MATLAB 高级编程与应用
课程作业报告一
音乐合成

张同和
zhang-th21@mails.tsinghua.edu.cn

Dep.EE,Tsinghua University

2023-07-21

目录

1 简单的合成音乐

1.1 拼接生成《东方红》

I 解决方案

1. **首先对输入的音乐进行编码.**我们用一个实四维行矢量来表示一个音符,其格式为

$$\vec{n} = (name, flat\_sharp, octave, beats) \tag{1}$$

*name*代表其还原后的音名的简谱记法,如*me*对应的*name*项为3
$flat\_shap \in \{-1, 0, 1\}$, 用于表征该音是否是经由升降调得到。升调为1,降调为-1,自然音为0.
octave 用于表征该音所属的八度,按照音乐理论的传统表述,以现代钢琴键盘中央音区作为第四个八度,向左、向右依次定义其它八度的绝对位置。
beats 用于表征该音的时值,可以取连续非负实数值。
并将所有音符组成的空间记为$\tilde{N}$.

2.定义一个将音符转换为频率的函数note2pitch

$$note2pitch : \tilde{N} \to R_{\geq 0} \times \{0, 1, ...11\}^1 \tag{2}$$

此函数将输入的音符向量$\vec{n}$转换为其音高的频率值p(单位:Hertz),并返回其所在八度内的相对位置y(以C为位置0,B为位置11,为每个半音定义一个相对位置)。此函数的具体实现基于十二平均律,本质上是计算一个等比数列的通项,细节详见代码。
此函数的主要用于确定所属调性基准音的绝对频率。之所以还要返回其所在八度中的绝对位置(以C为参照),是为了便于在后续算法中对本调内的其他音进行比较。

3.定义一个将相对音程转换为绝对频率的函数place2pitch

$$place2pitch : \{0, ...11\} \times Z_{\geq 0} \times \tilde{N} \to R_{\geq 0} \tag{3}$$

此函数输入的参数:
*place_in_octave*表征某音符在本八度中的相对位置,*place_of_octave*表征该音符所在的八度,$base_k ey$是此音符所在调性的基准音(用于表征其调性)。

4.为适应简谱乐谱输入,定义一个批量转换简谱记号为音乐的函数simplified2pitch

$$simplified2pitch : Matx\tilde{N} \to (R_{\geq 0})^L \times (R_{\geq 0})^L \tag{4}$$

其中L为乐谱音符总数,Mat为一切简谱乐谱组成的空间(形式上,每个简谱乐谱都是一个矩阵,其每一行为一个音符的四维向量表示,音乐按照各行从上到下的顺序演奏)。此函数将输入的乐谱和所属调性(*sheet*, *base_key*) 转换成系列频率和音符时长组成的矩阵[freq,dur],其中freq、dur分别为两个长为L的列向量,每个entry分别代表对应行输入音符的频率和时延。

[1]虽然可能有溢出0 11的风险,但是音乐从业者不会给出降do或升ti的这两种可能导致越界的记号。

为便于批处理,这一函数的实现细节中使用了矩阵向量运算。
此外,为了适配简谱的表达习惯,此函数内部首先借助一个子函数f_alter先将简谱中的升降号与音名综合起来,表示每个音符的绝对音高,再调用了place2pitch函数。

5.手动为每个音符构建一个正弦基波后借助sound函数播放,遍历整个乐谱得到音频 为保证节拍合理,我们在各个音符播放后强制程序暂停确定的节拍数(所对应的时长).这样便得到了音频的简单表示。

II 客观效果 见此音频:DongFangHong raw.
III 主观感受 音符之间有啪啪的声音。整体感觉音调准确。

1.2 包络修正抑制音间杂声

I 解决方法
为长度为L的时间序列引入的包络函数是

$$f(t) = \begin{cases} t/0.2) & t \in [0, 0.2L) \\ 1 & t \in [0.2L, 0.7L) \\ e^{-4t} & t \in [0.7L, L) \end{cases} \tag{5}$$

在播放之前,将此包络与音频向量做Hadamard Product.

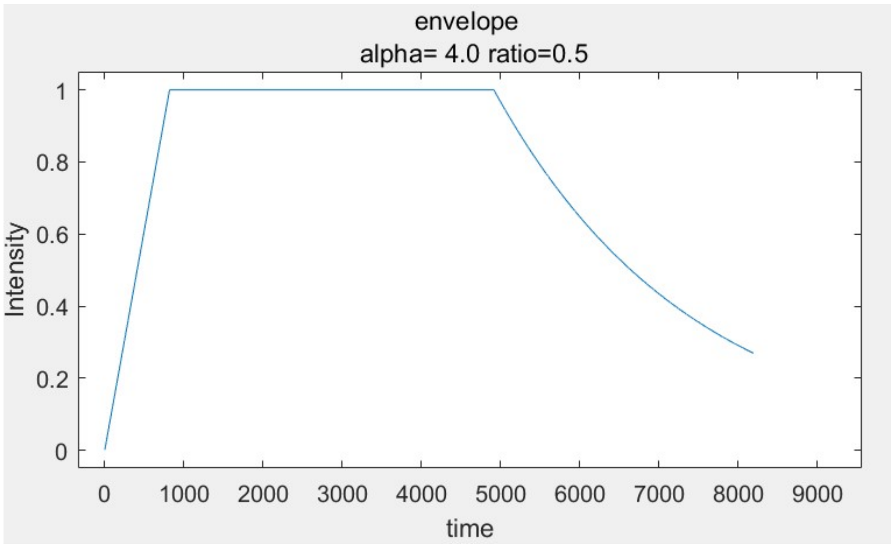


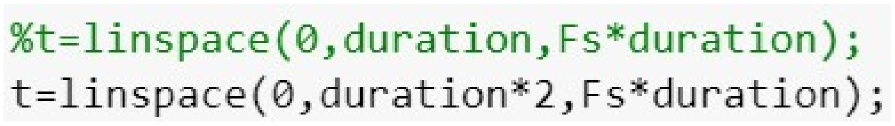
fig 1: 音符包络的数学图像

II 客观效果 见此音频:DongFangHong enveloped.
III 主观感受 调节α的值可以抑制或增强声音之间的杂音。相比不加包络的情况,合适的α对音间杂音的消除效果明显。但是若如果α较大,每个音将会过于短促。

1.3 升降八度与升降半音

I 解决方法

升降八度的最快方法:直接倍增倍减每个音符向量采样总时长*duration*,而不改变采样和播放时的采样率F_s. 如果将采样时长缩短一半,则音调会降一个八度;如果增长一倍,则音调会增加一个八度。

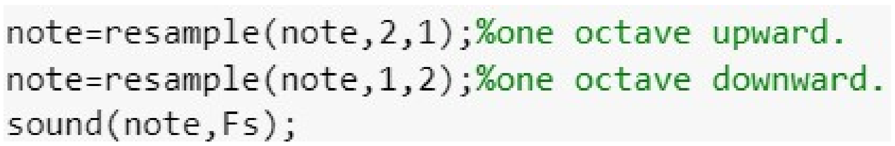


```
%t=linspace(0,duration,Fs*duration);
t=linspace(0,duration*2,Fs*duration);
```

fig 2: 简便方法完成升降八度

升降半音的方法:一种底层方法是直接根据我的设计,修改音乐所属调性的基准音高。这样直接将所有频率进行了重新计算,可以完成任务,甚至都不用写别的函数。

如果要用resample函数,则应该将原有旋律重新采样为之前采样率的2倍(升调)或$\frac{1}{2}$(降调),再按原有采样率播放。副作用:这样会导致每个音的实际时值,导致音符比标记值更冗长或更短促。



```
note=resample(note,2,1);%one octave upward.
note=resample(note,1,2);%one octave downward.
sound(note,Fs);
```

fig 3: Resample方法完成升降八度.
实际使用中前两行至少注释掉一行

II 客观效果 见此音频:DongFangHong modulation.

