

音乐合成大作业

无 55 黄超 2015011098

(为简洁起见, 原作业要求加以总结概括后作为本报告中的问题描述; 由于之后小节的代码在先前的基础上加以修改和完善, 故代码展示部分均显示最终代码, 仅在输入参数上有所不同, 实现向下兼容, 具体输入会在描述中说明)

一、 实验目的

- 1、 熟悉运用 Matlab 基本指令, 对信号进行时域、频域分析
- 2、 通过电子音乐合成实验巩固加深对傅里叶变换的理解
- 3、 编写 GUI, 熟悉 Matlab 图形界面编程

二、 实验内容

1. 根据十二平均律计算《东方红》第一小节的曲调, 根据节拍拼出音乐并播放

为方便编程, 这里对唱名和节拍的表达方式进行定义: 在给定乐曲的音调 (函数接口中为 'C'、'D'、'E' 等字符) 后, 将基本音符 (不带高音点或低音点) 记为 1~7, 将带一个高音点的音符 (较基本音符高一个八度) 记为 11~17, 两个高音点的记为 21-27; 而将带一个低音点的音符记为 -1~-7, 将带两个低音点的音符记为 -11~-17。对于节拍的表达, 这里假设 BPM (Beat Per Minute) 为 120, 即一拍为 0.5 秒, 节拍表示为拍数的倍数, 如 1 表示一拍, 0.5 表示半拍, 以此类推。

定义了唱名和节拍的表达方式后, 便可以表达一段乐谱。附带资源中的 'dfh_notation.mat' 记录了《东方红》第一小节的曲谱, 采用

load 命令导入工作空间后，调用 compose.m 函数即可实现演奏。曲谱如下图所示。

	1	2	3	4	5	6	7	8
1	5	5	6	2	1	1	-6	2
2	1	0.5000	0.5000	2	1	0.5000	0.5000	2

下面对 compose.m 文件及其关联函数进行解释。

```
function [finish] = compose(harmo,notation,base,choice)
```

compose 函数接受四个参数，harmo 为元胞数组，用来记录谐波分量，在后面会用到，本题中只需要输入一个空元胞即可；notation 为曲谱数组，本题中即为从 dfh_notation.mat’ 中导入的 dfh 数组；base 为字符型变量，表示这段曲子的音调，这里取 ‘F’；choice 为可选参数，用以调整包络类型，在本题中不涉及。

```
main_freq = melody(base);
```

之后脚本调用 melody 函数计算基本音符对应的频率。输入参数 base 即定义的音调。如下所示，melody 函数先对 base 所代表的音调进行识别并计算，得到该音调下 do 的频率。

```
switch upper(my_tune)
case 'C'
    start = 110*2^(15/12);
case 'D'
    start = 110*2^(17/12);
case 'E'
    start = 110*2^(19/12);
case 'F'
    start = 110*2^(20/12);
case 'G'
    start = 110*2^(22/12);
case 'A'
    start = 440;
case 'B'
    start = 440*2^(1/12);
otherwise
    start = 0;
```

```

disp('Unknown tune.')
end

```

之后根据十二平均律，以及 7 个基本音符之间的倍数关系（仅在 mi、fa 之间间隔半音，其余间隔为全音），如下代码所示，可以计算出基本音符对应的频率，即返回的 freq 行向量。

```

stage = 1/12*[0,2,4,5,7,9,11];
freq = start*2.^stage;

```

由于 notation 数组仅最后一行表达节拍，之前各行表示音调（多行因为想要实现和弦功能，这在后文会提到，目前只有一行），如下代码所示，在抽取除最后一行外的所有行后，可以依照上文规定计算各音调频率，在此不再赘述。

```

number = notation(1:end-1,:);
tune = zeros(size(number));
for row = 1:size(number,1)
    for col = 1:size(number,2)
        temp = number(row,col);
        if temp <= -11 && temp >= -17
            temp = -temp - 10;
            tune(row,col) = main_freq(temp)/4;
        elseif temp >= 11 && temp <= 17
            temp = temp - 10;
            tune(row,col) = main_freq(temp)*2;
        elseif temp >= 1 && temp <= 7
            tune(row,col) = main_freq(temp);
        elseif temp <= -1 && temp >= -7
            temp = -temp;
            tune(row,col) = main_freq(temp)/2;
        elseif temp >= 21 && temp <= 27
            temp = temp - 20;
            tune(row,col) = main_freq(temp)*4;
        end
    end
end
end

```

之后对于每个节拍对应的每个音符，compose 函数会调用 make_song 函数生成一段长达 24000 个样点（3 秒）的、以该音符

频率为基音的正弦序列，compose 函数根据节拍对这一序列做时域的适当截取。由于本题不涉及包络，故采用无包络直接截取的方式，之后该语句将会被注释以利用包络。最后将每个音符依次拼接形成 finish 行向量，表示合成的乐曲。代码如下：

```
beat = notation(end,:);
finish = [];
for m = 1:size(tune,2)%number of beats,0.5s as one beat
    part = zeros(1,24000);%initial length 24000
    for i = 1:size(tune,1) %number of track
        mix = zeros(9,2);
        for h = 1:size(harmo,1)
            if tune(i,m) > 0.99*harmo{h,1} && tune(i,m) < 1.01*harmo{h,1}
                mix = harmo{i,2};
                break
            end
        end
        prod = make_song(mix,tune(i,m));
        if i == 2
            weight = 0.5;
        elseif i == 3
            weight = 0.3;
        elseif i ==4
            weight = 0.2;
        else
            weight = 1;
        end
        part = part + weight * prod;
    end
    % part = part(1:beat(m)*4000).*envelope_linear(beat(m)*4000,choice);
    part = part(1:beat(m)*4000);
    finish = [finish,part];
end
```

利用 sound 函数，设置采样率为 8000，播放所得到的 finish 向量可发现音调很单薄，并且在各音调迭接处有明显的“啪”的杂音，原因为本题中未考虑包络和谐波分量。

