MATLAB音乐合成大作业报告

马嘉成 无18 2021011966

作业中涉及的所有文件的列表和简要说明（具体功能描述及变量说明见附件的具体文件）

1. harmonics.m: 输入基频、[基频及各次谐波分量的幅度]、时长、总体响度和采样率，得到一段设定时长的由基频及各次谐波分量构成的波形
2. env.m: 包络生成器，输入时长及采样率，输出该时长的预设包络
3. gapgen.m: 生成一段设定时长的空拍
4. musicgen.m: 输入特定格式的包含音乐信息的矩阵，此函数依据音乐信息调用上述3个函数生成一段音乐。有两种模式，将lenequal变量设为0~2s之间的正值可以使每个音持续时间等长，通过控制各音开始时间的间隔控制节拍。若设为负值即可恢复成每个音的结尾对应下一个音的开头，保留迭接效果。
5. appendnote.m(旧方案) : 输入已有音乐、要接到末尾的音符片段、迭接时长及采样率，输出接入该音符后的音乐
6. differential.m: 做差分，功能与自带diff(vec,n)函数一致，但对输入的vec补终值以保证结果与输入的vec向量长度一样
7. smoothwindow.m: 利用Hann窗对源数据做平滑，提取包络
8. findbeatstarts.m: 利用包络提取和差分的结果，实现自动寻找各个音符开始时间
9. toneanalyse.m: 利用傅里叶分析的原理提取基频和各谐波分量
10. task_x.mlx: 对第x小问的具体解答，x从1~11
11. fmt.wav: 即将对其分析节拍、音调和音色的音频
12. Guitar.mat: task7&8用到的吉他音乐数据
13. test.mlx: 对时域谱自相关提取基频的实验性代码
14. test2.mlx: 对频谱自相关提取基频的实验性代码
15. DFH.wav: Task11合成的东方红
16. ReFmt.wav: Task9中自动分析节拍、音调和泛音后利用得到的信息反过来重新演奏的fmt.wav，以便和原曲进行对比
17. freqs.xlsx: 我将涉及到的各个频率存到了该表格中，这样每次只要载入这个表格就可以了，不用把频率写在代码里
18. dongfanghong.xlsx: Task11中用到的东方红曲谱

解答&遇到的问题&解决方案&一些自己的想法(详细代码见附件)

Task1

本题由于刚开始做，一些系统性的设计思路尚不成熟，所以没有用到诸如env.m, harmonics.m等文件的函数。在之后的其他任务中，实现方法会逐渐迭代。对于本题，因为担心程序中数值计算误差的累积，我先根据十二平均律手动计算出各个音符的基频，与表对照无误。然后我用sin函数生成了各个音对应的数组。数组的长度决定了音符的持续时间。最后我把各个音符串接成一个大数组，就可以播放出东方红音乐。

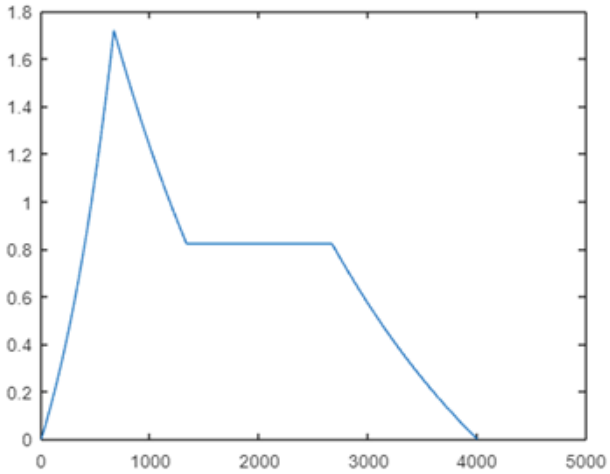
```
clear
clc

thalf = [0:0.000125:0.25]; % 半拍时间向量，采样率8000
tfull = [0:0.000125:0.5]; % 一拍时间向量
%半拍
Gaph = zeros([1,hlent]); % Gap空拍
Ch = sin(523.25*2*pi()*thalf); % 523.25Hz的音
Dh = sin(587.33*2*pi()*thalf); % 以下类似
Gh = sin(392.25*2*pi()*thalf);
Fh = sin(349.23*2*pi()*thalf);
DLh = sin(293.66*2*pi()*thalf);
%全拍
Gapf = zeros([1,flent]);
Cf = sin(523.25*2*pi()*tfull);
```

```
Df = sin(587.33*2*pi()*tfull);
Gf = sin(392.25*2*pi()*tfull);
Ff = sin(349.23*2*pi()*tfull);
DLf = sin(293.66*2*pi()*tfull);
%sound
sound([Cf,Ch,Dh,Gf,Gapf,Ff,Fh,DLh,Gf,Gapf],8000); % 播放
```

播放效果音调正确，但注意到在每个音开始与结束的时候，会出现较为明显的破音，这是因为相位的不连续产生了高频分量。

Task2

为了解决Task1中遇到的问题，应该确保每个音开始和结束时其相位都接近或等于0，即波形在时域上连续。所以我设计了包络，使用分段函数使得包络首尾均为0值。



将这个包络点乘上sin生成的音频，发现每个音的开始和结束阶段过渡较为平滑，没有破音了。

增加的具体关键代码为

```
hlent = length(thalf); % 半拍采样点个数
flent = length(tfull); % 一拍采样点个数
t3h = linspace(0,1,ceil(hlent/3)); % 半拍的1/3长度时间向量
t6h = linspace(0,1,ceil(hlent/6)); % 半拍的1/6长度时间向量
envh = [exp(t6h)-1,exp(1-t6h/2.5)-1,(exp(0.6)-1)*ones([1,hlent/3-1]),exp(0.6*(1-t3h))-1]; % 半拍所用包络
t3f = linspace(0,1,ceil(flent/3)); % 一整拍 同理
t6f = linspace(0,1,ceil(flent/6));
envf = [exp(t6f)-1,exp(1-t6f/2.5)-1,(exp(0.6)-1)*ones([1,ceil(flent/3)-1]),exp(0.6*(1-t3f))-1];
% 全拍所用包络
```

Task3

所谓升高或降低音调，就是将频域进行伸缩变换，或者说是对时域进行相反的伸缩变换（FT相似性）。如果将播放时的采样率翻倍，就相当于在时域上压缩至1/2，频域上乘2，就达到了升八度的效果。降低八度也是同理。所以我采用了这样的代码实现升高或降低八度

```
sound(music,16000); %通过播放时采样率倍乘来实现升八度
sound(music,4000); %通过播放时降低采样率实现降八度
```

对于升高半个音阶，原理类似，将频率变为原来的$2^{1/12} \approx 1.06 = \frac{53}{50}$倍即可

```
musicH = resample(music,50,53); %通过升采样升半个音阶
```

Task4

写这道题时我意识到自己之前给每个音都单独写一个数组存起来的方法是不合适的。这样不方便加谐波分量，而且通用性与代码的复用性太差。所以我写了musicgen函数以及其调用的gapgen,和harmonics 函数，做好封装。

我规定音乐数据格式为

$$\begin{bmatrix} time_1 & pitch_1 & mag_1 & harmonics_1 \\ time_2 & pitch_2 & mag_2 & harmonics_2 \\ time_3 & pitch_3 & mag_3 & harmonics_3 \\ time_4 & pitch_4 & mag_4 & harmonics_4 \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix}$$

其中time为该音持续时间(s)；pitch为音调，由对应基频的序号所表示；mag为整体幅度，在整个音的幅度上乘一个系数；harmonics为一行向量，各元素表示该次谐波分量的幅度。

本题中，我为所有音设置的谐波分量均为`harmon = [1 0.2 0.3 0.2 0.1 0.1 0]; %谐波分量幅度(1x 2x 3x 4x.....7x)不需要的分量补0占位`

