Digital Signal Processing Laboratory Report - Mini Project: Generate and Play the Music by Computer

Haodong Zhang, 12113010

***Abstract*—Using Matlab, this experiment generated a piece of music from a musical score. We began by introducing fundamental music theory concepts, encompassing pitch, key signatures, and note duration. Subsequently, we systematically translated musical symbols from the score into computerized musical data, involving frequency and musical duration. With the obtained frequency and duration, we generated waveform data and could play the music.**

In an effort to simulate a realistic music signal, we considered decay and harmonics. We analyzed various decay functions and harmonic ratios, scrutinizing the differences and effects caused by different parameters. Finally, we attempted to mimic authentic piano sound. We imported and analyzed real piano sounds, adjusting our data parameters by analogizing these attributes. Ultimately, we obtained music resembling the sound of a real piano.

***Index Terms*—Music Generating, Digital Signal Processing, Matlab, Simplified Numeric Musical Score**

I. Introduction

THIS project aims to use Matlab tools to convert a piece of music notation of the numeric musical sore into playable audio and waveform files on a computer. Music files typically come in various formats, but their main data section are usually stored thorough array or matrix. Therefore, for a musical score, we can use computer programs, including Matlab, to transform it into an array or matrix and then play the corresponding music using a computer. Through this project, we will first introduce and understand the basics of music theory, mastering fundamental elements of music, including tone, rhythm, timbre, fundamental frequency, and chords. We will then convert and represent these basic elements in our data, allowing us to play the corresponding music based on the data. To mimic the real piano timbre, this project explores how to adjust parameters to make the generated music sound closer to a real piano.

Specifically, this project achieved the following objectives: 1. Calculated the frequency corresponding to each note based on the numbers and the pitch alterations in the musical score. 2. Generated waveform data for the entire music score in a simplified musical notation and converted this waveform data into a music file (.wav format). 3. Explored various envelope decay functions because actual music does not consist of constant amplitude waveforms. Experimented with different decay functions to observe their impact on the perception of music. 4. Explored the existence of harmonics beyond the fundamental frequency indicated in the musical score (which will be elaborated in subsequent sections). Different coefficients (known as harmonic energy ratios) correspond to different harmonics in various music pieces. Thus, the project involved investigating the impact of selecting different harmonic energy ratios on the perception of music. 5. Aimed to make the generated music closely resemble that produced by real instruments. For this purpose, the project specifically selected the timbre of a piano and studied how to adjust the aforementioned parameters to make the generated music more closely resemble the timbre of a piano.

This is a lab report of the Digital Signal Processing (EE323) in the Southern University of Science and Technology (SUSTech).

Haodong Zhang is a undergraduate student majoring in communication engineering, Department of Electronic and Electrical Engineering, Southern University of Science and Technology (SUSTech), Shenzhen 518055, China (email: 12113010@mail.sustech.edu.cn).

II. Basic Music Theory and Related Knowledge

A. Simplified Numeric Musical Score

Firstly, let's introduce the concept of simplified numeric musical score in music theory. numeric musical score is a simple method of musical notation, using basic numerals 1, 2, 3, 4, 5, 6, 7 to represent the seven basic degrees of the scale. These numerals correspond to the do, re, mi, fa, sol, la, ti (or si in Chinese) in terms of sound, and in English they are represented by C, D, E, F, G, A, B. Rests are indicated by 0. Figure 1 shows the numeric musical notation of the classic music: 'Castle in the Sky.'

The simplified notation primarily includes numeric notes and some other symbols, such as dots and lines. The meanings of these symbols will be explained in detail in subsequent chapters. These numbers, in combination with the symbols, determine the frequency and duration of each sound. To achieve this, we need to use coding to represent the concepts associated with each numeral and other symbols in the numeric notation. This information will be input into a program to generate a musical signal and then play it.

B. The Tone of Music

Within the simplified numeric musical score, 1, 2, 3, 4, 5, 6, 7 (C, D, E, F, G, A, B) represent seven different pitches, which we refer to as tones. The reason for these different tones is that they correspond to different frequencies of sound waves. Table 1 shows the correspondence between the musical notes in the key of C and their respective frequencies. As indicated in Table 1, the lower frequency of a note is half that of the same note in the mid-range, and similarly, the mid-range frequency is half that of the same note in the higher range. In other words, between the corresponding mid-range and low-range notes, the frequency is halved, resulting in a difference of one

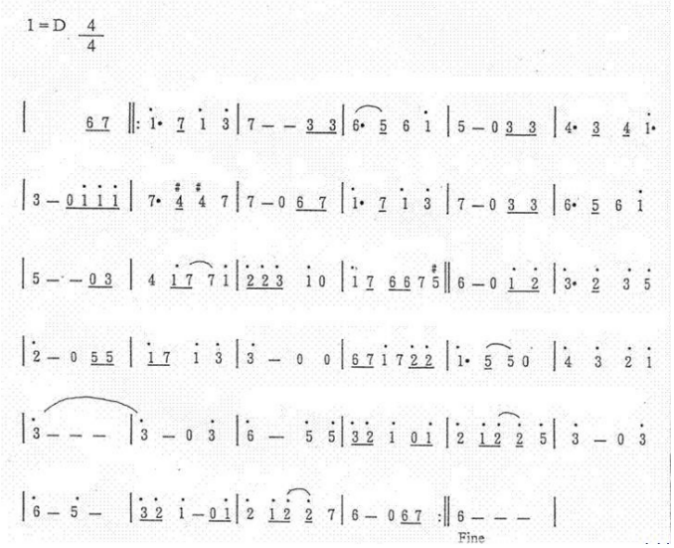

Fig. 1: Numeric Musical Notation of the Music 'Castle in the Sky'

	1(C)	2(D)	3(E)	4(F)	5(G)	6(A)	7 (B)
Low pitch	262	294	330	349	392	440	494
Middle pitch	523	587	659	698	784	880	988
High pitch	1046	1175	1318	1397	1568	1760	1976
Frequency ratio between adjacent musical notes	1. 12	1.12	1.057	1. 12	1. 12	1. 12	1. 058

TABLE I: Corresponding Frequencies (Rounded) for Middle Notes in the Key of C

octave. The small dot placed below the basic musical note in the simplified notation is called a lower point, indicating the lowering of the basic note by one group, essentially decreasing by a pure octave. Conversely, the small dot placed above the basic musical note in the simplified notation is called an upper point, signifying the raising of the basic musical note by one group, essentially increasing by a pure octave.