2. 采用适当的包络修正乐音，消除相邻乐音间的“啪”的杂声

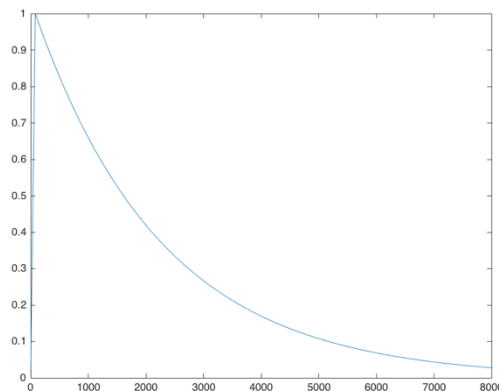
```
function [cover] = envelope_linear(time,choice)
```

本题调用 envelope_linear.m 函数计算各音符的包络，如上所示，该函数接受两个参数，time 表示待计算的样点数，choice 为可选参数，默认为 1（表示采用吉他的包络，0 为管乐器的包络，在后文会涉及）。本题中模拟吉他的包络，即冲激部分采用线性函数近似，衰减部分为指数衰减函数。代码如下所示。

```
up = 80/time;  
t(1:round(up*time)) = 1/(time*up)*s(1:round(up*time));  
t(round(up*time):end) = exp(-(s(round(up*time):end)-time*up)*2*log(6)/(1-  
up)/time);
```

其中 up 参数表示冲激上升部分占总时长的比例，值得注意的是，这一参数应当和序列长度有关，在先前的实验中，我采用的是一个绝对的数值，这可能导致对于长度不同的音冲激上升速度不同，听上去弹拨的力度会发生变化，像是不同乐器演奏的。log

(6) 为调整的衰减参数，规定了序列终点衰减的比例，衰减幅度不够将无法很好避免“啪”的杂音，衰减幅度过大将导致每个音过于短促。综合调整后，包络形状如下所示：



在 compose.m 文件中加入包络，即采用下面这条语句：

```
part = part(1:beat(m)*4000).*envelope_linear(beat(m)*4000,choice);
```

再次演奏《东方红》，可以发现“咻”的杂音得到了很好的消除,且在各音符切换处圆润了不少。

3. 分别将 2 中的音乐升高一个八度、降低一个八度、升高半个音阶

本题调用 tune_up.m 文件，其中用到的关键函数是 resample。resample 函数接收三个参数，第一个为待处理信号，之后两个输入参数 P 和 Q 的比值 P/Q 若大于 1，则表示升采样，反之表示降采样。若为升采样，则在相同采样率的播放情况下，音调将会降低（样点数变多，原样点间插入 0）；反之音调会升高。

因此升高八度，升高半个音阶都应该使 P/Q 小于 1，P/Q 分别为 $\frac{1}{2}$ 和 $\frac{1}{2^{1/2}}$ 。值得注意的是，这里的 P，Q 都应该是整数，因此后者应当放大取整后输入，精度取决于放大倍数（这里取 10000 的放大倍数）。而降低八度则只需把 P/Q 设置为 2。代码如下，输入的参数 song 即为上一题产生的乐曲片段。

```
function [song1,song2,song3] = tune_up(song)

song1 = resample(song,1,2);
song2 = resample(song,2,1);
half = round(1e4*2^(1/12));
song3 = resample(song,1e4,half);
```

4. 在 2 的音乐中加入谐波分量，体验效果

如 1 中所述，控制谐波的数组为 harmo 元胞数组。这里对其结构先做简单介绍。harmo 元胞数组有两列，第一列为 110~1710 四个八度的准确频率，第二列为 9*2 的矩阵。该矩阵第一列为 2~10，记录了该频率谐波的倍数，第二列记录各次谐波幅度和基波幅度的比值。在本题中，

设计所有频率的基波对应的各次谐波组成和幅度都相同，设置为二次谐波 0.2，三次谐波 0.3,即谐波矩阵均为下图所示：

	1	2
1	2	0.2000
2	3	0.3000
3	4	0
4	5	0
5	6	0
6	7	0
7	8	0
8	9	0
9	10	0

这里调用 specific_harmo.m 文件计算 harmo 元胞数组，输入为上述谐波矩阵。具体内部先生成各基频作为第一列，之后将第二列各矩阵均赋为输入的谐波矩阵。代码如下所示：

```
function [harmo] = specific_harmo(mix)

low_bound = 110;
table = 2.^([0:1:48]/12);
base_tune = low_bound * table;
harmo = cell(length(base_tune),2);

for i = 1 :length(harmo)
    harmo{i,1} = base_tune(i);
    harmo{i,2} = mix;
end
```

之后将生成的 harmo 元胞数组作为输入送入 compose 函数，其他参数不变，即可得到加入谐波的《东方红》。听后感觉声音饱满了不少，不再像只有基音时那么空灵，但还是显得有些“闷”，可能原因是缺少高次谐波分量。

5. 自选音乐合成片段

这里我选择的是徐梦圆的《China-X》作为合成曲目（曲谱见附带资源）。选择该曲本身就是电子音乐，因此预期合成效果能和原曲较接近。此处仍采用 compose 函数，输入 notation 矩阵为附带资源

中 'chinax_notation.mat' 的 chinax 矩阵, 谐波元胞数组仍采用上一题得到的 harmo 数组, 音调选 'E' 调。合成后发现和原曲仍有较大差距, 除开和弦、混音等专业操作, 可能的原因还是在谐波分量不够真实。

6. 播放 fmt.wav 已有资源, 体验效果

该曲为真实吉他弹奏, 声音饱满动听, 且带有和弦、琶音等专业操作, 效果比单音合成版本好很多。

7. 对 realwave 进行预处理得到与 wave2proc 类似的波形

预处理调用 preprocess.m, 其目的在于去除非线性谐波和噪声, 且提示为从时域进行操作。考虑到二者均为随机信号, 且绘出 realwave 波形可发现其明显具有周期, 因此可以利用将各周期取出之后相加取平均的方法, 消去随机信号的影响。考虑到 realwave 有十个周期, 但总样本点数为 243, 周期数不为整数。因此需要先利用 resample 函数将周期变为整数后进行求和取平均操作, 之后利用 repmat 函数将一个周期复制十遍, 之后再次利用 resample 函数将信号长度恢复到初始长度。代码如下, 其中 reshape 函数将行向量变为一个矩阵, 便于进行按行相加运算。