最终代码如下

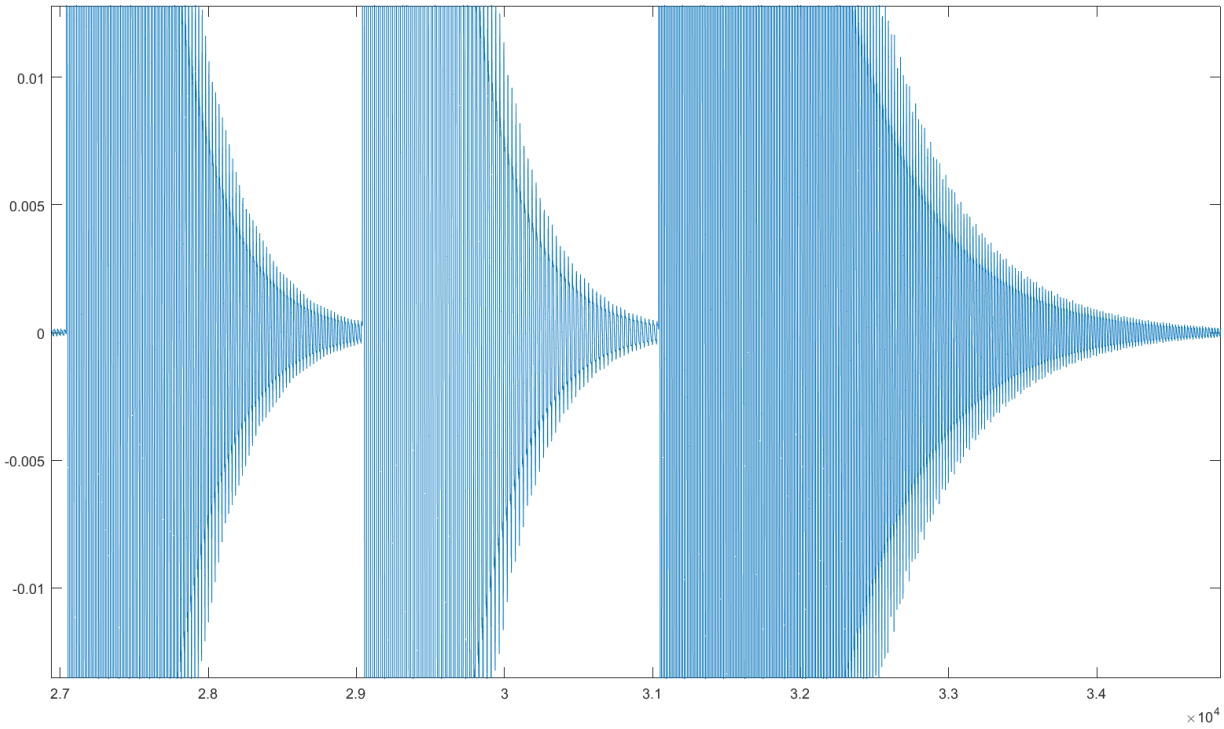
```
clear
clc

freq = [174.61 184.99 196 207.65 220 233.08 246.94 261.63 277.18 293.66 311.13 329.63 349.23
369.99 392 415.30 440 466.16 493.88 523.25 554.36 587.33 622.25 659.25 698.45]; %各个音符的频率，防止误差累积，不用数值计算方法
srate = 8000; %采样率
xfadet = 0.120; %渐变用时
harmon = [1 0.2 0.3 0.2 0.1 0.1 0]; %谐波分量幅度(1x 2x 3x 4x.....7x)不需要的分量补0占位

%music
musicdat = [0.5 0.25 0.25 0.5 0.5 0.5 0.25 0.25 0.5 0.5; 20 20 22 15 0 13 13 10 15 0; 1 1 1 1 1
1 1 1 1 1]';%音乐数据矩阵[持续时间,音符,幅度]
harmonrepeat = repmat(harmon,length(musicdat(:,1)),1); % 表示谐波分量矩阵
musicdat = [musicdat,harmonrepeat]; %矩阵拼接，形成最终音乐数据
music = musicgen(freq,musicdat,srate,xfadet,-1); % 生成音乐

%sound
sound(music,8000);
plot(music);
```

其中musicgen中实现**迭接**的代码为

```
%vec为音乐数组，已事先分配好空间并初始化为0
vec(currIndex-pamt+1:currIndex-pamt+notelen)=vec(currIndex-pamt+1:currIndex-
pamt+notelen)+notetmp; %将当前音符接入已有音乐末尾
```

思路就是线性叠加

从最终音乐波形中，我截取出一段



可以看出生成音乐的功能基本实现，实现了连接功能，但连接不够明显。这是由于包络设置的衰减过快导致的。另外，我认为这个包络在音持续时长更长（例如1~2秒）时的效果更好，若时间太短就会使包络也跟着衰减更快，没有吉他拨弦后的余音。

对此两个问题我的解决办法是对musicgen函数进行改进。原来的musicgen函数声明为

```
function vec = musicgen(freqs,musicdat,samplerate,xfadetime)
```

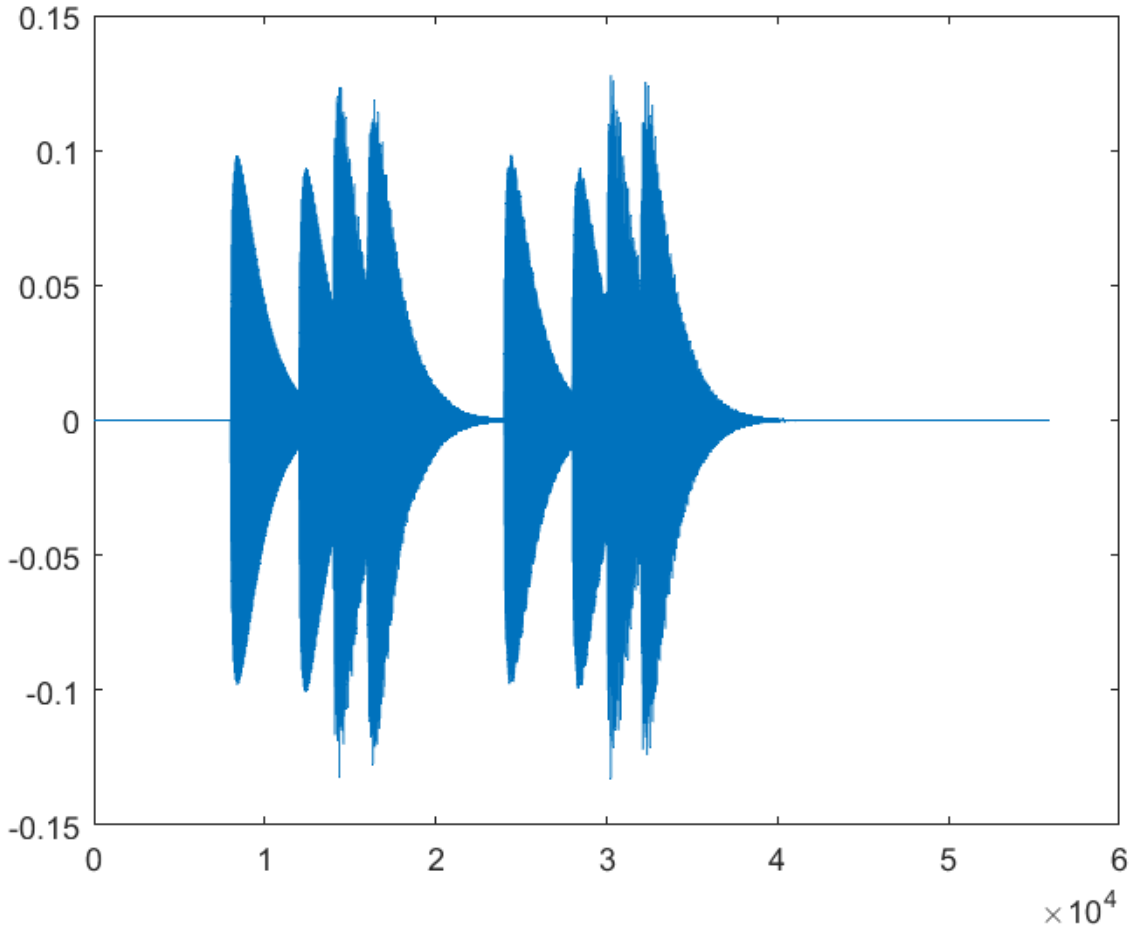
我将其改为

```
function vec = musicgen(freqs,musicdat,samplerate,xfadetime,lenequal)
```

