Mini Project——计算机生成和播放音乐

12210262 韩璐冰

一、实验目的

本实验旨在利用 MATLAB 编程生成和播放音乐，具体目标包括：

1. 理解音乐的基本要素（如音调、节拍、音色、基频、和弦等），并掌握其在数字化表示中的特点。
2. 通过对简谱的分析与编码，探索音乐要素与数据的对应关系。
3. 编写 MATLAB 程序，将数字简谱转化为音符频率和波形数据，并生成可播放的音乐文件。
4. 模拟乐器的音色特性，通过调整基频与谐波能量比例生成接近实际乐器音色的音乐信号。
5. 综合运用数字信号处理方法，分析不同参数（如包络衰减函数、音色调整）对听感的影响。

二、实验过程与结果分析

1. 音符频率计算

（1）代码

简谱中1、2、3、4、5、6、7（C、D、E、F、G、A、B）表示七种高低不同的音，称之为音调。不同的音调对应不同频率的音波。简谱上出现的数字1 - 7加 上升降调就有了12种音符，再加上上下的小圆点，就有了12 x N（N个八度）种音符。因此，我们用3个变量`tone`、`rising`、`noctave`，分别控制1 - 7的基本音级、升降调、相差多少个八度。一个八度分为12个半音音程，相邻音程的频率比例为$2^{1/12}$，相

邻八度之间的频率比例为2。假设主音为440Hz，编写如下函数：

```
% MATLAB练习1
function freq = tone2freq(tone, noctave, rising)
% tone: 输入数字音符，数值范围1到7
% 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

shift = (tone - 1) * 2 + rising; % 音符的位移
if tone <= 3 && tone > 0
    freq = 440 * 2^noctave * 2^(shift / 12);
elseif tone > 3
    freq = 440 * 2^noctave * 2^((shift - 1) / 12);
elseif tone==0
    freq = 0;
end
end
```

用这个函数，我们能够输出440Hz主音频率下的1 - 7音符，高若干八度，低若干八度，以及升降调的频率。

不同的简谱可能对应不同的调号（即不同的主音频率）。因此，可以在函数中增加一个变量`scale`，用于输入对应的调号，以便根据调号调整音高。

```
% MATLAB练习2
function freq = tone2freq(tone, scale, noctave, rising)
% tone: 输入数字音符，数值范围1到7
% 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
% scale: 调号
% freq为输出的频率

% 不同调号主音频率表
freq_base = struct('C', 261.5, 'D', 293.5, 'E', 329.5, 'F', 349, 'G', 391.5, 'A', 440, 'B', 494);

% 获取基准音频
if isfield(freq_base, scale)
    base_freq = freq_base.(scale);
else
    error('Invalid scale input');
end

% 计算频率
shift = (tone - 1) * 2 + rising; % 音符的位移
if tone <= 3 && tone > 0
    freq = base_freq * 2^noctave * 2^(shift / 12);
elseif tone > 3
    freq = base_freq * 2^noctave * 2^((shift - 1) / 12);
elseif tone==0
```

```
    freq = 0;
end
end
```

至此，我们能够计算出一张简谱中任何一个音符的频率。

（2）结果与分析

输入不同音符进行测试：

```
%test1
f = tone2freq(3, 'C', 1, 0)
f = tone2freq(5, 'C', 0, -1)
f = tone2freq(7, 'A', -1, -1)
f = tone2freq(1, 'D', 0, 1)
f = tone2freq(2, 'E', 1, 1)
```

输出结果为：

```
f = 658.9387
f = 369.8168
f = 391.9954
f = 310.9524
f = 783.6875
```

与音符频率对照表比较，结果输出正确。（附表如下）

C调音符与频率对照表

音符	频率/Hz	音符	频率/Hz	音符	频率/Hz
低音1	262	中音1	523	高音1	1046
低音1#	277	中音1#	554	高音1#	1109
低音2	294	中音2	587	高音2	1175
低音2#	311	中音2#	622	高音2#	1245
低音3	330	中音3	659	高音3	1318
低音4	349	中音4	698	高音4	1397
低音4#	370	中音4#	740	高音4#	1480
低音5	392	中音5	784	高音5	1568
低音5#	415	中音5#	831	高音5#	1661
低音6	440	中音6	880	高音6	1760
低音6#	466	中音6#	932	高音6#	1865
低音7	494	中音7	988	高音7	1976

2. 生成音符波形

（1）代码

将音符转化为对应频率的正弦波，即可用sound函数听到声音，音符持续的时间对应波形的长度。因此，加上输入变量`rhythm`、`fs`，编写输出音符波形的函数：

```
% MATLAB练习3
function waves = gen_wave(tone, scale, noctave, rising, rhythm, fs)
% 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
f = tone2freq(tone, scale, noctave, rising);
t = linspace(0,rhythm,fs*rhythm);
waves = sin(2*pi*f*t);
plot(t,waves)
sound(waves,fs)
end
```

（2）结果与分析

测试输出C调下音符`2#`的波形（查表可知f=311Hz），持续1秒：

```
%test2
wave = gen_wave(3,'C',0,-1,1,8192);
```

