音乐合成实验

无13班 兰琪 2021012693

1.2.1 简单的合成音乐

（1）合成正弦音乐

计算《东方红》片段各个乐音的频率

$1=F$ 时，$f(1)=f(F)=349.23\text{Hz}$，

$f\left(\dot{6}\right)=f(D)=f(F)\times 2^{-3/12}=293.67\text{Hz}$

$f\left(\dot{7}\right)=f(E)=f(F)\times 2^{-1/12}=329.63\text{Hz}$

$f(2)=f(G)=f(F)\times 2^{2/12}=392.00\text{Hz}$

$f(3)=f(A)=f(F)\times 2^{4/12}=440.00\text{Hz}$

$f(4)=f({}^{b}B)=f(F)\times 2^{5/12}=466.17\text{Hz}$

$f(5)=f(C)=f(F)\times 2^{7/12}=523.25\text{Hz}$

$f(6)=f(D)=f(F)\times 2^{9/12}=587.33\text{Hz}$

唱名	$\dot{6}$	$\dot{7}$	1	2	3	4	5	6
音名	D	E	F	G	A	${}^{b}B$	C	D
频率（Hz）	293.67	329.63	349.23	392.00	440.00	466.17	523.25	587.33

生成幅度为 1 、抽样频率为 8kHz 的正弦信号

取一节的时间长度为 1 秒

```
sample_rate = 8000;
section_duration = 1;
```

产生长为 1 节（ 2 拍）的时间序列

```
t_double = (0 : 1/sample_rate : section_duration-1/sample_rate)';
```

1 节时间内的样点数

```
count_double = length(t_double);
```

确定 3 组 21 个频率值，包含低音、中音、高音 1 2 3 4 5 6 7

```
Freq_0 = 349.23;
freq = Freq_0 * 2.^([0,2,4,5,7,9,11]/12);
freq_flat = freq / 2;
freq_sharp = 2 * freq;
```

产生正弦单音

```
double_tone_mat = sin(t_double * 2*pi*[freq_flat,freq,freq_sharp]); % 2   拍
       tone_mat = double_tone_mat(1:count_double/2,:);              % 1   拍
  half_tone_mat = double_tone_mat(1:count_double/4,:);              % 1/2 拍

tones = mat2cell([half_tone_mat;tone_mat;double_tone_mat], ...
    [count_double/4,count_double/2,count_double], ones(21,1))';
```

其中 `tones` (21*3 `cell`) :

	1/2 拍 (1)	1 拍 (2)	2 拍 (3)
降调 (1~7)	half_tone_flat	tone_flat	double_tone_flat
(8~14)	half_tone	tone	double_tone
升调 (15~21)	half_tone_sharp	tone_sharp	double_tone_sharp

采用变调和时长调整标记，用于在 `tones` 索引

```
s =  -7; % sharp
f =   7; % flat
d = -21; % double
h =  21; % half
```

取一维索引时， `tones` 的 63 个元素按照 1/2 拍、 1 拍、 2 拍分为三大组；每大组按照降调、中音、升调分为三小组；每小组按照 1~7 的顺序排列单音。通过引入零点偏移 28 （`中音 1 索引` - 1），可以将减法运算 `-s` / `-f` ， `-h` / `-d` 作为标记，在 `tones` 中索引，并且 `-s` / `-f` 与 `-h` / `-d` 可以相连，不分先后。

例如 `tones{28 + 2-s-h}` 将取第 16 个元素，即 1/2 拍 $\dot{2}$ 。

用 sound 函数播放每个乐音

播放 6 ， 7 ， 1 ， 2 ， 3 ， 4 ， 5 ， 6

```
clips = generate_clips(tones, [6-f, 7-f, 1, 2, 3, 4, 5, 6]);
sound(clips, sample_rate);
```

其中 `generate_clips` 如下

```
function clips = generate_clips(tones, table)
    clips = [];
    for i = 1 : length(table)
        tone = tones{28 + table(i)};
        clips = [clips; tone];
    end
end
```

播放的音调是正确的。

拼出《东方红》片断

```
clips = generate_clips(tones, [5, 5-h, 6-h, 2-d, 1, 1-h, 6-f-h, 2-d]);
sound(clips, sample_rate);
```

合成的片段具有正确的音调和节拍，但是每个声音都沉闷单调，相邻乐音之间切换时有杂声，并且不同音调的音色特征差距较大，没有明显的音色特点，很容易听出不是乐器演奏。

（2）合成具有包络的音乐

包络修正

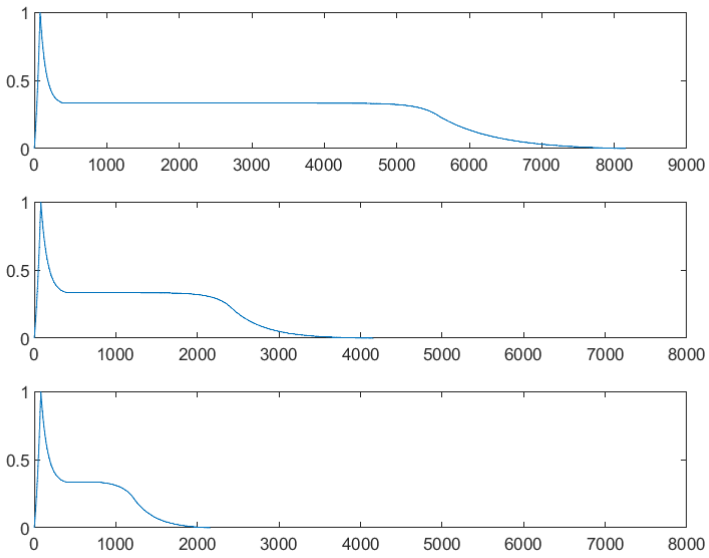
迭接部分时长

```
overlapping_length = 0.02;
```

