MATLAB 音乐合成大作业

实验报告

班级：无 56

姓名：暴志鹏

学号：2015010675

一、 简单的音乐合成

1.1 简单合成东方红

实验代码见 codes/ex1/ex1_1.m

```
clear all;close all; clc;
w0 = 220; %基频率
index = 2^(1/12);
basic = w0/index^4;
freq = 1:24; %all
freq(1) = basic;
for i = 2:24
    freq(i) = freq(i-1)*index;
end
fs = 8000;%sample rate
t1 = (1:fs/2)/fs;% 四分音符
t2 = (1:fs/4)/fs;% 八分音符
bF = sin(2*pi*freq(1)*t1);
bG = sin(2*pi*freq(3)*t1);
bA = sin(2*pi*freq(5)*t1);
bB = sin(2*pi*freq(7)*t1);
bC = sin(2*pi*freq(8)*t1);
bD = sin(2*pi*freq(10)*t1);
bE = sin(2*pi*freq(12)*t1);
...


music = [C,hC,hD,G,G,F,hF,hbD,G,G];
sound(music)
```

对于这道题，我采取了打表的方法来记录音符，这种方法比较麻烦但是一劳永逸，改进方法可以写成循环，这个会在后面的代码中体现。东方红片段共用到了三种音符：二分音符、四分音符和八分音符。但是由于这道题很简单，没有包络，幅度和相位在相同音符衔接处变化不大，我没有做 2 分音符。实际上，仅用 8 分音符一种

也可以表示。

这个音乐音调完全正确，但是在不同音符的衔接处有明显的啪啪声，原因在于相位不连续。

1.2 加入包络的音乐合成

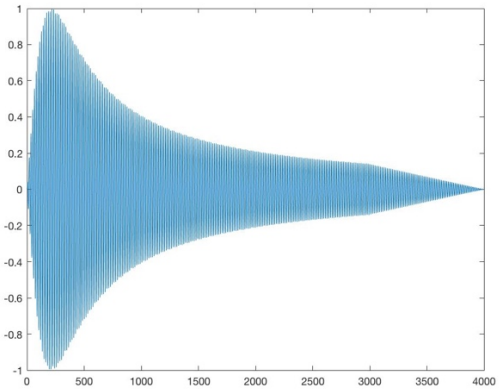
代码见 codes/ex1/ex1_2.m

根据提供的资料，得知乐音的声学形状大约是一个小山形状，思考这样的函数，我想到了之前接触过的形如$x+\frac{1}{x}$的函数，他的形状如钩，那它的倒数就符合要求。为了保证最终结尾为 0，对该函数最后进行了一些线性处理。关键代码如下：

```
%y = 1./(0.04/sample*t+1./(3200/sample*t));
a = 0.1;
b = 3500;
cover0 = 1./(a/fs*t0+1./(b/fs*t0));
k0 = 4./(a*3/4+1./(b*3/4))/fs;
tt0 = (3/4*fs+1:fs);
cover0(3*fs/4+1:fs) = -k0*(tt0-fs);
cover0 = cover0/max(cover0);
```

由于各个音符时间不一样长，共有三个包络函数：cover0，cover1，cover2。

加入包络之后音乐明显有了改善，衔接处的啪啪声消失音乐也有了动感。加入包络后画出音符时域图像如下：



1.3 简单方法使音乐升高 / 降低 8 阶以及用 resample 函数让音乐提升半个音阶。

代码见 codes/ex1/ex1_3.m

本题目没有什么难度，掌握 sound 和 resample 的原理和接口即可。代码如下，降低八度只要将 2*fs 变成 fs / 2 即可。

```
music = [C,hC,hD,wG,F,hF,hbD,wG];
%plot (music)
sound(music,2*fs);
clear sound;
music2 = resample(music,10000,round((10000*2^(1/12))));
sound(music2);
```

1.4 加入谐波分量

代码见 codes/ex1/ex1_4.m

本题也较为简单，就是在原来打好的表的基础上再加一次，没有本质变化和关键代码。我认为做的不足的地方主要是代码不够美观，过于冗杂。简化代码的方法可以写成两个循环，外循环对每个音符做处理，内层循环对谐波分量做处理。这个代码写法在 ex1 中

均没有实现，在 ex3 中改变风格(原因在于之后的合成音乐用循环写不方便对应音符名字)。

加入谐波分量后，音乐明显有了厚度，用 1，0.2，0.3 这个组合声音很像风琴。

1.5 音乐合成

代码见 codes/ex1/ex1_5.m

由于提示我们合成的乐音比较像风琴，于是我选择了一首适合风琴演奏的音乐：《雪绒花》。

大体上包络处理等都和前问一样，单引入了一些新的改变。

a. 尝试做混音，即在风琴的基础上，加入另一种新的乐音，新的乐音音调采取降一个 8 度，包络形状答大体一样，包峰位置和平缓地方的值有所不同。

```
freq = 1:24; %all
freq(1) = basic;
for i = 2:24
    freq(i) = freq(i-1)*index1;
end
freq2 = freq/2;
```

```
c = 1;
d = 100;

cover33 = 1./(2*c/fs*t3/3+1./(2*d/fs*t3/3));
k33 = 4./(c*9/8+1./(d*9/8))/fs;
tt33 = (9/8*fs+1:3*fs/2);
cover33(9*fs/8+1:3*fs/2) = -k33*(tt33-3*fs/2);
cover33 = cover33/max(cover33);
```

b. 加入混响，这一点受到谷老师布置的语音去噪大作业的启发，

引入几个延时因子会显得音乐有回响，更接近真实乐器。

```
delay1 = 4000;
delay2 = 6000;
zeo1 = zeros(1,delay1);
zeo2 = zeros(1,delay2);
l = length(music);
m_1 = [zeo1,music(1:l-delay1)];
m_2 = [zeo2,music(1:l-delay2)];
music = music + 0.2*m_1+0.3*m_2;
```

