MATLAB 音乐合成

姓名： 陈相宁

班级： 无 53

学号： 2015011033

非原创等级声明：

作业 1：Note2frequency 函数参考了学长报告
作业 2：包络函数参数部分与同学讨论得到
作业 3：独立完成。
作业 4：独立完成
作业 5：钢琴曲的音符、节奏参考了学长报告
作业 6：独立完成
作业 7：独立完成
作业 8：独立完成
作业 9：独立完成
作业 10：fft_plot 函数参考了学长实验报告
作业 11：独立完成。

一、简单的合成音乐

作业 1：

1） 计算乐音频率：

完成 Note2Frequency 函数计算简谱对应的频率。

```
Note2Frequency(key,note,adjust)

input:
key: type: char, i.e. 'A','B'...;
note: type: int;
adjust: type: int, default:0, 1 means '#', -1 means 'b'.

output;
f: type: double, frequency of matched Note.
```

设计思路：

除了（3，4），（7，8）之间为$2^{\frac{1}{12}}$比例之外，其余相邻音符之间皆为$2^{\frac{2}{12}}$比例，对于某个音调，首先得到 Do 的基频，再依次递推即可。

注：0 代表低音 7，-1 代表低音 6，8 代表高音 1，9 代表高音 2，以此类推。-inf 表示停顿。

代码如下：

```
function f = Note2Frequency(key,note,adjust)
% input:
% key: type: char, i.e. 'A','B'...;
% note: type: int;
% adjust: type: int, default:0, 1 means '#', -1 means 'b'.
%
% output;
% f: type: double, frequency of matched Note.

% parameter checkout
validateattributes(key,{'char'},{'nonempty'});
validateattributes(note,{'numeric'},{'nonempty'});
if nargin == 2
    adjust = 0;
end

if note < 1 || note > 7         %deal with low and high pitched
    f = 2^floor((note-1)/7) * Note2Frequency(key,mod(note - 1,7) + 1,adjust);
    return
end

list = 2.^((0:11)/12)*220;      %generate A,bB,B,C,bD,D,bE,E,F,bG,G,bA
match = containers.Map({'A','B','C','D','E','F','G'},{1,3,4,6,8,9,11});
Do_Frequency = list(match(key) + adjust);

if note <= 3                    % Do Re Mi
    f = Do_Frequency * 2^(2 * (note - 1) / 12);
else                            % Fa So La Si
    f = Do_Frequency * 2^((2 * (note - 1) - 1) / 12);
end
end
```

2） 合成《东方红》片段：

包装 Generate_Song1 函数合成乐曲。

```
Generate_Song1(song,bpm,SampleFrequency,wave,key,adjust)

input:
song: type: n*2 matrix, the first colume represents the note('rest'
meansrest in music, while the second colume represents the duration
of time;
bpm: type: int, beats per minute;
SampleFrequency: type: int, the frequency of sampling signal;
wave: type: string, different sampling wave, e.g. 'sin','square'...;
key: type: char, i.e. 'A','B'...;
adjust: type: int, default:0, 1 means '#', -1 means 'b'.

output:
f: type: 1*n double matric.
```

设计思路：

Song第一例表示音符，第二列表示节拍。

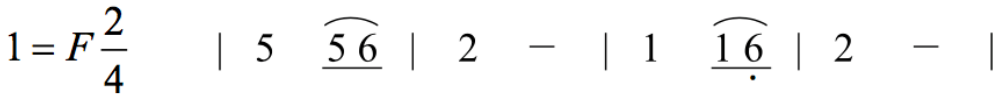


图 1.3: 乐曲《东方红》前四小节曲谱

bpm代表beats per minute，因此1个节拍持续时间为$\frac{60}{bpm}$。查阅资料得，东方红bpm为140。

节拍规则如下：

正常每个音符节拍都为1.

(1)

代表延时1个节拍，即音符2对应2个节拍。

(2)

代表半拍，即音符5和6对应0.5个节拍。

(3) 附点代表1.5个节拍。

wave代表采样波形，默认sin函数，但是用square，sawtooth函数采样音色会有少许差别。

代码如下：

```
function f = Generate_Song1(song,bpm,SampleFrequency,wave,key,adjust)
% input:
% song: type: n*2 matrix, the first colume represents the note('rest' means
% rest in music, while the second colume represents the duration of time;
% bpm: type: int, beats per minute;
% SampleFrequency: type: int, the frequency of sampling signal;
% wave: type: string, different sampling wave, e.g. 'sin','square'...;
% key: type: char, i.e. 'A','B'...;
% adjust: type: int, default:0, 1 means '#', -1 means 'b'.
%
% output:
% f: type: 1*n double matric.

% parameter checkout
validateattributes(bpm,{'numeric'},{'nonempty'});
validateattributes(SampleFrequency,{'numeric'},{'nonempty'});
validateattributes(wave,{'char'},{'nonempty'});
validateattributes(key,{'char'},{'nonempty'});
if nargin == 5
    adjust = 0;
end

f = [];
if wave == 'sin'
    op = @sin;
elseif wave == 'square'
    op = @square;
else
    op = @sawtooth;
end
for i = 1:size(song,1)
    t = 0 : 1/SampleFrequency : (60/bpm)*song(i,2);
    if song(i,1) == -inf
        f = [f,zeros(size(t))];
    else
        f = [f,op(2*pi*t*Note2Frequency(key,song(i,1),adjust))];
    end
end

f = f/max(f);       %normalization
end
```

Dong_Fang_Hong1.m播放《东方红》片段：

```
% clear,clc;

dong_fang_hong = [5,1;5,0.5;6,0.5;2,2;1,1;1,0.5;-1,0.5;2,2];
sound(Generate_Song1(dong_fang_hong,140,8000,'sin','F'),8000);
```

作业 2：

指数衰减包络：

完成 Generate_Wrap 函数生成包络。

input:

p_impulse: type: double, percentage of impulse;

p_decline: type: double, percentage of decline;

```
p_disappear: type: double, percentage of disappaer;
amp_keep: type: double, amplitude relatively;
list: type: 1*n matrix, the original note;
t: type: 1*n matrix, the corresponding time.

output:
f: type: 1*n matrix.
```