Additionally, there exists a certain pattern in the frequency ratios between adjacent musical notes. As observed in Table 2, it is evident that the frequency ratios between adjacent musical notes—2:1, 3:2, 5:4, 6:5, and 7:6—yield a frequency ratio of approximately 1.12 ($2^{1/6}$). Meanwhile, the ratios 4:3 and 1i:7 yield around 1.057 ($2^{1/12}$). Therefore, the frequency ratios between adjacent musical notes are not equal. However, we can still calculate the frequency of all notes using the aforementioned proportional relationships.

Moreover, upon a closer observation of the frequency ratio relationships in Table 2, we realize that it's possible to divide a set of notes into 12 equal parts, ensuring that the frequency ratios between adjacent musical notes are equal, precisely $2^{1/12}$. Consequently, after an interval of 12 notes, the frequency doubles. This musical temperament is known as the 'twelve-tone equal temperament' or 'equal temperament,' dividing the universally recognized octave into twelve equal semitones ($2^{1/12}$). Following this rule, we can calculate the frequencies produced by the sharps and flats indicated in the simplified notation. The symbol ' # ' (or 'b') in the simplified notation represents a sharp (or flat), where a sharp indicates an increase of one semitone ($2^{1/12}$) above the original note, and similarly, a flat represents a decrease. Therefore, by including sharps and flats, the seven musical degrees expand to the twelve degrees within the twelve-tone equal temperament. Accordingly, we can calculate the frequencies after applying the sharps or flats using the symbols '#' or 'b' in the simplified notation.

C. The Key Signature of Music

In the beginning of a musical score, we can see symbols like 1=C or 1=D, indicating that the score is notated in the key of C or D. This also specifies the frequency of 1 (the tonic) in the musical score. The keys of C or D are what we typically refer to as key signatures. There is a specific relationship between the frequency of 1 (the tonic) among different key signatures, consistent with what was introduced in Section B. Thus, the key signature determines the frequency of 1 in the musical score. When combined with the principles introduced in Section B, it allows the calculation of the frequencies of all the notes in the entire score.

The primary approach involves initially computing the frequencies of the 7 notes in the key of C. Following this, the frequencies of the tonic notes in other keys (if the musical score is written in a different key) are then calculated. Finally, the frequencies of the remaining notes are computed. We can utilize Matlab to perform these conversions, matching the tone, key signature, the number of octaves up or down, the number of keys up or down, and corresponding symbols in the musical score such as 1=D, a small dot, and the symbols #' or b'. The specific coding methods will be further detailed within the code. The codes converting tone to frequency can be wrote in a Matlab function as follows:

$tone2freq.m$:

```
function freq = tone2frequency(tone, scale, noctave, rising)
% Encoding instructions:
% 'tone' represents the numerical notation in the musical score,
      denoting the pitch of each note, ranging from 1 to 7.
% 'scale' indicates the key signature in the music, input as C,
      D, E, F, G, A, or B.
% 'noctave' signifies the number of octaves higher or lower for
      each pitch, with the numerical range as integers.
% 0 denotes the middle octave, positive values represent 'noctave
      ' higher octaves, and negative values represent 'noctave'
      lower octaves.
% 'rising' indicates whether there's a sharp or flat key
      signature. 1 stands for sharp, -1 for flat, and 0 signifies
      no sharp or flat.
C_frequency = 261.5;
index = [2^(2/12), 2^(2/12), 2^(1/12), 2^(2/12), 2^(2/12),
      2^(2/12), 2^(1/12)];
f = C_frequency;
if scale == "C"
    f_scale = f;
    int_scale = 1;
elseif scale == "D"
    for i = 2:2
        f = f*index( mod(i-2, 7)+1);
    end
    f_scale = f;
    int_scale = 2;
elseif scale == "E"
    for i = 2:3
        f = f*index( mod(i-2, 7)+1);
    end
    f_scale = f;
    int_scale = 3;
```

```
elseif scale == "F"
    for i = 2:4
        f = f*index( mod(i-2, 7)+1);
    end
    f_scale = f;
    int_scale = 4;
elseif scale == "G"
    for i = 2:5
        f = f*index( mod(i-2, 7)+1);
    end
    f_scale = f;
    int_scale = 5;
elseif scale == "A"
    for i = 2:6
        f = f*index( mod(i-2, 7)+1);
    end
    f_scale = f;
    int_scale = 6;
elseif scale == "B"
    for i = 2:7
        f = f*index( mod(i-2, 7)+1);
    end
    f_scale = f;
    int_scale = 7;
end

if tone>=2
    for i = 2:tone
        f_scale = f_scale*index( mod(i+(int_scale-3), 7)+1);
    end
end
if tone == 0
    freq = 0;
else
    freq = f_scale*2^noctave*2^(rising/12);
end
end
```

The programming approach for its code is similar to the analysis above. We initially set the fundamental frequency as the frequency corresponding to the C note in the C major scale. We use the frequency ratios between the notes to calculate the fundamental frequencies for different keys. Then, based on the input key, we determine the fundamental frequency. Subsequently, we calculate the frequencies corresponding to different musical note numbers. Finally, we adjust the output frequency based on the octave, sharp or flat notes, resulting in the final frequency corresponding to each note. We can run this codes and verify that the codes are correct.

