Matlab大作业——音乐合成实验报告

无63　　林仲航　　2016011051

1、简单的合成音乐

（1）

每个乐音的频率为：

1	2	3	4	5	6	7	i
349.23	392.00	440.00	466.16	523.26	587.33	659.25	698.46

在MATLAB代码中，将简谱乐音表示为一个序列，且持续时间也为一个序列。通过自定义的calc_f函数计算出每个乐音对应的频率。定义歌曲序列song来表示由8000Hz采样频率采样下的歌曲。在每个乐音对应的时间区间内载入不同频率的正弦信号，从而合成出一段简单的音乐。

关键代码如下：

```
song_tone = [5,5,6,2,1,1,6,2];          %歌曲乐音序列
last_time = [1,0.5,0.5,2,1,0.5,0.5,2]; %对应的每个乐音持续拍数
music_nums = length(song_tone);         %乐音数目

begin_time = 0;                         %每个乐音的开始时间
for i = 1:music_nums
    f = calc_f(song_tone(i),basic_f)
    range = (t>=begin_time & t<begin_time+last_time(i)*one_step);           %对应乐音所在的时间范
围
    song = song +  range * A .* sin( 2*pi*f.*(t-begin_time));               %对应范围加载正弦信号
    begin_time = begin_time + last_time(i)*one_step;
end

sound(song,fs);
```

音乐总体效果上，每个乐音间较不连续，且存在的'啪'的杂音。降低要求来看，还是可以听出歌曲的调的。

（2）

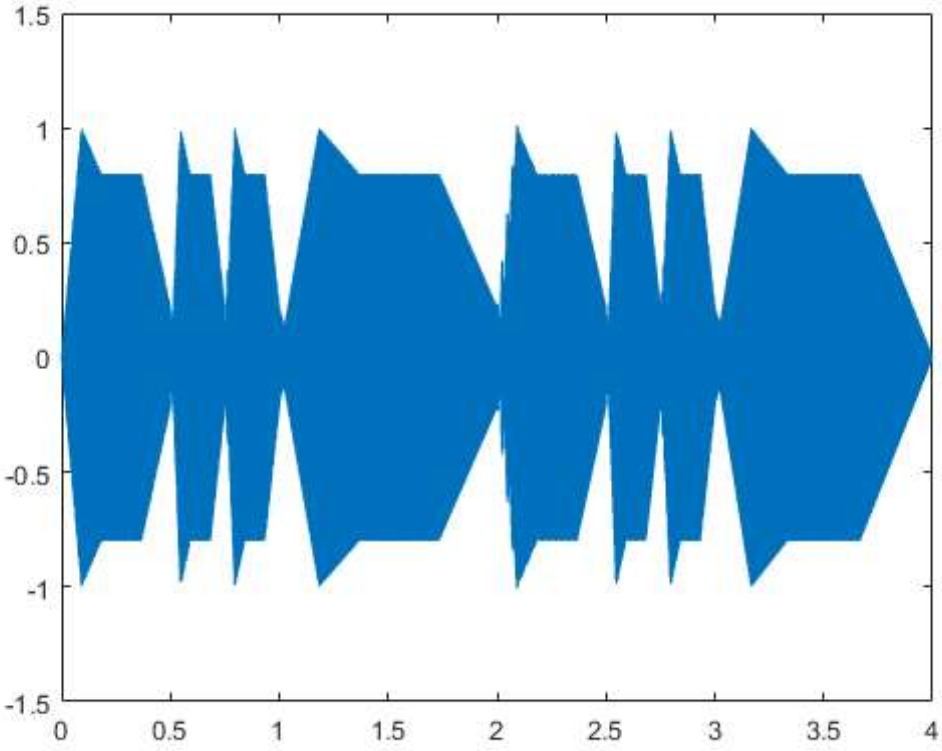
通过采用包络来修正乐音，且使得上一个乐音的衰减段与下一个乐音的冲激段相叠加，的确消除了相邻乐音间的杂音。在MATLAB代码中，乐音的正弦信号产生基本同（1），只是持续时间稍微延长至原来的1.1倍，使得相邻乐音间存在重叠。随后通过自定义函数A_adjust来对该乐音的幅度进行调整。采用了两种调幅方法：1.课本上的分段线性函数；2.一段冲激加上一段指数衰减的分段函数；分别定义为method1与method2.

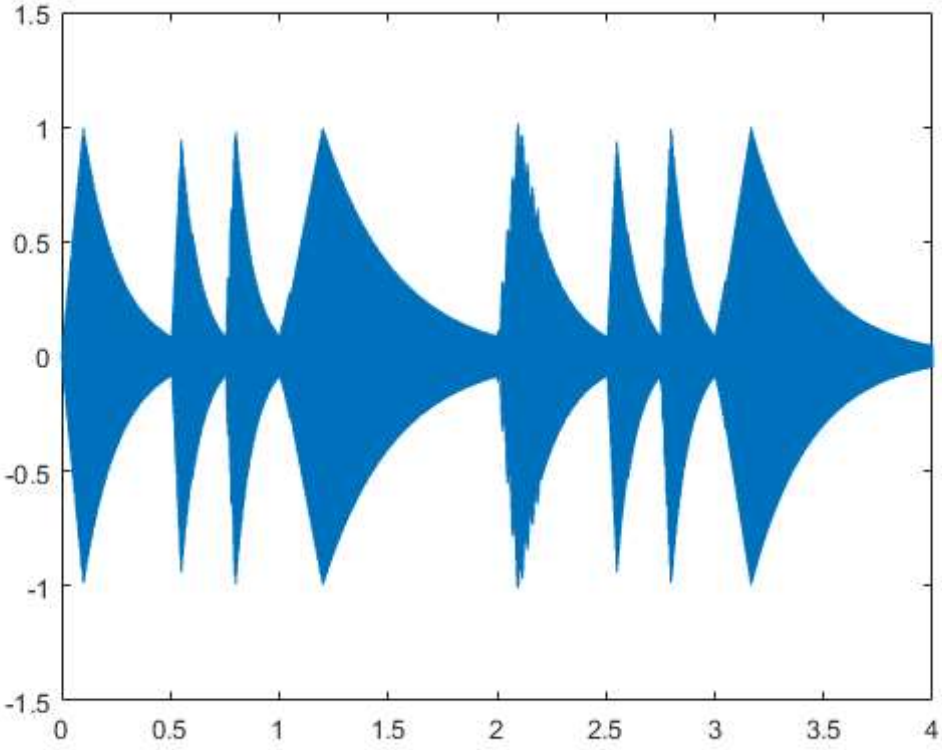
设t0为该乐音起始演奏时间，time为持续时间，函数表达式如下：

$$method1 = \begin{cases} \frac{6}{time} * (t - t0), & t0 <= t < t + \frac{time}{6} \\ \frac{-1.2}{time} * (t - t0 - \frac{time}{6}) + 1, & t0 + \frac{time}{6} <= t < t0 + \frac{time}{3} \\ 0.8, & t0 + time/3 <= t < t0 + \frac{2}{3}time \\ \frac{-2.4}{time} * (t - t0 - time), & t0 + \frac{2}{3}time <= t < t0 + time \end{cases}$$

$$method2 = \begin{cases} \frac{20}{time} * (t - t0), & t0 <= t < t + \frac{time}{20} \\ exp(-\frac{80}{19time} * (t - t0 - \frac{time}{20})), & t0 + \frac{time}{20} <= t < t0 + time \end{cases}$$

对应的东方红歌曲波形如下：





(3)

升高八度与降低八度，即将所有乐音对应的频率都进行乘2/除于2的操作。由于我制作的音乐信号为采样频率8000Hz的信号，故只需对其重新采样就行。所以只需在sound(song,fs)中将采样频率fs更改为16kHz/4kHz，即可实现升高/降低八度。

```
在命令行输入：
    sound(song,16000);
    sound(song,4000);
```

