音乐合成大作业

*** 王禹 无 48 2014011241 ***

原创性声明

此代码和实验报告为本人原创，仅在第二部分的预处理部分，参看了李思涵同学的实验报告，以理解处理的过程，特此声明。

I. 简单的音乐合成

1. 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?

   F 大调下的各唱名频率表如 **Table 1-1** 所示，而针对钢琴则有各音名与频率的对照表格 **Table 1-2**

   Table 1-1 F 大调各唱名频率表

1	**2**	**3**	**4**	**5**	**6**	**7**
F	G	A	bB	C	D	R
349.23Hz	392Hz	440Hz	466.16Hz	523.25Hz	587.33Hz	659.25Hz

   Table 1-2 钢琴音名与频率对照表

声学用音名	**音乐用音名**	**频率（Hz）**	**声学用音名**	**音乐用音名**	**频率（Hz）**
A0	A2	27.5	F4	f1	349.23
B0	B2	30.87	G4	g1	392
C1	C1	32.7	A4	a1	440
D1	D1	36.71	B4	b1	493.38
E1	E1	41.2	C5	c2	523.3
F1	F1	43.65	D5	d2	587.33

G1	G1	49	E5	e2	659.26
A1	A1	55	F5	f2	698.46
B1	B1	61.74	G5	g2	783.99
C2	C	65.41	A5	a2	880
D2	D	73.42	B5	b2	987.77
E2	E	82.41	C6	c3	1047
F2	F	87.31	D5	d3	1174.7
G2	G	98	E6	e3	1318.5
A2	A	110	F6	f3	1396.9
B2	B	123.47	G6	g3	1568
C3	C	130.8	A6	a3	1760
D3	D	146.83	B6	b3	1975.5
E3	E	164.81	C7	c4	2093
F3	F	174.61	D7	d4	2349.3
G3	G	196	E7	e4	2637
A3	A	220	F7	f4	2793.8
B3	B	246.94	G7	g4	3136
C4	c1	261.6	A7	a4	3520
D4	d1	293.66	B7	b4	3951.1
E4	e1	329.63	C8	c5	4186

本实验中采用的是标准钢琴的音准符号，即小字组为 C4-B5。

本题较为简单，直接选取 F 大调下对应音调的频率，然后根据节奏写入信号向量中，确定 8kHz 的采样率后，再使用 sound 函数直接播放出来即可。

本题的代码在注释中，由于太过简单被后续的问题覆盖。

2. 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 1.5 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 1。

