音乐合成大作业

无04 2019012137 张鸿琳

1.2.1 简单的合成音乐

(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?

《东方红》为F大调，由“十二平均律”可得，唱名1到7对应频率分别为349.23Hz、392Hz、440Hz、466.16Hz、523.25Hz、587.33Hz、659.25Hz，那么所给《东方红》片段的各个乐音的频率与时值依次为（按照四分音符组织）：

乐音	**5**	**5**	**6**	**2**	**1**	**1**	**6(降八度)**	**2**
频率(Hz)	523.25	523.25	587.33	392	349.23	349.23	293.66	392
时值(s)	0.5	0.25	0.25	1	0.5	0.25	0.25	1
相对F半音数	7	7	9	2	0	0	-3	2

编写代码，先对单个音进行播放，再连续播放该片段，代码如下(对应hw_1_2_1_1.m文件)：

```
clear all; close all; clc;

freq=8000;%采样频率

%确定每个音符的频率和时值
note1=generatenote(7,4);
note2=generatenote(7,8);
note3=generatenote(9,8);
note4=generatenote(2,2);
note5=generatenote(0,4);
note6=generatenote(0,8);
note7=generatenote(-3,8);
note8=generatenote(2,2);

%单独播放每个音符
sound(note1,freq);pause(1);
sound(note2,freq);pause(1);
sound(note3,freq);pause(1);
sound(note4,freq);pause(1);
sound(note5,freq);pause(1);
sound(note6,freq);pause(1);
sound(note7,freq);pause(1);
sound(note8,freq);pause(1);

```

```
%连续播放乐曲
note=[note1 note2 note3 note4 note5 note6 note7 note8];
sound(note,freq);

function note=generatenote(tune,time)%产生相应时长与频率的正弦信号
freq=8000;%采样频率
T=1/freq;%采样周期
beat=0.5;%一拍时间
const=2^(1/12);%紧邻音调之间倍数关系
baseF=349.23;%存储F调频率

time2=0:T:2*beat;%2分音符
time4=0:T:beat;%4分音符
time8=0:T:beat/2;%8分音符

if time==2
    note=sin(2*pi*baseF*const^tune*time2);
elseif time==4
        note=sin(2*pi*baseF*const^tune*time4);
elseif time==8
        note=sin(2*pi*baseF*const^tune*time8);
end
end
```

单独播放音调时，可以基本确认音调正确，而连续播放时，可以比较明显地听出《东方红》的曲调，但是音色艰涩难听，且有杂音。

(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。

为了消除相邻乐音之间因相位不连续带来的杂音，尝试增加包络，我首先尝试了线性包络，音色不佳，后根据老师建议改为了指数衰减的包络，编写代码如下(对应hw_1_2_1_2.m文件)：

```
clear all; close all; clc;

freq=8000;%采样频率

%确定每个音符的频率和时值
note1=generatenote(7,4);
note2=generatenote(7,8);
note3=generatenote(9,8);
note4=generatenote(2,2);
note5=generatenote(0,4);
note6=generatenote(0,8);
note7=generatenote(-3,8);
note8=generatenote(2,2);

%连续播放乐曲
note=[note1 note2 note3 note4 note5 note6 note7 note8];
sound(note,freq);

function note=generatenote(tune,time)%产生有包络的信号
freq=8000;%采样频率
T=1/freq;%采样周期
```

```
beat=0.5;%一拍时间
const=2^(1/12);%紧邻音调之间倍数关系
baseF=349.23;%存储F调频率

time2=0:T:2*beat;%2分音符
time4=0:T:beat;%4分音符
time8=0:T:beat/2;%8分音符

%产生包络信号
envelope2=generateenv(time2);
envelope4=generateenv(time4);
envelope8=generateenv(time8);

if time==2
    note=sin(2*pi*baseF*const^tune*time2).*envelope2;
elseif time==4
        note=sin(2*pi*baseF*const^tune*time4).*envelope4;
elseif time==8
        note=sin(2*pi*baseF*const^tune*time8).*envelope8;
end
end

function envelope=generateenv(t)%产生指数衰减包络信号
envelope=exp(-4*t/max(t));
end

function envelope=generateenv1(t)%产生分段线性包络信号，尝试后最终没有采用该包络
envelope=10/max(t)*(t<=max(t)*0.1&t>0).*t...
+5/max(t)*(t>max(t)*0.1&t<=max(t)*0.2).*(max(t)*0.3-t)...
+0.5*(t>max(t)*0.2&t<=max(t).*0.8)...
+2.5/max(t)*(t>max(t)*0.8&t<=max(t)).*(max(t)-t);
end
```

采用了指数衰减包络后，播放效果有了很大提升，杂音消失，且有一点弦乐的感觉。

(3) 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）

将音乐整体升高一个八度或降低一个八度其实就是将所有音调频率变为原来的2倍或1/2倍，可以通过直接改变播放时长来实现，只要改变sound函数的采样频率即可，而利用resample函数将音乐升高半个音阶，则是通过重新采样的方法来提升频率（在sound采样频率不变的情况下），假如按一定比例从原周期信号中剔除掉一部分，则也相当于原信号的播放时间减少，音调频率也就改变了。编写代码如下(对应hw_1_2_1_3.m文件):

```
clear all; close all; clc;

freq=8000;%采样频率

%确定每个音符的频率和时值
note1=generatenote(7,4);
note2=generatenote(7,8);
note3=generatenote(9,8);
note4=generatenote(2,2);
note5=generatenote(0,4);
```

```
note6=generatenote(0,8);
note7=generatenote(-3,8);
note8=generatenote(2,2);

%连续播放乐曲
note=[note1 note2 note3 note4 note5 note6 note7 note8];
sound(note,freq*2);%高八度
pause(3);
sound(note,freq/2);%低八度
pause(8);

note_new=resample(note,1000,1059);%升高半个音阶
sound(note_new,freq);

function note=generatenote(tune,time)%产生有包络的信号
freq=8000;%采样频率
T=1/freq;%采样周期
beat=0.5;%一拍时间
const=2^(1/12);%紧邻音调之间倍数关系
baseF=349.23;%存储F调频率

time2=0:T:2*beat;%2分音符
time4=0:T:beat;%4分音符
time8=0:T:beat/2;%8分音符

envelope2=generateenv(time2);
envelope4=generateenv(time4);
envelope8=generateenv(time8);

if time==2
    note=sin(2*pi*baseF*const^tune*time2).*envelope2;
elseif time==4
        note=sin(2*pi*baseF*const^tune*time4).*envelope4;
elseif time==8
        note=sin(2*pi*baseF*const^tune*time8).*envelope8;
end
end

function envelope=generateenv(t)%产生指数衰减包络信号
envelope=exp(-4*t/max(t));
end
```

播放后可听到音调确实整体发生了明显变化，播放时间也随之改变了。

(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，听起来像不像象风琴？）

为了在音乐中加入谐波分量，只要对原代码稍作修改即可，我按照老师建议，采用了1:0.2:0.3的谐波幅度比例，修改后代码如下(对应hw_1_2_1_4.m文件)：

```
clear all; close all; clc;

freq=8000;%采样频率

%确定每个音符的频率和时值
```

```
note1=generatenote(7,4,1)+0.2*generatenote(7,4,2)+0.3*generatenote(7,4,3);
note2=generatenote(7,8,1)+0.2*generatenote(7,8,2)+0.3*generatenote(7,8,3);
note3=generatenote(9,8,1)+0.2*generatenote(9,8,2)+0.3*generatenote(9,8,3);
note4=generatenote(2,2,1)+0.2*generatenote(2,2,2)+0.3*generatenote(2,2,3);
note5=generatenote(0,4,1)+0.2*generatenote(0,4,2)+0.3*generatenote(0,4,3);
note6=generatenote(0,8,1)+0.2*generatenote(0,8,2)+0.3*generatenote(0,8,3);
note7=generatenote(-3,8,1)+0.2*generatenote(-3,8,2)+0.3*generatenote(-3,8,3)
;
note8=generatenote(2,2,1)+0.2*generatenote(2,2,2)+0.3*generatenote(2,2,3);

%连续播放乐曲
note=[note1 note2 note3 note4 note5 note6 note7 note8];
sound(note,freq);

function note=generatenote(tune,time,n)%产生有包络的信号
freq=8000;%采样频率
T=1/freq;%采样周期
beat=0.5;%一拍时间
const=2^(1/12);%紧邻音调之间倍数关系
baseF=349.23;%存储F调频率

time2=0:T:2*beat;%2分音符
time4=0:T:beat;%4分音符
time8=0:T:beat/2;%8分音符

envelope2=generateenv(time2);
envelope4=generateenv(time4);
envelope8=generateenv(time8);

if time==2
    note=sin(n*2*pi*baseF*const^tune*time2).*envelope2;
elseif time==4
        note=sin(n*2*pi*baseF*const^tune*time4).*envelope4;
elseif time==8
        note=sin(n*2*pi*baseF*const^tune*time8).*envelope8;
end
end

function envelope=generateenv(t)%产生指数衰减包络信号
envelope=exp(-4*t/max(t));
end
```