同样，要将音乐上升半个音阶，对应每个乐音的频率需乘(2^1/12)；故而只需使用resample函数，以8000*1.0595Hz采样即可，得到新音乐序列new_song。将new_song以默认频率播放，相当于原歌曲在时间上被拉长。

关键代码为：

```
new_fs = 8000 * 1.0595;              %新频率
[new_song,new_t] = resample(song,t,new_fs); %以新的频率采样
sound(new_song,8000);           %在默认8000Hz下播放，相当于歌曲时间拉长
```

(4)

通过在给每个乐音段赋值正弦函数时，加上2次、3次谐波分量，从而使得合成出的音乐有谐波分量。通过给不同谐波赋予不同权重来控制该分量的占比，从而可以产生不同的效果。同时，在权重上加入一个较小的随机数来使得乐音更加自然（产生了随机性）。

关键代码如下：

```
weight = [1,0.3,0.1];                          %各谐波分量
range_song = range .* ( (weight(1)+0.005*rand(1))*sin( 2*pi*f.*(t-begin_time)) +
(weight(2)+0.005*rand(1))*sin( 4*pi*f.*(t-begin_time)) +  (weight(3)+0.005*rand(1))*sin(
6*pi*f.*(t-begin_time)));                %加入随机数来模拟真实环境，该段范围内的音乐信号,加入谐波（未进行包络处理）
range_song1 = A_adjust(range_song,t,begin_time,lasting_time(i)*one_step*end_mul,method);    %包络处理
```

从音色上看的确比原有的单频乐音丰富了许多，类似于管弦乐器。在刚开始时未将音乐信号幅度归一化，导致出现杂音。归一化后音色正常，但与正常乐器相比仍有差距。

(5)

我选择了Canon的一个较为简单的简谱进行合成。由于我的程序只需将简谱转为各个乐音对应的数字以及其持续的时间，即可进行合成。故而工作量只在于将简谱转换为程序所需的序列即可。

由于Canon歌曲有两个声道，故我合成了两个信号song与song2。将这两个信号进行加和后归一化，即可倾听美妙的Canon乐曲。

为了修改音色使得音乐有更好的效果，在包络上采取了更多的函数进行拟合，但总体效果上依旧和真实乐器有着较大差距。

关键代码如下：

```
song = song_creation2(song_tone,last_time,one_step,fs,basic_f,3);%生成音乐
song2 = song_creation2(song_tone2,last_time2,one_step,fs,basic_f,3);     %生成第二个声道的音乐
Canon_song = song + 0.3*song2;                          %两通道合成
Canon_song = Canon_song/max(Canon_song);                %限制幅度,归一化
sound(Canon_song);
```

通过手动输入音色与时间序列实在是非常麻烦，如果可以进一步尝试的话我偏向于采用图像识别分析简谱。

2.用傅里叶级数分析音乐

（1）

载入'fmt.wav'并读入。

```
[song,fs] = audioread('fmt.wav');
sound(song,fs);
```

效果上比刚才的合成音乐好多了，十分像真实的吉他声。

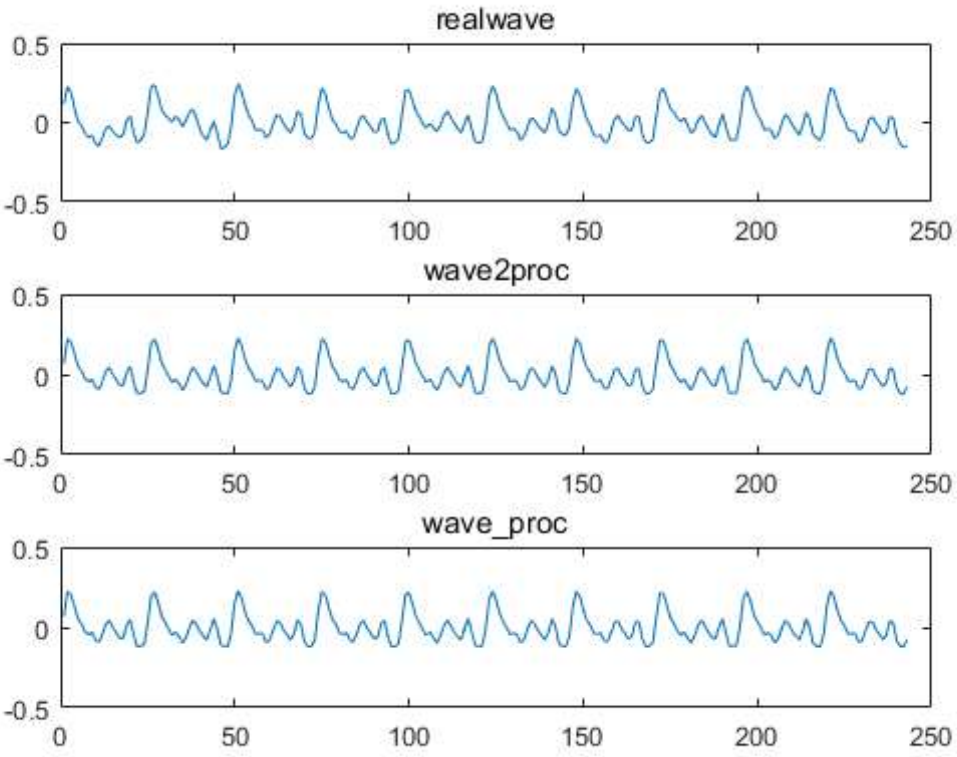
（2）

如图，可以大致看出realwave存在10个周期，若要滤除噪声等干扰，则需要用到提示中所说的时域滤波，查询资料后，决定采用平均值滤波方法。具体描述如下：

1. 重新采样使realwave信号长度为10的整数倍；
2. 将每个周期对应值相加，随后进行平均。得到经过处理的单周期信号。
3. 将单周期信号重复10次；
4. 利用resample函数进行重采样，得到与realwave长度相同的wave_proc；

   关键代码如下：

```
load('Guitar.mat');
f = resample(realwave,10,1);                          %提高采样率重采样
new_f = 0*realwave;
for k = 1:length(realwave)
    for i = 0:9
        new_f(k) = new_f(k) + f(i*length(realwave)+k);
        %对应周期的信号相加
    end
    new_f(k) = new_f(k)/10;                           %求平均
end
wave_proc2 = repmat(new_f,10,1);
wave_proc = resample(wave_proc2,1,10);                %降采样
```