使用包络对音量变化进行修饰，这里我们使用的包络如 **Figure 2-1** 所示，而该包络形状下的声音信号如 **Figure 2-2 表示。**

其中生成该包络模型需要的参数有：

- 各个阶段的时间比例
- 峰值音量水平

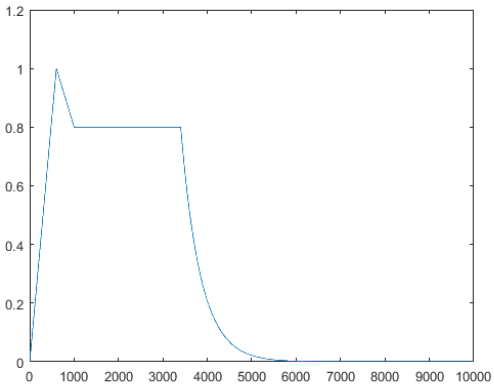

Figure 2-1 包络形状

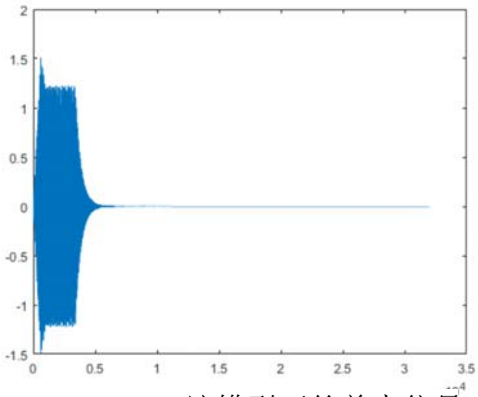

Figure 2-3 该模型下的单音信号

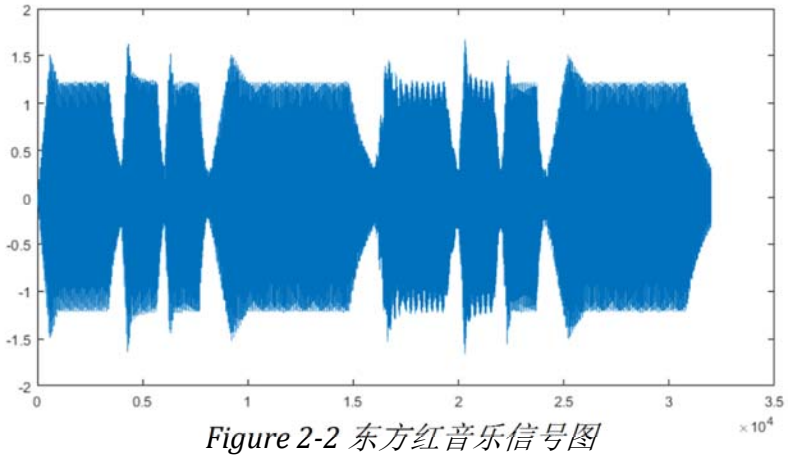

Figure 2-2 东方红音乐信号图

- 持续音量水平
- 衰减指数参数

具体的生成模型的程序见程序清单 **Code List 2-1**。

Code List 2-2-1

refine_toneshape.m

```
function signal_AP = refine_toneshape(t, t_start, duration,
tonefre ,signal)
    if nargin ==5
        harmonic = [0.2 0.3 0.2 0.15 0.2 0.25 0.2];
    end
    % Init
    t_local = t - t_start;
    signal_AP = zeros(size(signal));

    % Parameter
    implus_ratio = 0.15;
    decay_ratio = 0.1;
```

```
    stay_ratio = 0.6;
    peak_level = 1;
    stay_level = 0.8;
    dissolve_level = 9;

    % Time end
    implus_end = duration * implus_ratio;
    decay_end = implus_end + duration * decay_ratio;
    stay_end = decay_end + duration * stay_ratio;

    % Time inversal
    implus = (t_local >= 0 & t_local <implus_end);
    decay = (t_local >= implus_end & t_local < decay_end);
    stay = (t_local >= decay_end & t_local < stay_end);
    dissolve = (t_local >= stay_end);

    % Process
    signal_AP(implus)=linspace(0, peak_level, sum(implus));
    signal_AP(decay)=linspace(peak_level, stay_level, sum(decay));
    signal_AP(stay)=stay_level;
    signal_AP(decay)=linspace(peak_level, stay_level, sum(decay));
    signal_AP(dissolve)=stay_level*exp(dissolve_level * (stay_end -
t_local(dissolve)) / duration);

%     plot(1:length(signal_AP),signal_AP);
%     set(gca,'XLim',[0 10000]);
%     set(gca,'YLim',[0 1.2]);

    % Add Harmonic and Get wave (contain f 2f 3f)
    temp = 0;
    for i = 1:length(harmonic)
        temp = temp + sin(2 * (i + 1) * pi * tonefre * t) * harmonic(i);
    end
    temp = temp + sin(2 * pi * tonefre * t);
    signal_AP = signal_AP .* temp + signal;
    % plot(1:length(signal_AP),signal_AP);
End
```

上述代码的注释已经很明确的将每一块代码的意图表达清楚，故在此不再赘述算法的详细运行步骤，

最终生成的东方红歌曲的声音波形如图 Figure 2-3 所示。

3. 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）。

   最为直接的升高和降低一个八度的做法为直接将播放时的采样率变为原来的 2 倍或 1/2 倍，即以两倍速或半倍速播放；而要想提高 半个音阶，则使用 resample 函数进行适当的差值，从而升高半个音阶。

具体的代码见下 **Code List 3-1**

Code List 3-1

simple_resample.m

```
f_sample = 8e3;

tone = [5, 5, 6, 2, 1, 1, 6, 2];    % Tone
rising = [0, 0, 0, 0, 0, 0, -1, 0]; % Rising add 1 each eight degrees
beat = [1,0.5,0.5,2,1,0.5,0.5,2];   % Beats

music = make_music(f_sample, tone, beat, rising);
sound(music,f_sample);
pause(5);
sound(music,2 * f_sample);
pause(3);
sound(music,0.5 * f_sample);
```

Code List 3-2

use_funct_resample.m

```
f_sample = 8e3;

tone = [5, 5, 6, 2, 1, 1, 6, 2];
rising = [0, 0, 0, 0, 0, 0, -1, 0];
beat = [1,0.5,0.5,2,1,0.5,0.5,2];

[Q, P] = rat(2^(1/12), 10^(-9));  % Find the rational approximation.
music = make_music(f_sample, tone, beat, rising);
sound(music, f_sample);
% sound(resample(music, P, Q), f_sample);
```

