Matlab 音乐合成大作业 实验报告

一、实验目的

1. 学习使用matlab工具在时域和频域处理信号；
2. 了解基本的乐理知识，掌握使用傅里叶级数分析音乐的方法，增进对傅里叶级数的理解；
3. 使用傅里叶级数等方法处理信号，并合成一段音乐。

二、实验平台

Matlab R2023a

三、实验内容

1. 简单的合成音乐

(1)简单合成音乐

首先我定义好采样频率、每拍节奏、基调等参数：

```
global sample_freq;
global base_tone_freq;
global beat_time;
sample_freq = 8e3;
% 1 = F
base_tone_freq = 349.23;
% beat_time = 0.5, or BPM = 120
beat_time = 0.5;
```

然后定义曲谱和节奏：

```
% 曲谱
tone = [5, 5, 6, 2, 1, 1, -1, 2];
beat = [1, 0.5, 0.5, 2, 1, 0.5, 0.5, 2];
```

其中，beat的单位是拍，tone是唱名，"do, re, mi"开始依次编号为1, 2, 3...，1比1少一个周期（每周期为7），因此为-6。

为了映射到十二平均律对应的频率，我定义了唱名与指数之间的关系：

```
% 将唱名映射至以2^(1/12)为底的指数，1对应指数为1
tone_mapping = [0, 2, 4, 5, 7, 9, 11];
```

进而计算乐音的音调，频率和时长：

```
remain = mod(tone - 1, 7) + 1;
exponent = tone_mapping(remain) + (tone - remain)/7 * 12;
freq = base_tone_freq * (2^(exponent/12));
width = beat * beat_time;
```

生成正弦波：

```
time_step = sample_freq^(-1);
t = 0:time_step:width;
y = amp * sin(2 * pi * freq * t);
```

拼接正弦波，形成音乐：

```
first_part = cell2mat(arrayfun(@(x, y) gen_tune(x, y), tone, beat,
UniformOutput=false));
sound(first_part, sample_freq);
```

启动`exer_1_1.m`，发现确实听见了东方红的音调，但是声音较单薄，且音符邻接位置有啪的杂音。

(2)添加包络

我采用方程$y = ae^{kx} + b$来拟合包络，使得包络线有着类似于指数的曲线，同时便于求解：

```
function [output, envelop] = exponential_envelop(y, amp1, amp2, coe)
    [row, col] = size(y); % row = 1, col = n
    coe = col^(-1) * coe;
    func = @(x) exp(coe * x);
    [output, envelop] = func_envelop(y, amp1, amp2, func);
end

function [output, envelop] = func_envelop(y, amp1, amp2, func)
    [row, col] = size(y); % row = 1, col = n
    [a, b] = func_fit(1, amp1, col, amp2, func);
    x = 1:1:col;
    envelop = a * func(x) + b;
    output = y .* envelop;
end
function [a, b] = func_fit(x1, y1, x2, y2, func)
    % 计算指数函数参数
    a = (y2 - y1) / (func(x2) - func(x1));
    b = y1 - a * func(x1);
end
```

其中，k是由音符的时长来确定的。

生成包络：

```
% 包络修正
function [y0, y1, y2, y3] = generate_fixed(width, sample_freq)

    time_step = sample_freq^(-1);

    t0 = 0:time_step:width * 0.09;
    t1 = 0:time_step:width * 0.05;
    t3 = 0:time_step:width * 0.95;
    t2 = 0:time_step:(width * 0.01);

    [unused, y0] = exponential_envelop(t0, 0, 1, -5);
    [unused, y1] = exponential_envelop(t1, 1, 0.9, -5);
    [unused, y2] = exponential_envelop(t2, 0.9, 0.9, 1);
    [unused, y3] = exponential_envelop(t3, 0.9, 0, -2);
end
```

其中，t0, t1, t2, t3分别对应的是冲激，衰减，持续和消失。考虑到迭接部分，总的时长为1.1，有10%的迭接部分。

```
function [result, overlap] = gen_tune(tone, beat)
    global amp;
    global sample_freq;
    global tone_mapping;
    global overlap_ratio;
    amp = 1;
    % 将唱名映射至以2^(1/12)为底的指数，1对应指数为1
    tone_mapping = [0, 2, 4, 5, 7, 9, 11];
    overlap_ratio = 0.1;

    time_step = sample_freq^(-1);

    [freq, width] = trans_freq_width(tone, beat);

    [y0, y1, y2, y3] = generate_fixed(width);

    t = 0:time_step:(length([y0, y1, y2, y3])-1) * time_step;

    result = amp * sin(2 * pi * freq * t) .* [y0, y1, y2, y3];
    overlap = round(overlap_ratio * length(y3));
end
```

迭接部分的代码：

```
% 初始化空数组用于存储结果
result = [];
loop = 1:length(tone);
overlap_last = 0;
% 循环迭代
for i = loop
    % 调用 gen_tune 函数
    [local_result, overlap] = gen_tune(tone(i), beat(i));
    % 将结果的首尾相加，中间拼接
    if i == 1
        % 第一次迭代，直接将结果添加到结果数组
        result = local_result;
    else
        % 非第一次迭代，将上一次结果的末尾与当前结果的开头相加，并将结果添加到结果数组
        result = [result(1:end-overlap_last), (result(end-overlap_last+1:end) +
local_result(1:overlap_last)), local_result(overlap_last+1:end)];
    end
    overlap_last = overlap;
end
```

启动`exer_1_2.m`，音符间的啪声消失，并且响度出现了强弱变化。

(3)升八度、降八度，升半音

sound的采样率翻倍，则升八度，速度加快：

```
% 升八度
sound(music, sample_freq * 2)
```

采样率减半，则降八度，速度减慢：

```
% 降八度
sound(music, sample_freq / 2)
```

升半音可以通过resample重采样实现，重采样的采样率为原来的$2^{-1/12}$倍，但是sound的采样率不变，因此sound的播放的速度更快了：

```
% 升半音
tsin = timeseries(music', 1:length(music));
tsout = resample(tsin, 1:(2^(1/12)):length(music));

rs_music = tsout.Data;

sound(rs_music, sample_freq);
```

上述操作在改变声音频率的同时也改变了时长，而且其变化倍数互为倒数。

(4)增加高次谐波

这次把生成正弦波的函数改写为含有高次谐波的波形：

```
function wave = gen_waveform(freq, len)
    global amp;
    global sample_freq;
    global tone_mapping;
    global overlap_ratio;
    amp = 1;
    % 将唱名映射至以2^(1/12)为底的指数, 1对应指数为1
    tone_mapping = [0, 2, 4, 5, 7, 9, 11];
    overlap_ratio = 0.01;
    time_step = sample_freq^(-1);
    t = 0:time_step:(len-1) * time_step;

    harmonic = [1,0.2,0.3];
    for m = 1:length(harmonic)
       if m == 1
           wave = amp * sin(2 * pi * freq * t);
       else
           wave = wave + harmonic(m) * amp * sin(2 * pi * m * freq * t);
       end
    end
end
```

启动`exer_1_4.m`，听到的声音更加厚重了。

(5)自选合成一段音乐

选择C418的 *Sweden* 作为合成的音乐，由于存在高音和低音部，且和声较多，合成时按照高声部0.75，低声部0.25的强度合成：

```
clear
close all;
clc

sample_freq = 16e3;
% 1 = D
base_tone_freq = 293.66;
beat_time = 1.25;
amp = 1;
% 将唱名映射至以2^(1/12)为底的指数, 1对应指数为1
tone_mapping = [0, 2, 4, 5, 7, 9, 11];
overlap_ratio = 0.1/0.95;

% 曲谱

bar1 = {
    {
        [-5,    5,      6,      -2,     1,      2,      -4,     3,      5,
 -2];
        [1,     0.5,    0.5,    1.5,    0.25,   0.25,   1.5,    0.25,   0.25,
2]
    };
    {
        [-3,    -100,   1,      -100,   -2,     -100,   0];
        [1,     1,      1.5,    0.5,    1.5,    0.5,    2]
    };
    {
        [-1,    -100,   3,      -100,   0,      -100,   2];
        [1,     1,      1.5,    0.5,    1.5,    0.5,    2]
    };
    {
        [-12,   -11,    -10,    -8,     -9,     -10,    -13];
        [1,     1,      1,      1,      1,      1,      2]
    }
};

bar2 = {
    {
        [-5,    8,      6,      5,      -2,     1,      2,      -4,     5,
 3,     -2];
        [0.5,   0.5,    0.5,    0.5,    1.5,    0.25,   0.25,   1.5,    0.25,
0.25,  2]
    };
    {
        [-3,    -100,       1,      -100        -2,   -100,   0];
        [0.5,   1.5,        1.5,    0.5,        1.5,  0.5,    2]
    };
    {
        [-1,    -100,       3,      -100,       0,    -100,   2];
        [0.5,   1.5,        1.5,    0.5,        1.5,  0.5,    2]
    };
```

```
    {
        [-12,   -11,    -10,    -8,     -9,     -10,    -13];
        [1,     1,      1,      1,      1,      1,      2]
    }
};

% 句柄定义
my_play_multi = @(bar) play_multi(bar, amp, sample_freq, tone_mapping,
overlap_ratio, base_tone_freq, beat_time);

music1 = my_play_multi(bar1);
music2 = my_play_multi(bar2);

music = [music1, music2];

t = 0:1/sample_freq:(length(music) - 1)/sample_freq;

tin = timeseries(music',t);
tout = resample(tin, t);
music = tout.Data;

sound(music, sample_freq);
plot(music);


function result = play_multi(melody, amp, sample_freq, tone_mapping,
overlap_ratio, base_tone_freq, beat_time)
    my_play_single = @(tone, beat) play_single(tone, beat, amp, sample_freq,
tone_mapping, overlap_ratio, base_tone_freq, beat_time);

    result = [];
    music = [];
    len = length(melody);
    for i = 1:len
        if i == 1
            music = my_play_single(melody{i}{1}, melody{i}{2});
        else
            music_current = my_play_single(melody{i}{1}, melody{i}{2});
            music_len = min(length(music), length(music_current));
            music = [music(:, 1:music_len);music_current(1:music_len)];
        end
    end
    [row, ~] = size(music);
    result = [0.25, 0.25, 0.25, 0.25] * music;
end


function result = play_single(tone, beat, amp, sample_freq, tone_mapping,
overlap_ratio, base_tone_freq, beat_time)

    % 初始化空数组用于存储结果
    result = [];
    loop = 1:length(tone);
    overlap_last = 0;
    % 循环迭代
    for i = loop
        % 调用 gen_tune 函数
```

```
        [local_result, overlap] = gen_tune(tone(i), beat(i), amp, sample_freq,
tone_mapping, overlap_ratio, base_tone_freq, beat_time);
        % 将结果的首尾相加，中间拼接
        if i == 1
            % 第一次迭代，直接将结果添加到结果数组
            result = local_result;
        else
            % 非第一次迭代，将上一次结果的末尾与当前结果的开头相加，并将结果添加到结果数组
            result = [result(1:end-overlap_last), (result(end-
overlap_last+1:end) + local_result(1:overlap_last)),
local_result(overlap_last+1:end)];
        end
        overlap_last = overlap;
    end
end

function [result, overlap] = gen_tune(tone, beat, amp, sample_freq,
tone_mapping, overlap_ratio, base_tone_freq, beat_time)

    [freq, width] = trans_freq_width(tone, beat, base_tone_freq, beat_time,
tone_mapping);

    [y0, y1, y2, y3] = generate_fixed(width, sample_freq);

    wave = gen_waveform(freq, length([y0, y1, y2, y3]), amp, sample_freq);

    result = wave .* [y0, y1, y2, y3];
    overlap = round(overlap_ratio * length(y3));
end

function wave = gen_waveform(freq, len, amp, sample_freq)

   time_step = sample_freq^(-1);
   t = 0:time_step:(len-1) * time_step;

   harmonic = [1, 1.46, 0.96, 1.10, 0.05, 0.11, 0.36, 0.12, 0.14, 0.06];
   for m = 1:length(harmonic)
       if m == 1
           wave = amp * sin(2 * pi * freq * t);
       else
           wave = wave + harmonic(m) * amp * sin(2 * pi * m * freq * t);
       end
   end
end
```