需要说明的是，最后的 sound 做了一些 tricky 的事情，查到《雪绒花》的节拍是 96，也就是说四分音符并不是完整的一秒，改变时间 t1，t2 较为麻烦，所以用了前问用过的方法，压缩演奏时间，但是带来的结果是音调略有变化，但不影响最终结果。

总体来说，合成的音乐较为动听，听起来像是钢琴和萨克斯的合奏效果，但是混音部分做的不够好，没法完全听出来是两种乐器，我觉得原因应该在于谐波分量没有相应调整以及包络整体形状还是太像，即使峰值变化也难题体现两种音乐的差别。

我将音乐写成了 1_5.wav，可以直接播放。

二、 用傅立叶级数分析音乐

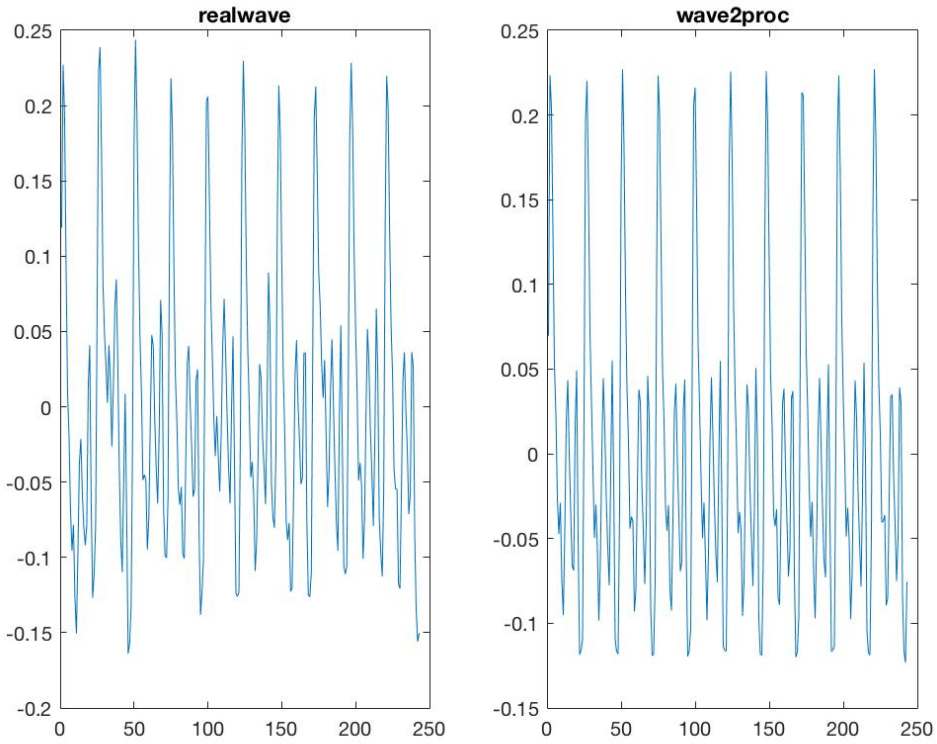
2.1 读 wave 文件，画出波形

代码见 codes/ex2/ex2_1.m

本题很简单，代码如下：

```
clear all;
close all;
clc;
load Guitar.mat;
music = audioread('fmt.wav');
figure;
subplot(1,2,1);
plot(realwave);
title('realwave');
subplot(1,2,2);
plot(wave2proc);
title('wave2proc');
sound(music);
```