需要指出的是，不管使用何种方式进行音调的改变，每一拍的时间长度均会产生变化，从而导致整首歌曲的时间发生了变化。

4. 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，听起来像不像风琴？）

在第二点中的 **Code List 2-1** 中，已经将谐波分量添加到音乐中了。在这里仅将谐波分量的代码摘录出来，这部分代码见 **Code List4-1**。

Code List 4-1

Part of refine_toneshape.m

```
% Add Harmonic and Get wave (contain f 2f 3f)
    temp = 0;
    for i = 1:length(harmonic)
        temp = temp + sin(2 * (i + 1) * pi * tonefre * t) * harmonic(i);
    end
    temp = temp + sin(2 * pi * tonefre * t);
    signal_AP = signal_AP .* temp + signal;
```

5. 自选其它音乐合成，例如贝多芬第五交响乐的开头两小节。

只需直接将最顶层的 m 文件中的 tone,rising 和 beat 改为对应歌曲的简谱即可。

如程序清单 **Code List5-1** 所示，演奏的为歌曲《月光》的第一句话。

Code List 5-1

simple_resample.m

```
f_sample = 8e3;

tone = [6, 3, 7, 7, 5, 6, 1, 7, 6, 5, 5, 6, 2, 3];
rising = [0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0];
beat = [0.5,0.25,1.25,0.5,0.25,1.25,0.5,0.75,0.5,0.25,0.25,0.5,0.25,1];

music = make_music(f_sample, tone, beat, rising);
sound(music,f_sample);
pause(5);
sound(music,2 * f_sample);
pause(3);
sound(music,0.5 * f_sample);
```

II. 简单的音乐合成

6. 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？

由于 wavread 函数在 MATLAB 2015a 及之后的版本被取消，故采用的较新的函数 audioread 函数。代码见下代码清单 **Code List 6-1.**

Code List 6-1

LoadMusic.m

```
fmt = audioread('fmt.wav');
sound(fmt,8000);
```

7. 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。

根据观察可以发现，wave2proc 大约包含 10 个完整地周期，为了去除非线性噪声，我们可以取 10 个周期的均值，从而抵消热噪声的影响。而由于原采样点仅有 243 个，不是 10 的倍数，不便于取平均。因此使用 resample 函数 10 倍插值采样取点，取得平均后，复制平均后的波形为 10 个，然后降低采样率为原来的 1/10。代码见下代码清单 **Code List 7-1** 和 **Code List 7-2.**

Code List 7-1

PreProcessScript.m

```
clear all;
load Guitar.MAT;

theorypreprocess = PreProcess(realwave, 10);
residual = norm(theorypreprocess - wave2proc)
```

Code List 7-2

PreProcessScript.m

```
function preproceeded = PreProcess( realwave, times )
    % 10倍插值
    ten_times = resample(realwave, times, 1);
    % 取平均降噪
    noise_reduction = mean(reshape(ten_times, length(realwave),
times), 2);
    % 恢复原10周期的波形
    preproceeded =
resample(repmat(noise_reduction,times,1),1,times);
end
```