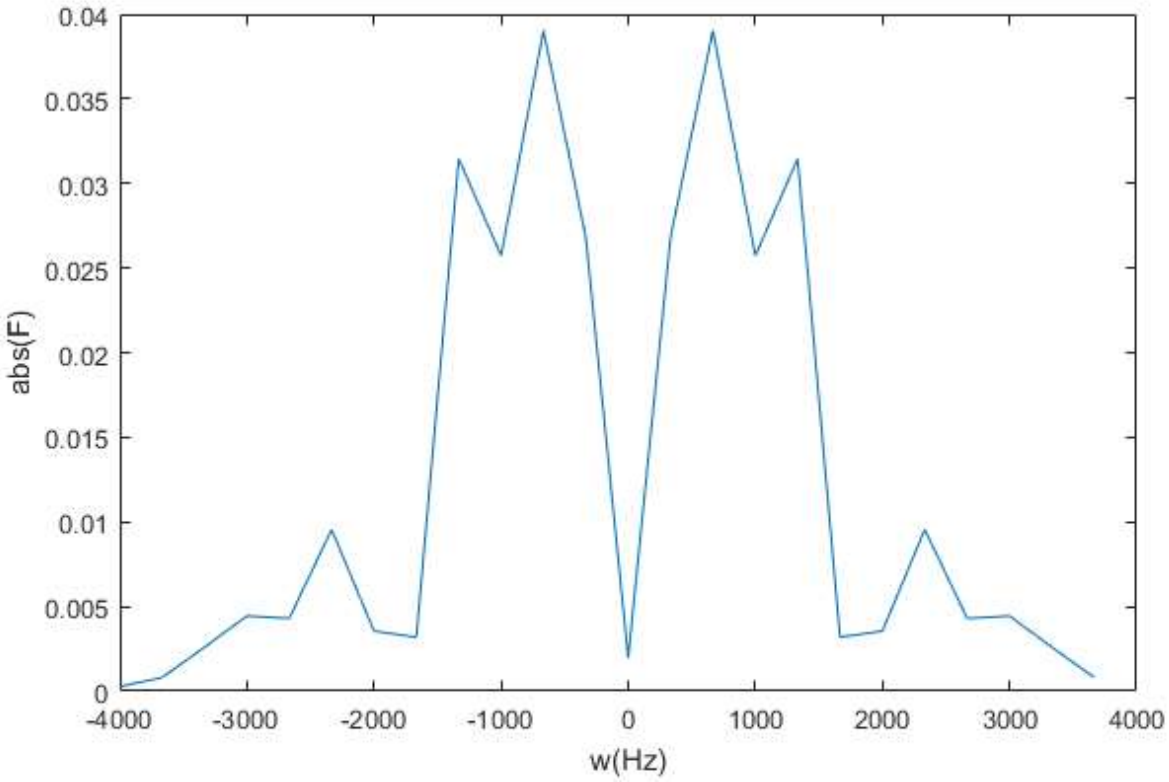
如图，可以看出经过处理的wave_proc与wave2proc十分相像；实际上，通过计算相关系数可以发现，相关系数达到1.0000。

通过绘制频谱图可以发现，经过处理后，非线性谐波分量和噪声的确被抑制了。

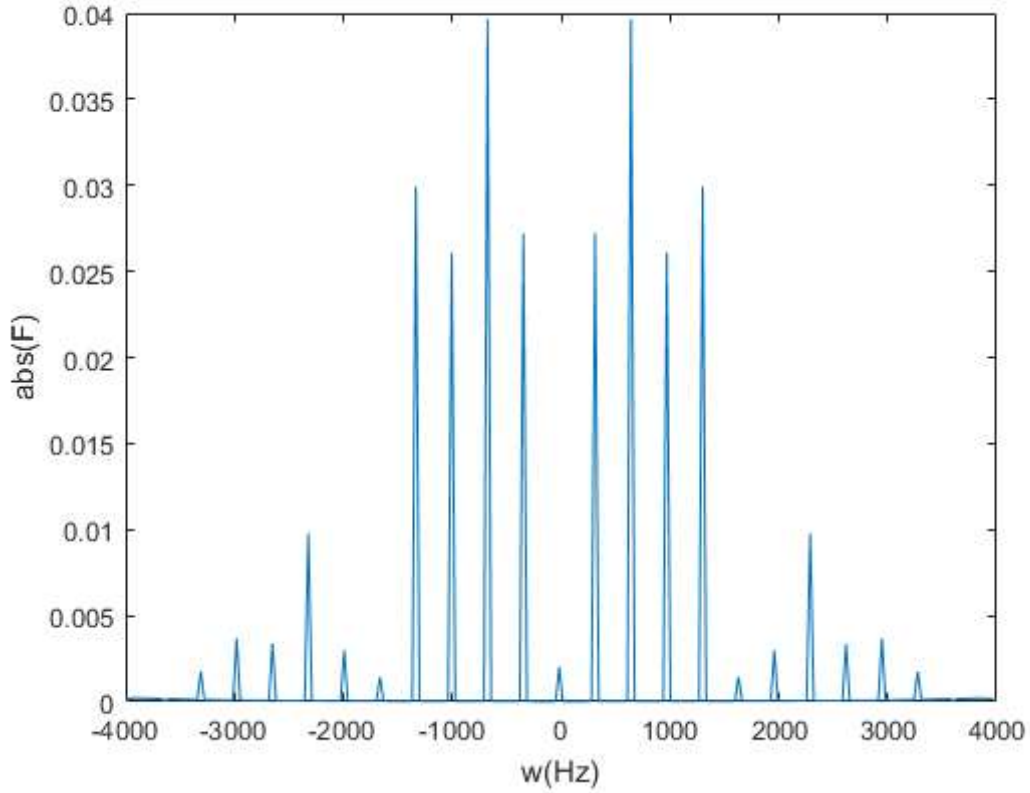
(3)

利用自定义的函数fft_analyse()对信号进行傅里叶变换，当输入单个周期（取wave2proc前24个点时），频率谱如下，非常难以辨认基频以及谐波分量。

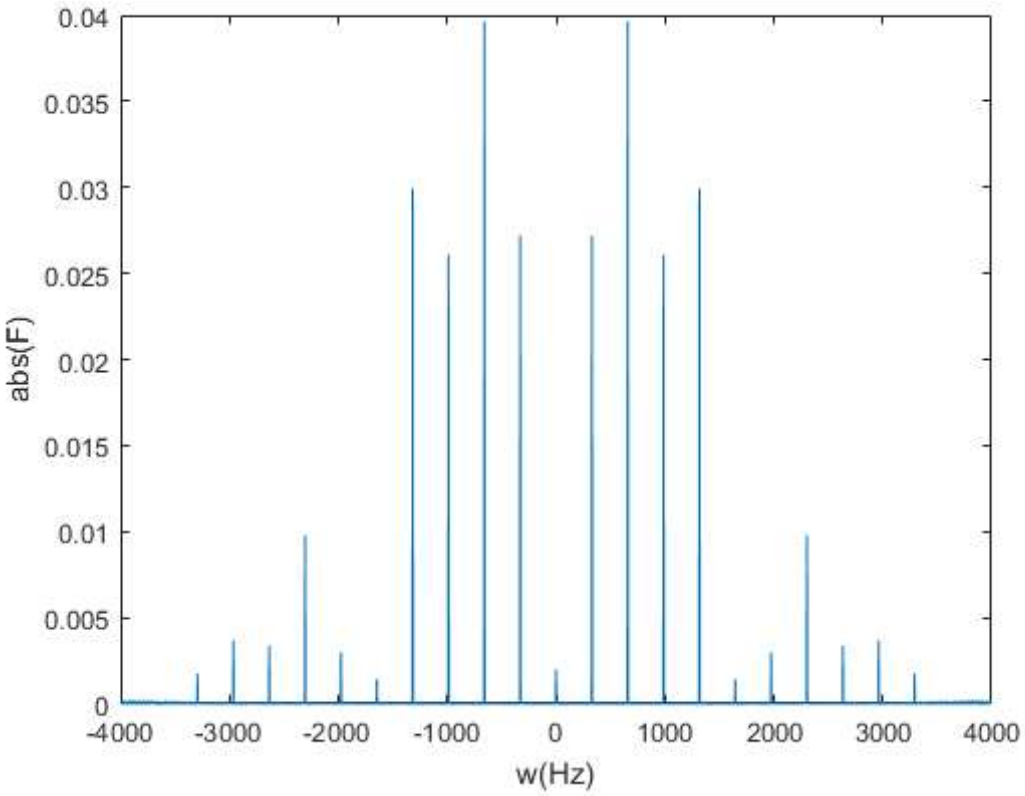
```
single_cycle = wave2proc(1:24);     %取单周期
[F,f] = fft_analyse(single_cycle,fs);%进行傅里叶变换
```