设计思路：

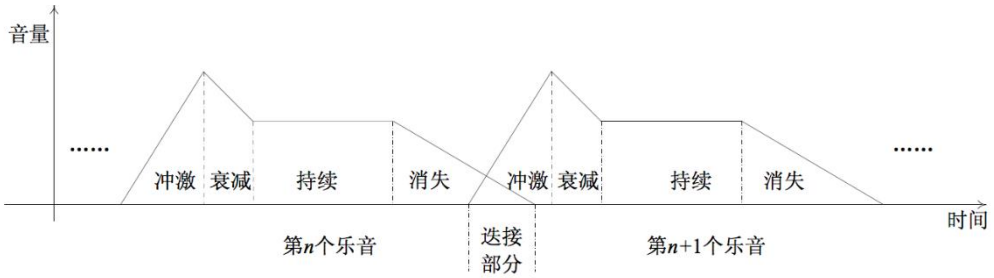


图 1.5: 音量变化

按如图包络去除高频分量：

包络皆为指数函数。

p_impulse为冲激部分所占比例。

p_decline 为衰减部分所占比例。

p_disappear 为消失部分所占比例。

amp_keep 为持续部分幅度/最高点幅度。

代码如下：

```
function f = Generate_Wrap(p_impulse,p_decline,p_disappear,amp_keep,list,t)
% input:
% p_impulse: type: double, percentage of impulse;
% p_decline: type: double, percentage of decline;
% p_disappear: type: double, percentage of disappaer;
% amp_keep: type: double, amplitude relatively;
% list: type: 1*n matrix, the original note;
% t: type: 1*n matrix, the corresponding time.
%
% output:
% f: type: 1*n matrix.

% parameter checkout
validateattributes(p_impulse,{'numeric'},{'>',0,'<',1});
validateattributes(p_decline,{'numeric'},{'>',0,'<',1});
validateattributes(p_disappear,{'numeric'},{'>',0,'<',1});
validateattributes(amp_keep,{'numeric'},{'>',0,'<',1});

p_keep = 1 - p_impulse - p_decline - p_disappear;
L = length(list);
L_impulse = floor(L*p_impulse);
L_decline = floor(L*(p_impulse+p_decline));
L_keep = floor(L*(p_impulse+p_decline+p_keep));
ta = t(1:L_impulse);
td = t(L_impulse+1:L_decline);
tr = t(L_keep+1:end);

% envelope for Attack
ea = (1-exp(-ta))/(1-exp(-ta(end)));
% envelope for Decay
ed = (1-amp_keep)*(exp(-(td-td(end)))-1)/(exp(-(td(1)-td(end)))-1)+amp_keep;
% envelope for Release
er = amp_keep*(exp(amp_keep-(tr-tr(1))/(tr(end)-tr(1))*amp_keep)-1)/...
    (exp(amp_keep)-1);

wave_impulse = list(1:L_impulse) .* ea;
wave_decline = list(L_impulse+1:L_decline) .* ed;
wave_keep = list(L_decline+1:L_keep) .* amp_keep;
wave_disappear = list(L_keep+1:end) .* er;

f = [wave_impulse,wave_decline,wave_keep,wave_disappear];

end

```

初步完成 Generate_Song2 函数。

```
Generate_Song2(song,bpm,SampleFrequency,wave,key,wrap,adjust,harmonic
)


input:
song: type: struct, the first colume represents the note('rest' means
rest in music), while the second colume represents the duration of
time;
bpm: type: int, beats per minute;
SampleFrequency: type: int, the frequency of sampling signal;
wave: type: string, different sampling wave, e.g. 'sin','square'...;
key: type: char, i.e. 'A','B'...;
```

```
wrap: type: 1*4 char matrix; represents p_impulse, p_decline,
p_disappear,amp_keep;
adjust: type: int, default:0, 1 means '#', -1 means 'b';
harmonic: type: 1*n matrix, represents harmony

output:
f: type: 1*n double matric.
```

设计思路：

在 Generate_Song1 的基础之上，增加包络部分。为了适应后面的作业，将输入数据从 list 变为 struct，不过在作业 2 中只会用到 note 和 beats 两个属性，不会应用到和弦的部分。因此将 wrap 作为 1*4 的矩阵输入到函数即可。

代码如下：

```
function f = Generate_Song2(song,bpm,SampleFrequency,wave,key,wrap,adjust,harmonic)
% input:
% song: type: struct, the first colume represents the note('rest' means
% rest in music), while the second colume represents the duration of time;
% bpm: type: int, beats per minute;
% SampleFrequency: type: int, the frequency of sampling signal;
% wave: type: string, different sampling wave, e.g. 'sin','square'...;
% key: type: char, i.e. 'A','B'...;
% wrap: type: 1*4 char matrix; represents p_impulse, p_decline, p_disappear,
% amp_keep;
% adjust: type: int, default:0, 1 means '#', -1 means 'b';
% harmonic: type: 1*4 matrix, represents harmony.
%
% output:
% f: type: 1*n double matric.

% parameter checkout
validateattributes(bpm,{'numeric'},{'nonempty'});
validateattributes(SampleFrequency,{'numeric'},{'nonempty'});
validateattributes(wave,{'char'},{'nonempty'});
validateattributes(key,{'char'},{'nonempty'});
if nargin == 6
    adjust = 0;
    harmonic = [];
elseif nargin == 7
    if length(adjust) ~= 1
        harmonic = adjust;
        adjust = 0;
    else
        harmonic = [];
    end
end
```

```
% get wave
f = [];
if wave == 'sin'
    op = @sin;
elseif wave == 'square'
    op = @square;
else
    op = @sawtooth;
end

for i = 1 : length(song)
    t = 0 : 1/SampleFrequency : (60/bpm)*song(i).beats;
    tem = zeros(size(t));
    if song(i).note ~= -inf
        % no harmonic inout
        if length(harmonic) == 0
            if length(song(i).note) == length(song(i).amp)
                for k = 1 : length(song(i).note)
                    tem = tem + song(i).amp(k) * ...
                        op(2*pi*t*Note2Frequency(key,song(i).note(k),adjust));
                end
            else
                base = Note2Frequency(key,song(i).note,adjust);
                for k = 1 : length(song(i).amp)
                    tem = tem + song(i).amp(k) * op(k*2*pi*t*base);
                end
            end
        % with harmonic input
        else
            for k = 1 : length(harmonic)
                tem = tem + harmonic(k) * op(k*...
                    2*pi*t*Note2Frequency(key,song(i).note,adjust));
            end
        end
        tem = Generate_Wrap(wrap(1),wrap(2),wrap(3),wrap(4),tem,t);
    end
    f = [f,tem];
end

f = f/max(f);       %normalization
end
```