D. Duration of Musical Notes

The duration of each musical note is indicated by adding short dashes and dots to the basic note in the simplified numeric musical score.

There are two uses of short dashes: The short dash written to the right of the basic note is called a “dot,” or “augmentation dot.” The more augmentation dots there are, the longer the duration of the note. The basic note without augmentation dots is called a “quarter note,” and each additional augmentation dot extends the time by one quarter note. The short dash written below the basic note is called a “staccato dot.” The more staccato dots there are, the shorter the note. Each additional staccato dot represents a halving of the original note’s duration.

The small dot placed to the right of a note is called a “dotted note,” indicating an extension of half of the previous note’s duration. Dotted notes are often used with quarter notes and various notes shorter than quarter notes. Notes with dots are called ”dotted notes. The transformation above allows us to generate the duration for each note using related code in Matlab. The code is as follows. Here, rhythm is an array representing all the factors related to the duration of each note. The first number in the array represents the basic duration of each note, while the subsequent numbers represent the information regarding dashes or dots accompanying each note. We adopt the following encoding scheme: 1 indicates an augmentation dot, 2 indicates a staccato dot, and 3 indicates a dotted note. This information allows us to calculate the duration for each note based on the symbols provided.

```
duration_basic = rhythm(1);
duration_total = duration_basic;
for i = 2:length(rhythm)
    if rhythm(i)==0
        duration_total = duration_basic;
    end
    if rhythm(i)==1
        duration_total = duration_total + duration_basic;
    end
    if rhythm(i)==2
        duration_total = duration_total *0.5;
    end
    if rhythm(i)==3
        duration_total = duration_total + duration_total *0.5;
    end
end
```

III. Generate Waveforms of Different Frequencies

In Matlab, to play the sound of a musical note, we should first convert it into a sine function, representing the waveform corresponding to that note. Then, we can utilize the ”sound” function to play the sound of that note. Typically, determining the factors of a sine function involves two aspects: the frequency and the duration for which the function continues. The frequency corresponds to the frequency obtained for each note as mentioned earlier, while the duration represents the length of time derived for each note. Setting a sampling rate (here, we set it as 8192) will allow us to obtain a sine waveform function corresponding to a note. We can plot it and use the ”sound” function to play it. Similarly, we can generate a waveform function for each note in the entire musical score (utilizing the frequency and duration for each note). By connecting all the waveforms corresponding to each note together, we can obtain the complete music corresponding to this musical score. Subsequently, we can play this music. Here, we will first demonstrate how to generate waveform data for a single note. We write the following function:

gen_wave.m :

```
function waves = gen_wave(tone, scale, noctave, rising, rhythm,
    fs)
duration_total = rhythm;
freq = tone2frequency(tone, scale, noctave, rising);
t = linspace(0, duration_total, fs* duration_total);
if tone == 0
    waves = zeros(1, fs* duration_total);
else
    waves=sin(2*pi*freq*t);
end
end
```

Using this simple piece of code, we can generate the waveform data for a single note. It is noticeable that that the 'rhythm' here is the time data we generated directly using the code introduced in the previous section for section D, which is directly utilized here. For example, here We can plot the generated waveform of a music notion whose parameters are shown in the codes as follows:

```
y = gen_wave(1,"C",1,0,1, 8192)
sound(y,8192);
t = linspace (0,1,8192) ;
plot(t,y);
xlim ([0,0.02]) ;
title (sprintf('The waveform generated of a music
    note\n12113010, haodong zhang'))
```

And the waveform graph are shown in Figure 2.

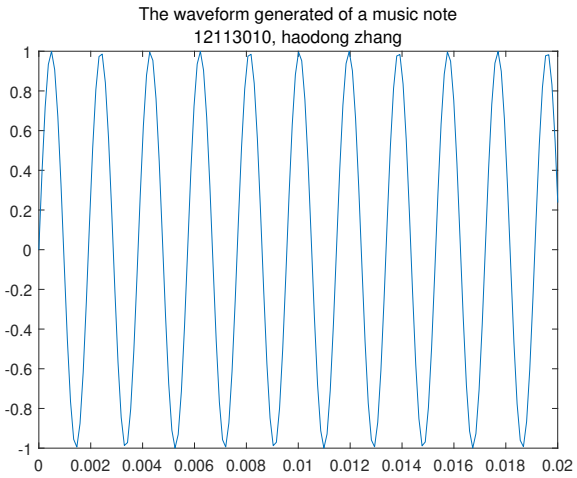

Fig. 2: The Waveform Generated of a Single Music Note

Here we only display the waveform graph from 0 to 0.02 seconds. It's evident that it's indeed a sine function, and the frequency matches the chosen note's frequency. By using the 'sound' function, we can hear a 'du' sound lasting for 1 second. The higher the frequency, the sharper the sound we perceive,

IV. Volume Decay Analysis

The waveform generated above is ideal, with the sine function's amplitude consistently set to 1. However, considering the vibrations during musical instrument performance, the oscillation does not sustain at a fixed amplitude; it decays. Thus, an envelope decay function can more realistically simulate music production. Therefore, we should multiply the generated waveform data above by an envelope decay function.

Different envelope decay functions produce different effects. Here, we have chosen to experiment and analyze three types of decay functions: linear decay, squared decay, and exponential decay. We will plot the corresponding waveform graphs and play them to compare the differences in effects brought about by different decays.