新加入的*lenequal*意为各音延续时长是否相等。若设为一正值$x \in (0,2)$，则每个音的延续时长均为x秒。通过控制相邻两个音开始时间的间隔，我可以控制节拍，实现代码为

```
for x = 1:1:length(musicdat(:,1)) %此循环用于自动加入连接，使音符之间连贯
        if musicdat(x,2) ~= 0 %音乐矩阵的第二列为音符序号
            [notelen,notetmp] =
harmonics(freqs(musicdat(x,2)),lenequal,limitMag*musicdat(x,3),musicdat(x,4:end),samplerate);
            [~,envelope] = env(lenequal,samplerate);
            notetmp = notetmp.*envelope;
        else
            [notelen,notetmp] = gapgen(lenequal,samplerate); %如果音符序号为0，代表一个空拍
        end
        legallen = floor(musicdat(x,1)*samplerate); %已经生成音乐的有效长度（即0时刻到当前最后一个音的开始时刻）
        vec(currIndex+1:currIndex+notelen)=vec(currIndex+1:currIndex+notelen)+notetmp; %将当前音符接入已有有效音乐末尾
        currIndex = max(currIndex,currIndex+legallen);
    end
```

$notelen_i$为第i个音开始到第i+1个音开始的时间，这样在加入第i+1个音时，从第i个音开始处延后$notelen_i * samplerate$个点开始接入。实测将lenequal设为1.5后，得到结果如下



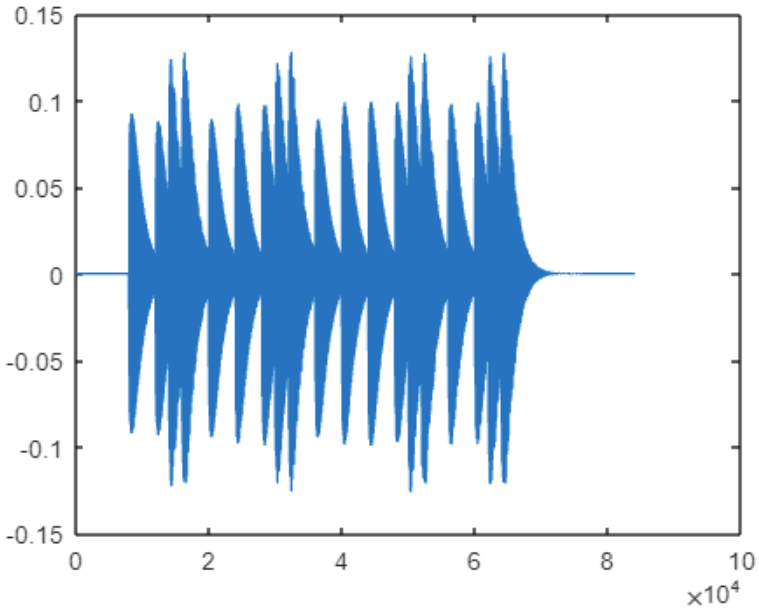
可见迭接效果明显，听起来也更加自然。

最后，在加入谐波分量后，音色确实更加饱满了，有些拨弦类乐器的感觉了。

Task5

```
clear
clc

freq = [174.61 184.99 196 207.65 220 233.08 246.94 261.63 277.18 293.66 311.13 329.63 349.23
369.99 392 415.30 440 466.16 493.88 523.25 554.36 587.33 622.25 659.25 698.45]; %各个音符的频率，防止误差累积，不用数值计算方法
srate = 8000; %采样率
xfadet = 0.120; %渐变用时
harmon = [1 0.2 0.3 0.1 0 0 0]; %谐波分量幅度(1x 2x 3x 4x......7x)，不需要的补0占位

%music
musicdat = [0.5 0.25 0.25 0.5 0.5 0.5 0.25 0.25 0.5 0.5 0.5 0.5 0.25 0.25 0.5 0.5 0.25 0.25 0.5;
13 13 17 20 20 22 25 22 20 20 17 17 20 17 13 10 13 17 20; 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1]';%音乐数据矩阵[持续时间,音符,幅度]
harmonrepeat = repmat(harmon,length(musicdat(:,1)),1);
musicdat = [musicdat,harmonrepeat];
music = musicgen(freq,musicdat,srate,xfadet,1.3); %生成音乐

%sound
sound(music,8000);
plot(music);
```

通过重新写谱，我用以上代码演奏了清华大学校歌"西山苍苍，东海茫茫，吾校庄严，巍然中央"一句的音乐。效果个人还比较满意，波形图如下



Task6&7&8

首先打开并播放fmt.wav

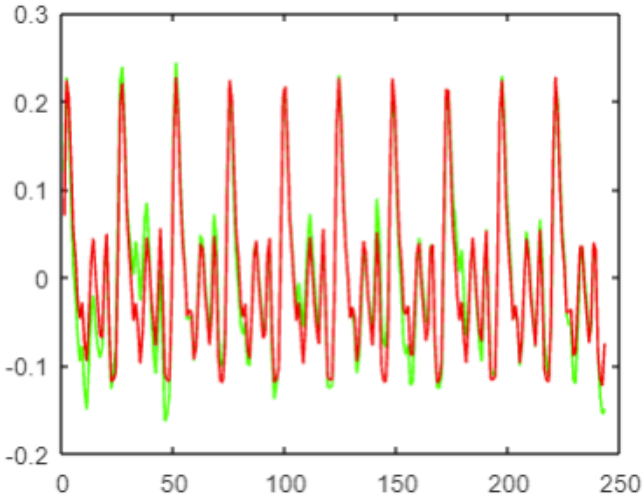
```
fmt = audioread('fmt.wav');
sound(fmt,8000);
```

像是一把真实的吉他演奏的，非常自然，顺畅，比前几问中我合成的音乐更加悦耳。这很可能是由于演奏这段音乐的吉他拥有更加丰富的泛音。

载入Guitar.MAT，观察到待处理波形近似为10个整周期，故对wave2proc做如下处理

```
load("Guitar.MAT");
upsample = resample(realwave,10,1); % 对原音频做升采样，变为2430个点
meansample = upsample(1:243); % 将整个时域波形截为10个片段，各片段求平均
for x = [1:9]
    meansample = meansample + upsample(x*243+1:(x+1)*243);
end
meansample = meansample/10;
sample = repmat(meansample,[10,1]);
sample = resample(sample,1,10); % 降采样还原为243个点
```

升采样是为了正好能使片段十等分。通过求平均，将高频随机噪声去除，得到较为理想的待处理波形。最后得到对比图像如下

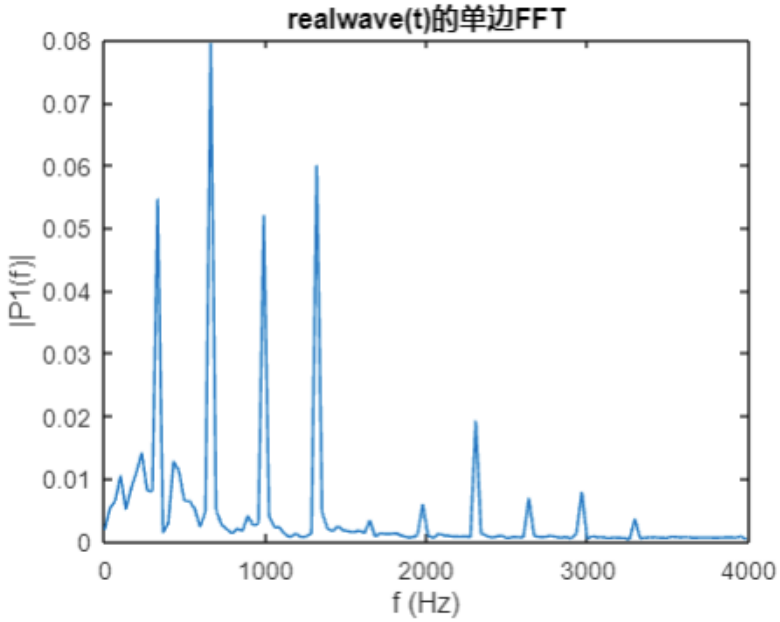


可见处理效果较好。

下面需要分析波形的频域信息。对realwave直接做傅里叶分析

```
Fs = 8000; %采样率
L = length(realwave); %信号长度
T = 1/Fs; %采样周期
t = (0:L-1)*T %时间轴
F = fft(realwave);
FP2 = abs(F/L); %计算幅度
FP1 = FP2(1:floor(L/2)+1); %取半边
FP1(2:end-1) = 2*FP1(2:end-1); %除0频以外其余频率翻倍
f = Fs*(0:(L/2))/L; %计算对应频率
plot(f,FP1)
title("realwave(t)的单边FFT")
xlabel("f (Hz)")
ylabel("|P1(f)|")
```