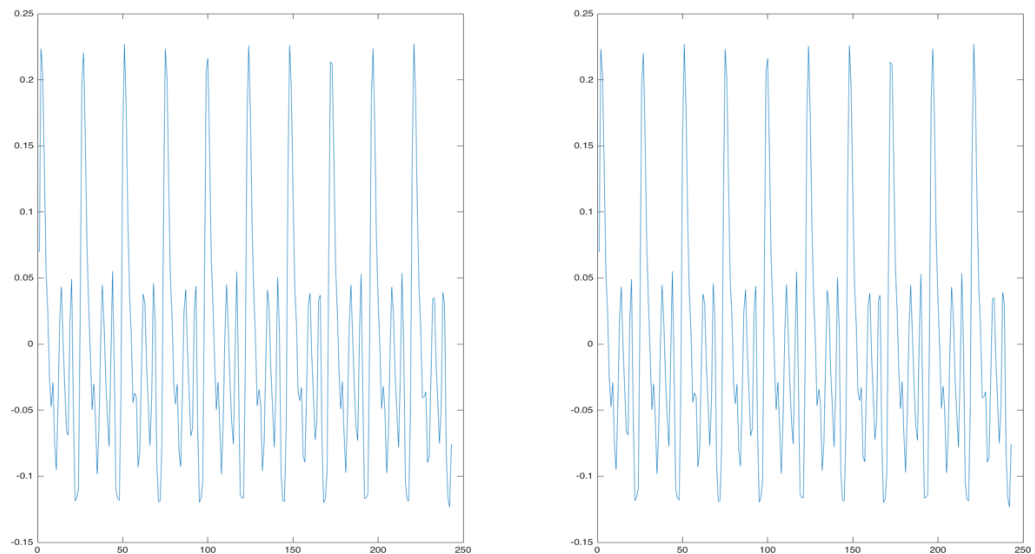
```
function [new_wave] = preprocess()
load('Guitar.mat')
expand = resample(realwave,10,1);
cut = reshape(expand,243,10)';
average = sum(cut,1)/10;
repeat = repmat(average,1,10);
new_wave = resample(repeat,1,10)';
subplot(1,2,1)
plot(new_wave)
subplot(1,2,2)
```



```
plot(wave2proc)
error = sum(abs(wave2proc-new_wave))
```

画出处理后的波形（下左图），和提供的 wave2proc（下右图）

对比可发现，二者几乎一样，各样点之差取绝对值后相加仅为 0.02299，可知这种方法是可行的。但由于处理时其实已知道信号的周期，因此在实际处理中用途有限。

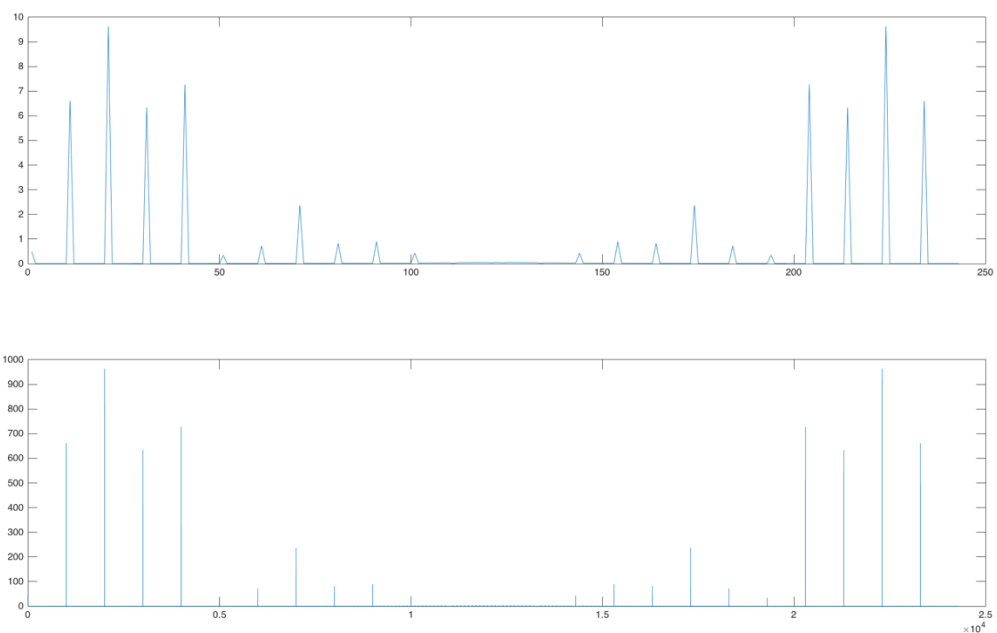


8. 分析 7 中 wave2proc 的基频，并用傅里叶级数或变换的方法分析谐波分量

本题调用 f_analysis.m 来进行傅里叶分析。代码中利用 Matlab 自带的快速傅里叶变换算法 fft()，对音频进行处理。代码绘出仅用一段 wave2proc 进行分析（下上图）和将其复制一百遍进行分析（下下图）的频谱图。这里对傅里叶变化结果取模值以查看幅频特性，同时，根据抽样定理，有效的频域区间应当为总区间长度的一半（图上频率尚未转化成真实频率）。对比可知，重复之后频域分辨率变高，波形更像是冲激函数而非 Sa 函数。原因是重复若干次后会增强信号的周期