原始波形如 **Figure7-1** 所示，而使用上述函数得到的波形如 **Figure7-1** 所示。

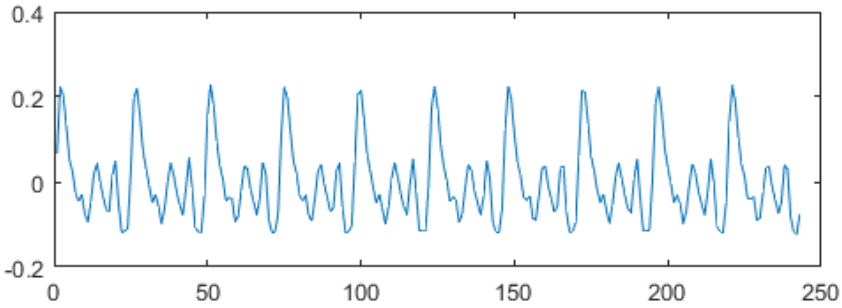

Figure 7-1 wave2proc 原始波形

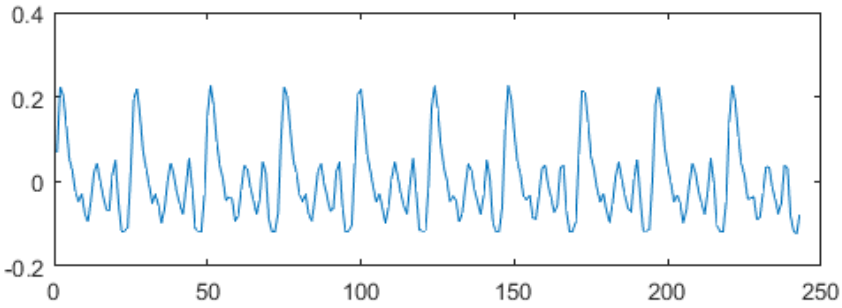

Figure 7-2 PreProcess 波形

8. 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。

将音频信号进行 fft 操作，得到这段声音的频谱。为了得到这段音乐的基频，可以先选择能量最高的信号作为当前基频，分别查看这个信号 1/2，1/3，1/4 到 1/16 频率处附近是否有超过阈值的较高能量，如果有则成为新的基频。重复上述过程直至找不到更小的频率。则此时的频率为基频。具体的算法实现可以见程序清单 **Code List8-1** 和 **Code List8-2**

Code List 8-1

GetBaseScript.m

```
clear all; close all;
load Guitar.MAT;

f_sample = 8e3;
permit_error = 0.1;
[ tone, basefre, harmonic ] = GetBaseandHarmonic( f_sample,
wave2proc, permit_error );
disp('Tone:');
disp(tone);
disp('Frequency:');
disp(basefre);
disp('Harmonic Coeff:');
disp(harmonic);

```

Code List 8-2

GetBaseandHarmonic.m

```
function [ tone, basefre, harmonicoeff ] =
GetBaseandHarmonic( f_sample, wave2proc, permit_error,
subplotpar, figureno, plotpar)
%GETBASEFRETONE Summary of this function goes here
    if (nargin==3)
        plotpar = 1;
        subplotpar = [1;1;1];
        figureno = 1;
    else
        if nargin == 5
            plotpar = 1;
        end
    end

    % Get tones data
    load tones_data.mat;

    % Get Frequency Spectrum
    signal = repmat(wave2proc, [8, 1]);
    F = abs(fft(signal));
    len = length(signal);
    f = ((0:len-1) /len * f_sample)';
    if(plotpar)
        figure(figureno);
        subplot(subplotpar(1),subplotpar(2),subplotpar(3));
        plot(f,F);
    end

    % Get the first Maxium
    [localmax_val, localmax_pos] = max(F);
    F(F<0.1 * localmax_val) = 0;
    basefre = f(localmax_pos-1);
    basecoeff = localmax_val;
    % Get the Basefre
    for i = 2:16
        [tempmax_val, tempmax_pos] =
maxium_exist_nearby((localmax_pos-1)/i, F, permit_error);
        if(tempmax_val > 0.5 * localmax_val)
            basefre = f(tempmax_pos-1);
            basecoeff = tempmax_val;
        end
    end

    % Get the tone by basefre
    [tone, basefre] = find_tone(basefre);

    % Get harmonic
    for i=2:4
        temp(i-1) = maxium_exist_nearby(round(basefre*i/f(2)),
F, 0.1)/basecoeff;
    end
```

```
50
51      harmonicoeff = temp;
```

9. 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。

   本问个人感觉是整个大作业中最难的部分，要求全自动的分析出所有的节奏。

   这里我详细说明一下我的代码，我首先做的是寻找包络，每 150 个点我划分成一个片断，取每个片断的最大值为该片段的采样点，即包络上的点。取完包络后的信号图如 **Figure 9-1 左图**所示。

   取完包络后便开始进行音符的划分。由于吉他波形的特性，其音符的划分应当是在极速上升的顶点处。因此我选取连续六个采样点，计算前五个采样点的均值与第六个点的值作比较，若是比值大于我设定的阈值，则可以视为是音频分割点。经过这样的处理之后，可以获得数目较少的音频分割点。之后遍历分割点，由于我认为乐谱中最短的音符为 1/8 音符，因此低于 1/2 个 0.5s 的间隔的音频分割点将被做一个合并处理，幅值较低的点将被合并到幅值较高的一个点去。经过这样的处理之后的幸好图如 **Figure9-1 右图**所示。

   取得分段后，再将每一小段用 8 中的算法进行频率的分析，获得基频与高次谐波分量的值。每一小段的时域与频域的波形如 **Figure9-2** 与 **Figure9-3 所示。**

