信号与系统——MATLAB 综合实验

音乐合成实验报告

学号：2019011008

无 92 刘雪枫

2021 年 9 月 5 日

目录

一、实验目的

1. 了解基本的乐理知识；
2. 深入理解离散量与连续量的关系，理解离散傅里叶变换在信号处理中的应用；
3. 学习 MATLAB 工具，在合成音乐的过程中学会运用 MATLAB 进行频谱分析与信号处理；
4. 在动手实验的过程中提高实践能力以及解决问题的能力，体会信号处理的乐趣。

二、实验平台

本次实验使用 Windows 平台上的 MATLAB R2021a (64-bit) 进行实验。

三、目录结构

本次实验报告的目录结构为：

report.pdf：实验报告文件；

src：存放源代码文件目录；

wave：存放各个问题所得到的音频文件目录。

四、实验原理

（一）乐音

人类听觉可以感受到的声音大体上可划分为噪音、语音、乐音，等等。其中，乐音的基本特征可以用基波频率（简称“基频”）、谐波频率分量和包络波形来描述。乐音的声调由基频决定，而不同乐器发出的乐音的音色则由高次谐波分量决定。此外，包络也是描述乐音特性的重要因素。本实验将通过模拟乐器发出的谐

波分量和包络来合成电子音乐。

（二）时频分析

振动随时间的变化关系称作振动的时域波形；组成振动的各单频正弦波的强度与频率的关系称作振动的频域波形。时域波形$f(t)$与频域波形$F(\omega)$的关系由傅里叶变换式给出：

$$F(\omega)=\int_{-\infty}^{+\infty} f(t)\mathrm{e}^{-j\omega t}\,\mathrm{d}t$$

$$f(t)=\frac{1}{2\pi}\int_{-\infty}^{+\infty} F(\omega)\mathrm{e}^{j\omega t}\mathrm{d}t$$

以上是针对连续信号的情况。计算机不可能存储连续的信号，对此可以将连续信号离散化。即每隔一段固定的时间 T_s 对连续信号 $f(t)$ 进行抽样，得到离散序列 $x(n)$。每秒钟的采样次数：$F_s=\frac{1}{T_s}$ 称为“采样率”。如果信号是周期信号，则只需对整数倍个周期进行采样。设采样总时间为 T_1，则基频一定为 $f_1=\frac{1}{T_1}$ 的整数倍。设采样点数为 N，由此得到离散傅里叶变换：

$$X(k)=\sum_{n=0}^{N-1} x(nT_s)\mathrm{e}^{-jk\omega_1 nT_s}$$

由此便可得到离散周期信号的频谱，用于近似连续周期信号的频谱。通过对频谱进行分析和设置，我们就能够得到离散时域信号，进而合成音乐。

五、实验内容

（一）简单的合成音乐

1. 简单合成《东方红》

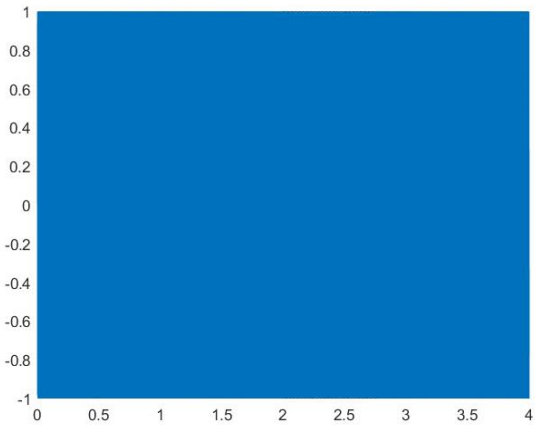
先用单频正弦波来演奏《东方红》，只需要按《东方红》曲谱逐个音符进行拼接即可。将 F 大调的各个音的频率储存在矩阵 tunes 中，将曲谱储存在矩阵 song 中，其中第一列代表音调，第二列代表持续时间（时间为 1 代表四分音符）。

拼接单频正弦波的关键代码如下：

```
for i = 1 : 1 : len
    f = tunes(song(i, 1));
    time_len = song(i, 2) * beat_len;
    t = linspace(0, time_len - 1 / Fs, Fs * time_len)';
    tmp_res = sin(2 * pi * f * t);
    res = [res; tmp_res];
end
```

由此得到的向量 res 即为音乐向量。

音乐向量的波形如下：

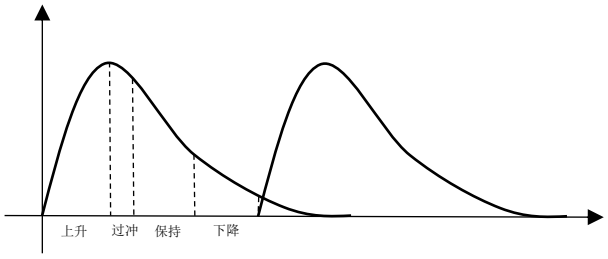


可以看到，音乐的幅度处处相等。播放音乐，发现在相邻两个音符之间存在“啪”的一声，非常影响音乐质量。究其原因，可能是相邻两个音符之间的相位不连续造成音符连接处存在高频分量导致的。