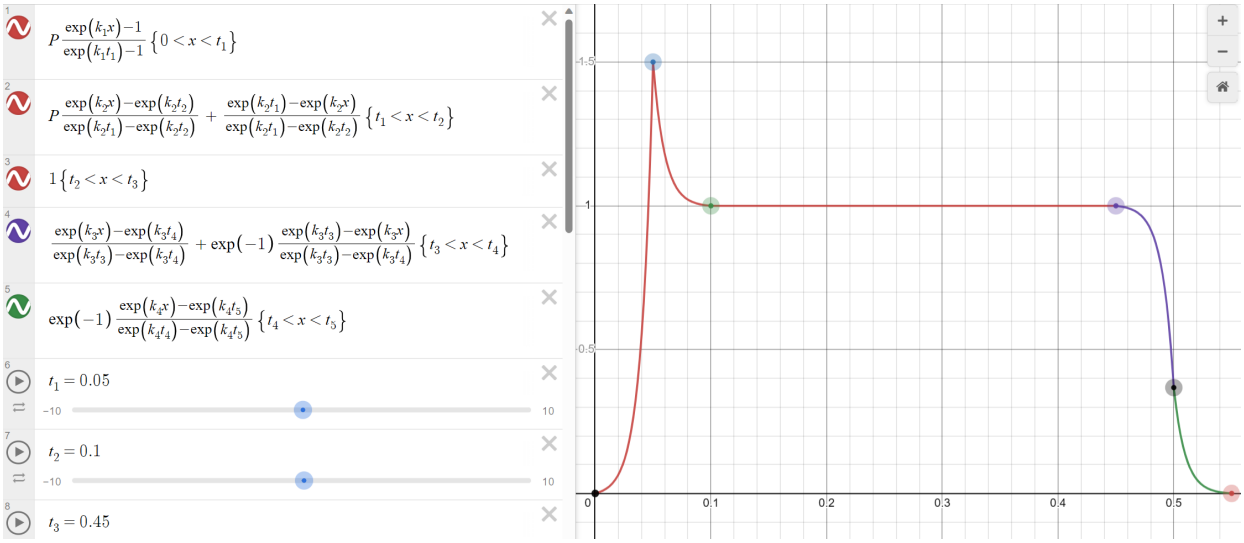
对 2 拍，1 拍，1/2 拍单音分别调整包络修正参数

```
double_envelope = music_envelope( ...
    [0.01,0.05,0.20,0.70,1.02], 3, [100,-100,30,-10], ...
    sample_rate);
envelope        = music_envelope( ...
    [0.01,0.05,0.15,0.30,0.52], 3, [100,-100,40,-20], ...
    sample_rate);
half_envelope   = music_envelope( ...
    [0.01,0.05,0.10,0.15,0.27], 3, [100,-100,50,-30], ...
    sample_rate);
```

产生包络



`music_envelope` 将产生下图 5 段曲线，包含 4 段指数曲线和 1 段直线。



图中直线部分 $y=1$，右数第二个控制点为 (t_4, e^{-1})。包络用 $0 \sim t_1$ 的指数段体现冲激上升；$t_1 \sim t_3$ 段体现冲激过后音量降调并保持一段时间不变；$t_3 \sim t_5$ 段用两段指数控制衰减，可以使包络比较光滑。包络将通过纵向压缩使最高值 $=1$。

通过使 $t_2=t_3$ 并调整其他参数也可以产生类似钢琴的包络。

包含迭接部分时长，产生单音

```
t_double = (0 : 1/sample_rate : overlapping_length+section_duration)';
t_double = t_double(1:end-1);

count_double = length(t_double);
count        = length(0:1/sample_rate:overlapping_length+section_duration/2)-1;
count_half   = length(0:1/sample_rate:overlapping_length+section_duration/4)-1;

double_tone_mat = sin(t_double * 2*pi*[freq_flat,freq,freq_sharp]);
       tone_mat = double_tone_mat(1:count,:);
  half_tone_mat = double_tone_mat(1:count_half,:);
```

将包络与单音相乘

```
double_tone_mat = double_envelope .* double_tone_mat;
       tone_mat =        envelope .*        tone_mat;
  half_tone_mat =   half_envelope .*   half_tone_mat;

tones = mat2cell([half_tone_mat;tone_mat;double_tone_mat], ...
    [count_half,count,count_double],ones(21,1))';
```

修改 `generate_clips` 以适用于连接

```
function clips = generate_clips(tones, table, sample_rate, overlapping_length)
    tail_duration = overlapping_length * sample_rate;
    clips = [];
    tail = zeros(tail_duration,1);
    for i = 1:length(table)
        tone = tones{28 + table(i)};
        clips = [ ...
            clips; ...
            [tail;zeros(length(tone)-2*tail_duration,1)]+tone(1:end-tail_duration) ...
            ];
        tail = tone(end-tail_duration+1:end);
    end
    clips = [clips;tail];
end
```

播放《东方红》片段

```
clips = generate_clips(tones,[5,5-h,6-h,2-d,1,1-h,6-f-h,2-d], ...
    sample_rate, overlapping_length);
sound(clips,sample_rate);
```

