matlab 音乐合成 实验报告

无07 陈宇阳 2020010816

1.2.1-1 正弦波《东方红》

自然大调生成

首先需要辅助函数，生成每一个调所对应的音。后续就可以通过调用函数得到想要的大调的低中高声部的21个音的频率，同时将第22个音频率设置为0，作为休止符。

传入大调所对应的英文字符，返回一个`[1,22]`的矩阵。

在理解了十二平均律和自然大调的音程以后，基于国际标准音中央la 440Hz可以简单计算出每个音的具体频率，借助八度之间频率的倍数关系可以简化代码。

核心代码：

```
function tunes = my_get_tunes(major)
    C_ori = [261.6,293.6,329.6,349.2,392,440,493.8];
    mid_do = C_ori(1);
    switch (major)
        case 'A'
            mid_do = C_ori(6);
        case 'B'
            mid_do = C_ori(7);
        case 'C'
            mid_do = C_ori(1);
        case 'D'
            mid_do = C_ori(2);
        case 'E'
            mid_do = C_ori(3);
        case 'F'
            mid_do = C_ori(4);
        case 'G'
            mid_do = C_ori(5);
        otherwise
            error('Error: The parameter major is invalid!');
    end
    major_dis = [2,2,1,2,2,2];
    mid_tunes = [mid_do,zeros(1,6)];
    for i = 2:7
        mid_tunes(i) = mid_tunes(i-1)*2^(major_dis(i-1)/12);
    end

    tunes = [mid_tunes./2;mid_tunes;mid_tunes.*2];
    tunes = reshape(tunes',[1,21]);
    tunes = [tunes,0];
end
```

播放《东方红》

使用特定时长的特定频率的正弦波即可播放出想要的音。通过设置beat_len即每一拍的速度可以整体控制歌曲的播放速度。

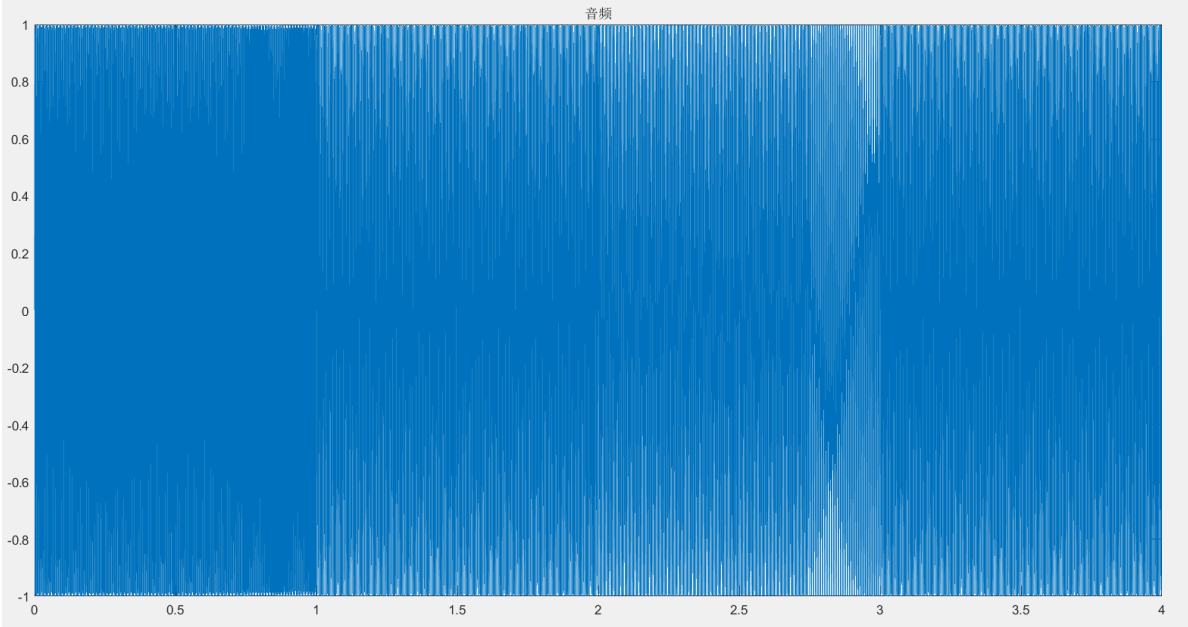
通过函数的方式可以更加直观地写出歌曲的谱面。

核心代码：

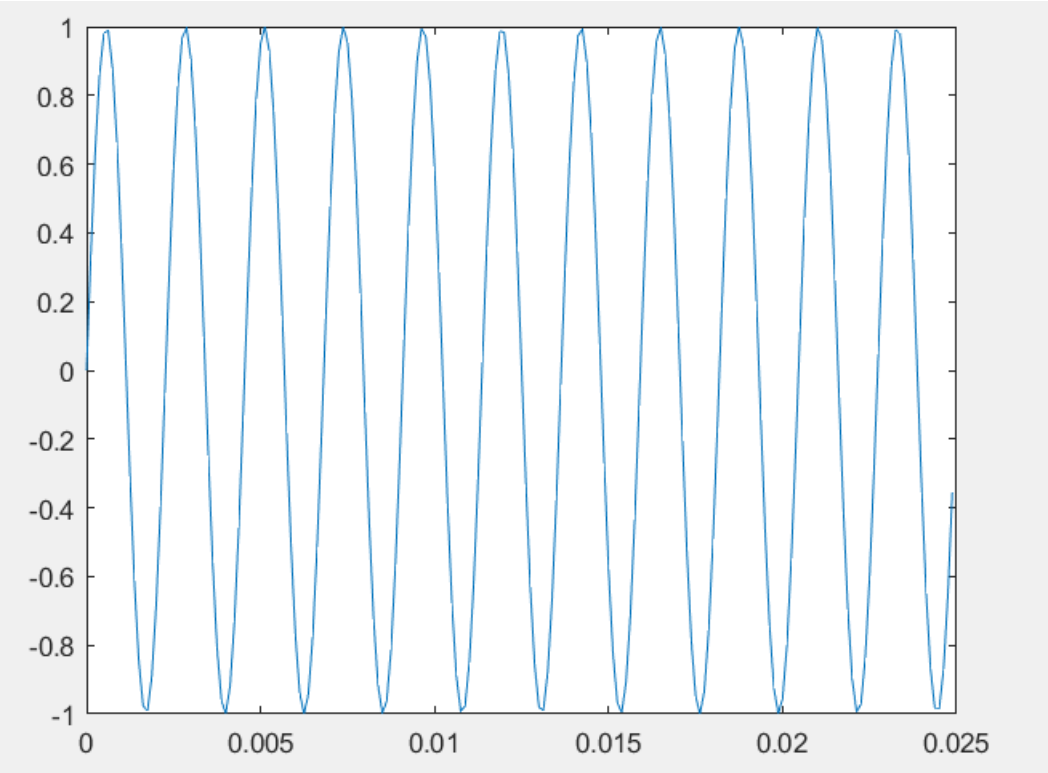
```
low = @(x) x;
mid = @(x) x + 7;
high = @(x) x + 14;
pause = @(x) 22;

song = [...
    mid(5), 1; mid(5), 0.5; mid(6), 0.5; ...
    mid(2), 2; ...
    mid(1), 1; mid(1), 0.5; low(6), 0.5; ...
    mid(2), 2];

len = size(song);
len = len(1);
res = [];
for i = 1 : 1 : len
    f = tunes(song(i, 1)); %对应唱名的频率
    time_len = song(i, 2) * beat_len;
    t = linspace(0, time_len - 1 / Fs, Fs * time_len)'; % 采样
    tmp_res = sin(2 * pi * f * t);
    res = [res; tmp_res];
end
```

> 借鉴参考：beat_len的设计以及使用函数让乐谱更加直观的方法参考了无92 刘雪枫学长的版本。

将生成的音频矩阵plot出来，显示如下：



放大：



可见确实是幅度为1的正弦波。

- 注意matlab当中，或者说计算机当中，一切都是离散的，没有连续的函数或者图像，音频也是由一个个采样的点生成。此处的采样频率(8000)一定要足够高，要远高于正弦波本身的频率，才能体现出正弦波本身频率的特性。

1.2.1-2 包络《东方红》

上一题所合成的东方红听上去很"机械"，因为只有对的音高，而没有合适的音强弱变化。在乐曲的每个音之间还存在"啪"的杂音，推测是因为相邻音之间的频率变化导致了相位的突变，最终产生了高频的冲激。