当将wave2proc直接进行fft_analyse时，可以发现，现在可以清晰地辨认出基频分量以及各次谐波分量。



当将wave2proc重复10次作为周期信号输入时，可以看到，频谱现在更加接近于冲激函数组合的形式，更加清晰可认。



通过将0点处的值设为阈值，找到所有大于阈值的频谱点，最后发现，基频为329Hz，对应的音调为E调（即C调的Mi音）。

增加时域的数据量后，相当于增加了信号的周期数，使其越发接近周期信号，而周期信号的傅里叶变换为delta函数的组合。故而更加容易分辨出音调。

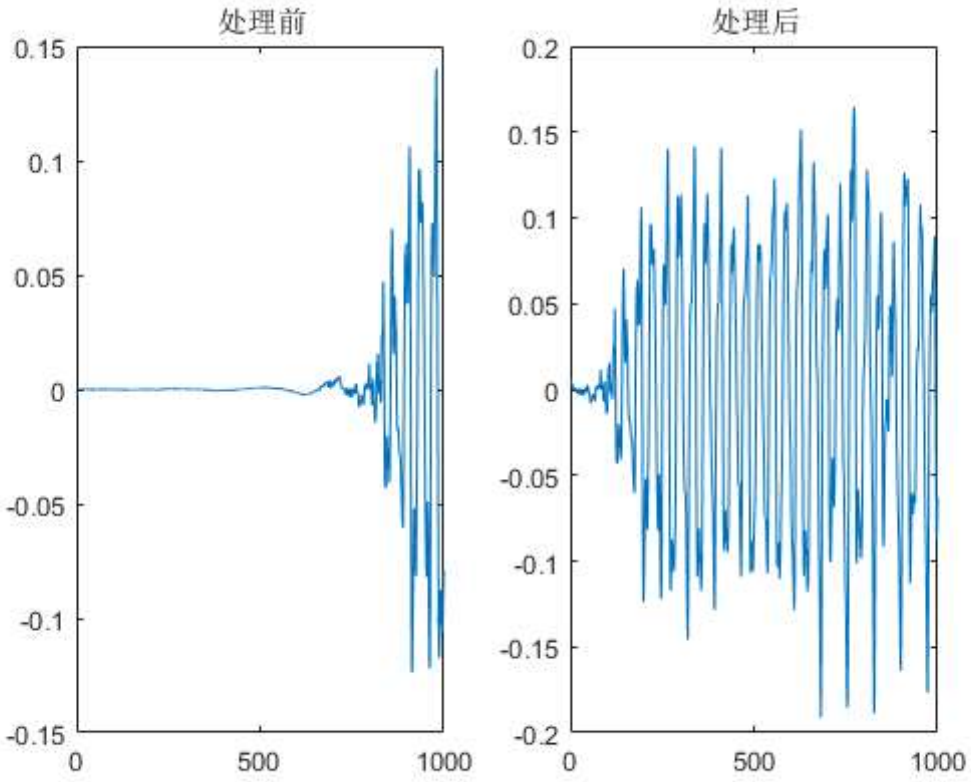
(4)

a.静音处理：

首先需要对信号开头和结尾的静音部分进行裁剪。取一个特定阈值后将信号中 静音部分去除。

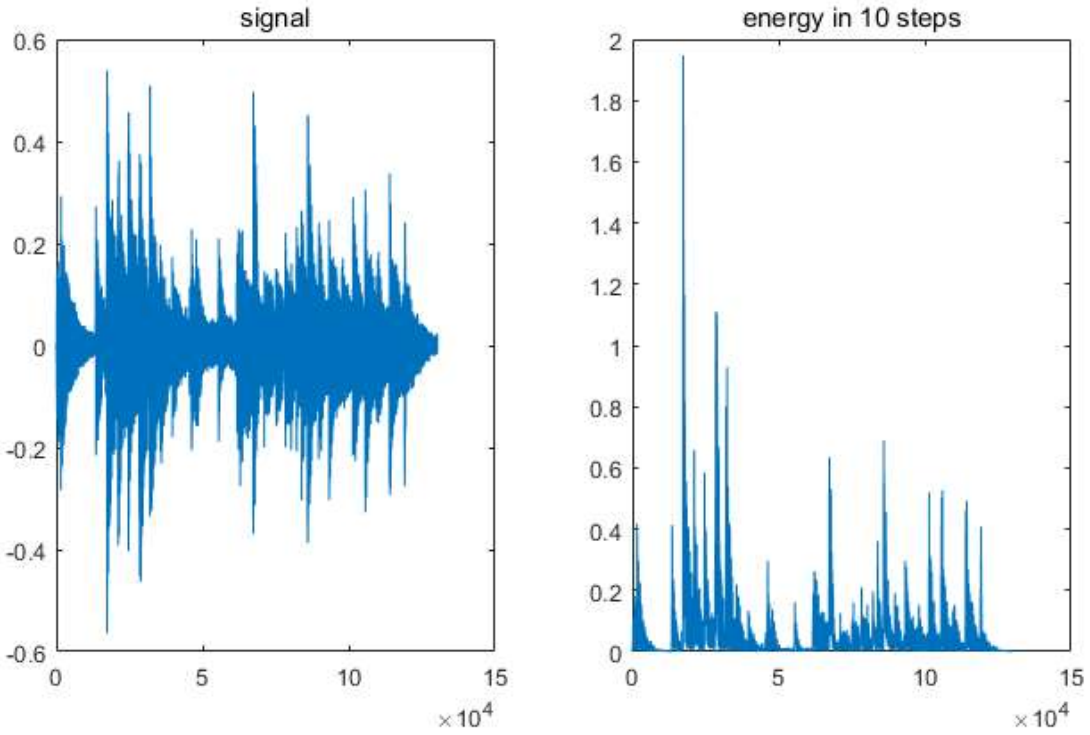
```
threshold = 0.005;              %静音阈值
A = find(fmt>threshold);        %找到所有大于阈值的index
begin = A(1);                   %取第一个index
final = A(end-1);               %取最后一个index
fmt = fmt(begin:final);         %定义为信号有效范围
```

效果如下：



b.寻找每个乐音开始的时间

从绘制出的音乐波形来看，每个乐音开始阶段都有一个冲激阶段。所以只要找到这个冲激阶段就可以大致知道乐音的开始时间。在这里我采用近似算每25个点的能量来寻找冲激段。通过自定义函数calc_energy，可以得到一个粗略的能量谱，绘制出的曲线如下：

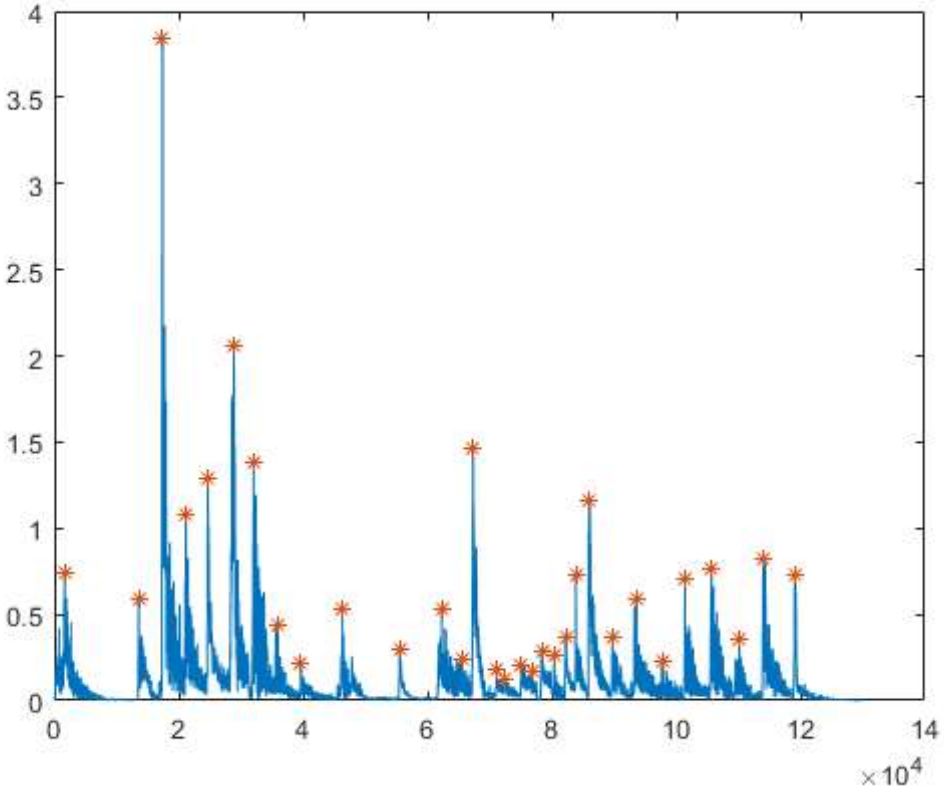


从图中可以明显看出能量谱极值与signal中每个乐音开始的冲激段相对应，故只需找到冲激段的位置即可。故而考虑通过遍历来寻找这些尖峰所在点。故而考虑以下算法来进行计算：