性，同时采样点数的增加也会增加傅里叶变换的精度。为求出峰值所在位置和峰值大小，这里采用 Matlab 自带的 findpeaks 函数，并设置阈值为最高值的 1/100。为求基频，运用公式 $f_{\text{real}} = \frac{f_{\text{show}}}{\text{length}} * F_s$ ，其中 length 代表样本点数，Fs 为采样频率。之后将频率、幅度相对于基频幅度做归一化操作后，将幅度、频率两行合并存为矩阵。代码及结果如下所示：

```
function [base,result] = f_analysis()
load('Guitar.mat')
f0 = abs(fft((wave2proc)));
f = abs(fft(( repmat(wave2proc',1,100))));
subplot(2,1,1)
plot(f0)
subplot(2,1,2)
plot(f)
[amp,loc] =
findpeaks(f(1:floor(length(f)/2)+1),'MinPeakHeight',1/100*max(f));
loc = loc/length(f)*8000;
base = loc(1);
loc = round(loc/loc(1));
amp = amp/amp(1);
result = [loc;amp];
```



base =
3.295473251028807e+02

1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	1.4572	0.9587	1.0999	0.0523	0.1099	0.3589	0.1240	0.1351	0.0643

由输出可知，基频是 329.55Hz，是 E4 调。

9. 编程实现自动分析 fmt.wav 中每个音符的音调和节拍

此问较为复杂，大体而言可以分为三个步骤。最终生成的输出数据在附带资源的 ‘song_info.mat’ 中。也可在下文 GUI 第二工作区部分查看最终效果。如下图所示，共找到 29 个音。

patch			patch		
29x3 cell			29x3 cell		
1	2	3	1	2	3
1 'A3'	E3 C4 bD5	3	16 'A3'	'bD5'	0.7500
2 'B3'	[]	1	17 'G4'	E5 C3	0.5000
3 'A3'	'B3'	1	18 'F4'	C3 bA3	0.5000
4 'D4'	4x3 char	1	19 'E4'	'C3'	0.5000
5 'E4'	B3 A3	1	20 'D4'	E5 E4 G4	0.5000
6 'G3'	'E5'	1	21 'E5'	C4 E4	1
7 'A3'	'F3'	1	22 'B3'	'E4'	0.7500
8 'F3'	D4 B3	1	23 'D4'	B3 D3 A3	1
9 'D4'	F3 G3	2	24 'C4'	D3 A3	1
10 'B3'	E5 E3 bA3	2	25 'F3'	'B3'	1
11 'E4'	'E3'	1.7500	26 'A3'	B3 F3	1
12 'A3'	F3 G3 bB3	0.2500	27 'B3'	A3 E3	1
13 'A3'	4x3 char	1	28 'A3'	B3 E3	1.2500
14 'E4'	A3 G3	1	29 'bA3'	E3 B3 A4	2.7500
15 'A4'	A3 E4	1			

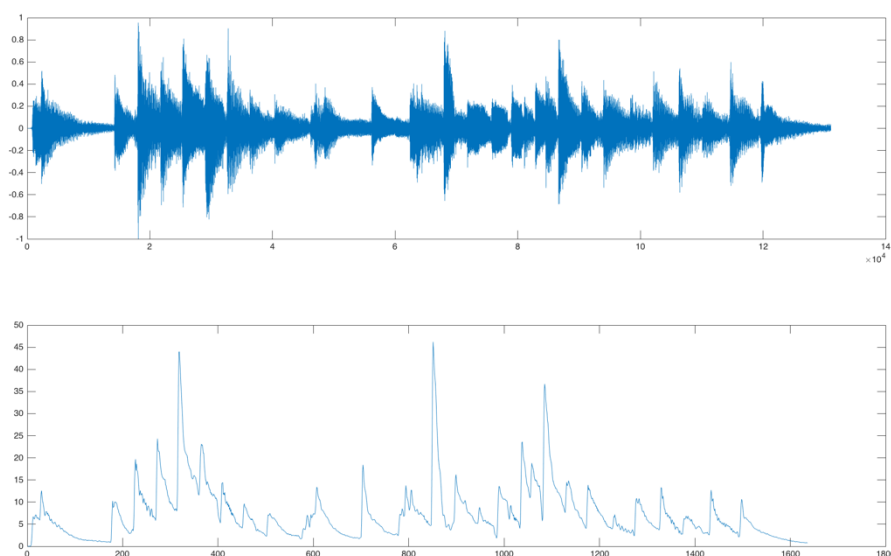
1) 判断每个音符的起始位置，计算节拍

这里调用 tune_analysis.m 文件，函数接口如下：

```
function [tune_freq,mag_freq,time] = tune_analysis(song,fs)
```

调用前先利用 wavread 函数读入 ‘guitar.wav’ 的数据，之后 将 data 和 fs 作为该函数的输入即可。前两个输出是元胞数组，二者每行分别记录了每个片段的频率信息和幅度信息。

为了分析每个音符的起始位置，我上网查找了一些资料，发现这一问题和判断语音端点的问题非常类似。语音端点的检测最简单的方法是利用短时平均能量和短时过零率进行判断，但考虑到音乐不像语音，在各音符之间没有太多的空白，因此短时过零率判断并不准确。相比之下，利用短时平均能量已经可以很好的区分各音符的起始位置。如下图所示，由于每次弹奏乐器都会使能量在短时出现一个峰值，因此只需要找到峰值对应的时间点即可标定音符出现的位置。



在计算短时平均能量时，利用了网上提供的计算短时平均能量算法，分帧进行运算，如下所示：

```
song = song / max(abs(song));  
FrameLen = 240;  
inc = 80;  
amp = sum((abs(enframe(filter([1 -0.9375], 1, song), FrameLen, inc))), 2);
```

在计算前先进行高通滤波目的在于更好地提取出高频部分的能量。在得到短时平均能量之后，选取峰值时，未避免毛

刺，在 findpeaks 函数中设置 MinPeakProminence 参数为 3，可以很好地选择起伏较大的峰，从而去除毛刺。同时设置 MinPeakDistance 参数为 $0.1 \cdot f_s / \text{inc}$ ，其中 f_s 为采样率，inc 为每个窗之间的移动距离，这样确保两个峰之间距离在时间尺度上不会小于 0.1s。这种选择是因为在真实乐器演奏中，和弦和两个独立的音的区分标准是发生的时间差，而琶音介于二者之间。在本实验中，统一将琶音视为和弦。而广义的和弦不一定要求同时产生（考虑实际弹奏情况也是如此），只要两个音的时间差小于某一个阈值，即可认为是和弦。假设一拍为 0.5s，那么这里认为的最小单位是 0.125s，即 1/4 拍。小于这个长度的音将被认为是和弦。以下为划分音符部分的代码。

```
[en,plc] =  
findpeaks(amp, 'MinPeakProminence', 3, 'MinPeakDistance', 0.1*fs/inc);%0.1  
s is the lowest bound  
plc = plc * inc / fs; % get the start time of each tune (unit: second)  
forward = [plc(2:end);length(song)/fs];  
time = forward - plc;
```

