MATLAB 综合实验之音乐合成

姓名: 张智帅

学号： 2016010483

日期： 2018 年 7 月 29 日

目　录

第1章 实验内容

1.1 简单的合成音乐

1.1.1

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？

【分析】

首先根据书上知识算出《东方红》片段中各个乐音的频率。

乐音（F 调）	5	6	2	1	6（低八）
频率 Hz	587.33	523.25	392	349.23	296.66

封装匿名函数 note 方便得到同时与频率和节拍相关的向量。使用 t 函数构造时间向量，用 note 函数嵌套 t 函数得到基本波形，再对应每种频率写一个函数，如 do, re, mi.

这样虽然看起来嵌套了多层，但是引用起来方便。任何音符都可以用类似 so(t(0.5))（即半拍 so）的形式表达。不需要多写一个节拍矩阵，而且方便修改波形。不论是增加谐波分量还是调整衰减，只需要修改 note 函数。

【代码】

```
%% tempo
fs = 44100;  %采样率
Meter = 250;     %拍数，控制节奏
tMeter = 60 / Meter;     %一拍的时间
%% 音符时值函数
t = @(beat)(0:1/fs:2*beat*tMeter);
%% Note
note = @(freq,t)(sin(freq*2*pi*t));
la_1 = @(t)note(296.66,t);%la_1,  f = 296.66Hz
do = @(t)note(349.23,t);%do,  f = 349.23Hz
re = @(t)note(392,t);%do,  f = 392Hz
so = @(t)note(523.25,t);%so,  f = 523.25Hz
la = @(t)note(587.33,t);%la,  f = 587.33Hz
%% 东方红
meter1 = [so(t(1)) so(t(0.5)) la(t(0.5)) re(t(2))];%东方红
meter2 = [do(t(1)) do(t(0.5)) la_1(t(0.5)) re(t(2))];%太阳升
line = [meter1 meter2]';
sound(line,fs);
```

【效果】 每个音符之间有明显的“啪”的杂声，每一个音符声音比较单薄，很容易听出是电子乐。

1.1.2

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。

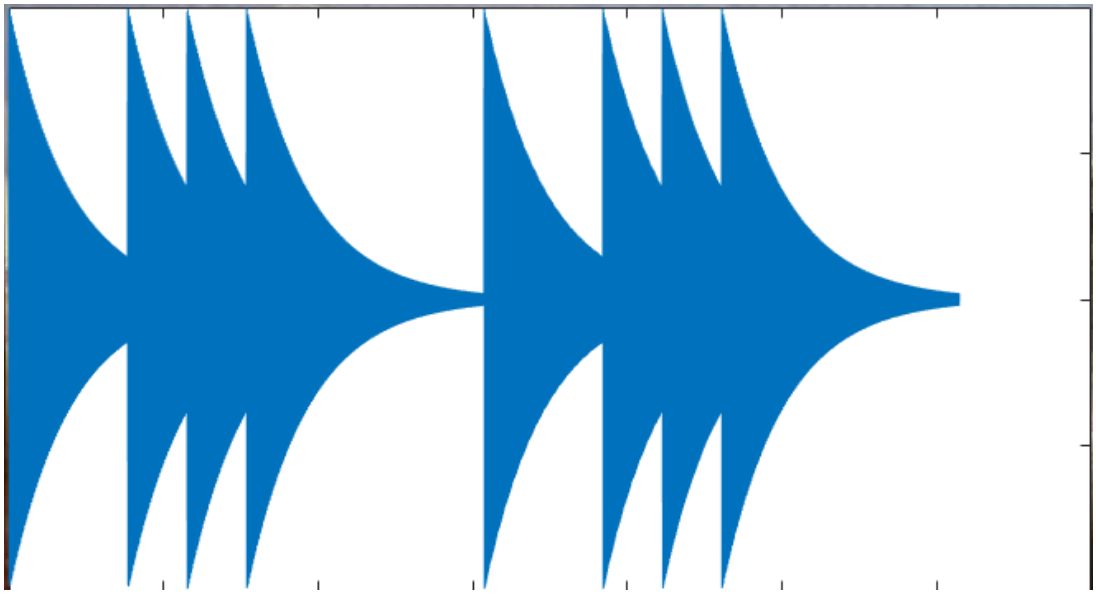
【分析】

给上一小问中的音符添加指数衰减包络即可。仅需改动 note 函数。

【代码】

```
note = @(freq,t)(sin(freq*2*pi*t)).*exp(4*-t);
```

【效果】 “啪”杂声基本消失，整段音乐听起来非常清脆，像敲击瓷器。音乐波形如下：



但是实际上包络不完美，冲击上升的波形听起来不太自然。

1.1.3

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。

【分析】

定义一对函数 lift/drop，可实现任意 n 阶调音。原理是根据十二分音律调整抽样数，resample 时间向量。

【代码】

定义：

```
%% 升高/降低n阶
lift = @(n,t)resample(t,fs,round(2^(n/12))*fs);
drop = @(n,t)resample(t,fs,fs/round(2^(n/12)));
```

调用：

```
tt = @(beat)lift(8,t(beat));
meter1 = [so(tt(2)) so(tt(1)) la(tt(1)) re(tt(4))];%东方红
meter2 = [do(tt(2)) do(tt(1)) la_1(tt(1)) re(tt(4))];%太阳升
```

【效果】

通过调节 n，可以完成任意阶的调音。注意到，由于调音的原理是 resample 了时间向量，所以会在变调的同时变速。升高音调则加速，降低音调则减速。所以如果想要只调音调，需要同时调整时间 t.

1.1.4

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？）

【分析】

仅需调整 note 函数，在其中增加分量即可。从此也能看出来设置 note 函数的可扩展性较强。

【代码】

```
note = @(freq,t)(sin(freq*2*pi*t))...
    +0.2*sin(2*freq*2*pi*t)...
    +0.3*sin(3*freq*2*pi*t)...
    .*exp(4*-t);
```

【效果】

通过给 note 函数增加谐波分量，可以轻松调节音色。基波幅度为 1 ，二次谐波幅度 0.2 ，三次谐波幅度 0.3 ，听起来有点像风琴。

经查资料，由弦振动模型可解得真实琴弦振动的谐波含量，基波幅度为 0.936 ，二次谐波幅度 0.011 ，三次谐波幅度 0.011，这样听起来相对厚重、真实。

1.1.5

自选其它音乐合成，例如贝多芬第五交响乐的开头两小节。

【分析】

选了一首相对较长的曲子，李健的《假如爱有天意》。自己把整首曲子输入成了曲谱如下所示。但是整首曲子长达近 5 分钟，而 sound 函数无法暂停，因此在展示的 GUI 中只节选了一小段。