输出波形为：

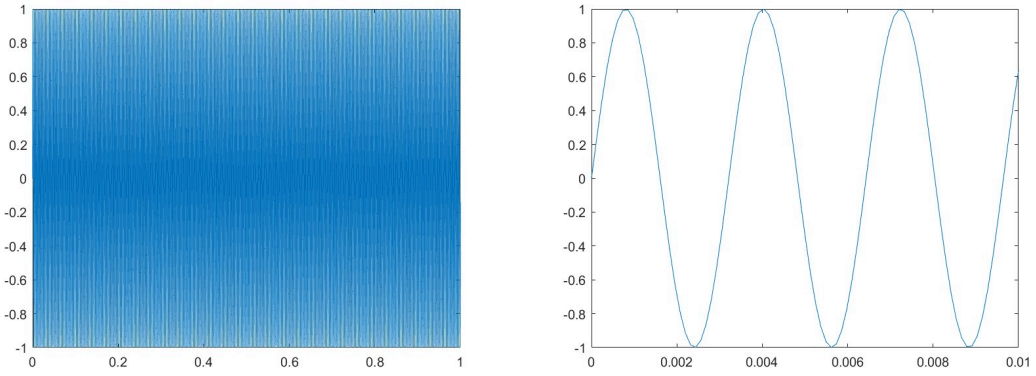
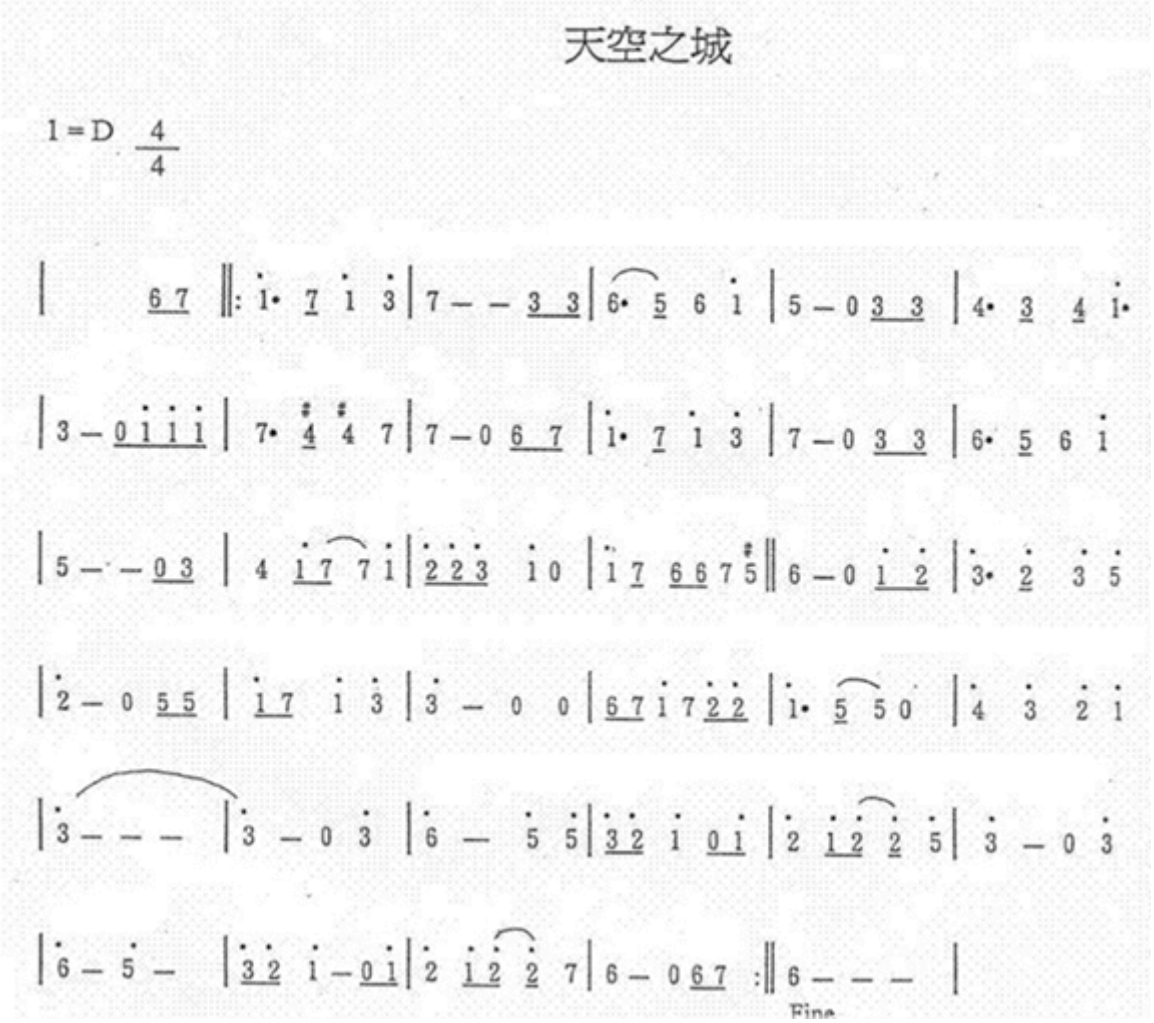


Fig.1 单音符波形图（左：原图，右：局部放大图）

如图，输出频率为311Hz，持续时长1s的正弦波。用sound函数播放声音，效果很好。

3. 数字简谱转化

基于上述基础操作，我们可以正确输出简谱中任意单个音符的波形，并用计算机发出声音。将单个音符相连，即可生成一段完整的音乐。接下来，我们将天空之城的数字简谱（如下图）转换为波形文件，用函数`audiowrite`将其写入`.wav`音乐文件中，并用函数`audioplayer`、`play`播放。



（1）代码

1>编写函数`gen_music`，其输入参数与`gen_wave`一样，但`tone`、`noctave`、`rising`、`rhythm`都是等长度的数组，对应简谱中的每个音符。`for`循环遍历数组，用函数`gen_wave`得到每个音符的波形，最后将所有波形拼接，得到整首音乐的波形。代码实现如下：

```
function wave_music = gen_music(tone, scale, noctave, rising, rhythm, fs)
% 生成乐曲的波形图
t = linspace(0, sum(rhythm), fs*sum(rhythm));
l = length(tone);
wave_segments = cell(1,l);
for m=1:l
    wave_segments{m} =
gen_wave(tone(m),scale,noctave(m),rising(m),rhythm(m),fs);
end
wave_music = cell2mat(wave_segments);
figure
plot(t,wave_music)
xlabel('t/s');
ylabel('Amplitude')
title('music wave')
end
```