We will continue the analysis on a single note, enriching the 'gen_wave.m' function from the previous section by adding the envelope decay component

The new function considering decay are as follows:

gen_wave.m :

```
function waves = gen_wave(tone, scale, noctave, rising, rhythm,
    fs, attenuation)
duration_total = rhythm;
freq = tone2frequency(tone, scale, noctave, rising);
t = linspace(0, duration_total, fs* duration_total);
if tone == 0
    waves = zeros(1, fs* duration_total);
else
    waves=sin(2*pi*freq*t);
end
if attenuation == "null"
    waves = waves;
elseif attenuation == "line"
    waves = waves.*(1+(0.4-1)/rhythm*t);
elseif attenuation == "exp"
    waves = waves.*exp(-t/1);
elseif attenuation == "squared"
    waves = waves.*(0.6*t.^2-1.2*t+1);
end
end
```

For linear decay: The waveform figure is shown in Figure 3.

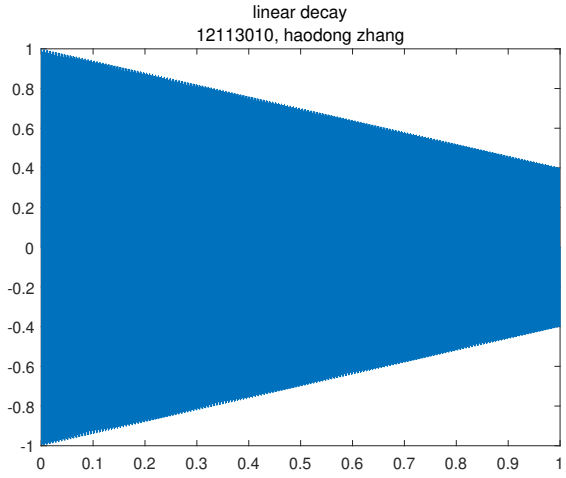

Fig. 3: Linear Decay

Comparing the sound without decay and the sound with linear decay, you can notice that the sound with linear decay gradually reduces in volume as time progresses, and it decreases uniformly. This is due to the linearity of the decay.

For squared decay: The waveform figure is shown in Figure 4.

For squared decay, upon comparison, it can be noted that its decay, in terms of perception, is faster in the initial stages compared to linear decay. However, afterward, it tends to stabilize; the rate of decay gradually decreases, and towards the end, the change in volume becomes minimal.

For exponential decay: The waveform figure is shown in Figure 5.

For exponential decay, it can be observed that its decay rate is much faster compared to the previous two. This aligns more with certain string or piano instruments where a note, once struck, rapidly diminishes. Therefore, exponential decay is more in line with the characteristics of musical instruments compared to the first two decay types. Hence, we adopt exponential decay as the primary function for decay here.

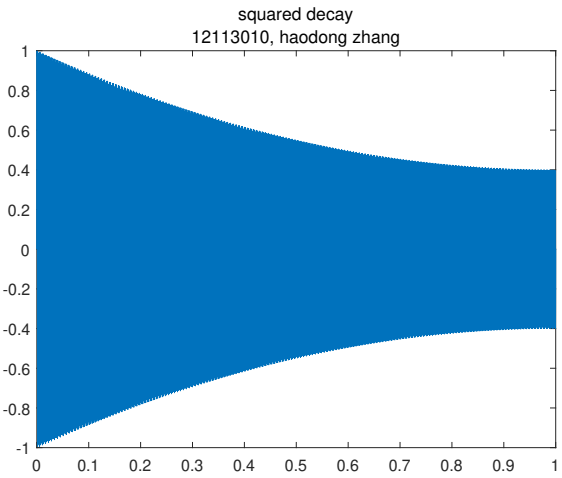

Fig. 4: Squared Decay

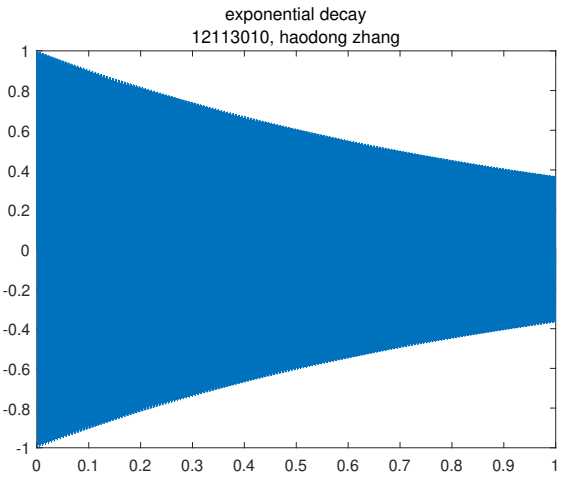

Fig. 5: Exponential Decay

As we further simulate real musical instruments later on, we will also make certain modifications to the exponential decay function to use it as the decay function.

V. Overtones and Harmonic Frequencies Analysis

Actually, in the musical notation discussed earlier, the pitches refer to the fundamental frequencies of the music. However, when playing music with an instrument, apart from producing the fundamental frequency as noted in the music notation, the instrument's sound production principle leads to the creation of varying numbers of standing waves.

Standing waves occur when a string or a part of it vibrates at lengths that are integer multiples of half wavelengths, as the ends are fixed. Therefore, the frequencies produced contain the fundamental frequency and its integer multiples, known as harmonic frequencies. When an instrument is played, it generates harmonic frequencies that are integer multiples of the fundamental frequency, and the primary energy is concentrated at the fundamental frequency. The energy proportions for the 2nd, 3rd, 4th, 5th... harmonics are different for each instrument. Adjusting these proportions leads to entirely different timbres in the sound waves, which is one of the primary reasons different instruments produce distinct tones.