启动`exer_1_5.m`，可以听见音乐。

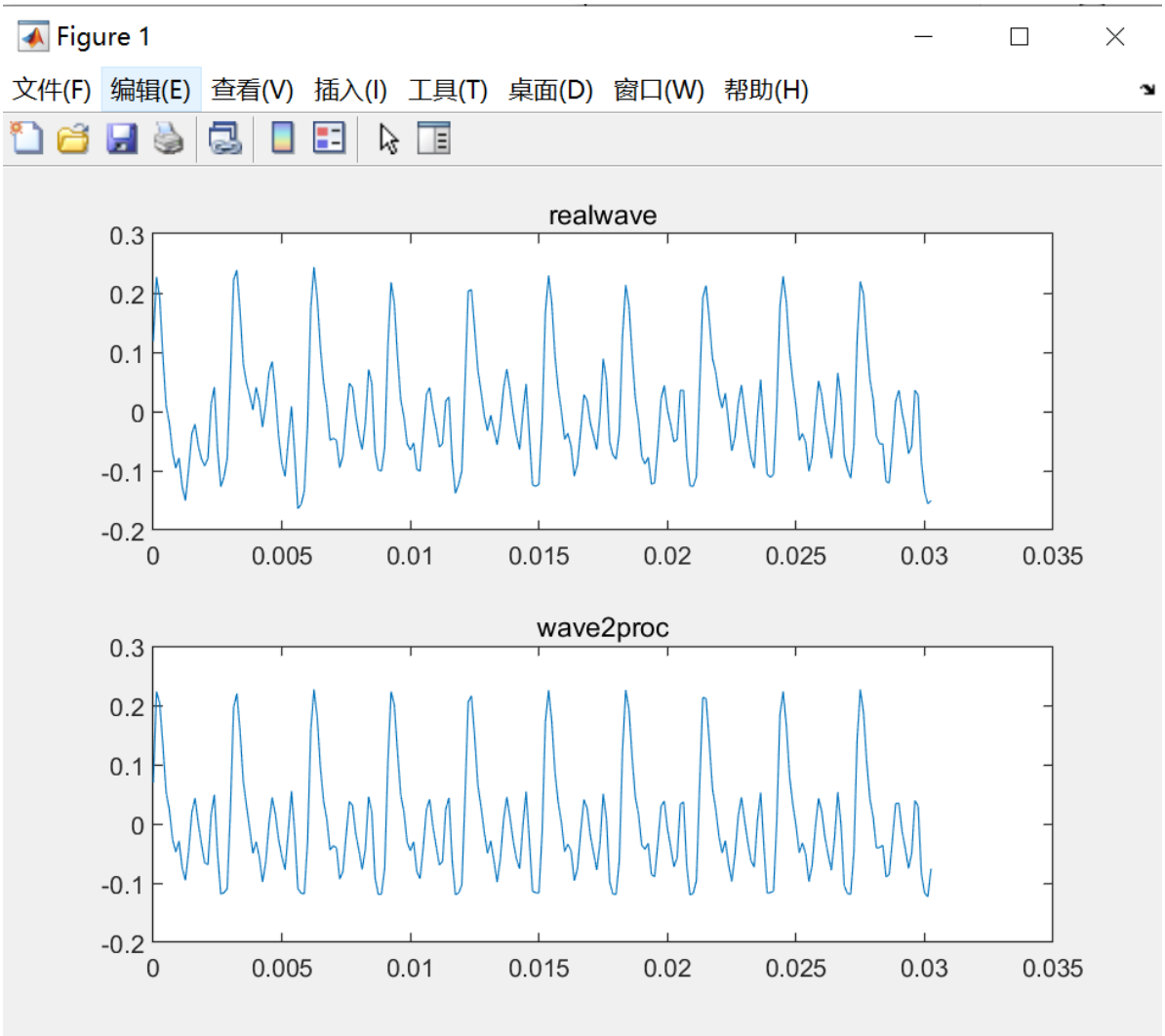
2. 用傅里叶级数分析音乐

(6)听真实的音乐

```
clear;
close all;
clc;
fs = 8000;
% load 命令载入附件光盘中的数据文件“guitar.mat”
load('音乐合成所需资源\Guitar.MAT');
```

```
l = length(realwave);
t = (0:1/fs:((l-1)/fs))';
figure;
subplot(2, 1, 1);
plot(t, realwave);
title("realwave");

subplot(2, 1, 2);
plot(t, wave2proc);
title('wave2proc');


% 先用 wavread 函数载入光盘中的 fmt.wav 文件
[fmt, fs] = audioread('音乐合成所需资源\fmt.wav');
sound(fmt, fs);
```

启动`exer_2_6.m`，得到的真实吉他乐谱波形（`realwave`）和处理后的波形（`wave2proc`）：



可以看到，真实的吉他声波受到很多偶然因素的影响，幅度和波形会不断变化，而处理后的波形更加规则和稳定。

(7)处理吉他波形

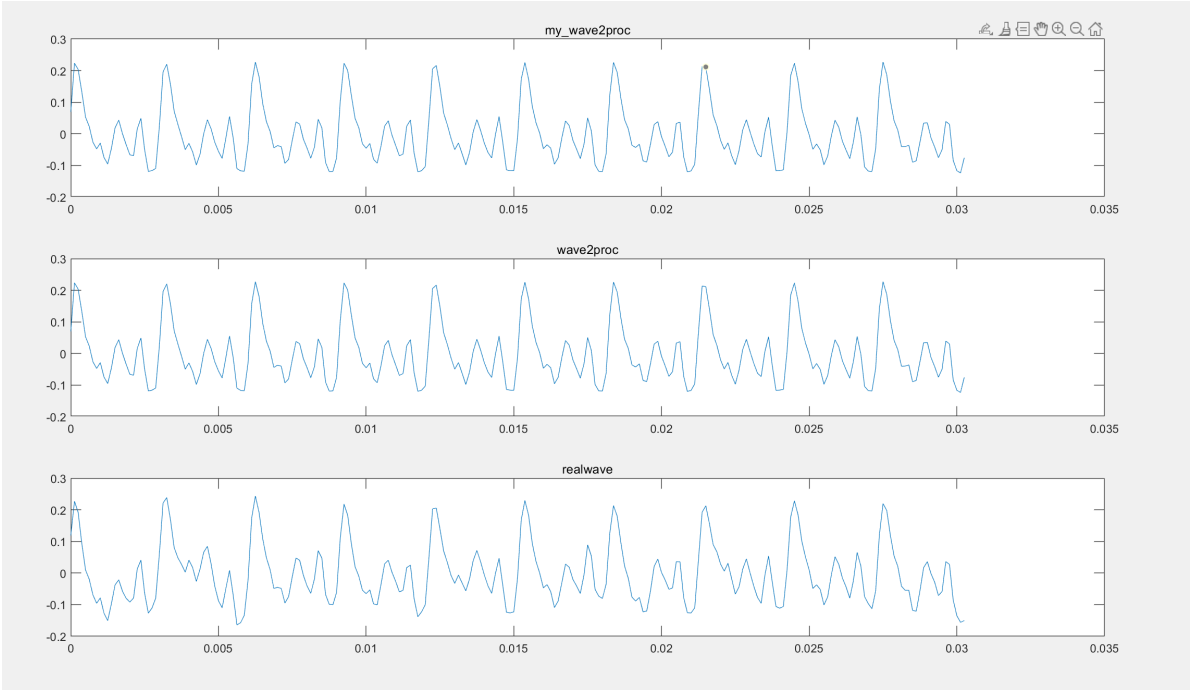
观察到吉他波形里有10个峰，因而判断这段信号中有10个周期。考虑将这10个周期分割叠加求平均值，将求得结果再重复10次：

```
clear;
```

```
close all;
clc;
fs = 8000;
part_div = 10;
load('音乐合成所需资源\Guitar.MAT');
l = length(realwave);
% resample
rsmp = resample(realwave, part_div, 1);
% 10 parts
part_len = round(length(rsmp) / part_div);
% sum all
for i = 1:part_div
    if (i == 1)
        res = rsmp(1:part_len);
    else
        res = res + rsmp((i - 1) * part_len + 1: i * part_len);
    end
end
% take average
res = res / part_div;
% repeat & resample
avg_rsmp = repmat(res, part_div, 1);
rsmp_final = resample(avg_rsmp, 1, part_div);
% plot
figure;
t = (0:1/fs:(l-1)/fs)';
subplot(3, 1, 1);
plot(t, rsmp_final);
subtitle("my\_wave2proc");
subplot(3, 1, 2);
plot(t, wave2proc);
subtitle("wave2proc");
subplot(3, 1, 3);
plot(t, realwave);
subtitle("realwave");
```