画出两者波形如下：



播放效果非常真实，仿佛是真的吉他在演奏。

2.2 去除噪声

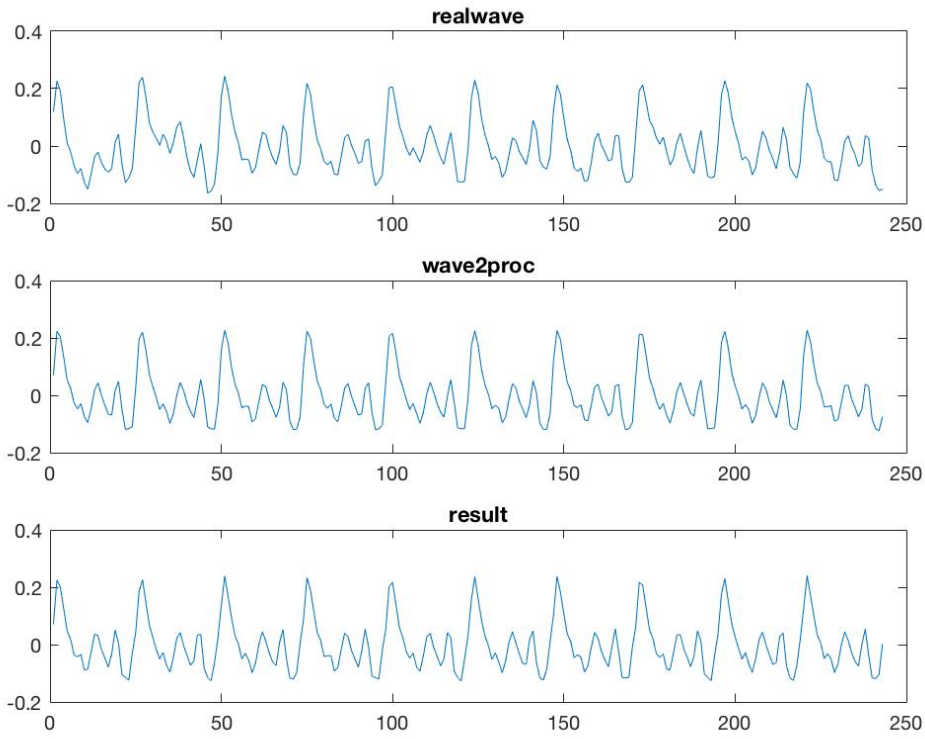
代码见 codes/ex2/ex2_2.m

观察二者波形会发现，虽然二者都是有十个周期，但是显然 realwave 的波形参差较大，而后者较为均匀，对于这道题，我才用的方法是取 realwave 每个周期的均值再做周期延陀来减小噪声。最后还需要说明，由于最后做了周期延陀导致最终长度是 10 的倍数，所以还需要做一个 resample 来保证长度一样。

代码如下：(在 sublime 编辑器下注释的中文显示有问题)

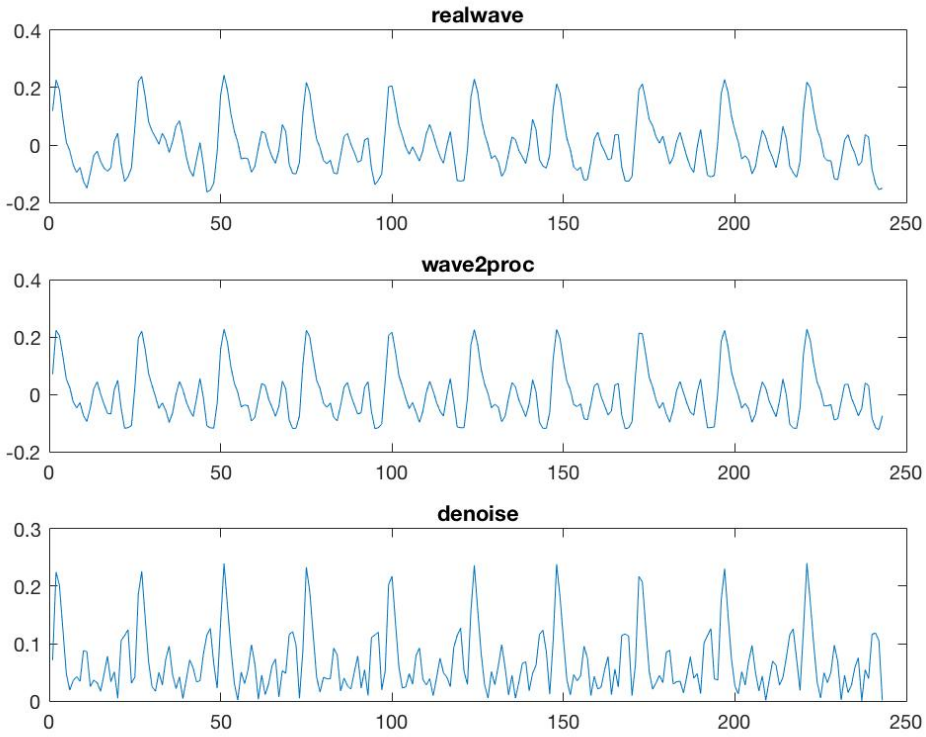
```
8   middle_l = length(middle);
9   result = zeros(origin_l,1);
10  for i = 1:10
11      result = result + middle((i-1)*origin_l+1:i*origin_l);
12  end
13  result = result/10;
14  result = resample(result,1,10);%Ñ¹Ëõ1£¯10
15  result = [result;result;result;result;result];
16  result = [result;result];
17  l1 = length(result);
18  l2 = origin_l;
19  result = resample(result,1000,round(1000*(l1/l2)));%¶¯³¤
20  %noise = realwave - result
21  %F1 = fft(realwave);
22  %F2 = fft(noise);
23  %F = F1-F2;
24  %res_final = abs(ifft(F));
25
26  similarity = max(xcorr(result,wave2proc))/max(xcorr(wave2proc));%¾ÆËãÏàËÆ¶È
```

对比的波形如下。可以发现其值几乎相同，最后一点略高我认为是 resample 重采样导致。



在这个基础上，比较发现 wave2proc 并不是完全的周期延陀，我打算用普减法去解决这个问题，将原信号减去均值 result 信号视为噪声，在频域相减再反变化到时域得到最终结果。实验代码