2>对应简谱，分别写出`tone`、`scale`、`noctave`、`rising`、`rhythm`、`fs`作为函数`gen_nusic`的输入参数。可知该曲谱为D调，四分音符为一拍，每小节四拍。符号||:和:||中间的部分重复一次，共267个音符。每拍的时间设为`base_time`，根据简谱中的增时线和减时线判断是几拍。采样率设为44100Hz。代码实现如下：

```
% 简谱数据
tone = [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 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];
scale = 'D';
noctave = [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
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 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 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 1 0 1 1 1 1
0];
rising = [zeros(1,27), 1, 1, zeros(1,35), 1, zeros(1,69), zeros(1,25), 1, 1,
zeros(1,35), 1, zeros(1,70)];
base_time = 0.5;
rhythm = base_time * [0.5 0.5 1.5 0.5 1 1 3 0.5 0.5 1.5 0.5 1 1 2 1 0.5 0.5
1.5 0.5 0.5 1.5 2 0.5 0.5 0.5 0.5 1.5 0.5 1 1 2 1 0.5 0.5 1.5 0.5 1 1 2 1 0.5
0.5 1.5 0.5 1 1 3 0.5 0.5 1 0.5 0.5 1 1 0.5 0.5 0.5 1 1 1 0.5 0.5 0.5 1 1 2 1
0.5 0.5 1.5 0.5 1 1 2 1 0.5 0.5 0.5 0.5 1 1 2 1 1 0.5 0.5 1 1 0.5 0.5 1.5 0.5
1 1 1 1 1 1 4 2 1 1 2 1 1 0.5 0.5 1 0.5 0.5 1 0.5 0.5 0.5 1 2 1 1 2 2 0.5 0.5
2 0.5 0.5 1 0.5 0.5 0.5 1 2 1 0.5 0.5 1.5 0.5 1 1 3 0.5 0.5 1.5 0.5 1 1 2 1
0.5 0.5 1.5 0.5 0.5 1.5 2 0.5 0.5 0.5 0.5 1.5 0.5 1 1 2 1 0.5 0.5 1.5 0.5 1 1
2 1 0.5 0.5 1.5 0.5 1 1 3 0.5 0.5 1 0.5 0.5 1 1 0.5 0.5 0.5 1 1 1 0.5 0.5 0.5
1 1 2 1 0.5 0.5 1.5 0.5 1 1 2 1 0.5 0.5 0.5 0.5 1 1 2 1 1 0.5 0.5 1 1 0.5 0.5
1.5 0.5 1 1 1 1 1 1 4 2 1 1 2 1 1 0.5 0.5 1 0.5 0.5 1 0.5 0.5 0.5 1 2 1 1 2 2
0.5 0.5 2 0.5 0.5 1 0.5 0.5 0.5 1 2 1 0.5 0.5 4];
fs=44100;
```

3>得到音乐波形后，写入`music.wav`文件，并用`play`播放。代码实现如下：

```
% 生成音乐波形并播放
wave_music = gen_music(tone,scale,noctave,rising,rhythm,fs);
audiowrite('music.wav', wave_music, fs);
player = audioplayer(wave_music, fs);
play(player)
```

（2）结果与分析

由数字简谱输出的波形图如下：

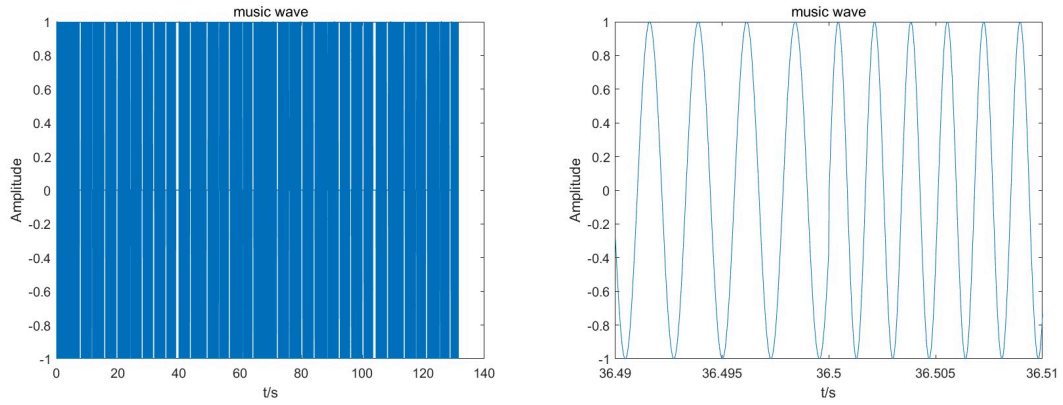


Fig.2 整曲音乐波形图（左：原图，右：局部放大图）

该波形图为不同频率的正弦波拼接，总时长为*2′11″*。`play`播放可听到正确的旋律，输出结果正确。

4. 音量波动模拟

真实情况下，乐器演奏时振动会有衰减，不会以固定幅度持续振动，因此一个包络衰减函数能够更加真实的模拟音乐的产生。

（1）包络衰减函数

用一个单音符进行接下来的包络衰减测试：

```
% MATLAB练习4
T=1;
wave = gen_wave(1,'A',0,0,T,fs);
t=linspace(0,T,T*fs);
```

- **指数衰减**

$$y(t) = A_0 \cdot e^{-at}$$

初期衰减快，后期逐渐变慢，振幅渐近于零。常见于物理系统中能量快速损失的场景，如阻尼振动、放射性衰变、电子电路的电容放电等。

```
wave1=wave.*exp(-t/T);
sound(wave1,fs)
```

- **线性衰减**

$$y(t) = A_0 \cdot (1 - \beta t), 0 \leq t \leq \frac{1}{\beta}$$

振幅随时间线性减小，直到完全衰减为零。多用于人工设计的系统或信号衰减，比如调制信号的渐弱部分。

```
wave2=wave.*(1-t/(1.5*T));
sound(wave2,fs)
```

- **平方衰减**

$$y(t) = \frac{A_0}{(1+\beta t)^2}$$

初期衰减较快，随着时间增长，振幅减弱速度逐渐变缓，逐渐接近零。常用于光波、声波或电磁波的强度随距离减弱的场景。

```
wave3=wave./((1+t/T).^2);
sound(wave3,fs)
```