为了使得信号刚好能被平均分为10个等长的部分，采用重采样的方式，将采样点变成原来的10倍；在平均完成后，采用重采样恢复到原来的采样率，同时也能保证repmat操作中，周期之间的邻接处具有更好的连续性。

启动`exer_2_7.m`，得到的图像如下：



可以看到，我们处理的`my_wave2proc`几乎和`wave2proc`一样稳定。

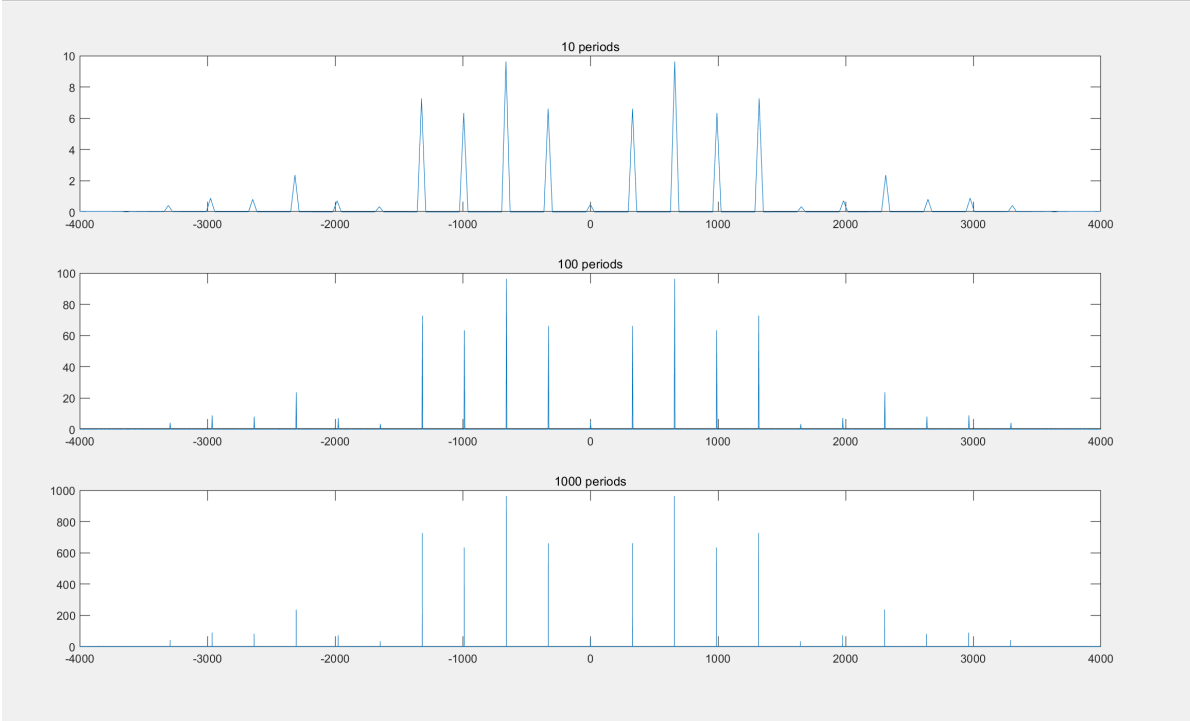
(8)利用傅里叶变换分析频谱

考虑到样本个数可能对于频谱分析而言还不够多，采用时域周期扩展的方法，增加时域的数据量，使得频谱宽度更窄：

```
clear;
close all;
clc;

fs = 8000;
load('音乐合成所需资源\Guitar.MAT');

figure;

subplot(3, 1, 1);
W1 = fft(wave2proc);
W1 = fftshift(W1);
W1 = abs(W1);
freq = linspace(-fs/2, fs/2, length(W1));
plot(freq,W1);
title('10 periods');

subplot(3, 1, 2);
W2 = fft(repmat(wave2proc, 10, 1));
W2 = fftshift(W2);
W2 = abs(W2);
freq = linspace(-fs/2, fs/2, length(W2));
plot(freq, W2);
title('100 periods');

subplot(3, 1, 3);
W3 = fft(repmat(wave2proc, 100, 1));
W3 = fftshift(W3);
W3 = abs(W3);
freq = linspace(-fs/2, fs/2, length(W3));
plot(freq, W3);
```

```
title('1000 periods');

[w, f] = findpeaks(w3, freq);

% 分析得到：
% 基频：329.40Hz,音调：e1。
% 谐波分量有
%
%           2.00, 3.00, 4.00, 5.00, 6.00, 7.00, 8.00, 9.00, 10.00
% 幅度分别为
%           1.46, 0.96, 1.10, 0.05, 0.11, 0.36, 0.12, 0.14, 0.06
```

启动`exer_2_8.m`，得到的图像如下：



三张图分别是时域波形有10个，100个，1000个周期的频谱。可以看到，随着周期数的增加，频谱宽度越来越窄，1000个周期时频谱已经近似为一条条线。

出现这种现象的原因分析：简单而言，就是"时域的扩张带来频域的收缩"。

具体而言，其实我们可以把原信号看成是一段无限长的周期信号与一个窗函数的乘积，换句话说，是用矩形窗函数对它进行采样：

$$f(t) = f_{\text{period}}(t) \cdot p(t)$$
$$p(t) = (u(t) - u(t-T))$$

因而它在频域的傅里叶变换为此二者的卷积：

$$F(\omega) = F_{\text{period}}(\omega) * P(\omega)$$

$P(\omega)$是矩形窗的傅里叶变换，它自然是一个sinc函数，而$f_{\text{period}}(t)$是一个周期函数，其频谱由一组冲激函数的和构成。这一组冲激函数与sinc函数卷积，那么得到的频谱形状一定是重复出现的sinc函数，每个峰的强度取决于$F_{\text{period}}(\omega)$。

现在我们重复时域信号，实际上可以认为是在扩展矩形窗$p(t)$的宽度，随着时域脉冲宽度的扩张，我们知道频域sinc函数的宽度会收缩，极限情况下，$p(t)$无限宽时，$f(t)$实际上就是无限长的周期信号，从而它在极限情况下的图像是一组冲激函数的和。

实际上，matlab处理的信号是离散时间信号，但是由离散信号和连续信号分析方法的类比，我们不难得到离散情形下的类似解释。

从图像上求得的基频音调以及谐波分量如下：

```
% 分析得到：
% 基频：329.40Hz,音调：e1。
% 谐波分量有
%
%           2.00, 3.00, 4.00, 5.00, 6.00, 7.00, 8.00, 9.00, 10.00
% 幅度分别为
%           1.46, 0.96, 1.10, 0.05, 0.11, 0.36, 0.12, 0.14, 0.06
```

(9)自动分析音调和节拍

本小节的难度相对大一点。

首先是分析节拍。

在参考了matlab文档以及2021年matlab大作业《B站，我来了》相关资料，采用的处理方式如下：

第一步：时域平方求能量。

```
[fmt, fs] = audioread('音乐合成所需资源\fmt.wav');
plot_wave = @(wave) plot(plot_wave_t(wave, fs), wave);
% sound(fmt, fs);
raw = fmt .^ 2;
```

这一步求得了信号对应的能量，也去除了幅度的负数部分。

第二步：多次求峰值包络

```
[env, ~] = envelope(raw, 5,'peak');
for i = 1:2
    [env, ~] = envelope(env, 90,'peak');
end
```