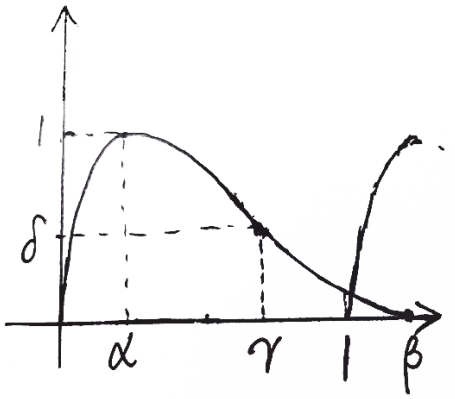
本问题的完整代码位于文件 hw_1_2_1_1.m 中，得到的音乐文件为 hw_1_2_1_1.wav。

2. 加入包络

为了解决相位不连续的问题，并模拟真实乐器的强度变化，需要给正弦波加上包络。为了让音符接触处的幅度为零，并且模拟声音先增强后逐渐减弱的过程，引入形如下图的包络函数：



为了拟合该包络函数，我们计划采用三段抛物线进行拟合：



上图中，0~α、α~γ、γ~β 三段分别为三段抛物线，从左向右一次记作 C_1、C_2、C_3。抛物线的 C_1 顶点位于 $(\alpha, 1)$ 处，且经过点 $(0, 0)$，因此可以写出它的方程：

$$C_1: y = -\frac{1}{\alpha^2}(x-\alpha)^2 + 1$$

同理，C_2、C_3 的顶点分别为 $(\alpha, 1)$ 和 $(\beta, 0)$，因此写出它们的方程：

$$C_2: y = B(x-\alpha)^2 + 1$$

$$C_3: y = C(x-\beta)^2$$

由于 C_2 过点 (γ, δ)，因此可以得到：

$$B = \frac{\delta - 1}{(\gamma-\alpha)^2}$$

由于包络函数应当尽量光滑，因此令在各段抛物线连接处满足函数值连续、函数值的一阶导数连续。因此 C_2 和 C_3 的连接处满足方程：

$$2B(\gamma-\alpha) = 2C(\gamma-\beta)$$

$$B(\gamma-\alpha)^2 + 1 = C(\gamma-\beta)^2$$

注意方程是过定的，因此 α、β、γ、δ 四个参量不独立。采用 α、β、γ 作为自由参量得到：

$$C = \frac{\gamma - \alpha}{\gamma - \beta} B$$

$$\delta = 1 + \frac{\gamma - \alpha}{\alpha - \beta}$$

合适选取参量 α、β、γ 即可得到包络函数。取：

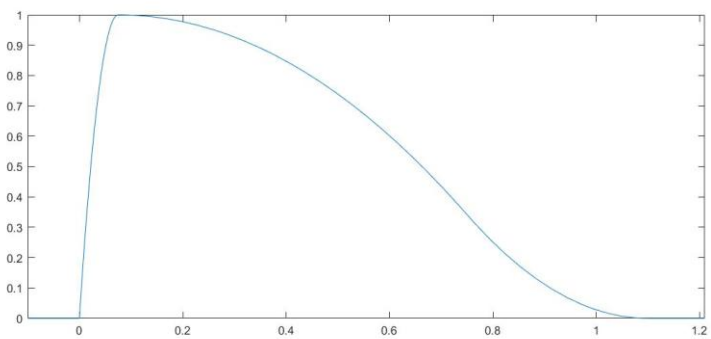
$$\alpha = 0.075, \beta = 1.1, \gamma = 0.75$$

得到包络函数，封装为 MATLAB 函数 `envelope`，写在函数文件 envelope.m 中。由于在拼接音符时会用到重叠部分的长度，因此函数也将 β 参量一并返回。函数代码如下：

```
function [y, beta_param] = envelope(x)
    alpha_param = 0.075;
    gamma_param = 0.75;
    beta_param = 1.1;
    delta_param = 1 + (gamma_param-alpha_param)./(alpha_param-
beta_param);
    A = -1./(alpha_param.^2);
    B = (delta_param-1)./(gamma_param-alpha_param).^2;
    C = (gamma_param-alpha_param)./(gamma_param-beta_param) .*
B;

    y = ...
        (x >= 0 & x < alpha_param) .* (A .* (x-alpha_param).^2
+ 1) + ...
        (x >= alpha_param & x < gamma_param) .* (B .* (x -
alpha_param).^2 + 1) + ...
        (x >= gamma_param & x < beta_param) .* C .* (x -
beta_param).^2;
end
```