Dong_Fang_Hong2.m播放包络处理后的《东方红》片段：

```
wrap1 = [0.9,0.05,0.05,0.001];      % organ;
wrap2 = [0.05,0.9,0.05,0.001];      % piano
wrap3 = [0.05,0.4,0.5,0.8];         % guitar
wrap4 = [0.25,0.25,0.25,0.5];       % all middle

dong_fang_hong = [struct('note',5,'beats',1),...
    struct('note',5,'beats',0.5),...
    struct('note',6,'beats',0.5),...
    struct('note',2,'beats',2),...
    struct('note',1,'beats',1),...
    struct('note',1,'beats',0.5),...
    struct('note',-1,'beats',0.5),...
    struct('note',2,'beats',2)];
harmonic = [1,0.2,0.3];
% harmonic = [661 963 634 727 35 73 237];
harmonic = harmonic / harmonic(1);

sound(Generate_Song2(dong_fang_hong,140,8000,'sin','F',wrap1),8000);
% raise half
% sound(Generate_Song2(dong_fang_hong,140,8000,'sin','F',wrap1,1),8000);
% add harmonics, simulate organ
% sound(Generate_Song2(dong_fang_hong,140,8000,'sin','F',wrap1,harmonic),8000);
% sound(Generate_Song2(dong_fang_hong_C,140,8000,'sin','C',wrap4),8000);
```

经过对 harmonic 参数的实验，我发现 amp_keep 既不能太大也不能太小，否则音符衔接处都会出现爆裂的声音。大体得出了几组较有代表性的数据：

```
wrap1 = [0.9,0.05,0.05,0.001];
wrap2 = [0.05,0.9,0.05,0.001];
wrap3 = [0.05,0.4,0.5,0.8];
wrap4 = [0.25,0.25,0.25,0.5];
```

wrap1 近似竖琴的音色。

wrap2 近似钢琴的音色。

wrap3 近似吉他声音。

wrap4 是所有部分都平均，即 p_impulse，p_decline，p_disappear = 0.25，amp_keep = 0.5。

作业 3:：

1） 升高、降低 8 度：

方法 1： 改谱子即可，即每个音调+或-7。

方法 2： 使用 resample 函数。升高 8 度：data = resample(data, 1, 2)，降低 8 度：data = resample(data, 2)。

2） 升高半个音阶：

方法 1： 由于 Generate_Song1 与 Generate_Song2 函数皆有 adjust 变量作为变调输入（默认为 0，1 为‘#’，-1 为‘b’），因此，将 adjust 作为 1 输入即可。

方法 2： 使用 resample 函数。$2^{\frac{1}{12}} = \frac{1657}{1564}$，因此，data = resample(data, 1564, 1657)。

作业 4：

增加谐波分量。

设计思路：

直接在 Generate_Song2 函数中增加 harmonic 输入。

由于 Generate_Song2 函数已根据之后的需求进行了更改，因此这里仍然不需要更改 struct，直接输入即可。

```
harmonic: type: 1*4 matrix, represents harmony.
```

harmonic 每个元素依次对应基频，2 次谐波，3 次谐波……

根据[1 0.2 0.3]谐波输入后，声音确实比较像风琴的声音，变得更真实，更具变化与起伏。

作业 5：

自选音乐合成。

选取 Summer 这首钢琴曲，Piano.m 合成。

包络选择：

```
wrap2 = [0.05,0.9,0.05,0.001];
```

谐波选择：

```
harmonic = [1,0.2,0.3];
```

合成的音乐效果较好，跟钢琴音色很像。

代码如下：

```
wrap2 = [0.05,0.9,0.05,0.001];      % piano
harmonic = [1,0.2,0.3];

summer = ...
Summer = [];
for i = 1 : size(summer,1)
    Summer = [Summer,struct('note',summer(i,1),'beats',summer(i,2))];
end
% use piano
sound(Generate_Song2(Summer,140,8000,'sin','E',wrap2,-1,harmonic),8000);
```

二、用傅里叶级数分析音乐

作业 6：

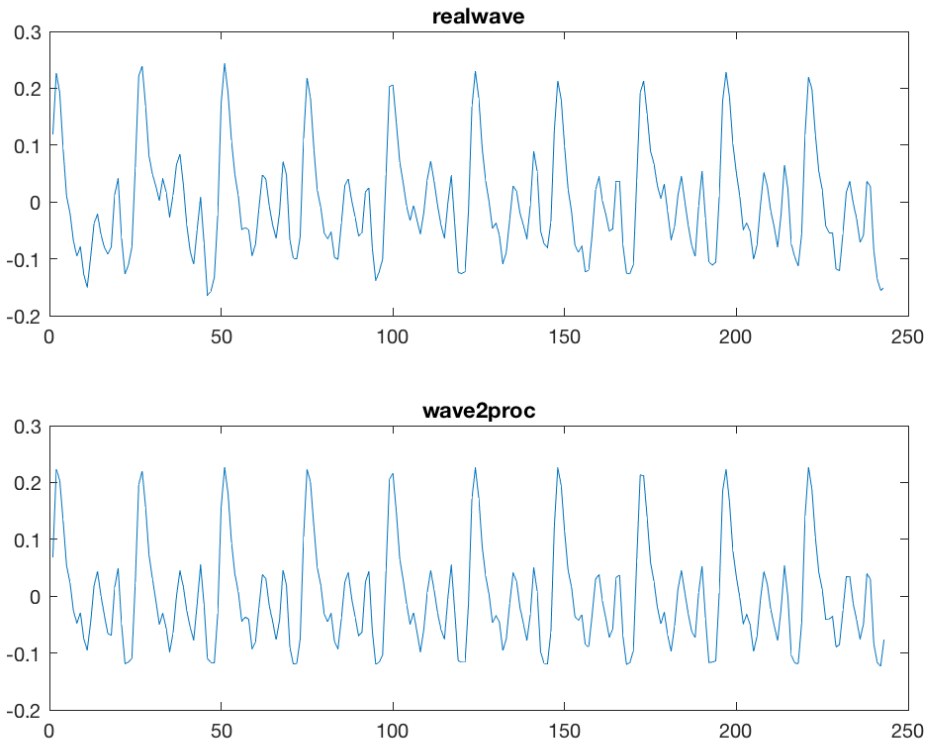
```
>> data = audioread('fmt.wav');
>> sound(data)
```