如上图中注释部分，实验结果如下图：



可以发现效果不好，分析原因我觉得主要有两点：

a. wave 长度太短导致噪声谱不准确；

b. 这个文件长度为 243，直接做 FFT 实际上有很大误差，由于 FFT 内部实现原理是二分法，一来二分次数不多，二来很多时候不是均匀二分，会对结果有些影响。

2.3 分析 fmt.wav 的基频和音调

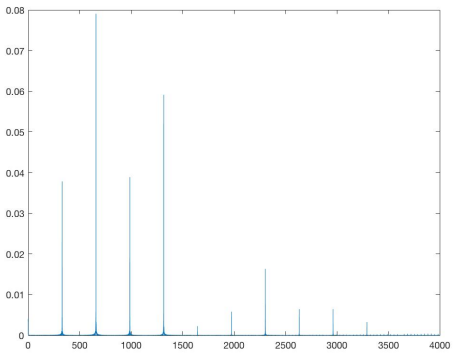
代码见 codes/ex2/ex2_3.m

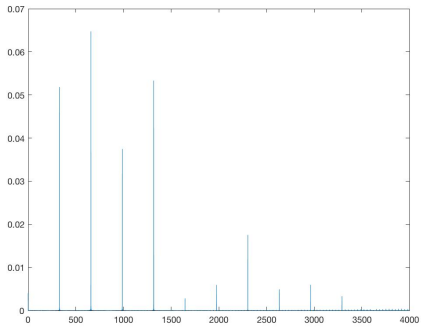
这段音乐的基频是 329.6Hz，为 E 调。

这道题主要是考察我们对傅立叶变化的理解，关于用 FFT 做频域分析，主要代码是：

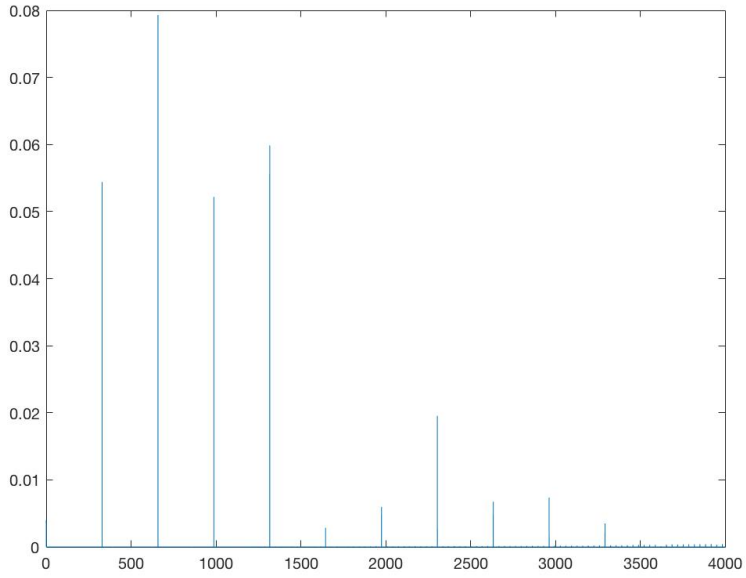
```
i = 1;
while 2^i<l
    i=i+1;
end
f_l = 2^i;

F = fft(element,f_l)/l;
F = abs(F);

f = (1:f_l/2)/f_l*fs;
f = f';
plot(f,2*abs(F(1:f_l/2)));
```

这里，需要说明的是我对语音片段做的 FFT 的长度选择的 2 的幂次。原因在于之前所说的，FFT 主要是用了二分法来做快速运算，为了提高准确性和速度，分割的点应该选 2 的幂次。





上图左边是延拓 128 次的结果，右边是延拓 1024 次的结果，两次结果有明显不同，说明算法有问题。查阅 FFT 文档后发现，原来 FFT 函数的第二个参数实现的机制并不是对原来信号进行 resample 这样的操作，而是直接在后面补零。这显然会对结果造成影响。于是我放弃了这个算法，还是直接做 FFT 变换。得到结果如下：



关于周期延拓后分析效果变好的原因我认为在于首先根据傅里叶变换的性质，周期延拓相当于多个时域平移，频域上的结果是频谱位置不变，值有一个相同系数的变化(傅里叶变换的时域平移)。这保证了结果的可行性，而增加信号长度做 FFT 时增加了频域取点个数，对于合成音乐，可以有效提取信息。

最后，为后续工作做一点铺垫，我保存了这个波的谐波系数到 data 中，代码为：

```
m = max(F);
tmp = find(F>m*0.4);
s = tmp(1);
data = zeros(2,10);
i = 1;
while i <11
    p1 = round(i*s-s/2);
    p2 = round(i*s+s/2);
    tmp = F(p1:p2);
    m_1 = max(tmp);
    pos = find(tmp==m_1);
    data(1,i) = m_1;
    data(2,i) = pos+s*(i-1)+s/2;
    i = i+1;
end
data(2,:) = data(2,:)/f_l*fs;
stand = data(1,1);
data(1,:)=data(1,:)/stand;
```

结果为：

1	2	3	4	5	6	7
1	1.4572	0.9587	1.0999	0.0523	0.1099	0.3589
329.6039	658.8220	988.0401	1.3173e...	1.6465e...	1.9757e...	2.3049e...