III 主观感受 可以很直接地听出转调前后的音高变化,说明算法是很成功的。

1.4 增加谐波分量

I 解决方法引入一个描述乐器频谱的函数inst(),根据输入的乐器名,返回一个一维行矢量,依次表征各次谐波对基频分量的的相对强度。如果设置返回值为[1,0.2,0.3],则可以模仿管风琴。

II 客观效果 见此音频:DongFangHong in organ.

III 主观感受 模拟管风琴的音色后,乐曲声音明显更加厚重、真实。

1.5 自选音乐合成

I 解决方案 这里采用F.Chopin:Nocturne op.9 no.1 in B flat minor作为自选音乐,演奏家版本推荐A.Rubinstein的演奏. 我手动将乐曲的旋律转换为简谱,截取前几小节用单音合成。

我使用了改变基准音的方法实现转调(降B小调)。

II 客观效果 见此音频 Nocturne 1.

III 主观感受 电子合成音乐可以表现乐曲的优美旋律,悦耳动听。

2 用傅里叶级数分析音乐

2.1 载入真实音乐

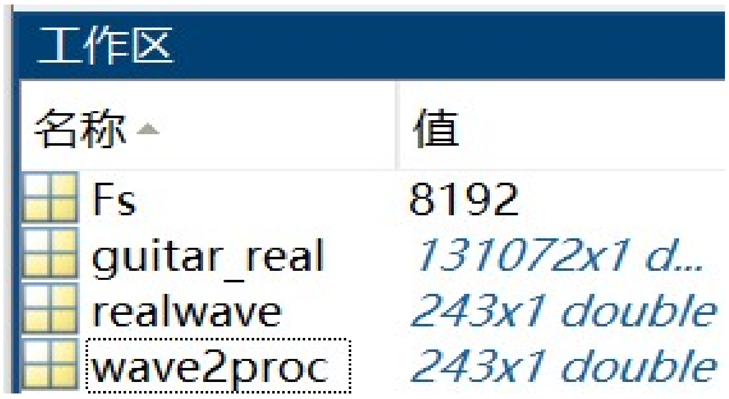
注:大作业文档中建议使用wavread函数载入.wav文件,但是此函数已经于2015年被MATLAB舍弃。一种替代方案是使用audioread函数。

I 解决方案

```
load Guitar.MAT
Fs=8192;
guitar_real=audioread("fmt.wav");
sound(guitar_real,8192);
```

.

II 客观效果



名称	值
Fs	8192
guitar_real	131072x1 d...
realwave	243x1 double
wave2proc	243x1 double

fig 4: 读入数据之后的工作区

III 主观感受

比电子音乐好听太多了!!!重点是有和弦,而且泛音很丰富。

2.2 去除非线性谐波和噪声

处理前后的时域波形对比如下:

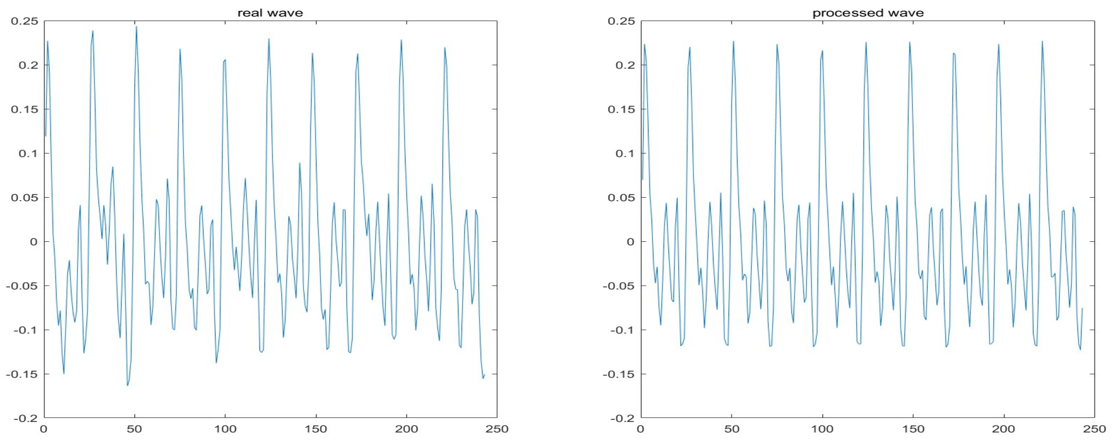


fig 5: 吉他声音信号处理前后时域对比

信号处理之后的时域图像明显更加规整。

查阅资料(见附录[1],[2]) 后我怀疑这一方法是通过平均去噪来实现的。其具体机理大致是：通过对近似独立同分布的信号进行简单采样平均后,由于方差的数学性质,噪声的方差将会被反比于采样数地削减,信噪比也就正比于采样数地上升。于是考虑设计如下算法来进行线性化与去噪:

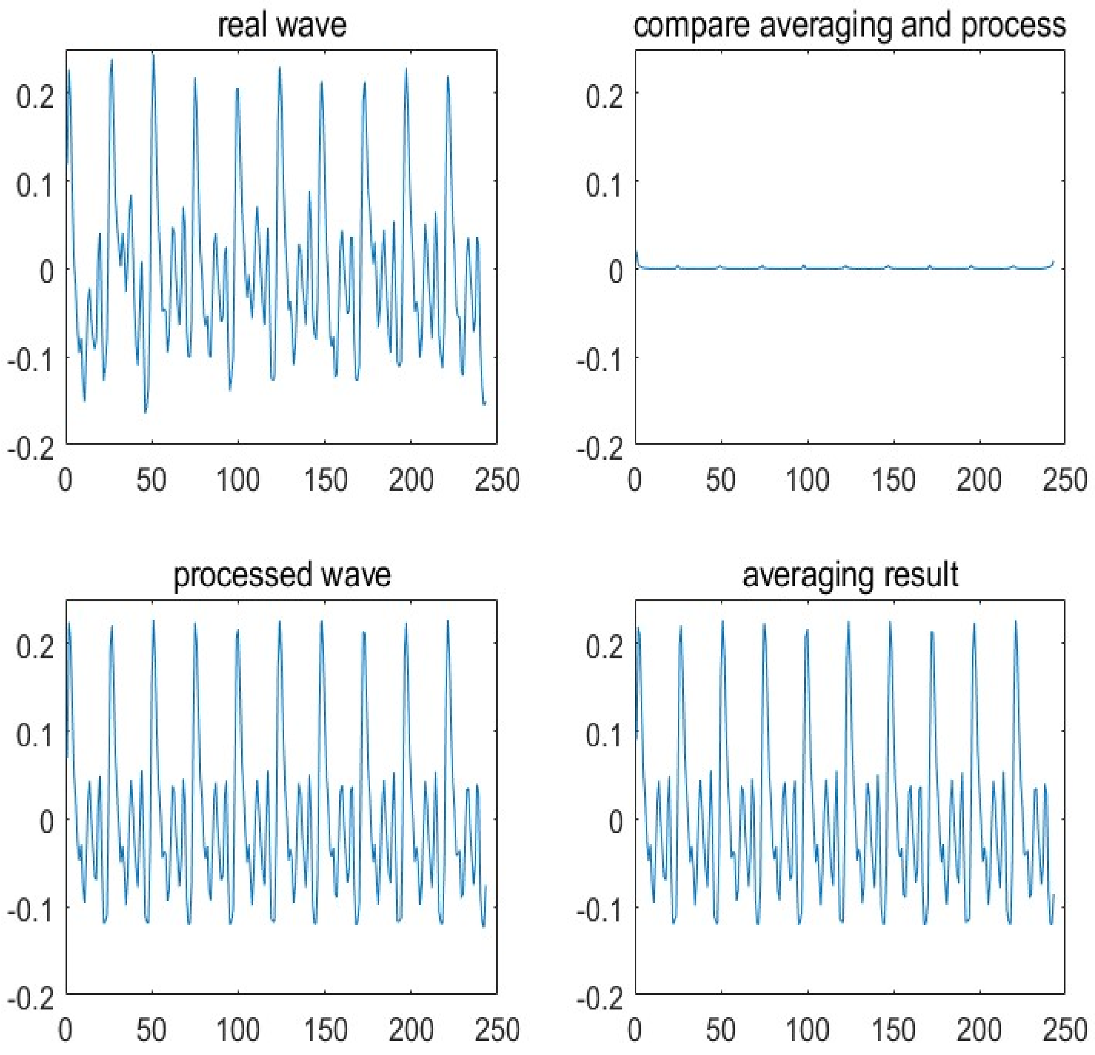


fig 6: 吉他声音用平均去噪与实际处理结果对比

可以发现,使用平均去噪方法和实际处理得到的结果大致相同,统计二者差距的绝对值发现,二者的差异相比信号强度而言非常小,说明这两种算法的效果基本一致,有很大概率认为wave2proc正是使用了平均去噪方法。

2.3 载入真实音乐

I 解决方案 使用时域、频域采样和快速傅里叶变换对连续信号进行频谱分析。为得到更好的尖峰,我们将时域数据重复了30次。基本算法分析详见附录A1

II 客观结果

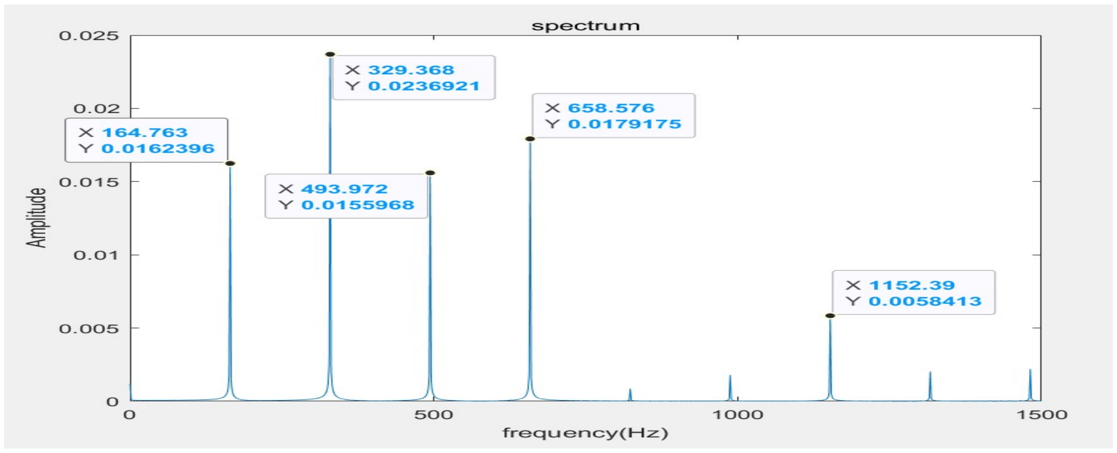


fig 7: 吉他声音信号的谱分析(低频区,拷贝2次)

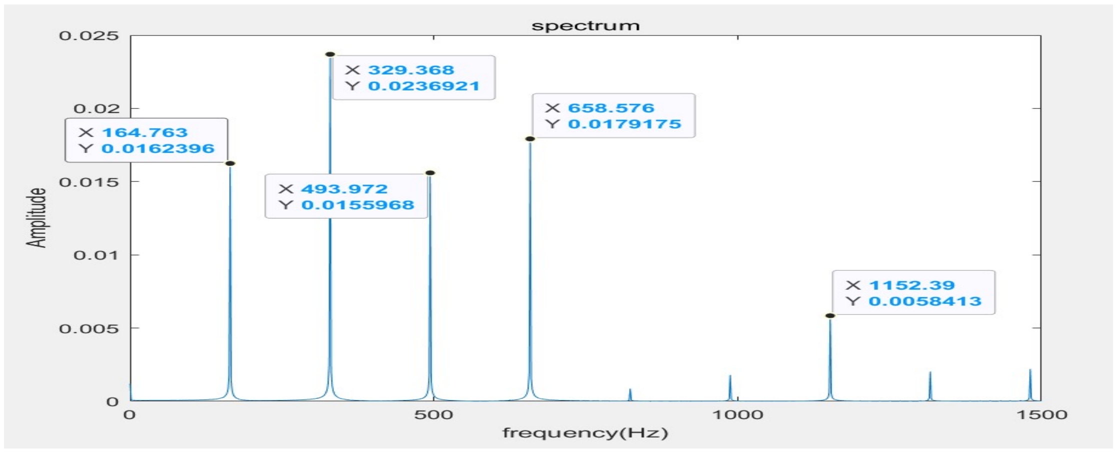


fig 8: 吉他声音信号的谱分析(时域拷贝30次;低频区响应)

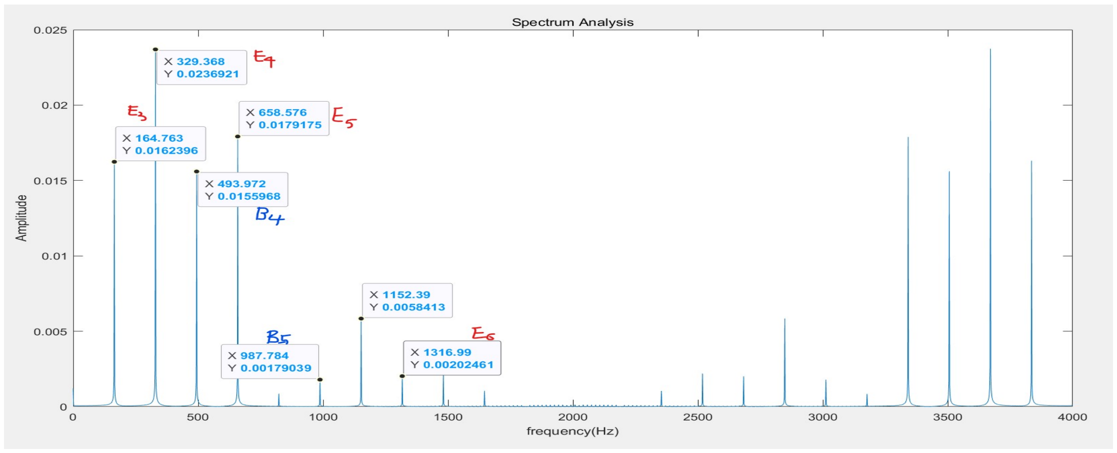


fig 9: 吉他声音信号的谱分析(时域拷贝30次;标注部分音高))