【代码】

```
%假如爱有天意
%定义时间
t05 = t(0.5);t1 = t(1);t15 = t(1.5);t2 = t(2);t3 = t(3);t4 = t(4);t6 = t(6);t8 = t(8);t12 = t(12);

%伴奏
meter1 = [zero(t1) mi(t1) so(t1) la(t3) so(t2) fa(t1) mi(t3)];
meter2 = [zero(t1) mi(t1) so(t1) la(t3) xi(t15) la(t05) so(t05) re(t05) mi(t3)];
meter3 = [zero(t1) mi(t1) so(t1) la(t3) xi(t3) xi(t2) re1(t1)];
meter4 = [do1(t3) do(t2) mi(t1) re(t15) xi_1(t05) so_1(t1) la_1(t12)];
line1 = [meter1 meter2 meter3 meter4];

%当天边那颗星出现 你可知我又开始想念 有多少爱恋只能遥遥相望 就像月光洒向海面
meter5 = [mi(t2) mi(t1) re(t15) do(t05) xi_1(t1) do(t2) re(t1) mi(t2)];
meter6 = [mi(t05) so(t05) la(t2) la(t1) xi(t15) la(t05) so(t05) re(t05) mi(t3) zero(t2)];
meter7 = [mi(t05) so(t05) la(t2) la(t1) so(t2) mi(t05) re(t05) mi(t15) fa(t05) mi(t05) re(t05) do(t2)];
meter8 = [la_1(t05) xi_1(t05) do(t2) re(t05) mi(t05) re(t15) xi_1(t05) so_1(t1) la_1(t3) zero(t3)];
line2 = [meter5 meter6 meter7 meter8];

%年少的我们曾以为 相爱的人就能到永远 当我们相信情到深处在一起 听不见风中的叹息
meter9 = [mi(t2) mi(t1) re(t15) do(t05) xi_1(t1) do(t2) re(t1) mi(t2)];
meter10 = [mi(t05) so(t05) la(t2) la(t1) xi(t15) la(t05) so(t05) re(t05) mi(t3)];
meter11 = [zero(t2) mi(t05) so(t05) la(t2) la(t1) so(t2) mi(t05) re(t05) mi(t15) fa(t05) mi(t05) re(t05) do(t2)];
meter12 = [la_1(t05) xi_1(t05) do(t2) re(t05) mi(t05) re(t15) xi_1(t05) so_1(t1) la_1(t3) zero(t1)];
line3 = [meter9 meter10 meter11 meter12];

%谁知道爱是什么 短暂的相遇却念念不忘 用尽一生的时间 竟学不会遗忘
meter13 = [mi(t1) so(t1) la(t3) xi(t3) xi(t2) re1(t1) do1(t2)];
meter14 = [la(t05) la(t05) la(t2) la(t1) so(t15) la(t05) so(t05) re(t05) fa(t1) mi(t2) zero(t1)];
meter15 = [mi(t1) so(t1) la(t3) xi(t3) so_(t2) xi(t1) do1(t3)];
meter16 = [do(t2) mi(t1) re(t15) xi_1(t05) so_1(t1) la_1(t3) zero(t3)];
line4 = [meter13 meter14 meter15 meter16];

%伴奏
meter17 = [mi(t2) mi(t1) re(t15) do(t05) xi_1(t1) do(t2) re(t1) mi(t2) mi(t05) so(t05)];
meter18 = [la(t2) la(t1) xi(t15) la(t05) so(t05) re(t05) mi(t6)];
meter19 = [mi1(t2) fa1(t05) mi1(t05) re1(t3) re1(t2) mi1(t05) re1(t05) do1(t3)];
meter20 = [do1(t2) mi1(t1) re1(t15) xi(t05) so(t1) la(t6)];
line5 = [meter17 meter18 meter19 meter20];

%如今我们已天各一方 生活得像周围人一样 眼前人给我最信任的依赖 但愿你被温柔对待
meter21 = [mi(t2) mi(t1) re(t15) do(t05) xi_1(t1) do(t15) do(t05) re(t1) mi(t2)];
meter22 = [mi(t05) so(t05) la(t2) la(t1) xi(t15) la(t05) so(t05) re(t05) mi(t3)];
meter23 = [zero(t2) mi(t05) so(t05) la(t2) la(t1) so(t2) mi(t05) re(t05) mi(t15) fa(t05) mi(t05) re(t05) do(t2)];
meter24 = [la_1(t05) xi_1(t05) do(t2) re(t05) mi(t05) re(t15) xi_1(t05) so_1(t1) la_1(t3) zero(t1)];
line6 = [meter21 meter22 meter23 meter24];

%多少恍惚的时候 仿佛看见你在人海川流 隐约中你已浮现 一转眼又不见
meter25 = [mi(t1) so(t1) la(t3) xi(t3) xi(t2) re1(t1) do1(t2)];
meter26 = [la(t05) la(t05) la(t2) la(t1) so(t15) la(t05) so(t05) re(t05) fa(t1) mi(t2) zero(t1)];
meter27 = [mi(t1) so(t1) la(t3) xi(t3) xi(t2) re1(t1) do1(t3)];
meter28 = [do1(t2) mi1(t1) re1(t15) xi(t05) so(t1) la(t6) la_(t6)];
line7 = [meter25 meter26 meter27 meter28];
```

```
53
54      %(伴奏)短暂的相遇却念念不忘 多少恍惚的时候 仿佛看见你在人海川流 隐约中你已浮现 一转眼又不见
55 —    meter29 = [zero(t05) so_1(t05) so_1(t05) la_1(t05) la_1(t05) la_1_(t05)...
56          xi_1(t05) do(t05) do_(t05) re(t05) re_(t05) mi(t05) fa(t6) so(t6) mi(t3)...
57          zero(t1) mi(t1) so(t1) la(t3) xi(t3) xi(t2) re1(t1) do1(t2)];
58 —    meter30 = [la(t05) la(t05) la(t2) la(t1) so(t15) la(t05) so(t05) re(t05) fa(t1) mi(t2) zero(t1)];
59 —    meter31 = [mi(t1) so(t1) la(t3) xi(t3) xi(t2) re1(t1) do1(t2)];
60 —    meter32 = [la(t05) la(t05) la(t2) la(t1) xi(t15) do1(t05) xi(t05) so(t05) xi(t1) la(t3)];
61 —    meter33 = [mi(t1) so(t1) la(t3) xi(t3) la(t2) xi(t1) do1(t12) zero(t3)];
62 —    meter34 = [do(t2) mi(t1) re(t15) xi_1(t05) so_1(t1) la_1(t6) zero(t6)];
63 —    line8 = [meter29 meter30 meter31 meter32 meter33 meter34];
64
65      %一转眼又不见 当天边那颗星出现 你可知我又开始想念 有多少爱恋今生无处安放 冥冥中什么已改变 月光如春风拂面
66 —    meter35 = [mi(t2) mi(t1) re(t15) do(t05) xi_1(t1) do(t2) re(t1) mi(t2)];
67 —    meter36 = [mi(t05) so(t05) la(t2) la(t1) xi(t15) la(t05) so(t05) re(t05) mi(t3) zero(t2)];
68 —    meter37 = [mi(t05) so(t05) la(t2) la(t1) so(t2) mi(t05) re(t05) mi(t15) fa(t05) mi(t05) re(t05) do(t2)];
69 —    meter38 = [la_1(t05) xi_1(t05) do(t2) re(t05) mi(t05) re(t15) xi_1(t05) so_1(t1) la_1(t3) zero(t2)];
70 —    meter39 = [la_1(t05) xi_1(t05) do(t2) re(t05) mi(t05) re(t8) do(t4) la_1(t6)];
71 —    line9 = [meter35 meter36 meter37 meter38 meter39];
72
73 —    song = [line1 line2 line3 line4 line5 line6 line7 line8 line9];
74 —    sound(song,fs);
```

【效果】

可以演奏出琴声版的乐曲。

可以看到，虽然看起来很长，但实际上跟简谱占的篇幅相差无几。

1.2 用傅里叶级数分析音乐

1.2.6

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？

【分析】

wavread 函数已经被 Matlab 删去了，应使用 audioread 函数。直接读取并播放即可。

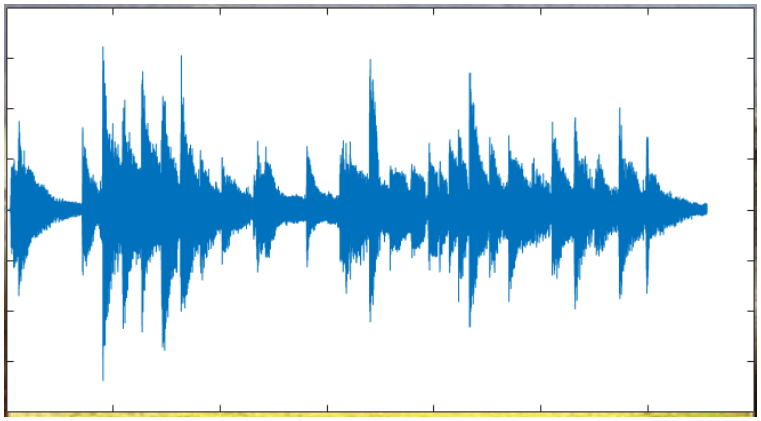
【代码】

```
a = audioread('fmt.wav');
sound(a);
```