In this context, we continue to use a single note as an example, and once again modify the 'gen_wave.m' function to include more harmonic components. By adjusting different coefficient proportions, we will analyze the impact of the harmonic frequency ratios.

The new function considering decay are as follows:

$gen\_wave.m$:

```
function waves = gen_wave(tone, scale, noctave, rising, rhythm,
        fs, attenuation)
 duration_total = rhythm;
freq = tone2frequency(tone, scale, noctave, rising);
t = linspace(0, duration_total, fs* duration_total);
if tone == 0
    waves = zeros(1, fs* duration_total);
else
    waves=sin(2*pi*freq*t);
    k =  [1,0.1,0.1,0.05,0.05,0.05];
    for i=2:length(k)
        waves=waves+k(i)*sin(2*pi*freq*i*t);
    end
    waves=waves/max(waves);
end
if attenuation == "null"
    waves = waves;
elseif attenuation == "line"
    waves = waves.*(1+(0.4-1)/rhythm*t);
elseif attenuation == "exp"
    waves = waves.*exp(-t/1);
elseif attenuation == "squared"
    waves = waves.*(0.6*t.^2-1.2*t+1);
end
end
```

Using the updated function, let's generate the data for the single note as before and plot the graph. For the harmonic ratios, we set the array as shown in the code. The corresponding graph is depicted in Figure 7. As shown in the figure, setting

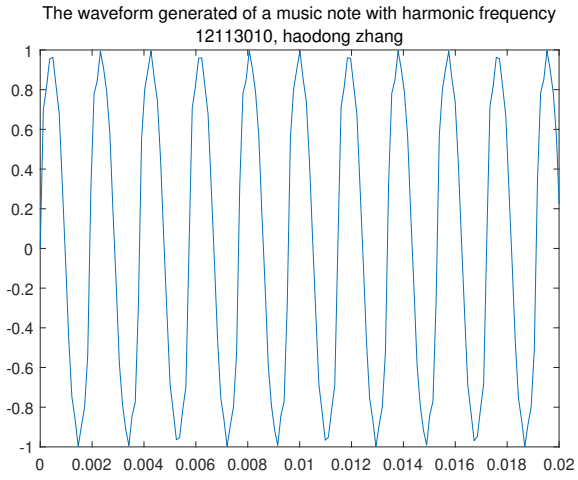

Fig. 6: The Waveform Generated of a Music Note with Harmonic Frequency

the aforementioned harmonic ratios results in the generated waveform not being a perfect sine function. This is because it includes other frequency components that are multiples of the fundamental frequency, leading to a distorted sine wave and a change in the sound.

The variation in timbre for a single note is not easily discernible. Hence, we begin by generating a complete set of waveform data for an entire piece of music. Subsequently, we adjust the harmonic ratios to perceive and analyze the effects of different harmonic ratios on timbre. Therefore, we create the 'gen_music.m' function file to convert the entire musical score into waveform data, play it back, and save the waveform data as a music file (.wav). The code of 'gen_music.m' function are as follows:

gen_music.m :

```
function music = gen_music(toneMatrix, scale, noctaveMatrix,
    risingMatrix, rhythmMatrix, fs, attenuation)
music = [];
for i = 1:length(toneMatrix)
    music = [music, gen_wave(toneMatrix(i), scale, noctaveMatrix(i)
        , risingMatrix(i), rhythmMatrix(i), fs, attenuation)];
end
audiowrite("music.wav",music,fs);
end
```

Next, taking the music 'Castle in the Sky' as an example, we create an array composed of all the numbers representing the musical notes based on the sheet music, called 'toneMatrix.' Subsequently, we assemble the octave and pitch for each note together into 'noctaveMatrix' and 'risingMatrix,' respectively. Then, utilizing the previous code, we write the array form 'rhythmMatrix' for each note's duration. By inputting these parameters into the above-mentioned function, we can generate the waveform data for the entire piece of music and create a music file. Playing this file enables us to listen to the complete 'Castle in the Sky' music. The codes which generate whole music data are as follows:

```
toneMatrix = [6 7 1 7 1 3 7 3 3 6 5 6 1 5 0 3 3 ...
    4 3 4 1 3 0 1 1 1 7 4 4 7 7 0 6 7 1 7 1 ...
    3 7 0 3 3 6 5 6 1 5 0 3 4 1 7 7 1 2 2 3 ...
    1 0 1 7 6 6 7 5 6 0 1 2 3 2 3 5 2 0 5 5 ...
    1 7 1 3 3 0 0 6 7 1 7 2 2 1 5 5 0 4 3 2 ...
    1 3 3 0 3 6 5 5 3 2 1 0 1 2 1 2 2 5 3 0 ...
    3 6 5 3 2 1 0 1 2 1 2 2 7 6 0 6 7 6];
noctaveMatrix = [0 0 1 0 1 1 0 0 0 0 0 0 1 0 0 0 0 ...
    0 0 0 1 0 0 1 1 1 0 0 0 0 0 0 0 1 0 1 ...
    1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 1 1 1 ...
    1 0 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 ...
    1 0 1 1 1 0 0 0 0 1 0 1 1 1 0 0 0 1 1 1 ...
    1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 ...
    1 1 1 1 1 1 0 1 1 1 1 1 0 0 0 0 0 0];
risingMatrix = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
    0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 ...
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
    0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 ...
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0];
rhythmMatrix = 1.5*[1/8 1/8 3/8 1/8 1/4 1/4 3/4 1/8 1/8 3/8 1/8
    1/4 1/4 2/4 1/4 1/8 ...
    1/8 3/8 1/8 1/8 3/8 2/4 1/8 1/8 1/8 1/8 3/8 1/8 1/4 1/4 2/4
    1/4 1/8 1/8 3/8 1/8 ...
    1/4 1/4 2/4 1/4 1/8 1/8 3/8 1/8 1/4 1/4 3/4 1/8 1/8 1/4 1/8
    1/8 1/4 1/4 1/8 1/8 ...
    1/8 1/4 1/4 1/4 1/8 1/8 1/8 1/4 1/4 2/4 1/4 1/8 1/8 3/8 1/8
    1/4 1/4 2/4 1/4 1/8 ...
    1/8 1/8 1/8 1/4 1/4 2/4 1/4 1/4 1/8 1/8 1/4 1/4 1/8 1/8 3/8
    1/8 1/4 1/4 1/4 1/4 ...
    1/4 1/4 4/4 2/4 1/4 1/4 2/4 1/4 1/4 1/8 1/8 1/4 1/8 1/8 1/4
    1/8 1/8 1/8 1/4 2/4 ...
    1/4 1/4 2/4 2/4 1/8 1/8 2/4 1/8 1/8 1/4 1/8 1/8 1/8 1/4 2/4
    1/4 1/8 1/8 4/4];
music = gen_music(toneMatrix,"D",noctaveMatrix, risingMatrix,
    rhythmMatrix,8192,"exp")
sound(music,8192)
```