由图中可以看出,这一段乐曲频谱主要组分对应于E调(观测到了E3:164.76Hz; E4:329.37Hz; E5:658.58Hz这些频率)。

此外,还能观测到少量B调频率:B4:494.0Hz;B5:987.8Hz. 频谱分系出的数值与参考资料中的数据高度相符,说明这段音频以E调为主。**基频是164.76Hz,属E3调**。

分析时域数据拷贝对频谱分析效果的影响

有限支集信号f(t)可以被理解为一个T-周期性信号f_T(t)与时间窗函数$W(t)$相乘的结果。为简化数学分析,这里选时间窗为矩形脉冲。

$$f(t) = f_T(t) \cdot W(t) \tag{6}$$

对上述公式两侧做傅里叶变换

$$F[f] = F[f_T] * Sa(\Omega) \tag{7}$$

由此可见,**对有限信号的傅里叶变换相当于对周期性(真实)信号的傅里叶变换与Sa函数相卷积**如果我们可以增加有限信号的时域拷贝数N,则等效于时间窗的宽度正比于N地上升,时间窗也将逐渐逼近于直流信号。在频域上对应Sa函数的主瓣收缩,在函数列的意义下逐渐逼近于Dirac函数。考虑到

$$W \to \delta F[f] \to F[f_T] * \delta = F[f_T] \tag{8}$$

这说明如果增加时间窗宽度,就可以将有线信号的频谱在函数列意义下逐点收敛到无限信号的谱上,得到愈发逼真的结果。因此增加时域拷贝数可以减少频谱分析的失真。这一分析已经在实验上所证实(见上图fig5,fig6)。□

III 主观感受

太激动了,真实音乐相比电子合成要丰富许多!

2.4 自动分析出乐曲的音调和节拍

I 解决方案

Algorithm 1 自动分析乐曲音调节拍

```
对输入音频**求包络**,并根据**包络的极大值点**切分音频为若干段落(精度可控)
将每个音频段落使用**平均降噪**方法进行预处理
对每个音频段落进行**实时FFT频谱分析**
选取频谱中**强度最高的频率分量**作为此乐段的音调频率
将各个音频段落的**频率**映射为**音符记号**
将各个音频段落的时长**折算为节拍数**
将各个音符的唱名、时值等信息,成组输出到文档中
```

II 客观结果

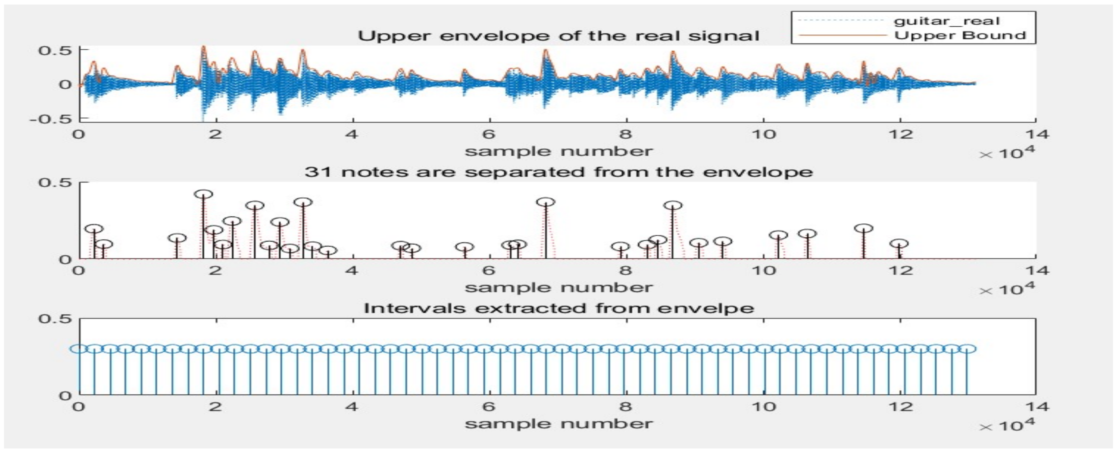


fig 10: 用包络分离节拍

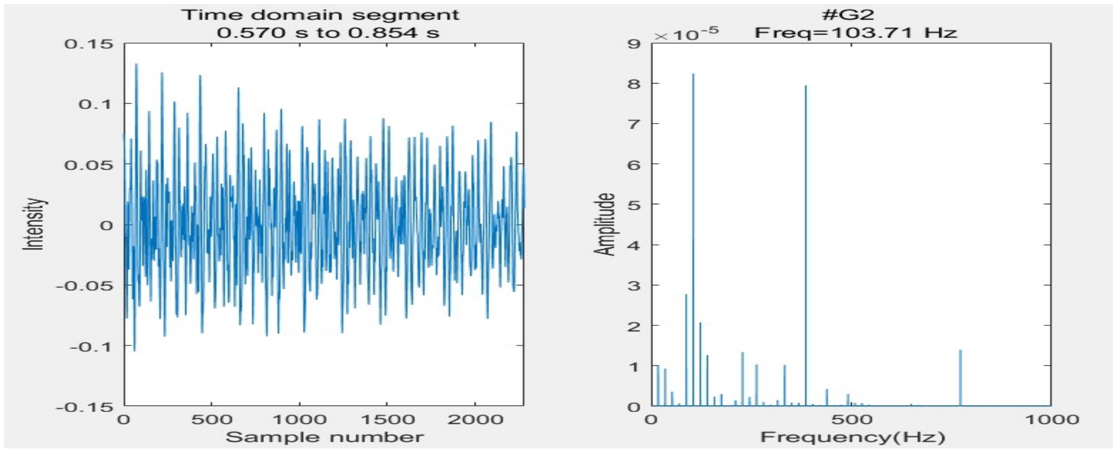


fig 11: 对各拍音乐进行实时频谱分析的一个样例
音调标注在频谱上放

这一问题最困难的地方并不是分离节拍,而是对带有和弦、每个音都具有泛音列的复杂信号进行频谱抽离和归类。

原生的算法会倾向于捕获和弦中的某些成分，但是未必是主音成分。这可能导致生成的旋律只是一个”伴奏”,而不是主要音调(以至于笔者初次尝试时感到结果是完全错误的)。

III 主观感受

整体上效果良好，但由于算法还不够复杂，分析精度可以进一步提高。

3 基于傅里叶级数的合成音乐

3.1 单泛音列合成《东方红》

I 解决方案

只需将(8)小问频谱结果排列成矩阵后送到(4)中的inst函数中。为保证可扩展性,我们引入了一个write_spectrum函数。频谱结果的一个样例是

II 客观结果 见此音频 DongFangHong Synthesized Single.