这一步我采用matlab提供的envelope函数求能量信号的包络，并且求多次，经过实验，采用这一组参数求得的包络毛刺较少，并且可以较完整的反映能量信号的涨落。

第三步：根据包络划分节拍

由于每一个节拍通常都有一个强度峰值，因此只需要找到峰值，就不难划分节拍。我采用下面的标准确定峰值点：

- 幅度阈值条件：这个峰值与在它之前，且离它最近的那个极小值之间的比值超过一个阈值；
- 搜索区间条件：这个峰值的邻近处没有比它更大的峰值。

在划分峰值后，我还划分了节拍的起止点。如果一个点满足下面的三个条件之一，它会被定义为起止点：

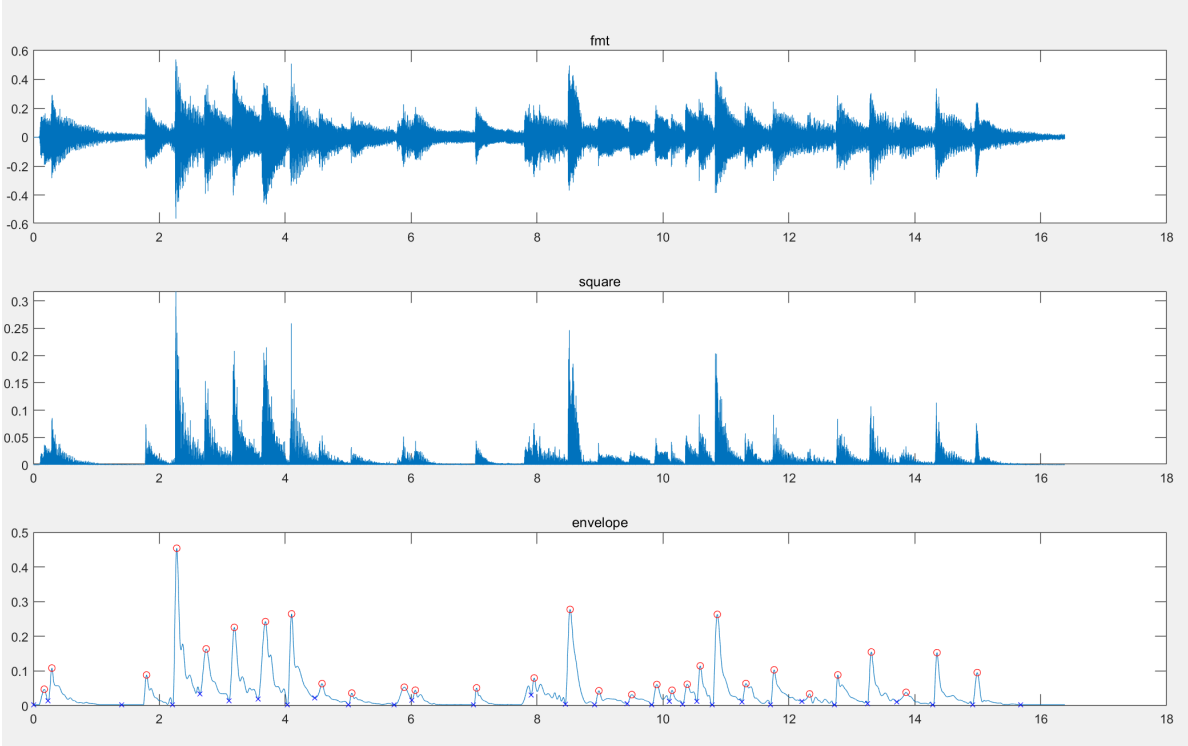
- 它是峰值前最近的那个极小值点；
- 它是第一个峰值之前，离第一个峰最近的最小值点；
- 它是最后一个峰值之后，离最后一个峰最近的最小值点。

节拍划分代码如下：

```
function [peak_x, prev_valley_x] = my_find_peak(y, threshold_ratio,
threshold_interval, fs)
    % 被确定为峰值的条件：该极大值必须与前一个极小值的比值大于 threshold_ratio，或者它前面没
有极小值；距离该极大值点
    % threshold_interval 范围内没有比它更大的极大值。

    % config
    debug = false;

    peak_x = [];
    prev_valley_x = [];
    % 计算极大值和极小值
    maxima = islocalmax(y);
    minima = islocalmin(y);
    maxima_x = find(maxima);
    if debug
        fprintf("ratio = %f, interval = %f\n", threshold_ratio,
threshold_interval);
        disp(['maxi num: ', num2str(length(maxima_x))]);
    end

    % 添加第一个最低点
    [~, min_idx] = min(y(1:maxima_x(1)));
    prev_valley_x = cat_element(prev_valley_x, min_idx(end));

    % 遍历极大值点
    for i = 1:length(maxima_x)
        maxima_xi = maxima_x(i);
        % 找到在当前极大值点之前且离它最近的第一个极小值
        prev_min_index = find(minima(1:maxima_xi-1), 1, 'last');
        % 区间内峰值最大值
        % 在区间内找到极大值
        [peaks, peak_locations] = findpeaks(y(max(maxima_xi - threshold_interval
+ 1, 1):maxima_xi + threshold_interval));

        % 找到极大值最大的点
        [interval_max_y, max_peak_index] = max(peaks);
        interval_max_x = peak_locations(max_peak_index) + maxima_xi -
threshold_interval;

        % 判断比值和间隔是否满足阈值条件
        if ~isempty(prev_min_index)
            ratio = y(maxima_xi) / y(prev_min_index);
            if (ratio > threshold_ratio)
                if y(maxima_xi) >= interval_max_y
                    peak_x = cat_element(peak_x, maxima_xi);
                    prev_valley_x = cat_element(prev_valley_x, prev_min_index);
                elseif ratio > threshold_ratio
                    if debug
                        fprintf("x = %f, y = %f, prev_min_x = %f, prev_min_y =
%f, ratio = %f, interval_max_x = %f,interval_max_y = %f, interval = %f\n",
maxima_xi/fs, y(maxima_xi), prev_min_index/fs, y(prev_min_index), ratio,
interval_max_x / fs, interval_max_y, abs(interval_max_x - maxima_xi) / fs);
                    end
                end
            else

            end
```

```
        else
            % 没有极小值，这是第一个极大值
            if y(maxima_xi) >= interval_max_y
                peak_x = cat_element(peak_x, maxima_xi);
            else
                if debug
                    fprintf("no prev_min: x = %f, y = %f, interval_max_x =
%f,interval_max_y = %f, interval = %f\n", maxima_xi/fs, y(maxima_xi),
interval_max_x / fs, interval_max_y, abs(interval_max_x - maxima_xi) / fs);
                end
            end
        end
    end
    % 再加上末尾的最低点
    [~, min_idx] = min(y(maxima_x(end):end));
    min_idx = maxima_x(end) - 1 + min_idx;
    prev_valley_x = cat_element(prev_valley_x, min_idx(1));
end

function y = cat_element(list, x)
    if isempty(list)
        y = x;
    else
        y = [list, x];
    end
end
```

节拍划分结果如图，第一张图是读取的时域波形；第二张图为能量时域波形；第三张图为峰值包络图，以及划分的节拍结果。其中红圈标示了每个节拍的峰值，蓝叉标示了每个节拍之间的边界点。一共划分了30个节拍。



其次是分析音调。

按照节拍的起止点，将原音乐的时域波形划分为30个部分，每个部分重复512次，通过傅里叶变换进行频谱分析。

```
t = plot_wave_t(fmt, fs);
        tone = fmt(valleys(i):valleys(i + 1));
        tone = tone(1:round(length(tone) * 0.8));
        tone = tone .* gausswin(length(tone));
        for j = 1:5
            tone = [tone;tone];
        end
        % 频域变换（只取正值）
        Tone = abs(fftshift(fft(tone)));
        f = linspace(-fs/2, fs/2, length(Tone));
        Tone = Tone(f > 0);
        f = f(f > 0);
```

第一步，分析基频。

采用下面的标准确定基频：

- 幅度阈值条件：基频的强度至少是频谱中最大峰强度的0.3倍；
- 搜索区间条件：最大频率的频率值与基频的频率值之比距离最近的整数不超过0.025，并且这个整数应当尽可能大。

在满足上述标准的前提下，选择频率之比最接近整数的一个频率值作为基频值。

```
% 求峰值
[max_amp, max_idx] = max(Tone);
[peak_freq_amp, peak_freq_index] = findpeaks(Tone);
ths = 0.3;
peak_filtered_index = peak_freq_index(peak_freq_amp / max_amp > ths);

% 暴力搜索
search_harmonic_idx = peak_filtered_index(peak_filtered_index <= max_idx);
base_freq_idx = max_idx;
search_valid = false;
ths2 = 0.025;
search_result = search_harmonic_idx(abs(round(max_idx ./ search_harmonic_idx) -
max_idx ./ search_harmonic_idx) < ths2);
if (~isempty(search_result))
    max_int = max(round(max_idx ./ search_result));
    search_result = search_result(abs(max_int - max_idx ./ search_result) <
ths2);
    [~, min_idx] = min(abs(round(log(f(search_result) / 220) * 12 / log(2)) -
log(f(search_result) / 220) * 12 / log(2)));
    final_result = search_result(min_idx);
    base_freq_idx = min(final_result);
    search_valid = true;
end

base_freq = f(base_freq_idx);
[tone_name, standard_freq] = get_tune_name(base_freq);
```

第二步，分析谐波。

采用下面的标准确定谐波：

- 幅度阈值条件：谐波的强度至少是基频强度的0.01倍；
- 搜索区间条件：n次谐波频率与基频的频率之比处于区间$[0.9n, 1.1n]$之间，且是这个区间内的最大值。