The whole music waveform are shown in Figure 8. Following this, we will attempt to adjust different harmonic ratios to explore their impact using the whole music. We will explore two primary influences: 1. The impact of the relative weight of harmonic coefficients compared to the fundamental frequency. 2. The effect of the quantity of harmonics on the timbre. For each analysis, we will plot the waveform of a single note for analysis and listen to the entire music to provide an auditory perception analysis.

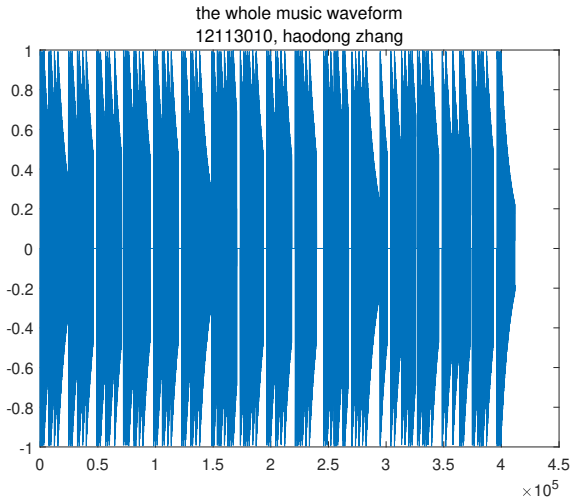

Fig. 7: The Whole Music Waveform

1. The impact of the relative weight of harmonic coefficients compared to the fundamental frequency.

We set two sets of harmonic ratio coefficients, k, as follows: k1 = [1, 0.8, 0.7, 0.6, 0.5] and k2 = [1, 0.1, 0.1, 0.05, 0.05]. We compare the effects generated by these two sets of coefficients. The waveforms are shown in Figure 8 and Figure 9.

From the graphs, we can observe distinct differences. When the harmonics have greater weight, the changes in the waveform are more pronounced. Alternatively, if the weight of the harmonics is smaller, the changes in the waveform are smoother and more gradual. Then, when we listen to the entire piece of music, we can analyze the differences in auditory perception. When the weight of the harmonics is significant, the sound becomes brighter and sharper. This implies that a higher proportion of other harmonics (higher-order harmonics) increases the clarity and brightness of the sound, making it brighter and more distinct. Additionally, the sound becomes more detailed and richer, as more frequency components are included, resulting in a more complex tonality. Conversely, when the weight of the harmonics is smaller, the sound becomes softer and rounder. This indicates that a lower proportion of other harmonics usually leads to a softer, rounder sound, as more energy is concentrated on the fundamental frequency, resulting in a more mellow tone. Moreover, fewer other harmonics may result in a relatively simple sound, lacking in complexity and detail.

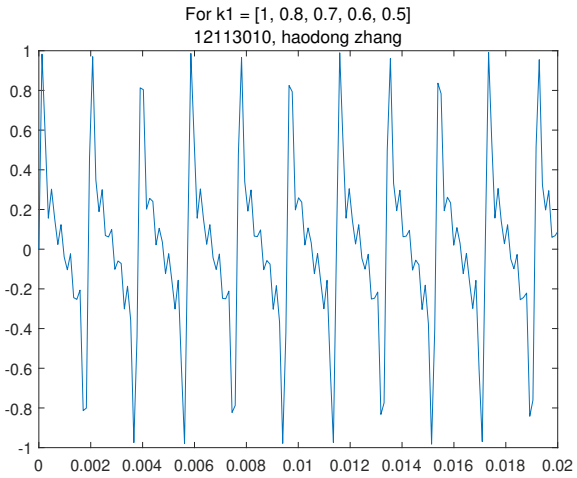

Fig. 8: For k1 = [1, 0.8, 0.7, 0.6, 0.5]