运行代码可以听到音乐确实更具有“厚度”了，且确实有点像风琴。

(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。

为了能够更快捷地输入乐曲信息，我对原代码架构进行了重新整合，并输入了《欢乐颂》的前几个小节的乐曲信息，代码如下(对应hw_1_2_1_5.m文件)：

```
clear all; close all; clc;

%演奏欢乐颂前四个小节

freq=8000;%采样频率
singletime=0.5;%切换播放音符的时间

```

```
tune_info=[16,4;16,4;17,4;19,4;19,4;17,4;16,4;14,4;12,4;...
    12,4;14,4;16,4;16,8/3;14,4;14,2];%输入乐曲信息

song=generatesong(tune_info);%生成乐曲
playsong(song,freq,singletime,tune_info);%播放乐曲

function envelope=generateenv(t)%产生指数衰减包络信号
envelope=exp(-4*t/max(t));
end

function note=generatenote(tune,time)%产生有包络的信号
const=2^(1/12);%紧邻音调之间倍数关系
baseD=293.66;%存储D调频率

envelope=generateenv(time);

note=sin(2*pi*baseD*const^tune*time).*envelope...
    +0.2*sin(2*2*pi*baseD*const^tune*time).*envelope...
    +0.3*sin(3*2*pi*baseD*const^tune*time).*envelope;

end

function song=generatesong(tune_info)%生成乐曲
freq=8000;%采样频率
T=1/freq;%采样周期
beat=0.5;%一拍时间

song=zeros(size(tune_info,1),freq*beat*2);

for i=1:size(tune_info,1)
    t=0:T:beat*4/tune_info(i,2);
    temp=generatenote(tune_info(i,1),t);
    song(i,1:size(temp,2))=temp;
end
end

function playsong(song,freq,singletime,tune_info)%播放乐曲
freq=8000;%采样频率
T=1/freq;%采样周期
beat=0.5;%一拍时间
sound(zeros(8000,1),freq);%稍作等待
pause(1);
for i=1:size(song,1)
    sound(song(i,1:freq*beat*4/tune_info(i,2)),freq);
    pause(singletime);
end
end
```

运行代码，可以听到乐曲很流畅地播放。

1.2.2 用傅里叶级数分析音乐

(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?

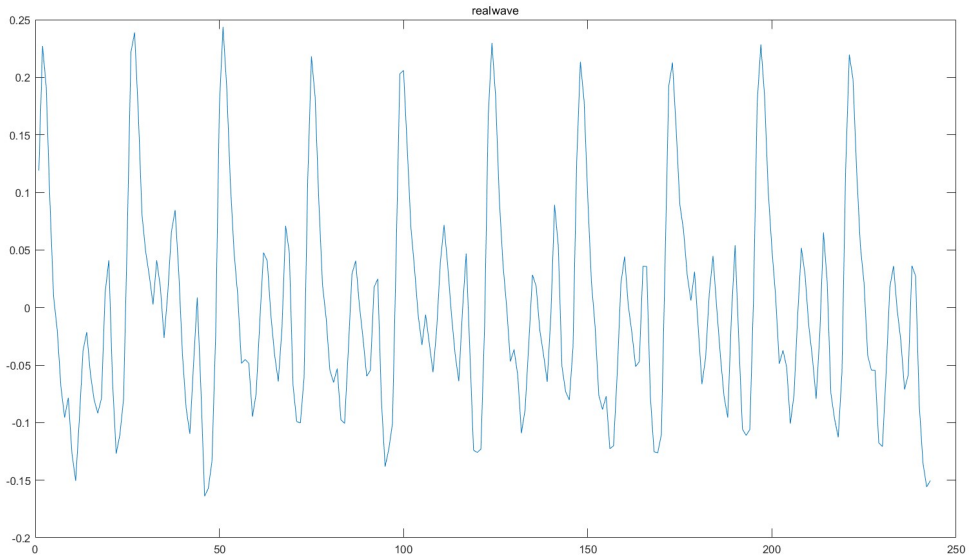
编写如下代码读取并播放音乐文件(对应hw_1_2_2_6.m文件):

```
clear all; close all; clc;

realsong=audioread("fmt.wav");%读取音乐文件
sound(realsong);%播放音乐
```

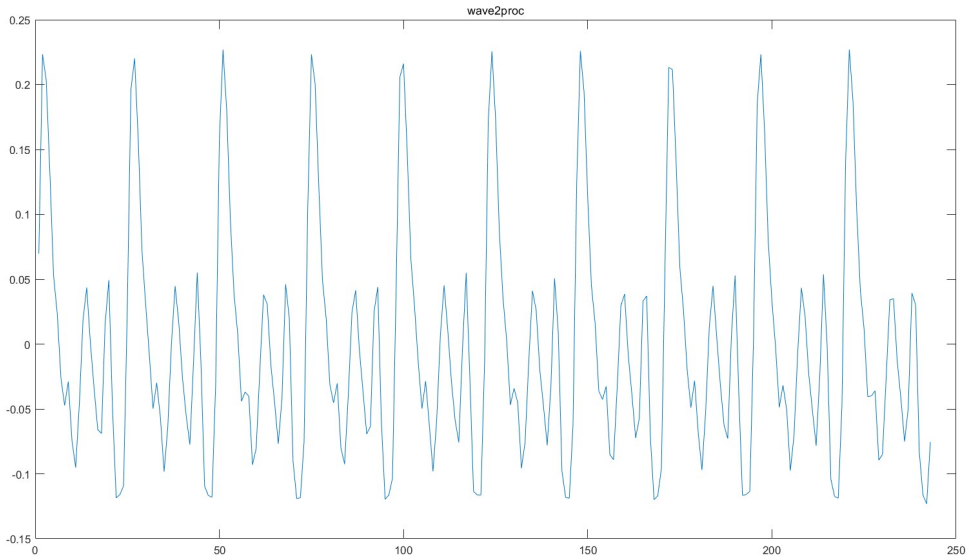
所播放的音乐显然比此前生成的乐曲真实多了。

(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。

首先，读取“guitar.mat”文件，并画出realwave的图像，如下：

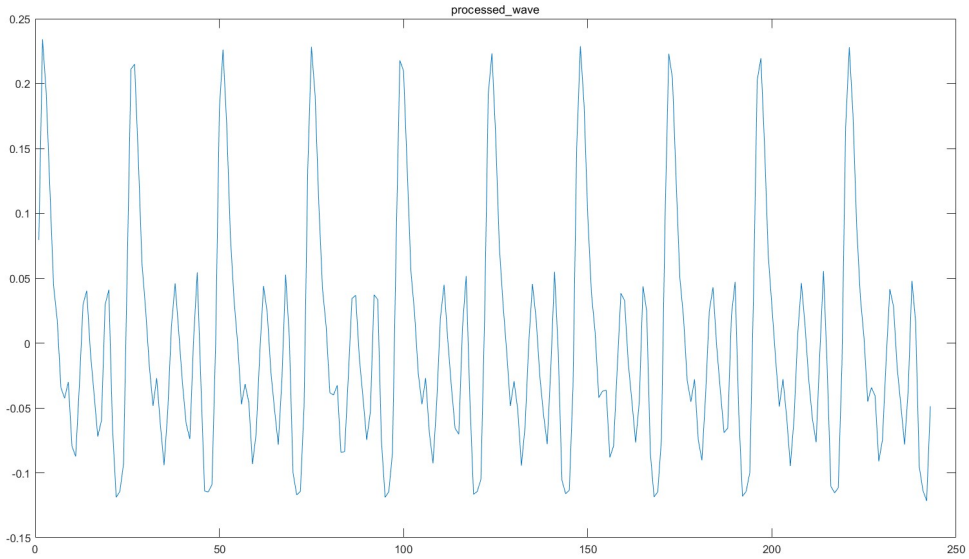


而wave2proc的图像如下：

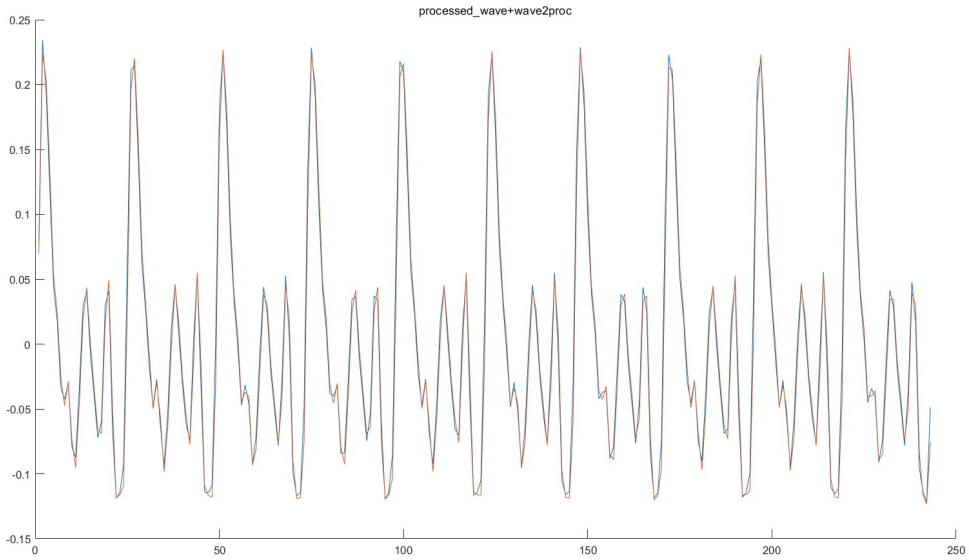


对比可以发现，wave2proc波形更近似于周期信号，大致可以猜测出wave2proc可能是由realwave对单个周期内的图像进行拆分取平均后再周期延拓多次得到的，这种操作有助于减少非线性谐波和噪声的干扰。为了验证这种猜想，我编写代码复现了这个过程，操作的难点在于确定出周期并找到各个单周期波形的交接点，我想到了《信号与系统》课程中讲授的匹配滤波算法，即可以通过寻找自相关函数的

