# 音乐合成实验报告

李栋庭 2020011222 无 16

2022.7.15

## 1 音乐合成

### 1.1 合成《东方红》

[**题目描述**] 请根据《东方红》片断的简谱和“十二平均律”计算出该片断中各个乐音的频率，在 MATLAB 中生成幅度为 1 、抽样频率为 8kHz 的正弦信号表示这些乐音。请用 sound 函数播放每个乐音，听一听音调是否正确。最后用这一系列乐音信号拼出《东方红》片断，注意控制每个乐音持续的时间要符合节拍，用 sound 播放你合成的音乐，听起来感觉如何?

[**主体思想**] 计算频率

[**核心代码**]

```
clear,clc;


fs = 8000;


%%%%%% test %%%%%%
% sound(get_tone(5,1,0));
% sound(get_tone(6,1,0));
% sound(get_tone(6,1,-1));
% sound(get_tone(2,1,0));
% sound(get_tone(1,1,0));


%%%%%% dongfanghong %%%%%%


part_1=[get_tone_1(5,0.5,0),get_tone_1(5,0.25,0),get_tone_1(6,0.25,0)];
part_2=get_tone_1(2,1,0);
part_3=[get_tone_1(1,0.5,0),get_tone_1(1,0.25,0),get_tone_1(6,0.25,-1)];
% sound(part_1,fs);
```

```
% sound(part_2,fs);
% sound(part_3,fs);

dongfanghong=[part_1,part_2,part_3,part_2];

plot(0:1/fs:4+7/fs,dongfanghong);
xlabel('t(s)')
ylabel('dongfanghong')

sound(dongfanghong,fs);
```

函数:

```
function y = get_tone_1(tone,rythm,upordown)
%1 = F 2/4
    %抽样频率
    fs = 8000;
    %序列
    t = 0:1/fs:rythm;
    %音调
    freqs=[349.23,392,440,493.88,523.25,587.33,659.25];
    freqs=freqs.*(2^upordown);
    %正弦序列
    y = sin(2*pi*freqs(tone)*t);
end
```

[**核心代码说明**] 根据音调，节奏，音调升降得到正弦序列。

[**运行结果**] 实现结果同预想一致, 相邻乐音之间有“啪”的杂声, 噪声较大。

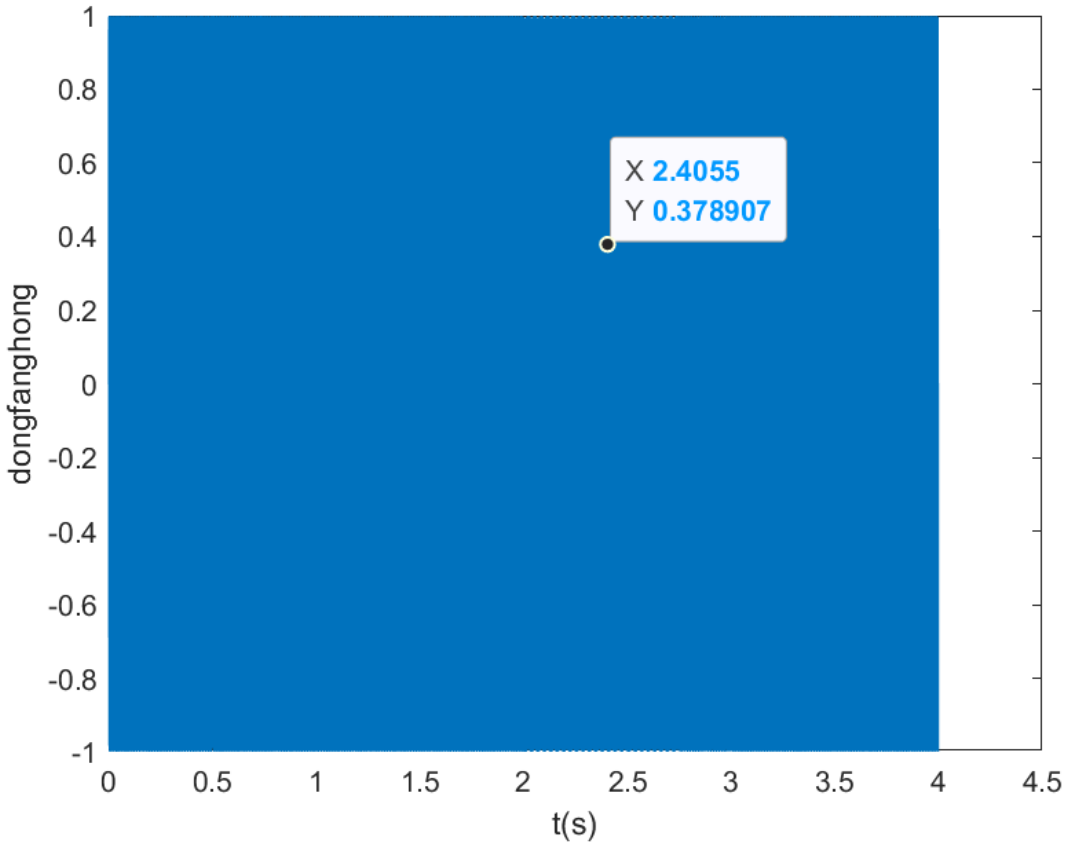


图 1: 声音序列

[**结果分析**] 如上, 即可用单频信号合成音乐, 可以调节节奏快慢, 基调。

### 1.2 消除噪声

[**题目描述**] 你一定注意到 (1) 的乐曲中相邻乐音之间有“啪”的杂声，这是由于相位不连续产生了高频分量。这种噪声严重影响合成音乐的质量，丧失真实感。为了消除它，我们可以用图所示包络修正每个乐音，以保证在乐音的邻接处信号幅度为零。此外建议用指数衰减的包络来表示。

[**主体思想**] 用指数衰减的包络来修正每个乐音。

[**核心代码**]

函数:

```
function y = get_tone_2(tone,rythm,upordown)
%1 = F 2/4
    %抽样频率
    fs = 8000;

    %序列
```

```
    t = 0:1/fs:rythm;
    %音调
    freqs=[349.23,392,440,493.88,523.25,587.33,659.25];
    freqs=freqs.*(2^upordown);
    %正弦序列
    y = sin(2*pi*freqs(tone)*t).*(t/rythm).*exp(-7.*t./rythm);
end
```

[**核心代码说明**] 给声音加上 $ax * e^{-bx}$ 的包络，通过一定的调参得到 $ax * e^{-bx}$ 中最合适参数 a，b。

[**运行结果**] 实现结果同预想一致，极大地消除了相邻乐音之间有“啪”的杂声。

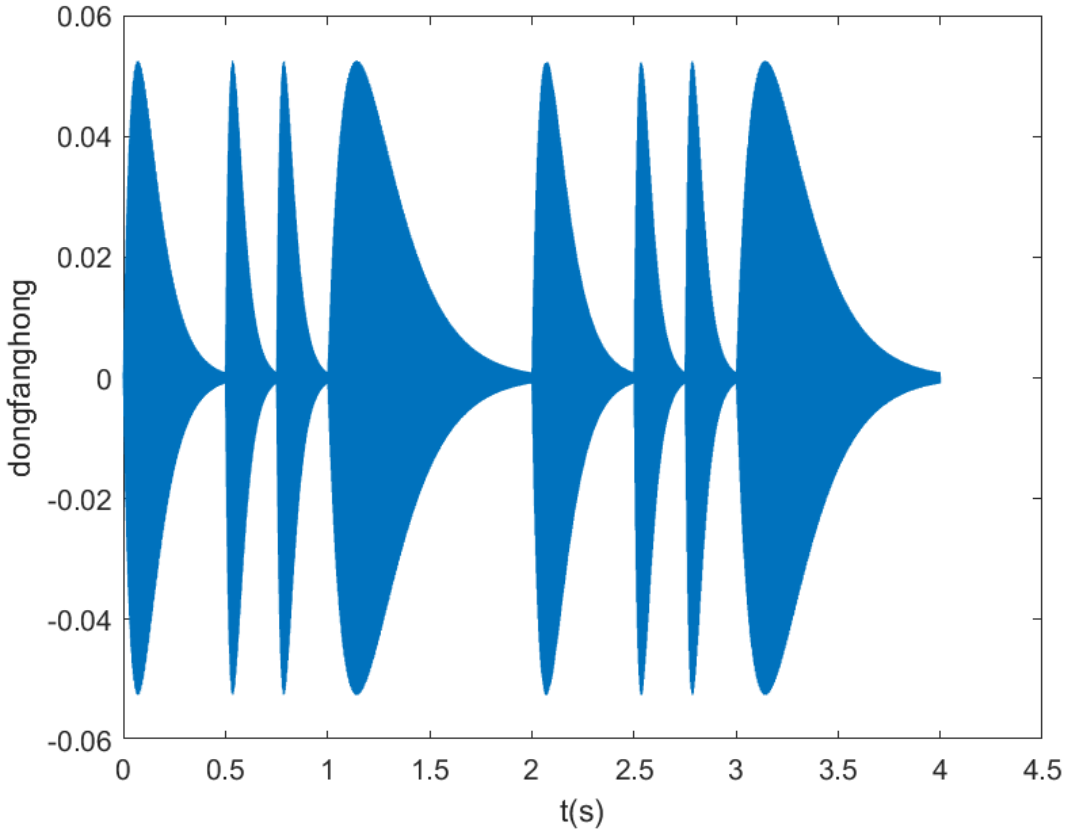


图 2: 声音序列

[**结果分析**] 我们用指数包络修正每个乐音，以保证在乐音的邻接处信号幅度为零，从而避免相位不连续产生了高频分量。

### 1.3 升降调

[**题目描述**] 请用最简单的方法将 (2) 中的音乐分别升高和降低一个八度。(提示：音乐播放的时间可以变化）再难一些，请用 resample 函数（也可以用 interp 和 decimate 函数）将上述音乐