【效果】

听起来很像真实拨弦乐器的声音。拨弦感非常强，包络肯定不是指数衰减型的。有很多谐波分量并且相对复杂。

音乐波形如图：



1.2.7

先知道待处理的 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。

【分析】

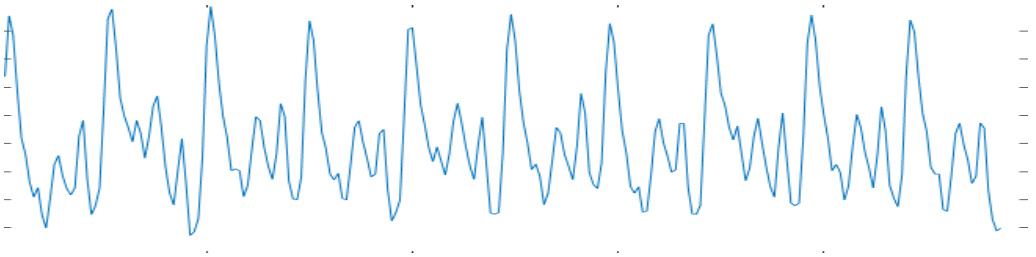
题干要求在时域完成滤波。假设噪声是白噪声，则可通过多个周期求平均的方法来消除一定的噪声。由于给定的 realwave 只有 243 个采样点，却有 10 个周期。直接划分周期会有很大误差。因此利用 resample 函数，将其扩充 10 遍，插值，再叠加取平均。

【代码】

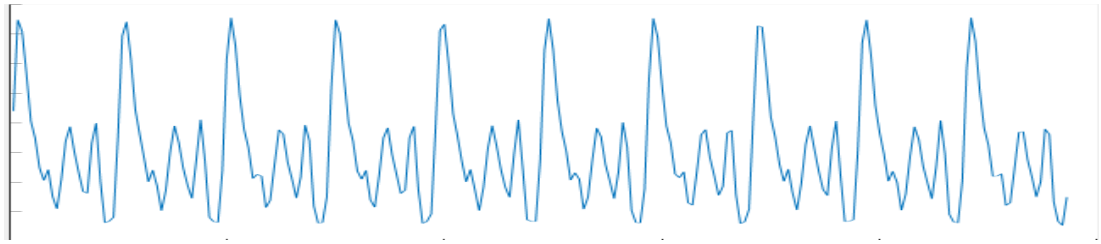
```
% 对realwave进行处理
rep_time = 10;
len = length(realwave);
temp = resample(realwave,rep_time,1);
waverepeat = repmat(temp,rep_time,1);
for i = 1:rep_time-1
    waverepeat(1:len) = waverepeat(1:len) + waverepeat(i*len+1:i*len+len);
end
temp = waverepeat(1:len)/rep_time;
waverepeat = repmat(temp,rep_time,1);
mywave = resample(waverepeat,1,rep_time);
%得到去噪波形mywave
```

【效果】

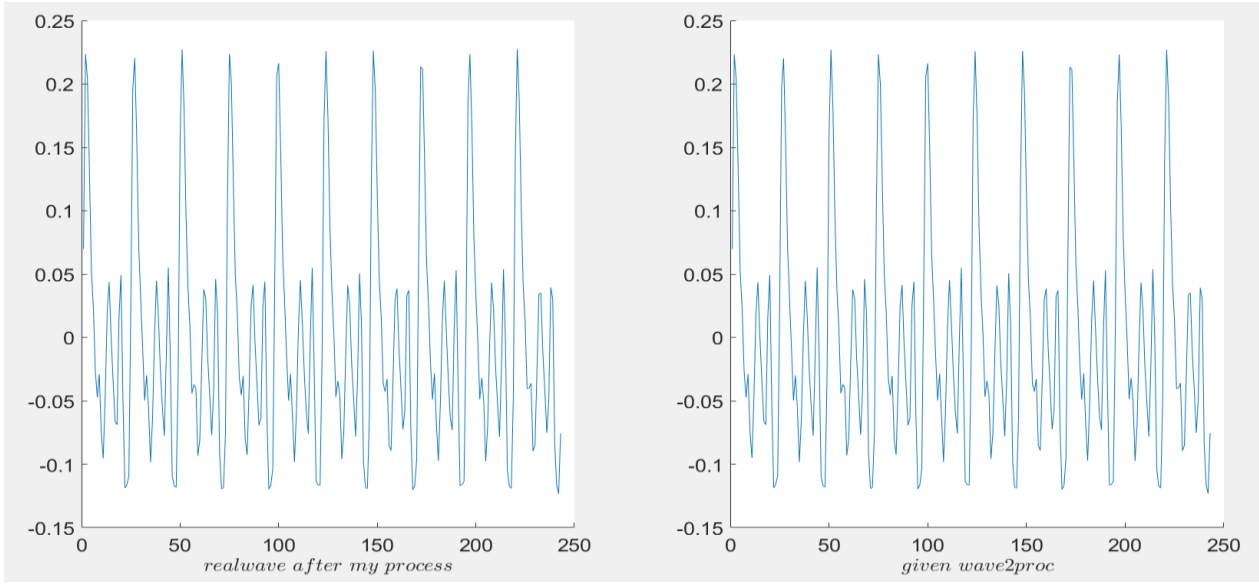
原声 realwave 噪声较大，波形不整齐，周期性不明显：



经过上述处理得到相对整齐的波形：



与给定的 wave2proc 几乎完全相同。对比图如下：



1.2.8

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。

【分析】

直接做傅里叶分析，会发现每个峰都比较宽，误差较大。

由信号与系统知识知，时域数据变多时，频域信号幅度增大而带宽变窄，趋进 δ 函数。此时方便找出峰值。

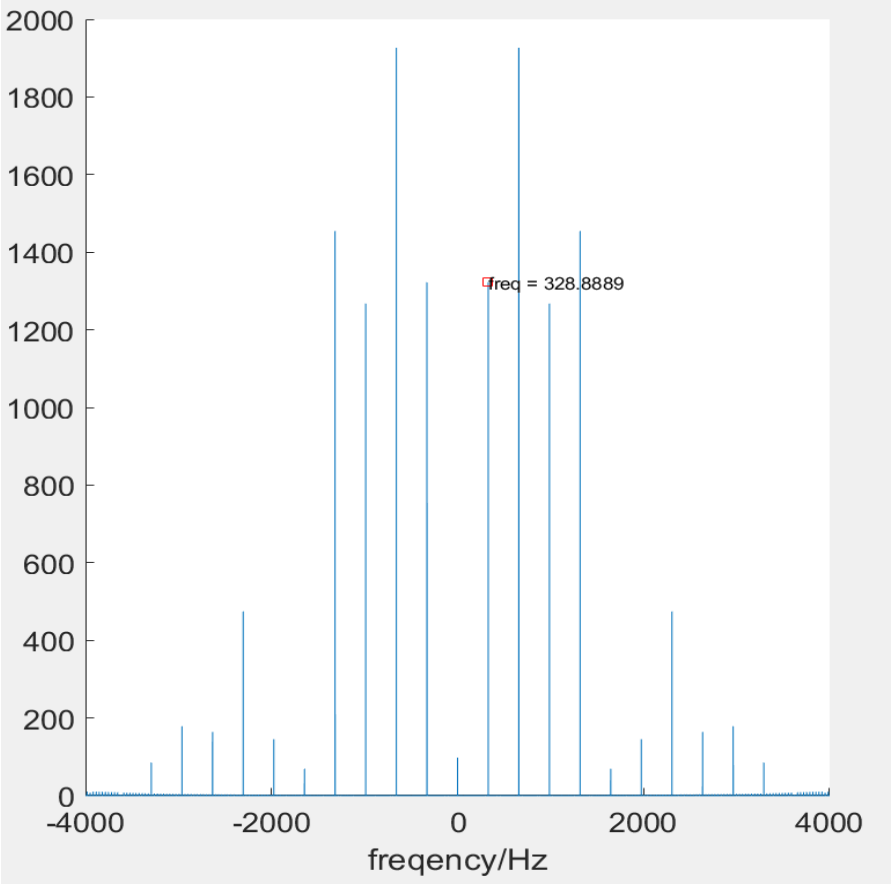
因此，把时域数据量扩充 100 倍，插值、取平均。再做傅里叶分析。

【代码】

```
%% FFT with repetition
sig = repmat(sig,100,1);        %重复100遍
x = fftshift(fft(sig));      %作FFT，并移动零频点到频谱中间
f = 4000*linspace(-1,1,length(x));        %-4000到4000，共有24300个点
amp = 2*abs(x);
%% Find the Base Frequency
benchmark = 1200;
[~,locs] = findpeaks(amp,'MinPeakHeight',benchmark);    %寻找峰值
loc1 = locs(round(length(locs)/2));
for i = loc1-2:loc1+2      %在附近搜寻
    if amp(i) > benchmark
        loc_base = i;
        benchmark = amp(i);
    end
end
loc_dis = abs(loc_base-length(x)/2);
freq_base = loc_dis*8000/length(x);     %基频
```