   从此该问的分析过程完成，将数据存入 data.txt 中。data.txt 中的数据如 **Table9-1** 中所示。

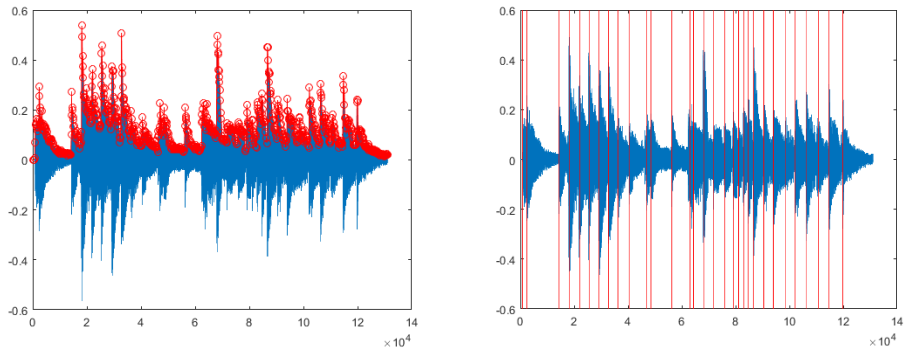

Figure 9-1 采样后的信号（左）与划分出的片断（右）

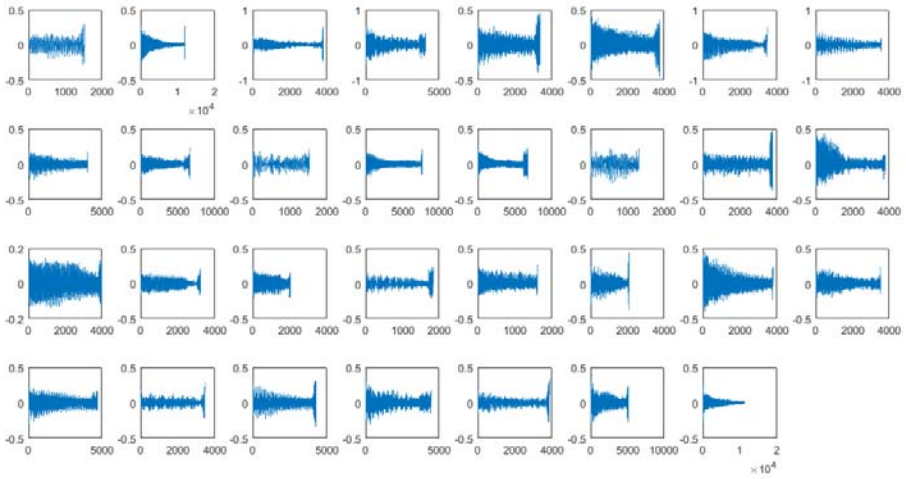

Figure 9-2　　每一小段的时域波形

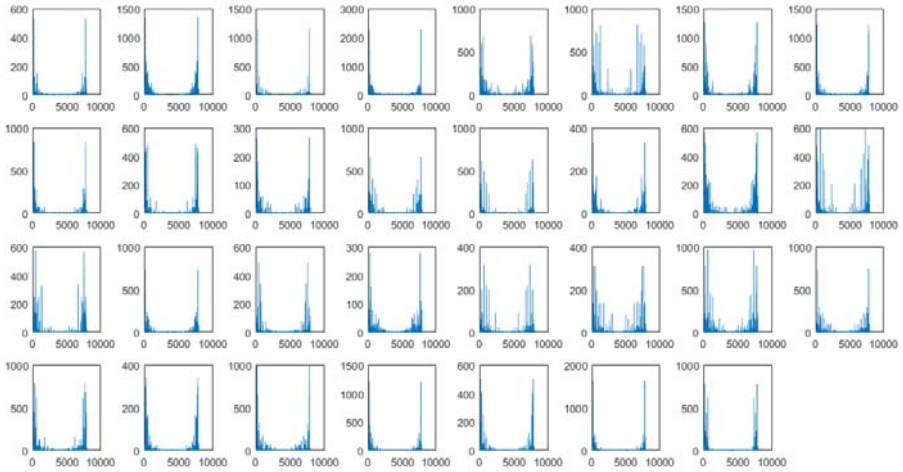

Figure 9-3　　每一小段的频域波形

Table 9-1　　data.txt 数据集表

声学用音名	基频频率	二次谐波频率	三次谐波频率	四次谐波频率
A3	218.241	0	0.291373	0
A3	221.877	0.268365	0.148425	0
B3	247.377	0.281712	0.112727	0
A3	221.486	0.157466	0.135241	0