升高半个音阶。(提示：视计算复杂度，不必特别精确)

[**主体思想**] 可以在生成序列时直接处理，也可以重采样。

[**核心代码**]

升八度：

```
clear,clc;

fs = 8000;

%%%%%% dongfanghong %%%%%%

% 升八度
part_1=[get_tone_3(5,0.5,1),get_tone_3(5,0.25,1),get_tone_3(6,0.25,1)];
part_2=get_tone_3(2,1,1);
part_3=[get_tone_3(1,0.5,1),get_tone_3(1,0.25,1),get_tone_3(6,0.25,0)];

dongfanghong=[part_1,part_2,part_3,part_2];
xlabel('t(s)')
ylabel('dongfanghong')

plot(0:1/fs:4+7/fs,dongfanghong);
sound(dongfanghong,fs);
```

降八度：

```
clear,clc;

fs = 8000;

%%%%%% dongfanghong %%%%%%

% 降八度
part_1=[get_tone_3(5,0.5,-1),get_tone_3(5,0.25,-1),get_tone_3(6,0.25,-1)];
part_2=get_tone_3(2,1,-1);
part_3=[get_tone_3(1,0.5,-1),get_tone_3(1,0.25,-1),get_tone_3(6,0.25,-2)];

dongfanghong=[part_1,part_2,part_3,part_2];
xlabel('t(s)')
```

```
ylabel('dongfanghong')

plot(0:1/fs:4+7/fs,dongfanghong);
sound(dongfanghong,fs);
```

升高半个音阶:

```
clear,clc;

fs = 8000;

%%%%%% dongfanghong %%%%%%

part_1=[get_tone_3(5,0.5,0),get_tone_3(5,0.25,0),get_tone_3(6,0.25,0)];
part_2=get_tone_3(2,1,0);
part_3=[get_tone_3(1,0.5,0),get_tone_3(1,0.25,0),get_tone_3(6,0.25,-1)];

dongfanghong=[part_1,part_2,part_3,part_2];

% 升高半个音阶
fs_up=fs/2^(1/12);
dongfanghong_up=interp1(0:1/fs:4+7/fs,dongfanghong,0:1/fs_up:4);

plot(0:1/fs:4+7/fs,dongfanghong);
xlabel('t(s)')
ylabel('dongfanghong')

sound(dongfanghong_up,fs);
```

函数:

```
function y = get_tone_3(tone,rythm,upordown)
%1 = F 2/4
    %抽样频率
    fs = 8000;

    %序列
    t = 0:1/fs:rythm;
    %音调
```

```
    freqs=[349.23,392,440,493.88,523.25,587.33,659.25];
    freqs=freqs.*(2^upordown);
    %正弦序列
    y = sin(2*pi*freqs(tone)*t).*(t/rythm).*exp(-7.*t./rythm);
end
```