- **高斯衰减**

$$y(t) = A_0 \cdot e^{-\frac{t^2}{2\sigma^2}}$$

振幅随时间以高斯分布方式减少。通常用在物理模拟中，效果比指数衰减更加平滑。

```
wave4=wave.*exp(-(t/T).^2);
sound(wave4,fs)
```

- **双曲线衰减**

$$y(t) = \frac{A_0}{1+\beta t}$$

振幅初期衰减较慢，后期变得更快，逐渐趋于零。常用于描述某些扩散过程、热传导过程，或者受外部干扰较小的缓慢衰减系统。

```
wave5=wave./(1+t/T*2);
sound(wave5,fs)
```

(2) 比较分析

绘制五种包络衰减后的波形图：

```
figure
subplot(321)
plot(t,wave1)
title('指数衰减')
subplot(322)
plot(t,wave2)
title('线性衰减')
subplot(323)
plot(t,wave3)
title('平方衰减')
subplot(324)
plot(t,wave4)
title('高斯衰减')
subplot(325)
plot(t,wave5)
title('双曲线衰减')
```

输出图像如下：

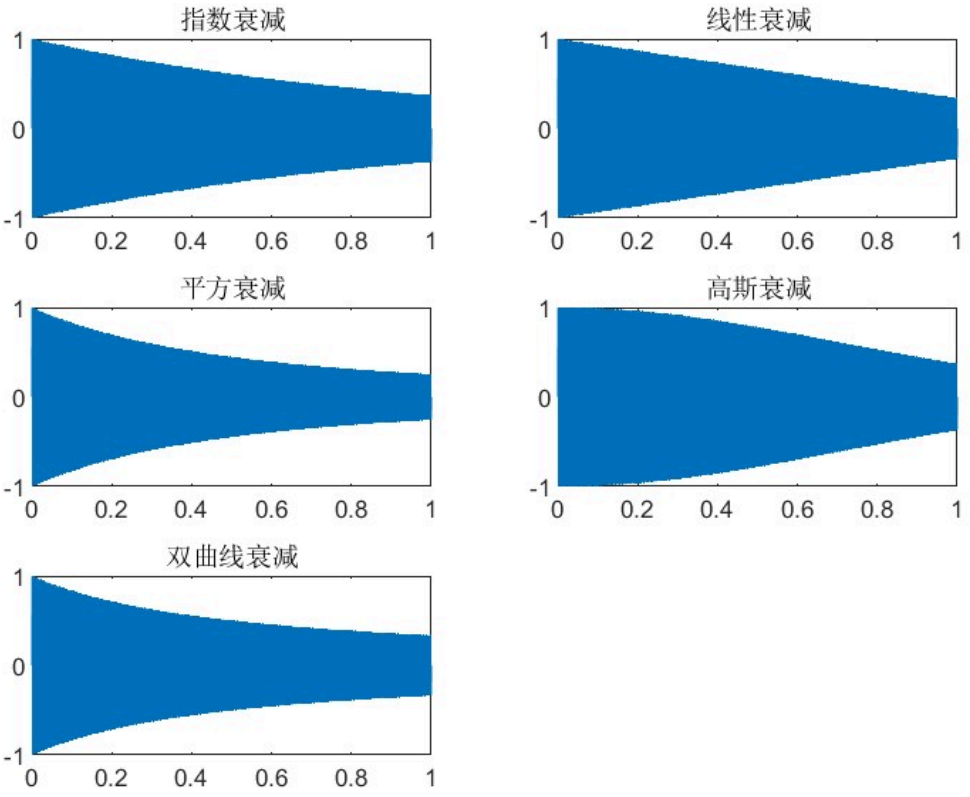


Fig.3 包络衰减波形图

由图可知，指数衰减、平方衰减、双曲线衰减都是先快速衰减后缓慢降低，线性衰减与时间呈线性关系，高斯衰减则是先缓慢下降，中间快速衰减，最后再缓慢降低。

实际情况下，对于钢琴、吉他等乐器，当一个音符被弹奏后，振动的能量会随着时间逐渐减少，且衰减速率与能量损失成正比，表现为短促明亮的音头和悠长的尾音（尤其在钢琴或钟声中明显）。这种衰减反映了大多数物理振动系统（如琴弦、鼓膜等）的能量耗散过程——起初，振动的能量密度大，摩擦、空气阻力的作用更显著，单位时间的损耗也更多；振幅较小时，系统的非线性效应（如材料的弹性阻尼）变弱，能量耗散速率降低。这种规律是振动系统和能量耗散共同作用的结果。

因此，真实声音的衰减显然是非线性的，大多数情况下更符合指数衰减模型。在听感上，指数衰减、平方衰减和双曲线衰减没有明显差别，但是平方衰减和双曲线衰减在后期衰减过于缓慢，会导致尾音无限拖长，不符合实际情况。因此指数衰减的播放效果最好，最符合实际。

不过，某些乐器（如小提琴、二胡）的音符如果由弓拉奏出，并自然停止拉奏后，可能更符合高斯衰减。这是因为这些乐器的振动在开始和结束时往往会有更平滑的过渡，而不是一开始就迅速衰减。

最后，我们选择**指数衰减**模型。

5. 不同音色调整

乐谱中的音调都是指音乐的基频，而用乐器演奏音乐时，除了发出乐谱中的基频声音外， 由于乐器的发声原理，还产生数量不等的驻波。所以乐器弹奏时会产生包括基频和若干整数倍频率的谐频，而主要的能量集中于基频。对于2倍、3倍、 4 倍、5倍……的谐频，不同乐器这些谐频的能量比例各不相同。如果我们调整这个比例，将产生音色完全不同的声波。接下来我们调整这个比例，获得不同的音色。

（1）谐波能量比例调整

用一个单音符进行音色测试：

```
% MATLAB练习5
f= tone2freq(1,'A',0,0);
T = 1;
t=linspace(0,T,T*fs);
w = linspace(-fs/2,fs/2,length(t));
```