III 主观感受

效果并不很好，因为(8)小问只截取了吉他演奏中的非常短的片段,因此(8)并未捕获吉他的所有音调的泛音列。合成的乐曲相当于用吉他的一个单音的音色代替了不同音调的音色，听起来自然较为单调。但是相比标准设置,已经带有了很明显的音色区别。

3.2 多泛音列合成《东方红》

I 解决方案

这一问题在工程技术上比较复杂。

沿用之前的算法1,我们可以对每个音符进行频谱分析。

得到此音符的频谱后,我们在频带中抽取音调最强的频率所在泛音列的幅频响应，得到一个泛音列。

对所有此乐段中的音符重复上述操作,可以得到许多泛音列。

对这些泛音列进行排列、转唱名后,可以将一个泛音列矩阵输出到Microsoft Excel文档中,其中每一行代表一个基准音的泛音列的幅频响应。(使用sort.py文件)

在演奏乐曲时,我们根据每个输入的音符的音高,在泛音列矩阵中检索最相邻的音调。

如果确实存在比较接近的音调,则直接根据所在泛音列的幅频响应,合成谐波。

如果矩阵中已知的基准音都与此音调相距较远,则用与之最近的基准音所在泛音列的幅频特性,近似合成此音调。

最后加入包络处理,并逐个音调地播放音频,即可得到吉他音色的《东方红》。

II 客观结果

详见此视频DongFangHong synthesized。

III 主观感受

相比简单正弦波,利用从乐曲中抽取的音调信息更能体现出音色的厚重，比如合成音乐里可以明显听出吉他的低音音色和高音音色交相辉映，能够明显听出泛音列的一些结构。而且不同的音的泛音结构不尽相同,比单音合成更加丰富。

这说明算法是基本成功的。

4 附录

4.1 数值方法进行频谱分析的算法推导

Definitions and notations:

analogue signal (continuous): $f(t)$

analogue Fourier transform (continuous): $F(\omega)$

$F(\omega) := \int_{-\infty}^{+\infty} f(t)e^{-jwt}dt.$

use discrete sampling to estimate continuous signals:

$f(t)$: time domain sample region total length T

time domain sample number N

time domain sample interval $\Delta t = \frac{T}{N}$

time domain sample rate $f_s = \frac{N}{T}$

In practice, we acquire the value of N from the length of digital signal while T is resumed via $T = \frac{N}{f_s}$

time domain sample initial point t_1

time domain sample signal

$\tilde{f}[n; w] := f(t_1 + n\Delta t)e^{-jw(t_1+n\Delta t)}$

time domain sample points

$\{t_1, t_1 + \Delta t, ...t_1 + (N-1)\Delta t\} = \{t_1, t_1 + \Delta t...t_1 + T - \Delta t\}$

$F(\omega)$: frequency domain sample region total length Ω

frequency domain sample number K

frequency domain sample interval $\Delta\omega = \frac{\Omega}{K}$

frequency domain sample initial point ω_1

discretization of frequency: $\omega \to w_1 + k\Delta w$

frequency domain sample points:

$\{w_1, w_1 + \Delta w, ...w_1 + (K-1)\Delta w\} = \{w_1, w_1 + \Delta w...w_1 + \Omega - \Delta w\}$

Discrete estimate of Fourier Transform

$\forall\omega \in \{w_1, w_1 + \Delta w...w_1 + \Omega - \Delta w\}$

$F(\omega) = \tilde{F}(w_1 + k\Delta w) = \sum_{n=0}^{N-1} f(t_1 + n\Delta t)e^{-j(w_1+k\Delta w)(t_1+n\Delta t)}(\frac{T}{N})$

N-point FFT(DFT) of a signal

$fft_N(x[n]) = \sum_{n=0}^{N-1} x[n]e^{-j(\frac{2\pi}{N})nk}$

(Simulation Theorem of Fourier Transform)

if $N = K, \Omega T = 2\pi N$:

In the sense of discrete estimation

$F(\omega)e^{j\omega t_1} \approx \frac{T}{N}e^{j\omega_1 t_1} fft_N(fe^{-j\omega_1 t})$

(when fft is applied to a continuous signal, it means we first perform a N-point sample over a length of T, then FFT to the discrete sample.)

This means

$ifN = K, \Omega T = 2\pi N = 2\pi K:$

i.e. **If we sample the same amount of points in time and frequency domains: N=K, and that the sample interval in frequency domain is the same as the inverse of the total length of time domain:** $f_s = \frac{1}{T}$

Then for $\forall\omega \in \{w_1, w_1 + \Delta w ... w_1 + \Omega - \Delta w\}:$

$F(w) \approx \frac{T}{N} e^{j(w_1 - w)t_1} FFT_{N,T}[f(t)e^{-jw_1 t}]$

where the points sampled in time domain are

$\{t_1, t_1 + \Delta t ... t_1 + T - \Delta t\}, \Delta t = \frac{T}{N}$

Proof of the simulation theorem:

for each discrete sample point in LHS:

$F(w)e^{jwt_1} = F(w_1 + k\Delta w)e^{j(w_1+k\Delta w)(t_1)} = \sum_{n=0}^{N-1} f(t_1 + n\Delta t)e^{-j(w_1+k\Delta w)(t_1+n\Delta t)}(\frac{T}{N})e^{j(w_1+k\Delta w)(t_1)} = (\frac{T}{N})\sum_{n=0}^{N-1} f(t_1 + n\Delta t)e^{-j(w_1+k\Delta w)(n\Delta t)}$

for each point in RHS:

$RHS = \frac{T}{N}e^{jw_1 t_1}\sum_{n=0}^{N-1} f(t_1 + n\Delta t)e^{-j(w_1)(t_1+n\Delta t)}e^{-j\frac{2\pi}{N}nk} = \frac{T}{N}\sum_{n=0}^{N-1} f(t_1 + n\Delta t)e^{-j(w_1)(n\Delta t)}e^{-j\frac{2\pi}{N}nk}$

$ln(\frac{LHS}{RHS}) = j(\frac{2\pi}{N}nk - k\Delta w n\Delta t) = jkn(\frac{2\pi}{N} - \Delta w\Delta t)$

The constraint says that $\Omega T = 2\pi N\ so \Delta\omega\Delta t = \frac{\Omega}{K}\frac{T}{N} = \frac{2\pi N}{KN} = \frac{2\pi}{K}$

$\frac{1}{2\pi jkn}ln(\frac{LHS}{RHS}) = \frac{1}{N} - \frac{1}{K}$

The constraint also says that $N = K$, this impels the above line to be zero, which means $LHS = RHS$ □

4.2 所有源码

The first task:

```
%main.mlx
%Main function of the first task.
%speed, in BPM
tempo=60;
%instrument
instrument_id="std";%"guitar";%"std";"organ"
instrument_file="harmonic_guitar.txt";
know_spectrum=true;
%duration of each note
duration=1;
%sample/second
Fs=8192;
%loudness(amplitude)
A_b=1;
%key
base_key={'N','E',4};
%music sheet (in simplified form)
sheet_name="DongFangHong.txt";%"back.txt";

%uniform pause length
```

```
    unif_p_l=0.02;

    %load the sheet and instrument
    sheet=readmatrix(sheet_name);
    [freq,dur]=simplified2pitch(sheet,base_key);

    [n_overtones,amps]=inst(instrument_id);
    total_num_notes=size(freq);

    %envelope correction to attenuate the crack sound between notes.
    alpha=5.0;
    ratio=0.1;
    %env=@(t) exp(-alpha.*t);
    env_hd=@env;

    for i=1:total_num_notes
        %pause for a fixed length for all the notes.
        pause(unif_p_l);
        %if we cut duration by half then the pitch will be cut lower by half,
        %so use constant duration here for each note.

        %t=linspace(0,duration,Fs*duration);
        t=linspace(0,duration,Fs*duration);

        %pitch of the base sound.
        p_b=freq(i);
        %disp(p_b);

        %create a sound wave with overtones imitating the instrument.
        if know_spectrum==false
            note=overtone(p_b,A_b,n_overtones,amps,t);
        elseif know_spectrum==true
            note=harmonic(p_b,instrument_file,t)
        end

        %kill the crack between notes.
        note=note.*env_hd(t,alpha,ratio);
        %note=resample(note,2,1);%one octave upward.
        %note=resample(note,1,2);%one octave downward.

        %play a note
        sound(note,Fs);
        %last for its duration.
        pause(dur(i)/(tempo/60));
    end
```

```
    %env.m
%the envelope of each note.
%t is a row matrix. alpha is the dampen factor
% return a exponential decaying envelope added to the second half of t.
%recommended value: alpha=4.0,  ratio=0.5
function y=env(t,alpha,ratio)