[**核心代码说明**] 升八度和降八度在生成序列时直接处理，升高半个音阶采用降采样的方法。

[**运行结果**] 实现结果同预期一致，分别升高和降低一个八度较为准确。

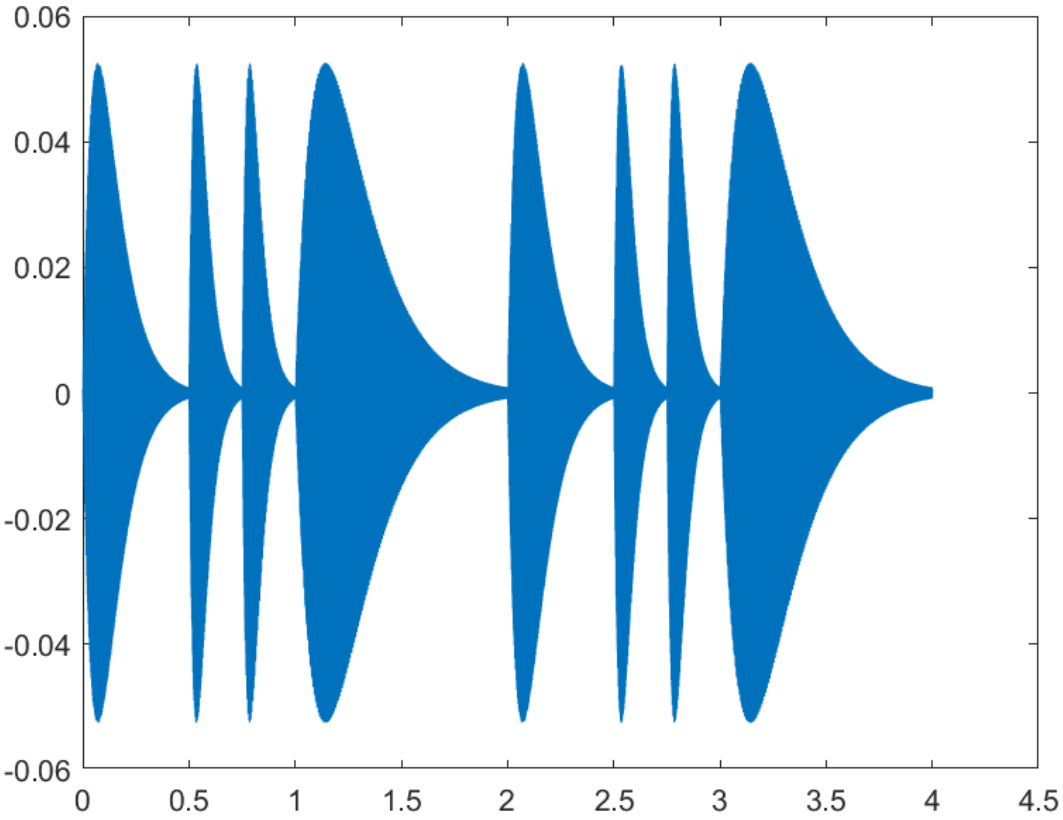


图 3: 升八度声音序列

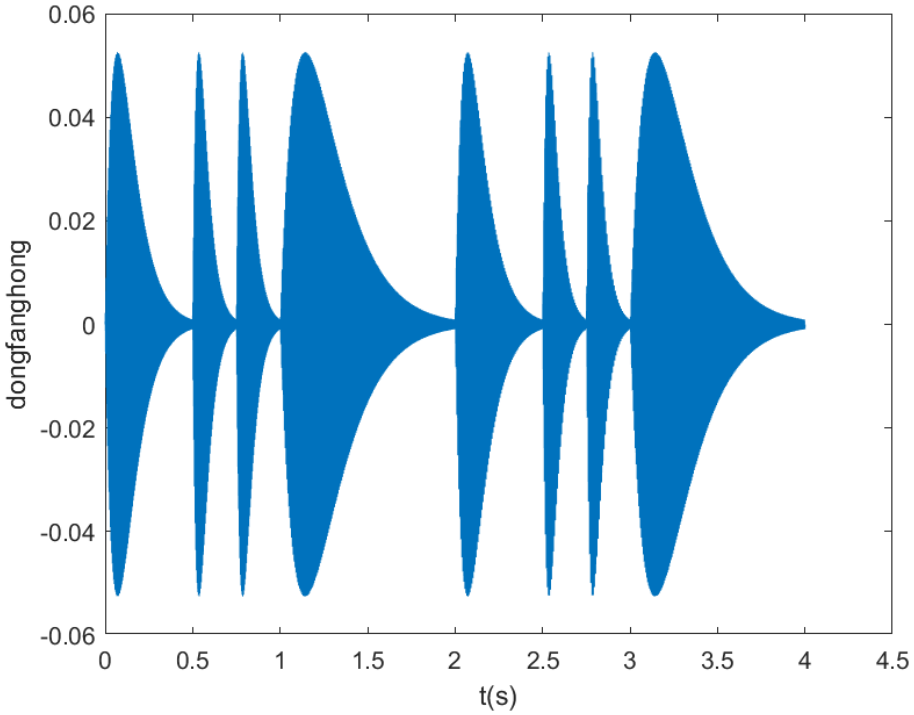


图 4: 降八度声音序列

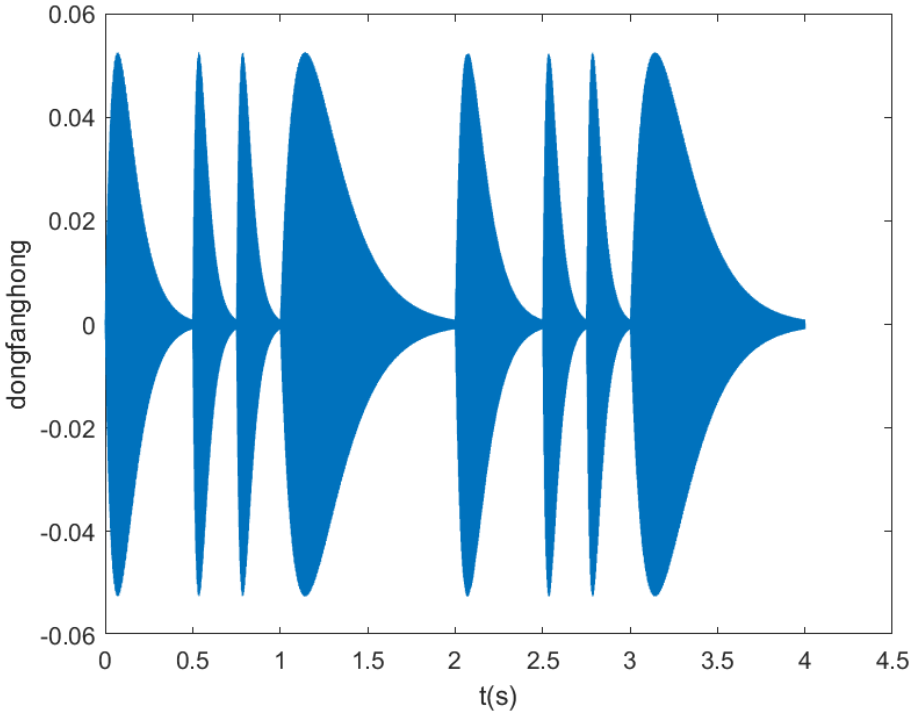


图 5: 升半阶声音序列

**[结果分析]** interp 函数可以进行重新采样，给声音变调提供便捷的方式。

## 1.4 加入谐波分量

**[题目描述]** 试着在 (2) 的音乐中增加一些谐波分量，听一听音乐是否更有“厚度”了？注意谐波分量的能量要小，否则掩盖住基音反而听不清音调了。（如果选择基波幅度为 1 ，二次谐波幅度 0.2 ，三次谐波幅度 0.3 ，听起来像不像象风琴?）

**[主体思想]** 改变 gettone 函数，加入谐波分量。

**[核心代码]**

函数：

```
function y = get_tone_4(tone,rythm,upordown)
%1 = F 2/4
    %抽样频率
    fs = 8000;

    %序列
    t = 0:1/fs:rythm;
    %音调
    freqs=[349.23,392,440,493.88,523.25,587.33,659.25];
    freqs=freqs.*(2^upordown);
    %正弦序列
    y = sin(2*pi*freqs(tone)*t).*(t/rythm).*exp(-7.*t./rythm);
    y = y+ 0.2*sin(4*pi*freqs(tone)*t).*(t/rythm).*exp(-7.*t./rythm);
    y = y+ 0.3*sin(6*pi*freqs(tone)*t).*(t/rythm).*exp(-7.*t./rythm);
end
```

**[核心代码说明]** 相较于上一题的代码，主要加入了高频谐波。

**[运行结果]** 运行结果同预期一致，音乐更有“厚度”。

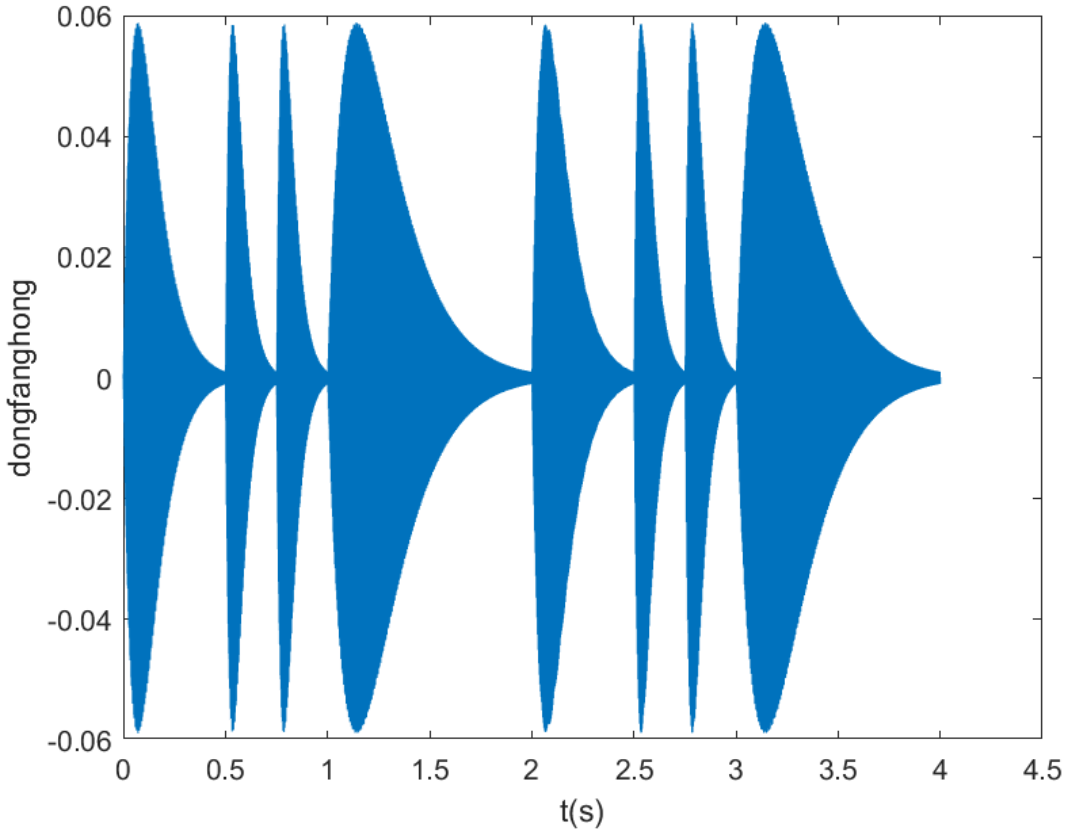


图 6: 声音序列

[**结果分析**] 音乐听起来更有“厚度”，选择基波幅度为 1 ，二次谐波幅度 0.2 ，三次谐波幅度 0.3 ，听起来较像象风琴，只选用一种频率使音乐过于单调。

## 1.5 自选其它音乐合成

[**题目描述**] 自选其它音乐合成，例如贝多芬第五交响乐的开头两小节。

[**主体思想**] 选择的曲目是《夜曲》的前奏，同前面题目，主要是改变节奏和音调。

[**核心代码**]

```
clear,clc;


fs = 8000;


%%%%%% 夜曲 %%%%%%
% 前奏
tone=[6,7,1,1,1,1,1,7,3,3,3,5,6,6,6,6,5,4,5,1,1,1,4,4,4,4,4,3,7,3,2,2,1,7,7,1,1
,7,6,7,1,1,1,1,1,7,3,3,3,5,6,6,6,6,5,4,5,1,1,1,4,4,4,4,4,3,2,1,7,7,1,6,6];
rythm=[0.25,0.25,0.5,0.25,0.25,0.25,0.75,0.5,0.25,0.25,1,0.25,0.25,0.25,0.25,0.25,
```

```
0.5,0.25,0.5,0.25,0.25,1,0.5,0.25,0.25,0.75,0.25,0.5,0.5,0.5,0.5,0.5,0.25,0.25,
0.25,0.75,1,0.5,0.25,0.25,0.5,0.25,0.25,0.25,0.75,0.5,0.25,0.25,1,0.25,0.25,0.25,
0.25,0.25,0.5,0.25,0.5,0.25,0.25,1,0.5,0.25,0.25,0.75,0.25,0.5,0.5,0.5,0.5,0.5,
0.25,0.25,2];
upordown=[0,0,1,1,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,1,1,1,0,0,
1,1,0,0,0,1,1,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,1,0,0];

yequ=zeros(1,int32(fs*60/87/2));

for idx = 1:73
    yequ=[yequ,get_tone_5(tone(idx),rythm(idx),upordown(idx))];
end

plot(0:1/fs:176592/fs,yequ);
xlabel('t(s)')
ylabel('yequ')

sound(yequ,fs);
```