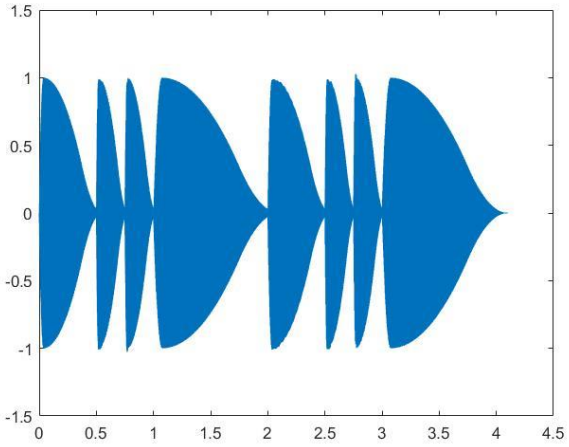
绘制出包络函数的图像如下：



将上一个实验中的单频正弦波与包络函数做乘积，然后将各个音符相互重叠地拼在一起（下一个音符在上一个音符未结束时便开始，开始位置位于包络函数自变量为 1 的位置）。用 `last_n_padding` 记录上一个音符的重叠部分的长度（采样点数），则关键代码为：

```
len = size(song);
len = len(1);
res = [];
[~, padding] = envelope(0);
last_nPadding = 0;
for i = 1 : 1 : len
    f = tunes(song(i, 1));
    time_len = song(i, 2) * beat_len;
    nTime_len = Fs * time_len * padding;
    t = linspace(0, time_len * padding - 1 / Fs, nTime_len)';
    tmp_res = envelope(t/time_len) .* sin(2 * pi * f * t);
    if (last_nPadding == 0)
        res = [res; tmp_res];
    else
        res = [res(1:end-last_nPadding); res(end-
last_nPadding)+tmp_res(1:last_nPadding);
tmp_res(last_nPadding+1:end)];
    end
    last_nPadding = round(nTime_len - Fs * time_len);
end
```

最终得到的声音波形如下：



播放音乐，可以听出原来两个音符之间“啪”的一生消失了，并且音乐有了

跳跃感，给人的感觉得到了明显的改善。

本问题的完整代码位于文件 hw_1_2_1_2.m 内，得到的音乐文件为 hw_1_2_1_2.wav。

3. 音乐变调

将音乐升高八度的方法非常简单。由于两个相邻的八度之间频率相差一倍，因此只需要在播放时将采样率扩大一倍即可：

```
sound(res, Fs * 2);
```

升高八度后的音乐文件为 hw_1_2_1_3_high.wav。同理将采样率缩小为原来的一半可以降低八度。降低八度后的音乐文件为 hw_1_2_1_3_low.wav。

如果要将音乐升高半个音阶，由于一个半个音阶的频率相差 $\sqrt[12]{2}$ ，因此可以用 `resample` 函数将采样率变为原来的 $\frac{1}{\sqrt[12]{2}}$ ，再按原采样率进行播放即可：

```
res_resamp = resample(res, round(Fs ./ 2.^ (1/12)), Fs);
```

得到的音乐文件为 hw_1_2_1_3.wav。

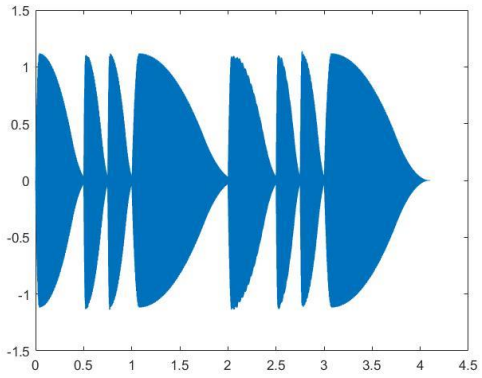
本问题的完整代码位于文件 hw_1_2_1_3.m 中。

4. 增加谐波分量

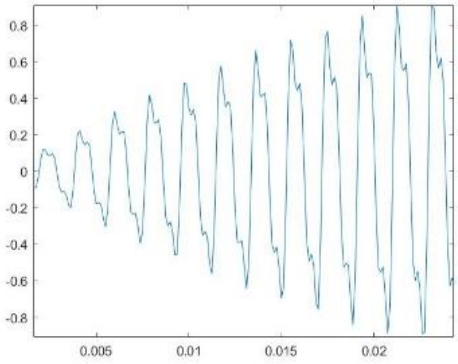
单频正弦波的声音难免让人感到有些刺耳，增加谐波分量可以改变音色，模拟出不同乐器弹奏的乐曲。增加谐波分量的方法很简单，只需要在生成单频正弦波的基础上加上一定强度的高频正弦波分量即可。设基波分量的振幅为 1，取二次谐波为 0.2、三次谐波为 0.3，关键代码如下：

```
tmp_res = envelope(t/time_len) .* (sin(2*pi*f * t) +
0.2*sin(2*pi*2*f * t) + 0.3*sin(2*pi*3*f * t));
```

得到的声音波形：



由于绘制的时间范围远大于周期数，因此和之前的波形无法看出区别。故取前 200 个采样点进行绘制：



可以看到，声音的波形不再是单频正弦波。由于有高次谐波分量，因此波形相对于正弦波有一些变形。

播放音乐，可以听出音乐确实类似于风琴声。

本问题的完整代码参见 hw_1_2_1_4.m，最终得到的音乐文件为 hw_1_2_1_4.wav。

5. 自选音乐合成

在自选音乐之前，为了方便起见，我们把前述功能封装成单独的函数。

我们把生成某大调的各个音的频率的功能封装成函数 `get_tunes`。该函数接收一个字符参数，表示大调，例如接收字符`'C'`表示 C 大调，返回各个音的频率列表。该函数代码位于文件 get_tunes.m 中。

我们再把生成音乐向量的功能封装成函数 `produce`，通过给定的乐谱向量、各个音的频率、采样率、节拍长度和谐波分量向量 `harm`（第 k 个元素为 k 次谐

波的强度）生成音乐向量。该函数位于函数文件 produce.m 中。

本问题我们选取的音乐是儿歌《宝贝宝贝》，俗称“两只老虎爱跳舞”。根据该音乐的简谱[①]输入音乐向量 song1（由于向量过长，故不在报告中写出，参见源代码）。经过处理得到音乐项链 res1：

```
res1 = produce(song1, tunes, Fs, beat_len, [1; 0.2; 0.1])
```