```

```
d1=round(length(t)*0.1);
d2=round(length(t)*(ratio));

t1=t(1:d1);
y1=t1/max(t1);

t2=t(d1+1:d1+d2);
y2=linspace(1,1,length(t2));

t3=t(d1+d2+1:end);
y3=exp(-0.0001*alpha.*(t3-max(t2)));

y=[y1,y2,y3];

figure
plot(y);
tit=sprintf("envelope \n alpha= %1.1f ratio=%1.1f",alpha,ratio);
title(tit);
ylabel("Intensity");
xlabel("time");
end
```

```
    %f_alter.m
% convert C# ---> 1.5    (natural name, sharp/flat)-->absolute position in
% this octave.
function y=f_alter(natural,alt)

    name=natural+0.5*alt;

    if (natural==3)
        if (alt==1)
            name=4;
        end
    end
    if (natural ==4)
        if  (alt==-1)
            name=3;
        end
    end
    %convert to the relative place w.r.t the C
    if name==1
        y=0;
    elseif name==1.5
        y=1;
    elseif name==2
        y=2;
    elseif name==2.5
        y=3;
    elseif name==3    %!!
        y=4;
    elseif name==4    %!!
        y=5;
    elseif name==4.5
```

```
        y=6;
    elseif name==5
        y=7;
    elseif name==5.5
        y=8;
    elseif name==6
        y=9;
    elseif name==6.5
        y=10;
    elseif name==7
        y=11;
    else
        disp("Erro: in f_alter, undefined name==");
        disp(name);
    end

end
```

```
%inst.m
%generat the sound characteristics of a given instrument
function [n_overtones,amps]=inst(name)
if name=="std"
    amps=[1];
    n_overtones=[1];
end

if name=="organ"
    amps=[1,0.2,0.3];
    n_overtones=size(amps,2);
end

if name=="guitar"
    A=readmatrix("spectrum_of_guitar.txt");
    %freq=A(:,1)';
    amps=A(:,2)';
    n_overtones=length(amps);
end
```

```



%overtone.m
%return a simple overtone of given base pitch.

function note=overtone(p_b,A_b,n_tone,amps,t)
p=p_b;
pitches=[p_b];
for i=2:n_tone
    pitches(i)=p*2;
    p=p*2;
end
note_waves=A_b*sin(2*pi*pitches'*t);
```

```
note=amps*note_waves;



%matrix: each row is a note at a freq.


end
```

The second task:

```
    % main_II.mlx
    % (6)(7)of the second task.
    load Guitar.MAT
    [guitar_real,fs]=audioread("fmt.wav");
    sound(guitar_real,fs);

    out=average_noise_reduction(realwave);
    plot_real_proc_average(realwave,wave2proc,out);

    [pitch,w,P]=spectrum_analysis(wave2proc,2,fs/2,"plot");

    write_spectrum(pitch,w,P,"spectrum_of_guitar.txt");
```

```
    %main_II_2.mlx
%the (8),(9) questions of the second task.
%extract beats and tunes from a raw music and write each note to file.
    load Guitar.MAT
    [guitar_real,fs]=audioread("fmt.wav");
    %sound(guitar_real,fs);
%output directory
    directory="sheet.txt";

%generate upper envelope of the sound track.
    L=length(guitar_real);
    resol_thrd=480;%resolution threshold
    [UB,~]=envelope(guitar_real,resol_thrd,'peak');

%plot the time domain graphs.
    figure
    subplot(3,1,1);
    hold on;
    plot(guitar_real,':');
    plot(UB);
    legend("guitar\_real","Upper Bound");
    title("Upper envelope of the real signal");
    xlabel("sample number");
    subplot(3,1,2);
    hold on;
    u=UB-mean(UB);
    u=u.*(double(u>0.05));
    plot(u,":","Color","r");