1. 设定一个步长pace，在信号中每次循环都选取pace个值中的最大值来代表该段的值，并写入max_set数组中；
2. 设定一个阈值threshold去除掉不符合条件的点；
3. 在max_set中寻找满足如下条件的点：在邻域内为极大值点，且其幅度大于阈值。这些点即为冲激点；
4. 找到冲激点对应的下标，从而确定每段开始的时间。

以上步骤包含在自定义函数find_localmax函数中，关键代码与效果图如下：

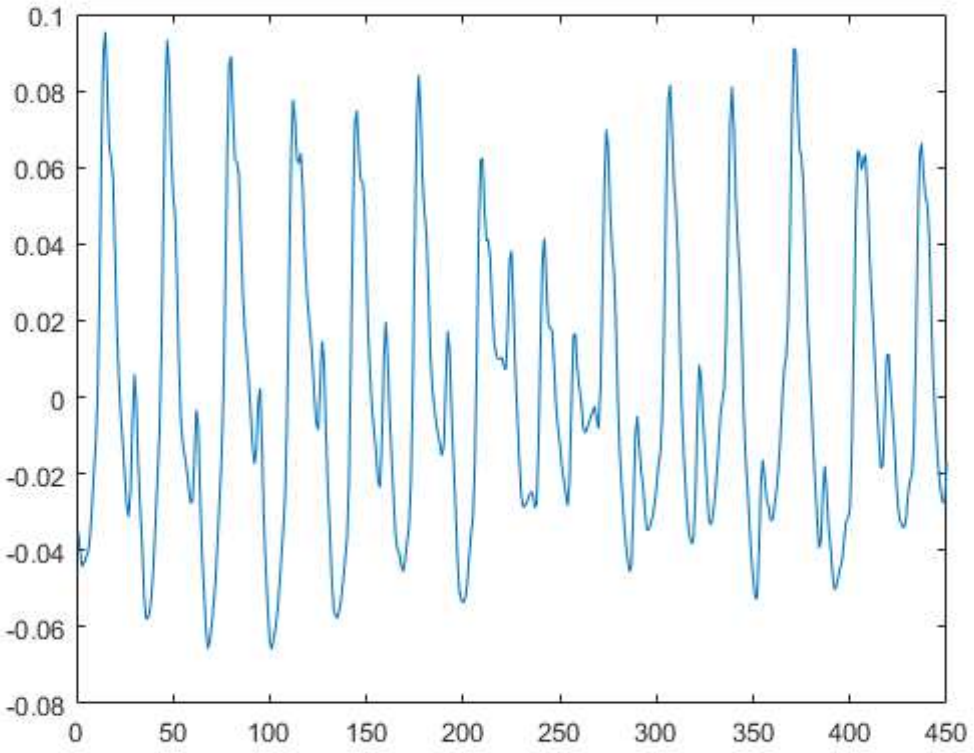
```
%寻找每一步的最大值点及其对应的下标并加入数组中
for i = 1:time
    if(i*pace > L)
        [this_max,this_max_index] = max(signal((i-1)*pace+1:L));
    else
        [this_max,this_max_index] = max(signal((i-1)*pace+1:i*pace));
    end
    this_max_index = this_max_index + (i-1)*pace;
    max_set = [max_set,this_max];
    max_index_set = [max_index_set,this_max_index];
end
%找出拐点
for i = 2:length(max_set)-1
    if(max_set(i)>max_set(i-1) & max_set(i)>max_set(i+1) &max_set(i)>threshold)
        real_point = [real_point,max_index_set(i)];
    end
end
```