2.4 自动分析音乐节拍和音调

代码见 codes/ex2/ex2_4.m; find_res.m;define_pat.m

对于这道题，主要有三个问题：分割音符、提取基频、分析音符。

a. 分割音符

我认为这其实是一个复杂的问题，基本思路是每一个音符开头都有一段阶跃，可以考虑搜寻局部最大值，当局部最大值由小变大时就代表可能是一个新的音符开头。这样先便利一遍可以保证所有

音符开头一定都在结果中。

代码如下，结果见下图 step1.

```
rate = 600; %采样率
treat = element;
l = length(treat);
i = 1;
pos = [];
pos_1 = 0;
pos_2 = 0;
m_1 = 0;
m_2 = 0;
while i<l-rate
    pos_1 = pos_2;
    m_1 = m_2;
    tmp = treat(i:i+rate-1);
    pos_2 = find(tmp==max(tmp));
    m_2 = tmp(pos_2);
    pos_2 = pos_2 + i-1;
    if(m_2>m_1)
        pos_l = length(pos);
        pos = [pos;pos_2];
    end
    i = i+rate;
end
```

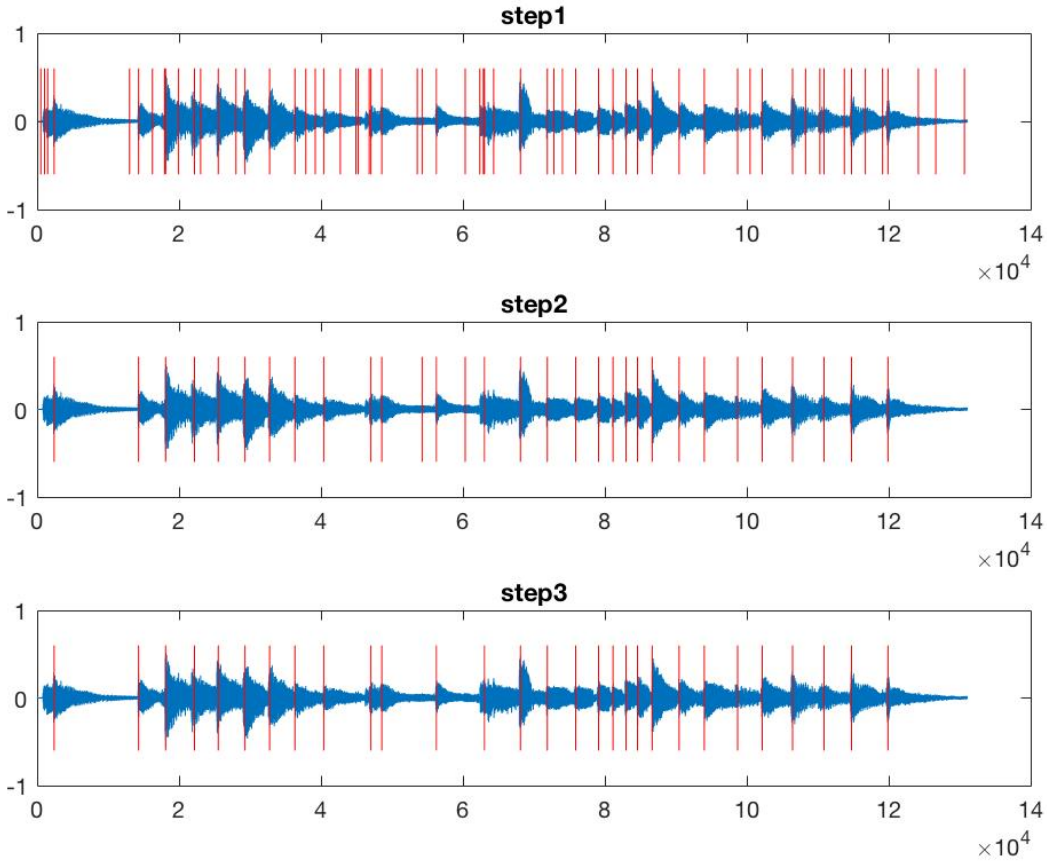
rate=600 是移动步长，为了保证所有点都在结果集合中，需要满足 rate<2000/3，其中 2000 是最短音符时间长度。

第二步是要对结果做修正，可能有一些音符衰减过程中的局部最大值也被包括进来，我的判断方法是音符的开头一定至少是以它为中心，长度为 4000 的信号的最大值。考虑时长误差，我设置这个值为 3000。代码如下，结果见下图 step2：

```
len = length(pos);
res1 = [];
for k = 1:len
    if (pos(k)<1500)
        continue;
    end
    if(pos(k)+1500>l)
        break;
    end
    tmp = treat(pos(k)-1500:pos(k)+1500);
    if(find(tmp == max(tmp))==1501)
        res1 = [res1,pos(k)];
    end
end
```

在这之后会发现还有两个错误值存在，原因是在长音符中中间段的局部最大值也满足上述条件。

为了处理这个问题，我尝试了很多算法，大部分算法在消除这两个错误点时也消除了一部分正确点。我并没有想到普适算法，即使可以得到正确结果也仅仅是只能针对这段音乐，不具有普遍性。于是我最终决定手动去除这两点。结果如 step3。