【效果】

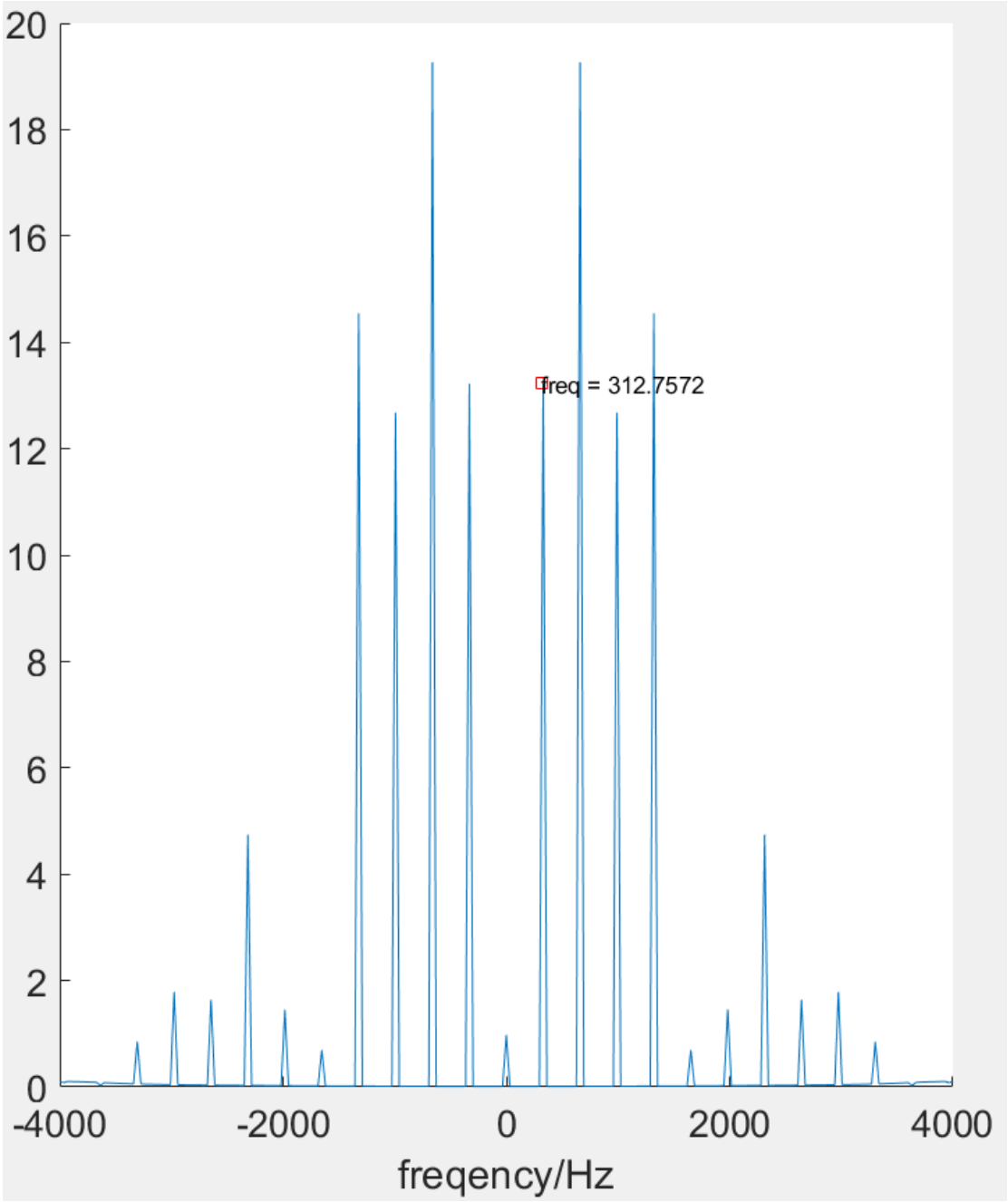
得到相当准确的频谱图：



找到图中标出的基频。与其最相近的是 e1（329.63Hz），相对误差 0.22%。

可判断这段音乐的基频是 328.9Hz，音名是 e1.

另一方面，如果不延拓复制 100 遍，直接傅里叶分析，找到的结果是 312.75Hz，与理论上比较准确的值有一定差距。图示如下：



1.2.9

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。

【分析】

经分析，分析音调和节拍应该分以下步骤：划分音节、筛选基频、记录各基频的各谐波幅度。

划分音节的方法如下。首先通过找局部极大值，获取包络。对包络再次求导，得到连续上升的点。再通过幅度进行筛选。

寻找基频的方法如下。对划分出来的每一个音进行傅里叶分析，仿照之前的方法获得其基频。根据幅度和间距关系进行筛选，其后又进行了二次筛选以排除某基频的谐波。

这可能是本综合实验中最复杂的部分，代码相对较多。

【代码】

首先是划分音符的函数，加窗找符合条件的极大值。

```
%% Divide function
%把一段音乐分出音节
function [index, music,sig,envelope,seg] = DivideMusic(file)
    sig = audioread(file);
    music = sig;
    sig = sig/max(abs(sig));
    %% envelope extraction
    envelope = sig;
    winWidth = 1000;  %尝试出来的，比较合适的窗宽度
    for k = winWidth:length(sig)
        envelope(k) = max(sig(k - winWidth + 1:k));
    end
    %% 分割线seg
    seg = 0.5 * sign(diff(envelope));  %升/降二值化
    seg(seg<0) = 0;
    seg(1:winWidth) = 0;
    index = find(seg>0);
    len = 200;
    for k = 1:length(index)
        if(index(k) + len < length(sig))
            m = max(envelope(index(k):index(k)+len));
        end
        if(m-envelope(index(k))<0.05)
            seg(index(k)) = 0;
        end
    end
```

```
    index = find(seg>0);
    for k = 1:length(index)
        if(seg(index(k))>0)
            seg(index(k)+1:index(k)+winWidth) = 0;
        end
    end
    seg(1000) = 0.5;     %微调第一个音符
    index = find(seg>0);


end
```

其次是提取基频的函数：

```
function [basic_info] = Process(beat,k)
%% 傅里叶处理
N = length(beat);
f = 8000*linspace(0,1,N);
temp = abs(fft(beat));
amp = temp/max(temp);
bbb = amp;
%% 获得频率分量
%找出所有的极大值
local_max_index = find(diff(sign(diff(amp)))<0)+1;
tmp = amp(local_max_index);
amp = 0;
amp(local_max_index) = tmp;
%筛选
max_index = find(amp>0.3);
tmp = amp(max_index);
amp = 0;
amp(max_index) = tmp;
max_index = find(amp>0);
record = zeros(2,length(max_index));
record(1,:) = f(max_index);
record(2,:) = amp(max_index);
%确定基频
N = length(record(1,:));
record = [record;zeros(N)];
unit = 2^(1/24);
basic_info = zeros(10,ceil(N/2));
for t = 1:10
    for it = 1:ceil(N/2)
        bound_a = record(1,it)*t/unit;
        bound_b = record(1,it)*t*unit;
        ind = find((record(1,:)>=bound_a) & (record(1,:)<=bound_b));
        if(length(ind)>1)
            a = find(record(2,:)==max(record(2,ind)));
```

```
            ind = a(1);
        end
        if(ind~=0)
            basic_info(t,it) = record(1,ind);
            record(3,ind) = 1;
        end
    end
end

[~,ind] = max(record(2,:));
[~,c] = find(basic_info==record(1,ind));
tmp = basic_info(:,c); %存储各谐波分量的频率
E = zeros(1,length(c));
if(length(c)>1)
    for it = 1:length(c)
        for t = 1:10
            if(tmp(t,it)~=0)
                E(it) = E(it) + record(2,record(1,:) == tmp(t,it))^2;
            end
        end
    end
    [~,c] = max(E(it));
    tmp = tmp(:,c);
end
if tmp(1)<170 %手工调整
    tmp = [tmp(2:10);0];
end
```

```
strength = zeros(10,1); %存储每一个谐波分量的强度
N = length(bbb);
freq = @(i)tmp(i);
loc = find(tmp>0);
for i=1:length(loc)
    appr = ceil(N*freq(loc(i))/8000.0);
    it = appr-5:appr+5+1;
    [r,l] = max(bbb(it));
    if i == 1
        base_loc = it(l);
        base_str = r;
    end
    strength(loc(i)) = r;
end
basic_info = [tmp; strength]; %返回的basic_info同时含有频率信息和强度信息
%% plot
hold on;
plot(f(1:round((N+1)/2)),bbb(1:round((N+1)/2)),'b');
plot(f(base_loc),base_str,'r*');
xlabel(['f',num2str(k),'=',num2str(freq(1))]);
end
```