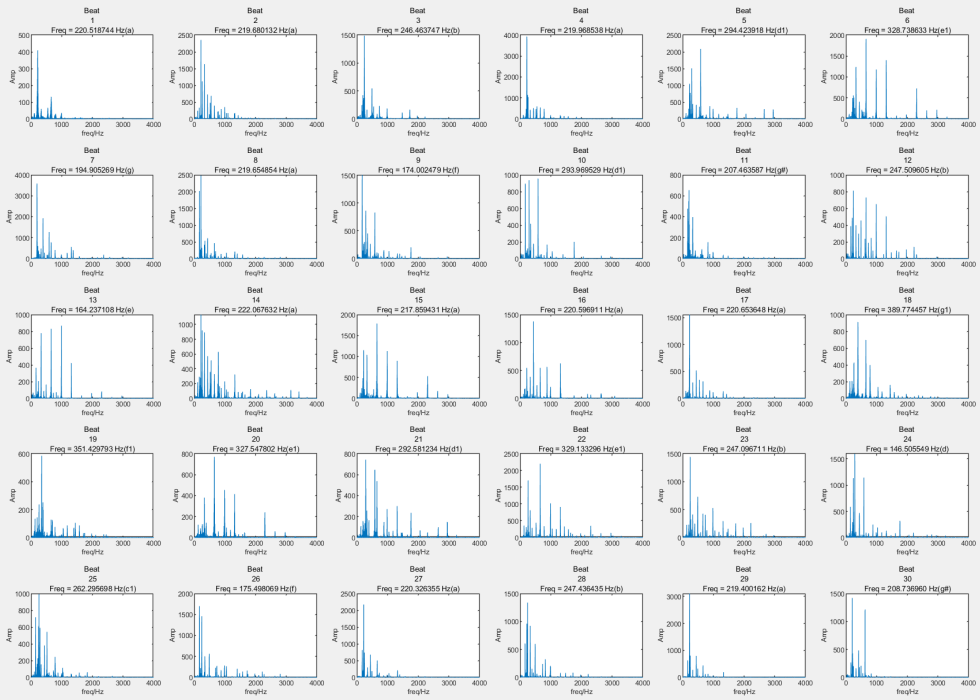
确定的谐波会根据基频强度进行归一化。

```
% 检测谐波强度（正负0.1附近）
boundary = 0.1;
harmonic_idxs = [];
harmonic_amps = [];
harmonic_mults = [];
save_harmonic_amps = [];

for j = 2:9
    harmonic_freq_idx = j * base_freq_idx;
    start = round(harmonic_freq_idx * (1 - boundary));
    stop = min(round(harmonic_freq_idx * (1 + boundary)),
length(peak_freq_amp));
    search_harmonic_idx = Tone(start:stop);
    [max_harmonic_amp, max_harmonic_idx] = max(search_harmonic_idx);
    max_harmonic_idx = start - 1 + max_harmonic_idx;
    % 归一化
    max_harmonic_amp = max_harmonic_amp / Tone(base_freq_idx);
    if (max_harmonic_amp > 0.01)
        harmonic_amps = [harmonic_amps, max_harmonic_amp];
        harmonic_idxs = [harmonic_idxs, max_harmonic_idx];
        harmonic_mults = [harmonic_mults, j];
        save_harmonic_amps = [save_harmonic_amps, max_harmonic_amp];
    else
        save_harmonic_amps = [save_harmonic_amps, 0];
    end
end
```

我写了函数`get_tune_name`，用于分析频率对应的音调：

```
function [tone, standard_f] = get_tune_name(base_f)
    % 定义音调值与音名的对应关系
    % 大字组音名用大写字母表示，小字组用小写字母表示，组号用数字表示，例如'a'表示小字组的A（按
照教材写法为A0），'B1'表示大字一组的B。
    % 一共收录132个音名。钢琴一般只有88个键，更低的音调人耳可能很难听到。
    tone_map = containers.Map({8.176, 8.662, 9.177, 9.723, 10.301, 10.913,
11.562, 12.250, 12.978, 13.750, 14.568, 15.434, ...
        16.352, 17.324, 18.354, 19.445, 20.602, 21.827, 23.125, 24.500, 25.957,
27.500, 29.135, 30.868, 32.703, ...
        34.648, 36.708, 38.891, 41.203, 43.654, 46.249, 48.999, 51.913, 55.000,
58.270, 61.735, 65.406, 69.296, ...
        73.416, 77.782, 82.407, 87.307, 92.499, 97.999, 103.826, 110.000,
116.541, 123.471, 130.813, 138.591, ...
        146.832, 155.563, 164.814, 174.614, 184.997, 195.998, 207.652, 220.000,
233.082, 246.942, 261.626, ...
        277.183, 293.665, 311.127, 329.628, 349.228, 369.994, 391.995, 415.305,
440.000, 466.164, 493.883, ...
        523.251, 554.365, 587.330, 622.254, 659.255, 698.456, 739.989, 783.991,
830.609, 880.000, 932.328, ...
        987.767, 1046.502, 1108.731, 1174.659, 1244.508, 1318.510, 1396.913,
1479.978, 1567.982, 1661.219, ...
        1760.000, 1864.655, 1975.533, 2093.005, 2217.461, 2349.318, 2489.016,
2637.020, 2793.826, 2959.955, ...
```

```
        3135.963, 3322.438, 3520.000, 3729.310, 3951.066, 4186.009, 4434.922,
4698.636, 4978.032, 5274.041, ...
        5587.652, 5919.911, 6271.927, 6644.875, 7040.000, 7458.620, 7902.133,
8372.018, 8869.844, 9397.273, ...
        9956.063, 10548.082, 11175.303, 11839.822, 12543.854, 13289.750,
14080.000, 14917.240, 15804.266}, ...
        {'C3', 'C3#', 'D3', 'D3#', 'E3', 'F3', 'F3#', 'G3', 'G3#', 'A3', 'A3#',
'B3', 'C2', 'C2#', 'D2', 'D2#', ...
        'E2', 'F2', 'F2#', 'G2', 'G2#', 'A2', 'A2#', 'B2', 'C1', 'C1#', 'D1',
'D1#', 'E1', 'F1', 'F1#', 'G1', ...
        'G1#', 'A1', 'A1#', 'B1', 'C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G',
'G#', 'A', 'A#', 'B', 'c', 'c#', 'd', ...
        'd#', 'e', 'f', 'f#', 'g', 'g#', 'a', 'a#', 'b', 'c1', 'c1#', 'd1',
'd1#', 'e1', 'f1', 'f1#', 'g1', 'g1#', ...
        'a1', 'a1#', 'b1', 'c2', 'c2#', 'd2', 'd2#', 'e2', 'f2', 'f2#', 'g2',
'g2#', 'a2', 'a2#', 'b2', 'c3', 'c3#', ...
        'd3', 'd3#', 'e3', 'f3', 'f3#', 'g3', 'g3#', 'a3', 'a3#', 'b3', 'c4',
'c4#', 'd4', 'd4#', 'e4', 'f4', 'f4#', ...
        'g4', 'g4#', 'a4', 'a4#', 'b4', 'c5', 'c5#', 'd5', 'd5#', 'e5', 'f5',
'f5#', 'g5', 'g5#', 'a5', 'a5#', 'b5', ...
        'c6', 'c6#', 'd6', 'd6#', 'e6', 'f6', 'f6#', 'g6', 'g6#', 'a6', 'a6#',
'b6'});
    % 判断基频值与音调值的差距，并输出对应的音名
    tone = '';
    tolerance = 0.02;
    for freq = keys(tone_map)
        if abs(base_f/freq{1} - 1) < tolerance
            tone = tone_map(freq{1});
            standard_f = freq{1};
            break;
        end
    end
end
```



启动`exer_2_9.m`，30个节拍对应的频谱，基频频率和音调如图所示。在控制台可以看见谐波分析的结果。

```
节拍     25 的基频为   262.295698 Hz, 对应的音名为    c1   (频率    261.626000 Hz)
* 含有2倍的谐波分量(522.1478 Hz)，幅度（相对基频）为0.54767
* 含有3倍的谐波分量(784.4059 Hz)，幅度（相对基频）为0.24545
* 含有4倍的谐波分量(1027.4157 Hz)，幅度（相对基频）为0.11282
* 含有5倍的谐波分量(1306.516 Hz)，幅度（相对基频）为0.043658
节拍     26 的基频为   175.498069 Hz, 对应的音名为    f    (频率    174.614000 Hz)
* 含有2倍的谐波分量(348.588 Hz)，幅度（相对基频）为0.29478
* 含有3倍的谐波分量(493.2248 Hz)，幅度（相对基频）为0.32951
* 含有4倍的谐波分量(739.8186 Hz)，幅度（相对基频）为0.16026
* 含有5倍的谐波分量(874.9711 Hz)，幅度（相对基频）为0.1008
* 含有6倍的谐波分量(986.4125 Hz)，幅度（相对基频）为0.16518
* 含有7倍的谐波分量(1233.0064 Hz)，幅度（相对基频）为0.01704
节拍     27 的基频为   220.326355 Hz, 对应的音名为    a    (频率    220.000000 Hz)
* 含有2倍的谐波分量(440.6107 Hz)，幅度（相对基频）为0.34829
* 含有3倍的谐波分量(663.5815 Hz)，幅度（相对基频）为0.2585
* 含有4倍的谐波分量(873.1203 Hz)，幅度（相对基频）为0.029101
* 含有5倍的谐波分量(1047.736 Hz)，幅度（相对基频）为0.063416
* 含有6倍的谐波分量(1324.4347 Hz)，幅度（相对基频）为0.10755
节拍     28 的基频为   247.436435 Hz, 对应的音名为    b    (频率    246.942000 Hz)
* 含有2倍的谐波分量(492.6493 Hz)，幅度（相对基频）为0.44702
* 含有3倍的谐波分量(737.8621 Hz)，幅度（相对基频）为0.17879
* 含有4倍的谐波分量(985.2643 Hz)，幅度（相对基频）为0.15262
* 含有5倍的谐波分量(1322.4319 Hz)，幅度（相对基频）为0.058599
节拍     29 的基频为   219.400162 Hz, 对应的音名为    a    (频率    220.000000 Hz)
* 含有2倍的谐波分量(440.7457 Hz)，幅度（相对基频）为0.84526
* 含有3倍的谐波分量(662.0913 Hz)，幅度（相对基频）为0.46685
* 含有4倍的谐波分量(826.1242 Hz)，幅度（相对基频）为0.13681
* 含有5倍的谐波分量(992.1334 Hz)，幅度（相对基频）为0.0643
节拍     30 的基频为   208.736960 Hz, 对应的音名为    g#   (频率    207.652000 Hz)
* 含有2倍的谐波分量(415.8048 Hz)，幅度（相对基频）为0.39088
* 含有3倍的谐波分量(624.5161 Hz)，幅度（相对基频）为0.98736
* 含有4倍的谐波分量(826.6538 Hz)，幅度（相对基频）为0.055446
* 含有5倍的谐波分量(1041.9387 Hz)，幅度（相对基频）为0.013571
* 含有6倍的谐波分量(1250.65 Hz)，幅度（相对基频）为0.015506
fx >>
```

