1760,-1320,-880,-440,0,440,880,
1320,1760])
xticklabels({'-4f','-3f','-2f','-f','0','
    ↪ f','2f','3f','4f'})
title(["The Frequency Spectrum";"Yulin"])
xlabel('f')
ylabel('Y')
```

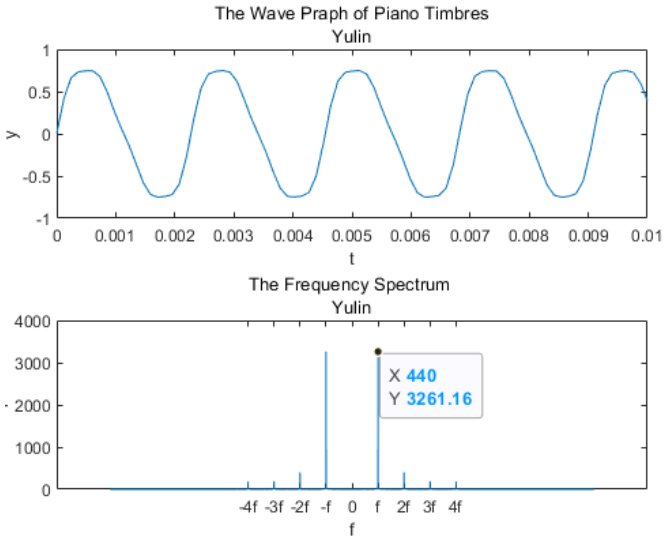

Fig. 4. Simulation of Piano Timbres

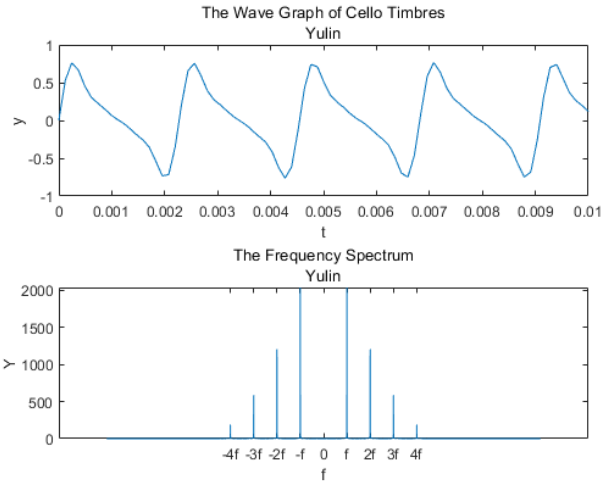

Fig. 5. Simulation of Cello Timbres

I will simulate the timbres of two musical instruments, the piano and the cello, to recreate and play Jay Chou's **Sunny Day**.

```
data = readmatrix("qingtian.xlsx");
rhythm = 1;
fs = 8192;
scale = 'G';
N = length(data(:, 1));
y = [];
energy = [0.8, 0.1, 0.05, 0.05]
for n = 1:N
    y = [y gen_wave_energy_exp(data(n, 1)
        ↪ , scale, data(n, 2), data(n,
        ↪ 3), data(n, 4)*rhythm, fs,
        ↪ energy)];
end
sound(y, fs)
figure
plot(y)
audiowrite("Piano_qingtian.wav", y, fs,
    ↪ BitsPerSample=16);
```

```
data = readmatrix("qingtian.xlsx");
rhythm = 1;
fs = 8192;
scale = 'G';
N = length(data(:, 1));
y = [];
energy = [0.5, 0.3, 0.15, 0.05]
for n = 1:N
    y = [y gen_wave_energy_exp(data(n, 1)
        ↪ , scale, data(n, 2), data(n,
        ↪ 3), data(n, 4)*rhythm, fs,
        ↪ energy)];
end
sound(y, fs)
figure
plot(y)
audiowrite("Cello_qingtian.wav", y, fs,
    ↪ BitsPerSample=16);
```

Then the two final wav files of **Sunny Day** are generated and saved.

VI. Conclusion

In this experiment, I learned some important properties of a song, and how to generate a song by computer. By using different decay functions such as exponential, linear, and square decay, I gained a deeper understanding of how sound dynamics affect the auditory experience. Additionally, I experimented with harmonic components to simulate the timbre of different instruments, such as the piano and cello.