可以看到该方法很好地确定了所有冲激点的位置，从而有利于下一步的分析；存在的不足是pace与threshold的值都需要手动进行设置，以便寻找到最佳参数。

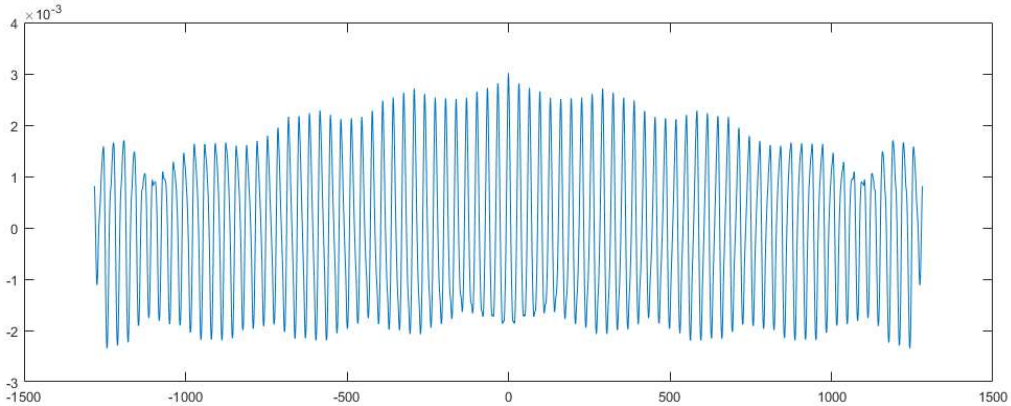
c.寻找每个乐音中隐藏的周期信号

通过截取某段乐音中的一部分信号，可以发现其中存在着周期性信号。如下图：

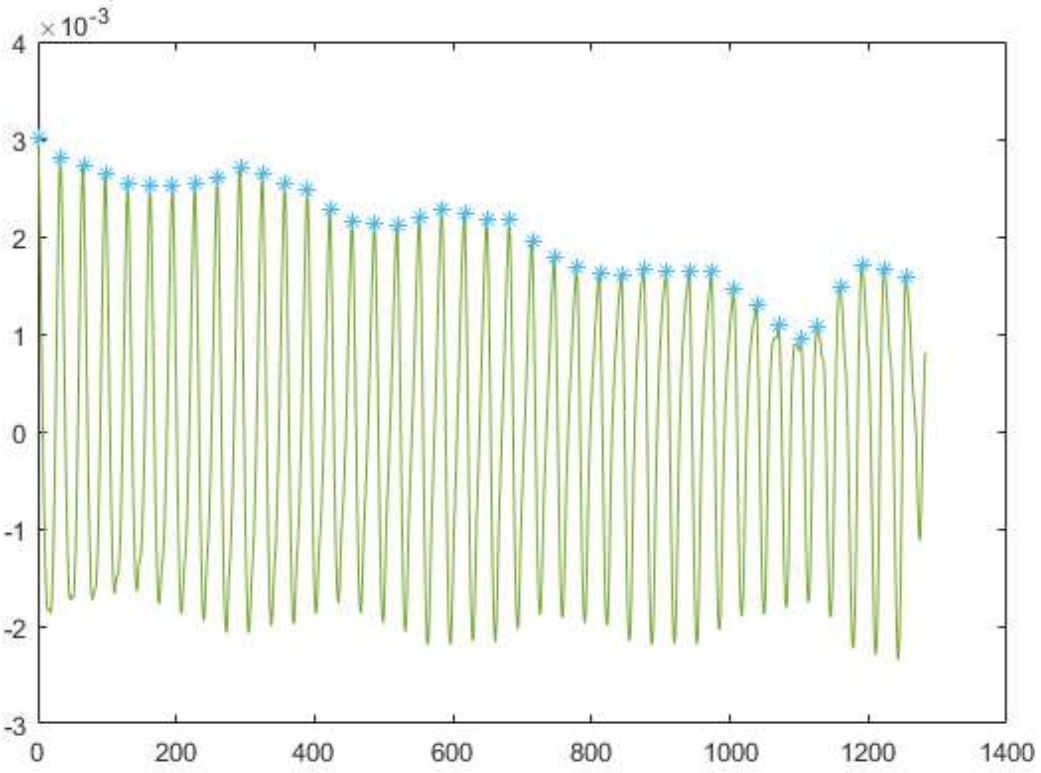


故而考虑从每段乐音信号中取一段进行分析。简单起见，可以选取每一段三分之一到三分之二的点作为要处理的信号。进而可以采用上一题中对信号的处理方式进行处理。但在这之前，需要确定信号的周期。考虑采用计算信号的自相关函数的方法，从而根据极大值间的距离得到大致的周期。

如图，通过MATLAB的xcorr函数计算信号的自相关函数，可以看出该信号存在明显的周期性，故只要求出极大值点之间的平均间隔，即可求出周期。



寻找极值点间的距离，依旧用到之前定义的find_localmax函数，设置步长为10，同时设置阈值为自相关函数最大值的0.3倍。找到的坐标点效果如下：

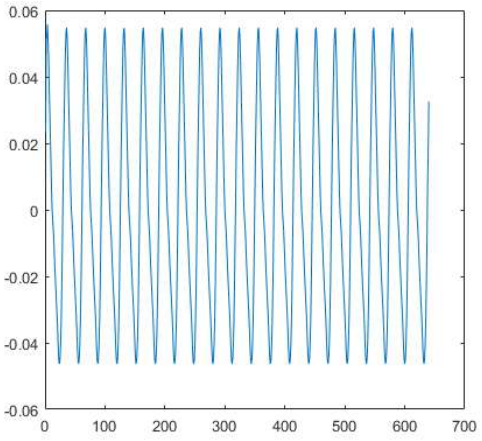


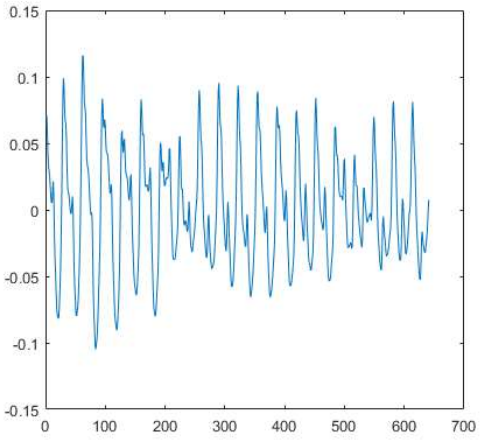
关键代码如下：

```
%先取x>=0部分的值
corr_f = corr_f(x>=0);
x = x(x>=0);

max_point = corr_f(x==0);
index = find_localmax(corr_f,10,0.3*max_point);      %寻找极值点对应下标

x_d = diff(index);       %求差
T = sum(x_d)/length(x_d);    %求平均周期
```

d.对输入信号进行去除噪声

方法与第7题相同，注意到c步中最后求出的周期T非整数，故我先对输入信号signal进行乘10倍的采样，随后近似认为新得到的信号周期为ceil(10*T)-1；随后进行平均值时域滤波，得到单个周期信号。将单个周期信号延拓，再降采样得到需要的信号。效果如图：

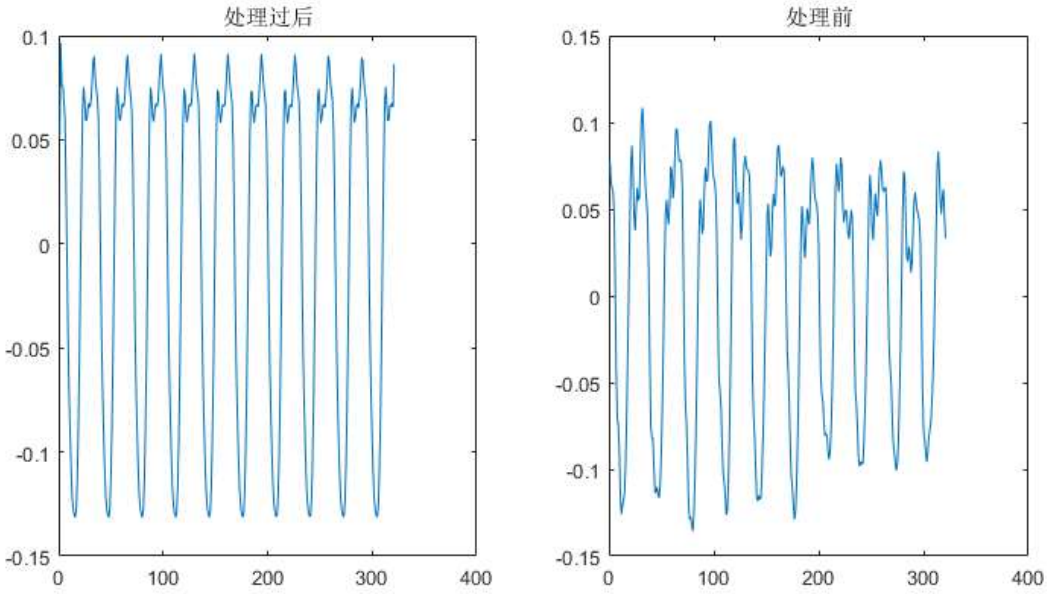




可以看出此时信号已失去幅度包络的特征，放大来看，单周期特征与原信号也不同，故该方法效果较差。应当在输入信号的基础上取较少的周期（例如10个），且将每个周期的最大值统一化，减少幅度的影响，便于分析，关键代码如下：

```
signal = signal(1:ceil(nums*T)-1);  %取整数个周期进行分析

signal2 = resample(signal,10,1);      %乘10采样
f = zeros(1,ceil(10*T)-1);            %生成单周期
multiplier = ones(1,nums);            %幅度因子
one_cycle = ceil(10*T)-1;             %10倍采样后单周期点数
%计算每个周期的修正因子
for k = 1:nums
    last = k*one_cycle;
    if(last > length(signal2))
        last = length(signal2);
    end
    multiplier(k) = max(signal2((k-1)*one_cycle+1:last))/ max(signal2(1:one_cycle));
end

for k = 1:ceil(10*T)-1
    for i = 0:nums-1
        f(k) = f(k) + signal2(i*(ceil(10*T)-1)+k)/multiplier(i+1);
    end
    f(k) = f(k)/nums;
end
%得到单周期序列f
new_f = repmat(f,1,nums);
real_signal = resample(new_f,1,10);
```