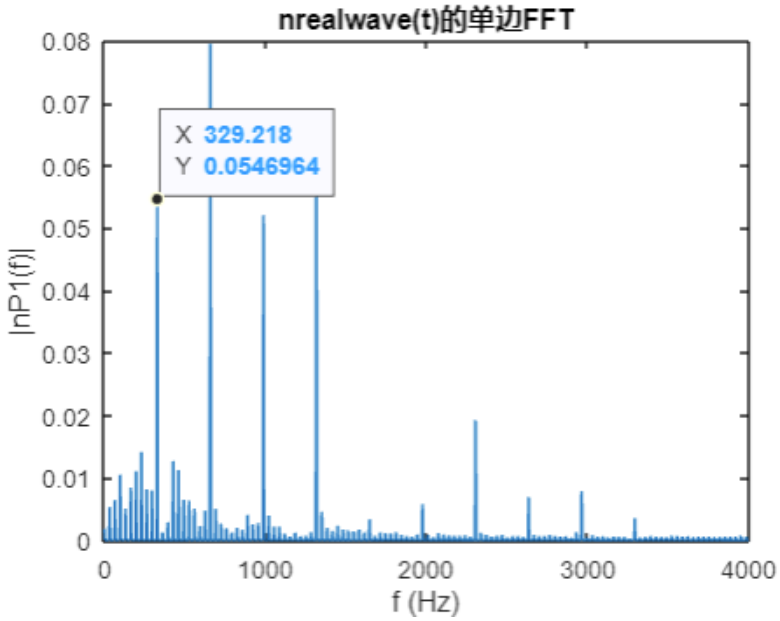
直接分析得到的频谱如下



易见各峰较宽，不光峰值有较大的幅度，其周边的临近频率也有较大的分量。将原序列重复10次

```
nrealwave = repmat(realwave,[10,1]);
nFs = 8000; %采样率
nL = length(nrealwave); %信号长度
nT = 1/nFs; %采样周期
nt = (0:nL-1)*nT; %时间轴
nF = fft(nrealwave);
nFP2 = abs(nF/nL);
nFP1 = nFP2(1:floor(nL/2)+1);
nFP1(2:end-1) = 2*nFP1(2:end-1);
nf = nFs*(0:(nL/2))/nL;
plot(nf,nFP1)
title("nrealwave(t)的单边FFT")
xlabel("f (Hz)")
ylabel("|nP1(f)|")
xlim([0 4000])
ylim([0.000 0.080])
ax = gca;
chart = ax.Children(1);
datatip(chart,329.2,0.0547);
```

得到频谱如下



频谱更接近理想的离散谱了。可见基频为329.218Hz，查表得音调为E4。

之所以重复会增加信号的频域分辨率，是因为重复后频率分量不改变，但由于主值区间点数N增加，$\Delta\omega = \frac{2\pi}{NT_s}$变小，频域分辨率提升，原先的连续谱近似变为若干峰值。

Task9

这一问我通过"音频节奏检测(Onset Detection)"https://blog.csdn.net/matrix_laboratory/article/details/53709638这篇文章获得了大致的思路，从和同学的交流中了解到了Hann窗，没有参考除官方文档和教材之外的其他代码。

我的想法大致是分析节拍关键是要标定每个音的起止处。由于吉他靠拨弦发生，其attack部分上升会较快，有一个类似阶跃的信号。这样只要能检测阶跃，就能掌握每个音的开始时间。但原信号较为杂乱，需要进行一定的处理。

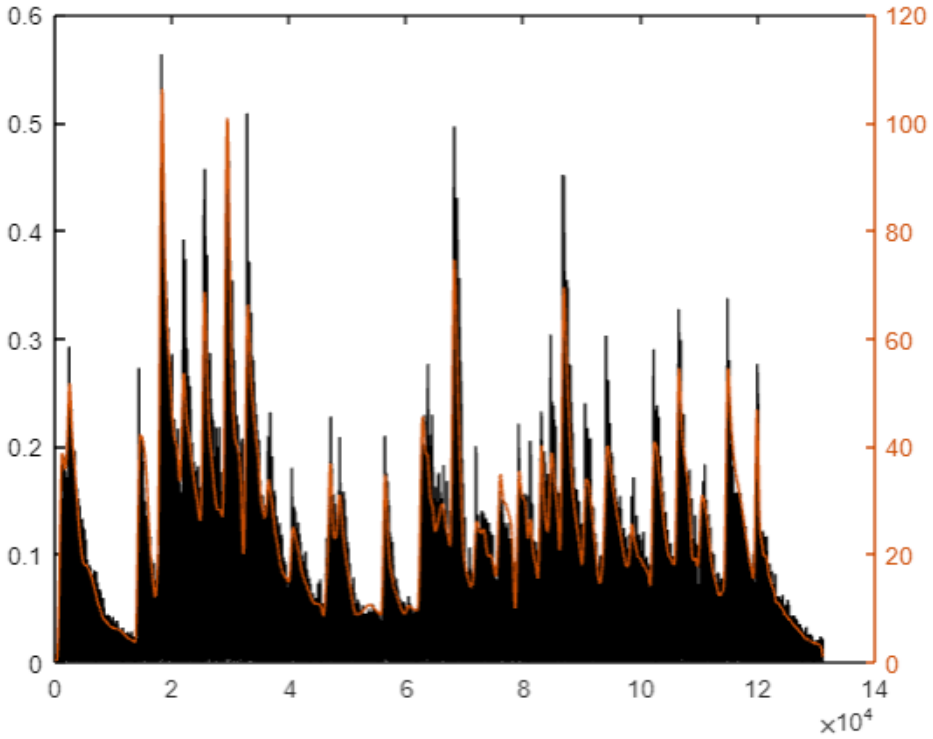
我从上面提到的那篇文章中看到有一个类似包络提取的步骤，刨除具体的每次振荡，只保留幅度随时间的大致变化，确实更有利于对阶跃特征的提取。经过和同学交流，我了解到Hann窗可以做到这一点。

首先，由于我并不关心信号的正负，只对其绝对值感兴趣，我将原信号取绝对值。

```
clear
clc

music = audioread('fmt.wav'); %读文件
cla reset
plot(music);
mabs = abs(music); %时域幅度谱
```

之后，我用Hann窗对该信号进行处理，得到包络

```
yyaxis left
plot(mabs);
hold on
windowsize = 1067; %平滑窗大小
offsetmax = windowsize-1;
negoffsetmax = -offsetmax;
offset = 114;%偏置大小
smusic = smoothwindow(mabs,windowsize,offset);
yyaxis right
plot(smusic);
hold off
```