播放声音，但是给人的听觉相对比较单调，因此决定加入低声部进行和声。低声部采用低音 do、re 等按节奏重复，以增强音乐的感染力。将低声部产生的音乐向量命名为 res2，经过多次尝试得到最佳效果，将高低声部按 1:0.35 比例混合播放：

```
res = res1 + 0.35 * res2;
res = res * 0.5;
sound(res, Fs);
```

得到的效果相对之前有了明显的提升。

然后再调节音乐播放速度。起初我们使用的一拍的时间长度是 0.5 秒，感觉较慢，因此参照标准音乐[②]调节速度。经过测定，原音乐的一拍接近于 0.47 秒，因此将每拍长度 beat_len 改为 0.47 秒。

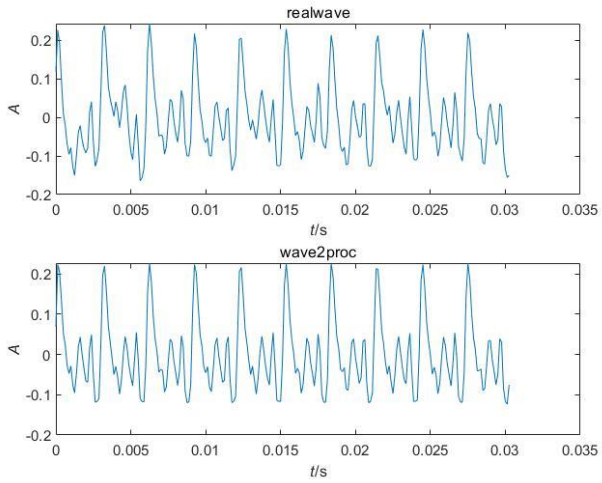
最后调整谐波分量。经过多次测试，决定令基波为 1，二次谐波为 0.2，三次谐波为 0.1，得到类似于钢琴的效果。

本问题的完整代码位于代码文件 hw_1_2_1_5.m 中，音乐文件为 hw_1_2_1_5.wav。

（二）用傅里叶级数分析音乐

载入 realwave 和 wave2proc 两个声音向量，并绘制波形：

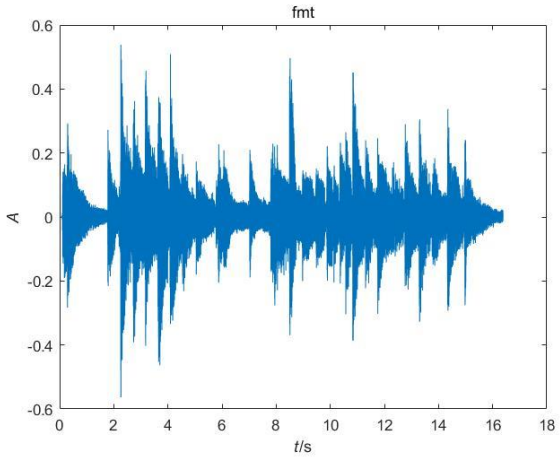
[①] 简谱来源于网络：https://www.bilibili.com/read/cv10111631/，2021 年 9 月 3 日。
[②] 标准音乐来源于网络：https://www.bilibili.com/video/BV1dU4y1W7qo?share_source=copy_web，2021 年 9 月 3 日。



可以看到，`wave2proc` 的波形非常均匀规律，而 `realwave` 的波形则不是非常规律。

6. 播放音乐 fmt.wav

播放音乐 fmt.wav，是真实的吉他声。使用 `audioread` 函数载入并绘制它的波形：



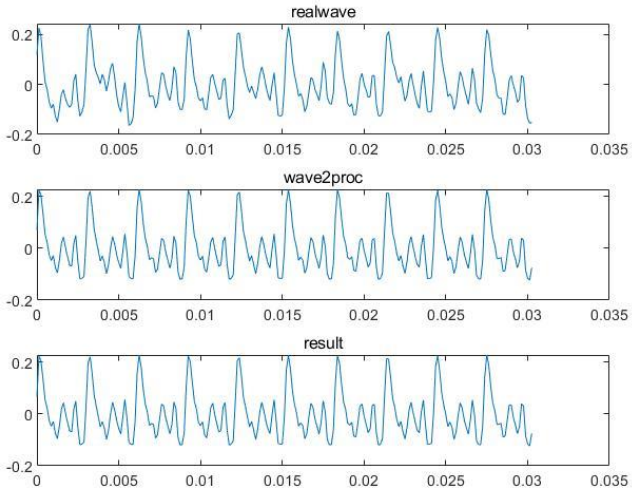
代码位于 hw_1_2_1_5.m 中。

7. 去除噪声

通过之前的观察可以看到，realwave 具有较大的噪声。由于噪声是无规律的，且音乐大约有 10 个周期，所以我们采取的办法是取该音乐 10 个周期的平均值。但是音乐并不是严格的整数倍的 10 个周期，因此增大其采样率，将采样率变为原来的至少 10 倍以保证采样点数为 10 的倍数，这里取 100 倍：

```
cont = resample(realwave, 100, 1);
group_len = round(length(cont) / 10);
res = zeros([group_len, 1]);
for i = 1 : 1 : 10
    res = res + cont((i - 1) * group_len + 1 : i * group_len);
end
res = res / 10;
res = [res; res];
res = [res; res; res; res; res];
res = resample(res, 1, 100);
```