写得相对冗长，而且用到了不少循环结构。笔者对此颇不满意，然而多次尝试优化无果。

顶层调用 DivideMusic 函数：

```
%% Devide Music
[index,music,sig,envelope,seg] = DivideMusic('fmt.wav');
```

顶层调用 Process 函数：

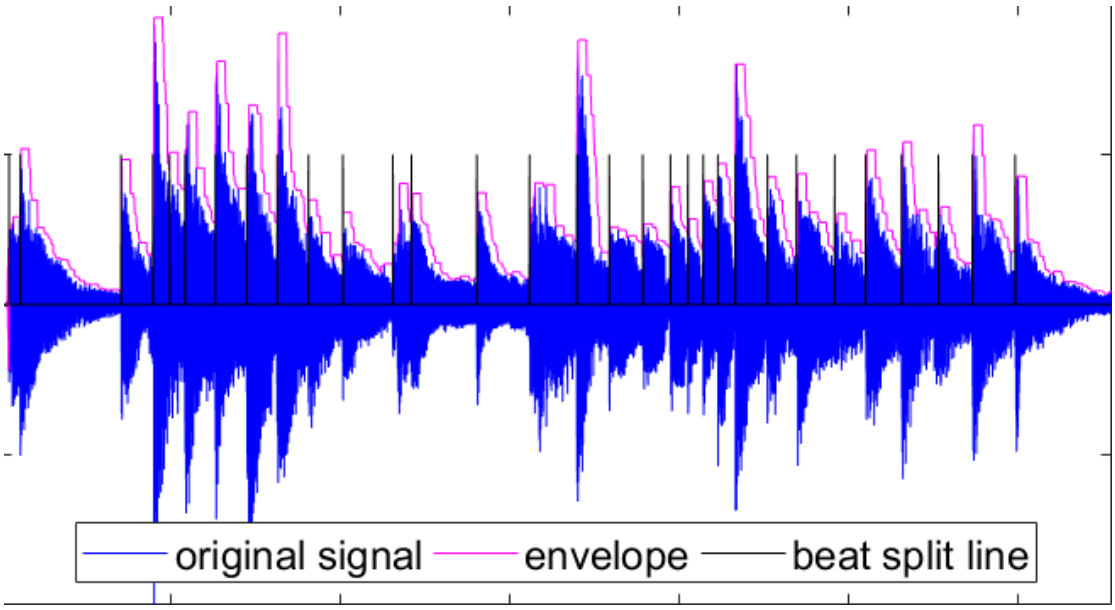
```
%% answer storage
music_data = zeros(20,length(index));    %每个音符所含的谐波频率
time = zeros(length(index),1);  %每个音符所占的时长
%% Analyze each Beat
figure('NumberTitle','off','Name','Base Frequency for each Beat');
for k = 1:length(index)
    %节选出beat
    if k < length(index)
        beat = music(index(k):index(k+1)-1);
    else
        beat = music(index(k):length(music));
    end
    %计算时间，依据是这段音乐全长16秒
    time(k) = length(beat)*16/length(music);
    subplot(4,8,k);

    basic_info = Process(beat,k);
    music_data(:,k) = basic_info;    %把解析出来的频率数据存入music_data
end
```

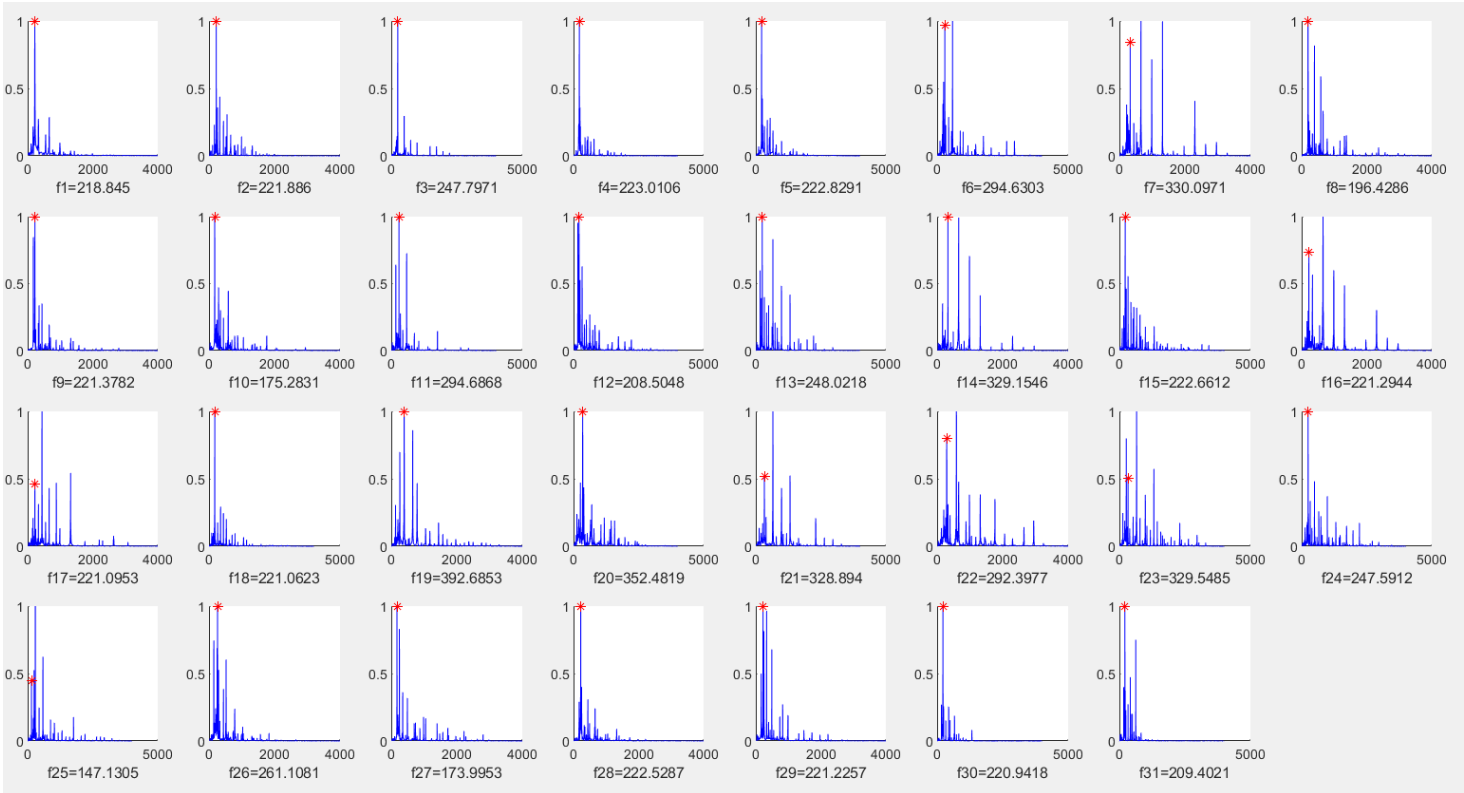
```
%% answer output
music_data = [music_data;time'];
xlswrite("music_data.xlsx",music_data);
```

【效果】

首先是划分音符。以上程序共划分出了 31 个音。用竖线分割如下：



对这 31 个音进行频谱分析，分别找到基频，如图所示：



输出到 Excel 文件 music_data.xlsx 中。所得数据是 31*21 的矩阵。每一行对应一个音符。前十列是频率信息，11-20 列是对应的幅度信息，第 21 列是音符时值。