我们尝试使用包络函数去修饰每一个音，让每一个音在末尾时衰减到接近于0 。这样可以让音乐听起来更加自然，也避免了相位的突变。

包络函数设计

本实验我一共设计了两个包络函数。第一个模仿教材上乐音音量变化的图进行的分段函数拟合，第二个使用了教材作业提示的人耳听觉指数衰减进行的拟合。

其中系数的计算就是通过设定关键点（如二次函数最高点、零点，指数函数x=0的点）处的函数值解方程得到的。

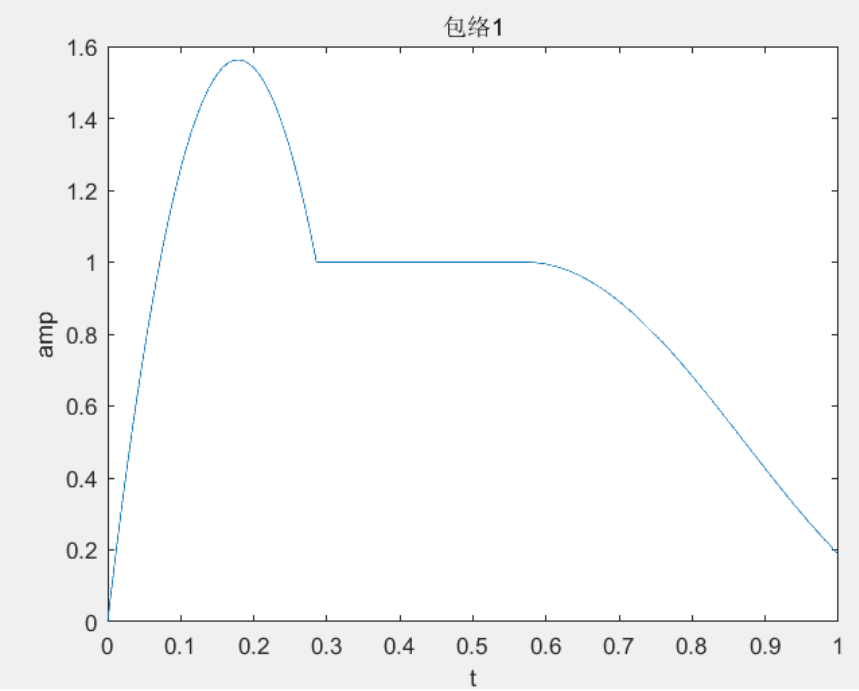
传入的参数是乐音持续的时间所对应的采样矩阵。

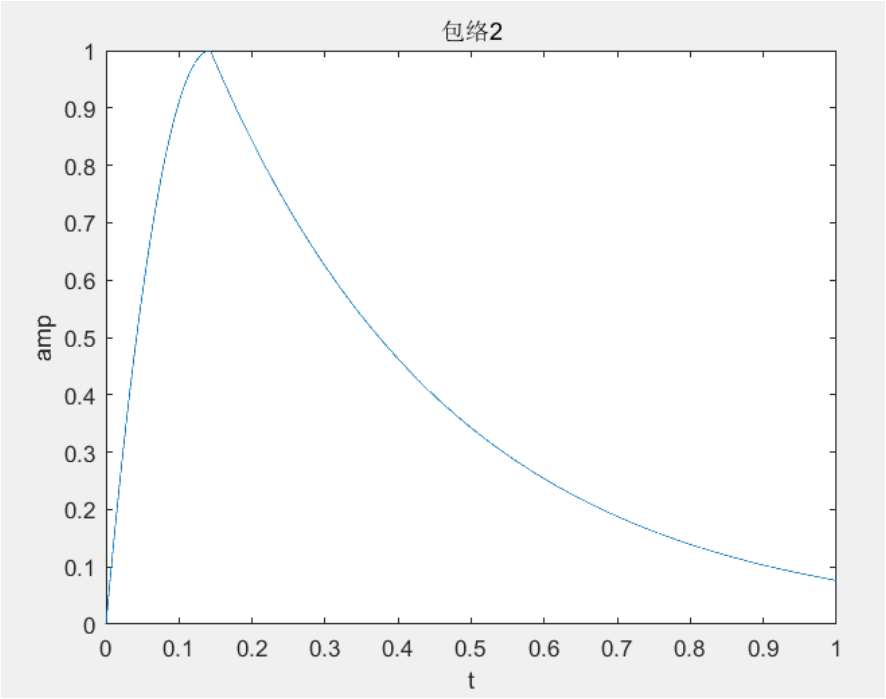
核心代码：

```
%  --------------包络1----------------
function y = envelop(t)
   len_t = length(t);
   T = t(end);
   y = linspace(0,T,len_t);
   % 二次函数的系数
   a = -49/T^2;
   b = 35/(2*T);
```

```
    % 三角函数的系数
    c = (7*pi)/(3*1.4*T);
    % 分段函数
    y = ...
        (t>=0 & t<2*T/7) .* (a.*t.^2+b.*t)+...
        (t>=2*T/7 & t<4*T/7).*(1)+...
        (t>=4*T/7) .* (0.5.*cos(c.*(t-(4*T/7)))+0.5);
end
% ----------------包络2----------------
function y = envelop_piano(t)
    len_t = length(t);
    T = t(end);
    y = linspace(0,T,len_t);
    % 二次函数的系数
    a = -49/(T^2);
    % 指数函数的系数
    b = 14/(T);
    % 分段函数
    y = ...
        (t>=0 & t<T/7) .* (a.*t.^2+b.*t)+...
        (t>=T/7) .* (exp(-3*(t-T/7)));
end
```

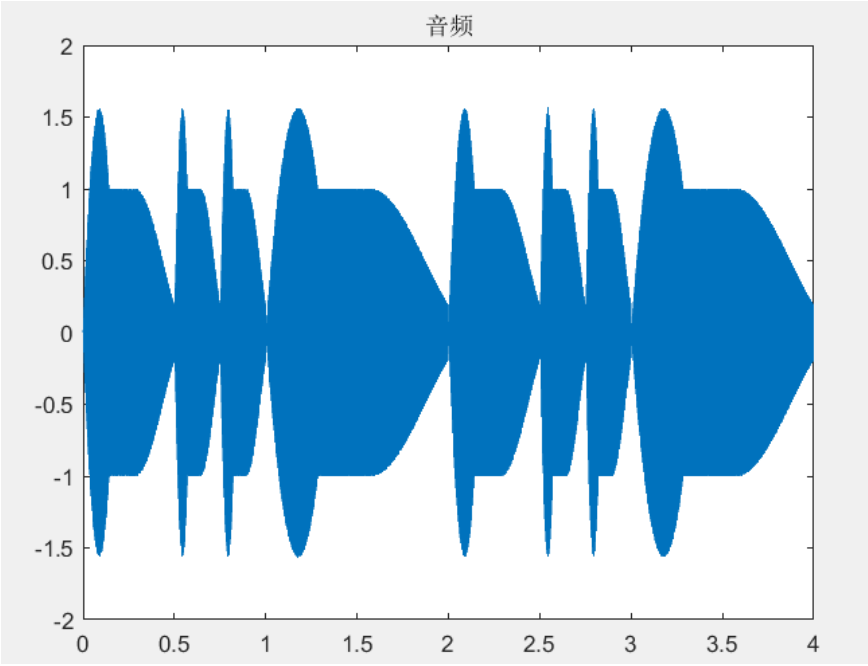
两个包络函数图像显示如下：

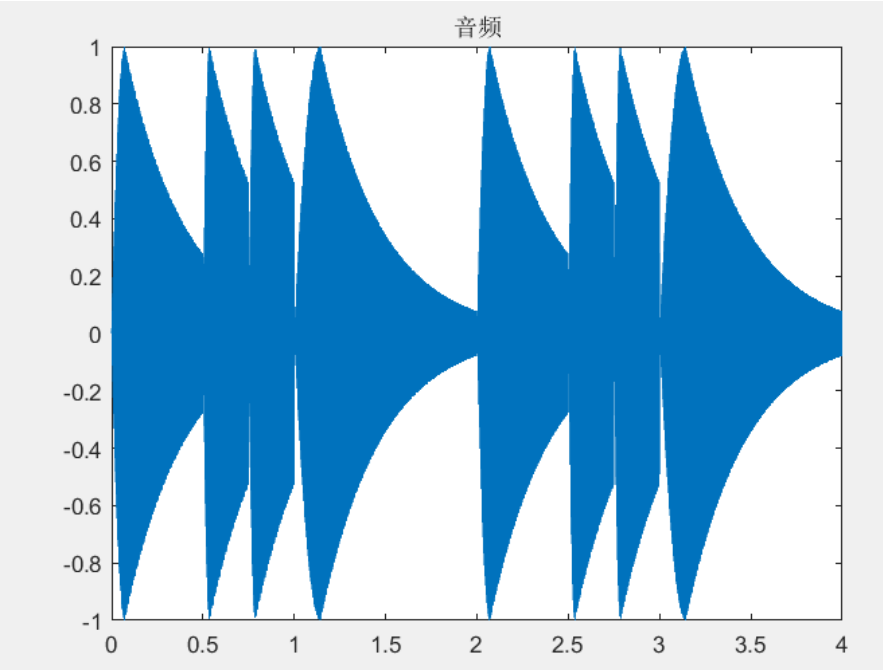




再播《东方红》

包络函数设计好后，由于上一实验乐音幅度均为1，故只需要给乐曲中每一个音乘以此包络，就可以得到有强弱变化的乐曲了。让我们听听看！





第一首乐曲没有了"啪"的杂音，因为每一个音结束后都几乎衰减到了0，但整体听感并没有第二首乐曲自然，但第二首乐曲"啪"音仍然存在，原因大家也可以从图中看见，乐音中间仍有明显缝隙。

> 思考：在较短时间内，乐音迅速衰减至0其实有一种类似"迷幻乐"的失真感。而真实乐器播放时应该存在延音，即后一个音开始时前一个音仍未结束。这时包络1和包络2结合的又自然又无杂音的最好的情况。但我没有太琢磨出这种情况应该怎么写，于是，一个饼就诞生了。

1.2.1-3 改变音高

升高、降低一个八度通过修改音频频率，修改采样频率都可以实现。直接修改for循环中每一个单音的频率从`2*pi`变为`4*pi`就可以实现**变调不变速**的音频升八度。而把sound采样频率提高为两倍则音调升高一倍，而时间缩短为1/2 。

`resample`函数本质上也是修改采样频率，提升一个半音相当于把采样频率提高到原来的$(2^{\frac{1}{12}})$倍，此方法同样会导致速度的改变。

其实如果手头上只有音频的最终矩阵，而不知道每个单音的时长，不能简单地实现变调不变速。第一种提高音调而不变速的方法仅仅是因为我们的音乐是简单的单音，可以轻易实现分割与延拓。

核心代码：

```
for i = 1 : 1 : len
    f = tunes(song(i, 1)); %对应唱名的频率
    time_len = song(i, 2) * beat_len;
    t = linspace(0, time_len - 1 / Fs, Fs * time_len)'; % 采样
    tmp_res = sin(2 * pi * f * t).*envelop(t);
    % 频率翻倍即可提高八度
    tmp_res_higher = sin(4 * pi * f * t).*envelop(t);
    res = [res; tmp_res];
    res2 = [res2;tmp_res_higher];
end
res3 = resample(res,2,1);
res4 = resample(res,round(Fs*(2)^(1/12)),Fs);

% res 为原音频，res2为高八度不变速音频，res3为降低八度速度降为1/2音频，res4为重采样提高半音音频。

```

```
sound(res,2*Fs);
% 此方式同样会高八度且速度变为2倍。
```

1.2.1-4 增加谐波分量

谐波分量是决定音色的关键。

通过矩阵相乘的形式简化每个音的谐波分量合成形式使代码更简洁。

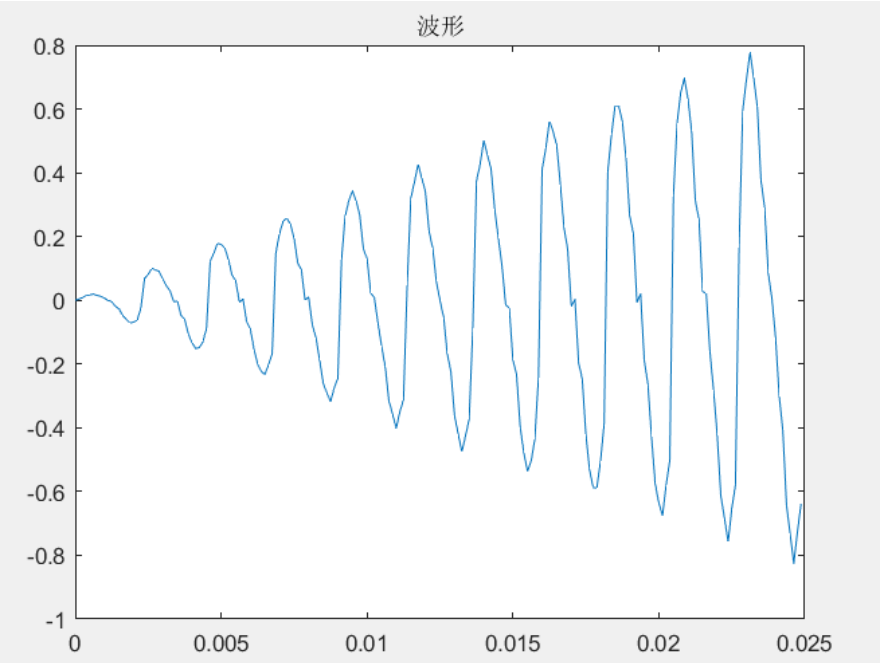
> 此处上网查找了别人拟合出来的钢琴的1-15次谐波分量。
>
> 根据乐谱合成钢琴音乐_KevinLeeee的博客-CSDN博客