此时片段听起来不如之前连贯，但切换音调时的杂音明显消失了。包络的应用使每个音的独立性增强，节奏效果变好但连贯性变差。

（3）调整音调

将 (2) 中的音乐分别升高和降低一个八度

升高

```
sound(clips(1:2:end), sample_rate);
```

降低

```
sound(clips, sample_rate/2);
```

使用 `resample` 函数升高一个半音

频率 $f \to 2^{1/12} \cdot f$，周期 $T \to 2^{-1/12}T$，播放时长 $t \to 2^{-1/12}t$，在相同采样率下播放，样点数 $\propto$ 时长 t，需使样点数 $N \to 2^{-1/12}N$，即 $2^{-1/12}$ 倍重采样。

```
clips = resample(clips,round(sample_rate*2^(-1/12)), sample_rate);
sound(clips, sample_rate);
```

（4）合成具有谐波的音乐

谐波相对幅度取 0.2， 0.3

```
 double_tone_mat = sin(t_double * 2*pi*[freq_flat,freq,freq_sharp]) ...
           + 0.2 * sin(t_double * 2*pi*[freq_flat,freq,freq_sharp]*2) ...
           + 0.3 * sin(t_double * 2*pi*[freq_flat,freq,freq_sharp]*3);
        tone_mat = double_tone_mat(1:count,:);
   half_tone_mat = double_tone_mat(1:count_half,:);

 double_tone_mat = double_envelope .* double_tone_mat;
        tone_mat =        envelope .*        tone_mat;
   half_tone_mat =   half_envelope .*   half_tone_mat;

 tones = mat2cell([half_tone_mat;tone_mat;double_tone_mat], ...
     [count_half,count,count_double], ones(21,1))';
```

播放《东方红》片段

```
clips = generate_clips(tones,[5,5-h,6-h,2-d,1,1-h,6-f-h,2-d], ...
    sample_rate, overlapping_length);
sound(clips,sample_rate);
```

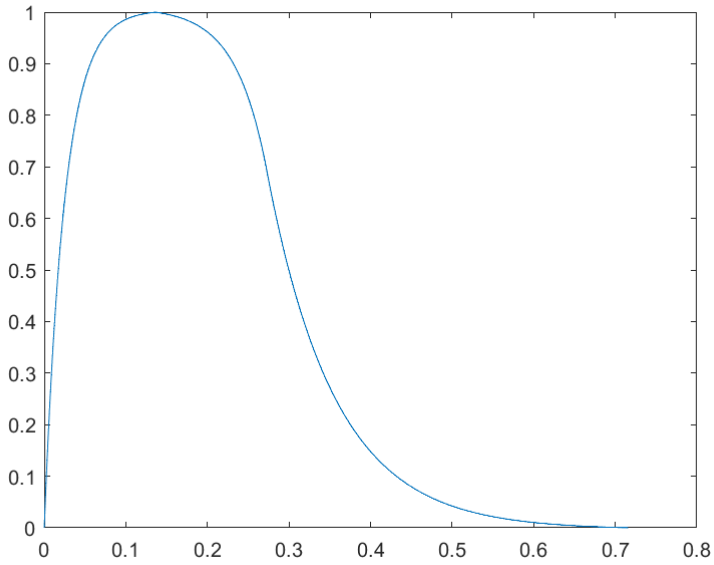
此时片段的音色更加清脆，声音确实像风琴。增加谐波分量或调整相对大小可以产生其他音色。我通过尝试得到了类似笛子或尺八的音色：`[0.5 0.3 0.1]`。

（5）合成其他音乐

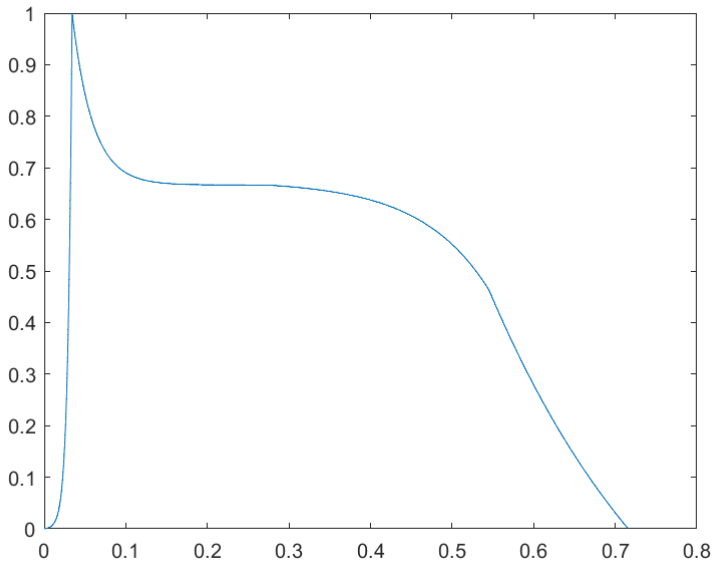
使用句柄类 `music_synthesis` 封装上述功能。

使用类 `music_const` 保存一些常数，包括速度标记（参考《百度百科》），频率值，谐波分量和包络参数。其中钢琴谐波和包络参考自“*程美芳. 钢琴音色识别与电子合成系统的设计与实现[D].电子科技大学,2014.*”

钢琴包络：



尺八包络：



（经过尝试确定的采用值，参考教材图 1.4 可知与实际情况应有较大差距，考虑要与 `music_envelope` 兼容，进一步调整需要重新设计 `music_envelope` ，由于时间有限，没有进行相关改进）