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	218.845	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0.16077
2	221.886	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1.45251
3	247.797	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0.45728
4	223.011	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0.24097
5	222.829	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0.22363
6	294.63	589.261	0	0	0	0	0	0	0	0	0.96953	1	0	0	0	0	0	0	0	0	0.43433
7	220.065	0	658.037	0	0	1316.07	0	0	0	0	0.38055	0	1	0	0	0.99705	0	0	0	0	0.45276
8	196.429	392.857	589.286	0	0	0	0	0	0	0	1	0.81807	0.58858	0	0	0	0	0	0	0	0.43762
9	221.378	440.586	0	0	0	0	0	0	0	0	1	0.35144	0	0	0	0	0	0	0	0	0.45007
10	175.283	350.566	0	0	0	0	0	0	0	0	1	0.30132	0	0	0	0	0	0	0	0	0.49597
11	294.687	586.658	0	0	0	0	0	0	0	0	1	0.72611	0	0	0	0	0	0	0	0	0.71924
12	208.505	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0.26709
13	248.022	492.93	0	986.898	0	0	0	0	0	0	1	0.33812	0	0.48349	0	0	0	0	0	0	0.94116
14	329.155	0	658.309	0	987.464	0	1315.33	0	0	0	1	0	0.9925	0	0.70544	0	0.41235	0	0	0	0.75964
15	222.661	441.013	662.237	0	0	0	0	0	0	0	1	0.36298	0.32032	0	0	0	0	0	0	0	0.67993
16	221.294	0	657.62	0	0	1317.33	0	0	0	0	0.73311	0	1	0	0	0.48655	0	0	0	0	0.4679
17	221.095	440.162	659.229	880.325	0	1320.49	0	0	0	0	0.46255	1	0.43169	0.47043	0	0.54326	0	0	0	0	0.48157
18	221.062	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0.39771
19	261.79	392.685	0	654.475	789.22	0	0	0	0	0	0.6982	1	0	0.86022	0.46713	0	0	0	0	0	0.25378
20	352.482	700.502	0	0	0	0	0	0	0	0	1	0.31065	0	0	0	0	0	0	0	0	0.21899
21	328.894	657.788	991.314	1315.58	0	0	0	0	0	0	0.51684	1	0.4315	0.52348	0	0	0	0	0	0	0.21094
22	292.398	588.694	0	0	0	1766.08	0	0	0	0	0.80102	1	0	0	0	0.34957	0	0	0	0	0.25061
23	329.548	656.984	986.533	1316.08	0	0	0	0	0	0	0.5057	1	0.38116	0.57302	0	0	0	0	0	0	0.4624
24	247.591	492.847	0	985.693	0	0	0	0	0	0	1	0.48133	0	0.37019	0	0	0	0	0	0	0.41821
25	294.261	0	586.749	0	0	0	0	0	0	0	1	0	0.6238	0	0	0	0	0	0	0	0.55103
26	261.108	522.216	0	0	0	0	0	0	0	0	1	0.60325	0	0	0	0	0	0	0	0	0.44519
27	173.995	349.882	0	0	0	0	0	0	0	0	1	0.36086	0	0	0	0	0	0	0	0	0.51648
28	222.529	441.379	0	0	0	0	0	0	0	0	1	0.30835	0	0	0	0	0	0	0	0	0.53113
29	221.226	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0.49011
30	220.942	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0.6145
31	209.402	416.703	625.405	0	0	0	0	0	0	0	1	0.4727	0.74982	0	0	0	0	0	0	0	1.39453

（music_data.xlsx 示意图，微调后数据略有变化）

由以上数据分析，可以得到各基频及其对应音名、节拍，如下表所示：

序号	测得频率/Hz	标准频率/Hz	相对误差	对应音名	持续时长/s	拍数
1	218.84	220	-0.53%	a	0.1608	0.5
2	221.89	220	0.86%	a	1.4525	3
3	247.80	246.94	0.35%	b	0.4573	1
4	223.01	220	1.37%	a	0.241	0.5
5	222.83	220	1.29%	a	0.2236	0.5
6	294.63	293.66	0.33%	d1	0.4343	1
7	330.10	329.63	0.14%	e1	0.4528	1
8	196.43	196	0.22%	g	0.4376	1
9	221.38	220	0.63%	a	0.4501	1
10	175.28	174.61	0.39%	f	0.496	1
11	294.69	293.66	0.35%	d1	0.7192	1.5
12	208.50	207.65	0.41%	bA	0.2671	0.5
13	248.02	246.94	0.44%	bA	0.9412	2
14	329.15	329.63	-0.14%	e1	0.7596	1.5
15	222.66	220	1.21%	a	0.6799	1.5
16	221.29	220	0.59%	a	0.4679	1
17	221.10	220	0.50%	a	0.4816	1
18	221.06	220	0.48%	a	0.3977	0.75
19	392.69	392	0.17%	g1	0.2538	0.5
20	352.48	349.23	0.93%	f1	0.219	0.5
21	328.89	329.63	-0.22%	e1	0.2109	0.5

22	292.40	293.66	-0.43%	d1	0.2506	0.5
23	329.55	349.23	-5.64%	f1	0.4624	1
24	247.59	246.94	0.26%	b	0.4182	1
25	294.26	293.66	0.20%	d1	0.551	1
26	261.11	261.63	-0.20%	c1	0.4452	1
27	174.00	174.61	-0.35%	f	0.5165	1
28	222.53	220	1.15%	a	0.5311	1
29	221.23	220	0.56%	a	0.4901	1
30	220.94	220	0.43%	a	0.6145	1
31	209.40	207.65	0.84%	bA	1.3945	3

可以看到，频率的误差多在 1%以内，分析是相当准确的。

由这张表可以写出乐谱（笔者不会五线谱，将就着写成了音名表示的简谱）：

---

a a--- b a a d1 e1 g a f d1-- bA bA- e1- a- a a

_a g1 f1 e1 d1 f1 b d1 c1 f a a a bA---

1.3 基于傅里叶级数的合成音乐

1.3.10

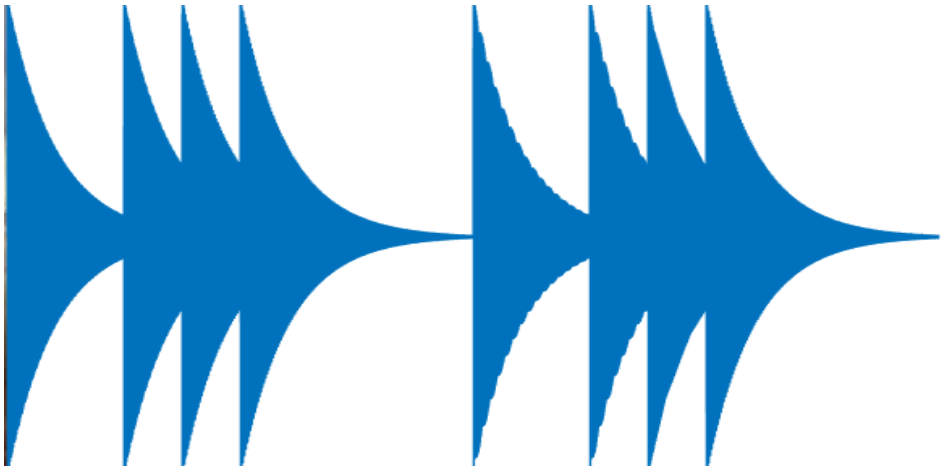
用 (7) 计算出来的傅里叶级数再次完成第 (4) 题，听一听是否像演奏 fmt.wav 的吉他演奏出来的？

【分析】

根据之前分解出的傅里叶各级数的系数，调整 note 函数即可。

【效果】

能听出来大概是弦乐，但是并不太像原来的声音。非常僵硬。波形如下：



1.3.11

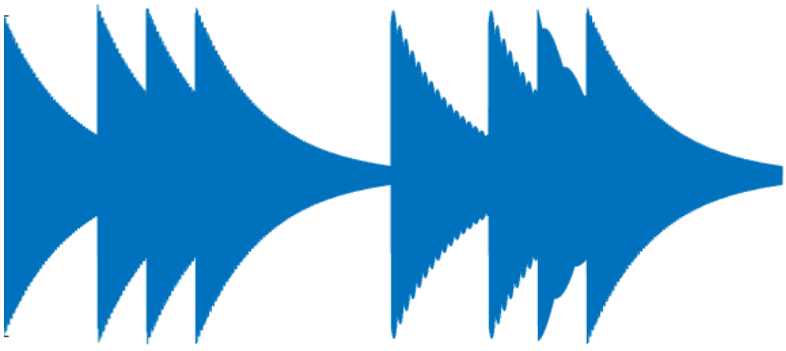
也许(9) 还不是很像，因为对于一把泛音丰富的吉他而言，不可能每个音调对应的泛音数量和幅度都相同。但是通过完成第 (8) 题，你已经提取出 fmt.wav 中的很多音调，或者说，掌握了每个音调对应的傅里叶级数，大致了解了这把吉他的特征。现在就来演奏一曲《东方红》吧。提示：如果还是音调信息不够，那就利用相邻音调的信息近似好了，毕竟可以假设吉他的频响是连续变化的。