得到结果后同时绘制出三个音乐的波形：

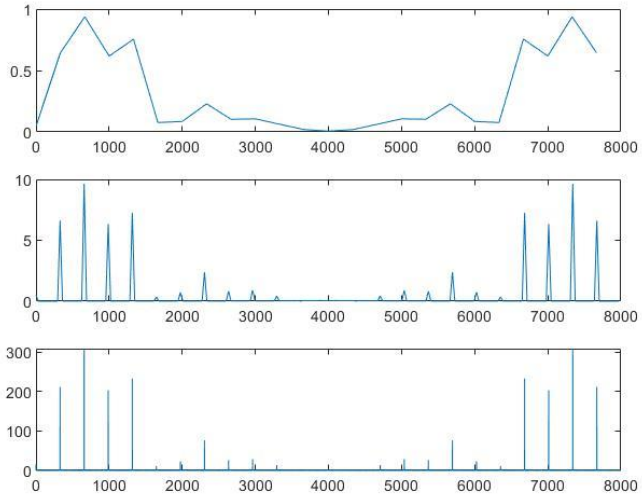


可以看到取均值处理后的音乐已经和 wave2proc 非常接近了。

本问题完整代码位于文件 hw_1_2_2_7.m 中。

8. 分析谐波分量

如下图：



使用 `fft` 函数对信号进行傅里叶变换。取给定的样本 `wave2proc` 中的一个周期进行傅里叶变换，得到上图的第一个图像；将整个信号进行傅里叶变换得到上图中第二个图像。由于周期较少，因此傅里叶变换可能不够准确因此将信号重复 32 次，再进行傅里叶变换并绘图：

```
subplot(3, 1, 3);
loop_time = 5;
for i = 1 : 1 : loop_time
    x = [x; x];
end
X3 = fft(x);
plot([0 : length(X3) - 1] / (length(x)/8000), abs(X3));
```

得到原图中的第三个图像。

由于采样率是 8000，`wave2proc` 的长度是 243，共有接近 10 个周期，每个周期大约 25 个采样点。因此基频约为 320Hz。对比十二平均律计算出的频率，大约是小字一组的 e1。

将基波分量归一化为 1，并将各次谐波分量打印出来：

```
delta = 10 * 2^loop_time;
n = [0 : length(X3) - 1];
res = abs(X3(mod(n, delta) == 0));
res = res / res(2);
disp([[0 : length(res) - 1]', res]);
```

打印即得到各次谐波分量的强度。由于离散序列的频谱是周期性的，因此无法得到原序列的各次谐波。而且注意到采样点数量有限，每个周期的采样点数量只有 25，因此最多有效谐波次数不会达到 (25-1)/2=12。故仅统计 11 次以内谐波的强度。且直流分量对振动没有影响，忽略直流分量，得到前 11 次谐波：

谐波次数	谐波强度	谐波次数	谐波强度
1	1.0000	7	0.3589
2	1.4572	8	0.1240
3	0.9587	9	0.1351
4	1.0999	10	0.0643
5	0.0523	11	0.0019
6	0.1099		

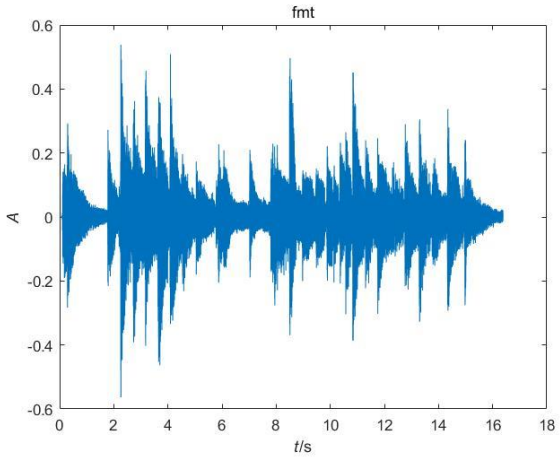
本问题的完整代码位于 hw_1_2_2_8.m 中。

9. 音调的分析与提取

本问题比较复杂，取 hw_1_2_2_9.m 作为启动脚本，调用其他函数。

(1) 自动划分音符起止时间

先绘制出 fmt 的音乐波形进行观察：4



可以看到，每个音符的特点是强度先上升到极值后缓慢下降。由于上升时间

极短，因此我们可以用极值点来代替音符的起始点。

对于连续信号，得到起始点的较好的方法其实是得到信号二阶导数的极值点（即信号由下降转为上升最强烈的点）。但是对于上升时间可以忽略不计的情况，用音符的强度极值点代替起始点，我们只需要找到一阶导数的极值点即可。