函数:

```
function y = get_tone_5(tone,rythm,upordown)
%1 = F 2/4
    %抽样频率
    fs = 8000;
    %节拍
    beat = 60/87;
    %序列
    t = 0:1/fs:rythm*beat;
    %音调降A大调
    freqs=[207.65,233.08,261.63,277.18,311.13,349.23,392];
    freqs=freqs.*(2^upordown);
    %正弦序列
    y = sin(2*pi*freqs(tone)*t).*(t/rythm).*exp(-10*t./rythm);
    y = y+ 0.2*sin(4*pi*freqs(tone)*t).*(t/rythm).*exp(-10*t./rythm);
    y = y+ 0.3*sin(6*pi*freqs(tone)*t).*(t/rythm).*exp(-10*t./rythm);
end
```

[**核心代码说明**] 选择基波幅度为 1 ，二次谐波幅度 0.2 ，三次谐波幅度 0.3 ，听起来较像象风琴。

[**运行结果**] 运行结果与预期一致。

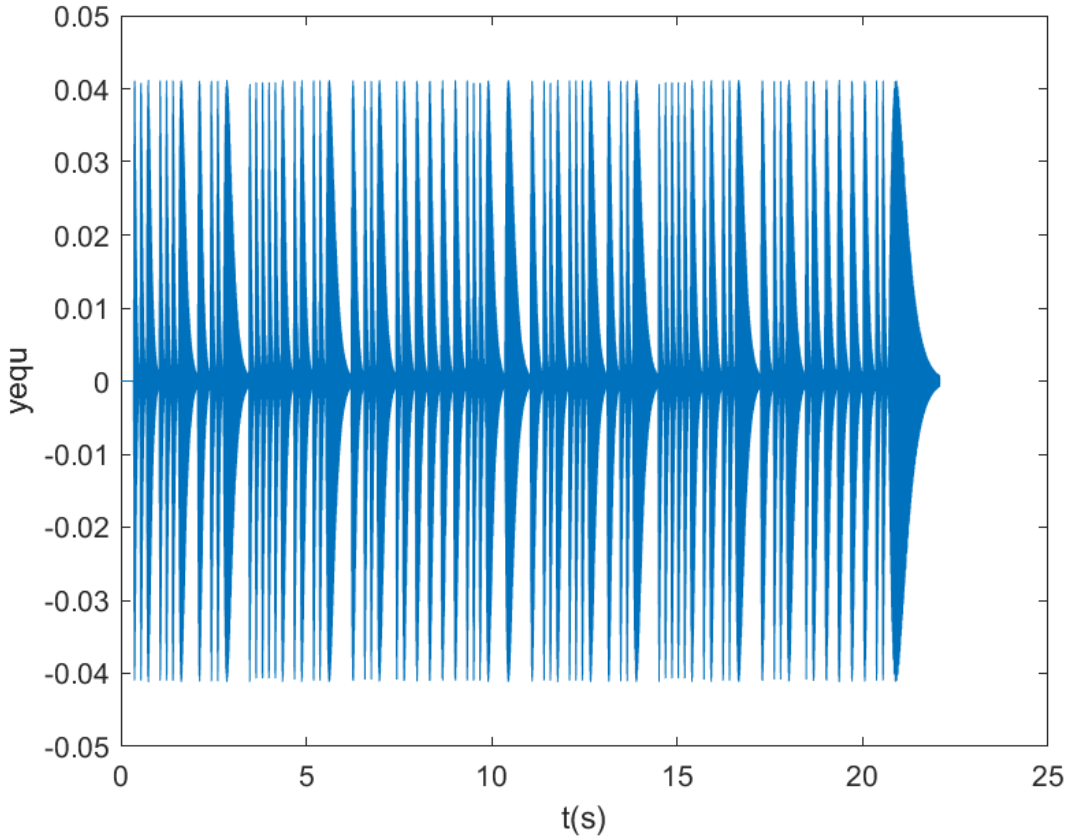


图 7: 声音序列

[**结果分析**] 听起来还可以，但是还是缺乏一些真实感。

### 1.6 播放 fmt.wav

[**题目描述**] 先用 wavread 函数载入光盘中的 fmt.wav 文件，播放出来听听效果如何？是否比刚才的合成音乐真实多了？

[**主体思想**] 用 wavread 函数载入光盘，然后直接播放。

[**核心代码**]

```
clear;clc;


fs=8000;
T=1/fs;
```

```
load("attachments/guitar.mat");

fmt=audioread("attachments/fmt.wav");

N = length(fmt);
t= (0:N-1)*T;

plot(t,fmt);
xlabel('t (s) ')
ylabel('fmt.mav')

sound(fmt);
```

[**核心代码说明**] 默认一拍 0.5 秒。

[**运行结果**]

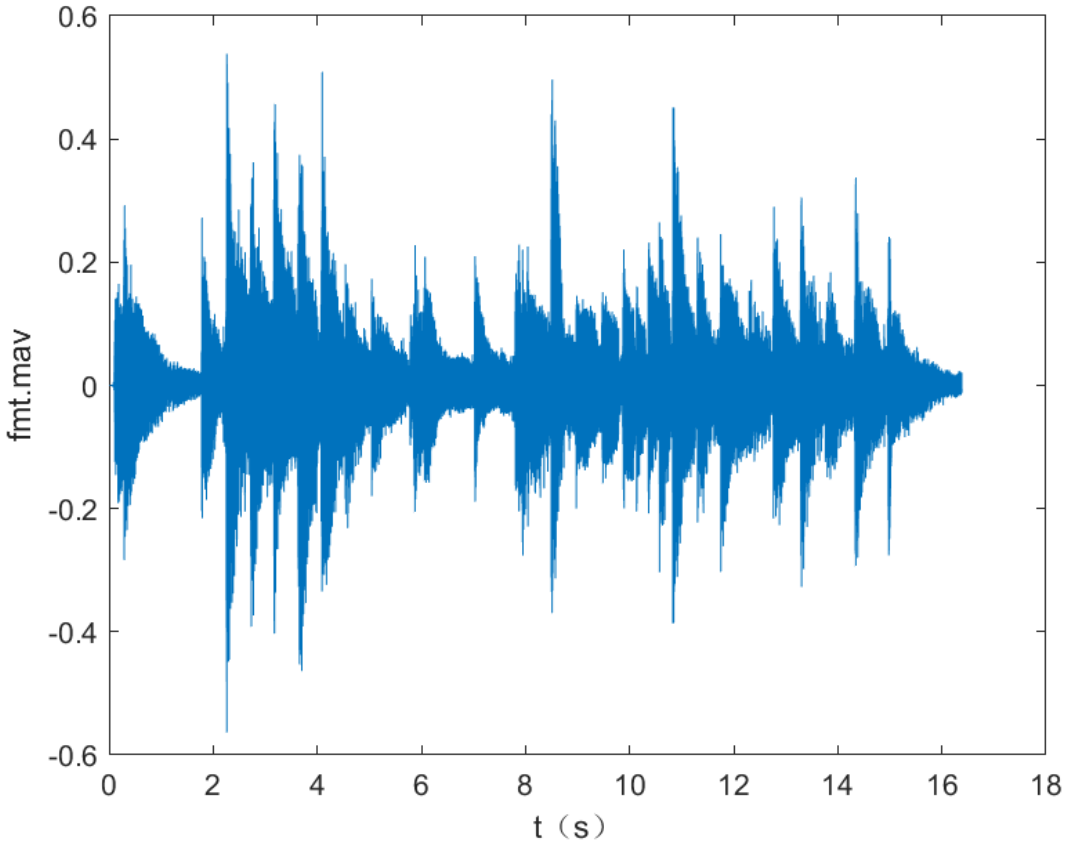


图 8: 声音序列

[**结果分析**] 确实比刚才的合成音乐真实多了，主要特点:

1. 叠音现象频繁, 吉他拨弦音衰减很慢

2. 和弦, 同时弹奏多个音

3. 泛音丰富

4. 不同的音振幅不同, 强弱分明

## 1.7 预处理 realwave

[**题目描述**] 你知道待处理的 wave2proc 是如何从真实值 realwave 中得到的么? 这个预处理过程可以去除真实乐曲中的非线性谐波和噪声, 对于正确分析音调是非常重要的。提示: 从时域做, 可以继续使用 resample 函数。

[**主体思想**] 仔细分析 wave2proc 的时域波形:

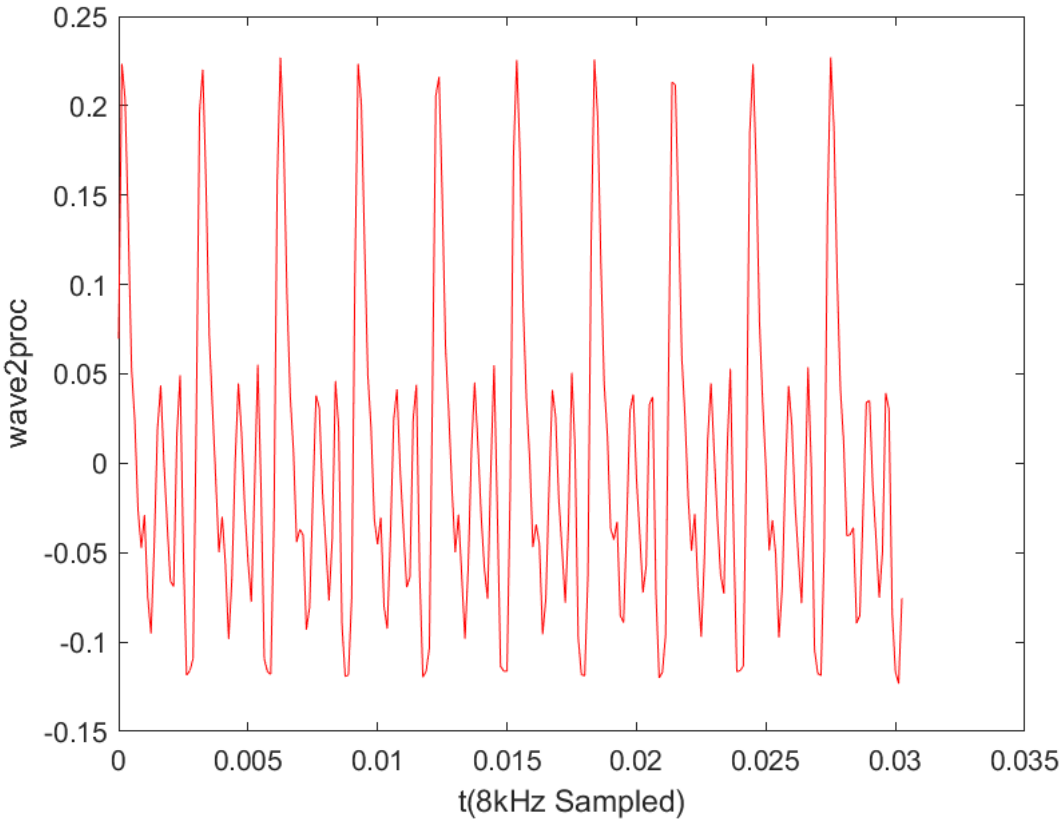


图 9: 声音序列

大致有十个周期, 而 length(wave2proc)=243, 利用 interp1(tup,resampledwave2proc,t)'; 进行重采样, 使其长度成为 10 的倍数。从时域上看, 十个周期几乎完全一样, 要从原波形 realwave 得到上图波形, 可采用平均去噪的方法。

[**核心代码**]

```
clear;clc;


load("attachments/guitar.mat");
```

```
fs=8000;
T=1/fs;
N=243;
t= (0:N-1)*T;

figure;
plot(t,realwave,'b-');
xlabel('t(8kHz Sampled)')
ylabel('realwave')

figure;
plot(t,wave2proc,'r-');
xlabel('t(8kHz Sampled)')
ylabel('wave2proc')
t_up=0:1/fs/10:0.03+9/fs/10;

prewave=interp1(t,realwave',t_up);

A = reshape(prewave,241,10).';
p = mean(A);
resampled_wave2proc = repmat(p.',10,1);

my_wave=interp1(t_up,resampled_wave2proc,t)';

figure;
plot(t,my_wave,'g-');
xlabel('t(8kHz Sampled)')
ylabel('mywave')
```

[**核心代码说明**] 主要采用重采样实现平均去噪的方法。

[**运行结果**]

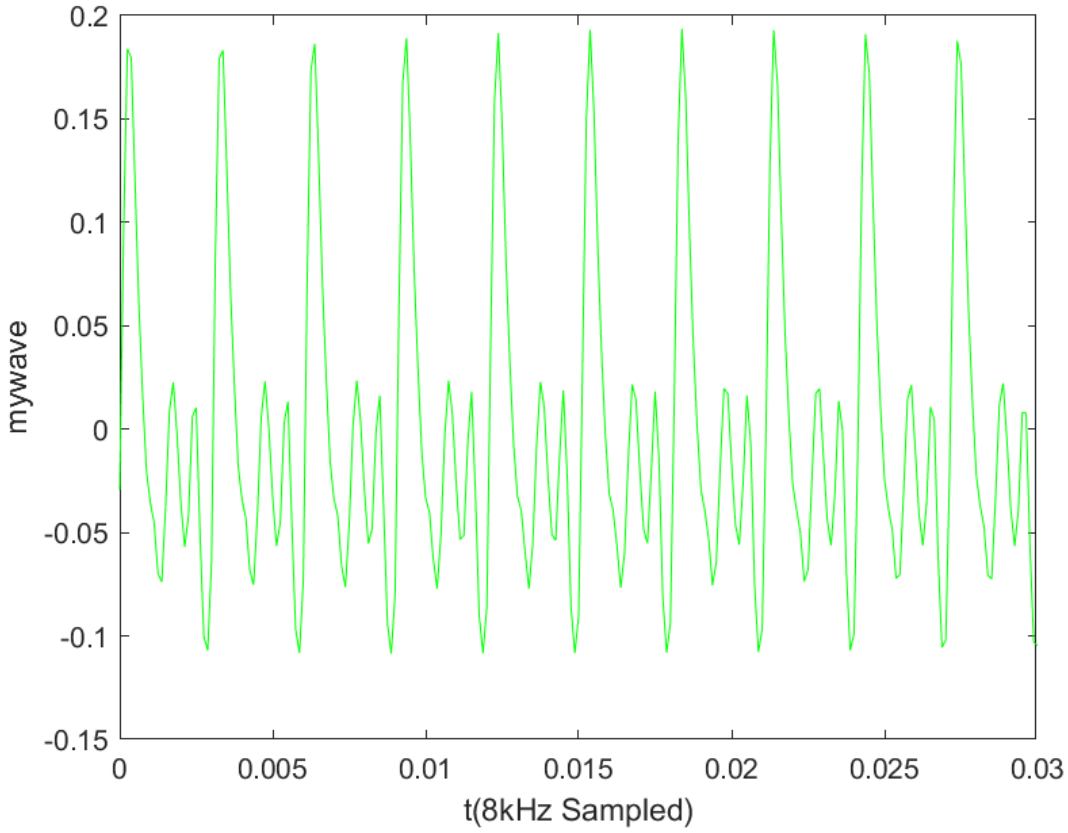


图 10: 声音序列

[**结果分析**] 由图像看出, 用以上方法得到的 mywave 与 wave2proc 几乎一样, 于是将以上过程作为去除非线性谐波和噪声的预处理过程

## 1.8 分析基频

[**题目描述**] 这段音乐的基频是多少? 是哪个音调? 请用傅里叶级数或者变换的方法分析它的谐波分量分别是什么。提示: 简单的方法是近似取出一个周期求傅里叶级数但这样明显不准确, 因为你应该已经发现基音周期不是整数（这里不允许使用 resample 函数）。复杂些的方法是对整个信号求傅里叶变换（回忆周期性信号的傅里叶变换）, 但你可能发现无论你如何提高频域的分辨率, 也得不到精确的包络（应该近似于冲激函数而不是 sinc 函数）, 可选的方法是增加时域的数据量, 即再把时域信号重复若干次, 看看这样是否效果好多了? 请解释之。

[**主体思想**] 近似取出一个周期求傅里叶级数但这样明显不准确, 对整个信号求傅里叶变换也不够精准, 所以将该周期信号重复 10 次, 在求傅里叶变换。

[**核心代码**]

```
clear;clc;


load("attachments/guitar.mat");
```

```
fs=8000;
T=1/fs;
N=243;
t= (0:N-1)*T;
figure;
fft0=fft(wave2proc);
P2=abs(fft0/N);
P1=P2(1:(N+1)/2);
f1=fs*(0:(N/2))/N;
plot(f1,P1);
xlabel('f1(Hz)')
ylabel('|P1(f)|')

%延拓
extend_wave=wave2proc;
for idx=1:9
    extend_wave=[extend_wave;wave2proc];
end
N_e=N*10;
figure;
fft1=fft(extend_wave);
P3=abs(fft1/N_e);
P4=P3(1:N_e/2+1);
f2=fs*(0:(N_e/2))/N_e;
plot(f2,P4);
xlabel('f2(Hz)')
ylabel('|P4(f)|')

freq_arg=find(P4>max(P4)/2)';
freqs=f2(freq_arg);
amps=P4(freq_arg);
amp_s=amps/amps(1);
disp(freqs(1));

ampandfreq = [amp_s';freqs];
```