核心代码：

```
amps = [1,0.340,0.102,0.085,0.070,0.065,0.028,0.085,0.011,0.030,0.010,0.014,0.012,0.013,0.004]'; % 据说是钢琴谐波分量
harmonics = 1:15;


for i = 1 : 1 : len
    f = tunes(song(i, 1)); %对应唱名的频率
    time_len = song(i, 2) * beat_len;
    t = linspace(0, time_len - 1 / Fs, Fs * time_len)'; % 采样
    waves = sin(2*pi*f.*(t*harmonics));
    tmp_res = waves*amps.*envelop(t);
    res = [res; tmp_res];
end
```

让我们听听看。。。没有想象中好听……属于是意料之内情理之中。

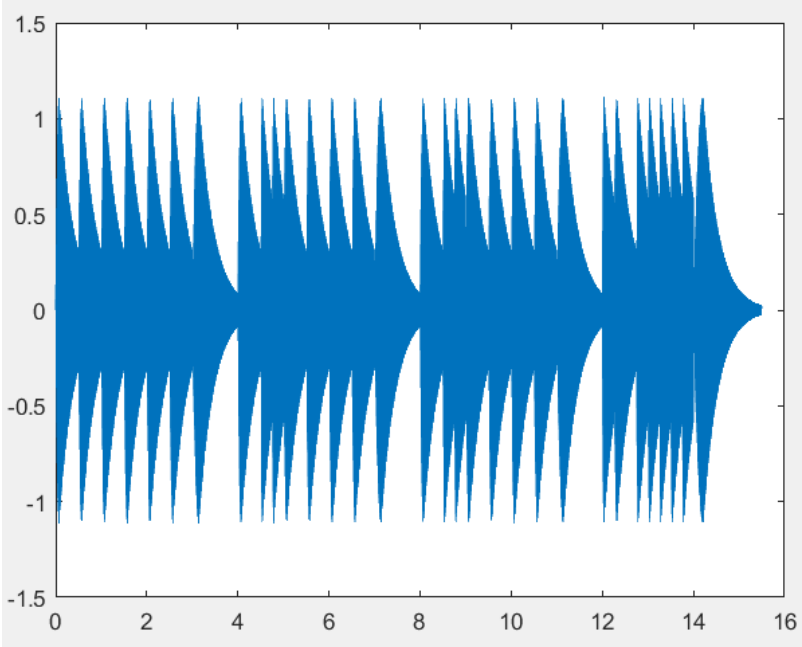
观察局部放大的波形图，可见确实不是圆滑的正弦波了。



1.2.1-5 自选音乐合成

使用钢琴的音色和指数包络为大家弹奏一曲《两只老虎爱跳舞》。

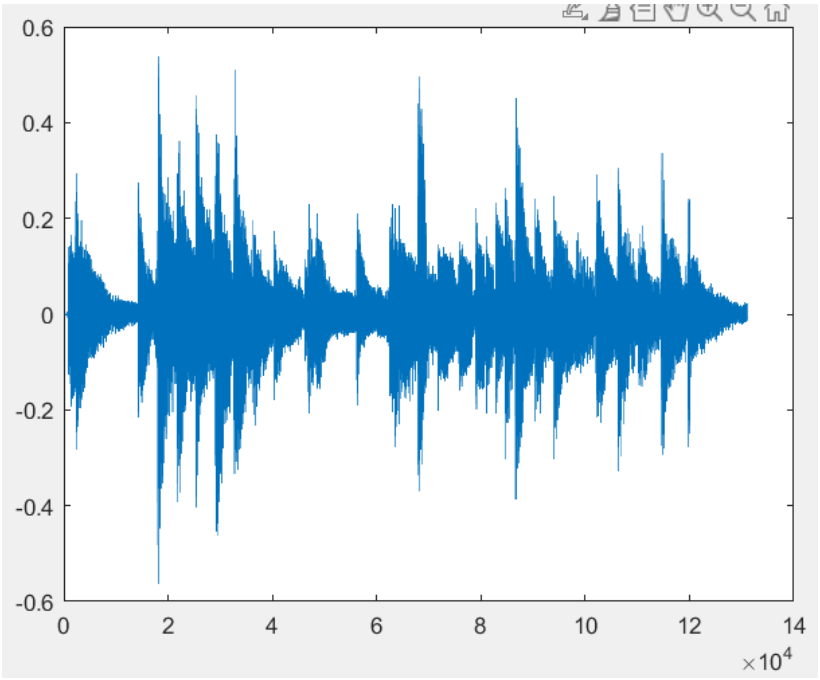
> 这也是我唯一会弹的"钢琴曲"。



1.2.2-6 载入wav文件

核心代码:

```
load("matlab\音乐合成大作业\assets\Guitar.MAT");
[wave,Fs] = audioread('./matlab/音乐合成大作业/assets/fmt.wav');
figure(1);
plot(wave);
sound(wave,Fs);
figure(2);
plot(realwave);
figure(3);
plot(wave2proc);
```

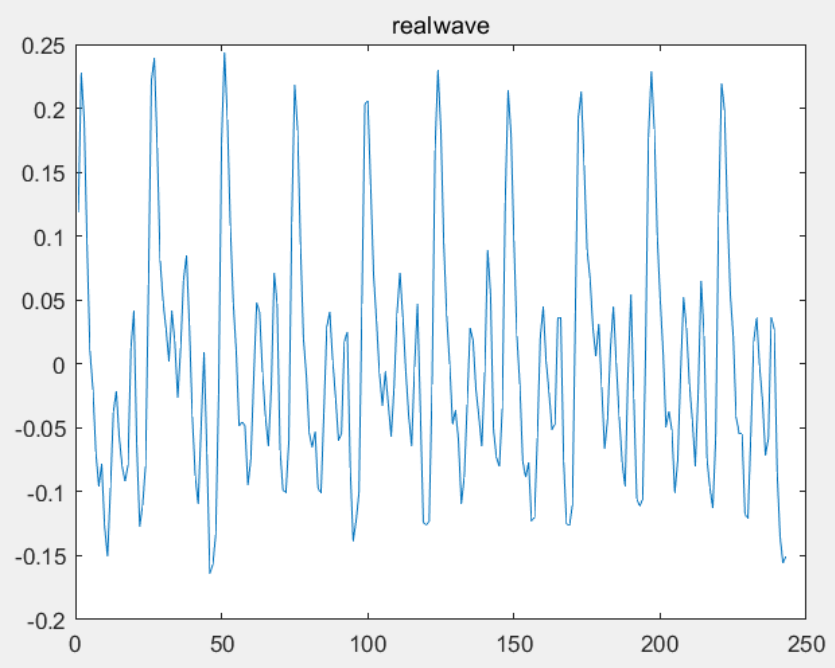


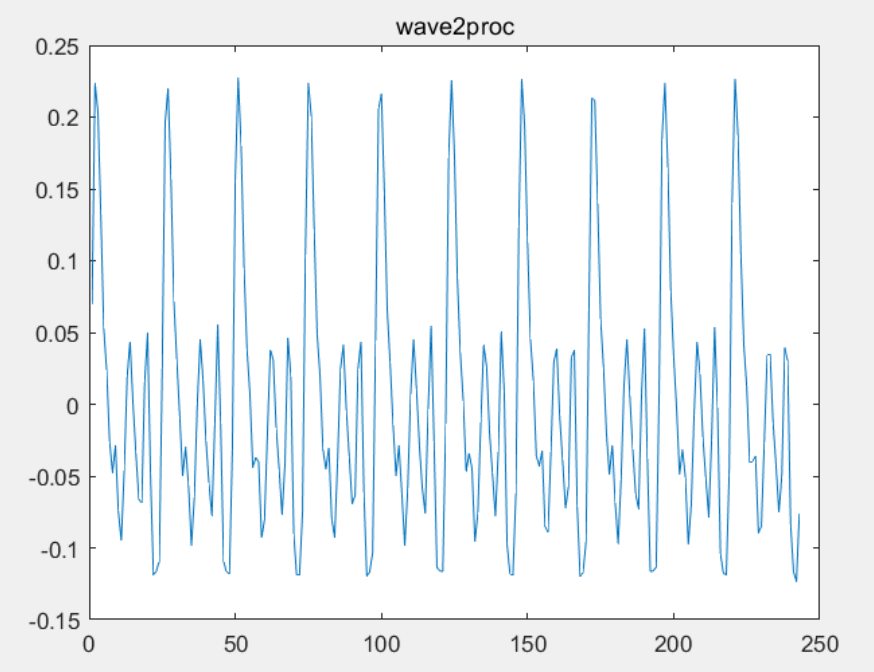
听听音乐，看看波形。

真实乐器弹奏出的音乐确实更加柔和且具备美感。

1.2.2-7 音频预处理

仔细观察两个波形的异同，注意到两者均是同一类型的波型循环了10个周期，但realwave中每个波有高有低有大有小，而wave2proc中的每一个周期中波型的"尺寸"更加相似。猜测在realwave中存在着加性白噪声，导致同一个波在不同周期中出现了偏差。要消除波形中的非线性谐波和噪声，可以通过把周期全部加起来然后取平均的方法。



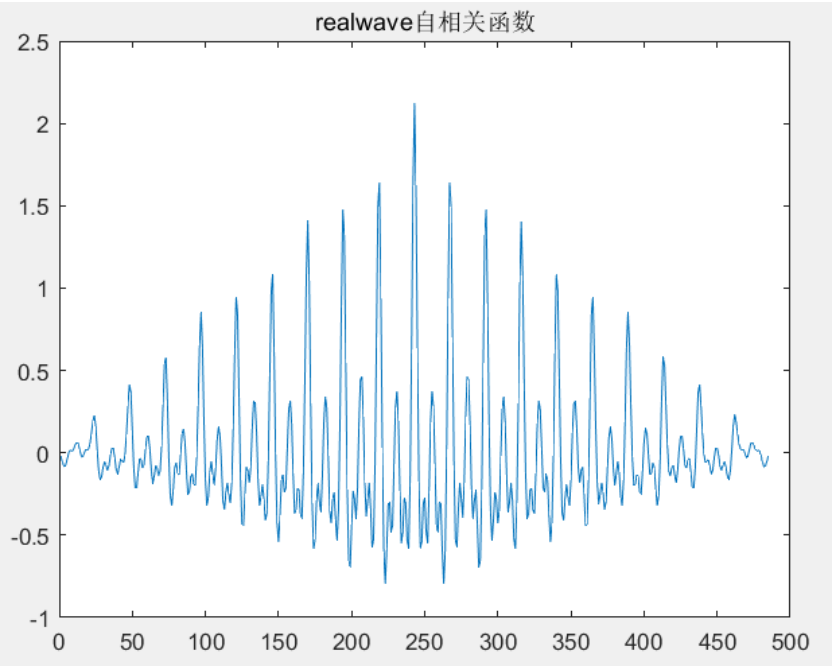


思路：

- 由于需要把不同周期的波相加，**此处需要用到matlab中的`circshift`循环移位函数**，进行9次的循环移位并相加，让每一个波形都能充分取到平均。
- 如何确定要循环移位多少个样本点呢？联系到信号与系统中学过的模式波匹配的算法，我们可以利用**自相关函数**：当两相似的波形卷积到一起的时候，其自相关函数会产生波峰。自相关函数可以使用`xccor`实现。
- 使用`findpeaks`找出自相关函数中的所有波峰位置，不出意外的话，前9个位置应该对应9次循环移位所需要移动的样本点，当然我们也可以把所有波峰对应的移动点都移动了，并除以总波峰数量