听起来就是真实听到的音乐，而非合成音。

猜测因为叠音，和弦的原因。

作业 7：

```
>> load('Guitar.MAT')
>> subplot(2,1,1),plot(realwave),title('realwave');
>> subplot(2,1,2),plot(wave2proc),title('wave2proc');
```



设计思路：

分析发现 wave2proc 大致有十个周期，且每个周期近似相同。

```
>> length(realwave)

ans =

   243
```

于是将 realwave 重新采样为长度 240 的信号，周期平均后复制十份后再重新采样为 243 的信号。

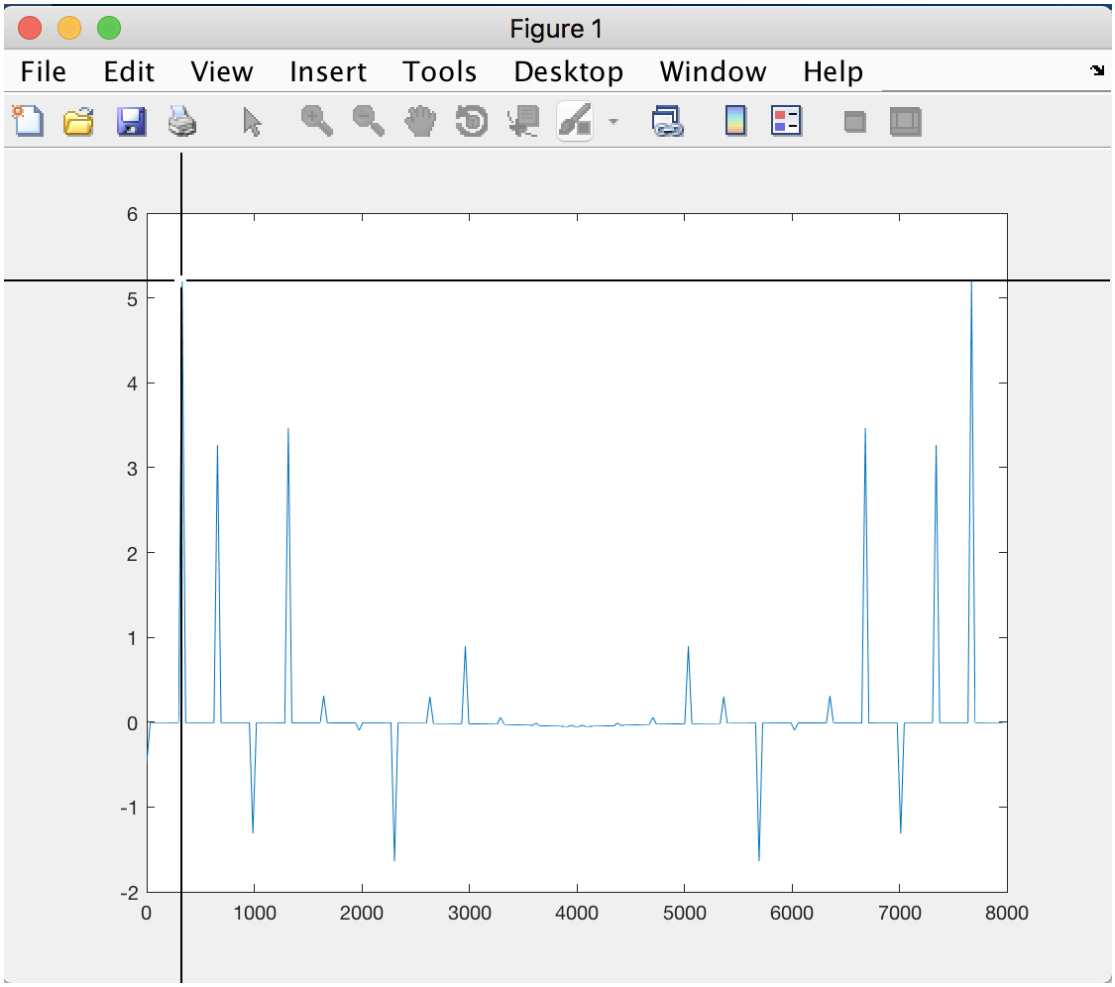
代码如下：realwave2wave2proc.m：

```
load guitar;
 % reshape to the multiple of 10
realwave_10 = reshape(resample(realwave,240,243),24,10);
realwave_mean = mean(realwave_10,2)';    % get the average period of realwave
temp = repmat(realwave_mean,1,10);      % replicate realwave_mean
generate_wave2proc = resample(temp,243,240);
```