**[核心代码说明]** 主要操作是将信号重复十次，代码直接给出基频，打印出来，并将频率与幅度保存在 code/hw_1_8_saved.mat 文件中。

**[运行结果]**

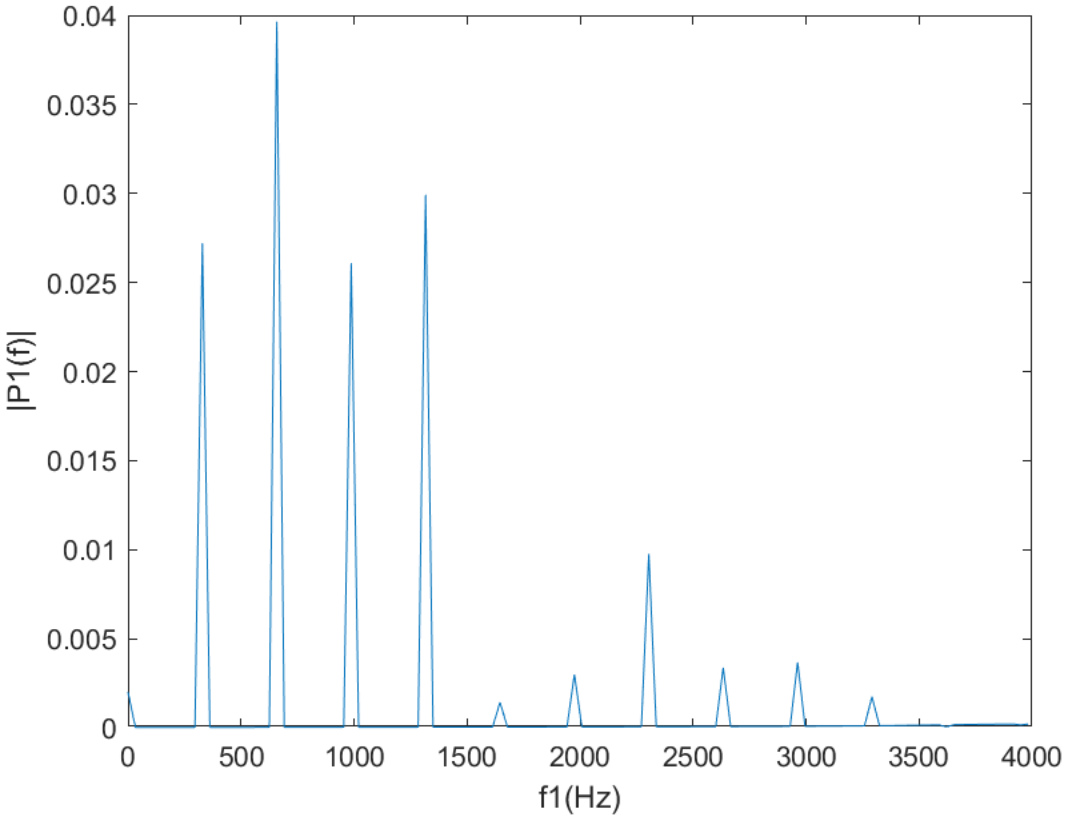


图 11: 未重复频谱序列

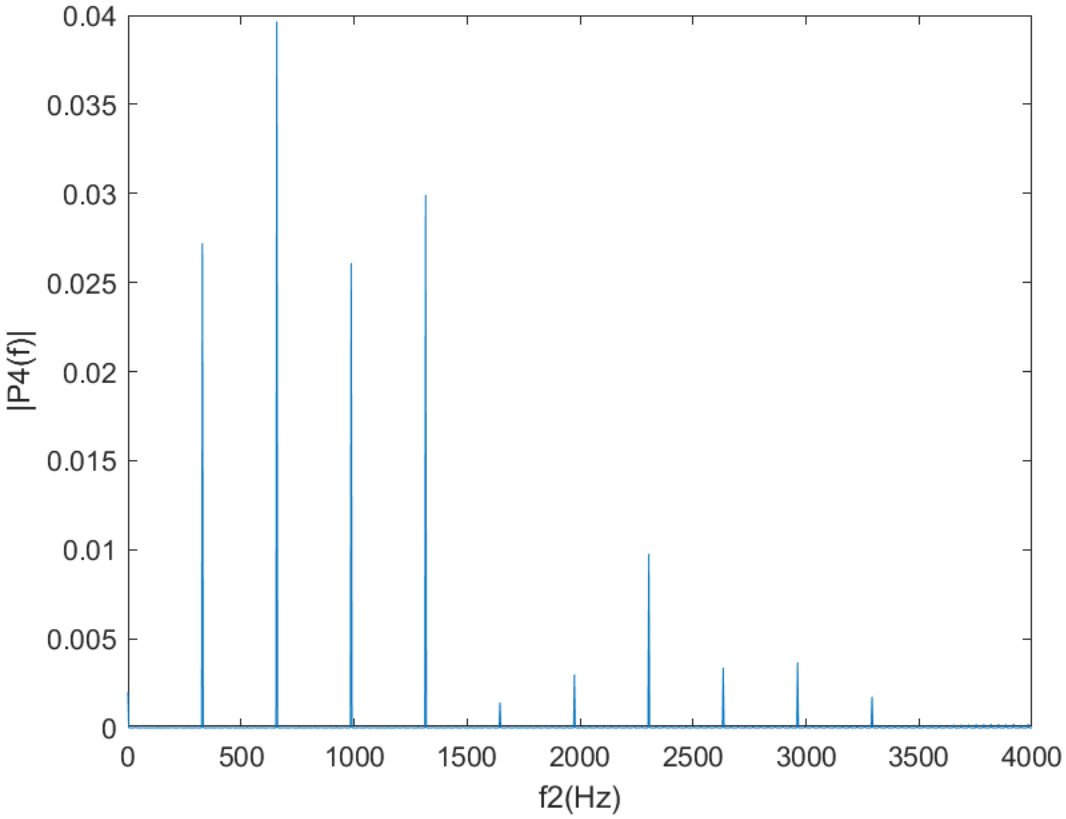


图 12: 重复后频谱序列

[**结果分析**] 基频为 329.2Hz, 结果与上一种方法相同, 频谱更接近冲激函数；当脉冲数增多直至趋于无穷, 即成为周期信号, 频谱由连续谱退化为离散谱, 由分立的冲激函数构成。

## 1.9 自动分析

[**题目描述**] 再次载入 fmt.wav ，现在要求你写一段程序，自动分析出这段乐曲的音调和节拍! 如果你觉得太难就允许手工标定出每个音调的起止时间，再不行你就把每个音调的数据都单独保存成一个文件，然后让 MATLAB 对这些文件进行批处理。注意：不允许逐一地手工分析音调。编辑音乐文件，推荐使用“CoolEdit”编辑软件。

[**主体思想**] 如何对不同音调的节拍进行切割是这里比较难以处理的地方，通过对音乐时域波形进行观察，大致的可以看出每个节拍开始时会有比较明显的跃迁，为了效率，我将音乐按照每 100 一批进行切割，再每批中找到最大值与最小值，如果其差值大于一定范围，即可确定这批时两个节拍的分割点，在重新切割处理。

比较难的一点是吉他有和弦存在，所以对每个音调处理时，会有多种频率，只能人工查验幅度来确定基频。

[**核心代码**]

```
clear;clc;
```

```
fs=8000;
T=1/fs;
fmt=audioread("attachments/fmt.wav");
N = length(fmt);
t= (0:N-1)*T;
[freqs,beats]=get_freq_beat(fmt');
tones = freq2tone_C(freqs);
tonesandbeats = [tones;beats];
% 分析音调
% 246.94
temp_B = fmt(125000:125100);
fourier_B=get_fourier(temp_B);
% 1--1+0.8+2.7 2--1
% 1 0.22
%207.65
%1--1+0.4 2--0.3 3--0.8]
%1 0.22 0.57
% 220
temp_A = fmt(79500:79600);
for idx=1:4;
    temp_A=[temp_A;temp_A];
end
fourier_A=get_fourier(temp_A);
```