播放《东方红》片段

```
The_East_Is_Red = music_synthesis( ...
        sample_rate = 8000, ...
        speed = 'Moderato', ...
        beat_tempo = '2/4', ...
        base_frequency = 'f1', ...
        envelope_description = music_const.envelope_piano, ...
        harmonic =  music_const.harmonic_piano ...
    );

The_East_Is_Red.clips_description = [
    '5 _5 _6 ' ...
    '2 - ' ...
    '1 _1 _b6 ' ...
    '2 -'
    ];

audiowrite('The_East_Is_Red_1.wav', The_East_Is_Red.clips,
The_East_Is_Red.sample_rate);
```

播放《夜明》片段

```
Dawn = music_synthesis( ...
        sample_rate = 8000, ...
        speed = 'Andantino', ...
        beat_tempo = '4/4' ...
    );

clips_description_1 = [
    '0 ' ...
    '6 . _#1 7 _6 _5 ' ...
    '_5 _6 3 - _3 _5 ' ...
    '6 . _#1 7 _6 _7 ' ...
    '7 - - _3 _5 ' ...
    '6 . _#1 7 _6 _5 ' ...
    '_5 _6 3 - _2 _1 ' ...
    '_6 3 _5 _3 _2 _1 _2 ' ...
    'b6 - - - '
    ];

% ... 省略部分内容, 详见 main.m ...

Dawn.base_frequency = 'bb';
Dawn.envelope_description = music_const.envelope_piano;
Dawn.harmonic = music_const.harmonic_piano;
Dawn.clips_description = clips_description_1;
clips_1 = Dawn.clips;

% ... 省略部分内容, 详见 main.m ...

audiowrite('Dawn.wav', [clips_1; clips_2; clips_3; clips_4; clips_5],
Dawn.sample_rate);
```

播放《玛丽有只小羊羔》片段

```
Mary_Had_a_Little_Lamb = music_synthesis( ...
        sample_rate = 8000, ...
        speed = 96, ...
        beat_tempo = '4/4', ...
        base_frequency = 'c1', ...
        envelope_description = music_const.envelope_piano, ...
        harmonic = music_const.harmonic_piano ...
    );

Mary_Had_a_Little_Lamb.clips_description = [
    '3 2 1 2 ' ...
    '3 3 3 - ' ...
    '2 2 2 - ' ...
    '3 3 3 - ' ...
    '3 2 1 2 ' ...
    '3 3 3 1 ' ...
    '2 2 3 2 ' ...
    '1 - - - ' ...
];

audiowrite('Mary_Had_a_Little_Lamb.wav', ...
    Mary_Had_a_Little_Lamb.clips, Mary_Had_a_Little_Lamb.sample_rate);
```

《夜明》分为 5 段，其中有用钢琴和尺八演奏相同音乐然后叠加的部分。叠加后的效果类似于增强了尺八的重音，让尺八音像弦音，在尺八音较弱时可以听到兼像弦音和管音的钢琴音，产生了一些混乱，与预期效果有差距。另因设计 `music_synthesis` 没有专为叠加做相关准备，连接部分长度与音长有关，当拼接的两片段最后一个音时长不同时需要补充 0 。

《玛丽有只小羊羔》片段的钢琴很想真实的钢琴，节奏清楚，可以明显听出类似按键的声音。

当频率较高时，钢琴音色听起来像管乐。采用相同包络和谐波分量的《玛丽有只小羊羔》片段就比《东方红》片段更像钢琴曲。考虑到采用的钢琴包络相比于参考材料，前半部分和后半部分的差距更大，音量接近饱和的时间和逐渐衰减的时间更长，中间变化过程更快，并且所有音调取了同一包络和谐波分量，这些因素可能使高音部分与真实乐器产生较大差距。

`music_synthesis` 基于句柄类设计，具有 `PostSet` 事件回调函数，当之前修改的属性不影响要访问的属性时，不会重复计算。

当前 `music_synthesis` 仅支持简谱描述，可以支持延时线、减时线、附点音符，但不支持直接音符描述，不支持连音线、和弦，对变调、变速、变音有相应功能支持，但封装程度不高，需要手动拆分乐谱，并多次更改相关参数。以上不支持或支持有限的功能可以留待日后改进。

1.2.2 用傅里叶级数分析音乐

载入 `guitar.mat`

```
load('Guitar.MAT');
```

（6）真实音乐

用 `audioread` 函数载入 `fmt.wav` 文件

```
[clips, sample_rate] = audioread('fmt.wav');
sound(clips, sample_rate);
```

这段音频比合成音乐真实，音调变化连贯，并且内容十分丰富，并且音色特征鲜明，可以从片段清晰分辨出乐器类型。

（7）去除非线性谐波和噪声

可以采用时间同步平均方法去除非线性谐波和噪声，为此，先确定基音周期。考虑真实值 `realwave` = 线性周期部分 `wave2proc` + 非线性谐波与噪声 `noise` ，设基音周期为 T ，则

$\mathrm{wave2proc}(t+T)=\mathrm{wave2proc}(t)$，自相关函数 $R_{\mathrm{wave2proc}}(t)$ 在周期 T 整数倍处出现峰值，

并假设自相关函数 $R_{\mathrm{noise}}(t)=N\delta(t)$ ，互相关函数 $R_{\mathrm{wave2proc,\,noise}}=R_{\mathrm{noise,\,wave2proc}}=0$ ，于是

$$R_{\mathrm{realwave}}(t)=R_{\mathrm{wave2proc}}+R_{\mathrm{noise}}+R_{\mathrm{wave2proc,\,noise}}+R_{\mathrm{noise,\,wave2proc}}=R_{\mathrm{wave2proc}}(t)+N\delta(t)$$