D4	295.733	1.15976	0.30382	0.150995
E4	328.56	1.2671	1.06385	1.42714
G3	195.508	0.692203	0.442121	0.104882
F3	174.217	0.257668	0	0.148819
F3	174.182	0.270406	0	0
D4	294.991	1.06088	0.236322	0
G#3	208.361	0.150377	0.220769	0.191999
B3	247.496	0.337939	0.164084	0.445849
E4	329.299	0.807464	0.579513	0.380663
A3	225.019	0.262114	0.522441	0
A3	220.366	0.482576	0.355619	0.108374
A3	219.667	0.156883	1.26074	0
A4	441.516	0.438571	0.581944	0
A3	219.852	0.326351	0.205998	0
G4	393.596	0.440834	0.10274	0.162641
F4	350.678	0.278284	0.113415	0.168139
E4	327.533	1.57579	1.09671	1.00893
D4	294.399	0.99004	0.236559	0
E5	656.894	0.437329	0	0
B3	247.536	0.402731	0.235914	0.29452
D3	146.357	1.76113	0.593854	1.36671
C4	260.306	0.484272	0.200368	0.117548
F3	174.005	0.234089	0.324946	0.126838
A3	220.949	0.253117	0.177299	0
A3	220.816	0	0	0.254118
A3	221.291	0.230628	0.106103	0

Code List 9-1

GetBaseandHarmonic.m

```
clear all;close all;

% Parameters
f_sample = 8e3;
divede = 150;
diff_t = 1.35;
min_len = 0.15;
permit_error = 0.05;

% Music Signal
fmt = audioread('fmt.wav');
lengthNum = ceil(size(fmt,1)/divede);
fmt_resize = [fmt; zeros(lengthNum * divede-size(fmt,1),1)];
max_sample.value = zeros(lengthNum, 1);
max_sample.position = zeros(lengthNum, 1);
for i=1:lengthNum
    [value, position] = max(fmt_resize((i - 1)*divede +
1:i*divede,1));
    max_sample.value(i) = value;
    max_sample.position(i) = position + divede * (i - 1);
end
figure(1)
plot(1:length(fmt_resize),fmt_resize);
hold on;
plot(max_sample.position,max_sample.value,'r-o');
hold off;

% Divide to small divisions
n = 0;
for i=6:lengthNum
    temp = max_sample.value(i)/mean(max_sample.value(i-5:i-1));
    if(temp > diff_t)
        n = n + 1;
        result_pos(n) = max_sample.position(i);
        result_val(n) = max_sample.value(i);
    end
end

% Get the beats point
j = 2;
while(j<length(result_pos))
    if(result_pos(j+1)-result_pos(j) < min_len * f_sample);
        if(result_val(j+1)<= result_val(j))
            result_pos(j+1) = [];
            result_val(j+1) = [];
        else
            result_pos(j) = [];
            result_val(j) = [];
        end
    else
        j = j + 1;
    end;
```

```
end
figure(2)
plot(1:length(fmt_resize),fmt_resize);
hold on;
for i=1:length(result_pos)
    plot([result_pos(i) result_pos(i)],[-0.6 0.6],'r');
    hold on;
end
hold off;

% Ananlyze the frequency
for i=1:length(result_pos)
    if i==length(result_pos)
        aimwav = fmt(result_pos(i):length(fmt));
    else
        aimwav = fmt(result_pos(i):result_pos(i + 1));
    end
    figure(3);
    subplot(4,8,i);
    plot(1:length(aimwav),aimwav);

%     signal = aimwav;
%     F = abs(fft(signal));
%     len = length(signal);
%     f = ((0:len-1) /len * f_sample)';
%     figure(4);
%     subplot(4,8,i);
%     plot(f,F);

    [tone(i), basefre(i), tempharmonicfre] =
GetBaseandHarmonic( f_sample, aimwav, permit_error, [4;8;i],
4);
    harmonicfre(i) = {tempharmonicfre};
end

% Get the beats
for i=1:length(result_pos)-1
    beat(i) = round((result_pos(i+1)-result_pos(i))/(0.5 * 0.5
* f_sample));
end
beat(length(result_pos)) = round((length(fmt)-
result_pos(length(result_pos)))/(0.5 * 0.5 * f_sample));
beat = beat /2;
save tone_base_harmonic.mat tone basefre harmonicfre;

% Save file
 fid = fopen('data.txt','wt');
 for i=1:length(result_pos)-1
    fprintf(fid,'%s\t%g\t',tone{i}, basefre(i));
    fprintf(fid,'%g\t',harmonicfre{i});
    fprintf(fid,'\n');
 end
 fclose(fid);
```

III. 简单的音乐合成

10. 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?