- **1：0**

只有基频，无驻波：

```
y0 = sin(2*pi*f*t);
y0_w = fftshift(fft(y,length(y0)));
```

- **0.8 : 0.1 : 0.05 : 0.05**

基频主导，谐波比例逐渐降低：

```
y1 =
0.8*sin(2*pi*f*t)+0.1*sin(2*pi*2*f*t)+0.05*sin(2*pi*3*f*t)+0.05*sin(2*pi*4*f*
t);
y1_w = fftshift(fft(y1,length(y1)));
```

- **0.55 : 0.2 : 0.12 : 0.06 : 0.03 : 0.025 : 0.015**

基频比例调低，谐波成分增加，且比例逐渐降低：

```
y2 =
0.55*sin(2*pi*f*t)+0.2*sin(2*pi*2*f*t)+0.12*sin(2*pi*3*f*t)+0.06*sin(2*pi*4*f
*t)+0.03*sin(2*pi*5*f*t)+0.025*sin(2*pi*6*f*t)+0.015*sin(2*pi*7*f*t);
y2_w = fftshift(fft(y2,length(y2)));
```

- **0.55 : 0.05 : 0.15 : 0.025 : 0.1 : 0.025 : 0.1**

基频比例调低，谐波成分增加，且奇次谐波比例更高：

```
y3 =
0.55*sin(2*pi*f*t)+0.05*sin(2*pi*2*f*t)+0.15*sin(2*pi*3*f*t)+0.025*sin(2*pi*4
*f*t)+0.1*sin(2*pi*5*f*t)+0.025*sin(2*pi*6*f*t)+0.1*sin(2*pi*7*f*t);
y3_w = fftshift(fft(y3,length(y3)));
```

- **0.8 : 0.025 : 0.025 : 0.05 : 0.1**

基频主导，谐波比例逐渐升高：

```
y4 =
0.8*sin(2*pi*f*t)+0.025*sin(2*pi*2*f*t)+0.025*sin(2*pi*3*f*t)+0.05*sin(2*pi*4
*f*t)+0.1*sin(2*pi*5*f*t)
y4_w = fftshift(fft(y4,length(y4)));
```

（2）比较分析

画出五种音色波形的时域图和频谱图：

```
y = {y0 y1 y2 y3 y4};
y_w = {y0_w y1_w y2_w y3_w y4_w};
figure
for n=1:5
    subplot(5,2,2*n-1)
    plot(t,y{n},'LineWidth', 1.25);
    xlim([0,0.01])
    xlabel('t/s');
    ylabel('Amplitude')
    title("音色"+num2str(n-1)+"--时域图")
    subplot(5,2,2*n)
    plot(w,abs(y_w{n}),'LineWidth', 1.25)
    xlim([-5000,5000])
    xlabel('f/Hz');
    ylabel('Magnitude')
    title("音色"+num2str(n-1)+"--频谱图")
end
```

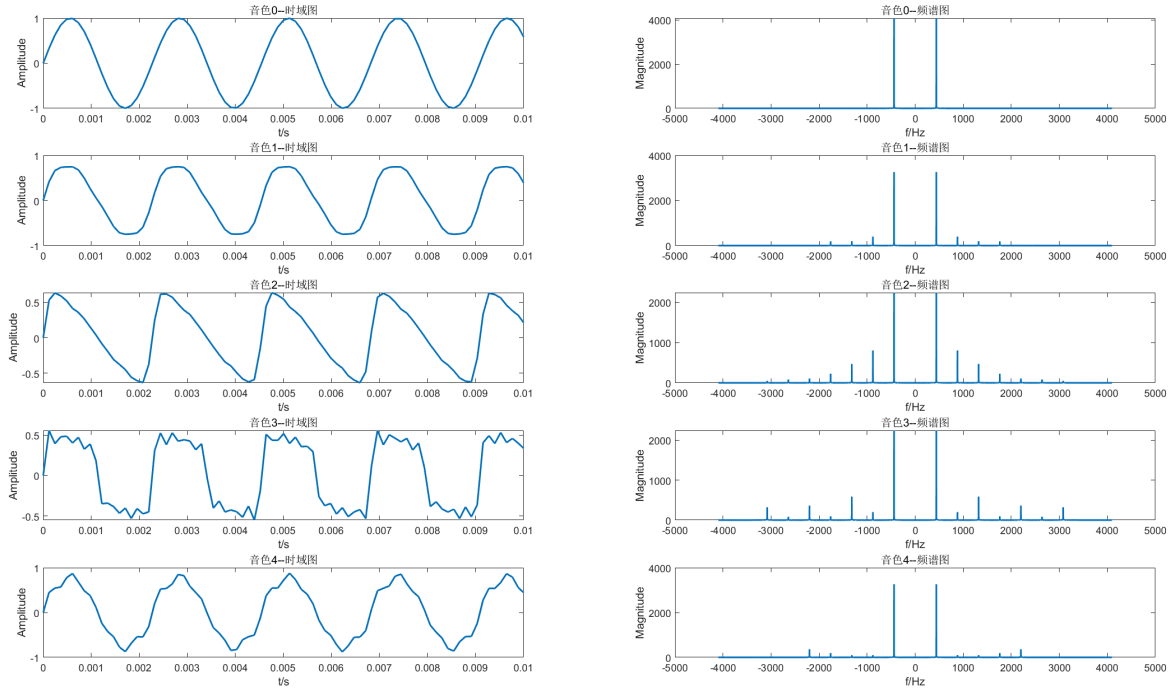


Fig.4 不同音色波形图（左：时域图，右：频谱图）

音色0仅包含基频成分，没有任何谐波或噪声干扰，声音清晰、单一，缺乏丰富的层次感。

音色1以基础音高为主，谐波能量衰减快。有较强的低频成分，但中高频的贡献较少，音色相对较为浑厚和低沉，类似于**钢琴**。

音色2基频能量较强，保持了音高的清晰度，但相较于钢琴，基频占比略低，第二和第三谐波能量相对较高，后续谐波的能量逐渐减小，显示出有较为均衡的中频成分。这种比例代表的是一种较为丰富、层次感较强的音色，声音类似于**小提琴**。音色既有低频的深度，又有一定的中频亮度，表现出一种既温暖又明亮的感觉。