将具有周期 T ，从 R_{realwave} 获取基波频率可以更好地避免噪声干扰。确定周期后，将音乐按周期叠加以抵消 `noise` ，从而得到 `wave2proc_2` 。

估计基音周期

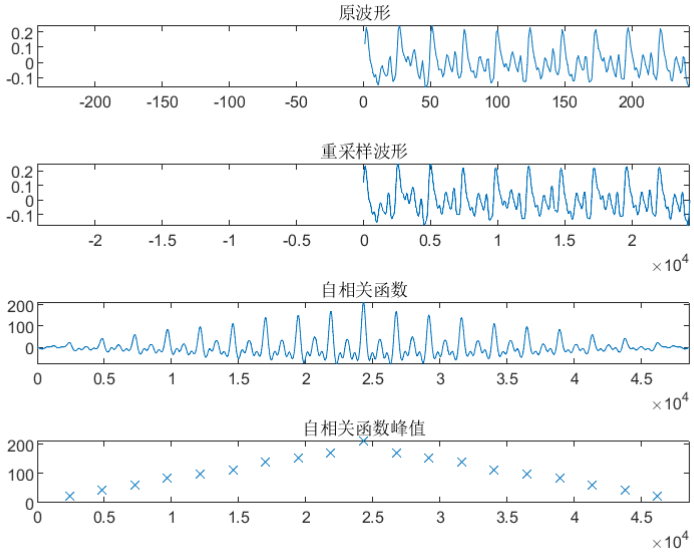
```
R_realwave = xcorr(realwave);                  % 自相关函数
[peak_y, peak_x] = findpeaks(R_realwave);      % 包含全部峰值
peak_x = peak_x (peak_y > 0.75 * max(peak_y)); % 仅保留主要峰值

point_count = floor(length(peak_x)/2);         % 逐差点数
point_distance = ceil(length(peak_x)/2);       % 逐差点间距
approx_T = (sum(peak_x(end-point_count+1:end)) - sum(peak_x(1:point_count))) ...
    / point_distance / point_count;
```

确定基音周期

```
resample_wave = resample(realwave, 100, 1);
R_resample_wave = xcorr(resample_wave);
[peak_y, peak_x] = findpeaks(R_resample_wave, MinPeakDistance=approx_T*100*0.8);

point_count = floor(length(peak_x)/2);
point_distance = ceil(length(peak_x)/2);
resample_wave_T = (sum(peak_x(end-point_count+1:end)) - sum(peak_x(1:point_count)))
...
    / point_distance / point_count;            % 重采样波形一周期样点数
```



时间同步平均

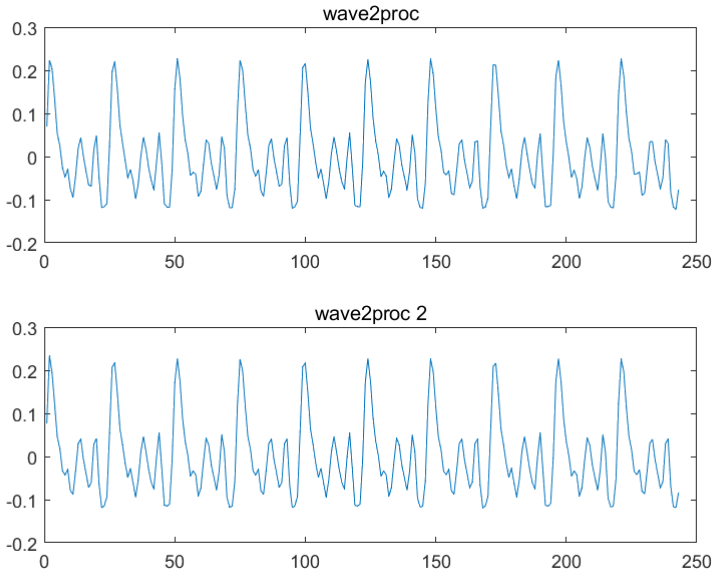
```
resample_wave_length = length(resample_wave);                          % 重采样波形样点数
resample_wave_T_count = floor(resample_wave_length / resample_wave_T); % 重采样波形周期数
resample_wave_single_T = tsa(resample_wave, 1,
0:resample_wave_T:resample_wave_length);
```

周期重复

```
wave2proc_2 = repmat(resample_wave_single_T, resample_wave_T_count, 1);
wave2proc_2 = [wave2proc_2; ...
    resample_wave_single_T(1 : resample_wave_length-length(wave2proc_2))];
```

还原采样率

```
wave2proc_2 = resample(wave2proc_2, 1, 100);
```



`wave2proc_2` 与参考波形 `wave2proc` 比较相似。

> 注：采用时间同步平均（TSA）消除非线性谐波和噪声的思路参考自 matlab tsa文档，csdn 相关主题文档，以及九字班某同学作业版本。确定使用该方法可以实现题目所述效果后，本人自行完成了相关代码。