函数 getfreqbeat:

```
function [freqs,beats] = get_freq_beat(demo)
    freqs = [];
    beats = [];
    batchs = [];
    slices=[];
    min_step = 0.31;
    % 四分之一拍
    min_gap = 19;
    batch_N = 100;
    pre_slice=-100;
    batch_len=int32(length(demo)/batch_N-1);
```

```
    for idx=1:batch_len
        batchs=[batchs;demo(idx*batch_N-batch_N+1:idx*batch_N)];
    end
    for idx=1:batch_len
        if max(batchs(idx,:))-min(batchs(idx,:))>min_step
            if(idx-pre_slice>min_gap)
                slices=[slices,idx];
                pre_slice=idx;
            end
        end
    end
    slices=[slices,batch_len];
    for idx= 1:length(slices)-1
        temp=demo(slices(idx)*batch_N+batch_N:int32((slices(idx)+slices(idx+1))*batch_N/2));
        freqs=[freqs,get_baseband(temp)];
        if slices(idx+1)-slices(idx)<25
            beats=[beats,0.5];
        else
            if slices(idx+1)-slices(idx)<45
                beats=[beats,1];
            else
                if slices(idx+1)-slices(idx)<85
                    beats=[beats,2];
                else
                    if slices(idx+1)-slices(idx)<125
                            beats=[beats,3];
                    else
                            beats=[beats,4];
                    end
                end
            end
        end
    end
end
```

[**核心代码说明**] 先分批确定节拍,再分批确定基频等,原始数据保存在 code/hw_1_9_saved.mat

文件中。

[**运行结果**] 以 C 大调为基准：

| 唱名 | 拍数 |
|---|---|
| 低音 6 | 半拍 |
| 低音 6，3 | 3 |
| 低音 7 | 1 |
| 低音 6 | 半拍 |
| 低音 6 | 半拍 |
| 2 | 半拍 |
| 低音 7 | 半拍 |
| 3 | 1 |
| 低音 5 | 半拍 |
| 5 | 半拍 |
| 低音 4 | 1 |
| 低音 4 | 1 |
| 低音 4 | 2 |
| 低音 3 | 3 |
| 3 | 2 |
| 低音 6 | 半拍 |
| 低音 6 | 1 |
| 低音 6 | 1 |
| 6 | 2 |
| 5 | 半拍 |
| 4 | 半拍 |
| 3 | 半拍 |
| 2 | 半拍 |
| 1 | 1 |
| 低音 7 | 1 |
| 低音 7，2 | 2 |
| 低音 4，0 | 1 |
| 低音 6 | 2 |
| 低音 6 | 2 |
| $^{b}$ 低音 6 | 2 |
| 低音 3，$^{b}$6 | 3 |

[**结果分析**] 吉他弹奏带有和弦 (同一时刻弹奏多个音), 对分析音调带来了困难, 由于本方法没有提前将每个音开始的时间标定出来, 而是仅由频率的变化推断音的开始和结束, 弹奏吉他过

程中手指摩擦琴弦产生的高频分量极易对音符的分割判断造成影响

手工分析这段吉他音, 大概有 32 个音, 而上述表格中的自动分析结果分析出了 31 个音 (和弦算作一个音, 即上表用逗号隔开的两个音即为同时弹奏的和弦), 还是较为准确的。

## 1.10 高级合成

[**题目描述**] 用 (7) 计算出来的傅里叶级数再次完成第 (4) 题，听一听是否像演奏 fmt.wav 的吉他演奏出来的?

[**主体思想**] 着在 (2) 的音乐中增加一些谐波分量，这些谐波分量来自 (7) 的傅里叶级数。

[**核心代码**]

```
function y = get_tone_10(tone,rythm,upordown)
%1 = F 2/4
    %抽样频率
    fs = 8000;
    %节拍

    %序列
     t = 0:1/fs:rythm;
    %音调F大调
    freqs=[349.23,392,440,493.88,523.25,587.33,659.25];
    freqs=freqs.*(2^upordown);
    %幅度
    load('hw_1_8_saved.mat');
    amp = ampandfreq(1,:);
    %正弦序列
    y = sin(2*pi*freqs(tone)*t).*(t/rythm).*exp(-10*t./rythm);
    y = y+ amp(2)*sin(4*pi*freqs(tone)*t).*(t/rythm).*exp(-10*t./rythm);
    y = y+ amp(3)*sin(6*pi*freqs(tone)*t).*(t/rythm).*exp(-10*t./rythm);
    y=  y+ amp(4)*sin(8*pi*freqs(tone)*t).*(t/rythm).*exp(-10*t./rythm);
end
```

[**核心代码说明**] 使用 (7) 的傅里叶级数进行改进。

[**运行结果**]

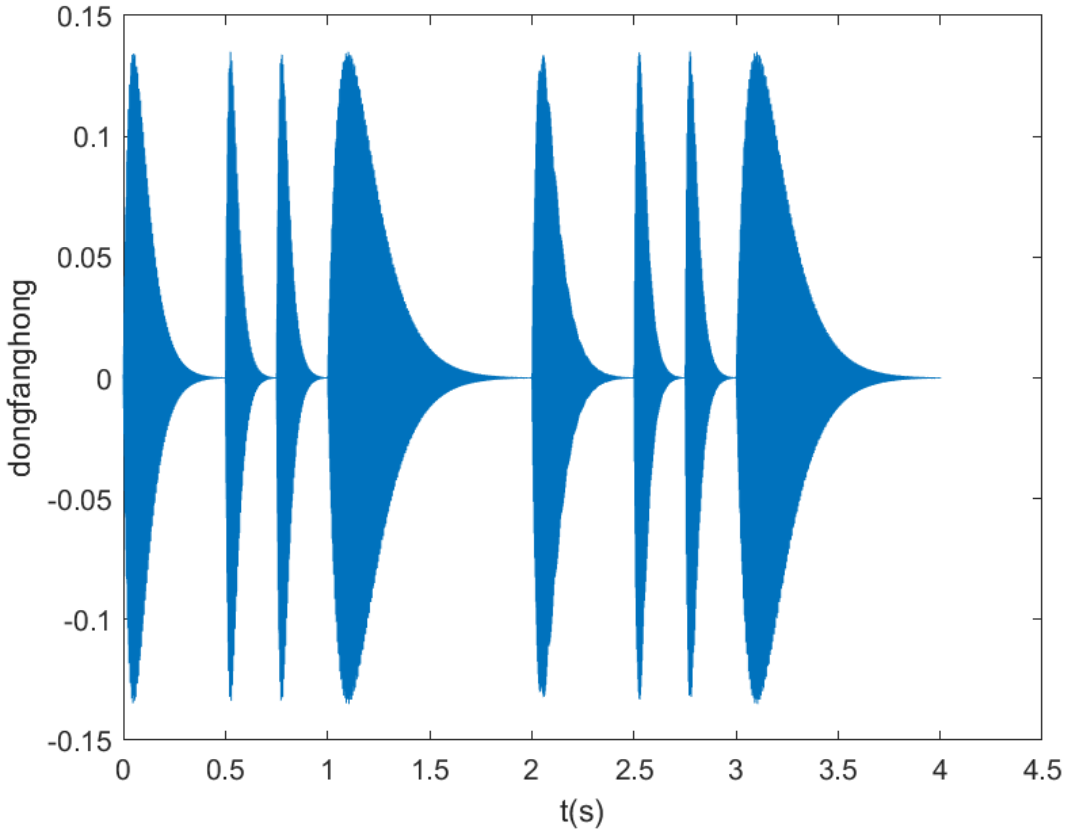


图 13: 声音序列

[**结果分析**] 听起来有拨弦的感觉, 音色虽然接近吉他但听起来还是不像, 这大概是因为吉他弹奏一个音符后, 音符衰减过程很缓慢, 下一个音的弹奏总是会与上个音的衰减阶段有相当部分的重合, 而 sound 函数的拼接是没有叠音效果的, 即上一个音符完全衰减至零后, 下一个音符才会发声; 另外, 吉他的和弦效果在东方红的谱子中无法体现。

## 1.11 演奏《东方红》

[**题目描述**] 也许 (9) 还不是很像，因为对于一把泛音丰富的吉他而言，不可能每个音调对应的泛音数量和幅度都相同。但是通过完成第 (8) 题，你已经提取出 fmt.wav 中的很多音调，或者说，掌握了每个音调对应的傅里叶级数，大致了解了这把吉他的特征。现在就来演奏一曲《东方红》吧。提示：如果还是音调信息不够，那就利用相邻音调的信息近似好了，毕竟可以假设吉他的频响是连续变化的。

[**主体思想**] 由于 fmt 音乐音调较低, 东方红前三个音高音 1, 高音 1, 高音 2 缺乏足够的泛音信息, 故取 amp = [1，0.22，0，0，0，0], 即只有基频与二次分量，其他音节在 (9) 中分析，详见 hw_1_9.m。

[**核心代码**]