得到平均数。

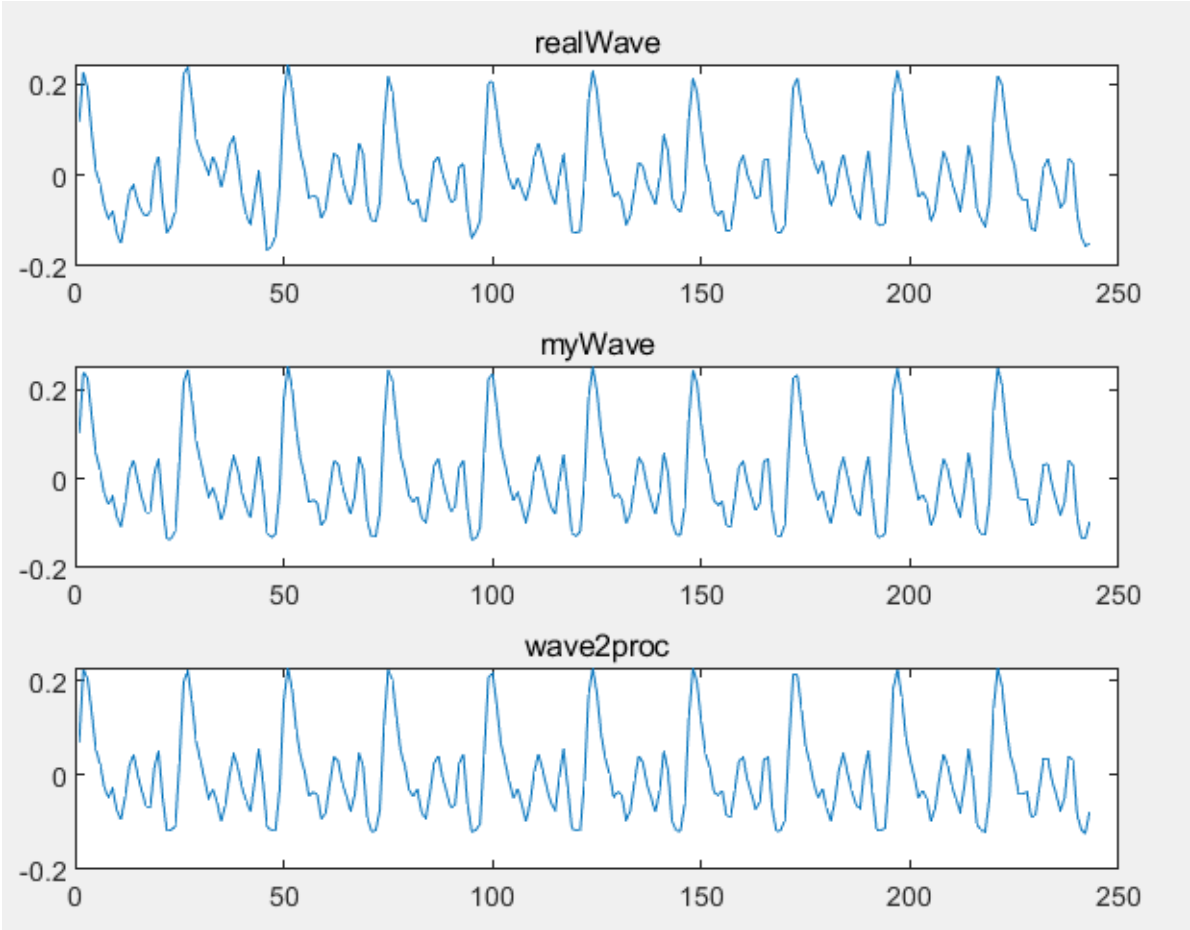
自相关函数如下：



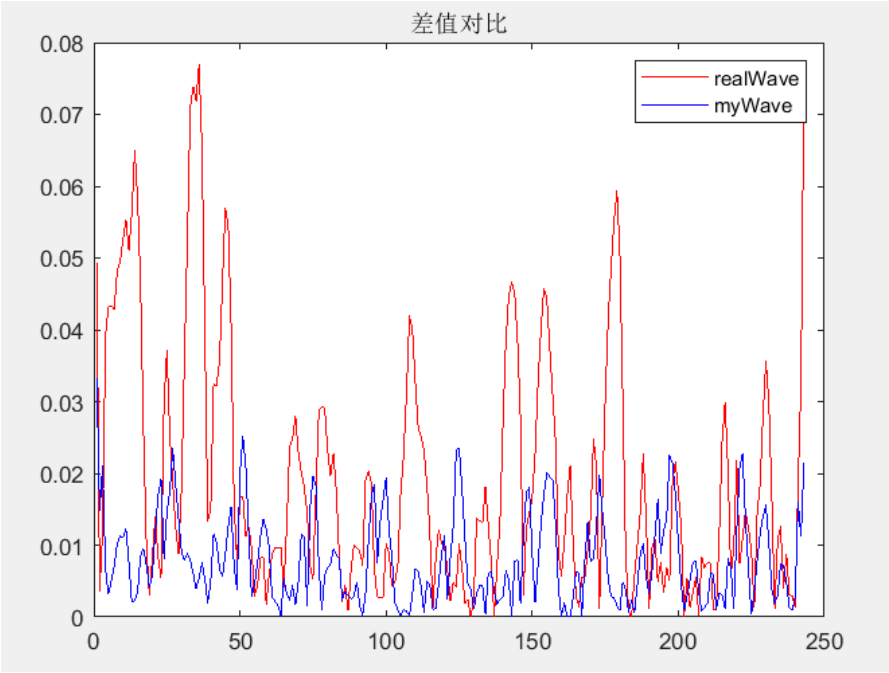
核心代码：

```
load("音乐合成大作业\assets\Guitar.MAT");
cor = xcorr(realwave,realwave);
[pks,locs] = findpeaks(cor,'MinPeakHeight',1);
plot(cor);
title('realwave自相关函数');
my_wave = realwave;
for i = 1:length(locs)
    my_wave = my_wave + circshift(realwave,locs(i),1);
end
my_wave = my_wave ./ length(locs);
```

使用此方法得到的预处理波形图和realwave以及wave2proc对比如下：



貌似较难观察不同，我们可以以wave2proc作为标准，通过比较每个点取差的绝对值判断myWave是否比realWave要更接近于wave2proc。



可见myWave确实要更加接近wave2proc，所以说这个预处理是成功的。

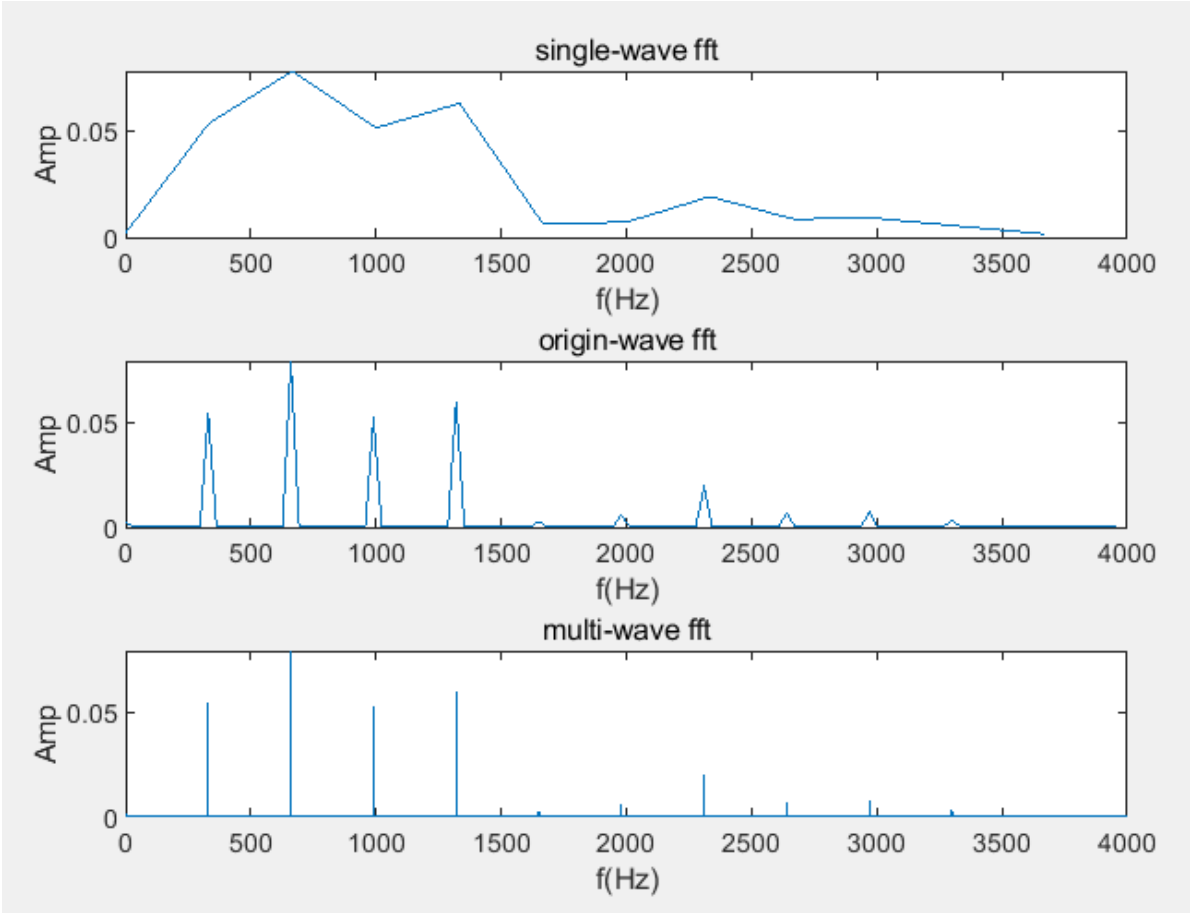
1.2.2-8 对音频作FT

使用matlab 的快速傅里叶变换 `fft` 函数对这段音频进行傅里叶变换。需要对变换后的矩阵的横坐标的跨度进行修改，使之变为[0,4000Hz]的频域范围。最后再plot出图像。