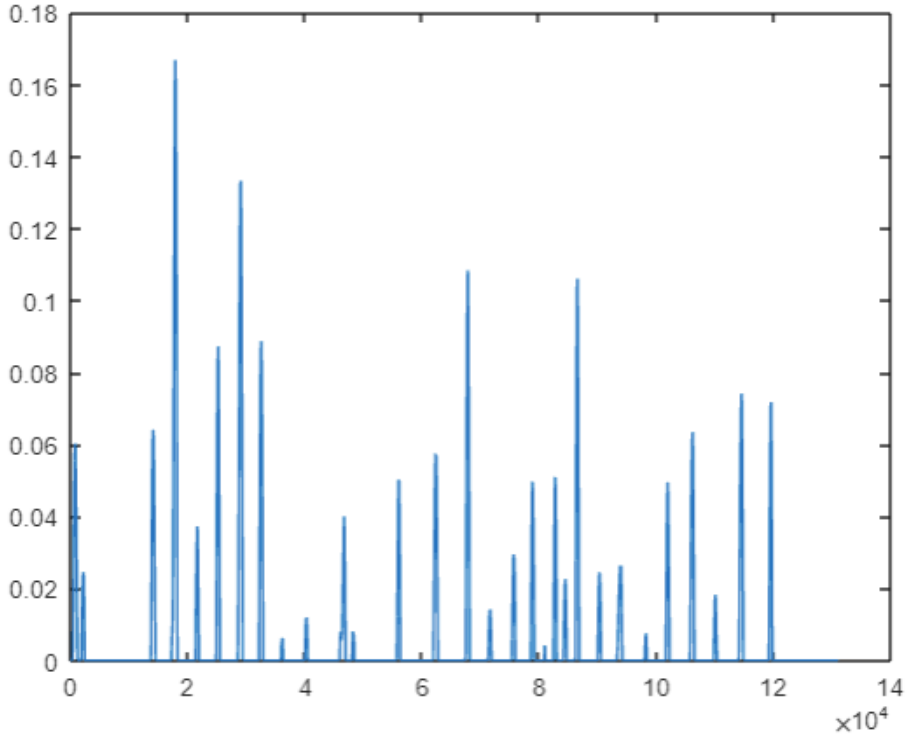
红色线为包络，黑色为绝对值信号。可见从大致形态上二者很接近，更重要的是开始时间对齐的效果较好。这是因为我对窗口长度`windowsize`和偏移量`offset`在livescript里通过拉slidebar的方式进行了调参。其中`offset`是我自己对这个方法的改进。我观察到吉他的包络呈现的是一个类似 **急升-指数衰减** 形态。而这样的包络在时域上既不窄，也不对称。如果不对平滑窗加一个偏置，则会使得到的包络峰值位置比每个音的起始位置偏后。所以我在计算每个点的包络值时，将对应的Hann窗向后移了一些，使得包络的峰值位置提前，和原信号峰值对齐。具体的实现代码如下

```
function vec = smoothwindow(music,windowsize,offset)
    %music待处理音乐波形 windowsize为Hann平滑窗大小，取奇数
    %offset提供偏置，可移动平滑窗中心位置以应对不对称的包络，正数为右移，取0时平滑窗居中，最多移
floor(windowsize/2)
    len = length(music); %要平滑的音频的长度
    hwindsize = floor(windowsize/2); %half windowsize;
    music = [zeros(windowsize-1,1);music;zeros(windowsize-1,1)]; %开头与末尾补零
    %wind = sum(music(windowsize+1-hwindsize+offset:windowsize+1+hwindsize+offset,1)); %初始化
    Hwind = hann(windowsize); %生成Hanning窗
    vec = zeros(len,1);
    %vec(1,1) = wind;
    for x=1:len %用Hann内积Hann窗
        %wind = wind-music(windowsize+x-1-hwindsize+offset,1)+music(windowsize+x-
1+hwindsize+offset,1);
        vec(x,1) = music((windowsize+x-hwindsize+offset):
(windowsize+x+hwindsize+offset),1)'*Hwind;
    end
end
```

接下来，只需对包络"求导"(这里是做差分)，得出斜率为较大正值的地方，就可以得出每个音大致的开始时间了。

```
cla reset
plot(smusic);
difn =1;
threshold = 0.01;%阈值
dmusic = differential(smusic,difn)-threshold;
plot(dmusic);
dmusic(dmusic<0)=0;
plot(dmusic);
```

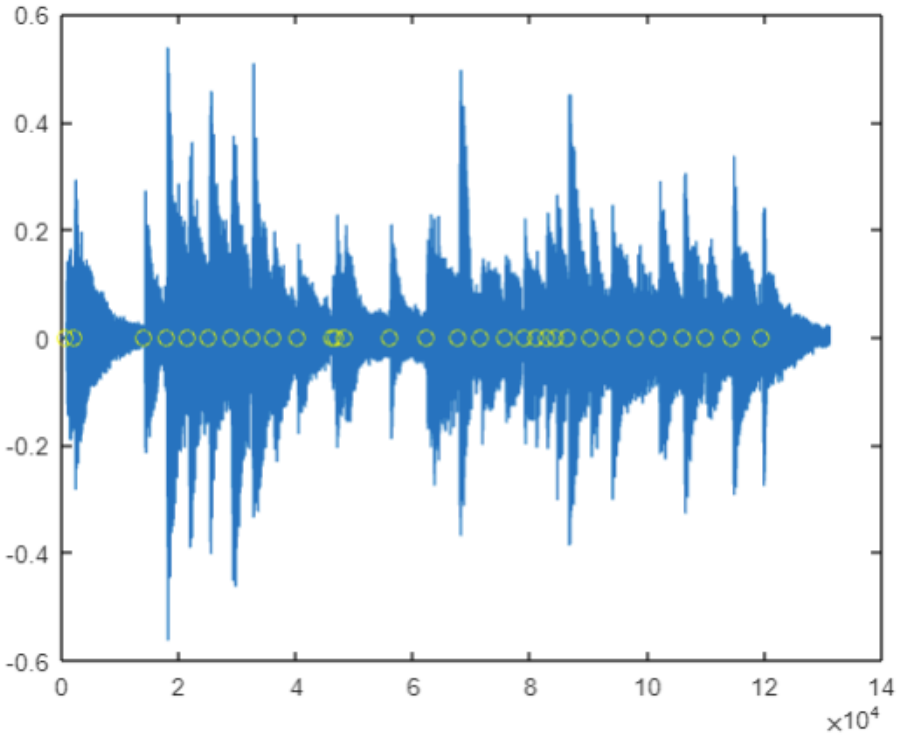
得到的结果如下



可见结果较为干净，便于进一步提取。接下来，通过`findpeaks`来寻找斜率最大的地方。由于吉他的attack阶段上升特别快，所以基本上峰值就是每个音的开始时间。一开始我直接用`findpeaks`，发现会在每个主峰的周围找到很多小峰，这是由于差分结果虽然乍一看是平滑的，但实际上仍有微小的起伏。于是我设置了一个判断，一旦发现了一个峰，后边一定范围内再有新的峰也会被过滤掉，实测由于"伪峰"之间相距很近，这种方法处理后的效果仍然较好。我将上述找起始点和过滤"伪峰"的功能封装为`findbeatstarts`函数，具体代码如下

```
function vec=findbeatstarts(music)
% music:对其分析各音符开始位置
    [~,locs]=findpeaks(music);
    vec=[];
    lastvalloc=-300; %上一处被采纳的峰值的位置
    for x=1:length(locs)
        if(locs(x)-lastvalloc<300) %如果距上一个峰太近，则忽略掉
            continue
        else
            vec=[vec,locs(x)];
            lastvalloc=locs(x);
        end
    end
end
```

最后将识别的结果叠加在原始波形上



可以看出识别较为准确。

在此基础上，我开始进行音调和谐波的识别和提取。

这部分的核心是`function [num, mag, basefreq, harmo] = toneanalyse(ndat,nFs,freqs,maxhmnc)`模块。其各个关键变量的含义如下

```
%num基波序号 mag音符幅度 basefreq基频 harmo各次谐波幅度系数向量 ndat待分析数据 freqs涉及到的各种频率 nFs采样率
%maxhmnc 取的最高次谐波
```

该模块的基本原理是傅里叶分析。首先对时域波形ndat进行傅里叶变换得到频域谱。

现在开始音调识别。所谓音调识别，就是找基频。我认为如果截取的音频合适，那么每一段音频频谱的最大峰值对应的频率不是基波也是谐波分量。所以我先找寻频谱的最大值。我还假设基频的幅度不会很小，因为如果基频幅度过小，那么听起来基频会被盖过去，显然在fmt.wav中没有这种情况。综上，我先对频谱做了这样的处理

```
[ampmax,idxmax] = max(nFP1);
vec2proc = nFP1-0.3*ampmax; %提取出来的峰值至少为0.3倍的最大幅值，确保基频选择较为准确
vec2proc(vec2proc<0)=0; %将幅频曲线中幅度较小的剔除，避免造成对峰值提取与分析的干扰
```