而对于离散信号，二阶导数对应于二阶差分，一阶导数对应于一阶差分，我们采用找信号强度的一阶差分的极值点的方法近似寻找音符的起始点。在寻找差分之前，我们需要对信号进行一些处理[①]。

i) 求位移绝对值

信号的强度是用能量度量的，而能量正比于信号的平方。因此我们需要将信号进行平方处理来求得能量。

但是，将信号进行平方，会拉大音乐较强部分与较弱部分的差距，导致将某些较强部分的毛刺作为节奏点，反而埋没了较弱部分的真正的节奏点。而考虑到在自变量大于零的情况下，平方函数是单调增的，并且我们只关心极大值点，暂时不关心信号的其他性质，因此我们采用绝对值函数代替平方函数。设原信号为 $x(n)$，有：

$$y_1(n) = |x(n)|$$

ii) 平滑包络求能量

将得到的幅度图像加窗平滑，求得图像的包络。这里采用汉宁窗，经过多次调试取窗的长度为采样率的十倍效果较好。设 $h_1(n)$ 为窗函数，则：

$$y_2(n) = \sum_{k=0}^{N_1-1} h_1(n) y_1(k-n)$$

iii) 差分求变化率

如之前所述，我们要求得一阶差分的极值点，因此对能量信号进行一阶差分：

$$y_3(n) = y_2(n+1) - y_2(n)$$

iv) 半波整流

[①] 本问题灵感的一部分来源于清华大学谷源涛老师《信号与系统》课程大作业《B 站，我来了!》（2021 年 6 月日稿）。

由于我么关心的是正的极值点，因此对于负值不予考虑，故将负值置为零：

$$y_4(n) = \max\{y_3(n), 0\}$$

v) 加窗平滑

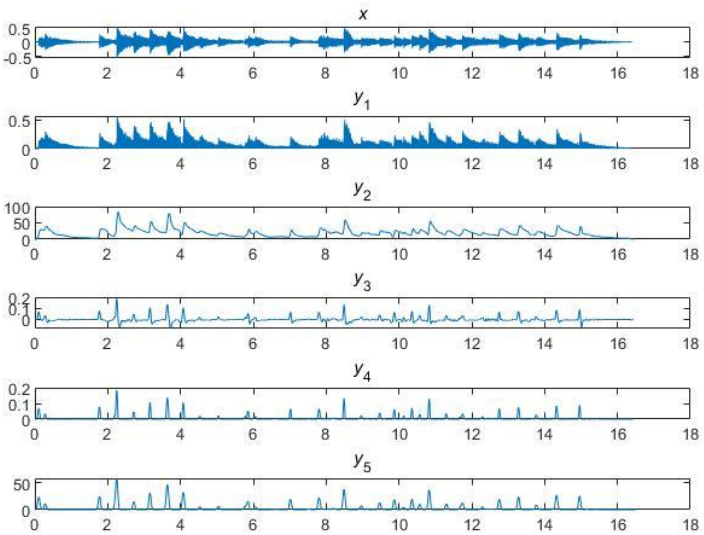
由于信号的毛刺较多，因此极值点较多，故预先对该信号加窗取平滑，以去除大部分毛刺造成的极值点。仍使用汉宁窗，经过多次实验，窗的长度取采样率的八分之一效果更佳。设窗函数为 $h_2(n)$，则有：

$$y_5(n) = \sum_{k=0}^{N_2-1} h_2(n) y_4(k-n)$$

根据上述思路，MATLAB 实现的关键代码如下：

```
y1 = abs(x);
y2WndLen = round(Fs / 10);
y2 = conv(y1, hanning(y2WndLen));
y2 = y2(round(y2WndLen/2):end);
y3 = diff(y2);
y4 = max(y3, 0);
y5WndLen = round(Fs / 8);
y5 = conv(y4, hanning(y5WndLen));
y5 = y5(round(y5WndLen/2):end);
```

绘制出各个序列的图像如下：



为了得到音符的起始点，我们需要得到 y_5 的极值点。因此，我们再对 y_5 取差分，得到 dy_5，再取得 dy_5 的由正值转为负值的点：

```
dy5 = diff(y5);
is_positive = (dy5(1) > 0);
primary_find_idx = [];
for i = 2 : 1 : length(dy5)
    if ((dy5(i) > 0) ~= is_positive)
        is_positive = ~is_positive;
        if is_positive == false
            primary_find_idx = [primary_find_idx; i];
        end
    end
end
```

显然这只是初步筛选，因为即使加窗平滑，我们仍然会有很多毛刺。因此我们需要对毛刺进行筛除。筛除的策略有两个：

i) 极值点处的值不能太小

ii) 极值点之间的距离不能过短

针对于 i) 我们采取的策略是，取所有筛查出的极值点中最大的三个点处的值的平均值的二十分之一作为阈值，达不到阈值的点要被筛掉，这部分的代码如下：

```
min_find_len = 3;
if length(primary_find_idx) < min_find_len * 2
    error('    Error: The music is too short!');
end
primary_find_val = y5(primary_find_idx);
primary_sort_res = sort(primary_find_val, 'descend');
level = mean(primary_sort_res(1:min_find_len)) ./ 20;
secondary_find_idx = primary_find_idx(y5(primary_find_idx) >=
level);
```