核心代码：

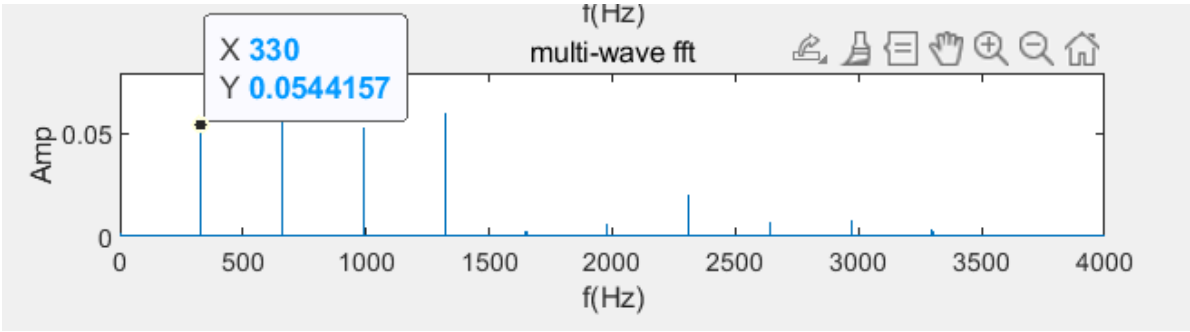
```
single_wave = wave2proc(1:round(L/10));
single_L = length(single_wave);

multi_wave = wave2proc;
for i = 1:1:32
    multi_wave = [multi_wave;wave2proc];
end
multi_L = length(multi_wave);

single_F = fft(single_wave);
F1 = abs(single_F/single_L);
F = fft(wave2proc);
F2 = abs(F/L);
multi_F = fft(multi_wave);
F3 = abs(multi_F/multi_L);

F_1 = F1(1:single_L/2+1);
F_1(2:end-1) = 2*F_1(2:end-1);
subplot(3,1,1);
plot(0:(Fs/single_L):(Fs/2-Fs/single_L),F_1(1:single_L/2));
title('single-wave fft')
xlabel('f(Hz)');
ylabel('Amp');

F_2 = F2(1:round(L/2)+1);
F_2(2:end-1) = 2*F_2(2:end-1);
subplot(3,1,2);
plot(0:round(Fs/L):(Fs/2-round(Fs/L)),F_2(1:round(L/2)-1));
title('origin-wave fft')
xlabel('f(Hz)');
ylabel('Amp');

F_3 = F3(1:round(multi_L/2+1));
F_3(2:end-1) = 2*F_3(2:end-1);
subplot(3,1,3);
plot(0:round(Fs/multi_L):(Fs/2-round(Fs/multi_L)),F_3(1:4000));
title('multi-wave fft')
xlabel('f(Hz)');
ylabel('Amp');
```

取一个周期，使用原信号和进行了32次的周期延拓后的三个信号进行傅里叶变换后波形如下：



虽然三个波形在特定频点上的值是一样的，但显然进行了信号重复的傅里叶变换得到的值要更加精确。

究其原因，我认为是信息量不同导致的。matlab中所做的离散傅里叶变换虽然可以是局部的，但可以认为局部也是全局的，只是时域其他位置的强度为0而已。时域波形非零区域越短，意味着这段音乐的高频率分量越多（参考冲激函数的FT）。而对音乐波形进行周期延拓，其实是在时域提供了更多的信息量，减少各种频率存在的可能性。因此随着周期数目的增加，信号的频域图谱也愈发收敛于特定的频率值。



这段音乐的基频是330Hz，对应的是小字一组的E，谐波分量对应2、3、4、7次分量，相对于基波的强度分别为：1.46，0.96，1.09，0.35 。

1.2.2-9 分析乐曲音调和节拍

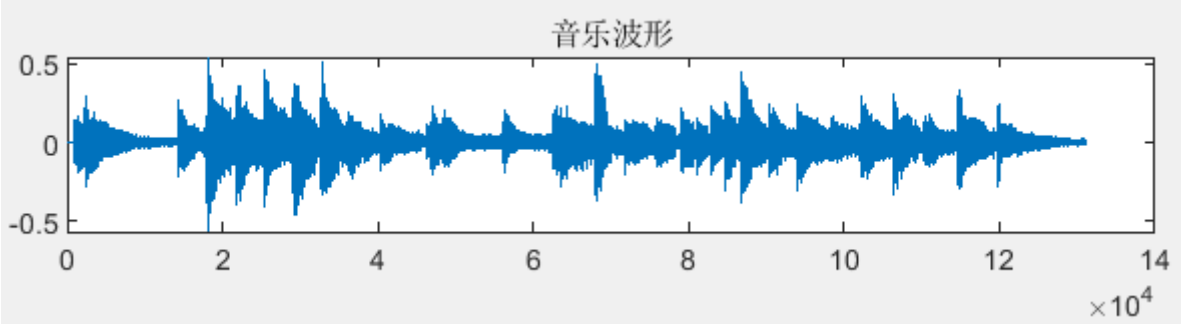
这就是本次大作业最难的一部分了。乐曲中每个乐音时长不定，强弱不定，更是存在和声和延音！

- 要分析乐曲的节拍，我们需要合理地切割乐曲，尽可能分为一个个单音。
- 得到单音以后，我们再通过傅里叶变换得到单音的频谱，从里面自动找出基音和谐波分量。
- 如果存在和声，还需要分辨出哪个是主音，哪个是和声。

分割乐曲

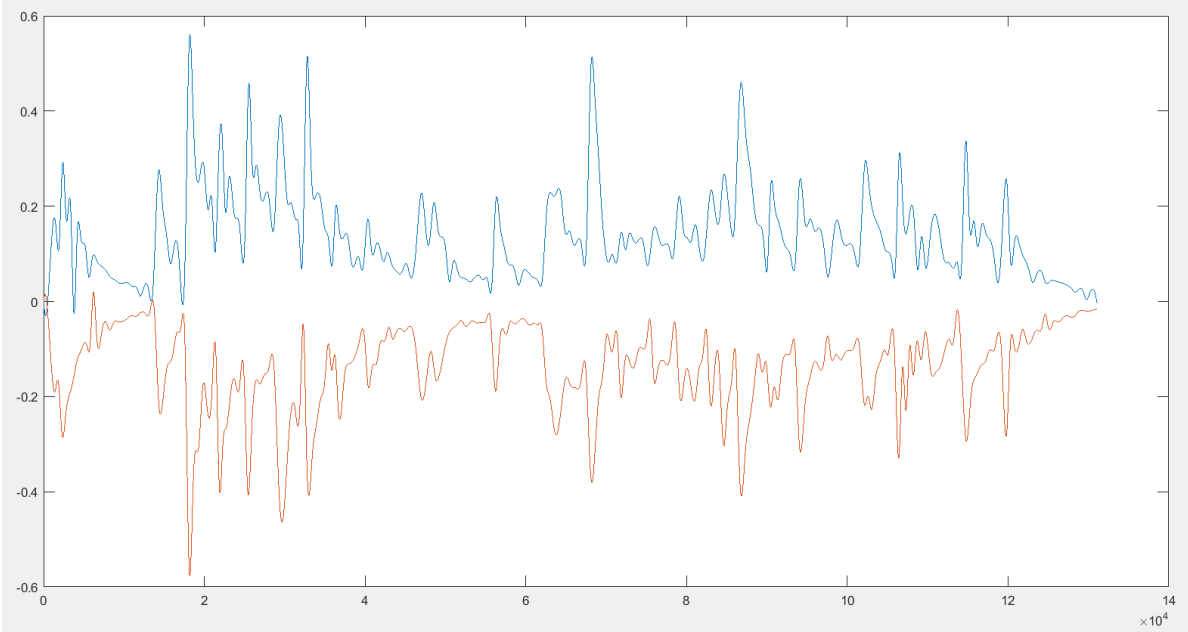
关于分割乐曲，我的想法是得到音乐波形的一个比较粗略的包络，然后基于包络的波峰和波谷来进行分割。

观察音乐的波形，其实可以看见吉他发出的每一个音其包络形状是大致相同的。我们可以基于包络去分割单音。



首先通过`envelop`函数得到包络，由于我们特别关注波峰波谷，因此将包络模式设置为`peak`，为减少噪音产生的额外波峰，精度不需要太高。

上包络和下包络生成如下：



为了进一步减少噪声影响，将上下包络翻转并取平均。

然后使用`findpeaks`函数找到波峰和波谷。

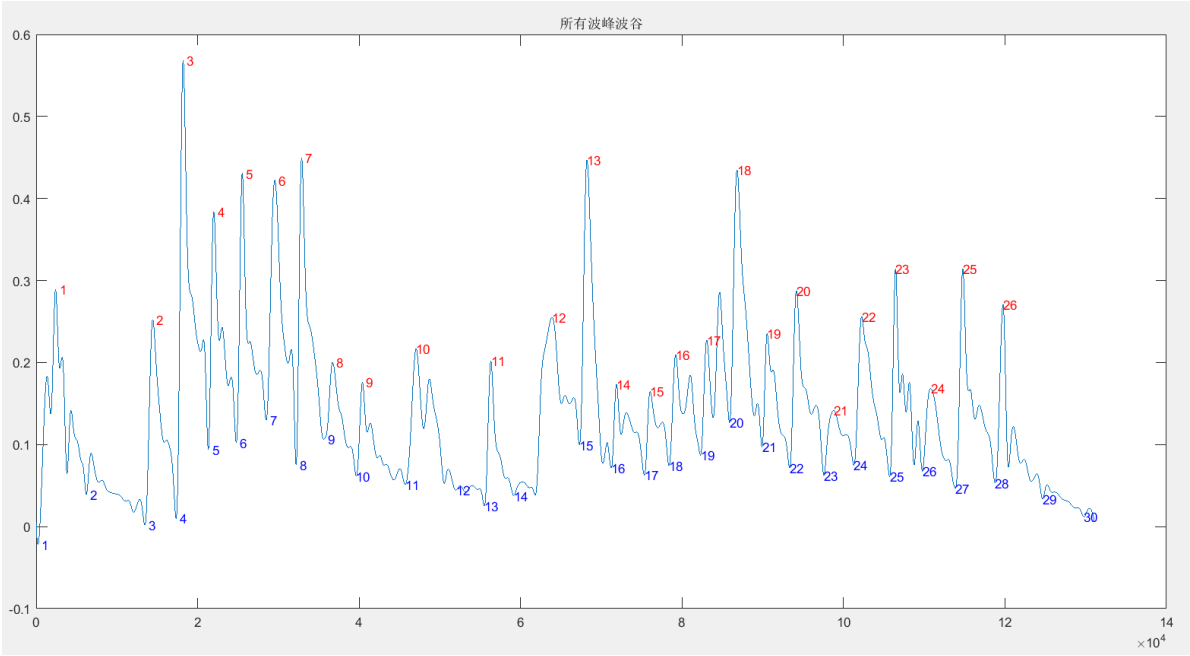
寻找波峰：`[pks,locs] = findpeaks(en,'MinPeakHeight',0.13,'MinPeakDistance',3000);`

限制条件：作为可以被人耳感知到的音，最小高度应大于0.13，且相邻两个音中间至少应该相隔3000个采样点。

波谷同样可以通过`findpeaks`找到，只需要使用波形最大值减去波形本身，就可以实现波谷到波峰的翻转，然后用同样的方法找到所有波谷。

核心代码：

```
% 获取包络
[up_en,low_en] = envelope(music,400,'peak');
en = (up_en-low_en)/2;
dis_en = abs(0.5685-en);

% 找到波峰波谷
[pks,locs] = findpeaks(en,'MinPeakHeight',0.13,'MinPeakDistance',3000);
[lows,low_locs] =
findpeaks(dis_en,'MinPeakHeight',0.43,'MinPeakDistance',3000);
lows = 0.5685-lows;

subplot(3,1,1);
plot(music);
title("音乐波形");
```