（8）用傅里叶变换分析谐波分量

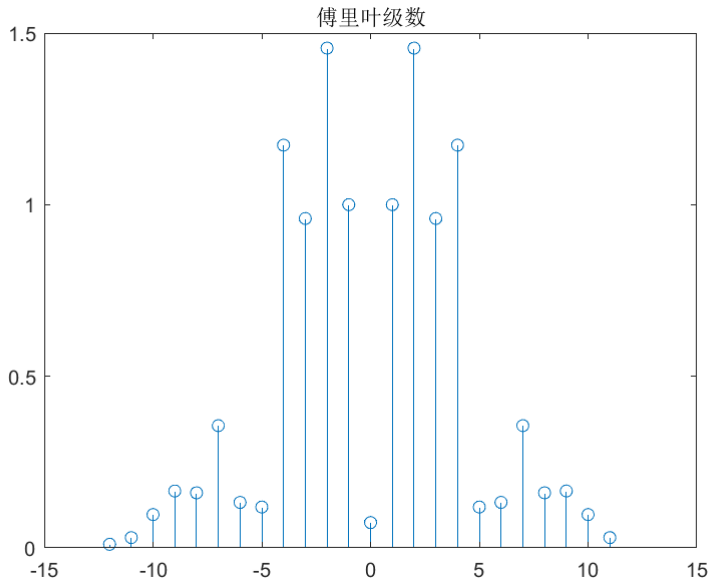
由（7）已经确定 `resample_wave_T` $= 24.333 \times 100$，基频

```
base_frequency = 1 / (resample_wave_T / 100 / sample_rate);
```

`base_frequency` $= 328.77\text{Hz}$，接近 E_1 调 329.63Hz。

近似取一个周期求傅里叶级数

```
wave2proc_T = round(resample_wave_T / 100);       % wave2proc 一周期样点数
wave2proc_single_T = wave2proc(1 : wave2proc_T);  % wave2proc 单周期
x_shift = (-wave2proc_T/2 : wave2proc_T/2-1)';    % 零点居中 x 坐标
DFS = abs(fftshift(fft(wave2proc_single_T))) / length(wave2proc_single_T);
stem(x_shift, DFS / DFS(x_shift == 1)); % 基频幅值调整为 1
title('傅里叶级数');
```



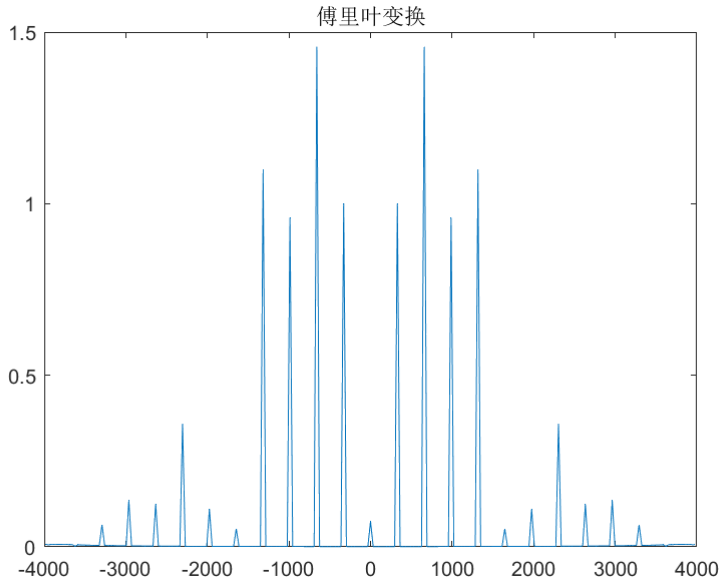
谐波($\times f_{E_1}$)	2	3	4	5	6	7	8	9	10	11
相对幅值	1.456	0.960	1.173	0.119	0.133	0.356	0.160	0.166	0.097	0.030

用傅里叶变换 DFT 分析谐波分量

$$
\begin{aligned}
CTFT\left[f\left(t\right)\cdot\left(u\left(t\right)-u\left(t-T\right)\right)\right] &= \int_0^T f(t)\exp(-j\omega t)\mathrm{d}t \\
&\approx \sum_{n=0}^{N-1} x(n)\exp(-j\omega n\Delta t)\Delta t \\
&= \Delta t\sum_{n=0}^{N-1} x(n)\exp\left(-j\frac{2\pi}{N}n\frac{N\Delta t\omega}{2\pi}\right) \\
&= \Delta t\cdot DFT[x(n)]\left(N\Delta t f\right)
\end{aligned}
$$

其中 $\Delta t = 1/$ `sample_rate`

```
wave2proc_length = length(wave2proc);
x_shift = sample_rate / wave2proc_length * ...
    (-floor(wave2proc_length/2) : ceil(wave2proc_length/2-1))';
DFT = abs(fftshift(fft(wave2proc)));
base_frequency_peak = max(DFT(x_shift-base_frequency<10 & x_shift-
base_frequency>-10));
plot(x_shift, DFT / base_frequency_peak);
title('傅里叶变换');
```



频率(Hz)	329.22	658.44	987.65	1316.87	1646.09	1975.31
相对幅值	1	1.457	0.959	1.100	0.052	0.110
频率(Hz)	2304.53	2633.74	2962.96	3292.18	3588.48	3917.70
相对幅值	0.359	0.124	0.135	0.064	0.006	0.008

因

$$
\begin{aligned}
CTFT\left[f\left(t\right)\cdot\left(u\left(t\right)-u\left(t-T\right)\right)\right] &= CTFT[f(t)] * CTFT[u\left(t\right)-u\left(t-T\right)] \\
&= CTFT[f(t)] * T\mathrm{Sa}\left(\frac{\omega T}{2}\right)\exp\left(-j\omega\frac{T}{2}\right)
\end{aligned}
$$