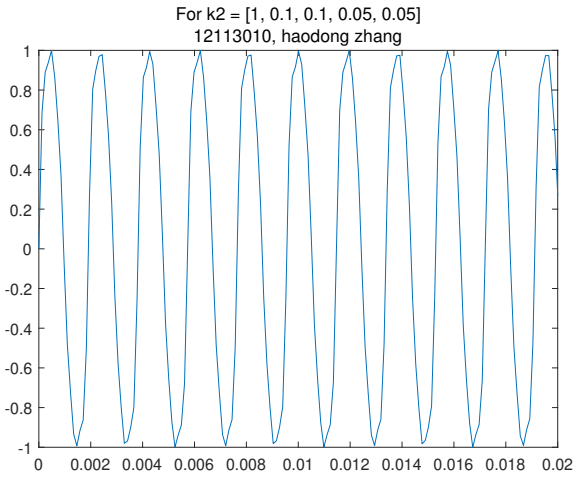

Fig. 9: For k2 = [1, 0.1, 0.1, 0.05, 0.05]

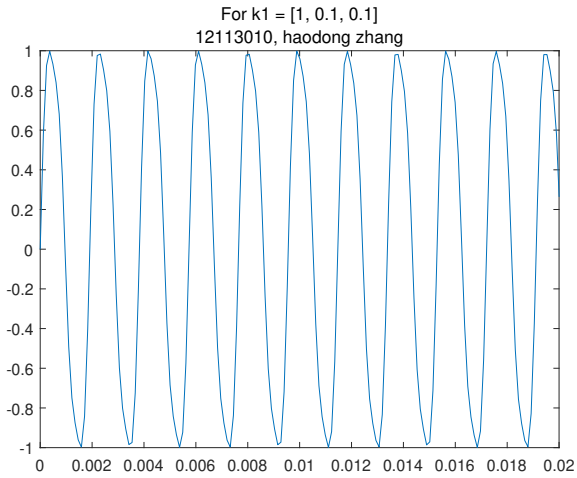

Fig. 10: For k1 = [1, 0.1, 0.1]]

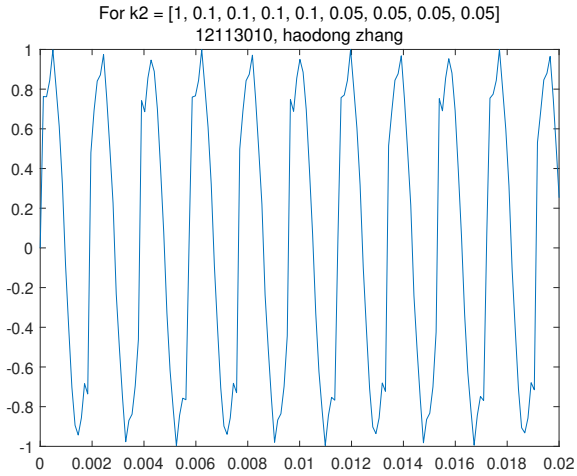

Fig. 11: For k2 = [1, 0.1, 0.1, 0.1, 0.1, 0.05, 0.05, 0.05, 0.05]

2. The effect of the quantity of harmonics on the timbre. We set two sets of harmonic ratio coefficients, k, as follows: k1 = [1, 0.1, 0.1] and k2 = [1, 0.1, 0.1, 0.1, 0.1, 0.05, 0.05, 0.05, 0.05]. We compare the effects generated by these two sets of coefficients. The waveforms are shown in Figure 10 and Figure 11.

From the above graphs, we can observe that with an increase in the number of harmonics, the waveform becomes more complex, and the oscillation becomes more intense. When listening to the entire piece of music, we can analyze the differences in auditory perception. Increasing the number of harmonics enriches the sound and makes it fuller. More harmonics may make the sound clearer, but it might also lead to a perception of sharper or more complex sound. Moreover, we noticed that when there are too many harmonics, the music might produce a certain hoarseness or roughness. This occurs because an excessive number of harmonics could affect the auditory experience. Therefore, the number of harmonics should not be too high. If it's too low, the music may not sound rich. Hence, selecting an appropriate number of harmonics is crucial.

VI. Imitating the Authentic Piano Timbre

Throughout the aforementioned process, we thoroughly discussed the impact of various factors on music. Next, we aim to generate music imitating the authentic timbre of a piano by altering the aforementioned parameters. Our approach involves downloading a single note from a piano, plotting the waveform and spectrogram corresponding to this note, and then using these plots as a reference to set the parameters for our virtual piano sound. This approach will bring our music closer to the sound characteristics of a piano.

We first draw the time-domain waveform of the real piano sound. Based on the characteristics of this waveform, we created an envelope decay function of the form $t^a e^{bt}$. Upon testing, we used a = 0.01 and b = -2.5, resulting in a virtual piano time-domain waveform that closely resembles the real waveform. We incorporated this decay envelope function into our function 'gen_wave.m,' corresponding to the parameter 'exp_pro.' Then, we plot the frequency domain graph of the real piano sound. We adjusted the proportions of harmonics and fundamental frequencies in our own music based on