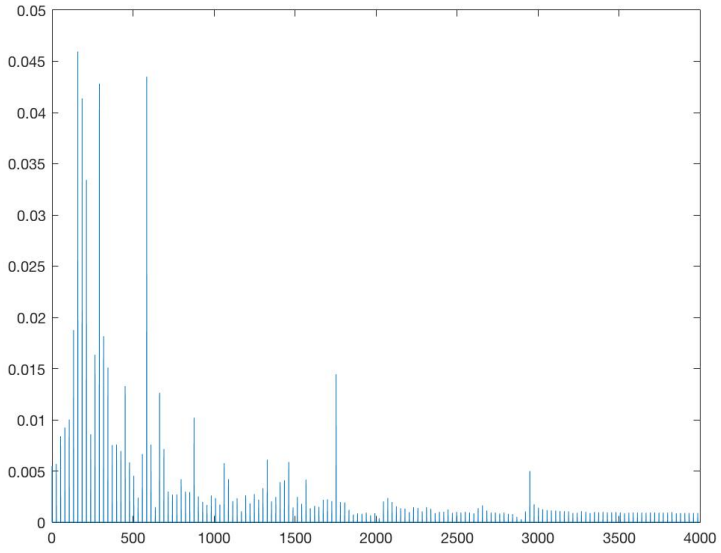
b. 提取基频

这部分代码基本上与上一问相同。取每个音符最前的 300 个点，周期延拓 1024 次，找到第一个大于全局最大值 40%的点。用这个算法得到下面的结果：

```
%{
m = max(F);
tm = find(F > m*0.4);
s = tm(1);
freq = s/l*fs;
%}
```

```
(1)  8    G   199.0438
(2)  2+4  bA  212.6505
(3)  4    bG  186.0725
(4)  4    bG  186.0725
(5)  4    bA  212.6505
(6)  4    bE  318.9628
(7)  4    bG  186.0725
(8)  4    F   159.4944
(9)  4    F   132.9163
(10)  4+8  F   159.4944
(11)  8    bA  212.6505
(12)  2    bA  212.6505
(13)  4+8  bE+ 637.8997
(14)  4+8  F   159.4944
(15)  4    E+  664.4778
(16)  4    bA  212.6505
(17)  4    bA  212.6505
(18)  8    C   265.8067
(19)  8    F+  345.5409
(20)  8    E+  664.4778
(21)  8    bA  212.6505
(22)  4    C   265.8067
(23)  4    bA  212.6505
(24)  4    bB  239.2286
(25)  4    F   132.9163
```

出现连续的 F，G 显然与原信号不符合。观察到一些音符的 FFT 图像如下：



这个算法显然是有问题的。观察全部波形后，我找到一个规律如果最大值在 550 以上，那么它一定不是基频，但是一定是基频的 2 倍或 3 倍。反之则一定是基频。用这个算法，我得到了不错的结果。并将这部分代码封装成函数 find_res.m，返回值是基础频率和谐波系数数组。

```
m_pos = find(F==max(F));
if (m_pos/l*fs>550)
    f1 = max(F(round(m_pos/2)-50:round(m_pos/2)+50));
    f2 = max(F(round(m_pos/3)-50:round(m_pos/3)+50));
    if(f1>f2)
        m_pos = round(m_pos/2);
    else
        m_pos = round(m_pos/3);
    end
end
freq = m_pos(1)/l*fs;

s = m_pos(1);
data = zeros(1,10);
i = 1;
while i <11
   p1 = round(i*s-s/2);
   p2 = round(i*s+s/2);
   tmp = F(p1:p2);
   m_1 = max(tmp);
   pos = find(tmp==m_1);
   data(i) = m_1;
   i = i+1;
```

结果如下，基本正确，但有些音符还是存在问题，例如有个别音符频率谱最大值在 5 次谐波上。改进方法是在前一步找全局最大值就在 600Hz 以下找。结果如下：

(1) 0.5 G 199.0438
(2) 3.0 bA 212.6505
(3) 1.0 bB 239.2286
(4) 1.0 bA 212.6505
(5) 1.0 D 292.3848
(6) 1.0 E 332.2519
(7) 1.0 bG 186.0725
(8) 1.0 bA 212.6505
(9) 1.0 F 159.4944
(10) 1.5 D 292.3848
(11) 0.5 bA 212.6505
(12) 2.0 E 332.2519
(13) 1.5 E 332.2519
(14) 1.5 bA 212.6505
(15) 1.0 A 221.5012
(16) 1.0 E+ 664.4778
(17) 1.0 bA 212.6505
(18) 0.5 G+ 398.6971
(19) 0.5 F+ 345.5409
(20) 0.5 E 332.2519
(21) 0.5 D 292.3848
(22) 1.0 A 221.5012
(23) 1.0 bB 239.2286
(24) 1.0 D 292.3848
(25) 1.0 C 265.8067
(26) 1.0 F 159.4944
(27) 1.0 bA 212.6505
(28) 1.0 bA 212.6505
(29) 1.5 bA 212.6505
(30) 3.0 bA 212.6505

c. 分析音符

这部分代码比较简单，只要打表计算距离最近即可。将此功能封装在 define_pat.m 中，接受的输入是基频和信号长度，输出是节拍和音调。其中，节拍 0.5 表示四分音符；音调‘+’表示高音。