由上图可以看见，波峰和波谷并不是一一对应的，我们需要去删减一些多余的波峰和波谷，让它们一一对应，以实现波形的分割。

判断基准：我们以前一个波谷为单音开始，以为后一个波谷作为单音结尾，两个波谷中间有且仅有一个波峰。

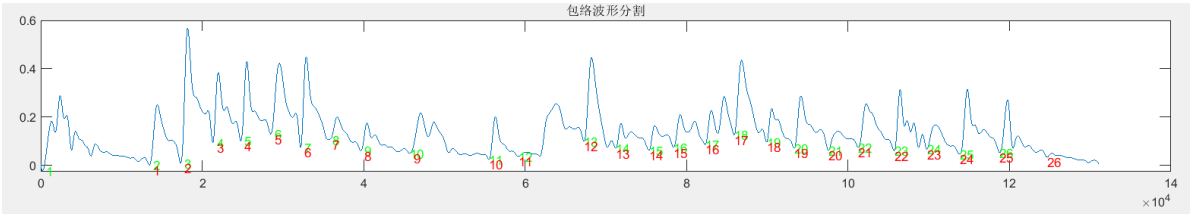
对所有样本点进行遍历，符合判断基准的加入矩阵，不符合的进行筛选，并对波峰波谷序号的对应进行调整。

核心代码：

```
% 两个极小值点作为一个音的起始和结束
beat_start = [];
beat_end = [];
shamt = 0; % 偏移量
for i = 1: length(low_locs)-1
    if (i+shamt <= 26)
    if (low_locs(i)<locs(i+shamt))&&(low_locs(i+1)>locs(i+shamt))
         beat_start = [beat_start;low_locs(i)];
         beat_end = [beat_end;low_locs(i+1)];

    elseif  (low_locs(i)>=locs(i+shamt))
```

```
        shamt = shamt+1;
        i = i-1;
    elseif (low_locs(i+1)<=locs(i+shamt))
        beat_end(end) = low_locs(i+1);
        shamt = shamt -1;
    end
    end
end
```

最终波峰波谷筛选结果如下：



可见对于单音的包络分割是比较成功的。

波形分割完成后，先把每个乐音的时长和波峰的振幅储存起来，以备后用。

频率分析

得到单音以后，为了避免前后音之间的互相影响，每个单音都只取中间的90%，再进行1024次的周期延拓提高变换的精度。

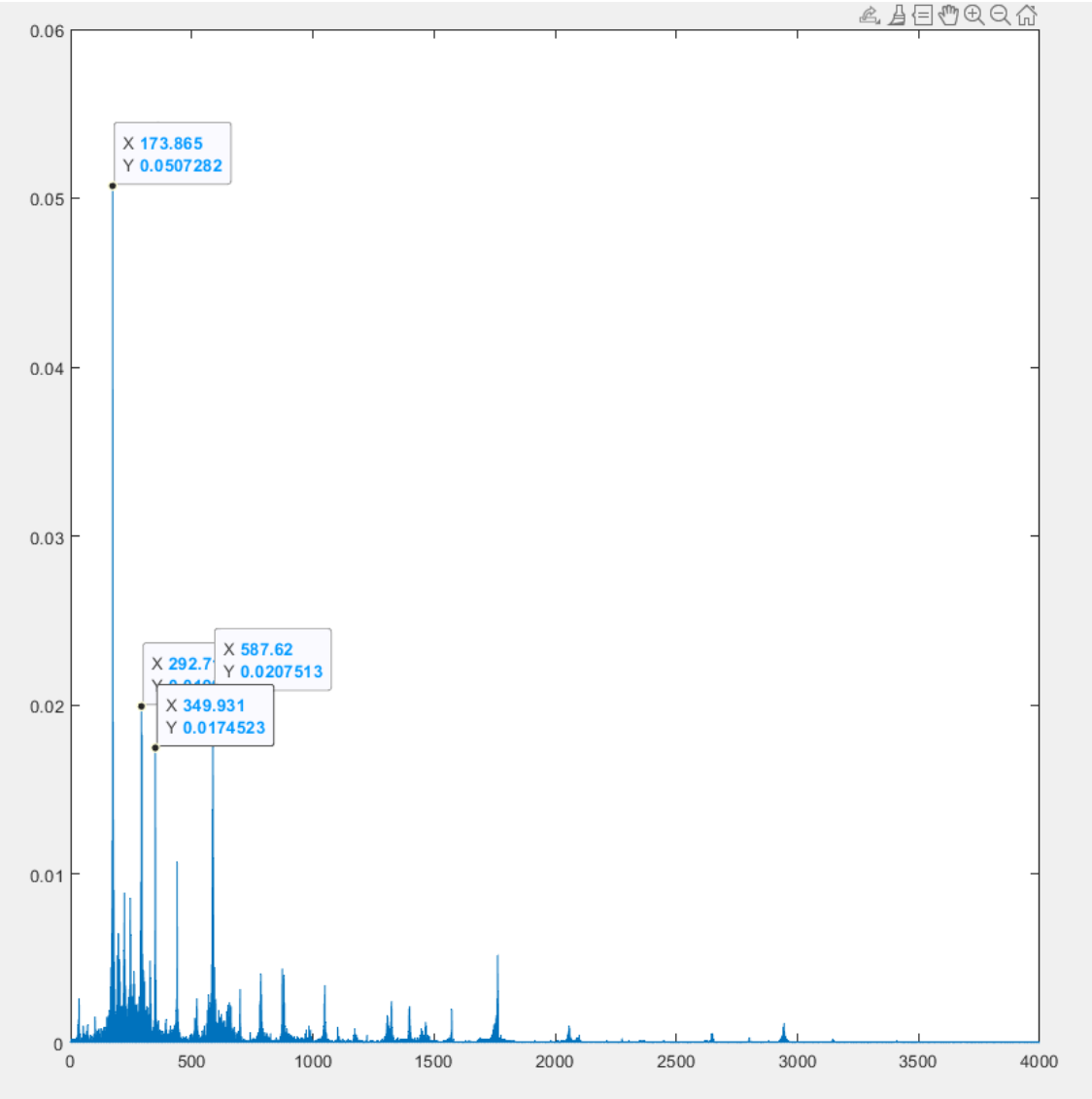
此处写了一个`my_fft`函数以简化代码结构，从"样本点域"变换到时域的傅里叶变换。

```
function [f , Amp] = my_fft(X)
L = length(X);
Fs = 8000; % 采样率设置为8000
Y = fft(X);
Amp = abs(Y/L);
Amp = Amp(1:round(L/2)+1);
Amp(2:end-1) = 2*Amp(2:end-1);
f = 0:Fs/L:Fs/2-Fs/L;
Amp = Amp(1:round(L/2));
end
```

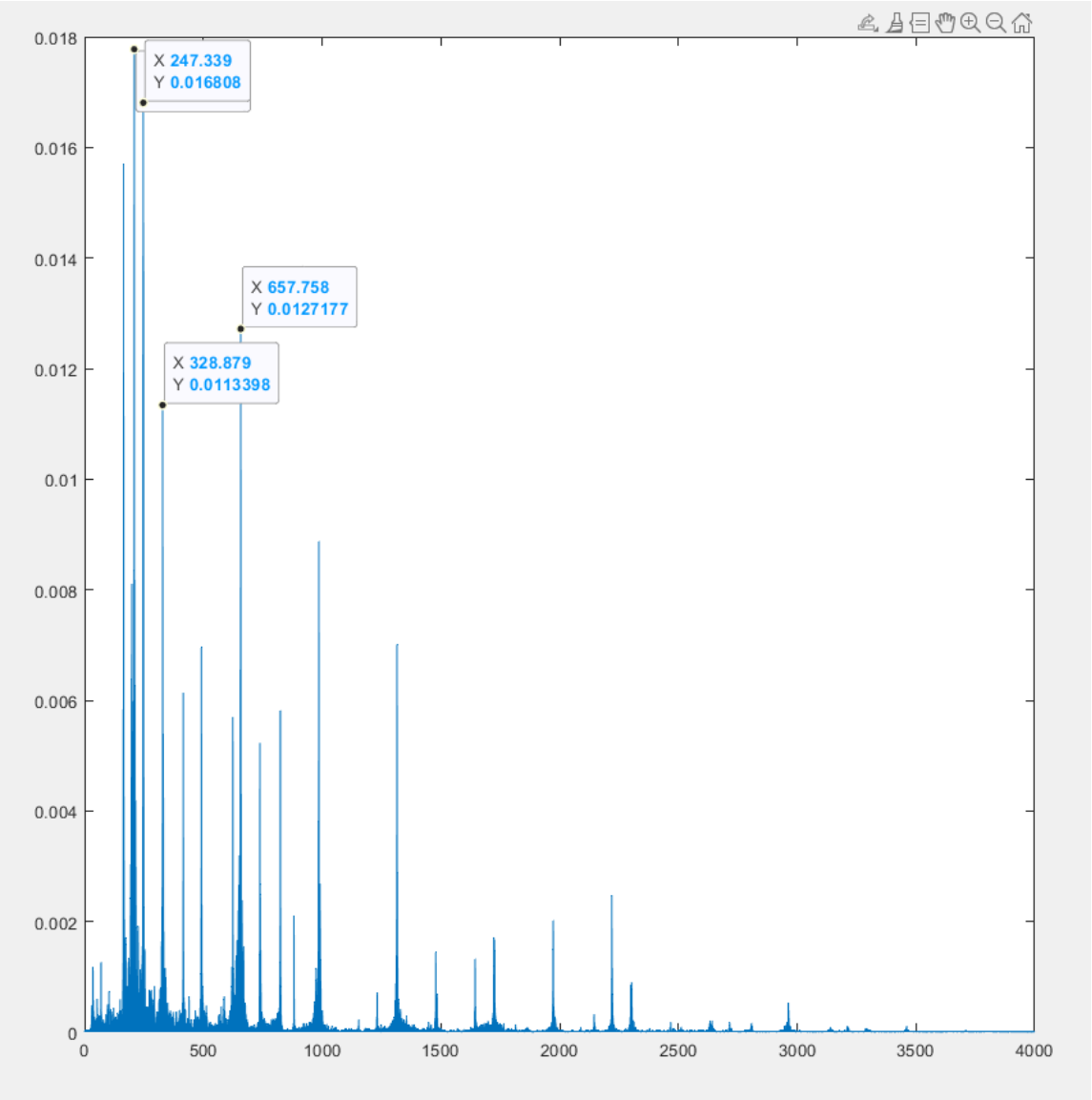
进行`fft`后对频域信号进行`findpeaks`，确定基频和谐波分量。问题的关键在于确定基频，只要确定基频，谐波分量只需要取乘以相应倍数处的值即可得到。

但基频较难确定，在古典吉他的音色当中，幅度最大的频率峰不一定是基频，频率最小的频率峰也不一定是基频。在存在和声的片段当中，更是会存在多个基频和谐波分量。

下图为存在两个基频的单音频谱情况：



下图为存在很多和声的单音频谱：



像上面这种地狱难度的频谱，让人观察频谱确认哪个是最”入耳“的基音都很难做到。这当然也是因为前面一步音频分割做得不够好导致的，前后乐声的杂音混进来也会产生不同频率分量的干扰。