修改后的效果图如下：

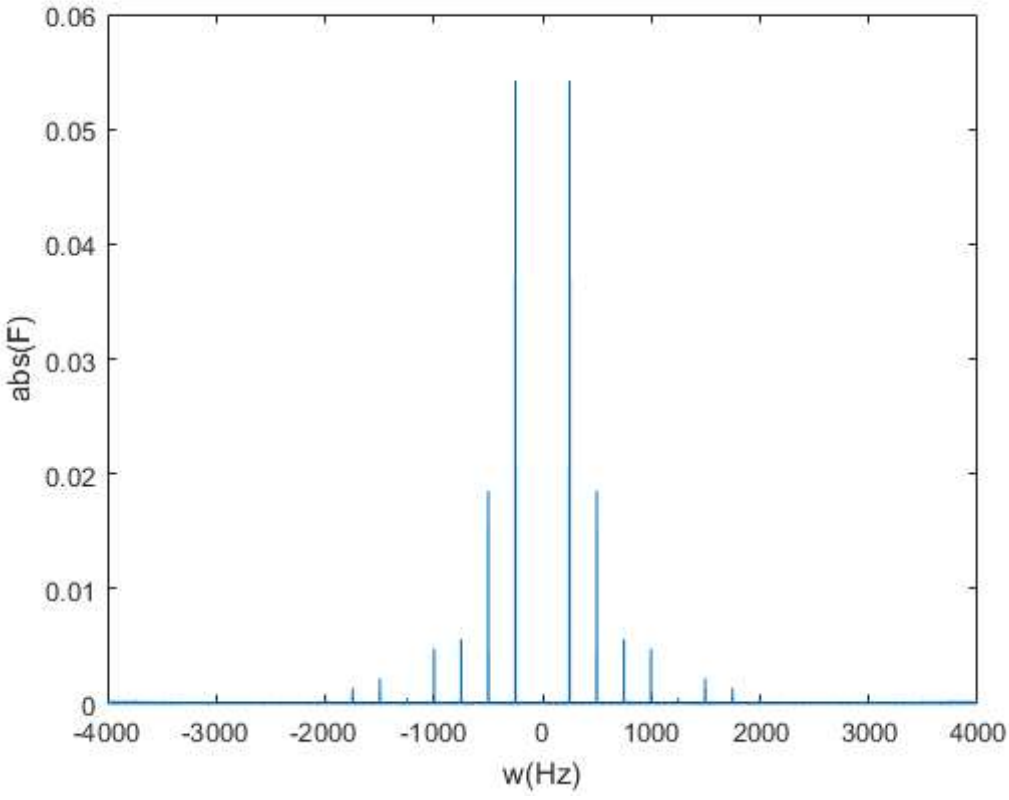


可以看出，此时单周期信号基本与处理前十分相似，且经过周期延拓后，便于进行傅里叶变换显示出冲激函数。

e.傅里叶变换及分析

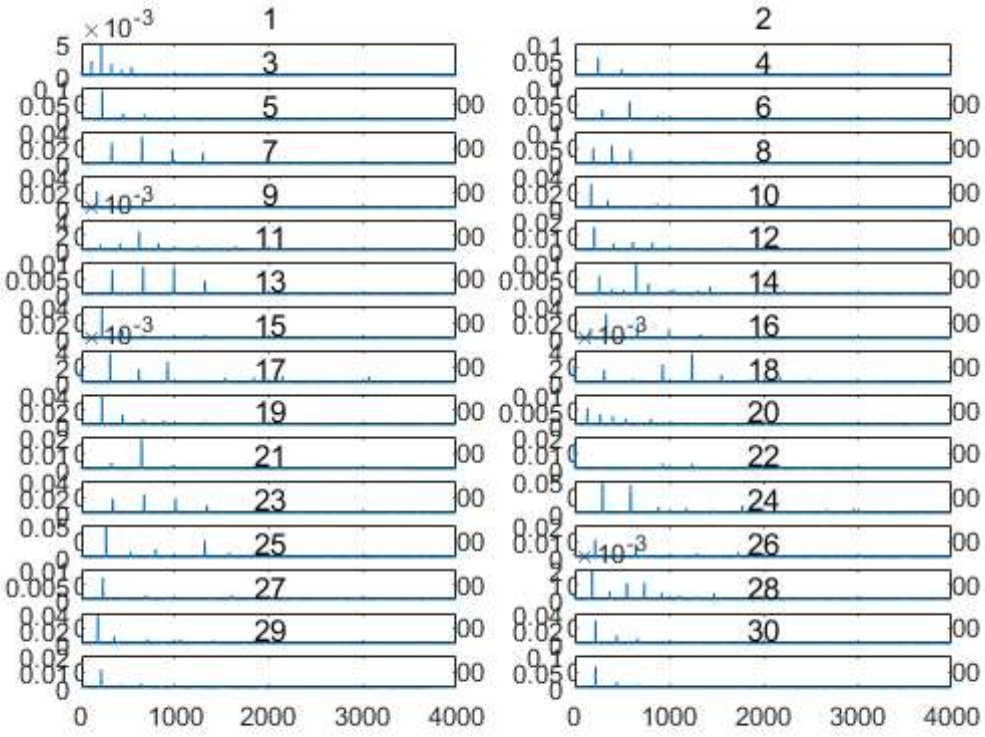
通过将上面得到的信号再次进行周期延拓后，通过自定义的fft_analyse()函数进行傅里叶变换，可以得到该周期信号的频谱。由于做了周期延拓，所以频谱十分接近于冲激函数的形式，方便了寻找基调与各次谐波的过程。

频谱图（第二个乐音的频谱图）如下：



故而我们可以得到基频、高次谐波等信息。接下来只需完成自动对整个乐曲进行处理、识别音调等工作即可。

全部乐音的频谱图如下：



f.确定乐曲的节拍

由上，我们根据冲激的特点求出了每个冲激点对应的下标，根据此下标我们可以大致计算每个乐音的持续时间。在未指明一拍的情况下，取乐曲中时长为中位数的一段乐音作为1拍，由此定义其它乐音的节拍。

```
time = [delta_time,end_time];
diff_t = diff(time);            %求每段的时间
steps = round(diff_t/min(diff_t));%得出节拍数
single_step = median(diff_t);   %选取中位数
step_time = single_step/fs;     %得出单拍的时间
index = find(diff_t == single_step);
jiepai = steps/steps(index(1));     %除于该单拍，得到其他乐音对应拍数
```

由此求得的单节拍时间为0.4688s。

g.由频谱确定基频以及各次谐波

由之前获得的频谱，通过自谦自定义的find_localmax函数处理，可以方便地得出各个极值点所在的位置等信息。接下来做的处理为：

1. 由低到高遍历这些频率点，将其对应到12个音调上；
2. 判断该频率对应的音调是否出现，若出现，则认为此频率分量为谐波。否则视为基波；

3. 创建一个矩阵储存如下信息：[音调，对应几次谐波，谐波分量强度（相对于基波分量），频率值]；矩阵每三行对应一种基波及其各次谐波的信息；
4. 将信息打印到msg.txt文档中；

   由以上大概可以确认出乐曲的音调，如下：

	节拍	音调	基波频率(Hz)
1	3	Ab	106.81
2	1	B	248.44
3	1	A	222.84
4	1	D	291.97
5	1	E	323.89
6	2/3	G	197.04
7	1	E	164.27
8	1	F	175.05
9	5/3	Ab	205.66
10	7/3	Ab	206.18
11	5/3	E	329.21
12	2/3	C	130.51
13	1/3	A	219.78
14	1	E	166.67
15	1/3	Eb	306.51
16	2/3	Eb	311.28
17	1/3	A	220.39
18	1/3	Db	134.91
19	1/3	E	321.29
20	1/3	Eb	311.28
21	1/3	E	334.73
22	2/3	D	29630
23	1	C	263.15
24	1	A	216.80
25	1	Bb	229.89
26	1	Gb	184.76
27	1	F	175.43
28	1	A	222.22

	节拍	音调	基波频率(Hz)
29	1	Ab	211.64
30	4/3	A	222.22

至此，第二部分完结。

(5)心得体会：

对音频信号进行时域处理、频域分析使我更深入地了解到傅里叶变换的奇妙之处。可以更加深入地理解音乐背后的原理。在一些信号处理的方法上还是略显简略，例如寻找每个乐音开始点上，只是简单地采用寻找冲激点的方法。在截取乐音周期信号上也显得相对的随机。但总体效果还是不错的，在调试中也了解了许多MATLAB的知识，可谓收获颇多。