函数:

```
function y = get_tone_11(tone,rythm,upordown)
%1 = F 2/4
    %抽样频率
    fs = 8000;
    %节拍

    %序列
     t = 0:1/fs:rythm/2;
    %音调F大调
    freqs=[349.23,392,440,493.88,523.25,587.33,659.25];
    freqs=freqs.*(2^upordown);
    %幅度
    load('hw_1_8_saved.mat');
    % 谐波
    amp=[1,0.22,0,0,0,0];
    if tone == 6 && upordown==-1
        amp=[1 ,0.9, 0.25, 0, 0, 0.2];
    else
        if tone == 1 && upordown==0
            amp = [1, 0.2, 0.1, 0.1, 0, 0];
        else
            if tone ==2 && upordown ==0
                amp = [1, 0.22, 0, 0, 0, 0];
            else
                if  tone == 7 && upordown==-1
                    amp=[1 ,1.5, 1.1,1, 0, 0]/4;
                else
                    if tone == 5 && upordown==-1
                        amp=[1 ,0.6, 0, 0, 0, 0];
                    end
                end
            end
        end
    end
    %正弦序列
    if tone==0
```

```
        t(t>0)=0;
        y=t;
    else
        y = sin(2*pi*freqs(tone)*t).*(t/rythm).*exp(-10*t./rythm);
        for idx=2:6
            y = y+ amp(idx)*sin(2*idx*pi*freqs(tone)*t).*(t/rythm).*exp(-10*t./rythm);
        end
    end
end
```

[**核心代码说明**] 主要改变分量的幅度。

[**运行结果**]

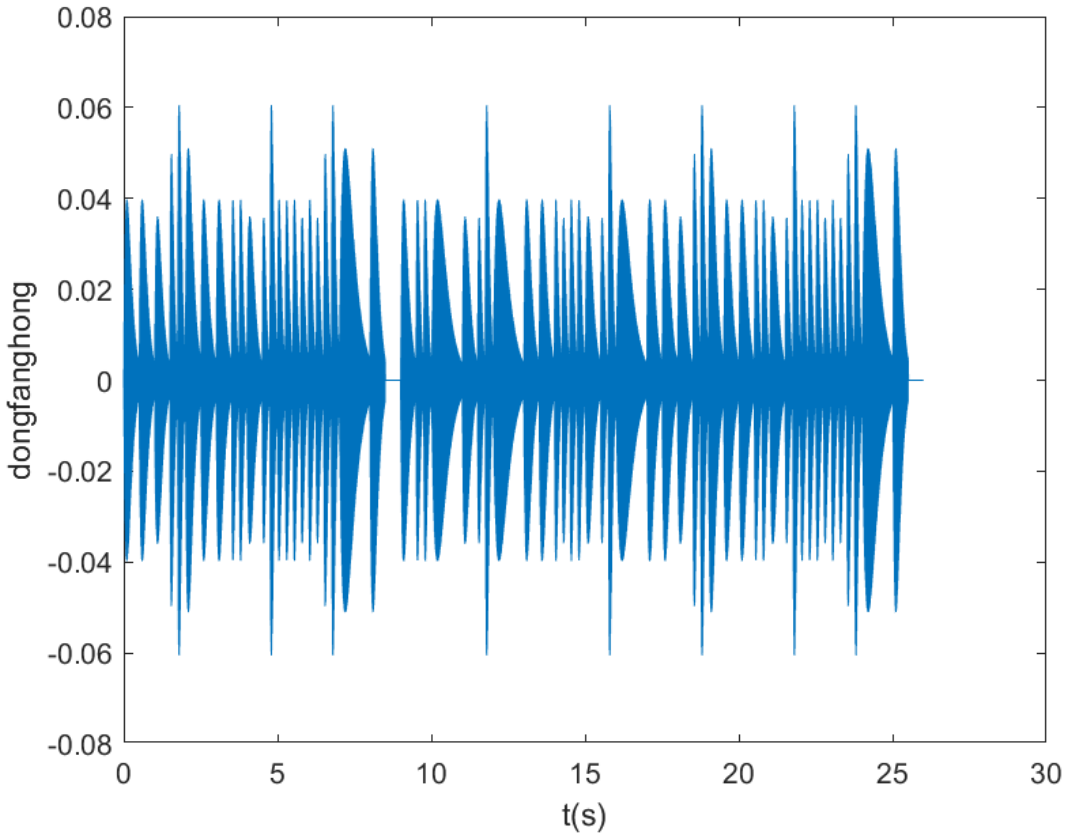


图 14: 声音序列

[**结果分析**] 有了明显的拨弦音, 音色趋于吉他, 但由于发音过于纯净, 没有吉他弹奏的混叠交错之感。

## 1.12 图形界面

[**题目描述**] 现在只要你掌握了某乐器足够多的演奏资料，就可以合成出该乐器演奏的任何音乐，在学完本书后面内容之后，试着做一个图形界面把上述功能封装起来。

[**主体思想**] 使用 guide 设计一款自动播放音乐的图形界面。

[**核心代码**] 代码太长，这里就不粘贴了，详见 code/commusic.m,code/soundmusic.m。

[**核心代码说明**] 主要是 gui 设计，音乐部分在前 11 题已经处理的较为完善。

[**运行结果**]

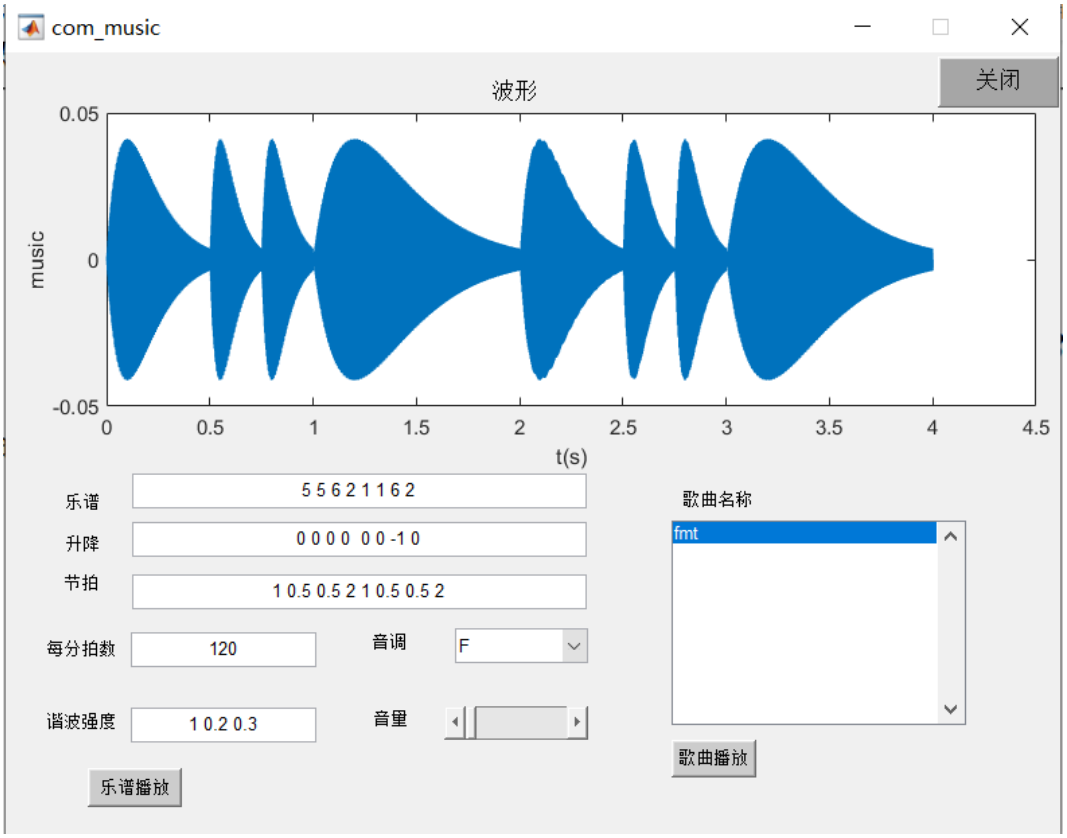


图 15: 图形界面实例

[**结果分析**] 图形界面较为简约。

## 1.13 写在最后

1. 经过反复的参数调整，最后发现：包络决定音色，谐波能带来拨弦颤动之音，但似乎对音色影响没有包络的影响那么大。2. 应当增加叠音接口，更接近真实。

3. 应当增加和弦接口，同时两个音符发声。

4. 自动分析音调过程中，标定各音开始时刻必不可少，若少了这一步，会对分析结果造成很大影响。

5. 按音匹配谐波分量过程不够自动化。

6. 在一首歌曲中, 节拍是有强弱之分的, 一半每小节的第一拍为强拍, 最后一拍为弱拍, 应当增加这个功能, 使歌曲更加有起伏感。