即

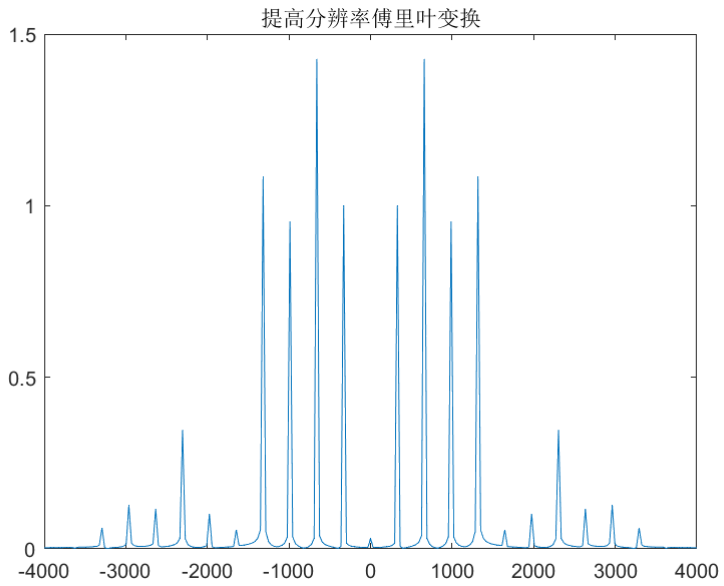
$$|DFT[x(n)](k)| = c_0 \cdot \left| CTFT[f(t)]\left(\frac{2\pi}{N\Delta t}k\right)\right| * \mathrm{Sa}\left(\frac{k}{\pi}\right)$$

其中 c_0 是幅值调整系数。

理想 $CTFT[f(t)]$ 为一组冲激函数，于是 DFT 将以 $\mathrm{Sa}\left(\frac{k}{\pi}\right)$ 为包络。通过减小 Δt 提高分辨率可以减小求和近似积分的误差，但不影响 Sa 包络；通过增加信号时长 $N\Delta t$ 可以增大 $CTFT[f(t)]\left(\frac{2\pi}{N\Delta t}k\right)$ 峰间距，再将横坐标取相同宽度，可以使峰更尖锐。

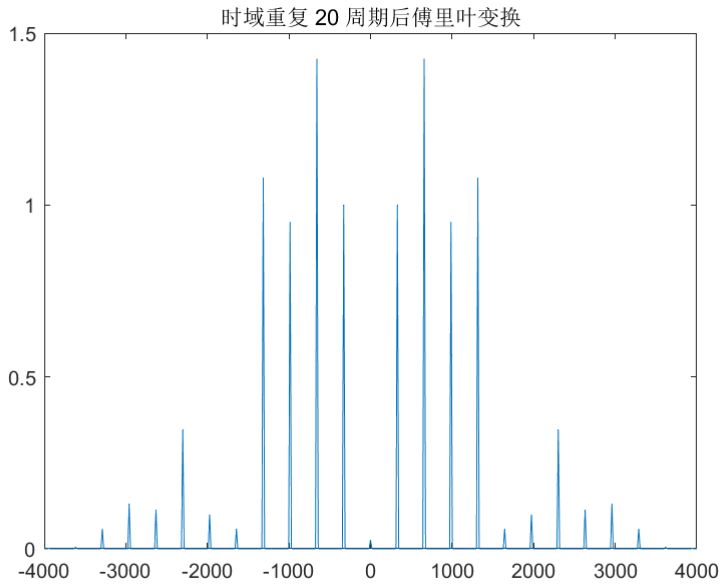
提高分辨率

```
x_shift = sample_rate * 100 / resample_wave_length * ...
    (-floor(resample_wave_length/2) : ceil(resample_wave_length/2-1))';
DFT = abs(fftshift(fft(resample_wave2proc_2)));
base_frequency_peak = max(DFT(x_shift-base_frequency<10 & x_shift-
base_frequency>-10));
plot(x_shift, DFT / base_frequency_peak);
xlim([-4000, 4000]);
title('提高分辨率傅里叶变换');
```



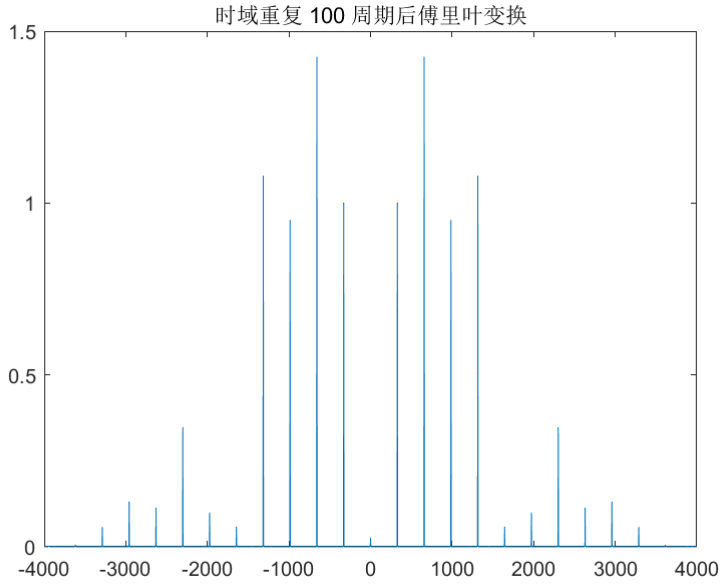
因（7）给出的 `wave2proc_2` 并非严格 10 个周期，现用 `resample_wave_single_T` 时域重复后傅里叶变换。时域重复 20 周期后傅里叶变换

```
repeat_resample_wave = repmat(resample_wave_single_T, 20, 1);
repeat_resample_wave_length = length(repeat_resample_wave);
x_shift = sample_rate * 100 / repeat_resample_wave_length * ...
    (-floor(repeat_resample_wave_length/2) : ceil(repeat_resample_wave_length/2-1))';
DFT = abs(fftshift(fft(repeat_resample_wave)));
base_frequency_peak = max(DFT(x_shift-base_frequency<10 & x_shift-
base_frequency>-10));
plot(x_shift, DFT / base_frequency_peak);
xlim([-4000, 4000]);
title('时域重复 20 周期后傅里叶变换');
```