将较矮的峰滤掉，可以让峰值的识别更为准确快速。

然后，我将频率小于最大值对应频率的峰找出来，用最大值对应频率比上找出来的峰值的频率。若接近整除，则该峰有可能就对应基频。我将候选的峰的数组序号归纳到`candidates`向量中。如果`candidates`为空，或`candidates`中频率最低的峰离最大值峰过近(频率比值接近1)，则认为最大值峰就是基频。如果`candidates`非空，且频率最低的峰离最大值峰足够远，则认为该峰就是基频峰。上述判断由如下代码实现

```
[~,locs] = findpeaks(vec2proc);
locstmp = locs(locs<idxmax);
if isempty(locstmp)
    basefreq = nf(idxmax);
    harmo = 1;
else
    guesstmp=nf(idxmax)./nf(locstmp);
    guesstmp=abs(guesstmp-round(guesstmp));
    candidates=locstmp(guesstmp<0.02);
    if isempty(candidates)
        basefreq = nf(idxmax);
```

```
        harmo = 1;
    else
        basefreq = nf(candidates(1));
        if abs(basefreq/nf(idxmax)-1)<0.015 %如果原始片段点数较少，则频域分辨率较差，可能把最大值附近的点当作新的基频，故额外加以判断
            basefreq = nf(idxmax); %若新的基频和最大幅度对应的频率差别过小，则直接将后者作为基频
            harmo = 1;
        else
            harmo = nFP1(candidates(1))/ampmax;
        end
    end
end
[~,loc] = min(abs(freqs-basefreq));
basefreq = freqs(loc);
num = loc;
```

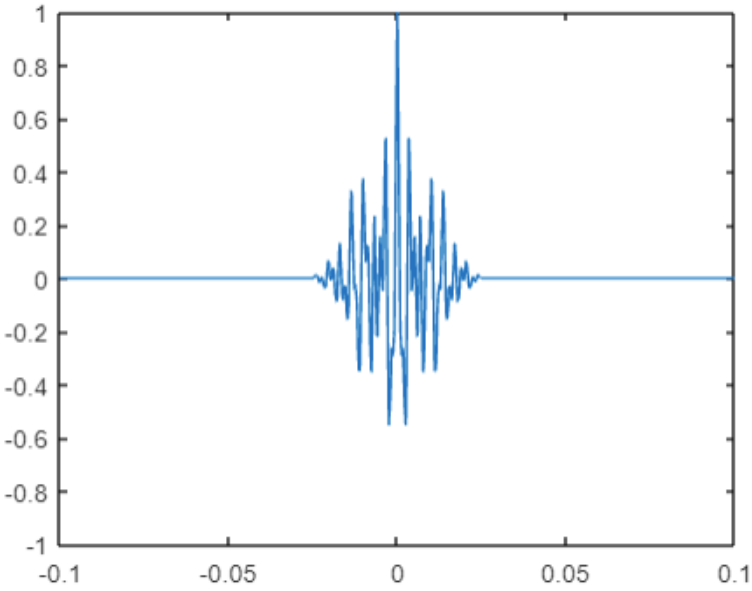
找到基频后就可以寻找各次谐波分量了

```
for x=2:maxhmnc
    thresh = floor(0.015*nL*x*basefreq/nFs); %允许在误差不超过+-1.5%的范围内搜索峰值
    freqtmp = x*basefreq;
    if freqtmp>=nFs/2
        harmo=[harmo,zeros(1,maxhmnc-x+1)];
        break
    end
    [~,locharm] = min(abs(nf-freqtmp)); %寻找最接近谐波频率的现有频率的位置locharm i.e.location of harmonic
    ampharm = max(nFP1(max(locharm-thresh,1):min(locharm+thresh,end))); %该谐波分量的幅度
    harmo = [harmo,ampharm/ampmax];
end
```

其实我本来还尝试过对时域谱用自相关来找基频。我尝试编写了如下实验性的代码

```
clear
clc
fs = 8000;
load("testwave294.mat"); %截取的一段基频为294Hz的音频时域谱
[autocor,lags] = xcorr(ndat(1:200),fs,'coeff');
plot(lags/fs,autocor)
axis([-0.1 0.1 -1 1])
[pklg,lclg] = findpeaks(autocor,'MinPeakheight',0.2*max(autocor));
f=fs/mean(diff(lclg))
```

其中0.2是一个阈值，我将幅度较小的峰值滤除，仅寻找幅度较大的峰值的空间排布。如信号与系统中所学，自相关在频域上是对频谱取模方，体现在时域上就是让信号的时间周期性更加明显。但这种方式只适用于基频幅度最大时。比如上述代码，在阈值为0.2时对这个特定波形的识别结果较为准确




```
f = 293.5780
```

但当阈值取0.3时就会将基频识别为220Hz。

经过进一步测试，对其他音调，这个阈值并不固定，所以这种方法并没有普适性。

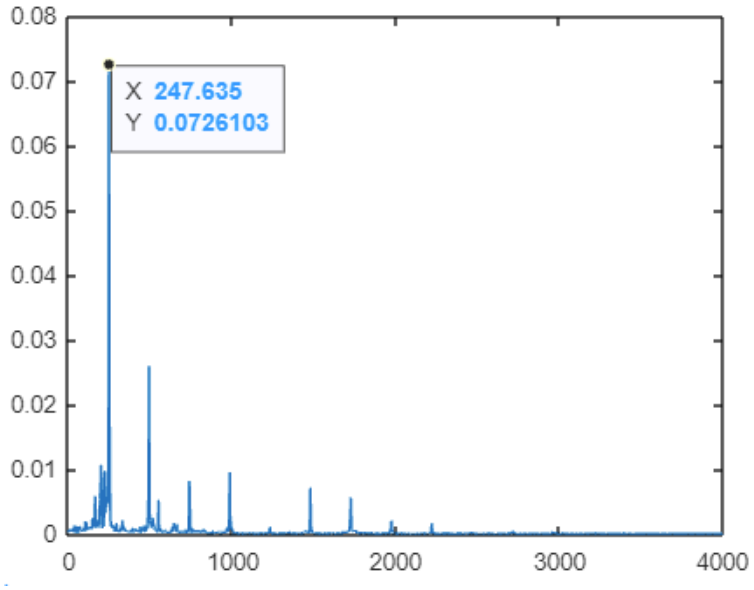
我又想到，因为**频谱中各次谐波分量的间隔频率恰好是一个基频。**如果能将**频谱中幅度沿频率轴的周期性**提取出来，也能找到基频。而且这个方法的实现逻辑比先找最大值再找基频的方法更加简明清晰。于是我又做了如下实验(test2.mlx)，**对频谱(而不是先前的时域谱)做自相关分析**

```
clear
clc

fs = 8000;
load("329.63Hz.mat"); %可以load其他测试文件
N = length(nFP1);
[autocor,lags] = xcorr(nFP1,fs,'coeff');
plot(lags/fs,autocor)
axis([-0.1 0.1 -1 1])
[pklg,lclg] = findpeaks(autocor,'MinPeakheight',0.3*max(autocor)); %阈值为0.3
f=mean(diff(lclg))*fs/2/N; %间隔数*频域分辨率
```