音色3的第二谐波的能量较低，第三谐波较高，第四谐波和第五谐波的能量又逐渐减小，显示出奇次和偶次谐波之间的不平衡，奇次谐波更为突出。这种声音悠扬婉转，类似于**单簧管、长笛**等管乐器。

音色4的第二谐波的能量较低，但随后谐波能量逐渐增强。这表明该乐器的音色高频部分更加突出，音色逐渐转向明亮，但也保留了一定的低频厚度。高频能量的增强使得音色在听感上更加穿透和清晰，类似于**竖琴或八音盒**。

播放比较后，选择声音更明亮的音色4。

6. 音乐文件生成

（1）天空之城

在讨论了音量的衰减和音色后，对原先的音乐文件生成进行优化。首先我们加上两个参数，kn代表不同的谐波比例，即音色选择；p代表选择包络衰减作用于单音符还是双音符（即简谱中有一些音符上方用括弧相连，代表这两个音符应该连续的弹奏，因此在进行衰减操作时，我们将衰减函数作用于这两个连续的音符上，避免中间突兀的音高，其他情况则只作用于单音符）。参数声明如下：

```
k0 = 1;
k1 = [0.8 0.1 0.05 0.05];
k2 = [0.55 0.2 0.12 0.06 0.03 0.025 0.015];
k3 = [0.55 0.05 0.15 0.025 0.1 0.025 0.1];
k4 = [0.8 0.025 0.025 0.05 0.1];
p = [zeros(1,10), 1, zeros(1,41), 1, zeros(1,39), 1, zeros(1,6), 1,
zeros(1,13), 1, zeros(1,14), 1, zeros(1,13), 1, zeros(1,41), 1, zeros(1,39),
1, zeros(1,6), 1, zeros(1,13), 1, zeros(1,14), 1, zeros(1,6)];
```

更新单音符波形生成函数，即在基频基础上加上驻波，k调整比例：

```
function waves = gen_wave2(tone, scale, noctave, rising, rhythm, fs, k)
% 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，k为谐波能量比例，用于调整音色。
f = tone2freq(tone, scale, noctave, rising);
t = linspace(0,rhythm,fs*rhythm);
waves = 0;
for n=1:length(k)
    waves = waves + k(n)*sin(2*pi*n*f*t);
end
waves = waves.*exp(-t/rhythm);
%plot(t,waves)
%sound(waves,fs)
end
```

更新音乐波形生成函数，生成单音符波形后还要做衰减操作（单/多音符衰减），可选择不同音色：

```
function wave_music = gen_music2(tone, scale, noctave, rising, rhythm, fs,
k, p)
t = linspace(0, sum(rhythm), fs*sum(rhythm));
l = length(tone);
wave_segments = cell(1,l);
% 对每个音符/多个音符做包络衰减
for m=1:l-1
    wave_segments{m} =
gen_wave2(tone(m),scale,noctave(m),rising(m),rhythm(m),fs, k);
    if p(m) == 0 && p(m+1) == 0
```

```
        tp = linspace(0, rhythm(m), fs*rhythm(m));
        wave_segments{m} = wave_segments{m}.*exp(-tp/rhythm(m)) ;
    elseif p(m) == 1 && p(m+1) == 0
        tp = linspace(0, rhythm(m)+rhythm(m-1), fs*(rhythm(m)+rhythm(m-1)));
        wave_segments{m} = [wave_segments{m-1},wave_segments{m}].*exp(-
tp/(rhythm(m)+rhythm(m-1))) ;
        wave_segments{m-1} = [];
    end
end
wave_segments{l} =
gen_wave2(tone(l),scale,noctave(l),rising(l),rhythm(l),fs, k);
tp = linspace(0, rhythm(l), fs*rhythm(l));
wave_segments{l} = wave_segments{l}.*exp(-tp/rhythm(l)) ;
% 音符拼接并画图
wave_music = cell2mat(wave_segments);
figure
plot(t,wave_music)
xlabel('t/s');
ylabel('Amplitude')
title('music wave(improved)')
end
```

生成改进的音乐波形文件，画出波形图并播放音乐：

```
wave_music_new = gen_music2(tone,scale,noctave,rising,rhythm,fs,k4,p);
audiowrite('music_new.wav', wave_music_new, fs);
player_new = audioplayer(wave_music_new, fs);
play(player_new)
```