三、 基于傅立叶级数的合成音乐

3.1 用(7)算出的级数演奏东方红

代码见 codes/ex3/ex3_1.m

这里用到了 ex2_3 保存的谐波数据。

本题没有什么难度，做出的一点小的改进就是用循环来代替打表。代码如下：

```
for i = 1:14
    tt0 = sin(2*pi*freq(list(i))*t0/fs);
    tt1 = sin(2*pi*freq(list(i))*t1/fs);
    tt2 = sin(2*pi*freq(list(i))*t2/fs);
    for j = 2:10
        tt0 = tt0+data(1,j)*sin(2*j*pi*freq(list(i))*t0/fs);
        tt1 = tt1+data(1,j)*sin(2*j*pi*freq(list(i))*t1/fs);
        tt2 = tt2+data(1,j)*sin(2*j*pi*freq(list(i))*t2/fs);
    end
    tt0 = tt0.*cover0;
    tt1 = tt1.*cover1;
    tt2 = tt2.*cover2;
    music_base0 = [music_base0;tt0];%2分
    music_base1 = [music_base1;tt1];%4分
    music_base2 = [music_base2;tt2];%8分
end
```

听起来像吉他的音色，但是仍然有一些木讷，原因应该在于是用了同一种谐波系数描述全部音符。

3.2 用 2_4 分析出的音调演奏东方红

这一题与前一题没有本质的区别，但是由于音色不足且没有整理出顺序，没法描绘全部音符，只选择了与东方红前两小节有关的。且由于东方红大部分是高音，分析出的结果无法完全表达，我考虑低音和高音的谐波系数一致，即高音 C 用低音 C 的谐波系数来代替。

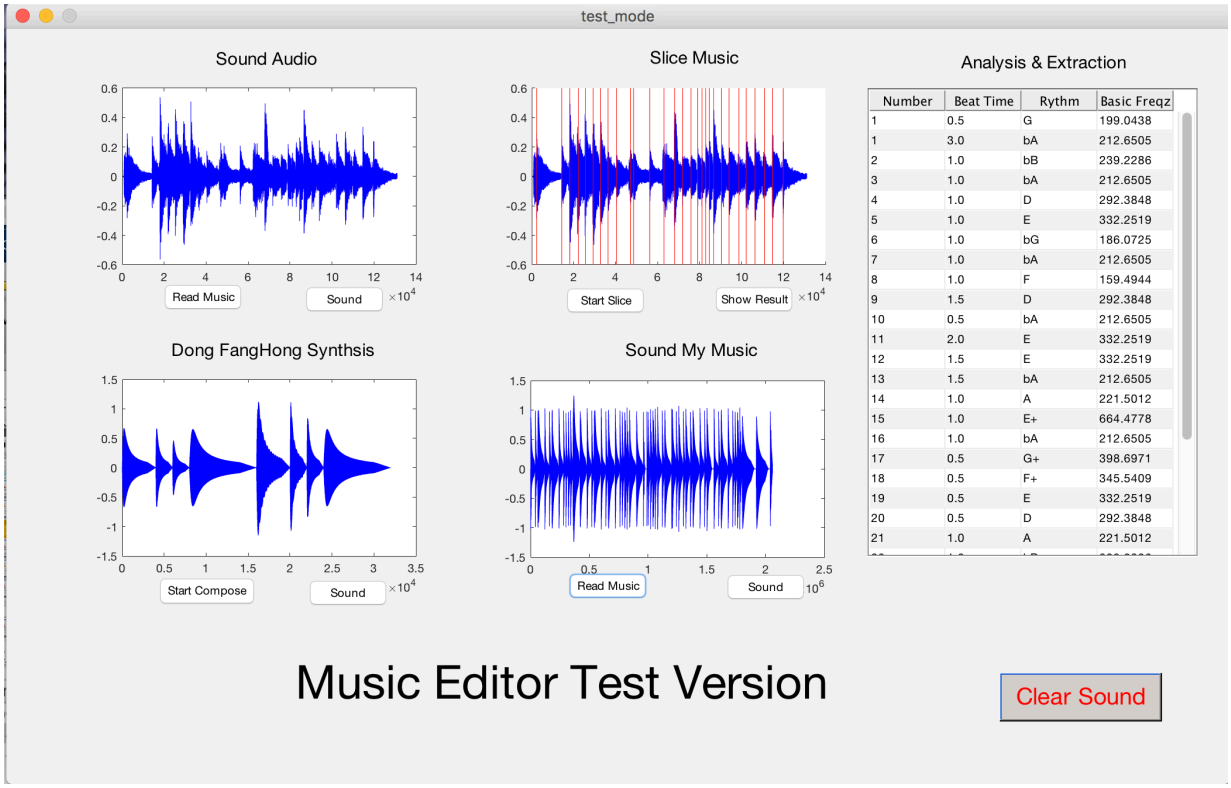
从结果上来看，效果较好，尤其是两段短音可以明显听出吉他的弦的声音。

```
list = [10,13,15,20,22]; % bD,F,G,C,D
load ex2_4.mat;
base = [];
%近似认为高音和低音谐波分量权重一致
%C,hC,hD,wG,F,hF,hbD,wG

base(1,:)=(data(5,:)+data(10,:)+data(21,:))/3;
base(2,:)=(data(9,:)+data(26,:))/2;
base(3,:)=data(18,:);
base(4,:)=data(25,:);
base(5,:)=data(19,:)*1.2;
%base(5,:)=base(1,:);
```