【分析】

读取之前分析出来的数据 music_data.xlsx，修改 note 函数，增加泛音。可以得到音色与这把吉他相近的声音。

【效果】

波形如下所示：



可见泛音相对丰富。听起来的确更厚重、真实。

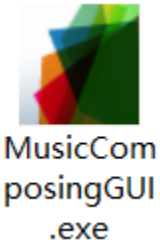
1.2.12

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。

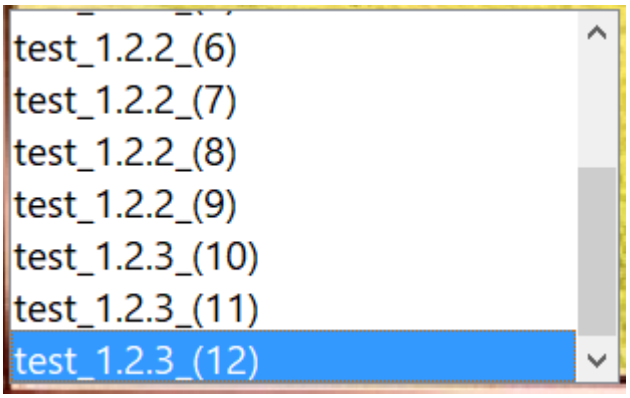
【说明】

做了完成度比较高的 GUI 图形界面，把以上所有问题都封装了起来。并且 package 成了 exe 可执行文件。

详见 MusicComposingGUI.exe 文件：



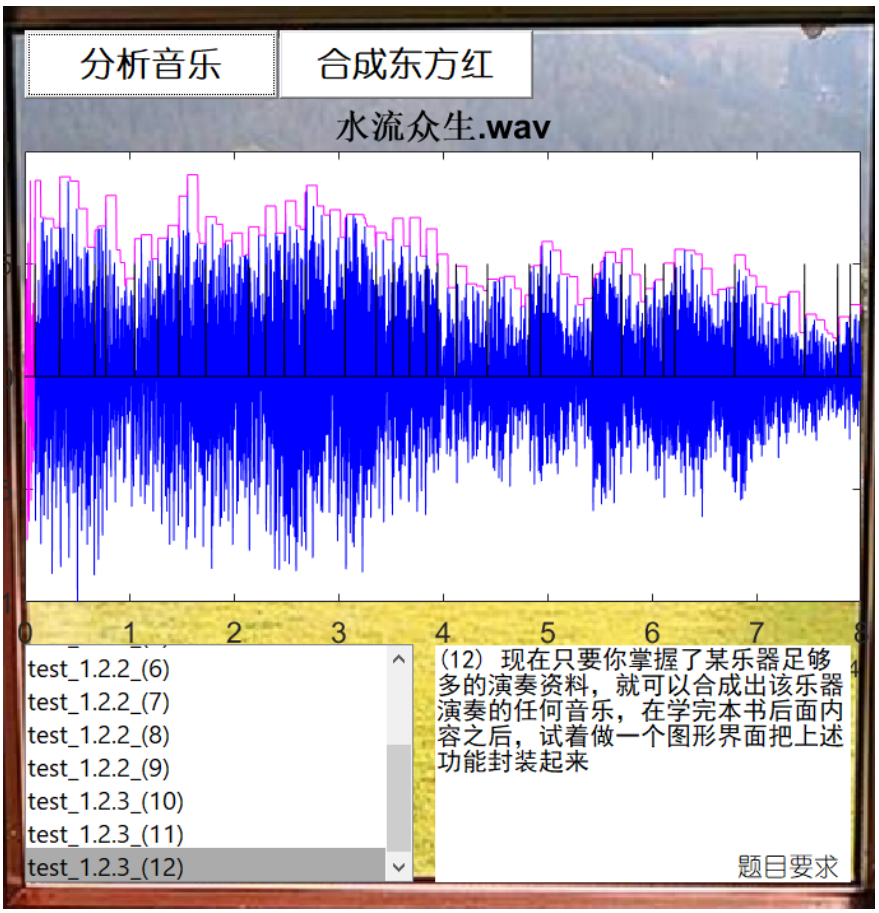
点击选项栏中的题号即可观察该题结果：



整体效果如下：

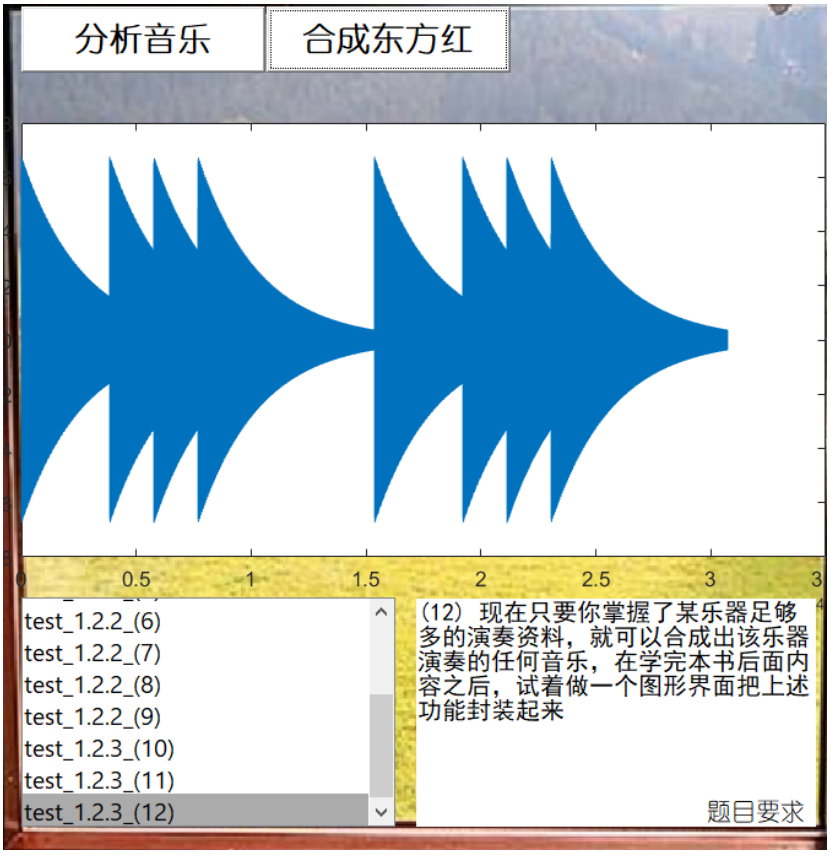
（自选分析与合成界面）

任选一个音乐片段，绘图区可以显示出其波形和音符划分。按照（9）的方法处理数据，并把数据存入 music_data_12.xlsx。



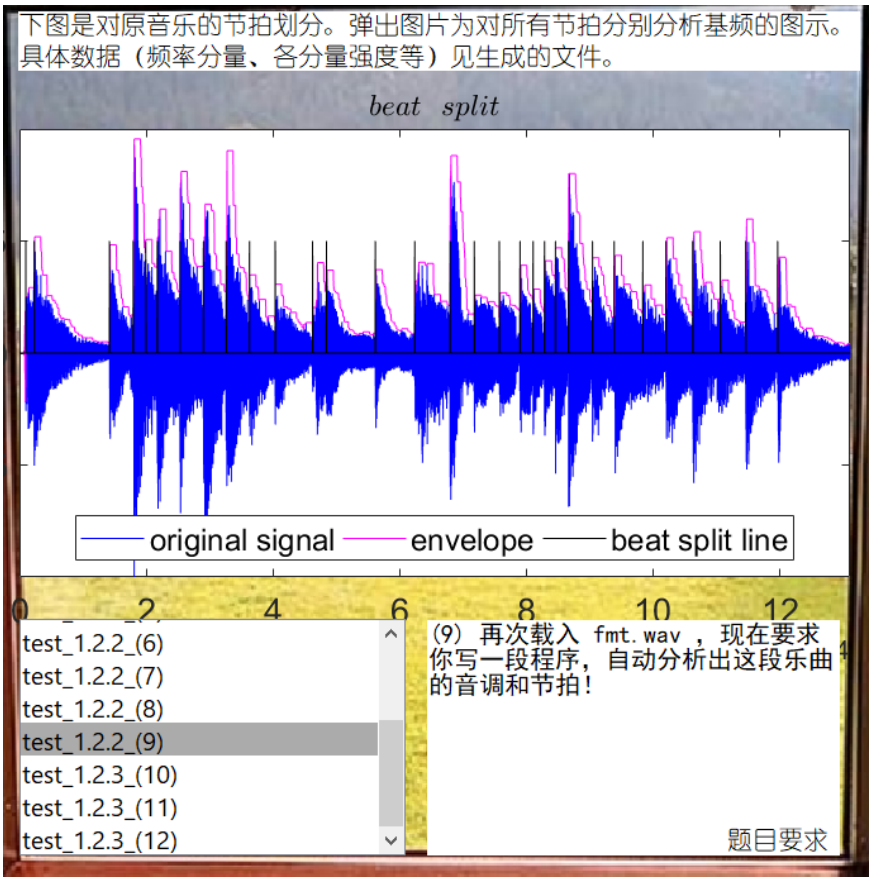
（点击“分析音乐”并且选择一段音乐（只截取了部分界面，下同））

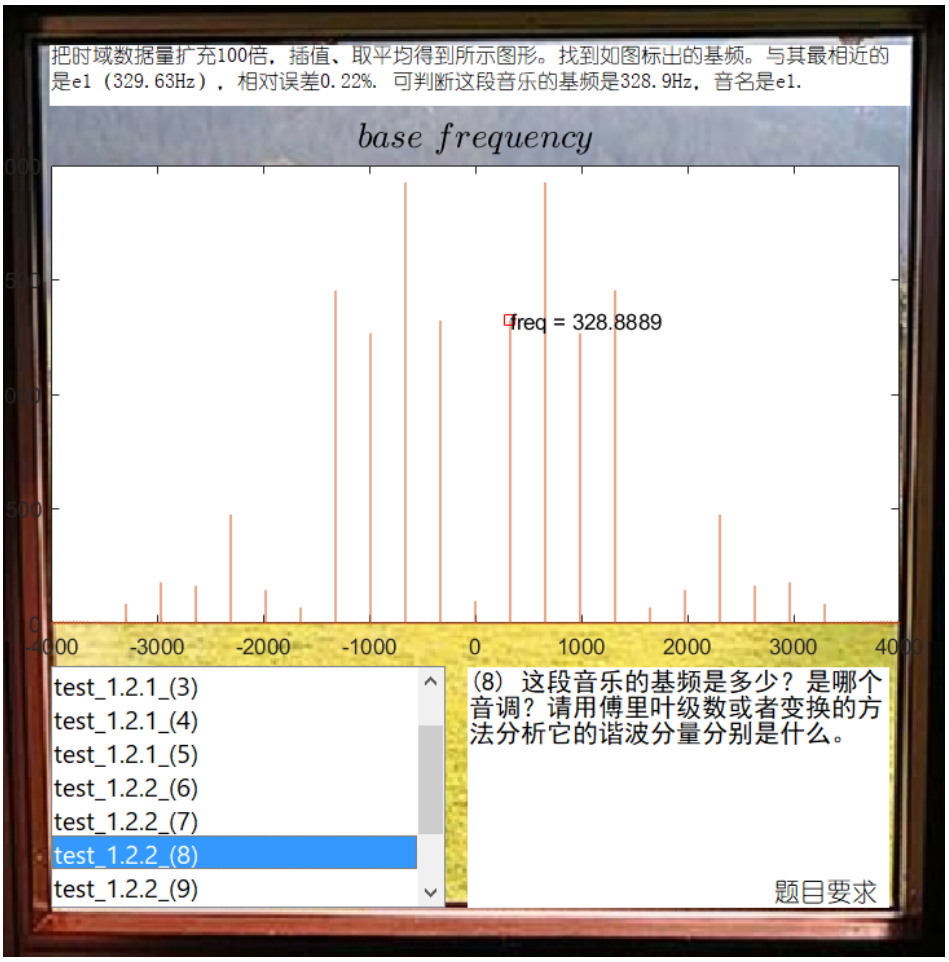
之后便可显示并播放根据选择的音乐进行了改造的《东方红》。即只换音色的《东方红》。



（只换音色的《东方红》波形）

其他所有小问也都封装了。效果如下：





（其他小问界面（节选））

已经通过 deploytool 封装成了 exe 可执行文件，可以直接点击 MusicComposingGUI.exe 文件运行。

第2章 总结

2.1 收获

2.1.1 复习了信号与系统

复习了信号与系统，深化了对时域频域转换关系的认识，深刻理解了抽样与抽样定理。

2.1.2 基本掌握了 Matlab 编程

这个暑假，从 6 月 30 日起，几乎是从零开始学 Matlab。不但认真学习了小蓝书，还学习了人民邮电出版社“求是科技”编著的《MATLAB 7.0 从入门到精通》。在编程的时候，还查阅了大量的文档。算是基本掌握了 Matlab 的应用，体会并且理解了矩阵编程。——尽管在编程的时候还是遗留了不少 C 风格的东西。

本次实验其实不算特别难，但是我走了不少弯路，写了非常久。最终在 29 日完成。总工作量大约写了七百余行。当然，其中过半用于图形界面的设定与控制。

2.1.3 做出了完成度较高的图形界面