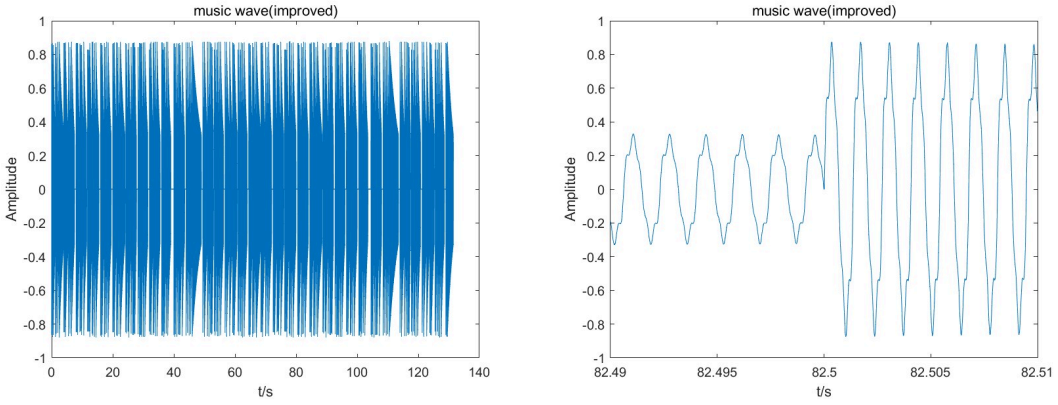


Fig.5 改进后的音乐波形图-天空之城（左：原图，右：局部放大图）

该音乐模仿竖琴的声音，音色明亮清脆、悦耳动听，音符连贯自然，不单调死板，较改善之前音效大大提升。

（2）千与千寻

挑选另一首我喜欢的歌《千与千寻》，他的数字简谱如下：

用相同的方法转换数字简谱，仍选择音色4。不同的是，这个简谱包含**双声道**，因此要分别生成两个通道的音乐波形。具体代码如下：

```
tone1 = [0 0 1 2 3 1 5 3 2 5 2 1 6 3 1 7 7 6 7 1 2 5 1 2 3 4 4 3 2 1 2 1 2 3
1 5 3 2 5 2 1 6 6 7 1 5 0 5 6 7 1 2 5 1 2 3 4 4 3 2 1 1 0 0 3 4 5 5 5 5 5 6
5 4 3 3 3 3 3 4 3 2 1 1 1 7 6 7 7 1 2 2 3 2 3 2 3 4 5 5 5 5 5 6 5 4 3 3 3 3
4 3 2 1 7 6 6 7 1 2 5 1 2 3 2 2 2 1 1 0 0 0 0 0 1 2 3 1 5 3 2 5 2 1 6 3 1 7
7 6 7 1 2 5 1 2 3 4 4 3 2 1 2 1 2 3 1 5 3 2 5 2 1 6 6 7 1 5 0 5 6 7 1 2 5 1
7 6 7 1 2 5 1 7 6 7 1 2 5 1 2 3 4 4 3 2 1 1 1 1 0 0 3 4 5 5 5 5 5 6 5 4 3 3
3 3 3 4 3 2 1 1 1 7 6 7 7 1 2 2 3 2 3 2 3 4 5 5 5 5 5 6 5 4 3 3 3 3 4 3 2 1
7 6 6 7 1 2 5 1 2 3 2 2 2 1 1 1 1 0 0 0];
tone2 = [0 0 0 5 1 7 5 6 6 7 1 7 5 7 3 4 4 3 2 2 5 0 5 1 7 5 6 6 3 3 4 3 2 3
1 1 2 3 0 0 0 1 1 1 2 1 7 6 6 6 1 5 5 3 6 6 6 3 4 4 6 5 6 5 6 5 0 1 1 1 2 1
7 6 6 6 1 5 5 3 1 1 4 3 3 2 5 5 3 0 0 0 0 0 0 5 1 7 5 6 6 7 1 7 5 7 3 4 4 3
0 2 2 5 0 5 1 7 5 6 6 3 3 4 3 2 3 1 1 1 4 3 5 4 4 3 5 2 2 5 5 5 0 0 0 1 1 1
2 1 7 6 6 6 1 5 5 3 6 6 6 3 4 4 6 5 6 5 6 5 0 1 1 1 2 1 7 6 6 6 1 5 5 3 4 4
4 3 3 2 5 5 3 3 3 5 1 3 5 1];
scale = 'F';
noctave1 = [zeros(1,12), -1, 0, 0, -1, -1, -1, -1, 0, 0, -1, zeros(1,19),
-1, -1, -1, 0, -1, 0, -1, -1, -1, 0, 0, -1, zeros(1,32), -1, -1, -1, -1,
zeros(1,25), -1, -1, -1, -1, 0, 0, -1, zeros(1,8), zeros(1,15), -1, 0, 0,
-1, -1, -1, -1, 0, 0, -1, zeros(1,19), -1, -1, -1, 0, -1, 0, -1, -1, -1, 0,
0, -1, 0, -1, -1, -1, 0, 0, -1, 0, -1, -1, -1, 0, 0, -1, zeros(1,34), -1,
-1, -1, -1, zeros(1,25), -1, -1, -1, -1, 0, 0, -1, zeros(1,13)];
noctave2 = [0, 0, 0, -1, 0, -1*ones(1,5), 0, -1*ones(1,10), 0, -1, 0,
-1*ones(1,14), zeros(1,8), -1, -1, -1, -1, 0, -1*ones(1,15), zeros(1,6), -1,
-1, -1, -1, 0, -1*ones(1,12), zeros(1,6), -1, 0, -1*ones(1,5), 0,
-1*ones(1,7), 0, -1*ones(1,3), 0, -1, 0, -1*ones(1,25), zeros(1,8),
-1*ones(1,4), 0, -1*ones(1,15), zeros(1,6), -1*ones(1,4), 0, -1*ones(1,15),
zeros(1,4)];
rising1 = zeros(1,282);
rising2 = zeros(1,200);
```

```
base_time = 0.5;
rhythm1 = base_time .* [1 1 0.5 0.5 0.5 0.5 1.5 0.5 1 1 1 0.5 0.5 1.5 0.5 2
1 1 1 0.5 0.5 1 1 0.5 0.5 1 0.5 0.5 0.5 0.5 2 0.5 0.5 0.5 0.5 1.5 0.5 1 1 1
0.5 0.5 1 0.5 0.5 2 0.5 0.5 1 1 0.5 0.5 1 1 0.5 0.5 1 0.5 0.5 0.5 0.5 3 1 1
0.5 0.5 1 1 1 1 0.5 0.5 0.5 0.5 1 1 1 1 0.5 0.5 0.5 0.5 1 1 0.5 0.5 1 1 0.5
0.5 1 0.5 0.5 0.5 0.5 2 0.5 0.5 1 1 1 1 0.5 0.5 0.5 0.5 1 1 1 0.5 0.5 0.5
0.5 0.5 0.5 1 0.5 0.5 0.5 0.5 1 1 0.5 0.5 1.5 0.5 0.5 0.5 3 1 1 1 1 1 0.5
0.5 0.5 0.5 1.5 0.5 1 1 1 0.5 0.5 1.5 0.5 2 1 1 1 0.5 0.5 1 1 0.5 0.5 1 0.5
0.5 0.5 0.5 2 0.5 0.5 0.5 0.5 1.5 0.5 1 1 1 0.5 0.5 1 0.5 0.5 2 0.5 0.5 1 1
0.5 0.5 1 1.5 0.5 1 1 0.5 0.5 2 0.5 0.5 1 1 0.5 0.5 1 1 0.5 0.5 1 0.5 0.5
0.5 0.5 3 3 3 1 1 0.5 0.5 1 1 1 1 0.5 0.5 0.5 0.5 1 1 1 1 0.5 0.5 0.5 0.5 1
1 0.5 0.5 1 1 0.5 0.5 1 0.5 0.5 0.5 0.5 2 0.5 0.5 1 1 1 1 0.5 0.5 0.5 0.5 1
1 1 0.5 0.5 0.5 0.5 0.5 0.5 1 0.5 0.5 0.5 0.5 1 1 0.5 0.5 1.5 0.5 0.5 0.5 3
3 3 1 1 1];
rhythm2 = base_time .* [1 1 1 2 1 2 1 2 1 0.5 0.5 0.5 0.5 0.5 0.5 2 1 3 2 1
2 1 2 1 2 1 2 1 0.5 0.5 0.5 0.5 0.5 0.5 3 3 3 3 1 1 1 1 1 1 1.5 0.5 1 1 1
0.5 0.5 2 0.5 0.5 1 1 0.5 0.5 2 1 1 0.5 0.5 0.5 0.5 2 1 1 1 1 1.5 0.5 1 1 1
0.5 0.5 2 0.5 0.5 1 1 1 2 1 1.5 0.5 1 3 1 1 1 1 1 1 2 1 2 1 2 1 0.5 0.5 0.5
0.5 0.5 0.5 2 1 2 1 2 1 2 1 2 1 2 1 2 1 0.5 0.5 0.5 0.5 0.5 0.5 3 3 2 1 2 1
2 1 2 1 2 1 3 3 3 1 1 1 1 1 1 1.5 0.5 1 1 1 0.5 0.5 2 0.5 0.5 1 1 0.5 0.5 2
1 1 0.5 0.5 0.5 0.5 2 1 1 1 1 1.5 0.5 1 1 1 0.5 0.5 2 0.5 0.5 1 1 1 2 1 1.5
0.5 1 3 3 1 1 1 1 1 1];
fs=44100;
p1 = [zeros(1,208), 1, 1, zeros(1,67), 1, 1, 0, 0, 0];
p2 = [zeros(1,140), 1, 1, zeros(1,51), 1, zeros(1,6)];
% 分别生成双声道音乐波形，并组合
wave_music_part1 = gen_music2(tone1,scale,noctave1,rising1,rhythm1,fs,k4,p);
wave_music_part2 = gen_music2(tone2,scale,noctave2,rising2,rhythm2,fs,k4,p);
wave_music_favor = [ wave_music_part1' , wave_music_part2'];
audiowrite('music_favor.wav', wave_music_favor, fs);
player_favor = audioplayer(wave_music_favor, fs);
play(player_favor);
```