通过前后两个音符做差可以求出每段的持续时间，考虑到标定的峰值发生点可能在真实乐音触发之后，因此下面进行傅里叶变换的时间取为持续时间的 0.8 倍。

之后对每一个片段进行傅里叶变换，同样利用 findpeaks 函数选取适当的峰。考虑到在某一个音的频点周围可能出现不止一个峰，因此还应当设置各峰之间的最小距离，考虑到由十二平均律计算出的各乐音频率之间差距最小在几到几十赫兹的量级，因此将各峰之间最小距离设置为 $20 \cdot \text{样本点长度} / f_s$ ，

对应于 20Hz。之后将频率按照前文所述公式换算成真实频率，

记录各频率的幅度。代码如下所示：

```
tune_freq = cell(length(plc),1);
mag_freq = cell(length(plc),1);
for i = 1:length(time)
    f =
abs(fft(( repmat(song(floor(fs*plc(i)):floor(fs*(plc(i)+time(i)*0.8)))'
,1,100)))); % adopting fft transform, 0.8 for avoiding interruption
    [amp,loc] =
findpeaks(f(1:floor(length(f)/2)+1),'MinPeakHeight',1/5*max(f),'MinPea
kDistance',20*length(f)/fs);
    loc = loc/length(f) * fs;
    tune_freq{i} = loc;
    mag_freq{i} = amp;
end
```

2) 分析每个音符中所含音调，计算和弦和谐波分量

这里调用 fetch_base.m 文件进行操作。该函数接口如下

所示：

```
function [base,harmo] = fetch_base(tune_freq,mag_freq)
```

输入即从上一部分得到的 tune_freq 和 mag_freq 两个元胞数组后，输出为 base 和 harmo 两个元胞数组，每一行对应一段音乐片段，base 中记录主旋律和和弦的频率，harmo 的每一列记录每个音调分析出的谐波分量。

一开始对于谐波和基音的分析我采取的是将幅度最大的频率作为基音，在高频段查找其谐波分量，经过一轮查找后即将被访问的频点幅度设为 0，之后再查找次大幅度频点，以此类推。但后来结合 8 题中傅里叶变换的结果我发现某些音的谐波分量幅度可能大于基音幅度。特别地，我在寝室中请会乐器的同学弹奏吉他并录音，在进行傅里叶变化后的却也发现了上

述现象, 因此这种方法是不合理的。之后我考虑从最低频开始, 向高频段查找谐波分量, 这样的确能保证找到尽可能多的谐波, 但由于和弦多是低频的乐音, 由于频率无法做到精确的倍数关系, 可能造成谐波的误判, 或是将幅度较大的主旋律乐音视作谐波, 同样也是不合理的。最终我想到可以把上述两个方法结合起来, 首先还是查找当前幅度最大的频点, 之后向低频段查找这一频率是否是某一个低频基音的倍频, 一般而言, 根据我实验计算的包括吉他、手风琴在内的乐器, 谐波幅度大于基音幅度的谐波多为二次和三次谐波, 因此这里考虑最大幅度频点是否是某一低频的二倍频或三倍频, 并且倘若该低频幅度大于最大幅度频点幅度的一半, 那么将低频段的频率作为基频, 否则该最大幅度频率就是基频。确定了基频之后, 只要在高于基频的频段查找谐波分量即可。之后同样将访问过的频点幅度设为 0, 从而在当前最大幅度频点进行查找, 以此类推, 直到遍历所有频点。这一方法的优点在于平衡了谐波和和弦的比例, 缺陷在于可能无法区分相差八度的两个音的和弦(这在实际情况中的确可能发生), 同时若某一频率是两个音的不同倍数的谐波, 但其只能使用一次, 会造成谐波分量的丢失。确定倍频部分同样需要设计一个阈值, 已消除由于频率漂移带来的影响。

此部分代码如下:

```
for i = 1:length(tune_freq)
    tune = tune_freq{i};
    mag = mag_freq{i};
    base{i} = [];
```

```

count = 0; %count the number of mixed tunes

while length(mag(mag == 0)) ~= length(mag)
    [max_v,max_i] = max(mag);
    peak = [tune(max_i),max_v];

    %now find the base if any exists below this freq (amplitude threshold
    is needed)
    for k = max_i:-1:1
        if tune(max_i) > tune(k) * down * 2 && tune(max_i) < tune(k) * up *
2 && mag(max_i)/mag(k) < 2
            peak(1) = tune(k);
            peak(2) = mag(k);
            mag(k) = 0;
            break
        end
        if tune(max_i) > tune(k) * down * 3 && tune(max_i) < tune(k) * up *
3 && mag(max_i)/mag(k) < 2
            peak(1) = tune(k);
            peak(2) = mag(k);
            mag(k) = 0;
            break
        end
    end
    count = count + 1;

    %clear the magnitude of base
    if peak(1) == tune(max_i)
        mag(max_i) = 0;
    end

    %find higher harmonic waves
    for j = 1:length(tune)
        if mag(j) == 0
            continue
        end
        if tune(j) > down * 2 * peak(1) && tune(j) < up * 2 * peak(1)
            harmo{i,count} = [harmo{i,count};2,mag(j)/peak(2)];
            mag(j) = 0;
        elseif tune(j) > down * 3 * peak(1) && tune(j) < up * 3 * peak(1)
            harmo{i,count} = [harmo{i,count};3,mag(j)/peak(2)];
            mag(j) = 0;
        end
    end
end