峰值确定周期以及单周期波形的交接点。实现了上述操作后，得到由realwave处理后的波形如下：



将我自行处理得到的波形和wave2proc波形进行对比，如下图：



可以看到，二者几乎完全一致，也就验证了wave2proc确实是采用了这样的处理方法得到的。

该问题用到的代码如下(对应hw_1_2_2_7.m文件)：

```
clear all; close all; clc;

load("Guitar.MAT");%读取文件

figure(1);
plot(realwave);%绘制realwave图像
title("realwave");
figure(2);
plot(wave2proc);%绘制wave2proc图像
title("wave2proc");

%通过取平均消除非线性谐波和噪声
%为了让结果更精准，先通过重新取样增加点的密度
res_wave=resample(realwave,100,1);
```

```
temp=zeros(1,size(res_wave,1));%暂存自相关数据

%计算(自相关函数)/(偏移后自身重叠区间长度)，便于确定峰值
for i=1:size(res_wave,1)
    temp(1,i)=sum(res_wave(i:size(res_wave,1)).*...
        res_wave(1:size(res_wave,1)-i+1))/(size(res_wave,1)-i+1);
end

peak=zeros(1,10);
k=1;

%找到(自相关函数)/(偏移后自身重叠区间长度)的极值点，确定周期
for i=2:size(temp,2)-1
    %不是所有极值点都是周期交界点，其需要大于某一值
    if(temp(1,i)>0.006 && temp(1,i)>=temp(1,i-1) && temp(1,i)>=temp(1,i+1))
        peak(1,k)=i;
        k=k+1;
    end
end

%根据所得周期交界点，将原信号拆分为多个单周期信号求平均
len=2430;
aver=res_wave(1:len);
for i=1:8
    aver=aver+res_wave(peak(1,i):peak(1,i)+len-1);
end
aver=aver/9;

%对所得平均信号进行周期延拓
processed_wave=[aver;aver;aver;aver;aver;aver;aver;aver;aver;aver];
processed_wave=resample(processed_wave,1,100);

%将利用上述操作得到的波形与wave2proc进行对比
figure(3);
plot(processed_wave);
title("processed\_wave");
figure(4);
hold on;
plot(processed_wave);
plot(wave2proc);
title("processed\_wave+wave2proc");
```

(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。

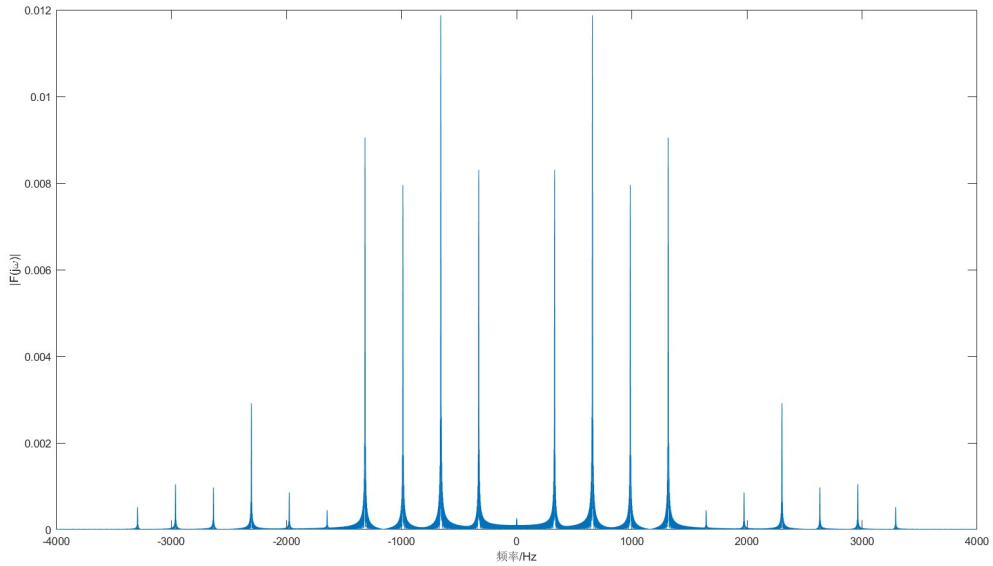
对处理得到的processed_wave（或wave2proc）直接进行傅里叶变换，会发现结果不能很好地近似于冲激函数，而是更接近于sinc函数，这是因为processed_wave（或wave2proc）相当于一个周期信号上乘上了一个一定范围的矩形信号（窗函数），窗函数的傅里叶变换为sinc函数形式，时域相乘则频域卷积，周期信号在频域表现为一系列冲激函数，故而二者卷积的结果就是很多平移后的sinc函数的叠加。为了让频域图像更接近于冲激函数，可以通过扩大窗函数时域展宽的方式来实现，因为窗函数时域展宽越大，则其频域展宽越小，其傅里叶变换结果也就越接近于冲激函数，而增大窗函数时域展宽，其

实也就是增多processed_wave（或wave2proc）时域图像中包含的周期数，可以通过把时域信号重复若干次来实现，这样效果就会好很多。按照这样的思路，编写代码如下(对应hw_1_2_2_8.m文件)：

```
clear all; close all; clc;

load("Guitar.MAT");%读取文件

%通过取平均消除非线性谐波和噪声
%为了让结果更精准，先通过重新取样增加点的密度
res_wave=resample(realwave,100,1);
temp=zeros(1,size(res_wave,1));%暂存自相关数据

%计算(自相关函数)/(偏移后自身重叠区间长度)，便于确定峰值
for i=1:size(res_wave,1)
    temp(1,i)=sum(res_wave(i:size(res_wave,1)).*...
        res_wave(1:size(res_wave,1)-i+1))/(size(res_wave,1)-i+1);
end

peak=zeros(1,10);
k=1;

%找到(自相关函数)/(偏移后自身重叠区间长度)的极值点，确定周期
for i=2:size(temp,2)-1
    %不是所有极值点都是周期交界点，其需要大于某一值
    if(temp(1,i)>0.006 && temp(1,i)>=temp(1,i-1) && temp(1,i)>=temp(1,i+1))
        peak(1,k)=i;
        k=k+1;
    end
end

%根据所得周期交界点，将原信号拆分为多个单周期信号求平均
len=2430;
aver=res_wave(1:len);
for i=1:8
    aver=aver+res_wave(peak(1,i):peak(1,i)+len-1);
end
aver=aver/9;

processed_wave=[aver;aver;aver;aver;aver;aver;aver;aver;aver;aver];
processed_wave=[processed_wave;processed_wave;processed_wave;...
                processed_wave;processed_wave;processed_wave;...
                processed_wave;processed_wave;processed_wave;...
                processed_wave];%重复10次便于进行傅里叶变换分析
processed_wave=resample(processed_wave,1,100);

%利用老师所给prefourier函数做傅里叶变换
Trg=[0,243/8000*10];
N=2430;
OMGrg=[-8000*pi,8000*pi];
K=20000;
[omg,FT]=prefourier(Trg,N,OMGrg,K);
F_info=FT*processed_wave;

%绘制频域图像
plot(omg/2/pi,abs(F_info));
xlabel("频率/Hz");
ylabel("|F(j\omega)|");
```

```
55
56  function [omg,FT]=prefourier(Trg,N,OMGrg,K)
57  % 输入参数：
58  % Trg  ：二维矢量，两个元素分别表示时域信号的起止时间；
59  % N    ：时域抽样数量；
60  % OMGrg：二维矢量，两个元素分别表示频谱的起止频率；
61  % K    ：频域抽样数量。
62  % 输出参数：
63  % omg  ：抽样频率；
64  % FT   ：实现傅里叶变换的矩阵~U~及系数；
65  T=Trg(2)-Trg(1);
66  t=linspace(Trg(1),Trg(2)-T/N,N)';
67  OMG=OMGrg(2)-OMGrg(1);
68  omg=linspace(OMGrg(1),OMGrg(2)-OMG/K,K)';
69  FT=T/N*exp(-1i*kron(omg,t.'));
70  end
```