```

```
%find the argmaxs of envelope and plot it.
    [pks, argmaxs]=findpeaks(u);
    stem(argmaxs,pks,"Color",[0,0,0]);
    caption_beats=sprintf('%d notes are separated from the envelope',1+length(argmaxs));
    title(caption_beats);
    xlabel("sample number");

   hold
%construct intervals for each notes from the local maximums.
    argmaxs=[1;argmaxs;L];
    intervals=linspace(1,L,length(argmaxs)+1);
    intervals(1)=1;
    intervals(end)=L;
    LL=length(argmaxs)+1;
    for i=2:LL-1
        intervals(i)=round((argmaxs(i-1)+argmaxs(i))/2);
    end
    subplot(3,1,3);
    stem(intervals,linspace(0.3,0.3,length(intervals)));
    set(gca,'YLim',[0,0.5]);
    title('Intervals extracted from envelpe');
    xlabel("sample number");
    hold off
%cut the song into single notes
    pitch=[1:1:length(argmaxs)-2];
    dur=pitch;
    plot_width=floor(2*(LL-1)/4)+1;
    figure
    for i=1:LL-1
        raw_note=guitar_real(intervals(i):intervals(i+1));
        dur(i)=length(raw_note)/fs;

        %play each note(raw music)
        sound(raw_note,fs);

        %pre-process each note, then analyze their spectrums.
        wave2proc=average_noise_reduction(raw_note);
        [pitch(i),w,P]=spectrum_analysis(wave2proc,10,fs/2,"find");

        %plot time domain analysis of each note.
            subplot(1,2,1);%subplot(plot_width,4,2*i);
            plot(raw_note);
            caption_real = sprintf('%2.3f s to %2.3f s',intervals(i)/fs,intervals(i+1)/fs);
            title(caption_real);
        %plot freq domain analysis of each note.
            subplot(1,2,2);%subplot(plot_width,4,2*i);
            plot(w/(2*pi),P);
            caption = sprintf('Freq=%f Hz',pitch(i));
            title(caption);
        pause(2);
    end
convert_pitch_duration_to_name_into_file(pitch,dur,directory);
```

```
%average_noise_reduction.m
%reduce the non-linear noise of a time-series by simple averagin.
function out=average_noise_reduction(realwave)
raw_len=size(realwave,1);
n=10;
s=round(raw_len/n)+1;
L=(s*n);
resampled=resample(realwave,L,raw_len);
%subplot(3,2,3);
%plot(resampled);
smpls=reshape(resampled,s,n);
avg=mean(smpls,2);
avrg_stretch=repmat(avg,1,n);
avrg_flatten=reshape(avrg_stretch,L,1);
out=resample(avrg_flatten, raw_len,L);
end
```

```
%plot_real_proc_average.m
function plot_real_proc_average(realwave,wave2proc,out)
subplot(2,2,1);
plot(realwave);
set(gca,'YLim',[-0.2,0.25],'XLim',[0,250]);
title("real wave");

subplot(2,2,2);
plot(abs(out-wave2proc));
title("compare averaging and process");
set(gca,'YLim',[-0.2,0.25],'XLim',[0,250]);

subplot(2,2,3);
plot(wave2proc);
title("processed wave");
set(gca,'YLim',[-0.2,0.25],'XLim',[0,250]);

subplot(2,2,4);
plot(out);
title("averaging result");
set(gca,'YLim',[-0.2,0.25],'XLim',[0,250]);

end
```

```
%get_interval.m
%return the average length of each note
function interval_len=get_interval(arg)
arg_1=arg(1:end-1);
arg_2=arg(2:end);
intervals=sort(arg_2-arg_1);
plot(intervals);
m1=mean(intervals(2:15));
m2=mean(intervals(16:22));
m3=mean(intervals(23:end));
interval_len=round(mean([m1,m2/2,m3/3]));
```

```
end
```

```
%harm.m
%return the harmonious spectrum synthesize result of a given spectrum P(w)
function [H,valid]=harm(w,P)

[A,locs]=findpeaks(P(1:round(length(P)/2)));
f=w(locs)/(2*pi);
[~,argm]=max(A);
fm=f(argm);

H=rand(2,7);
H(1,1)=fm;
H(2,1)=0;

spectrum=[fm/4,fm/2,fm,fm*2,fm*4,fm*8];
if fm/4 <20
     spectrum=[spectrum(2:end),fm*16];
end

I=nrest(spectrum,f);

if ismember(-1,I)==0
%       valid=1;
     H(:,[2:7])=[f(I);A(I)'];
     H(2,2:end)=H(2,2:end)/(sum(H(2,2:end)));

end
```

```
function H=harmonic_analysis(w,P_1,eps)
P=P_1(10:floor(length(P_1)/4))*10000;
plot(w/(2*pi),P_1');
H=randn(2,5);

[P_m,argmax]=max(P);
f_m=get_freq(w,argmax);

w_m=w(argmax);
H(1,1)=f_m;
H(2,1)=0.0;
base=0;

[peaks,locs]=findpeaks(P);
L_p=length(peaks);
for i=300:L_p
     disp("peaks");
     disp(peaks)
     disp("locs");
     disp(locs);
     disp("get_freq(w,i)=");
     disp(get_freq(w,i));
     if (multiple(f_m,get_freq(w,i),eps)==1)
```

```
        disp(i)
        base=get_freq(w,i);
        disp("base");
        disp(base);
        disp("i=");
        disp(i);
        break;
    end
end

n=2;
for j=i:L_p
    if(multiple(get_freq(w,j),base,eps)==1)
        if n<6
            H(1,n)=get_freq(w,j);
            H(2,n)=peaks(j);
            n=n+1;
        end
    end
end

%H(2,:)=H(2,:)/(sum(H(2,:)));
disp("base:");
disp(base);
end
```

```
%in_range.m
function whether=in_range(x)

whether=0;
if(x>0)
    if(x<2000)
        whether=1;
    end
end

end
```

```
%is_int.m

function whether=is_int(x,eps)
if(abs(x-round(x))<eps)
    whether=1;
else
    whether=0;
end
end
```

```
function whether=multiple(y,x,eps)
whether=0;
if( abs((y/x)-round(y/x))<eps ==1)
    whether=1;
```

```
end
```

```
%nrest.m
function y=nrest(x,X)
%eps=0.02;
d=x;
y=x;
for i=1:length(x)
    [d(i),y(i)]=min(abs(x(i)-X));
%      if d(i)>eps
%           y(i)=-1;
%      end
end
end
```

```
%note2pitch.m
%N A 4 --->440 HZ
%return the absolute place of this note within its octave. (y)
function [pitch,y]=note2pitch(note)

alter=note{1};
name=note{2};
octave=note{3};

A4_pitch=440;


if isa(name,'char')
if name=='C'
    y=0;
elseif name=='D'
    y=2;
elseif name=='E'
    y=4;
elseif name=='F'
    y=5;
elseif name=='G'
    y=7;
elseif name=='A'
    y=9;
elseif name=='B'
    y=11;
else
    disp("Undefined Key:");
    disp(name);
end
end

if alter=='N'
    y=y;
elseif alter=='S'
    y=y+1;
```

```
elseif alter=='F'
     y=y-1;
end


shamt=(y-9)+12*(octave-4);

pitch=A4_pitch*(power(1.05946309,shamt));

end
```

```

%place2pitch
%convert the place of the note on the keyboard to absolute frequency.

function freq=place2pitch(place_in_octave,place_of_octave,base_key)

[base_pitch,base_place]=note2pitch(base_key);  %ok

shamt_name=place_in_octave-base_place;        %

shamt_oct=place_of_octave-base_key{3};

shamt=shamt_name+shamt_oct*12;

freq=base_pitch*power(1.05946309,shamt);

%to write in this ways gives the correct result but it seems not right?
freq=freq.*(base_pitch/261.63);

end
```

```
%freq2notem.m
%twvl_name: C----0     B----11%
%eight_name: C,
function [twvl_name,octave]=freq2note(pitch)

key=round(log(pitch/440)/log(2^(1/12)))+9;
twvl_name=mod(key,12);
octave=4+(key-twvl_name)/12;

end
```

```
%convert a frequency to the nearest music note.
function formal=pitch2formal(pitch)
key=round(log(pitch/440)/log(2^(1/12)))+9;
twvl=mod(key,12);
octave=4+(key-twvl)/12;
[seven,sf]=arrayfun(@twvl2seven,twvl);
formal=arrayfun(@simplified2formal,seven,sf,octave);
end
```

```

%convert a simplified sheet music to a list of frequencies and durations
%sheet: N x 4 matrix.
function [freq,dur]=simplified2pitch(sheet,base_key)

%columns of sheet:      natural_name|sharp/flat|octave|duration
%flat or sharp, convert to relative place w.rt. the C in this octave.

sheet(:,1)=arrayfun(@f_alter,sheet(:,1),sheet(:,2));

freq=place2pitch(sheet(:,1),sheet(:,3),base_key);



dur=sheet(:,end);

end
```

```


%twvl2seven.m
%b=arrayfun(@twvl2seven,a,'UniformOutput',false);
function [seven,sf]=twvl2seven(twvl)

switch twvl
    case 0
        seven=1;
        sf=0;
    case 1
        seven=1;
        sf=1;
    case 2
        seven=2;
        sf=0;
    case 3
        seven=2;
        sf=1;
    case 4
        seven=3;
        sf=0;
    case 5
        seven=4;
        sf=0;
    case 6
         seven=4;
        sf=1;
    case 7
         seven=5;
        sf=0;
    case 8
        seven=5;
        sf=1;
    case 9
        seven=6;
        sf=0;
    case 10
        seven=6;
        sf=1;
    case 11
        seven=7;
        sf=0;
    otherwise
        dsp("wrong twvl");
        dsp(twvl);



end
```

```
%wave2sheet.m
function wave2sheet(realwave,fs,directory)
L=length(realwave);
[UB,~]=envelope(realwave,2,'peak');

figure
subplot(2,1,1);
hold on;
plot(realwave,':');
plot(UB);
legend("realwave","Upper Bound");
title("The upper envelope of real signal");

subplot(2,1,2);
hold on;
u=UB-mean(UB);
u=u.*(double(u>0.05));
plot(u,":","Color","r");
title("Separated beats");

[pks, locs]=findpeaks(u);
stem(locs,pks,"Color",[0,0,0]);

locs=[1;locs;L];
pitch=[];
dur=[];
figure
for i=1:length(locs)-2
    dur(i)=(locs(i+2)-i)/fs;
    raw_note=realwave(i:locs(i+2));
    wave2proc=average_noise_reduction(raw_note);
    pitch(i)=spectrum_analysis(wave2proc,2,fs/2,"find");
end

output=[pitch',dur'];
writematrix(output,directory);
```

```
%convert_pitch_duration_to_name_into_file.m
%input two arrays that contain the pitch and duration of each note
%conver the pitch to notes.
%write the results to file.
%key: is the absolute position on the keyboard, with centrl C being 0.
%twvl is the note written in 12. C--0  B--11;
%seven: C--1  #C----1.5   B--7
%sf:sharp---#  flat----b
%formal_twvl: #C4
function convert_pitch_duration_to_name_into_file(pitch,dur,directory)
key=round(log(pitch/440)/log(2^(1/12)))+9;
twvl=mod(key,12);
octave=4+(key-twvl)/12;
[seven,sf]=arrayfun(@twvl2seven,twvl);
formal_twvl=arrayfun(@simplified2formal,seven,sf,octave);
output=[pitch',key',twvl',seven',sf',octave',dur',formal_twvl'];
writematrix(output,directory,'Delimiter','tab');
end
```

```
%sort_harm_file.m
% write data to the harmony file.
function sort_harm_file(harm_file)
file_ID=fopen(harm_file,'a+');
for i=1:3
    fprintf(harm_ID,"%s\t",H_1(i,:));
    fprintf(harm_ID,"\n");
end
fprintf(harm_ID,"\n");
fclose(file_ID);
end
```

```


#sort.py
#sort lines in alphabetic order.

A=[]
with open('harmonic.txt',mode='r') as f:
    for line in f:
        A.append(line.split('\t'))
    L=len(A)
    B=sorted(A)

    f.close()


with open('harmonic.txt',mode='w') as f:
    for line in B:
        b="\t".join(tuple(line))
        bb=b.split()
        b1=bb[0:7]
        b2=bb[7:14]
        b3=bb[14:21]

        b1="\t".join(tuple(b1))
        b2="\t".join(tuple(b2))
        b3="\t".join(tuple(b3))

        f.write("%s\n" % b1)
        f.write("%s\n" % b2)
        f.write("%s\n" % b3)

        f.write("\n")
    f.close()
```

```
%spectrum_analysis.m
%fn: Nyquist samping frequency
%P:  Power spectrum
function [freq,w,P]=spectrum_analysis(x,n_copy,fn,mode)
%duplicate in time domain.
f=repmat(x,n_copy,1);
%config
N=size(f,1);
T=N/fn;
t_1=1;
dt=floor(T/N);

Omg=floor(2*pi*N/T);
K=N;
w_1=1;
dw=(Omg/K);

t=linspace(t_1,t_1+T-dt,N);
w=linspace(w_1,w_1+Omg-dw,K);

f1=f.*exp(-1i*w_1*t');
F1=fft(f1);
F=(T/N)*exp(1i*w_1*t_1).*exp(-1i*w.*t_1)'.*F1;

P=abs(F).^2;

%find maximum frequency.
P1=P(10:floor(length(P)/4));
[v,arg]=max(P1);
freq=w(arg)/(2*pi);

if mode=="plot"
    figure
    plot(w/(2*pi),P);
    title("spectrum");
    xlabel("frequency(Hz)");
    ylabel("Amplitude");
    set(gca, 'XLim',[0,4000]);
```

```
end
```

```
%write_overtone_to_file.m
function write_overtone_to_file(H, file_name)

harm_ID=fopen(file_name,"a+");

H_1=[pitch2formal(H(1,:));H];
pitch=H_1(2,1);
H_1(2,1)=H_1(1,1);
H_1(1,1)=pitch;

for i=1:3
    fprintf(harm_ID,"%s\t",H_1(i,:));
    fprintf(harm_ID,"\t");
end
fprintf(harm_ID,"\n");

fclose(harm_ID);
end
```

```
%write_spectrum.m
%file format: "overtones/HZ \tRelative Amplitude"
%P: power spectrum of the signal
%w: frequency sample array.
%file_name: output directory.
%pitch: all the frequencies are the overtone family of pitch.
function write_spectrum(pitch,w,P,file_name)
%neglect low amplitude noises.
P_2=P.*(double(P>0.0003));
%find the peaks in the sound.
[pks,locs]=findpeaks(P_2);
%extract the angular frequencies of corresponding peaks,
%then convert them to frequencies.
freq=w(locs)/(2*pi);
%normalize the amplitudes.
pks=pks.*1000;
amp=pks/(sum(pks));
%write frequency-amplitude pairs to file.
fileID = fopen(file_name,'w');
writematrix([freq',amp],file_name,'Delimiter','tab');
fclose(fileID);
end
```