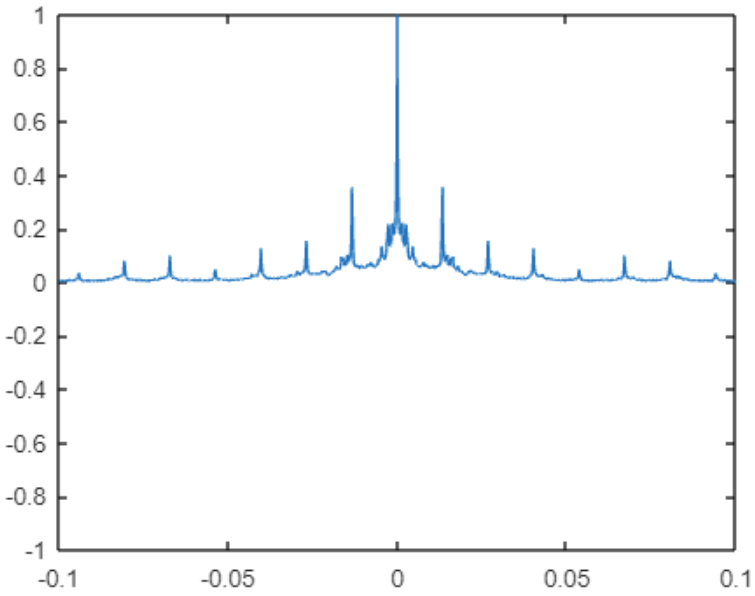
实验结果如下

- 基频246.94Hz

  傅里叶分析结果



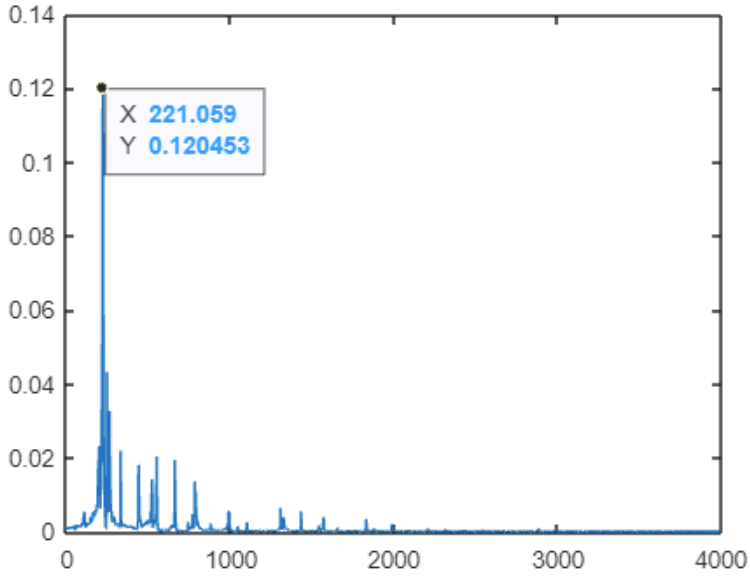
自相关分析结果



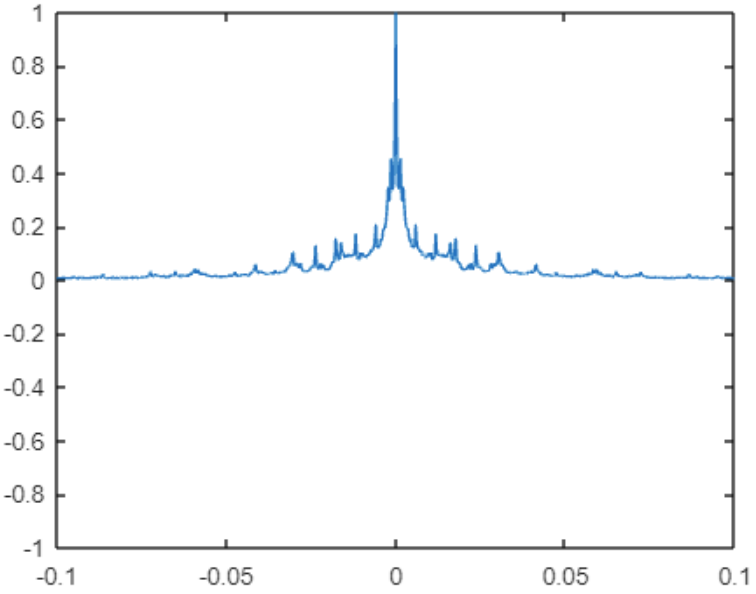

```
f = 245.2722
```

- 基频110Hz(存疑，可能是时域截取时有些问题)