得到processed_wave（或wave2proc）傅里叶变换频域图像如下：

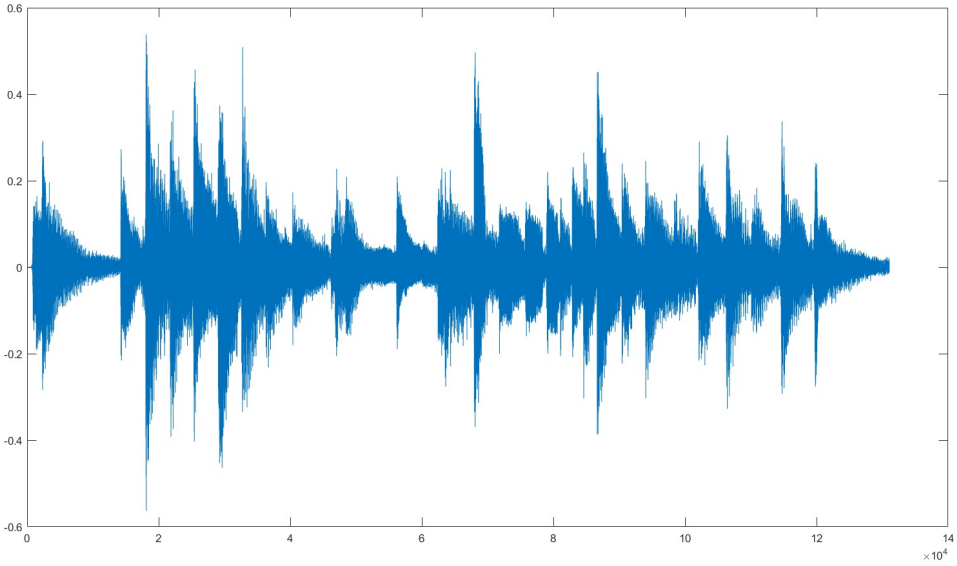


频域图像可以很好地近似为一系列冲激函数求和了。该段音乐的基频为329.2Hz，对应于E调，其基波与各个谐波分量的比例如下：

频率(Hz)	峰值	归一化峰值
329.2 (基波)	0.0083	1
658.4 (二次谐波)	0.0119	1.434
987.6 (三次谐波)	0.0080	0.964
1316.8 (四次谐波)	0.0091	1.096

(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。

这道题目涉及较多难点，首先要确定各个音符的起始与停止位置，而fmt.wav的波形如下：



通过观察，可以推断出，每个音符开始处都应为跳变处，按照这样的思路，只要找到跳变比较大的点位即可（可以通过比较某个点和其靠前一个区域内最大值的差异来判定其是否为一个跳变点，差异较大时则为一个新的音符的起始点）。

在确定音符的位置后，还需要判定音符的基波频率，也就是判断其音调。我的做法是将某一段音符取出后，利用老师所给的prefourier函数直接对其进行傅里叶变换，找到较大峰值对应的频率就是基波频率，然而我在观察部分音符的傅里叶变换图像后，发现有一些音符的谐波频率或其他一些高频分量的峰值是高于基波频率的峰值的，所以我进一步改进了策略，先找到傅里叶变换图像中的最大峰值对应的频率，再找相较于该频率更低范围内的较大值，若该较大值比乘以某倍数的全域最大值更大，则选取那个更低的频率作为基波频率。经测试，仅仅是这样处理还有可能有个别音调判定有明显问题，故而又加上了限定，使得基波频率不超过500Hz，且加上一个判断，使得假如某一频率存在较大的谐波分量，则更倾向于判定其为基波频率，这样判定效果会好一些。（为了减少衰减带来的影响，取了各个音符时域的靠前部分做傅里叶变换）编写代码如下(对应hw_1_2_2_9.m文件)：

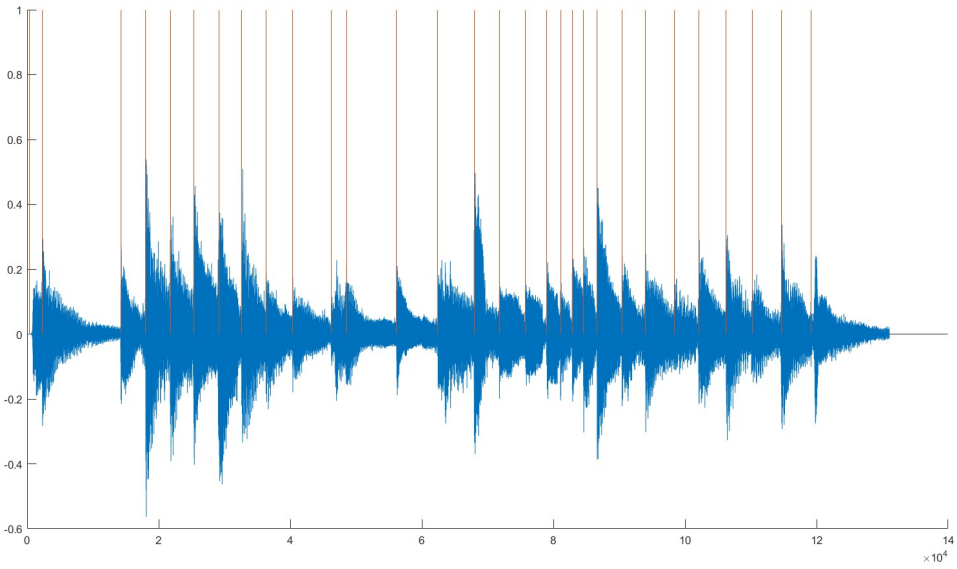
```
clear all; close all; clc;

[y,Fs]=audioread("fmt.wav");%读取文件

freq=8000;%采样频率

%通过峰值统计音符时值
time=zeros(50,1);%存储音符时值
pos=zeros(50,1);%存储音符起始点与终止点
pos(1)=352;%设定音符起始点
jump=0.25*8000;%各个音符时域的最小单位为0.25s
range=500;%临近音符允许的相交部分
range0=500;%确定和前面比较的范围，用于找跳变值
range1=300;%确定和前面比较的范围，用于找跳变值
ratio=1.5;%跳变程度
num=2;%存储音符数量
for i=201:size(y,1)
    if(i-pos(num-1,1)>jump-range)
        [temp,temp1]=max(y(i-range0:i-range1,1));
        if(y(i,1)>0 && y(i,1)/temp>ratio)
            pos(num,1)=i;
            time(num-1,1)=round((i-pos(num-1,1))/jump);
```

```
            num=num+1;
        end
    end
end
num=num-1;
time(num,1)=round((size(y,1)-pos(num-1,1))/jump);

%绘制各个音符对应的时域范围
figure(1);
hold on
plot(y);
a=pos(1:num,1);
b=ones(num,1);
stem(a,b,'Marker','none');

%通过prefourier函数确定各个音符的音调
tunes=zeros(num,1);%存储音符音调信息
pos(num+1,1)=size(y,1);
harmonic_info=zeros(num,3);%存储各个谐波分量归一化峰值信息
x=200;%用于取音符时域的中间部分
for i=1:num
    Trg=[0,(pos(i+1,1)-pos(i,1)+1-x)/8000];
    N=pos(i+1,1)-pos(i,1)+1-x;
    OMGrg=[-4000*pi,4000*pi];
    K=20000;
    [omg,FT]=prefourier(Trg,N,OMGrg,K);
    F_info=FT*y(pos(i,1):pos(i+1,1)-x,1);
    [tunes(i,1),harmonic_info(i,:)]=...
    gettune(omg/2/pi,abs(F_info(10001:20000,1)));
end

% tunes(26,1)=247;%可能需要对该音调进行修正

tune_info=analyze(time,tunes,num);%对得到的初步信息进行整合
% song=generatesong(tune_info,harmonic_info);%利用提取出的音调和时值信息生成曲子
% playsong(song,freq,tune_info);%播放曲子

function [omg,FT] = prefourier(Trg,N,OMGrg,K)
% 输入参数：
% Trg  : 二维矢量，两个元素分别表示时域信号的起止时间；
% N    : 时域抽样数量；
% OMGrg: 二维矢量，两个元素分别表示频谱的起止频率；
% K    : 频域抽样数量。
% 输出参数：
% omg  : 抽样频率；
% FT   : 实现傅里叶变换的矩阵~U~及系数；
T = Trg(2)-Trg(1);
t = linspace(Trg(1),Trg(2)-T/N,N)';
OMG = OMGrg(2)-OMGrg(1);
omg = linspace(OMGrg(1),OMGrg(2)-OMG/K,K)';
FT = T/N*exp(-1i*kron(omg,t.'));
end

function [tune,harmonic]=gettune(omg,F)%由傅里叶变换图像找基波和谐波信息
harmonic=zeros(1,3);%存储谐波信息
ratiox=0.71;
temp=zeros(4,1);
pos=zeros(4,1);
```

```
t=60;
[temp(1,1),pos(1,1)]=max(F);
if(omg(10000+pos(1,1),1)>500)
    [temp(1,1),pos(1,1)]=max(F(1:pos(1,1)-t,1));
end
if(omg(10000+pos(1,1),1)>500)
    [temp(1,1),pos(1,1)]=max(F(1:pos(1,1)-t,1));
end
[temp(2,1),pos(2,1)]=max(F(1:pos(1,1)-t,1));
[temp(3,1),pos(3,1)]=max(F(1:pos(2,1)-t,1));
[temp(4,1),pos(4,1)]=max(F(1:pos(3,1)-t,1));
res=pos(1,1);
tt=60;
ratioy=0.5;
for i=2:4
    if(temp(i)>ratiox*temp(1,1))
        x=max(F(2*res-tt:2*res+tt,1));
        if(x<ratioy*F(res,1))%若二次谐波分量很小
            res=pos(i,1);
        end
    end
end

tune=abs(omg(10000+res,1));
tt=50;
harmonic(1,1)=max(F(res-tt:res+tt,1));
for i=2:3
    if(i*res<=size(F,1))
        harmonic(1,i)=max(F(i*res-tt:i*res+tt,1))/harmonic(1,1);
    end
end
harmonic(1,1)=1;
end

function envelope=generateenv(t)%产生指数衰减包络信号
envelope=exp(-4*t/max(t))/max(t);
end

function note=generatenote(tune,time,harmonic)%产生有包络的信号
const=2^(1/12);%紧邻音调之间倍数关系
baseA=220;%存储A调频率

envelope=generateenv(time);

note=harmonic(1,1)*sin(2*pi*baseA*const^tune*time).*envelope...
    +harmonic(1,2)*sin(2*2*pi*baseA*const^tune*time).*envelope...
    +harmonic(1,3)*sin(3*2*pi*baseA*const^tune*time).*envelope;

end

function tune_info=analyze(time,tunes,num)%用于将提取出的初步信息进行整理
baseA=220;

tune_info=zeros(num,2);
for i=1:num
    tune_info(i,1)=round(12*log2(tunes(i,1)/baseA));
    tune_info(i,2)=8/time(i,1);
end
```

```
end

function song=generatesong(tune_info,harmonic_info)
%由提取出的音调和时值信息生成曲子
freq=8000;%采样频率
T=1/freq;%采样周期
beat=0.5;%一拍时间

song=zeros(size(tune_info,1),freq*beat*2);

for i=1:size(tune_info,1)
    t=0:T:beat*4/tune_info(i,2);
    temp=generatenote(tune_info(i,1),t,harmonic_info(i,:));
    song(i,1:size(temp,2))=temp;
end
end

function playsong(song,freq,tune_info)%播放曲子
T=1/freq;%采样周期
beat=0.5;%一拍时间
sound(zeros(8000,1),freq);%稍作等待
pause(1);
for i=1:size(song,1)
    sound(song(i,1:freq*beat*4/tune_info(i,2)),freq);
    pause(beat*4/tune_info(i,2));
end
end
```