The third task:

```
    % main.mlx
% generation task 3
% speed, in BPM
tempo=60;
%instrument
spectrum=false%true;
instrument_id="std";%"organ""guitar" %
```

```
instrument_file="harmonic_guitar.txt";
%duration of each note
duration=1;
%sample/second
Fs=8192;
%loudness(amplitude)
A_b=1;
%key
base_key={'N','E',4};
%music sheet (in simplified form)
sheet_name="DongFangHong.txt";%"back.txt";

%uniform pause length
unif_p_l=0.02;

%envelope correction to attenuate the crack sound between notes.
alpha=3.0;
env=@(t) exp(-alpha.*t);

%load the sheet and instrument
sheet=readmatrix(sheet_name);
[freq,dur]=simplified2pitch(sheet,base_key);

[n_overtones,amps]=inst(instrument_id);
total_num_notes=size(freq);

for i=1:total_num_notes
    %pause for a fixed length for all the notes.
    pause(unif_p_l);
    %if we cut duration by half then the pitch will be cut lower by half,
    %so use constant duration here for each note.

    %t=linspace(0,duration,Fs*duration);
    t=linspace(0,duration,Fs*duration);

    %pitch of the base sound.
    p_b=freq(i);
    %disp(p_b);

    %create a sound wave with overtones imitating the instrument.
    if spectrum==false
        note=overtone(p_b,A_b,n_overtones,amps,t);
    elseif spectrum==true
        note=harmonic(p_b,instrument_file,t);
    end

    %note=resample(note,2,1);%one octave upward.
    %note=resample(note,1,2);%one octave downward
    %play a note
    sound(note.*env(t),Fs);
    %last for its duration.
    pause(dur(i)/(tempo/60));
end
```

```
    %harmonic.m
%inut base pitch, filename, time axis
%output a harmonic wave mimicing the instrument's style.
function tone=harmonic(pb,instrument_file,t)
A=readmatrix(instrument_file);
B=[A(:,1),A(:,10:end)];
B=[B(:,1:7),B(:,10:end)];

known_spec=B(:,1);
%estimate this pitch with its nearest neighbor.
[dis,id]=min(abs(pb-known_spec));

if dis <3  %this is the note, we know its spectrum.
    id=randi(id);%in case there are multiple options...
    freqs=B(id,2:7);
    %disp("know this note");
elseif dis>=3%this NOT the note, we DO NOT know its spectrum.
%We have to use the nearest note's amplitude distribution to estimate.
%however, the frequencies should be re-defined.This is a very raw estimate.
    freqs=[pb/4,pb/2,pb,pb*2,pb*4,pb*8];
    %disp("imitate this note");
end
    amps=B(id,8:13).^(0.5);%B gives the power spectrum, convert to amplitude.
    tone=amps*sin(2*pi*freqs'*t);
    %harmonic wave, without envelope.
    %the sum of the square of amps sum up to one.
end
```

4.3 参考资料

参考文献

[1] https://en.wikipedia.org/wiki/Signal_averaging

[2] https://en.wikipedia.org/wiki/Hilbert_transform

最后特别感谢清华大学物理系王向斌教授的《大学物理**A2**》课堂。他逼迫我提前学会了一些**MATLAB**的技巧,更重要的是,在他的课堂上我学会用希尔伯特变换求解解析曲线的包络,这构成了最后几问我分隔节拍的基本灵感。