熟练掌握了 GUI 的应用，做出了完成度较高的图形界面。

内容：完成了所有的要求，封装了所有的小问。

外形：样式经过了精心挑选、设计（绝对是笔者的最高水平了）；绘图区标注大多使用了 Latex。

操作：操作简便；有提示；鲁棒性高；生成了可执行文件，免去了路径依赖等一系列常见问题。

2.2 遗憾

2.2.1 不能识别太长的音乐

不能识别太长的音乐是因为中间的处理函数写的时候图省事，只针对'fmt.wav'做了一些设定与调整，没有考虑到之后的拓展。

如果要识别长音乐，需要先 resample 并截取到 16 秒以内。即做这样的预处理（受李知航点拨，见第四章引用声明）：

```
[prim, fs] = audioread('twinklestar1.wav');
resample(prim, 8000, fs);
prim = prim(1:160000);
audiowrite('twinklestar1.wav', prim, 8000);
```

2.2.2 没用上 Matlab AppDesigner

这是 Mathworks 在 R2016a 中正式推出的 GUIDE 的替代产品。

由于缺乏教程，documentation 很多细节也没说清楚，所以没有用 AppDesigner 进行界面设计，而是沿用了 GUI.

2.2.3 没实现读谱

图像识别与处理没学明白。

试着找教程跟着做，最后得到的图像都是糊的，无法识别，如下图所示：。

在理论和实践上都存在诸多困难。最后时间不够，遂作罢。

第3章 引用声明

	引用来源	参考内容	非原创等级	说明
1	与同学李知航的交流	更改采样率的方法	完全参考	点拨了 3.2.1 所述的函数。函数很简单，却能使我的程序处理任何音乐。只要进行调整采样率和截取。
1	往届学生裘雨薇的实验报告	对照了第（9）题的基频表	借鉴参考	对照了数据，以此对自己的方法不断进行了调整
2	往届学生王健宇的实验报告	第（9）题获取包络的方法、确定基频的方法	改进参考	方法上收到了启发，有了先作包络、之后作差分求极大值、再筛选的思路。 借鉴了原报告中的 Extract 函数和 JudgeBasicFreq 函数。进行了算法优化和代码优化，精简了部分逻辑和细节，与我自己写的东西融合为了 Process 函数。

3	CSDN	Matlab 技巧	借鉴参考	借鉴了许多关于 Matlab GUI 操作技巧。
4	zhangquan1995 放在 GitHub 上的程序“基于 Matlab 的车牌识别系统”	Matlab GUI；图像识别	借鉴参考	GUI 入门；借鉴了许多用 Matlab 进行图像识别与处理的方法，但是并没有用上。