作业 8：

基频，音调：

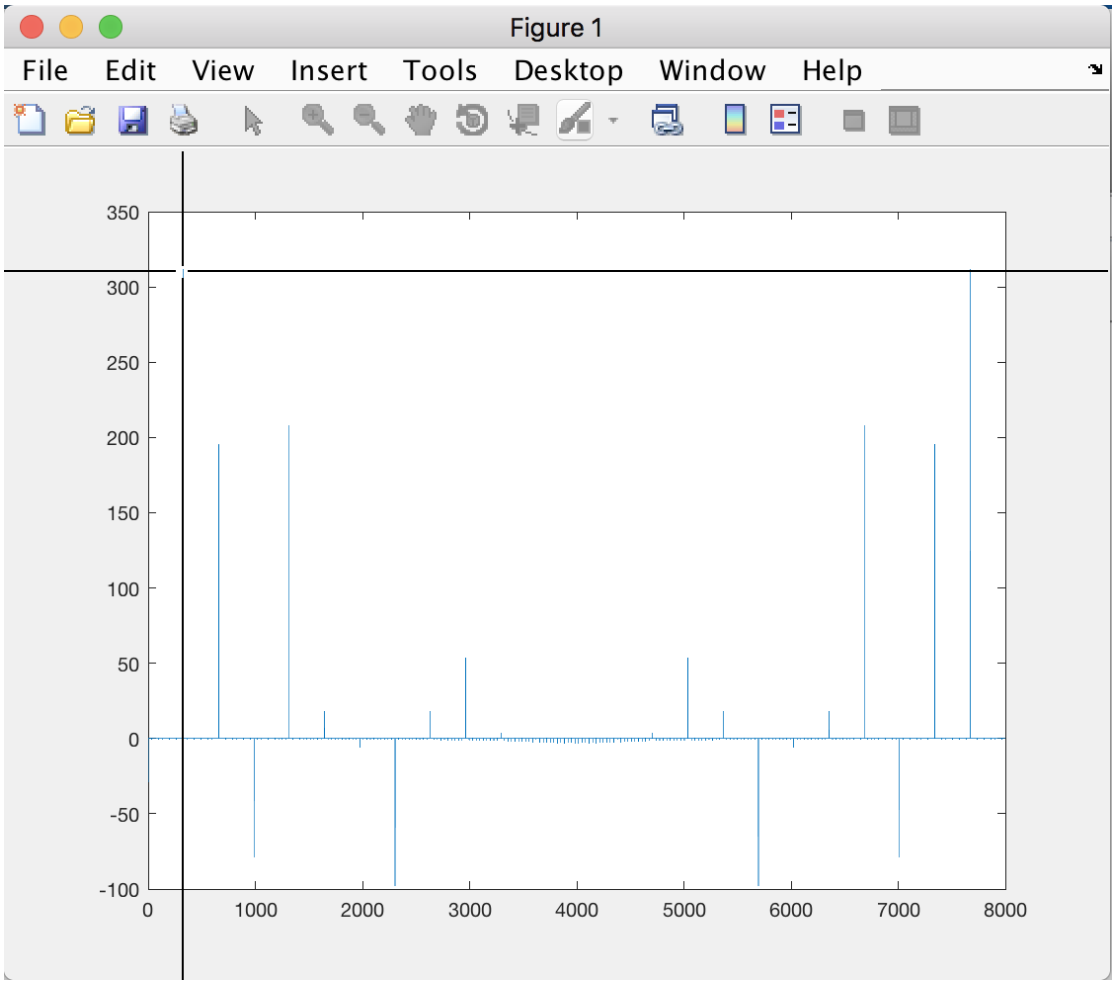
我对整个 wave2proc 信号求傅里叶变换，首先未经处理：



```
x =

   327.1889

y =

    5.2234
```

发现很难定位基频的准确位置。

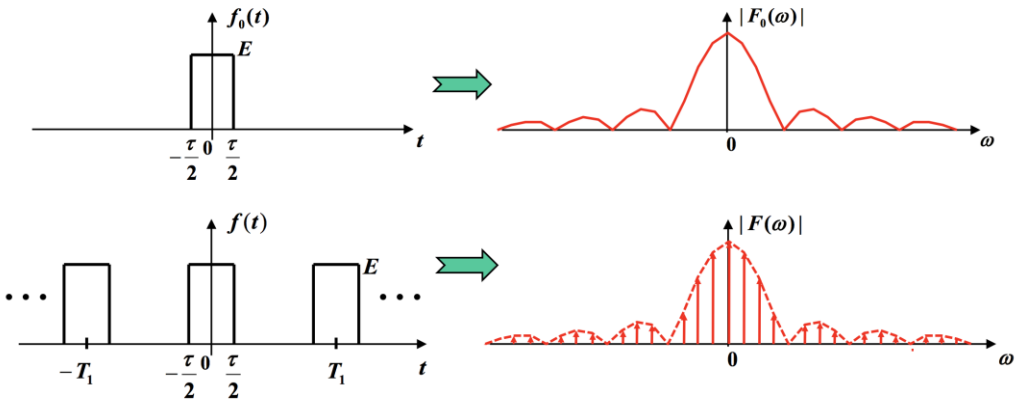
之后我将 wave2proc 复制 60 遍后再进行傅里叶变换：



```
x =

   327.1889


y =

   312.8832
```

现在数据明显准确很多，近似为频域上的冲激函数叠加。

理由：当脉冲数增多直至趋于无穷，即成为周期信号，频谱由连续谱退化为离散谱，由分立的冲激函数构成。



作业 9：

自动分析音乐音调和节拍。

1） 自动分割音符。

步骤 1：设定步长扫描整个音频。取每个区间内的最大值。这些区间需满足以下条件才能作为备选。

①该区间最大值比前一区间最大值大一定比例（ratio）；

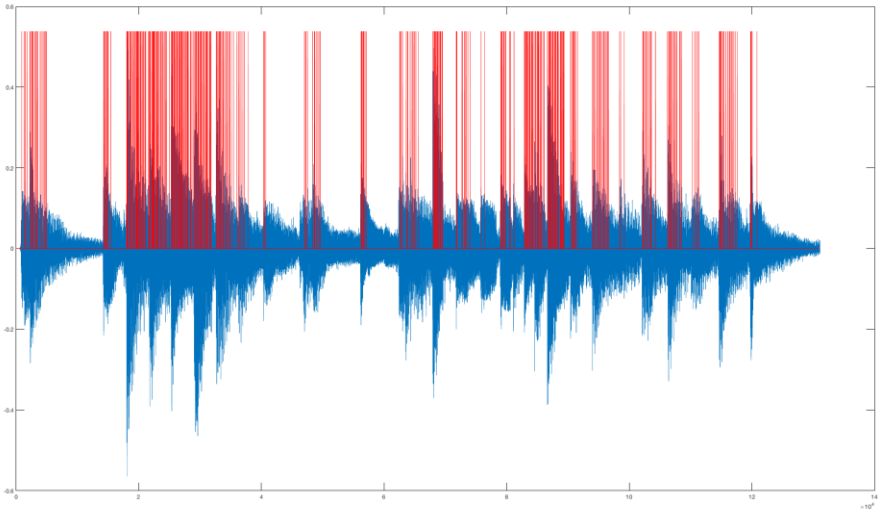
②该区间最大值大于一个最小值；

③该区间最大值的 index 与前一区间最大值 index 的间隔需大于一最小值（interval）。

步骤 1 代码如下：

```
% split note
step = 6;
P_former = 0;
index_former = -1000;
ratio = 1.8;
interval = 100;
min_note = 0.12;
id = [];
for i = 1 : step : L - step
    [P,index] = max(data(i:i+step));
    index = index + i - 1;
    if P>ratio*P_former & index>index_former+interval & P>min_note
        id = [id,index];
        index_former = index;
    end
    P_former = P;
end
```