```



```

elseif tune(j) > down * 4 * peak(1) && tune(j) < up * 4 * peak(1)
    harmo{i,count} = [harmo{i,count};4,mag(j)/peak(2)];
    mag(j) = 0;
elseif tune(j) > down * 5 * peak(1) && tune(j) < up * 5 * peak(1)
    harmo{i,count} = [harmo{i,count};5,mag(j)/peak(2)];
    mag(j) = 0;
elseif tune(j) > down * 6 * peak(1) && tune(j) < up * 6 * peak(1)
    harmo{i,count} = [harmo{i,count};6,mag(j)/peak(2)];
    mag(j) = 0;
elseif tune(j) > down * 7 * peak(1) && tune(j) < up * 7 * peak(1)
    harmo{i,count} = [harmo{i,count};7,mag(j)/peak(2)];
    mag(j) = 0;
elseif tune(j) > down * 8 * peak(1) && tune(j) < up * 8 * peak(1)
    harmo{i,count} = [harmo{i,count};8,mag(j)/peak(2)];
    mag(j) = 0;
elseif tune(j) > down * 9 * peak(1) && tune(j) < up * 9 * peak(1)
    harmo{i,count} = [harmo{i,count};9,mag(j)/peak(2)];
    mag(j) = 0;
elseif tune(j) > down * 10 * peak(1) && tune(j) < up * 10 * peak(1)
    harmo{i,count} = [harmo{i,count};10,mag(j)/peak(2)];
    mag(j) = 0;
end

end

%modify the tune
peak(1) = modify(peak(1));
base{i} = [base{i};peak];
end

end

```

这里还调用了 modify.m 文件，该文件作用在于将确定下来的基音频率转化为标准的十二平均律计算的频率，方便之后的运算。转化时同样需要设定阈值避免频率漂移的影响。。

3) 综合获得的谐波分量，总结这一乐器的相关特征

这里需要调用 create_harmo.m 文件，统计出现的各频率的谐波分量的幅度。接口如下：

```
function [tune_table,combine_harmo] = create_harmo(base,harmo)
```

利用上一步得到的 base 和 harmo 元胞数组可以构造出

tune_table 和 combine_harmo 两个元胞数组输出。

tune_table 即将 base 的每一行的第一个元素（对应幅度最大的基音，这里将其视为主旋律）作为第一列元素，将剩余元素（对应和弦）作为数组存入第二列。代码如下：

```
raw_base = raw_base(:,1);%the second col are amp, useless
tune_table{i,1} = raw_base(1);
tune_table{i,2} = raw_base(2:end);
```

构造 combine_harmo 的目的在于，原先每个音符的出现是以时间顺序的，现在要将出现的相同音符收集起来，并且若某一谐波出现了不止一次，应当取平均值存入。首先 combine_harmo 的第一列为 110~1760Hz 由十二平均律计算出的标准频率，第二列为对应 2~10 次谐波及其归一化幅度的矩阵。首先将时域的每个音归到频域的对应该位置，如下所示：

```
for m = 1:length(raw_base)
row = find(cellfun(@(x) ismember(raw_base(m),x),combine_harmo));%find the
correct tune(modified)
combine_harmo{row,2} = [combine_harmo{row,2};harmonic{i,m}];%add to below
end
```

这里用到了 cellfun 用以查找对应频率。

之后再在每个频率对应的谐波矩阵中进行求平均计算，即找到相同倍数的谐波的行相加取平均即可。代码如下：

```
for i = 1:size(combine_harmo,1)
new_mix = combine_harmo{i,2};
new_harmo = zeros(9,2);
new_harmo(:,1) = [2:1:10]';
if isempty(new_mix)
combine_harmo{i,2} = new_harmo;
continue
end
sortrows(new_mix,1);
for times = 2:10
```

```

sum = 0;
rows = find(new_mix(:,1) == times);
if ~isempty(rows)
    for h = 1:length(rows)
        sum = sum + new_mix(rows(h),2);
    end
    sum = sum / length(rows);
end
new_harmo(times-1,2) = sum;
end
combine_harmo{i,2} = new_harmo;
end

```

若为空矩阵，则存入谐波幅度均为 0 的 9*2 矩阵。

得到了 harmo 元胞数组之后, 由于掌握的演奏资料有限, 必然不可能所有频率的谐波矩阵存在非零值, 还需要考虑利用别的频点进行估计。因此需要调用 complement.m 文件进行修正。该函数接受之前得到的 harmo 元胞数组, 确保输出的每一频率对应的谐波矩阵都存在非零值。具体方法是, 先利用低频段补充高频段缺失的谐波信息 (由于采样率只有 8000, 必然不可能在高频段采集到很多的谐波, 这里假设高低八度的两个频率谐波组成类似), 之后对于仍然全零的谐波矩阵, 利用其最临近频率估计谐波矩阵。由此产生了最终需要的谐波元胞数组, 可以像 1 中所述, 直接用于 compose 函数产生带泛音的乐曲。代码如下:

```

function [new_harmo] = complement(raw)
new_harmo = raw;
for i = 26:size(new_harmo,1)
    mix1 = new_harmo{i-24,2};
    mix2 = new_harmo{i,2};
    if ~any(mix1(:,2)) && ~any(mix2(:,2))
        continue
    end
end

```

```

elseif ~any(mix2(:,2))
    new_harmo{i,2} = new_harmo{i-24,2};

elseif ~any(mix1(:,2))
    new_harmo{i-24,2} = new_harmo{i,2};
end
end

for i = 1:size(new_harmo,1)
    mix = new_harmo{i,2};
    if ~any(mix(:,2))
        count = 0;
        while i+count <= size(new_harmo,1) || i-count > 0
            if i+count <= size(new_harmo,1)
                mix2 = new_harmo{i+count,2};
                if any(mix2(:,2))
                    new_harmo{i,2} = mix2;
                    break;
                end
            end
            if i - count > 0
                mix2 = new_harmo{i-count,2};
                if any(mix2(:,2))
                    new_harmo{i,2} = mix2;
                    break;
                end
            end
            count = count + 1;
        end
    end
end
end
end