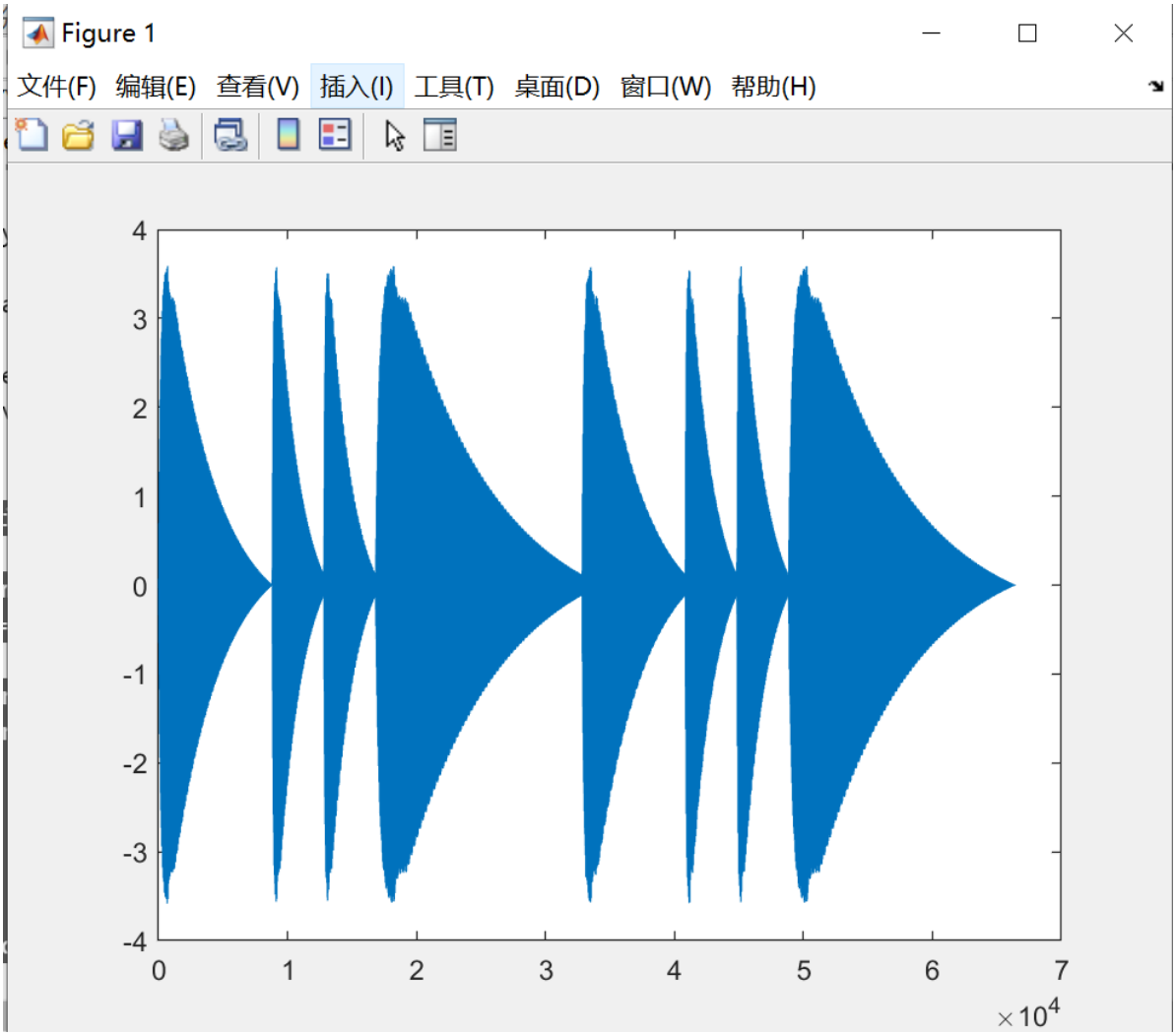
3. 基于傅里叶级数的合成音乐

(10)用谐波合成吉他音色

改写`gen_waveform`，用第(8)题分析的谐波分量再次完成第(4)题：

```
function wave = gen_waveform(freq, len, amp, sample_freq)

   time_step = sample_freq^(-1);
   t = 0:time_step:(len-1) * time_step;

   harmonic = [1, 1.46, 0.96, 1.10, 0.05, 0.11, 0.36, 0.12, 0.14, 0.06];
   for m = 1:length(harmonic)
       if m == 1
           wave = amp * sin(2 * pi * freq * t);
       else
           wave = wave + harmonic(m) * amp * sin(2 * pi * m * freq * t);
       end
   end
end
```

启动`exer_3_10.m`，可以听见东方红乐曲，其音色确实更像吉他了，但是音色仍然有些单调。



(11)为每个音调提供独特的谐波

我将`exer_2_9.m`的谐波分析结果保存在`harmonic.mat`中。对于谐波分析中的相同音调，我将其求和取平均值作为这个音调的谐波分量，一共得到了12个不同音调的谐波。

接下来修改`gen_waveform`：

```
function wave = gen_waveform(freq, len, amp, sample_freq, base_freqs,
harmonic_amps)

    time_step = sample_freq^(-1);
    t = 0:time_step:(len-1) * time_step;
    % 吉他泛音
    harmonic = get_harmonics(freq, base_freqs, harmonic_amps);
    disp(['play tone ', num2str(freq) , ' Hz(', get_tune_name(freq), ') with
harmonics ', num2str(harmonic)])

    for m = 1:length(harmonic)
        if m == 1
            wave = amp * sin(2 * pi * freq * t);
        else
            wave = wave + harmonic(m) * amp * sin(2 * pi * m * freq * t);
        end
    end
end
```

其中使用`get_harmonics`函数来获取谐波分量：

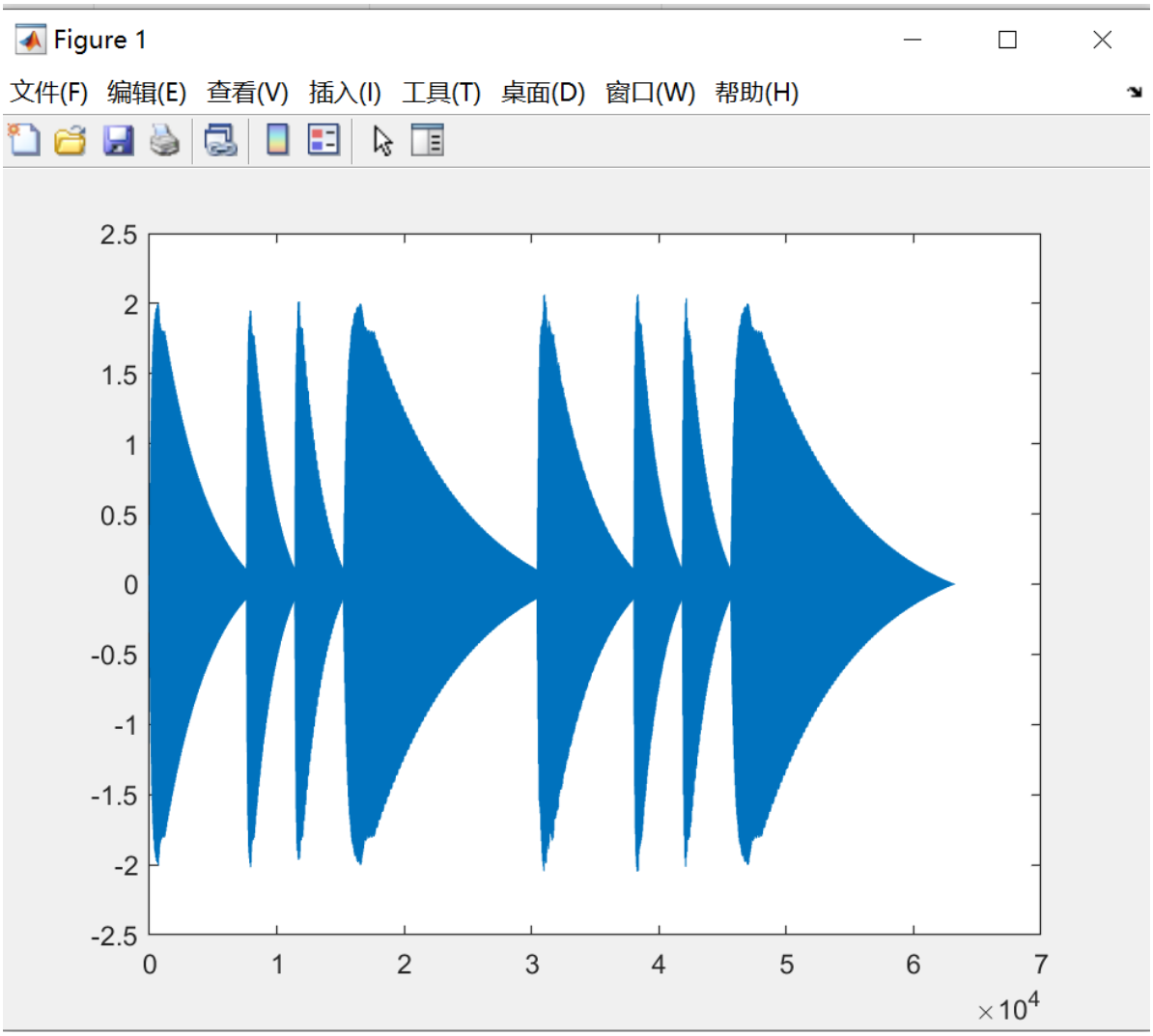
```
function harmonics = get_harmonics(base_freq, base_freqs, harmonic_amps)
    [~, min_idx] = min(abs(freq_mod12_convert(base_freqs) -
freq_mod12_convert(base_freq)));
    harmonics = [1, harmonic_amps{min_idx}];
end
```