步骤一后分割效果如下：



备选音符数量很多，还需后续处理

步骤 2：对满足一定距离（neighbour）的音符进行聚类处理。

cluster函数：`cluser(neighbour,id,data)`

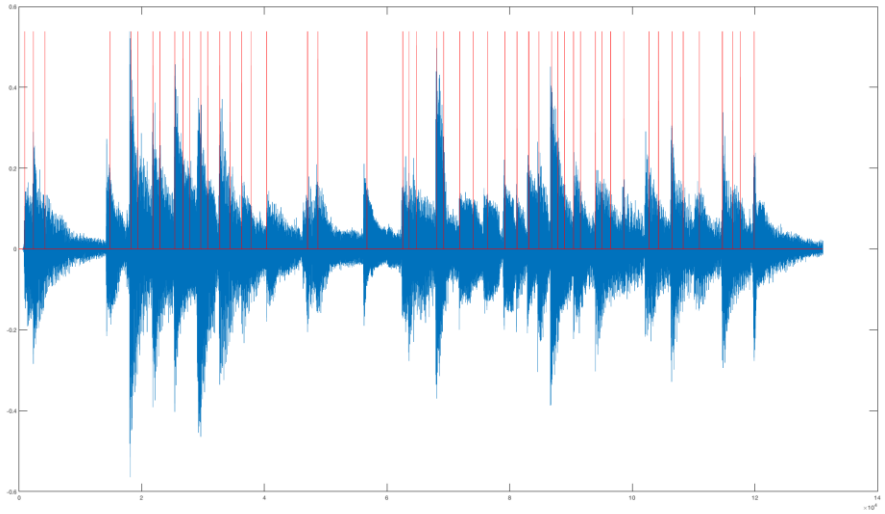
```
input:
neighbour: type: int, min interval to cluster;
id: type: 1*n matrix, data to be clustered;
data: according data.

output:
f: clustered id
```

根据 id 之间的最小距离聚类算法处理，并更改相应 id 与 data。代码如下：

```
function f = cluser(neighbour,id,data)
% input:
% neighbour: type: int, min interval to cluster;
% id: type: 1*n matrix, data to be clustered;
% data: according data.
%
% output:
% f: clustered id

if nargin == 3
    flag = 1;
    set = id(1);
    while flag+1 <= length(id)
        if id(flag+1)-set < neighbour
            if data(id(flag)) >= data(id(flag+1))
                set = id(flag);
                id(flag + 1) = [];
            else
                set = id(flag + 1);
                id(flag) = [];
            end
        else
            set = id(flag + 1);
            flag = flag + 1;
        end
    end
else
    flag = 1;
    set = id(1);
    while flag+1 <= length(id)
        if abs(id(flag+1)-set) < neighbour
            set = id(flag);
            id(flag + 1) = [];
        else
            set = id(flag + 1);
            flag = flag + 1;
        end
    end
end
f = id;
end
```

聚类处理后音符分割效果：

```
22      % cluster
23 -    neighbour = 1200;
24 -    id = cluster(neighbour,id,data);
```



发现数量以少了很多。但还有杂音需进一步处理。

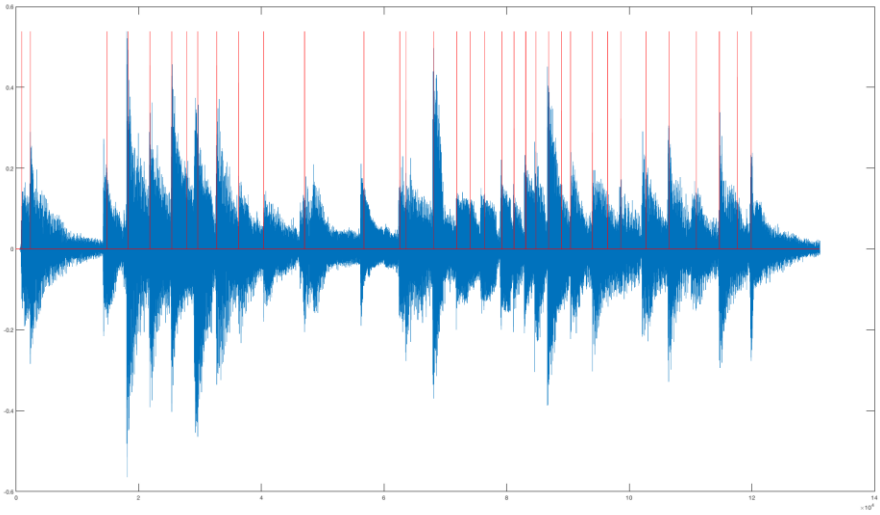
步骤 3：观察发现，每个音符大致为突然增大之后近似钟形函数衰减。于是继续过滤。

①该位置比幅度比之前位置幅度小一定比例（min_ratio）；

②该位置与上一位置之间间距小于一阈值（interval_note）。

```
25      % filter to minimize notes left
26 -    lid = 0;
27 -    interval_note = 1900;
28 -    while(lid ~= length(id))
29 -        lid = length(id);
30 -        flag = 1;
31 -        note_num = 32;
32 -        amp = data(id(1));
33 -        while(flag+1 <= length(id))
34 -            if amp > data(id(flag+1)) & id(flag+1)-id(flag)<interval_note
35 -                amp = data(id(flag + 1));
36 -                id(flag + 1) = [];
37 -            else
38 -                flag = flag + 1;
39 -                amp = data(id(flag));
40 -            end
41 -        end
42 -    end
```

过滤后效果如下：



可以发现，过滤后剩余 35 个音符，与目测 32 个音符相差不大，且分割位置绝大多数较为精确。满足后续工作条件。

2） 频率到音符、节拍：

首先得到各音符的节拍：

```
44      % get the beats of each note
45 -    beats = [];
46 -    for i = 1 : length(id) - 1
47 -        beats = [beats,id(i+1)-id(i)];
48 -    end
49 -    beats = [beats,length(data)-id(end)]/8000;
50 -    id = [id,length(data)];
```

之后，对于每个音符，复制 100 倍进行傅里叶变换，从频域寻找幅值足够大的频率。

```
[F,w,N] = fft_plot(repmat(data(id(k):id(k+1)),100,1));  % replicate 100 times
% get frequency large enough
F = F(1:floor(end/2));
f = find(F>max(F)*amp_limit)*8000/N;
f = f(f>120 & f<800);
f = cluster(cluster_limit,f);
```