  傅里叶分析结果



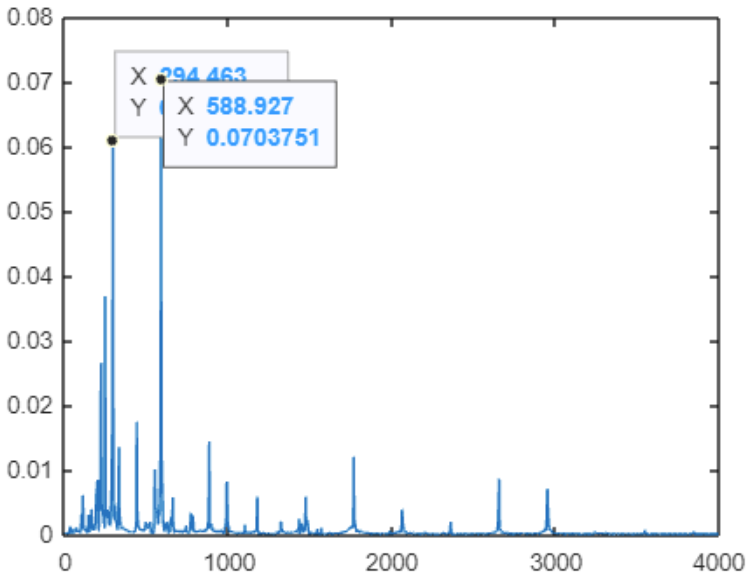
自相关分析结果



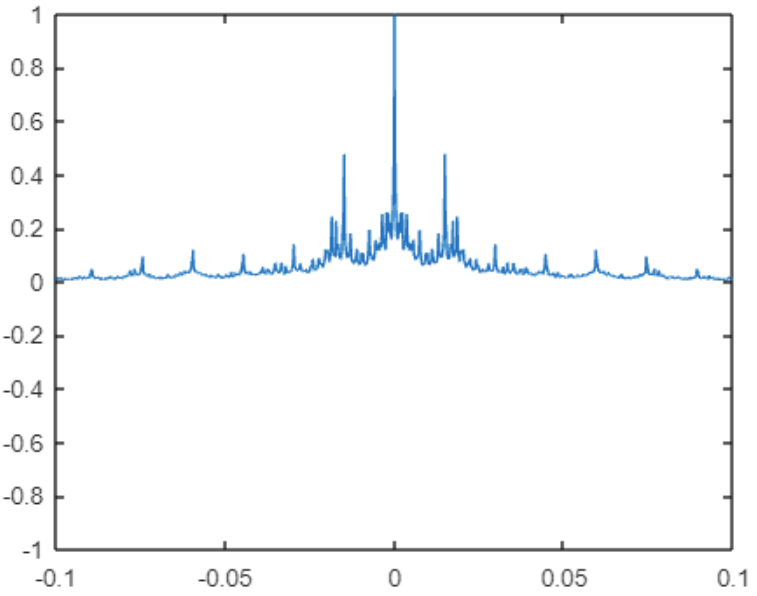

```
f = 20.9302
```

- 基频294Hz

  傅里叶分析结果



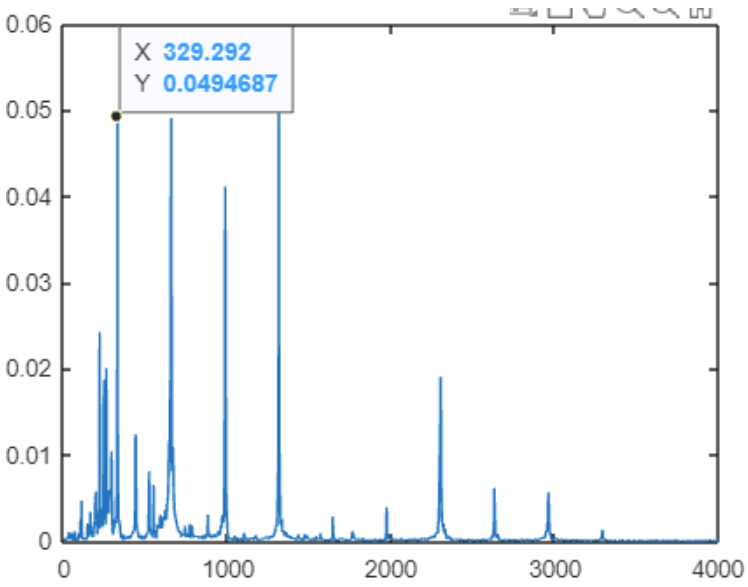
  自相关分析结果



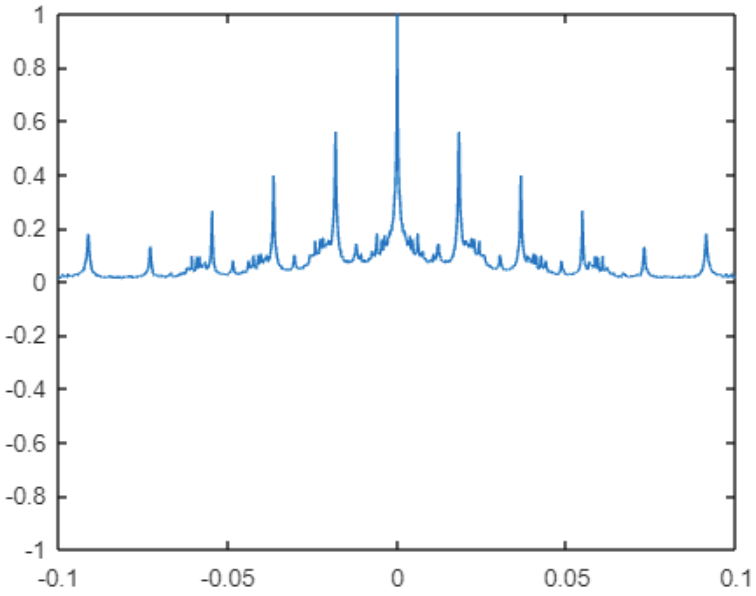

```
f = 294.3723
```

- 基频329.63Hz

  傅里叶分析结果



  自相关分析结果




```
f = 329.1995
```

可以看出，对频域做自相关的效果远好于对时域做自相关。当频谱比较干净，分辨率较高，谐波分量的幅度也较大，没有被噪声盖过时，这种方法能给出比较稳定且准确的结果。但对于110Hz的那个频谱，由于低频噪声较多，干扰峰过多，最后的识别效果偏差很大。另外，0.3这个阈值也是受到matlab官方文档的启发取得的，我还尝试了其他值，发现阈值的选取对结果有一定影响，而且我不敢保证这个阈值对大多数频谱适用。最后，为了鲁棒性，我仍然采用了最初的用最大值频率找基频的方法完成了本次作业，但我仍然相信这种方法值得进一步探索和改进。

最后，回到主线任务，我在task9中调用了刚刚的函数，并将最后的结果播放出来

```
maxhorder = 7;%取到的最高次谐波
rslt = [];
for x = 1:(length(beatstart)-1)

[loc,mag,basefreq,harmontmp]=toneanalyse(music(beatstart(x)+210:min(beatstart(x+1)-100,beatstart
(x)+10000)),Fs,freqs,maxhorder);
    timelen=(beatstart(x+1)-beatstart(x))/Fs;
    rslt = [rslt;timelen,loc,mag,basefreq,harmontmp];
end
[loc,mag,basefreq,harmontmp]=toneanalyse(music(beatstart(end)+210:min(end,beatstart(end)+10000))
,Fs,freqs,maxhorder);
timelen=(length(music)-beatstart(end))/Fs;
rslt = [rslt;timelen,loc,mag,basefreq,harmontmp];
musicdat = [rslt(:,1:3),rslt(:,5:end)];
music2play = musicgen(freqs,musicdat,Fs,0.12,1.7);
sound(music2play,Fs);
```

虽然部分音可能没有识别准确，但总体效果比较接近原来的音乐(可以听一听ReFmt.wav，就是把music2play导出的结果)，这也印证了我的方法是可行的，有效的。

Task10

对7中时域信号使用`toneanalyse`模块

```
clear
clc

freqs = readmatrix("freqs.xlsx")';%读取频率
load("Guitar.MAT");
srate = 8000; %采样率
xfadet = 0.120; %渐变用时
[num, mag, basefreq, harmon] = toneanalyse(wave2proc,srate,freqs,10);
```

得到结果如下

$$\begin{bmatrix} 0.686269973529535 & 1 & 0.657953552315022 & 0.754827953712237 & 0.0359097612727038 \\ 0.0753900051787460 & 0.246310577256699 & 0.0850768035951646 & 0.0927240632499852 & 0.0441313430428978 \end{bmatrix}$$

第一行为1~5次谐波，第二行为6~10次谐波

接下来用这个结果播放东方红

```
%music
musicdat = [0.5 0.25 0.25 0.5 0.5 0.5 0.25 0.25 0.5 0.5; 20 20 22 15 0 13 13 10 15 0; 0.7 0.7
0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.7]';%音乐数据矩阵[持续时间,音符,幅度]
harmonrepeat = repmat(harmon,length(musicdat(:,1)),1);
musicdat = [musicdat,harmonrepeat];
music = musicgen(freqs,musicdat,srate,xfadet,0.75); %最后一个参数调为正值(0~2)可使每个音的延音等长，否则
迭接固定为xfade长度

%sound
sound(music,8000);
plot(music);
```

仍然不是很像，因为真实的吉他不同的音可能有着不同的泛音，不能一概而论。

Task11

在Task9中，我已经得到了这把吉他多个音调的泛音信息，我将这个结果存为rslt.mat文件

在这个文件中，同一个音调可能出现了多次，也有音调没有出现过。我对出现的音调进行求平均，将结果作为该音的泛音。对没有出现过的音，我用离它最近的已有音调的谐波分布进行近似替代。

```
load("rslt.mat");%读取泛音数据
freqs = readmatrix("freqs.xlsx")';%读取频率
presentnotes = unique(rslt(:,2))';%已有记录的音
harmonicsmat = zeros(length(freqs),7);
cntvec = zeros(1,length(freqs));%记录每个音出现过几次
for x=1:length(rslt(:,2))
    harmonicsmat(rslt(x,2),:)=harmonicsmat(rslt(x,2),:)+rslt(x,5:end);
    cntvec(rslt(x,2))=cntvec(rslt(x,2))+1;
end
for x=1:length(freqs)
    if cntvec(x) ~=0
        harmonicsmat(x,:)=harmonicsmat(x,:)/cntvec(x); %求平均
    end
end
temp = setdiff([1:length(freqs)],presentnotes); %没出现过的音
for x = temp
    [~,loc]=min(abs(presentnotes-x)); %就近近似
    harmonicsmat(x,:)=harmonicsmat(presentnotes(loc),:);
end
%save("harmonicsmat.MAT","harmonicsmat",'-mat');
```

最后，我用`harmonicsmat`记录的泛音数据演奏了东方红，曲谱从提前写好的`dongfanghong.xlsx`中加载

```
datdfh = readmatrix("dongfanghong.xlsx");
datdfh = [datdfh,zeros(length(datdfh(:,1)),7)];
for x=1:length(datdfh(:,1))
    if datdfh(x,2)~=0
        datdfh(x,4:end) = harmonicsmat(datdfh(x,2),:);
    end
end
musicdfh = musicgen(freqs,datdfh,8000,1.2,1.5);
sound(musicdfh,8000);
```

结果已经保存在`DFH.wav`中，但用电脑播放器播放时会先过一个低通，使得听起来比MATLAB中的效果更沉闷。建议在MATLAB中体验真实效果。效果我自己觉得介于吉他和钢琴之间，我认为这是因为我使用的包络和真是吉他的包络仍然有差异，我甚至觉得有时包络对听觉感受的影响大于泛音。不过总之，最后的合成音乐基本达到了我的预期。听着由自己一手从0写出来的程序合成的音乐，还是相当有成就感的。

总结

通过这次实践，我从完全不会MATLAB到能用它比较自如地实现自己的一些想法，还是挺有成就感的。从编程的角度，我真切地体会到了这个编程语言的强大和灵活，并且发自内心地希望用它来进行日后的一些科研和自主探索。不过归根结底，这门课还是一门信号与系统的实践课。在完成这个大作业的过程中，我更加熟悉了DFT、相关、时域和频域的关系、采样定理等知识，对用计算机实现信号处理和分析的流程有了新的认识。总之，虽然写作业的过程经历了很多曲折，方案也根据实际效果和需求进行了多次推翻重来，但是我感觉自己学到了很多，并且乐在其中。