得到对该曲子的音符划分如下：



可以看到划分还是比较准确的，之后得到各个音符的音调、谐波分量与时值信息依次如下（0.5s为四分音符时值）：

音符位次	音符基波频率(Hz)	音调	时值(s)	基波分量	二次谐波分量	三次谐波分量
1	220	A3	0.25	1	0.0108	0.2640
2	221	A3	1.5	1	0.2618	0.1681
3	247	B3	0.5	1	0.4172	0.1226
4	222	A3	0.5	1	0.1640	0.1356
5	295	D4	0.5	1	1.1829	0.2451
6	329	E4	0.5	1	1.1168	0.9398
7	195	G3	0.5	1	0.7176	0.4935
8	174	F3	0.5	1	0.3081	0.0542
9	174	F3	0.5	1	0.3417	0.0506
10	294	D4	0.75	1	0.8768	0.1615
11	164	E3	0.25	1	0.7471	0.2382
12	247	B3	1	1	0.3896	0.2267
13	329	E4	0.75	1	0.9985	0.6947
14	222	A3	0.75	1	0.2643	0.2344
15	220	A3	0.5	1	0.0891	1.3015
16	440	A4	0.5	1	0.3741	0.4630
17	220	A3	0.5	1	0.3445	0.2273
18	394	G4	0.25	1	0.4670	0.0954
19	350	F4	0.25	1	0.2311	0.1264
20	329	E4	0.25	1	1.8672	1.0198
21	293	D4	0.25	1	1.0710	0.2393
22	262	C4	0.5	1	0.2451	0.2567
23	247	B3	0.5	1	0.5351	0.2525
24	294	D4	0.5	1	0.6460	0.1778
25	261	C4	0.5	1	0.5513	0.2126
26	174	F3	0.5	1	0.3087	0.0292
27	221	A3	0.5	1	0.2430	0.2379
28	248	B3	0.5	1	0.5868	0.1856
29	221	A3	0.5	1	0.2508	0.1366
30	209	A3(b)	2	1	0.4883	0.9010

直接按照上面的原则得到的音调（或许）基本是正确的，但是可能也有个别音调存在把谐波分量（或某种甚至不是谐波的分量）误判为基波的现象（一般是因为傅里叶变换结果中基波峰值太低）。利用提取出的信息恢复出一个曲子，将其和原曲进行对比，发现在一些特征明显的片段（比如第16个到第25个音符部分）还是十分一致的。

1.2.3 基于傅里叶级数的合成音乐

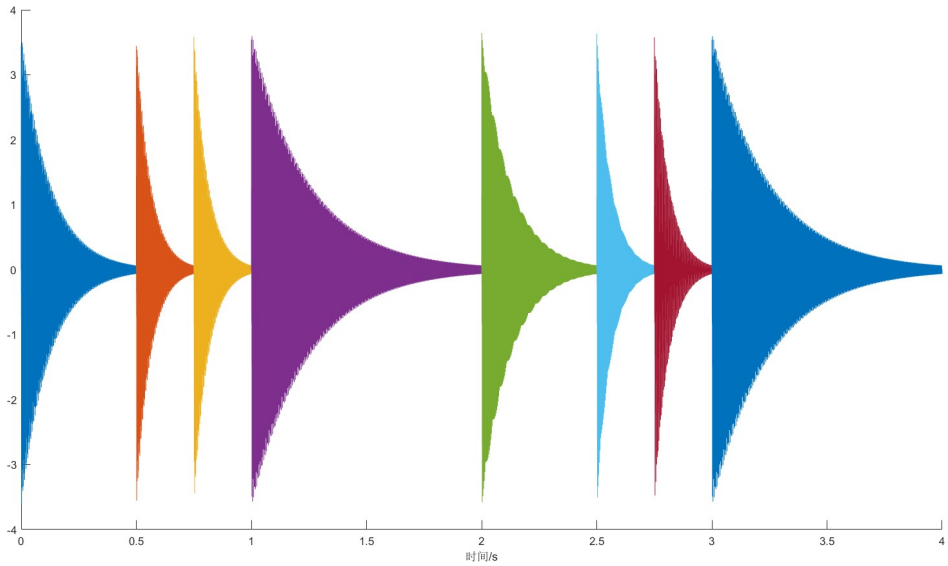
(10) 用(7)计算出来的傅里叶级数再次完成第(4)题，听一听是否像演奏fmt.wav的吉他演奏出来的?

利用此前计算出的谐波比例关系，编写代码如下(对应hw_1_2_3_10.m文件):

```
clear all; close all; clc;

freq=8000;%采样频率

tune_info=[7,4;7,8;9,8;2,2;0,4;0,8;-3,8;2,2];%输入乐曲信息

song=generatesong(tune_info);%生成乐曲
playsong(song,freq,tune_info);%播放乐曲

% beat=0.5;
% t=1;
% for i=1:size(song,1)
% hold on;
% plot(t:t+freq*beat*4/tune_info(i,2)-1,...
% song(i,1:freq*beat*4/tune_info(i,2)));
% t=t+freq*beat*4/tune_info(i,2);
% end

function envelope=generateenv(t)%产生指数衰减包络信号
envelope=exp(-4*t/max(t));
end

function note=generatenote(tune,time)%产生有包络的信号
const=2^(1/12);%紧邻音调之间倍数关系
baseF=349.23;%存储F调频率

envelope=generateenv(time);

note=sin(2*pi*baseF*const^tune*time).*envelope...
    +1.434*sin(2*2*pi*baseF*const^tune*time).*envelope...
    +0.964*sin(3*2*pi*baseF*const^tune*time).*envelope...
    +1.096*sin(4*2*pi*baseF*const^tune*time).*envelope;

end

function song=generatesong(tune_info)%生成乐曲
freq=8000;%采样频率
T=1/freq;%采样周期
beat=0.5;%一拍时间

song=zeros(size(tune_info,1),freq*beat*2);

for i=1:size(tune_info,1)
    t=0:T:beat*4/tune_info(i,2);
```

```
    temp=generatenote(tune_info(i,1),t);
    song(i,1:size(temp,2))=temp;
end
end

function playsong(song,freq,tune_info)%播放乐曲
T=1/freq;%采样周期
beat=0.5;%一拍时间
sound(zeros(8000,1),freq);%稍作等待
pause(1);
for i=1:size(song,1)
    sound(song(i,1:freq*beat*4/tune_info(i,2)),freq);
    pause(beat*4/tune_info(i,2));
end
end
```