这里将频率转换为最近的离它最近的标准音的频率，并取这个标准音的分量。衡量频率间距离的函数如下：

```
function tone = freq_mod12_convert(freqs)
% 将频率映射到模12的空间内。
    tone = mod(log(freqs) * 12 / log(2), 12);
end
```

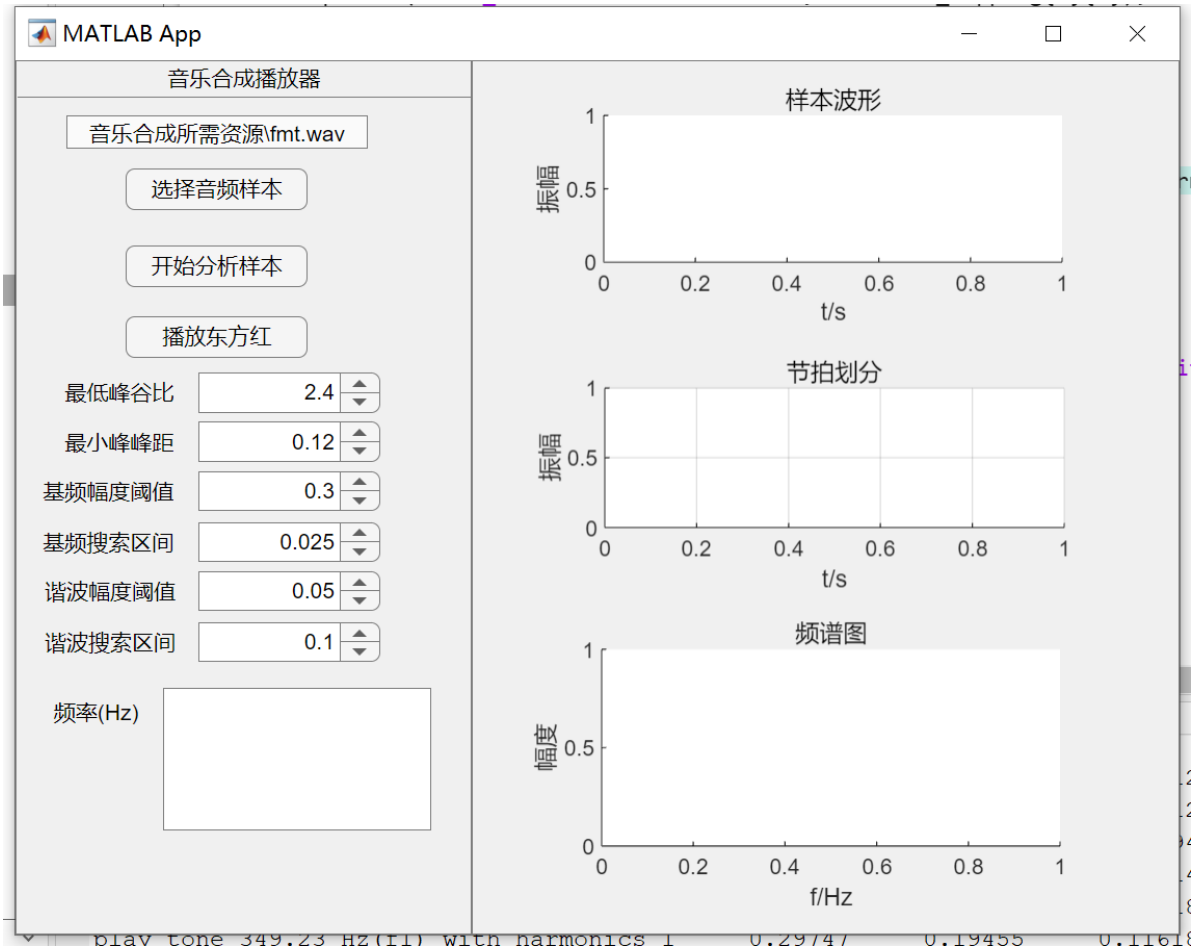
其中，音程上相差八度（即钢琴的12个键，2倍频率）的频率被映射到同一个数上，从而保证十二平均律中"声音频率翻倍，人耳的听感相同"这个条件。