得到频率后，寻找基频，过滤基频的整数倍，将所有数据保存在 baseband 中：

```
% remove multiply of baseband
temp = struct('note',[],'amp',[]);
for i = 1 : length(f)
    flag = 1;
    for j = 1 : length(temp.note)
        ratio = f(i) / temp.note(j);
        if ratio>=round(ratio)*(1-limit) & ratio<=round(ratio)*(1+limit)
            flag = 0;
            break;
        end
    end
    if(flag)
        temp.note = [temp.note,f(i)];
        temp.amp = [temp.amp,F(round(f(i)*N/8000))];
    end
end
temp.amp = temp.amp / max(temp.amp);
baseband = [baseband;temp];
```

然后，完成 Frequency2Note 函数，得到对应简谱：

```
Frequency2Note(baseband,ratio,beats)

input:
baseband: type: struct;
ratio: type: double;
beats: type: 1*n matrix.

output:
f: type: struct, song;
bpm: type: numerical
```

寻找最近的频率，如相差过大则丢弃。同时返回带有 beats，amp 特征的音符 struct。

代码如下：

```
function [f,bpm] = Frequency2Note(baseband,ratio,beats)
% input:
% baseband: type: struct;
% ratio: type: double;
% beats: type: 1*n matrix.
%
% output:
% f: type: struct, song;
% bpm: type: numerical

freq = [];
for i = -7 : 12
    freq = [freq,Note2Frequency('C',i)];
end
map = containers.Map(freq,-7:12);

min_beats = min(beats);
bpm = 60/min_beats;
for i = 1 : length(baseband)
    baseband(i).beats = beats(i)/min_beats;
    j = 1;
    while j <= length(baseband(i).note)
        success = 0;
        flag = 1;
        while flag <= length(freq)
            if baseband(i).note(j)>=freq(flag)*(1-ratio) & ...
                    baseband(i).note(j)<=freq(flag)*(1+ratio)
                baseband(i).note(j) = map(freq(flag));
                if j==1 || (j~=1 & baseband(i).note(j)~=baseband(i).note(j-1))
                    j = j + 1;
                    success = 1;
                    break;
                end
            else
                flag = flag + 1;
            end
        end
        if success == 0
            baseband(i).note(j) = [];
            baseband(i).amp(j) = [];
        end
    end
end
f = baseband;

end
```

最后，将简谱输出到 notation.txt 中，并用 Generate_Song2 函数播放简谱验证。结果比较令人满意，基本无走调情况，且节奏也比较准确。低音部分音质不太理想，噪声较多；而高音部分音质比较理想。经思考，我认为问题出在分割音符步骤 1 中，由于步长是人工设置，并在每个步长中找寻最大值，因此，无可避免的会将一些音符遗漏（A 应为音符，但却被步长划分在前一区间，被更大的 B 过滤掉）。

另外，虽然加入了和弦的部分，但是仍然很难真正模仿 fmt 吉他的音色。这应该还是包络的问题，真实音乐的包络应该复杂许多。

作业 9 完整代码如下：

```
% read data
data = audioread('fmt.wav');
L = length(data);

% split note
step = 6;
P_former = 0;
index_former = -1000;
ratio = 1.8;
interval = 100;
min_note = 0.12;
id = [];
for i = 1 : step : L - step
    [P,index] = max(data(i:i+step));
    index = index + i - 1;
    if P>ratio*P_former & index>index_former+interval & P>min_note
        id = [id,index];
        index_former = index;
    end
    P_former = P;
end
% cluster
neighbour = 1200;
id = cluster(neighbour,id,data);
% filter to minimize notes left
lid = 0;
interval_note = 1900;
while(lid ~= length(id))
    lid = length(id);
    flag = 1;
    note_num = 32;
    amp = data(id(1));
    while(flag+1 <= length(id))
        if amp > data(id(flag+1)) & id(flag+1)-id(flag)<interval_note
            amp = data(id(flag + 1));
            id(flag + 1) = [];
        else
            flag = flag + 1;
            amp = data(id(flag));
        end
    end
end
```

```
44      % get the beats of each note
45 -    beats = [];
46 -    for i = 1 : length(id) - 1
47 -        beats = [beats,id(i+1)-id(i)];
48 -    end
49 -    beats = [beats,length(data)-id(end)]/8000;
50 -    id = [id,length(data)];
51
52      %collect baseband
53 -    limit = 0.003;
54 -    amp_limit = 0.43;
55 -    cluster_limit = 10;
56 -    baseband = [];
57 -    for k = 1 : length(id) - 1
58 -        [F,w,N] = fft_plot(repmat(data(id(k):id(k+1)),100,1));  % replicate 100 times
59          % get frequency large enough
60 -        F = F(1:floor(end/2));
61 -        f = find(F>max(F)*amp_limit)*8000/N;
62 -        f = f(f>120 & f<800);
63 -        f = cluster(cluster_limit,f);
64          % remove multiply of baseband
65 -        temp = struct('note',[],'amp',[]);
66 -        for i = 1 : length(f)
67 -            flag = 1;
68 -            for j = 1 : length(temp.note)
69 -                ratio = f(i) / temp.note(j);
70 -                if ratio>=round(ratio)*(1-limit) & ratio<=round(ratio)*(1+limit)
71 -                    flag = 0;
72 -                    break;
73 -                end
74 -            end
75 -            if(flag)
76 -                temp.note = [temp.note,f(i)];
77 -                temp.amp = [temp.amp,F(round(f(i)*N/8000))];
78 -            end
79 -        end
80 -        temp.amp = temp.amp / max(temp.amp);
81 -        baseband = [baseband;temp];
82 -    end
83
84 -    [song,bpm] = Frequency2Note(baseband,0.05,beats);       % get the song and bpm
85
86 -    wrap1 = [0.9,0.05,0.05,0.001];     % organ;
87
88 -    sound(Generate_Song2(song,bpm,8000,'sin','C',wrap1),8000);
```

三、基于傅里叶级数的合成音乐

作业 10：

完成 fft_plot 函数，方便同时显示时域与频域波形。

```
input:
signal: type: 1*n matrix.

output:
f: fft of signal;
w: corresponding frequency.
picture of the fft of x
```