```

10. 利用 8 中的傅里叶级数再次合成《东方红》

由 8 收集到的 result 数组可以合成一个谐波矩阵(如下图所示), 进而利用 specific_harmo 函数构造所有频率具有相同谐波矩阵的 harmo 元胞数组, 送入 compose 函数即可产生乐曲。合成后较之于先前加谐波的乐曲(题 4 生成)更加饱满, 但由于所有音都是一样的谐波组成, 使得高低音段的区分不那么明显。

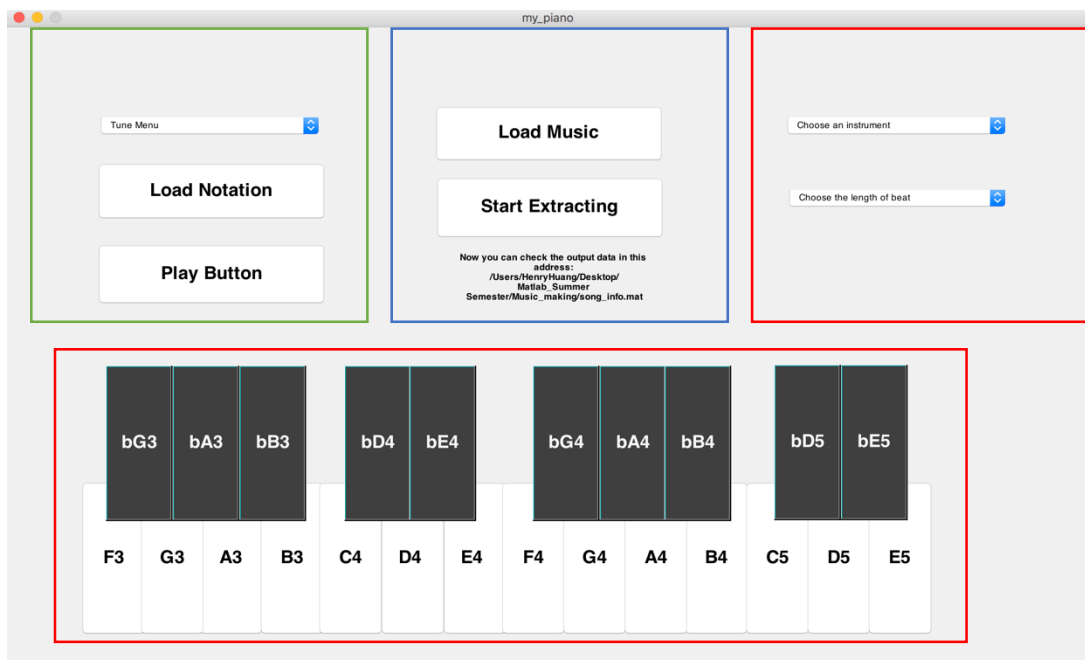
	1	2
1	2	1.4572
2	3	0.9587
3	4	1.0999
4	5	0.0523
5	6	0.1099
6	7	0.3589
7	8	0.1240
8	9	0.1351
9	10	0.0643

11. 利用 9 中获得的吉他特征（各音调的谐波分量）再次合成《东方红》

直接利用 9 中生成的最终版本的 harmo 矩阵即可。生成后各个音调音色确有不同，较之于无谐波乐曲更加饱满，且不同乐音区分度较好，有吉他弹拨弦的感觉（尤其是降调之后）。但和真实吉他曲相比，仍显得有些单薄，一方面是由于缺少和弦，另一方面也和谐波分量不够丰富有关。

12. 制作图形界面封装上述功能（my_piano.m&my_piano.fig）

设计的 GUI 界面如下所示：



主要功能区有三块，以绿、蓝、红三色表示，分别用以直接演奏给定乐谱的乐曲、提取曲谱和模拟键盘弹奏。为描述方便，以功能区

1、2、3 代指三者。总代码过长，此处只显示关键部分，完整代码详见源文件。

- 1) 功能区 1：由多选框，两个按钮组成。多选框用来选择曲调，未选取时默认为 C 调（简单起见，只支持 CDEFGAB 调），通过 get 函数获取选项，并用 switch 块进行选择。用以让用户从文件夹中选取想要弹奏的曲子，只支持以 1 中所述方式编排的 mat 文件作为曲谱(按照 1 中结构，实则表示音调的行可以不止一行，方便构造和弦，但这种数据结构限制了和弦和主旋律节拍一致，实际情况并非总是如此)。Play Button 按钮播放曲子。这里泛音采用的是 9 中提取的泛音元胞数组，已保存在附带资源 'harmonic.mat' 中，通过 load 自动读入。下面展示较为典型的 Load Notation 按钮的回调函数：

```
function NotationButton_Callback(hObject, eventdata, handles)
% hObject    handle to NotationButton (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)
[filename, pathname] = uigetfile('*.mat', 'Pick a Matlab Data
file');
temp = load([pathname,filename]);
fname = fieldnames(temp);
note = getfield(temp,char(fname));
var = get(handles.tune_choice,'Value');
switch var
    case 2
        T = 'C';
    case 3
        T = 'D';
    case 4
        T = 'E';
```

```

case 5
    T = 'F';
case 6
    T = 'G';
case 7
    T = 'A';
case 8
    T = 'B';
otherwise
    T = 'C';
end
global harmonic
global finish
global choice
finish = compose(harmonic,note,T,choice);

```

- 2) 功能区 2 这里重新构建一个顶层函数 song_analysis.m 用来封装提取曲谱的所有模块。同时，为方便阅读，设置 trans_tune 和 trans_time 两个函数，将频率和时间转化为音调名和节拍数（0.125s 为最小单位，一拍 0.5s）。由于不知道基调是什么，无法转化为唱名。代码如下：

```

function [patch,combine_harmo] = song_analysis(song,fs)

[tune_freq,mag_freq,time] = tune_analysis(song,fs);
[base,harmo] = fetch_base(tune_freq,mag_freq);
[tune_table,combine_harmo] = create_harmo(base,harmo);
combine_harmo = complement(combine_harmo);
patch = cell(length(time),3);
count = 1;
for i = 1: size(patch,1)
    if time(i) > 0.1
        patch{count,1} = trans_tune(tune_table{count,1});
        patch{count,2} = trans_tune(tune_table{count,2});
        patch{count,3} = trans_time(time(count));%0.5s is the
unit
        count = count + 1;
    end
end
end
end