播放出来的效果和fmt.wav相较于此前更相近了一点。其波形如下：



(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。

在该部分，先对fmt.wav进行分析，得到其二次、三次谐波信息，然后输入《东方红》曲目信息，根据《东方红》曲目的音调在该吉他的音调信息中进行检索，若有匹配的音调（若没有直接对应的音调，则取临近的音调），则载入其谐波分量信息，通过这种方式使得每个音调都具有不同的谐波分量比例，从而更好地模拟吉他的声音（实际上同一个音调其谐波分量也可能有很多种，这里忽略了这种差异，做了理想化近似）。编写代码如下(对应hw_1_2_3_11.m文件)：

```
clear all; close all; clc;

[y,Fs]=audioread("fmt.wav");%读取文件

freq=8000;%采样频率

%通过峰值统计音符时值
time=zeros(50,1);%存储音符时值
```

```
pos=zeros(50,1);%存储音符起始点与终止点
pos(1)=352;%设定音符起始点
jump=0.25*8000;%各个音符时域的最小单位为0.25s
range=500;%临近音符允许的相交部分
range0=500;%确定和前面比较的范围，用于找跳变值
range1=300;%确定和前面比较的范围，用于找跳变值
ratio=1.5;%跳变程度
num=2;%存储音符数量
for i=201:size(y,1)
    if(i-pos(num-1,1)>jump-range)
        [temp,temp1]=max(y(i-range0:i-range1,1));
        if(y(i,1)>0 && y(i,1)/temp>ratio)
            pos(num,1)=i;
            time(num-1,1)=round((i-pos(num-1,1))/jump);
            num=num+1;
        end
    end
end
num=num-1;
time(num,1)=round((size(y,1)-pos(num-1,1))/jump);

%通过prefourier函数确定各个音符的音调
tunes=zeros(num,1);%存储音符音调信息
pos(num+1,1)=size(y,1);
harmonic_info=zeros(num,3);%存储各个谐波分量归一化峰值信息
x=200;%用于取音符时域的中间部分
for i=1:num
    Trg=[0,(pos(i+1,1)-pos(i,1)+1-x)/8000];
    N=pos(i+1,1)-pos(i,1)+1-x;
    OMGrg=[-4000*pi,4000*pi];
    K=20000;
    [omg,FT]=prefourier(Trg,N,OMGrg,K);
    F_info=FT*y(pos(i,1):pos(i+1,1)-x,1);
    [tunes(i,1),harmonic_info(i,:)]=...
    gettune(omg/2/pi,abs(F_info(10001:20000,1)));
end

% tunes(26,1)=247;%可能需要对该音调进行修正

info=analyze(time,tunes,num);%整合输入曲目信息
tune_info=[7,4;7,8;9,8;2,2;0,4;0,8;-3,8;2,2];%输入乐曲信息

%得到即将演奏曲目的谐波信息
harmonic_new=getharmonic(info,tune_info,harmonic_info);

song=generatesong(tune_info,harmonic_new);%生成曲子
playsong(song,freq,tune_info);%播放曲子

function [omg,FT] = prefourier(Trg,N,OMGrg,K)
% 输入参数：
% Trg  : 二维矢量，两个元素分别表示时域信号的起止时间；
% N    : 时域抽样数量；
% OMGrg: 二维矢量，两个元素分别表示频谱的起止频率；
% K    : 频域抽样数量。
% 输出参数：
% omg  : 抽样频率；
% FT   : 实现傅里叶变换的矩阵~U~及系数；
T = Trg(2)-Trg(1);
```

```
t = linspace(Trg(1),Trg(2)-T/N,N)';
OMG = OMGrg(2)-OMGrg(1);
omg = linspace(OMGrg(1),OMGrg(2)-OMG/K,K)';
FT = T/N*exp(-1i*kron(omg,t.'));
end

function [tune,harmonic]=gettune(omg,F)%由傅里叶变换图像找基波和谐波信息
harmonic=zeros(1,3);%存储谐波信息
ratiox=0.71;
temp=zeros(4,1);
pos=zeros(4,1);
t=60;
[temp(1,1),pos(1,1)]=max(F);
if(omg(10000+pos(1,1),1)>500)
    [temp(1,1),pos(1,1)]=max(F(1:pos(1,1)-t,1));
end
if(omg(10000+pos(1,1),1)>500)
    [temp(1,1),pos(1,1)]=max(F(1:pos(1,1)-t,1));
end
[temp(2,1),pos(2,1)]=max(F(1:pos(1,1)-t,1));
[temp(3,1),pos(3,1)]=max(F(1:pos(2,1)-t,1));
[temp(4,1),pos(4,1)]=max(F(1:pos(3,1)-t,1));
res=pos(1,1);
tt=60;
ratioy=0.5;
for i=2:4
    if(temp(i)>ratiox*temp(1,1))
        x=max(F(2*res-tt:2*res+tt,1));
        if(x<ratioy*F(res,1))%若二次谐波分量很小
            res=pos(i,1);
        end
    end
end

tune=abs(omg(10000+res,1));
tt=50;
harmonic(1,1)=max(F(res-tt:res+tt,1));
for i=2:3
    if(i*res<=size(F,1))
        harmonic(1,i)=max(F(i*res-tt:i*res+tt,1))/harmonic(1,1);
    end
end
harmonic(1,1)=1;
end

function envelope=generateenv(t)%产生指数衰减包络信号
envelope=exp(-4*t/max(t))/max(t);
end

function note=generatenote(tune,time,harmonic)%产生有包络的信号
const=2^(1/12);%紧邻音调之间倍数关系
baseF=349.23;%存储F调频率

envelope=generateenv(time);

note=harmonic(1,1)*sin(2*pi*baseF*const^tune*time).*envelope...
    +harmonic(1,2)*sin(2*2*pi*baseF*const^tune*time).*envelope...
    +harmonic(1,3)*sin(3*2*pi*baseF*const^tune*time).*envelope;
```

```
125
126 end
127
128 function tune_info=analyze(time,tunes,num)%用于将提取出的初步信息进行整理
129 baseA=220;
130
131 tune_info=zeros(num,2);
132 for i=1:num
133     tune_info(i,1)=round(12*log2(tunes(i,1)/baseA));
134     tune_info(i,2)=8/time(i,1);
135 end
136 end
137
138 function song=generatesong(tune_info,harmonic_info)
139 %由提取出的音调和时值信息生成曲子
140 freq=8000;%采样频率
141 T=1/freq;%采样周期
142 beat=0.5;%一拍时间
143
144 song=zeros(size(tune_info,1),freq*beat*2);
145
146 for i=1:size(tune_info,1)
147     t=0:T:beat*4/tune_info(i,2);
148     temp=generatenote(tune_info(i,1),t,harmonic_info(i,:));
149     song(i,1:size(temp,2))=temp;
150 end
151 end
152
153 function playsong(song,freq,tune_info)%播放曲子
154 T=1/freq;%采样周期
155 beat=0.5;%一拍时间
156 sound(zeros(8000,1),freq);%稍作等待
157 pause(1);
158 for i=1:size(song,1)
159     sound(song(i,1:freq*beat*4/tune_info(i,2)),freq);
160     pause(beat*4/tune_info(i,2));
161 end
162 end
163
164 function harmonic=getharmonic(info,tune_info,harmonic_info)
165 harmonic=zeros(size(tune_info,1),3);
166 for j=1:size(tune_info,1)
167     temp=0;
168     for i=1:size(info,1)
169         if(info(i,1)==tune_info(j,1)-4)%改变了一个八度
170             harmonic(j,:)=harmonic_info(i,:);
171             temp=1;
172             break;
173         end
174     end
175     if(temp==0)
176         for k=1:8%搜索临近音调
177             for i=1:size(info,1)
178                 if(info(i,1)==tune_info(j,1)-4+k)
179                     harmonic(j,:)=harmonic_info(i,:);
180                     temp=1;
181                     break;
182                 end
```

```
                if(info(i,1)==tune_info(j,1)-4-k)
                    harmonic(j,:)=harmonic_info(i,:);
                    temp=1;
                    break;
                end
            end
            if(temp==1)
                break;
            end
        end
    end
end
end
```