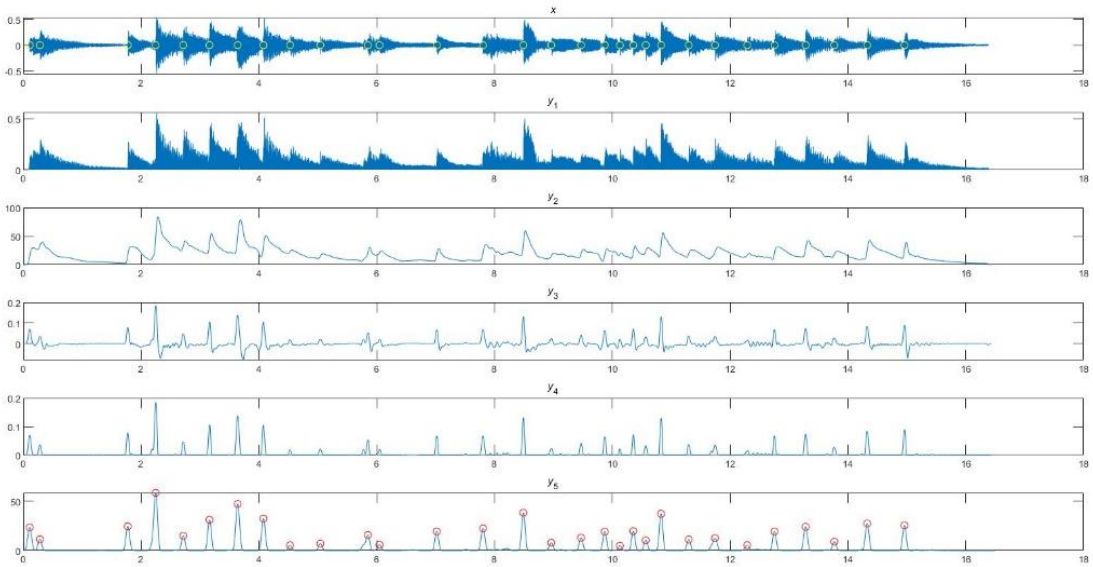
针对于 ii) 我们采取的策略是，若两个极值点之间的距离小于采样率的二十分之一，则只保留其中最大的。取二十分之一的理由是，对于一般的声音，人耳能分辨出来的两个声音的间隔约为 0.1s，我们将限制放宽到 0.05s 进行筛选，即

采样率的二十分之一。这部分的关键代码如下：

```
thirdary_find_idx = secondary_find_idx;
min_n_time_inteval = 0.05 * Fs;
i = 2;
del_cnt = 0;
while i <= length(thirdary_find_idx) - del_cnt
    if thirdary_find_idx(i) - thirdary_find_idx(i - 1) <
min_n_time_inteval
        if y5(thirdary_find_idx(i - 1)) <
y5(thirdary_find_idx(i))
            thirdary_find_idx(i - 1) = thirdary_find_idx(i);
        end
        thirdary_find_idx(i:end-1) =
thirdary_find_idx(i+1:end);
        del_cnt = del_cnt + 1;
    else
        i = i + 1;
    end
end
thirdary_find_idx = thirdary_find_idx(1:end-del_cnt);
```

之后我们便得到了我们选取的音符的起始点。

我们将提取出来的音符起始点分别绘制在 y_5 的图像和原信号 x 的图像上：



通过上图，尤其是第一个 x 的图像，我们可以看到我们几乎全部正确地提取到了各个音符的起始点！

根据自动选取的起始点，我们就可以自动划分单个音符了！

(2) 寻找基波频率

根据选取的起始点，将原音乐划分为多个片段，每个片段是单音符。为了得到更准确的频谱，将进行傅里叶变换时将每个音符重复 1024 次再进行傅里叶变换：

```
this_x = x(this_b : this_e);
this_repeat = 10;
for j = 1 : 1 : this_repeat
    this_x = [this_x; this_x];
end
this_repeat = 2^this_repeat;
this_X = abs(fft(this_x));
```

由于直流分量对我们没有影响，因此预先扣除掉基波频率：

```
dc_comp = this_X(1);  % Exclude DC component
this_X = this_X(2:end);
```

因为对于一个音符来说，基波一般是比较强的，因此我们采用以下的寻找基频的方案：

i) 找到频谱中强度最大的频点；

ii) 以最大强度的三分之一为阈值，筛选出强度超过阈值的频点；

iii) 我们设定基波频率不超过最强频点的频率，并且最强频点是基波或基波的一个高次谐波，故最强频点一定是基频的整数倍。因此我们从筛选出来的频点从低到高开始遍历，如果遇到了接近于最强频点的整数分之一的频率，则选为候选基频，并记录最强频点除以该频率得到的倍数。在该倍数条件下寻找最接近于整数倍的频率，选为基频。但是考虑到可能会有接近于直流分量的频率也有可能会比较强，且对于一段令人感到舒适的乐音来说，高频分量不会过强，因此若最强频点是

所选频率的超过很多倍则忽略该频率。这里我们选取 20 倍。

经过上述思路，我们就可以近似得到每段音符的基频。

(3) 提取谐波分量

提取基频之后，我们对每段音符的基频对照按十二平均律所计算出的标准频率进行标准化，讲基频变为与之最接近的标准频率。此外，注意到本问题中音乐的频率不是很高，基频范围也不会很高，因此只计算 110Hz~1500Hz 的标准音的频率，应该可以囊括本次实验的所有基频。