3.3GUI 的制作。

代码见文件 test_mode.m; test_mode.fig

这一题中用到的代码与之前没有什么本质区别，由于我编写的部分程序并非是普适性的，用代码跑也只能调出特定结果，所以有一部分 button 直接用了 load 操作来读取数据。



这个 GUI 共包含 4 个功能，第一个功能是读取音乐，这个功能是普遍的。可以读取工作目录下的所有可读取的文件，我在 ex3 中放了两个测试文件，test.mp4 和 t1.wav，输入文件名都可以播放。

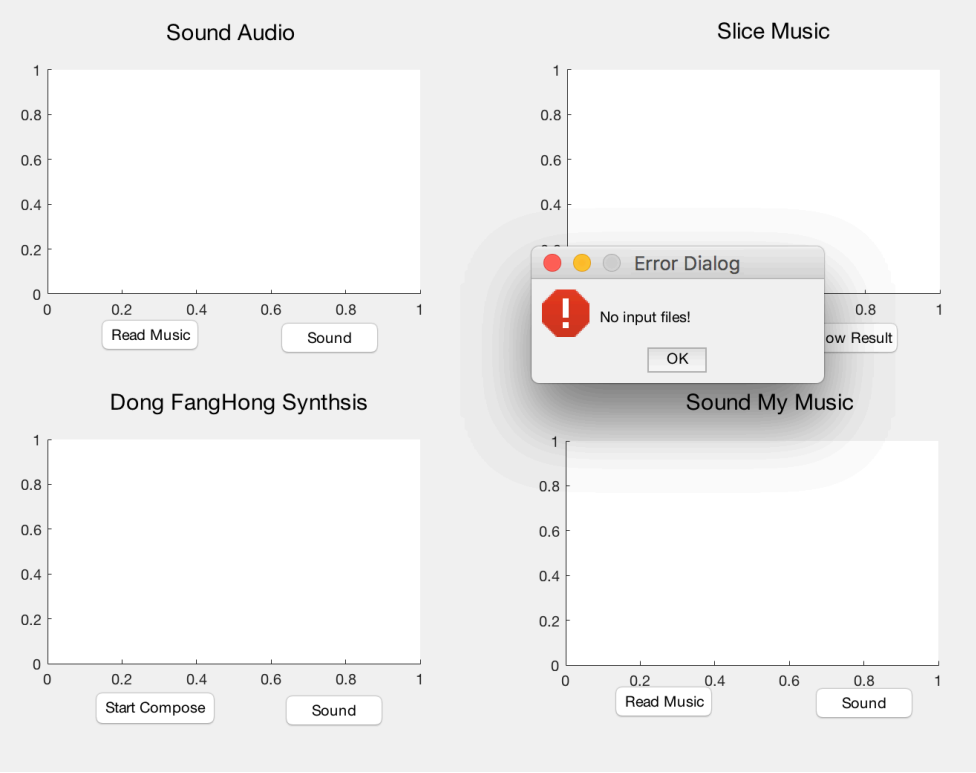
第二个功能是音乐分割和分析；对于读取的音乐进行 slice，slice 之后结果显示在右边。

第三个功能是用分析出的音乐演奏东方红，如左下角所示。

最后一个功能是演奏我所合成的音乐《雪绒花》。

第二三个功能是特定的，所以用了 load 操作。

代码部分没有特殊要说明的，用到的主要是函数 setappdata 和 getappdata 在不同 button 之间传递数据和控制信号。如果控制信号为空或者没有接受的数据，会提示用户先进行之前的操作，起到一个异常处理的作用。



此外，在调试的时候发现一直播放音乐很难受，于是加入了一

个按钮用来停止播放音乐。

四、 思考和体会

这个大作业是我独自完成的首个整体性工程，可以说在本学期之前我完全没有接触过 Matlab，本学期的接触也仅仅是停留在仿真画图的层次，经过信号与系统、音乐合成这两个大作业的洗礼，感觉到 Matlab 真的是一个很好用的工具，把 C++，python，终端操作的优点全部汲取到，熟悉这些语言的话很容易上手。

此外，结果信号与系统语音拼接大作业感觉这个大作业总体来说难度不大，比较困难的点在于 GUI 的编写，参考的资料不多而且在网上找的教程很多代码在 R2016 版本上无法运行，只能找博客上的代码设置断点一条一条运行，终于慢慢摸索出各个操作，也被 GUI 各个控件间的信息传递机制所折服。

还有一点不满足应该就是在于分析音乐部分要用到不少信号与系统的知识，但这部分知识学期中就掌握的不是很好，导致在处理其问题的时候傅里叶变换的性质都要翻书参考，很尴尬。

倘若还有遗憾的，那就是 ex1_5 自己合成的音乐不够满足，还有一些想要添加操作，如混音、颤音、音尾融合等由于时间关系都没有加进去。上交大作业后会继续完善！

结果这个大作业后，真的体会到了老师说的：信号与系统

很有趣！