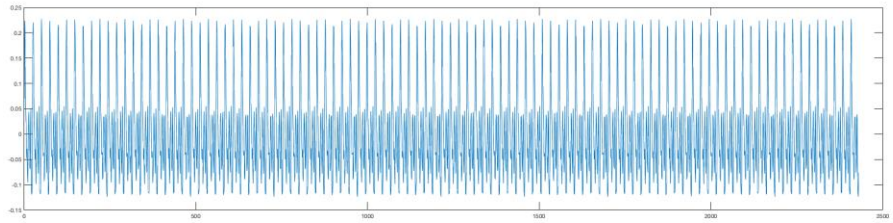
代码如下：

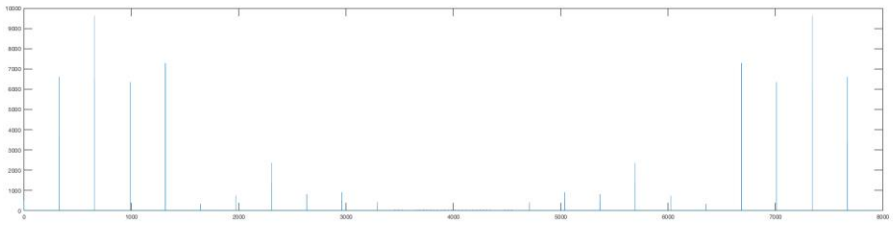
```
function [f,w,N] = fft_plot(signal)
% input:
% signal: type: 1*n matrix.
%
% output:
% f: fft of signal;
% w: corresponding frequency.
% picture of the fft of x
f = abs(fft(signal));
w = 8000 * (0:length(signal)-1) / length(signal);
N = length(signal);
subplot(2,1,1);plot(signal(1:floor(length(signal)/100)));
subplot(2,1,2);plot(w,f);
end
```

设计思路：

首先傅里叶变换寻找 wave2proc 谐波分量。（High_10.m）

```
load guitar;
wave2proc = repmat(wave2proc,1000,1);
fft_plot(wave2proc);

x = [];
y = [];
for i = 1 : 4
    [tx,ty] = ginput(1);
    x = [x,tx];
    y = [y,ty];
end
```





得到：

```
x =

   1.0e+03 *

    0.3239    0.6519    0.9852    1.3132


y =

   1.0e+03 *

    6.6168    9.6937    6.3889    7.3860
```

基频为 323Hz，存在 2、3、4 次谐波分量。

```
harmonic = [6.6168,9.6937,6.3889,7.3860];
harmonic = harmonic / harmonic(1);
sound(Generate_Song2(dong_fang_hong,140,8000,'sin','F',wrap3,harmonic),8000);
```

按照对应谐波并选择 wrap3（guitar）输入，再次得到《东方红》。

得到的声音有吉他拨弦的感觉，但是跟吉他音色还是存在一定差别。分析应该是吉他波形的包络较为复杂且吉他音符有许多叠音出现，而 Generate_Song2 函数时等到每个波形衰减至 0 后再进入下一个音符。

作业 11：

设计思路：

对于每个基频，通过对幅度的限定在 fmt 中依次寻找其整数倍谐波分量。并将幅度存储在 result 中。结果如下：

```
note: -1
 amp: [1 0.2497 0 0 0 0 0 0 0 0 0 0 0]

note: 0
 amp: [1 0.4834 0.2303 0.3030 0 0 0 0 0 0 0 0 0]

note: 1
 amp: [1 0 0 0 0 0 0 0 0 0 0 0 0]

note: 2
 amp: [1 0 0 0 0 0 0 0 0 0 0 0 0]

note: 3
 amp: [1 0 0.3030 0 0 0 0 0 0 0 0 0 0]

note: 4
 amp: [1 0 0 0 0 0 0 0 0 0 0 0 0]

note: 5
 amp: [1 0 0 0 0 0 0 0 0 0 0 0 0]

note: 6
 amp: [1 0 0 0 0 0 0 0 0 0 0 0 0]
```

观察发现，由于 fmt 音调较低，因此《东方红》中的高音的泛音较难得到。而低音分量较为完备。播放后发现效果好于作业 10，拨弦的感觉更明显，和弦也更丰富。

完整代码 High_11.m：

```
data = audioread('fmt.wav');
L = length(data);
freq = [];
for i = -1 : 6
    freq = [freq,Note2Frequency('F',i)];
end
map = containers.Map(freq,-1:6);

% find harmonic
[f,w] = fft_plot(data);
s = 8000 / L;
result = [];
for i = 1 : length(freq)
    start = round((freq(i)-5)/s);
    over = round((freq(i)+5)/s);
    m0 = max(f(start:end));
    if m0 > max(f) * 0.5
        j = 2;
        tem = struct('note',map(freq(i)),'amp',[1]);
        while round((freq(i)*j+5)/s) <= L/2
            start = round((freq(i)*j-5*j)/s);
            over = round((freq(i)*j+5*j)/s);
            m = max(f(start:over));
            if m >= m0 * 0.1
                tem.amp = [tem.amp,m/m0];
            else
                tem.amp = [tem.amp,0];
            end
            j = j + 1;
        end
        result = [result,tem];
    end
end
dong_fang_hong = [struct('note',5,'beats',1),
    struct('note',5,'beats',0.5),
    struct('note',6,'beats',0.5),
    struct('note',2,'beats',2),
    struct('note',1,'beats',1),
    struct('note',1,'beats',0.5),
    struct('note',-1,'beats',0.5),
    struct('note',2,'beats',2)];
% add to dong_fang_hong
for i = 1 : length(dong_fang_hong)
    dong_fang_hong(i).amp = result(dong_fang_hong(i).note+2).amp;
end
wrap3 = [0.05,0.4,0.5,0.8];        % guitar
sound(Generate_Song2(dong_fang_hong,140,8000,'sin','F',wrap3),8000);
```

小结：

本次 MATLAB 音乐合成大作业对我帮助较大。不管是对信号的理解还是 MATLAB 编程，debug 技术都有提升。

作业 9 是最有挑战性的一项。我尝试了多种方法，诸如互相关分割音符，考察频率的幅度过滤等，从效果看较令人满意。但是，由于途中使用了不少参数且需手工调参，仅仅对于 fmt 的效果并不足以表示这是一个通用的自动完成声音信号到乐谱转换的智能程序。事实也正是如此，我用着一套系统处理其他的音乐后效果并不理想。

我认为，希望通过调节参数的传统方法完成通用智能程序是非常困难的。日后还需要完善并改进深度学习技术训练计算机的学习能力。除此之外，生命科学学对于人脑的进一步探索对实现真正的人工智能也至关重要。