频率(Hz)	328.81	657.62	986.44	1315.25	1644.06	1972.87
相对幅值	1	1.425	0.951	1.080	0.058	0.100
频率(Hz)	2301.69	2630.50	2959.31	3288.12	3616.93	3945.75
相对幅值	0.346	0.114	0.129	0.058	0.004	0.002

时域重复 100 周期后傅里叶变换



频率(Hz)	328.81	657.62	986.44	1315.25	1644.06	1972.87
相对幅值	1	1.4254	0.9511	1.0796	0.0576	0.0999
频率(Hz)	2301.69	2630.50	2959.31	3288.12	3616.93	3945.75
相对幅值	0.3464	0.1137	0.1290	0.0578	0.0044	0.0020

此时 DFT 具有明显的冲激函数形式。

（9）分析音调和节拍

使用句柄类 `music_analysis` 封装上述功能

```
[guitar_clips, sample_rate] = audioread('fmt.wav');

guitar_analysis = music_analysis( ...
    clips = guitar_clips ...
    );

guitar_analysis.analyse();
disp(guitar_analysis.tone);
disp(guitar_analysis.beat);
```

`analyse()` 包含三个步骤：

1. 时间分隔 `divide()`

   `music_analysis` 没有基于音量区分节拍，而是对信号做短时傅里叶变换 $STFT$ 得到时频图，比较不同时刻频率向量的内积来确定频率的变换。相比于基于音量的方法，此方法识别连音线连接的音等音量变换不明显的节拍时具有优势，但同时有误标记和对参数敏感的问题。为避免误标记带来的影响，`analyse()` 包含一个后处理过程。

   完成分段后，取每段中间靠前的一段波形进行下一步分析。

2. 频率和谐波分析 `frequency_harmonic()`

   经过尝试，将估计基音周期的方法改为先对信号做两次自相关，再做离散傅里叶变换 DFT，取最高峰值对应频率，如果频率高出合理范围，取对应频率的一半，如果低于合理范围，取对应频率的两倍。其他过程与（7）（8）基本相同。

   经过测试，约有 `1/5` 的波形片段出现最高峰值对应频率超出合理范围的问题。对于 `realwave`，用乘 2 或除以 2 的方式可以得到基本正确的估计值。逐一查看 DFT 波形发现，除一段波形在合理区间内峰值不尖锐，两端波形应取 3 倍而非 2 倍，可以期待这一方法适用于其他出错波形片段中的大部分。

3. 节奏分析 `beat_analyse()`

   首先将频率相同并且音量不突然增大的两端分隔合并，然后取时间间隔中位数作为参考，根据其与其他时间间隔的比值确定节拍，并取整数近似。另外本阶段也对每个音调的谐波分量相对大小取均值。

将 `fmt` 的分析结果重新合成，音乐的主体节奏是正确的，可以听出几个具有标志性的长音，但大多数音调分析似乎不正确。并且因为合成程序不支持连音线，音乐节奏较快时跳跃感太强。

`music_analysis` 基于句柄类设计，具有 `PostSet` 事件回调函数，当修改参数或待分析音频后可以实时更新分析结果。并且由于音乐分析相比音乐合成参数更多，也更容易出现错误或警告，`music_analysis` 中包含了简单的抛出警告和错误处理功能，可以帮助定位出错参数，便于进行调整。

由于该类的参数很多，当前预设调节参数仅能保证基本适用于 `fmt.wav` ，对于其他音频可能无法给出准确结果，甚至可能出现错误或警告，其可靠性和易用性不高。当前 `music_analysis` 的功能实现情况与真正使用还有差距，改进空间很大。通过采用更优的分析思路，分析工具，参数预设，进一步提高稳定性和准确率是下一步改进的主要内容。

（10）用 (7) 得出谐波分量合成《东方红》片段

在 `music_const` 中增加

```
harmonic_guitar = [
    1.4254 0.9511 1.0796 0.0576 0.0999 0.3464 ...
    0.1137 0.1290 0.0578 0.0044 0.0020 ...
    ]
```

此时《东方红》片段听起来比（4）更轻盈，但听起来还是更像风琴，不像吉他。

（11）用（8）得出谐波分量合成《东方红》片段

```
% C 调谐波分量
harmonic_guitar_c = guitar_analysis.harmonic.c1;

% ... 省略部分内容，详见 main.m ...

The_East_Is_Red = music_synthesis( ...
        sample_rate = 8000, ...
        speed = 'Moderato', ...
        beat_tempo = '2/4', ...
        base_frequency = 'f1', ...
        envelope_description = music_const.envelope_guitar ...
    );
The_East_Is_Red.harmonic = harmonic_guitar_c;
The_East_Is_Red.clips_description = '5 _5';
clips_1 = The_East_Is_Red.clips;

% ... 省略部分内容，详见 main.m ...

clips = [clips_1; clips_2; clips_3; clips_4; clips_5; clips_3];

audiowrite('The_East_Is_Red_3.wav', clips, The_East_Is_Red.sample_rate);
```

此时《东方红》片段高音部分与（4）几乎无区别，低音部分听起来比（4）略低，但和吉他还是相差很多。

（12）设计图形界面

当前图形界面程序比较简陋，后续可以适当优化，丰富乐器种类，并考虑添加音乐分析功能。