这样生成的《东方红》曲子，其谐波分量确实更加多变了一些，或许也更接近真实的吉他了。

(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。

我对前面的内容进行了封装，编写了如下代码(对应hw_1_2_3_12文件夹中的gui.m文件和gui.fig)（由于代码整体较长，所以只展示了实际发挥作用的部分）：

```
function pushbutton4_Callback(hObject, eventdata, handles)
str=get(handles.popupmenu1, 'String');
val=get(handles.popupmenu1,'Value');
const=2^(1/12);%紧邻音调之间倍数关系
switch str{val};
case '分析参考曲目得到谐波分量'
    temp=1;
case '直接采用输入的谐波分量'
    temp=2;
end

if(temp==1)%选择第一个选项，分析参考曲目并根据其谐波分量演奏曲目
    filename=get(handles.edit2, 'String');
    [y,Fs]=audioread(filename);%读取文件
    axes(handles.axes1);%绘制参考曲目波形
    cla reset;

    plot(1/8000:1/8000:size(y,1)/8000,y);
    xlabel("时间/s");

    freq=8000;%采样频率

    %通过峰值统计音符时值
    time=zeros(50,1);%存储音符时值
    pos=zeros(50,1);%存储音符起始点与终止点
    pos(1)=352;%设定音符起始点
    jump=0.25*8000;%各个音符时域的最小单位为0.25s
    range=500;%临近音符允许的相交部分
    range0=500;%确定和前面比较的范围，用于找跳变值
    range1=300;%确定和前面比较的范围，用于找跳变值
    ratio=1.5;%跳变程度
    num=2;%存储音符数量
    for i=201:size(y,1)
        if(i-pos(num-1,1)>jump-range)
```

```
            [temp,temp1]=max(y(i-range0:i-range1,1));
            if(y(i,1)>0 && y(i,1)/temp>ratio)
                pos(num,1)=i;
                time(num-1,1)=round((i-pos(num-1,1))/jump);
                num=num+1;
            end
        end
    end
    num=num-1;
    time(num,1)=round((size(y,1)-pos(num-1,1))/jump);

    %通过prefourier函数确定各个音符的音调
    tunes=zeros(num,1);%存储音符音调信息
    pos(num+1,1)=size(y,1);
    harmonic_info=zeros(num,3);%存储各个谐波分量归一化峰值信息
    x=200;%用于取音符时域的中间部分
    for i=1:num
        Trg=[0,(pos(i+1,1)-pos(i,1)+1-x)/8000];
        N=pos(i+1,1)-pos(i,1)+1-x;
        OMGrg=[-4000*pi,4000*pi];
        K=20000;
        [omg,FT]=prefourier(Trg,N,OMGrg,K);
        F_info=FT*y(pos(i,1):pos(i+1,1)-x,1);
        [tunes(i,1),harmonic_info(i,:)]=gettune(omg/2/pi,...
        abs(F_info(10001:20000,1)));
    end

    % tunes(26,1)=247;%可能需要对该音调进行修正

    info=analyze(time,tunes,num);%整合输入曲目信息
    intunes=get(handles.edit3,'String');
    intunes=str2num(intunes);
    tune_info=zeros(size(intunes,2),2);
    for i=1:size(intunes,2)
        tune_info(i,1)=intunes(1,i);
    end
    intime=get(handles.edit5,'String');
    intime=str2num(intime);
    for i=1:size(intime,2)
        tune_info(i,2)=intime(1,i);
    end
    harmonic_new=getharmonic(info,tune_info,harmonic_info);
    beat=get(handles.edit6,'String');%一拍时间
    beat=str2double(beat);
    base=get(handles.edit1,'String');%存储基准频率
    base=str2double(base);
    base=round(12*log2(base/220));
    base=220*const^base;

    %利用提取出的音调和时值信息生成曲子
    song=generatesong(base,beat,tune_info,harmonic_new);

    axes(handles.axes2);%绘制演奏曲目波形
    cla reset;
    beat=0.5;
    t=1;
    for i=1:size(song,1)
        hold on;
```

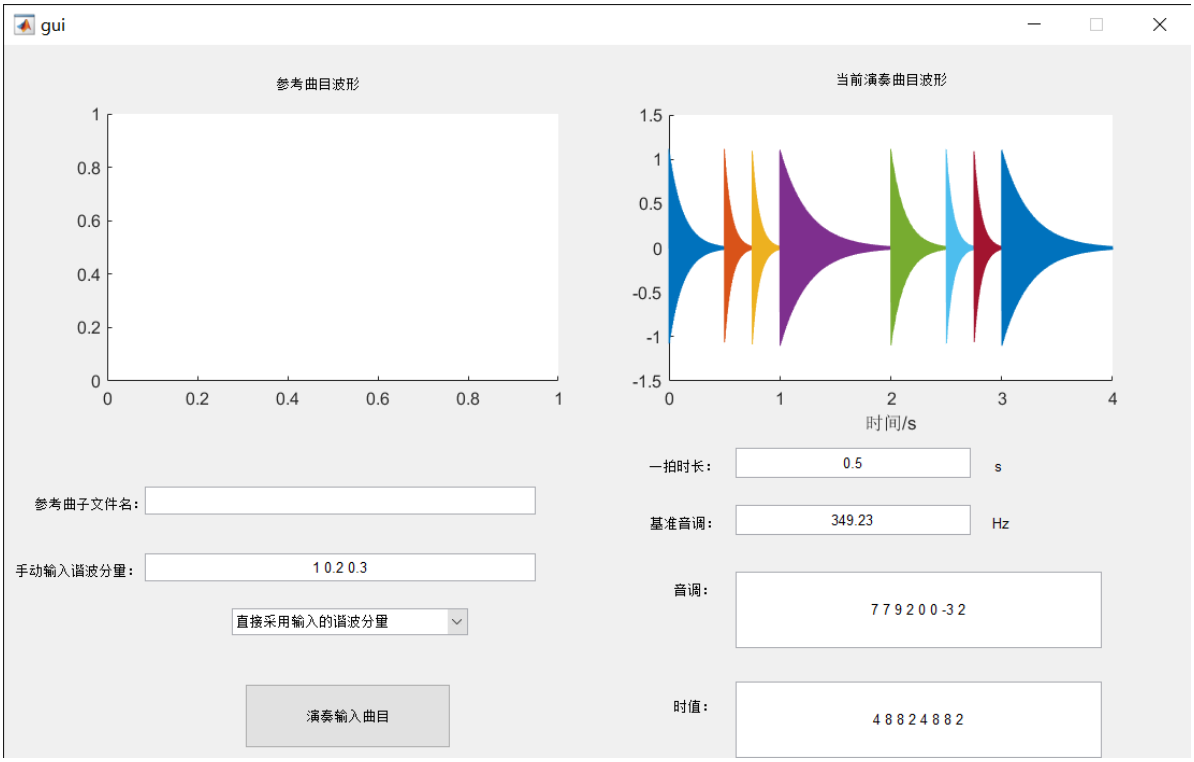
```
        xlabel("时间/s");
        plot(t/8000:1/8000:(t+freq*beat*4/tune_info(i,2)-1)/8000,...
        song(i,1:freq*beat*4/tune_info(i,2)));
        t=t+freq*beat*4/tune_info(i,2);
    end

    playsong(beat,song,freq,tune_info);%播放曲子
elseif(temp==2)%选择第二个选项，直接根据输入的谐波分量演奏曲目
    axes(handles.axes1);
    cla reset;
    freq=8000;
    intunes=get(handles.edit3,'String');
    intunes=str2num(intunes);
    tune_info=zeros(size(intunes,2),2);
    for i=1:size(intunes,2)
        tune_info(i,1)=intunes(1,i);
    end
    intime=get(handles.edit5,'String');
    intime=str2num(intime);
    for i=1:size(intime,2)
        tune_info(i,2)=intime(1,i);
    end
    harmonic_new=zeros(size(tune_info,1),3);
    constharmonic=get(handles.edit7,'String');
    if(constharmonic=="")%没有输入谐波分量
        for i=1:size(harmonic_new)
            harmonic_new(i,1)=1;
            harmonic_new(i,2)=0;
            harmonic_new(i,3)=0;
        end
    else
        constharmonic=str2num(constharmonic);
        for i=1:size(harmonic_new)
            harmonic_new(i,1)=constharmonic(1,1);
            harmonic_new(i,2)=constharmonic(1,2);
            harmonic_new(i,3)=constharmonic(1,3);
        end
    end
    beat=get(handles.edit6,'String');%一拍时间
    beat=str2double(beat);
    base=get(handles.edit1,'String');%存储基准频率
    base=str2double(base);
    base=round(12*log2(base/220));
    base=220*const^base;

    %利用提取出的音调和时值信息生成曲子
    song=generatesong(base,beat,tune_info,harmonic_new);

    axes(handles.axes2);%绘制演奏曲目波形
    cla reset;
    beat=0.5;
    t=1;
    for i=1:size(song,1)
        hold on;
        xlabel("时间/s");
        plot(t/8000:1/8000:(t+freq*beat*4/tune_info(i,2)-1)/8000,...
        song(i,1:freq*beat*4/tune_info(i,2)));
        t=t+freq*beat*4/tune_info(i,2);
```