```

```

function [tune_name] = trans_tune(freq)

low_bound = 110;

table = 2.^([0:1:48]/12);

base_tune = low_bound * table;

Name = [ ' A2'; 'bB2'; ' B2'; ' C3'; 'bD3'; ' D3'; 'bE3'; ' E3'; '
F3'; 'bG3'; ' G3'; 'bA3'; ...
        ' A3'; 'bB3'; ' B3'; ' C4'; 'bD4'; ' D4'; 'bE4'; ' E4'; '
F4'; 'bG4'; ' G4'; 'bA4'; ...
        ' A4'; 'bB4'; ' B4'; ' C5'; 'bD5'; ' D5'; 'bE5'; ' E5'; '
F5'; 'bG5'; ' G5'; 'bA5'; ...
        ' A5'; 'bB5'; ' B5'; ' C6'; 'bD6'; ' D6'; 'bE6'; ' E6'; '
F6'; 'bG6'; ' G6'; 'bA6'; ' A6'];

tune_name = [];

for i = 1:length(freq)

    idx = find(base_tune == freq(i));

    tune_name = [tune_name;Name(idx,:)];

end

end

function [new_time] = trans_time(part_time)

new_time = round((part_time*8))/4;

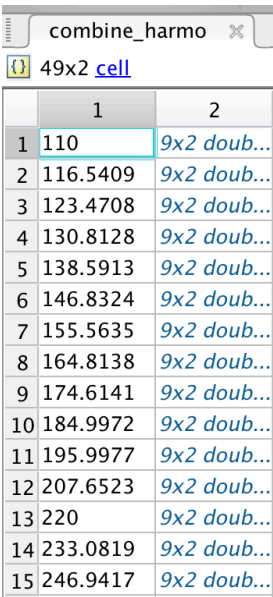
end

```

在 GUI 模块设计中,加入一个文本框显示当前运行状态,告知用户读写文件的地址,方便之后访问生成的数据文件,获得分析出的曲谱。按钮回调函数与上述代码类似,这里不再赘述,详见源代码。生成文件效果图如下所示(patch和 harmo 分别是音调节拍以及谐波元组)：

patch			
29x3 cell			
	1	2	3
1	'A3'	E3 C4 bD5	3
2	'B3'		1
3	'A3'	'B3'	1
4	'D4'	4x3 char	1
5	'E4'	B3 A3	1
6	'G3'	'E5'	1
7	'A3'	'F3'	1
8	'F3'	D4 B3	1
9	'D4'	F3 G3	2
10	'B3'	E5 E3 bA3	2
11	'E4'	'E3'	1.7500
12	'A3'	F3 G3 bB3	0.2500
13	'A3'	4x3 char	1
14	'E4'	A3 G3	1
15	'A4'	A3 E4	1

patch			
29x3 cell			
	1	2	3
16	'A3'	'bD5'	0.7500
17	'G4'	E5 C3	0.5000
18	'F4'	C3 bA3	0.5000
19	'E4'	'C3'	0.5000
20	'D4'	E5 E4 G4	0.5000
21	'E5'	C4 E4	1
22	'B3'	'E4'	0.7500
23	'D4'	B3 D3 A3	1
24	'C4'	D3 A3	1
25	'F3'	'B3'	1
26	'A3'	B3 F3	1
27	'B3'	A3 E3	1
28	'A3'	B3 E3	1.2500
29	'bA3'	E3 B3 A4	2.7500



	1	2
1	110	9x2 doub...
2	116.5409	9x2 doub...
3	123.4708	9x2 doub...
4	130.8128	9x2 doub...
5	138.5913	9x2 doub...
6	146.8324	9x2 doub...
7	155.5635	9x2 doub...
8	164.8138	9x2 doub...
9	174.6141	9x2 doub...
10	184.9972	9x2 doub...
11	195.9977	9x2 doub...
12	207.6523	9x2 doub...
13	220	9x2 doub...
14	233.0819	9x2 doub...
15	246.9417	9x2 doub...

3) 弹奏区可选用鼠标点击琴键或者用键盘控制。键位与琴键控制如下表所示：

键位	功能
Q,W	选择乐器（Q:手风琴(Melodica)， W:吉他(Guitar)）
1-4	控制节拍
5,6,7,9,0	bG4 向右五个黑键(注意：8 没有用)
S,D,F,H,J	bE4 向左五个黑键(注意：G 没有用)
Z,X,C,V,B,N,M	F3-E4 白键
R,T,Y,U,I,O,P	F4-E5 白键

设计目的在于方便弹出和弦，以及更快速控制节拍长短，由于 GUI 的触发机制是按键生效，所以不能判断按键持续时间，需要另加控制。值得一提的是如下面代码所示，可以实现用键盘调用按钮的回调函数：

```
case 'x'
G3_key_Callback(hObject,eventdata,handles)
```

另外由于弹奏方式和按曲谱演奏有所区别，故另外写了

‘key_press.m’ 文件用于弹奏，由 GUI 控制。使用鼠标弹琴后，需在 GUI 空白区域单击鼠标，方可继续使用键盘弹琴。主要代码如下：

```
function [song] = key_press(tune,time,harmonic,choice)
for h = 1:size(harmonic,1)
    if tune > 0.99*harmonic{h,1} && tune < 1.01*harmonic{h,1}
        mix = harmonic{h,2};
        break
    end
end
prod = make_song(mix,tune);
song = prod(1:time*4000).*envelope_linear(time*4000,choice);
```

其余代码详见源文件。

三、 实验总结

这次实验内容丰富有趣，既锻炼了 MATLAB 编程能力，又复习了信号与系统的相关知识，还学到了不少音乐的知识。在此特别感谢李逢双同学对我在音乐方面的帮助，告诉了我很多音乐方面的专业知识。

四、 参考

- 1、宋知用. MATLAB 在语音信号分析与合成中的应用[M]. 北京航空航天大学出版社, 2013.