3.基于傅里叶级数的合成音乐

（1）

为了简单起见，在演奏乐曲上，只考虑（174.61Hz，329.63Hz）范围内的乐音，即将简谱的曲调映射到该频率范围内的对应乐音。随后将这些音对应的各次谐波信息储存到'guitar_freq_use.mat'文件中。随后自定义了fourier_song_creation函数来实现带吉他谐波的歌曲信号制作。

随后程序基本同问题（4），可以听出的确声音更加像吉他演奏。

（2）

将东方红的简谱写成乐音序列与时间序列后输入，便可以通过之前的程序与fourier_song_creation结合产生歌曲信号，由于加入谐波的计算，产生歌曲需要的时间可能较长，需耐心等待。

fourier_song_creation中的关键代码：

```
range_song = range .* xiebo(f,t,begin_time,guitar_freq_weight); %乘带谐波的分量
%xiebo函数关键代码如下

n = round(log2(basic_f/174.61)*12+1);          %求对应的位置
n = mod(n,12);
[r,c] = size(weight);

for i = 1:c
    if(weight(n,i)~=0)
        y = y + weight(n,i)*sin(2*pi*i*basic_f*(t-begin_time));
    end
end
```

该程序读取之前写好的'dfh_song_tone.mat'以及'dfh_last_time.mat'中的数据，从而进行谱曲创作。总体来说声音更加偏向于钢琴声，且有些音的杂音偏大。在修改guitar_freq_use.mat后，将其大于11次谐波分量全部无视，杂音基本消去。此时乐声类似钢琴声，又带着吉他的音感。

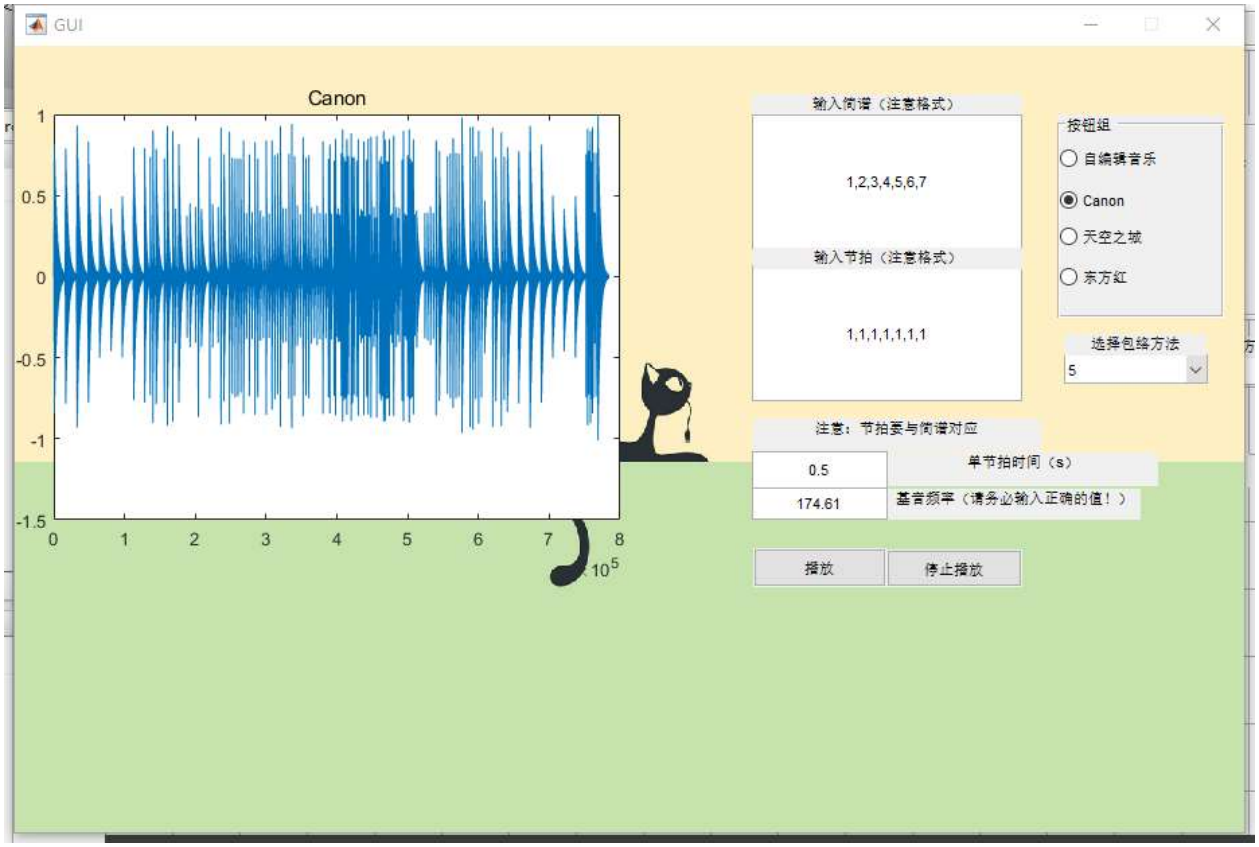
在fourier_song_creation中定义的包络方法method现增加至5种，音色听感大致如下：

1. 线性包络，听起来类似管弦乐器；
2. 线性冲激加指数衰减，听起来像钢琴声；
3. 正弦上升段加上sigmoid衰减段，听起来像吹小号；
4. 两端线性函数组合，类似于钢琴；
5. 线性冲激加指数衰减（衰减幅度不同），类似于钢琴与吉他（较推荐采用此方法）；

（3）GUI

设计了一个GUI程序进行展示，该程序只有合成音乐的功能，若要查看傅里叶分析音调功能，请允许task9.m文件以及参阅上文。

效果图如下：



4.文件整理

1. 简单的音乐合成
    1. task1.m——对应第一题
    2. task2.m——对应第二题
    3. task3.m——对应第三题
    4. task4.m——对应第四题
    5. Canon.m——对应第五题（演奏Canon乐曲，创作方法改为利用傅里叶级数谐波）
    6. sky_town——对应第五题（天空之城乐曲）
    7. 用到的函数文件包括：
        1. calc_f.m（计算频率）

    2. A_adjust.m（用于调整幅度）
    3. song_creation.m（未加入谐波的合成音乐）
    4. song_creation2.m（加入谐波的合成音乐）
2. 用傅里叶级数分析音乐
    1. task6.m——对应第一小题
    2. task7.m——对应第二小题
    3. task8.m——对应第三小题
    4. task9.m——对应第四小题
    5. 用到的函数文件：
        1. fft_analyse.m（利用fft进行分析）
        2. fourier_analyse.m（基于傅里叶变换矩阵）
        3. similar.m（判断两个数是否大致相等）
        4. recognize.m（识别乐音的基频以及各次谐波）
        5. translate.m（将频率翻译为对应基准为220Hz的音调）
        6. print_music_msg.m（将recognize得到的矩阵转为字符串信息）
        7. write_xiebo.m（将各次谐波信息写入mat文件中）
3. 基于傅里叶级数的合成音乐
    1. task10.m——对应第一小题
    2. dfh.m——对应第二小题
    3. 函数文件：
        1. fourier_song_creation.m（加上分析得到的各次谐波的音乐合成）
        2. xiebo.m（为音频加上谐波）
        3. guitar_freq_use.mat（记录谐波信息）
    4. GUI.fig,GUI.m——对应GUI程序
        1. sound_song.m：用于对文本框输入的乐谱进行谱曲，并播放。

5.参考资料

> 1、《信号与系统——MATLAB综合实验》——谷源涛等著；
>
> 2、《信号与系统》——郑君里
>
> 3、《几种时域滤波方法的比较》——CSDN博客，网址：https://blog.csdn.net/tianzhaixing2013/article/details/8877171
>
> 4、《快速傅里叶变换——MATLAB官方文档》，https://ww2.mathworks.cn/help/matlab/ref/fft.html