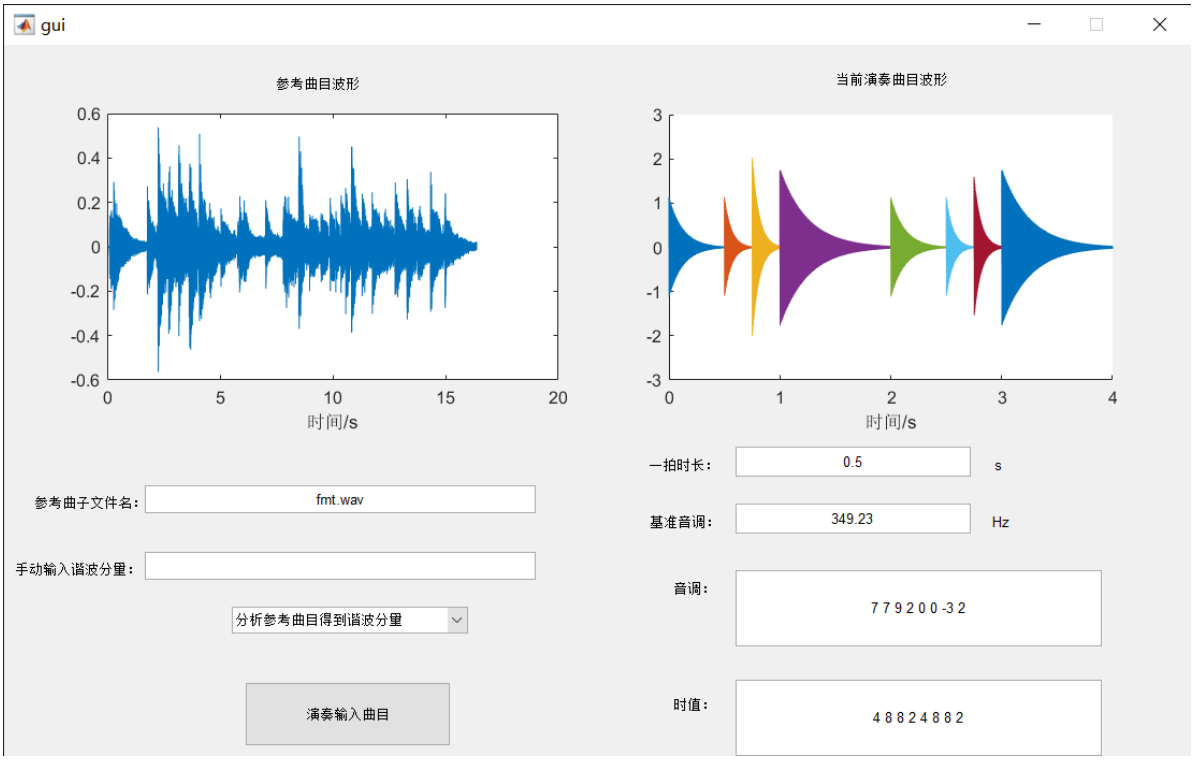
```
    end

    playsong(beat,song,freq,tune_info);%播放曲子
end
end

function envelope=generateenv(t)%产生指数衰减包络信号
envelope=exp(-4*t/max(t));
end

function note=generatenote(base,tune,time,harmonic)%产生有包络的信号
const=2^(1/12);%紧邻音调之间倍数关系

envelope=generateenv(time);

note=harmonic(1,1)*sin(2*pi*base*const^tune*time).*envelope...
    +harmonic(1,2)*sin(2*2*pi*base*const^tune*time).*envelope...
    +harmonic(1,3)*sin(3*2*pi*base*const^tune*time).*envelope;

end

function song=generatesong(base,beat,tune_info,harmonic_info)
%由提取出的音调和时值信息生成曲子
freq=8000;%采样频率
T=1/freq;%采样周期

song=zeros(size(tune_info,1),freq*beat*2);

for i=1:size(tune_info,1)
    t=0:T:beat*4/tune_info(i,2);
    temp=generatenote(base,tune_info(i,1),t,harmonic_info(i,:));
    song(i,1:size(temp,2))=temp;
end
end

function playsong(beat,song,freq,tune_info)%播放乐曲
T=1/freq;%采样周期
sound(zeros(8000,1),freq);%稍作等待
pause(1);
for i=1:size(song,1)
    sound(song(i,1:freq*beat*4/tune_info(i,2)),freq);
    pause(beat*4/tune_info(i,2));
end
end

function [omg,FT] = prefourier(Trg,N,OMGrg,K)
% 输入参数：
% Trg  : 二维矢量，两个元素分别表示时域信号的起止时间；
% N    : 时域抽样数量；
% OMGrg: 二维矢量，两个元素分别表示频谱的起止频率；
% K    : 频域抽样数量。
% 输出参数：
% omg  : 抽样频率；
% FT   : 实现傅里叶变换的矩阵~U~及系数；
T = Trg(2)-Trg(1);
t = linspace(Trg(1),Trg(2)-T/N,N)';
OMG = OMGrg(2)-OMGrg(1);
omg = linspace(OMGrg(1),OMGrg(2)-OMG/K,K)';
```

```
FT = T/N*exp(-1i*kron(omg,t.'));
end

function [tune,harmonic]=gettune(omg,F)%由傅里叶变换图像找基波和谐波信息
harmonic=zeros(1,3);%存储谐波信息
ratiox=0.71;
temp=zeros(4,1);
pos=zeros(4,1);
t=60;
[temp(1,1),pos(1,1)]=max(F);
if(omg(10000+pos(1,1),1)>500)
    [temp(1,1),pos(1,1)]=max(F(1:pos(1,1)-t,1));
end
if(omg(10000+pos(1,1),1)>500)
    [temp(1,1),pos(1,1)]=max(F(1:pos(1,1)-t,1));
end
[temp(2,1),pos(2,1)]=max(F(1:pos(1,1)-t,1));
[temp(3,1),pos(3,1)]=max(F(1:pos(2,1)-t,1));
[temp(4,1),pos(4,1)]=max(F(1:pos(3,1)-t,1));
res=pos(1,1);
tt=60;
ratioy=0.5;
for i=2:4
    if(temp(i)>ratiox*temp(1,1))
        x=max(F(2*res-tt:2*res+tt,1));
        if(x<ratioy*F(res,1))%若二次谐波分量很小
            res=pos(i,1);
        end
    end
end

tune=abs(omg(10000+res,1));
tt=50;
harmonic(1,1)=max(F(res-tt:res+tt,1));
for i=2:3
    if(i*res<=size(F,1))
        harmonic(1,i)=max(F(i*res-tt:i*res+tt,1))/harmonic(1,1);
    end
end
harmonic(1,1)=1;
end

function tune_info=analyze(time,tunes,num)%用于将提取出的初步信息进行整理
baseA=220;

tune_info=zeros(num,2);
for i=1:num
    tune_info(i,1)=round(12*log2(tunes(i,1)/baseA));
    tune_info(i,2)=8/time(i,1);
end
end

function harmonic=getharmonic(info,tune_info,harmonic_info)
harmonic=zeros(size(tune_info,1),3);
for j=1:size(tune_info,1)
    temp=0;
    for i=1:size(info,1)
        if(info(i,1)==tune_info(j,1)-4)%改变了一个八度
```

```
            harmonic(j,:)=harmonic_info(i,:);
            temp=1;
            break;
        end
    end
    if(temp==0)
        for k=1:8%搜索临近音调
            for i=1:size(info,1)
                if(info(i,1)==tune_info(j,1)-4+k)
                    harmonic(j,:)=harmonic_info(i,:);
                    temp=1;
                    break;
                end
                if(info(i,1)==tune_info(j,1)-4-k)
                    harmonic(j,:)=harmonic_info(i,:);
                    temp=1;
                    break;
                end
            end
            if(temp==1)
                break;
            end
        end
    end
end
end
```

该图形界面具有的大致功能为，可以通过输入曲目信息与谐波分量比例信息对该曲目进行演奏并绘制其波形，同时也可以选择对输入的文件中的参考曲目（文件需要放在同一文件夹下）进行分析，得到其谐波分量比例信息，然后再根据这些信息，模拟该乐器的演奏效果来演奏输入曲目并绘制其波形。其效果如下：





要输入的基础信息为一拍时长（这里等同于一个四分音符的时值了）、基准音调（输入某一频率后，会自动将其转化为最邻近的音调的准确频率）、音调（即需要演奏的曲目的音调信息，是基于基准音调确定的，1代表比基准音调高1个半音阶，以此类推，以空格相隔）、时值（和前面的音调相对应，4代表四分音符，8代表八分音符，以此类推，以空格相隔），此外如果选择“直接采用输入的谐波分量”选项，则还要手动输入谐波分量的比例（若没有输入则默认只有基波分量），从左至右依次为基波、二次谐波、三次谐波的比例关系（以空格相隔），如果选择“分析参考曲目得到谐波分量”，则还要将参考曲目的文件放于同一文件夹下，并输入其文件名。最后点击“演奏输入曲目”并等待，就可以看到波形在上方显示同时听到演奏的曲子（如果没有选择分析参考曲目，则参考曲目波形不会被绘制）。经测试可以正常运行。

总结

本次利用matlab完成音乐合成大作业，逐渐熟悉了matlab各种常见函数和基础语法的运用，最终基本成功实现了预期效果，并完成了程序的图形界面封装，不过还有很多不足有待完善：

- 由于本人音乐素养有限，不能完全确定利用程序分析出的fmt.wav的音调是否完全正确，只能将其播放和原曲对比，不过由于很多泛音被移除，有些地方仍不能很好地对比；
- 本次大作业只提取了最高到三次谐波分量，或许增加更多谐波分量效果会更好；
- 该大作业对音乐文件的时值和音调分析都是建立在fmt.wav的基础上，所以假如换了其他音乐文件进行分析可能效果不是很好。