启动`exer_3_11.m`，可以听到东方红乐曲，其音色不仅更像吉他了，而且每个乐符的音色有着微妙的不同。



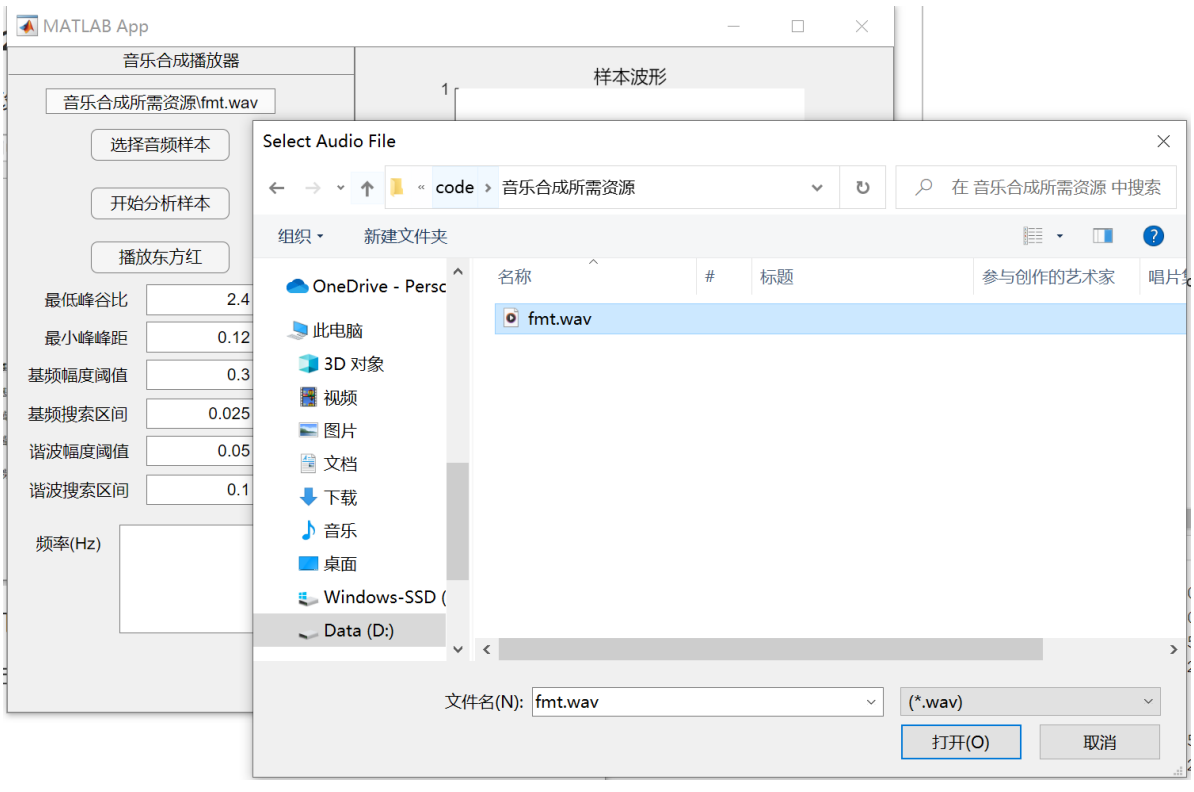
(12)封装为图形界面

我设计了一个Matlab APP，其界面布局如下：

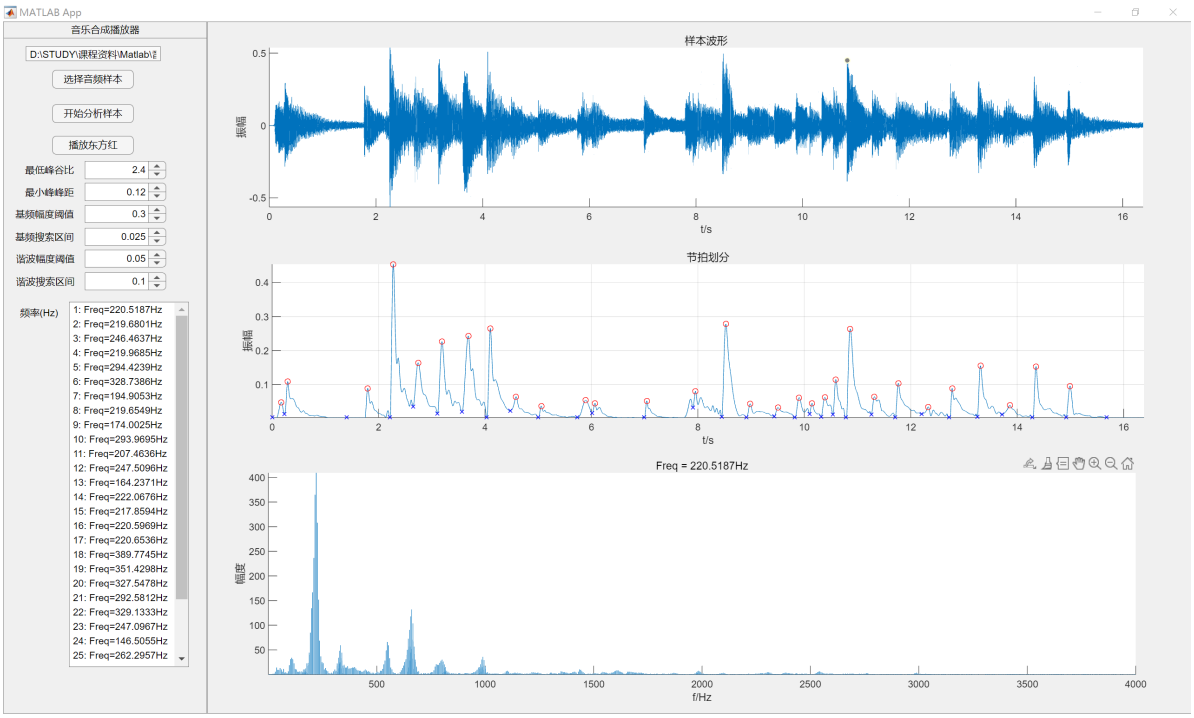


以下展示功能：

点击`选择音频样本`，可以选择需要采集的样本文件：



选择文件后，按钮上方的文本框会显示当前选择的文件路径。按下`开始分析样本`，得到样本的节拍，音调以及频谱图：

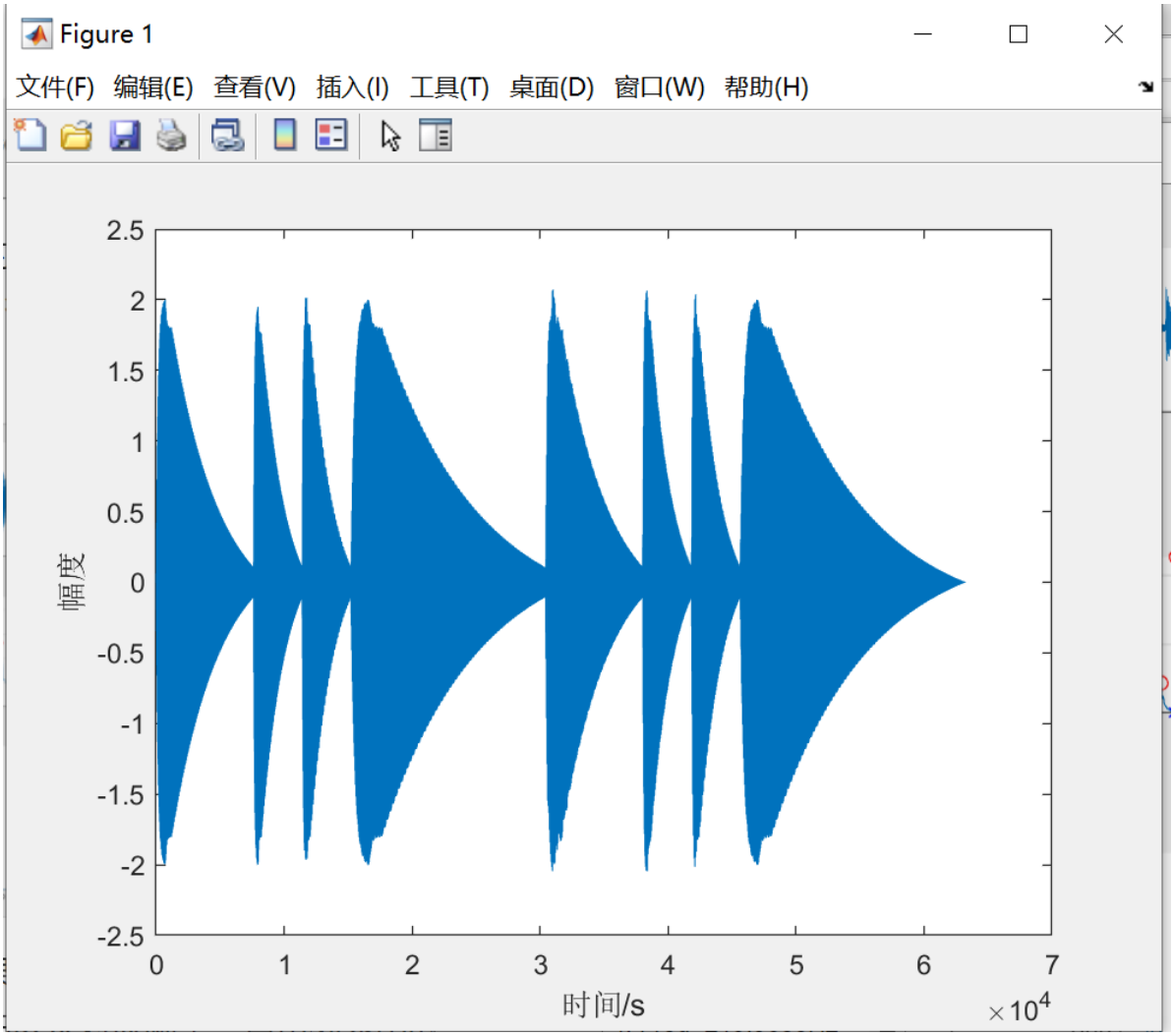


会在目录下生成分析结果（`harmonics.m`）。左下角的列表是分析得到的所有节拍以及对应频率。点击列表中的某一项，可以查看对应节拍的频谱图。

控制台会显示谐波分析的结果。

```
tone_name a 's harmonics:     0.6210    0.6267    0.2052    0.2762    0.2623    0.0081         0         0
tone_name b 's harmonics:     0.4778    0.2155    0.3644    0.2211         0         0         0         0
tone_name d1 's harmonics:     1.0921    0.2100    0.0613    0.1116         0         0         0         0
tone_name e1 's harmonics:     2.0868    1.1311    0.7370         0         0         0         0         0
tone_name g 's harmonics:     0.5378    0.3535    0.1147    0.0552    0.0770    0.1558         0         0
tone_name f 's harmonics:     0.2975    0.1946    0.1162    0.1006    0.1268    0.0308    0.0308         0
tone_name g# 's harmonics:     0.2551    0.5620    0.1471    0.0474    0.0270    0.0270         0         0
tone_name e 's harmonics:     2.5418    0.5382    2.7130    0.2394    2.8308         0    1.3766         0
tone_name g1 's harmonics:     0.6476    0.1411         0         0         0         0         0         0
tone_name f1 's harmonics:     0.2193    0.1147         0         0         0         0         0         0
tone_name d 's harmonics:     2.7238    0.7987    1.9491    0.1401    0.4195    0.3256    0.1694    0.2203
tone_name c1 's harmonics:     0.5477    0.2455    0.1128         0         0         0         0         0
```

点击`播放东方红`，可以播放东方红乐曲，其谐波分量由样本分析确定。播放过程中，控制台会显示每个音符的频率、音名以及谐波分量。



此APP还具有参数调节功能，通过调整参数到合理的值，使用者可以让节拍划分、基频分析、谐波分析的结果更加符合实际。

这六个参数可以划分为三组，每组两个参数，分别与第(9)题中的节拍划分、基频分析、谐波分析三步中的幅度阈值条件、搜索区间条件对应。

例如"最低峰谷比"对应的是节拍划分的幅度阈值条件：

> 由于每一个节拍通常都有一个强度峰值，因此只需要找到峰值，就不难划分节拍。我采用下面的标准确定峰值点：
>
> - 幅度阈值条件：这个峰值与在它之前，且离它最近的那个极小值之间的比值超过一个**阈值**；
> - 搜索区间条件：这个峰值的邻近处没有比它更大的峰值。

因此，这个阈值越小，峰值就更容易被判定为节拍的最大峰值，节拍数会增加。

而"最小峰峰距"对应的是节拍划分的搜索区间条件，这个区间越小，则搜索到节拍峰值的可能性就会更大。

其他的参数，也可以按照类似的方法进行分析。

这6个参数的默认值已经给出，使用者可以通过微调获得更好的样本分析效果。

四、实验小结

通过本次实验，我们实践了使用 MATLAB 工具进行音乐合成的过程。我学习了如何处理信号，在时域和频域进行分析，并应用傅里叶级数等方法合成音乐。这个实验不仅加深了我们对信号处理和傅里叶级数的理解，还提升了我在音乐合成方面的技能。

音乐合成是一件非常有趣的事情，通过音乐合成，我们可以创造出各种各样的音乐作品，表达情感、传递信息，甚至创造全新的音乐风格。音乐合成涉及到信号处理、音乐理论和创作技巧等多个领域的知识和技能。通过使用工具如 MATLAB，我们可以对音频信号进行时域和频域的处理，调整音色和音量等参数，以及创建各种音乐效果和特殊效果。

本次实验中最为困难的莫过于第9题。但是，在反复的调参、理论分析和直观思考中，我尝试了从工科和艺术的双重视角多方面地审视音乐艺术，在富有趣味的同时也深化了我对信号理论中诸如`fft`，采样定理等知识的理解，学会了拓展时域宽度求波形、周期叠加取平均等信号处理的技巧等等，取得了很多收获。感谢为Matlab高级编程课教学付出努力的谷老师和各位助教！

在今后的学习和研究中，我们可以进一步探索和应用更多的信号处理技术和音乐合成方法，以创作出更丰富和多样化的音乐作品。同时，我们也可以将所学的知识和技能应用到其他领域，如语音处理、图像处理等，为更广泛的应用场景提供解决方案。

五、参考文献

[1] MATLAB Documentation - MathWorks 中国

[2] 谷源涛.2021 春信号与系统大作业之“B 站，我来了！”.