即使用的高次谐波参数为：[1.4572 0.9587 1.0999 0]([2 次 3 次 4 次 5 次])，并应当对包络形状进行一定的修改，使其与吉他产生的声音包络较为相似。新的包络的代码详见代码清单 **Code List 10-1**

Code List 10-1

refine_guitar_toneshape.m

```
function signal_AP = refine_guitar_toneshape(t, t_start,
duration, tonefre ,signal, harmonic)
    if nargin ==5
        harmonic = [0.2 0.3 0.2 0.15];
    end
    % Init
    t_local = t - t_start;
    signal_AP = zeros(size(signal));

    % Parameter
    implus_ratio = 0.01;
    decay_ratio = 0.08;
    stay_ratio = 0.1;
    peak_level = 1;
    stay_level = 0.6;
    dissolve_level = 4;
    decay_level = 3;

    % Time end
    implus_end = duration * implus_ratio;
    decay_end = implus_end + duration * decay_ratio;
    stay_end = decay_end + duration * stay_ratio;

    % Time inversal
    implus = (t_local >= 0 & t_local <implus_end);
    decay = (t_local >= implus_end & t_local < decay_end);
    stay = (t_local >= decay_end & t_local < stay_end);
    dissolve = (t_local >= stay_end);

    % Process
    signal_AP(implus)=linspace(0, peak_level, sum(implus));
    signal_AP(decay)=peak_level*exp(decay_level * (stay_end -
t_local(decay)) / duration);
    signal_AP(stay)=stay_level;
    signal_AP(decay)=linspace(peak_level, stay_level,
sum(decay));
    signal_AP(dissolve)=stay_level*exp(dissolve_level *
(stay_end - t_local(dissolve)) / duration);

```

```
    % Get wave (contain f 2f 3f)
    temp = 0;
    for i = 1:length(harmonic)
        temp = temp + sin(2 * (i + 1) * pi * tonefre * t) *
harmonic(i);
    end
    temp = temp + sin (2 * pi * tonefre * t);
    signal_AP = signal_AP .* temp + signal;

    plot(1:length(signal_AP),signal_AP);
end
```

修改后的包络形状应当如 **Figure 10-1** 所示，而完整的东方红的歌曲的信号图如 **Figure 10-2** 所示。

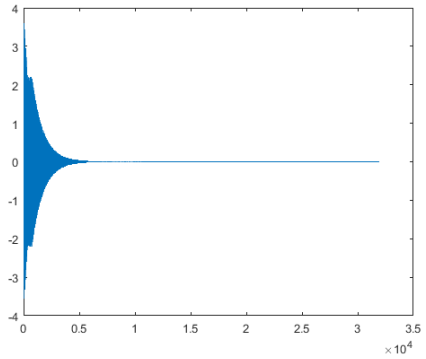

Figure 10-1 包络模型

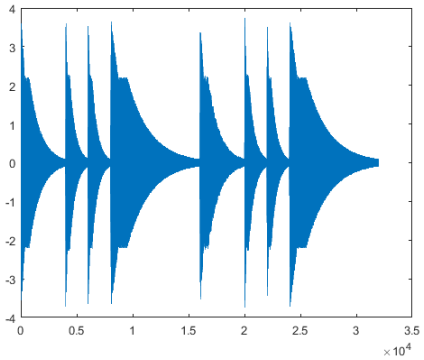

Figure 10-2 《东方红》的信号图

而大部分代码与第一大题是共用的，只有 make_music 不同。关键代码详见代码清单 **Code List 10-2**。

Code List 10-2

make_music_sameharmon.m

```
function music = make_music_sameharmon(f_sample, tones, beat,
rising)
    load tone_base_harmonic.mat

    beats_time = 0.5;

    % Find fre of tone
    fre = [349.23,392,440,466.16,523.25,587.33,659.25];
    tone2fre = zeros(1,length(tones));
    for i=1:length(tones)
        tone2fre(i) = fre(tones(i))* 2^(rising(i));
    end

    standard_time = linspace(0, 4*(1-1/f_sample),4*f_sample);
```

```
    music=zeros(size(standard_time));

    t_start = 0;

    % make music
    for i=1:length(tone2fre)
        music = refine_guitar_toneshape(standard_time, t_start,
beats_time*beat(i), tone2fre(i) ,music, ...
            [1.4572 0.9587 1.0999 0 0]);
        t_start = t_start + beats_time*beat(i);
    end

    % Make sure the maxium is lower than 1
    music = music/max(music);
```