经过多次试错以后，我总结出以下规律和方法：

1. 基音对应的频率大概率是多个频率峰对应频率的最小公倍数。
2. 基音的频率幅度不一定是最高，但一定不低，不可能小于最高频率峰的1/2 。
3. 幅度值达到最高频率峰的1/3的所有频率点纳入分析范围，频率峰之间相距3000个样本点为可分辨的频点。
4. 基音频点取倍数后所对应的单个频点可能幅度很低（因为样本点很密集），应该取该频点附近的幅度最大值作为谐波幅度值。
5. 存在和声的情况，很难说哪个才是”主频率“。

   我分别计算几个基音和相应2-5次谐波分量的幅度值之和，再进行比较，取基音加所有谐波分量幅度最大的频率作为”主基音“。
6. 经过第5步处理筛选得到的的基音，效果并不是很好——`sound`听见的音确实是音乐片段中存在的音，但往往是低沉的和声，而非想要的主旋律音。经过对比后发现，人耳判断一个音是否”抓耳“，是否是”主旋律“当中的音，不仅是通过音的幅度，也通过音的频率区间判断。于是我加入了权值进行加和，位于钢琴小字一组的频率更有机会称为主旋律，频率过低的音一定不是主音。选择了[260,500]Hz范围内的音的幅度进行了翻倍再进行第五步的加和。

7. 乐曲中还存在着低八度和声。以及有时和声由于声音低，弹拨力度其实与主旋律相当，幅度很大但并不会干扰主旋律。此时就是无论如何都解决不了的情况，计算机一定会把又低又高的频点识别为基频。这时我只能人为修正，让重新合成的乐曲听上去更好一点。

核心代码：

```
% 单音切割与频域分析
base = [];
beat = []; %记录每一个单音的时长
harmonic = zeros(length(beat_end),10);

for i = 1:length(beat_start)
L = beat_end(i) - beat_start(i);
beat = [beat;L];
segment = music(beat_start(i):beat_end(i));
segment = segment(round(0.05*L):round(0.95*L));
rep_seg = repmat(segment,1024,1);
% plot(rep_seg)
[f,F] = my_fft(rep_seg);
[max_peak,max_i] = max(F);
[f_pks,f_locs] =
findpeaks(F(1:round(length(F)/5)),"MinPeakHeight",max_peak/3,'MinPeakDistanc
e',3000);
% 根据总和声幅度最高的频率寻找基频
cal_base = zeros(length(f_locs),5);
for j = 1:length(f_locs)
    for k = 1:5
        cal_base(j,k) = max(F(f_locs(j)*k-1500:f_locs(j)*k+1500));
        if (f(k*f_locs(j))>=260 && f(k*f_locs(j))<=500)
            % fprintf("No. %d f=%f cal_base(%d,%d)=%f,
double\n",i,f(k*f_locs(j)),j,k,cal_base(j,k));
            cal_base(j,k) = 2*cal_base(j,k);
        end
    end
end
sum_base = sum(cal_base,2);
[~,idx] = max(sum_base);
% 频点修正
if (sound_fix(i))
    idx = sound_fix(i);
end
base_i = f_locs(idx);
this_base = f(base_i);
harmonic(i,1) = F(base_i);

fprintf("No.%d base_i %d\n",i,this_base);
% 谐波分量计算
base = [base;this_base];
% 寻找2-10次谐波
for j = 2:10
    temp_i = j*base_i;
    if (temp_i+1500 > length(F))
        harmonic(i,j) = 0;
    else
        harmonic(i,j) = max(F(temp_i-1500:temp_i+1500));%取范围内的最大值
```

```
    end
end
```

乐曲定调

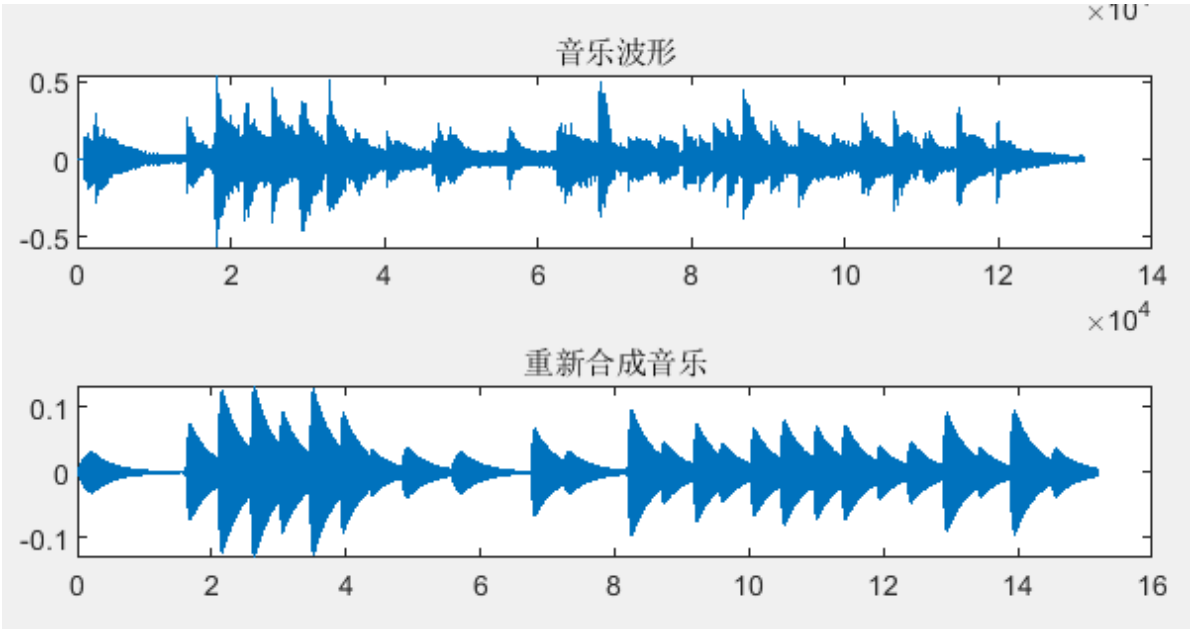
得到了每个乐段的基频以后，定调就是比较简单的事情了。与标准的频率进行比对选择匹配程度最高的调即可，注意要容易一定量的频率误差。本曲应该是C调的乐曲，把所有基音频率换为C调中的标准频率。

核心代码：

```
% 定调
tunes = ['C','D','E','F','G','A','B'];
all_tunes = [];
tunes_cnt = zeros(7,1); % 记录每个调的得分
for i = 1:7
    this_tune = my_get_tunes(tunes(i));
    all_tunes = [all_tunes;this_tune];
end
% 匹配基音
esp = 3; %误差在 esp Hz内可认为频率相等

for i = 1:length(base)
    temp = abs(base(i)-all_tunes);
    cnt = temp <= esp;
    tunes_cnt = tunes_cnt + sum(cnt,2);
end
[m,idx] = max(tunes_cnt);
major = tunes(idx);
tunes = my_get_tunes(major);
temp = tunes;
% 转换基音为标准音
for i = 1:length(base)
    for j=1:length(temp)
        if abs(base(i) -temp(j))<=3
            base(i) = temp(j);
        end
    end
end

har_amp = harmonic;
```

重新合成

增加谐波分量后，使用前面的指数包络对乐音进行修饰后播放，可见两段音乐波形比较相似，乐声听感也与吉他相似，乐曲听起来也比较像，只是没有了和声，以及中间有一些比较快的八分音符没有分割出来。有这个效果我已经非常感动了。



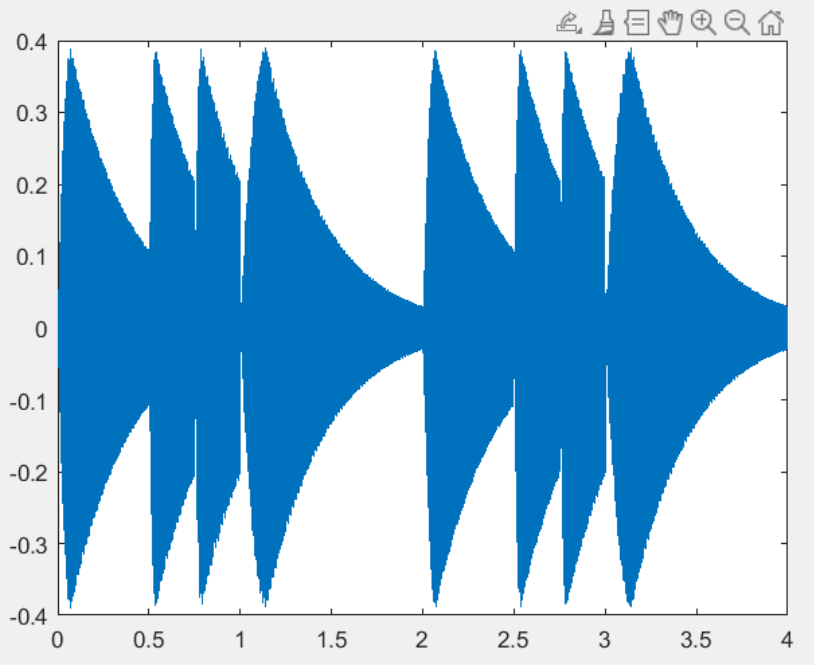
1.2.3-10 又播《东方红》

将1.2.2-8中计算出来的谐波分量以矩阵形式`save`起来，在本题中`load`即可。

核心代码（在exp1-8中）：

```
% 储存傅里叶级数
[f,F] = my_fft(multi_wave);
[pks,locs] = findpeaks(F,'MinPeakHeight',max(F)/5);
single_harmonics = zeros(1,10);
base = locs(1);
for i = 1:length(locs)
    times = round(locs(i)/base);
    single_harmonics(times) = pks(i);
end
save("single_har.mat",'single_harmonics');
```

只能说不太像，因为从上一题知道，不同基音是有不同的谐波分量的。

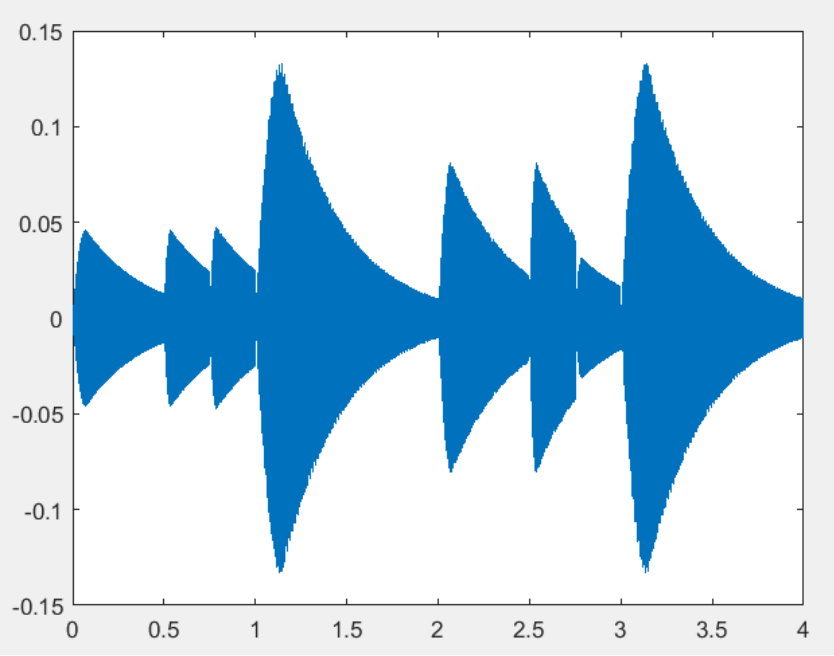


1.2.3-11 还播《东方红》

将1.2.2-9中计算出来的基音以及对应的谐波分量`save`起来，然后在本实验中`load`。东方红中每一乐音与吉他中基音对应，使用相应的谐波分量即可。注意同一基音有多个谐波分量组，我是直接选择了第一个匹配的谐波分量组，整体应该是大同小异的。