双声道输出的音乐波形如下：

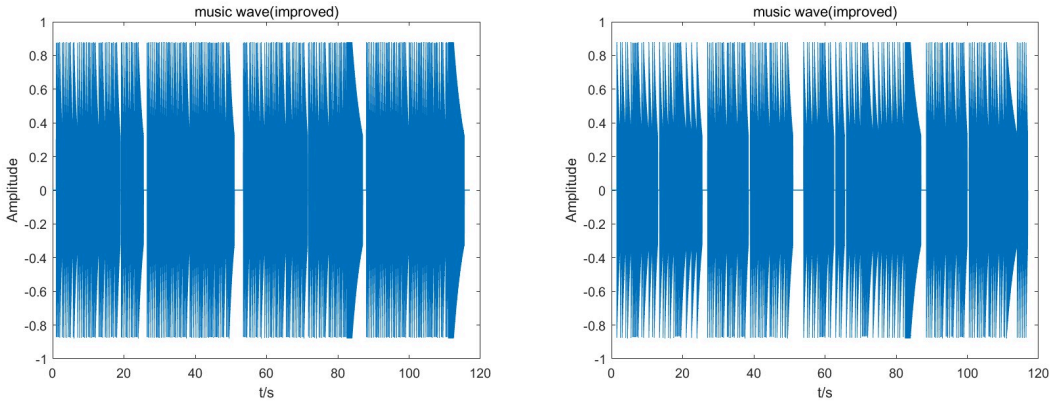


Fig.5 音乐波形图-千与千寻（左：声道1，右：声道2）

双声道使声音更加丰富，播放效果很好。

三、总结

本项目中通过MATLAB将简谱转化为音乐，关键步骤包括音符解析、音频生成、音量衰减、音色调整以及音频输出。解析简谱时，需要准确提取音符的时值、音高、调号等参数。利用MATLAB的音频处理功能生成音符对应的波形信号时，加入了声音衰减机制，使音符的结束更加自然。此外，通过调整波形的谐波成分，实现了音色的多样化，使生成的音乐更具真实感和表现力。最后，整合所有音符信号，并输出为可播放的音频文件。

在完成这个项目的过程中，从简谱解析到音频生成，每个环节都让我受益匪浅。在简谱解析阶段，需要将音乐的核心元素（音高、节奏等）提取并转化为程序可用的数据格式，这强化了我对音乐理论和数据结构的理解。在音频生成阶段，通过生成不同波形的信号，如正弦波，我掌握了如何利用MATLAB的音频处理函数模拟出真实的声音效果。

此外，在加入声音衰减和音色调整后，音乐表现力有了质的提升。通过研究声音衰减函数，如指数衰减函数，使音符的消失更符合自然规律；通过调整谐波成分，学习到如何模拟钢琴、长笛等不同乐器的音色特性。这部分的反复调试让我更加理解了音频信号的频谱特性与音质之间的关系，并对音频信号的特性和人耳感知之间的关系有了更加直观的理解。

整体而言，这个项目将编程与音乐艺术紧密结合，不仅让我加深了对MATLAB音频处理功能的应用，也让我体会到理论与实践结合的乐趣。这让我对音频信号处理有了全新的认识的同时，还学会了生成自己的电子音乐，收获了跨学科融合的成就感。