即每一个音调的高次谐波的系数是相同的，将相同的系数传入 refine_toneshape 的参数，并将 refine_toneshape 更改为 refine_guitar_toneshape，获得的新的音乐声音与吉他声音较为类似，但是和真实的吉他声音有较大的区别。

11. 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。

即在 10 问的基础上，每次根据传入不同的高次谐波的传入相应的系数，系数由第 9 问中得到，被保存在 tone_base_harmonic.mat 中。具体的代码详见代码清单 **Code List11-1**。

Code List 11-1

make_music_diffharmon.m

```
function music = make_music_diffharmon(f_sample, tones, beat,
rising, major)
    load tone_base_harmonic.mat
    load tones_data.mat
    beats_time = 0.5;

    for i=1:length(tones_names)
        flag = false;
        tmp = [major '4'];
        if(length(tones_names{i})==length(tmp))
            flag = true;
            for j =1:length(tones_names{i})
                if(tones_names{i}(j)~=tmp(j))
                    flag = false;

```

```
                end
            end
        end
        if(flag)
            p = i;
            break;
        end
    end
    base = tones_fre(p);
    fre = base * (2 .^ ([0, 2, 4, 5, 7, 9, 11]/12));
    tone2fre = zeros(1,length(tones));
    for i=1:length(tones)
        tone2fre(i) = fre(tones(i))* 2^(rising(i));
    end

    standard_time = linspace(0, 4*(1-1/f_sample),4*f_sample);
    music=zeros(size(standard_time));

    t_start = 0;

    for i=1:length(tone2fre)
        [~, temp_pos] = min(abs(basefre-tone2fre(i)));
        music = refine_guitar_toneshape(standard_time, t_start,
beats_time*beat(i), tone2fre(i) ,music, ...
            harmonicfre{temp_pos});
        t_start = t_start + beats_time*beat(i);
    end

    % Make sure the maxium is lower than 1
    music = music/max(music);
```

通过将声音的频率与字典中的频率作比较，获得最接近的频率所对应的高次谐波的系数，从而传入 make_guitar_toneshape。

获得的音乐听起来更像是吉他的声音，但是和真实的声音还是有所区别，应该是包络的形状选择有关。

12. 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。

用 MATLAB 封装了一个图形界面（**Figure 12-1**），本图形界面只允许输入大调调性。在音调处输入简谱唱名，在节拍处输入每个音符唱名的节拍，在升降调处输入升高/降低几个八度。

点击“播放”后，右侧会显示出波形，并播放声音；点击“清空”后，会将所有输入框清零。

具体实现代码见代码清单 **Code List 12-1**

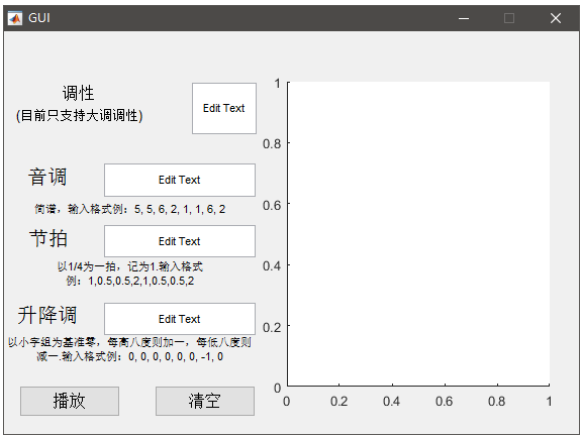

Figure 12-1　图形界面

Code List 12-1

Part of GUI.m

```
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)
major = char(get(handles.edit1,'string'));
tones = str2num(char(get(handles.edit2,'string')));
beat = str2num(char(get(handles.edit3,'string')));
rising = str2num(char(get(handles.edit4,'string')));
f_sample = 8e3;
try
music = make_music_diffharmon(f_sample, tones, beat, rising,
major);
plot((1:length(music))/f_sample,music)
sound(music,f_sample);
catch
    warndlg('²ÎÊýÊäÈë´íÎó£¡')
end


% --- Executes on button press in pushbutton1.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)
str_ = '';
set(handles.edit1,'string',str_)
set(handles.edit2,'string',str_)
set(handles.edit3,'string',str_)
```

```
30 set(handles.edit4,'string',str_)
31 plot(0)
```

说明，这里的 GUI.m 仅截取了实际编写部分，其余的固定生成部分未截取其中。

实验总结

本次实验总的来讲，跨度较大，在分割部分达到难度的顶峰，而第三问则较为简单。本实验对理解音乐合成与声音信号处理有着较大的帮助，设计十分合理与完善。