核心代码：

```
for i = 1 : 1 : len
    amps = [.03,zeros(1,9)]';
    f = tunes(song(i, 1)); %对应唱名的频率
    time_len = song(i, 2) * beat_len;
    guitar_idx = 0;
    for j = 1:length(base)
        if f == base(j)
            guitar_idx = j;
            amps = harmonic(j,:)'; % 对应的谐波分量组
            break;
        end
    end

    t = linspace(0, time_len - 1 / Fs, Fs * time_len)'; % 采样
    waves = sin(2*pi*f.*(t*harmonics));
    tmp_res = waves*amps.*envelop_piano(t);
    res = [res; tmp_res];
end
```

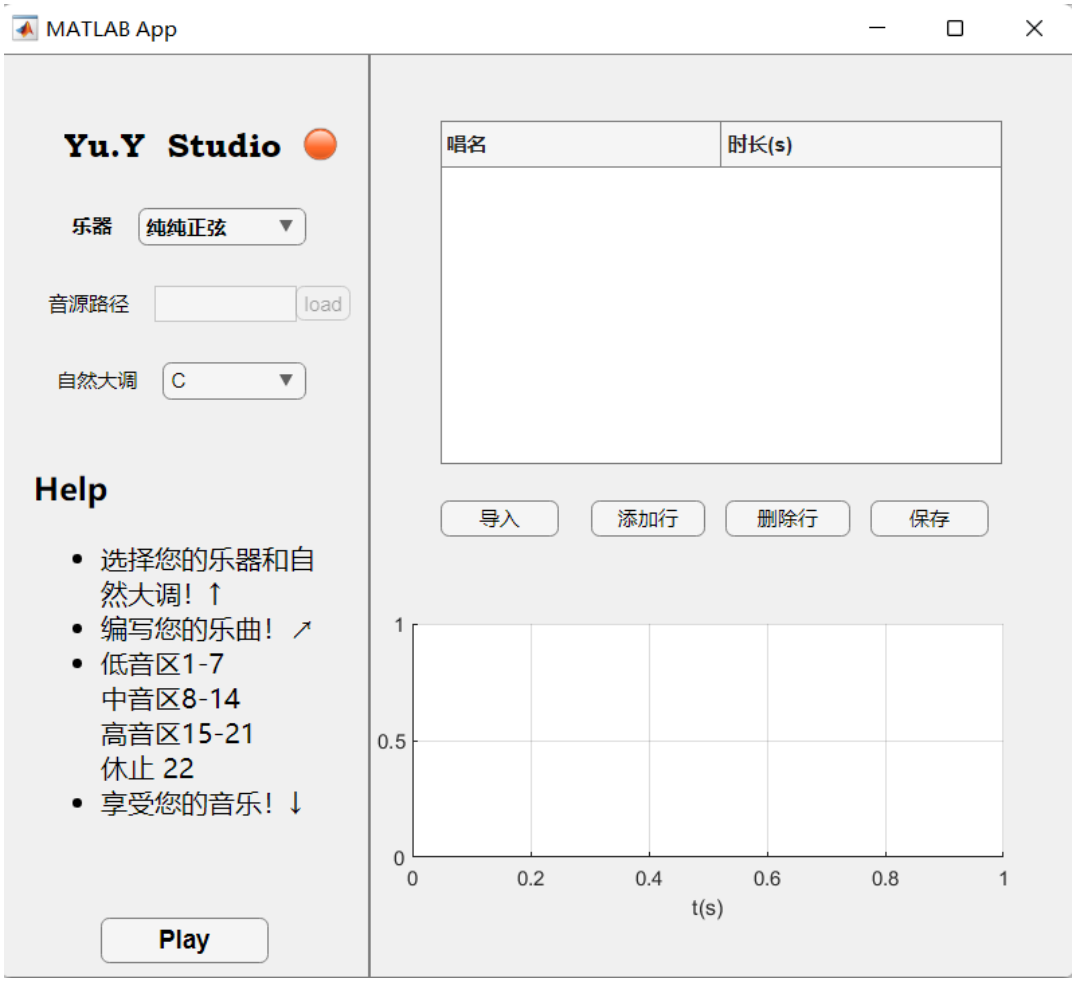


这一次听感与该吉他更加相似。此外乐音的幅度也出现了变化，这是因为谐波分量矩阵没有归一化，但听上去却更像吉他弹奏了，故没有作修改，绝对不是因为懒。

1.2.3-12 图形界面 Yu.Y Studio

> 图形界面的设计使用了`matlab App Designer`。其使用方法参考了大量CSDN博客，但核心内容都是自己设计与完成。

运行界面如下：

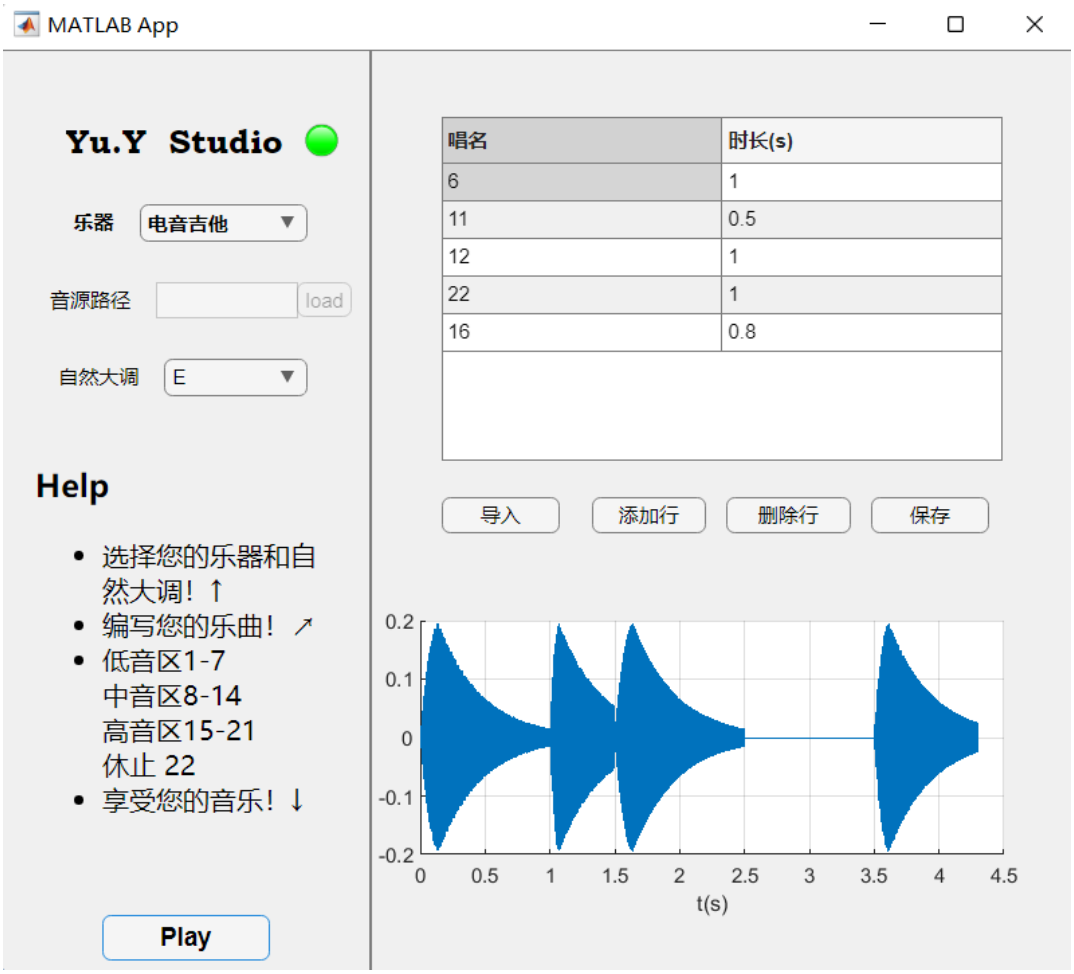


内置非常赛博朋克的钢琴和吉他音色！支持自选音源进行导入，选择储存了谐波分量矩阵的`.mat`文件即可导入。

乐谱支持在Excel中编辑然后导入，Excel文件位于`./sound/assets/sound.xlsx`中。编曲完成后可以保存数据到excel当中。

点击`Play`按钮即可播放音乐，右侧窗口显示音乐波形。

演示：



实验小结与未来展望

本次实验加深了我对matlab的理解，令我对它的强大的功能叹为用止，大部分我想要的信号处理的操作它都有封装完善的函数提供，设计GUI界面也非常方便，面向对象的思想也让编程变得简单。

此外，从时域和频域对一首音乐进行百般倒腾，我对于音乐作为一种信号的理解也加深了。什么是基频，和主音的对应关系是什么，什么是谐波，和和声的关系又是什么？频率是什么，节拍又是什么？音色从哪里来？为什么我始终爱听真实乐器的弹奏？电子乐的优势又在哪里？我都有了自己的答案。

此外，我对傅里叶变换的理解也加深了，傅里叶变换本质上只是一种运算，此运算可以把时域信号变换到频域，但绝不限于此功能，应该以开放的眼光看待傅里叶变换。

此外，我对于数字信号的理解也加深了。全程在离散的`linspace`的横坐标下进行信号处理，让我明白了采样频率的意义。

本次实验当然还有很多可以改进的地方。比如：

- 1.2.1-2中包络设计中指数衰减和相位截断之间的矛盾
- 乐音之间的过渡连接处理不好，没有延音，只有单音
- 1.2.2-9乐曲的分割难以做到细致，基音提取难以做到准确
- 图形界面其实也还非常粗糙。最不用户友善的地方应该是写乐谱唱名时需要使用1到22来写，用户需要自己在脑海中作低中高声部的运算。这是因为图形界面中table的储存方式是`cell`元胞，我在从元胞转换为矩阵时遇到了不小的困难。本来是想通过'#'和'b'作为前缀来表示高音和低音的，但后面发现字符和数字夹杂的且长度不等的字符串极其难以转换为mat，最终只好妥协，使用纯数字作为元胞中的元素，算是一大遗憾吧。

最后感谢谷哥哥和助教们的倾情指导，也要感谢姚铮老师在信统上传授的知识。没有你们，我难以完成本次实验。