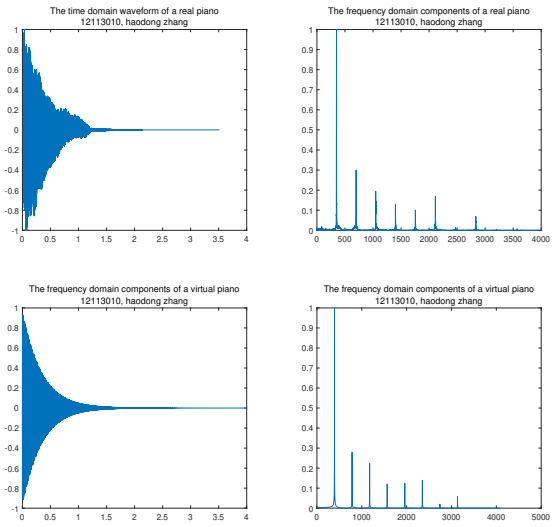

Fig. 12: Imitating the Authentic Piano Timbre

the relationship between these frequencies in the real piano sound. This completes the imitation of the piano timbre in our music. We can compare the time-domain and frequency-domain graphs of the two to assess the imitation effect, as shown in Figure 12. The code is as follows:

```
figure;
sample=audioread("4 F.mp3");
sample=sample(:,1);
sample=sample/max(sample);
samplet=(1:length(sample))/44100;
sound(sample,44100)
clf;subplot(2,2,1);plot(samplet,sample);
title(sprintf("The time domain waveform of a real piano\
    n12113010, haodong zhang"));
ylim([-1,1]);
samplel=length(samplet);
samplen=2^nextpow2(samplel);
sampley=fft(sample,samplen);
samplep=abs(sampley/samplel);
samplep=samplep/max(samplep);
samplep=samplep(1:ceil(samplen/2));
samplep(2:end-1)=samplep(2:end-1);
subplot(2,2,2);plot(0:(44100/samplen):(44100/2-44100/samplen),
    samplep);
title(sprintf("The frequency domain components of a real piano\
    n12113010, haodong zhang"));
xlim([0,4000]);

fs = 8192;
testt =linspace(0, 4, fs*4);
testf =391.803;
wave=sin(2*pi*testf*testt);
k =[1,0.285,0.222,0.12,0.1246,0.141,0.02,0.059];
for i=2:8
    wave=wave+k(i).*sin(2*pi*testf*i*testt);
end
wave=wave/max(wave);
wave=wave.*(testt.^0.01.*exp(-2.5*testt));
subplot(2,2,3);plot(testt,wave);ylim([-1,1]);
title(sprintf("The frequency domain components of a virtual piano
    \n12113010, haodong zhang"));
testl =length(testt);
testn =2^nextpow2(testl);
testy =fft(wave,testn);
testp =abs(testy/testl);
testp =testp/max(testp);
testp =testp(1:ceil(testn/2));
testp(2:end-1)=testp(2:end-1);
subplot(2,2,4);plot(0:(fs/testn):(fs/2-fs/testn),testp);ylim
    ([0,1]);
title(sprintf("The frequency domain components of a virtual piano
    \n12113010, haodong zhang"));
sound(wave,8192);
```

The final whole music waveforms are shown in Figure 13. And the music.wav are also generated. And the final timbre closely matches that of the piano!

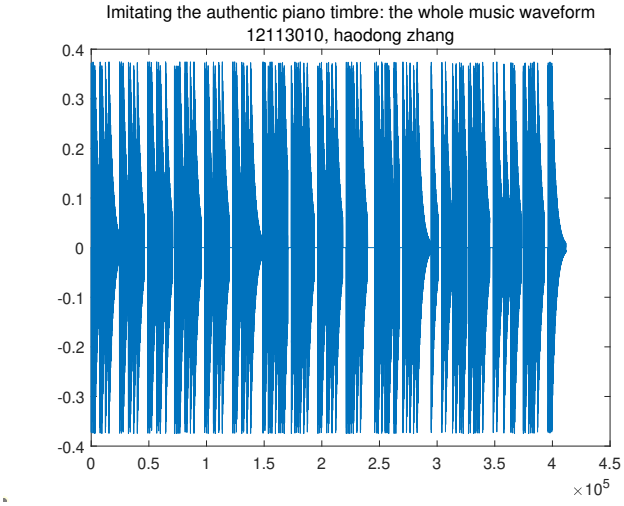

Fig. 13: Imitating the Authentic Piano Timbre: final whole music waveform

Finally, we present the ultimate 'gen_wave.m' function file: *gen_wave.m* :

```
function waves = gen_wave(tone, scale, noctave, rising, rhythm,
    fs, attenuation)
duration_total = rhythm;
freq = tone2frequency(tone, scale, noctave, rising);
t = linspace(0, duration_total, fs*duration_total);
if tone == 0
    waves = zeros(1, fs*duration_total);
else
    waves=sin(2*pi*freq*t);
    k =[1,0.285,0.222,0.12,0.1246,0.141,0.02,0.059];
    for i=2:length(k)
        waves=waves+k(i)*sin(2*pi*freq*i*t);
    end
    waves=waves/max(waves);
end
if attenuation == "null"
    waves = waves;
elseif attenuation == "line"
    waves = waves.*(1+(0.4-1)/rhythm*t);
elseif attenuation == "exp"
    waves = waves.*exp(-t/1);
elseif attenuation == "squared"
    waves = waves.*(0.6*t.^2-1.2*t+1);
elseif attenuation == "exp_pro"
    a = 0.01;
    b = -3;
    waves=0.7*waves.*(t.^a.*exp(b*t));
end
end
```

We also propose an improvement method: by imitating the corresponding harmonic ratio coefficients and decay functions for each note using a similar method as described above.

Instead of setting all the notes with the same coefficients, this approach would be closer to the real timbre of the piano.

VII. Conclusion

Through this experiment, we have acquired fundamental knowledge in music theory, such as how to read numeral sheet music, understanding basic concepts including pitch, key signatures, and note duration. We have also learned how to use Matlab to translate numeral sheet music into a piece of music. Additionally, we investigated the impact of decay functions and harmonic ratios on timbre, conducting thorough analyses and selecting more suitable decay functions. Most significantly, we attempted to impart a real piano timbre to the generated music signal. By setting parameters similar to those of a real piano, we achieved a music signal that closely resembles the timbre of a piano. Hearing piano-like music produced by a computer is truly inspiring! Finally, we reflected on our findings and proposed further enhancement methods.

Acknowledgments

Thanks to the teachers and teaching assistants for their guidance. Thank the teacher for teaching us theoretical knowledge and providing the lab manual for us, which raised many meaningful questions for us and guided and inspired us to think deeply.