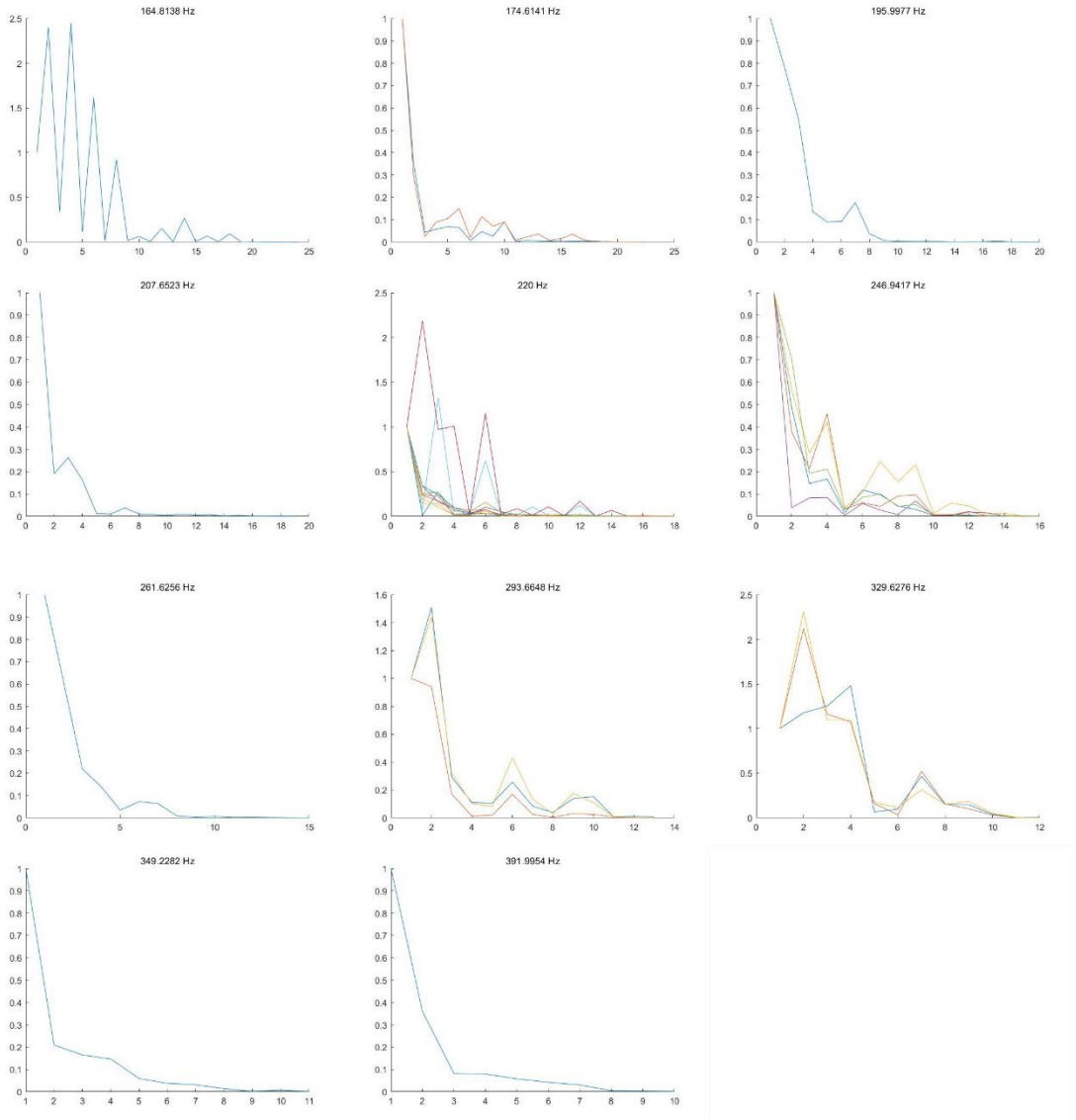
在这里编写了 `nearest_search` 函数用于在一个向量 `list` 中寻找与目标值向量 `target` 中的每个元素最接近的元素，并返回这些查找结果元素的下标组成的向量 `idx` 和元素的值组成的向量 `val`。关键代码如下：

```
idx = zeros(size(target));
for i = 1 : 1 : length(target)
    [~, idx(i)] = min(abs(list - target(i)));
end
val = list(idx);
```

然后再按照基频从大到小对音符进行排序，以便后续处理。

之后便提取各次谐波的强度。由于提取的基波频率可能有一定的误差，因此我们以各次谐波的频率为中心点，向左右分别沿伸一段小区间，在这个频率区间内取最大值，作为谐波分量的大小。中心点两边区间的长度定为相邻两个高次谐波距离的 0.1 倍。用单元数组 `component_record` 存放结果，单元数组的第一列是各个标准频率，第二列是一个矩阵，矩阵的每一列都代表提取的一次谐波分量样本，一列的第 k 个元素代表 k 次谐波分量的大小。

提取谐波分量后，以基频分量为 1 进行归一化，再去除未采集到样本的那些标准频率，只留下采集到样本的标准频率以及样本的谐波分量信息。将各个提取到的样本按频率分组并绘制，得到下图（绘此图的代码在源代码中已经注释掉）：



可以看到，对于同一个音符来说，大多数频率的变化特性是一致的，即频率分量随着频率增加而降低；且对于同一音符来说，不同的样本差别不是很大。

但是也有例外，例如 164.8Hz 的音符明显提取不准确。推测是由于它的实际基频是 329.6Hz，但是恰巧低八度的 164.8Hz 强度也比较强，因此错把 164.8Hz 当成了基频。这是因为它的偶次谐波都比较强，且偶次谐波的变化规律与 329.6Hz 的样本变化规律基本相同。其他的音符大概都比较正确。

由于乐曲的标准音的音调数量不是很多（比如上面我们一共只有 11 个音调），因此我们在自动提取谐波分量后，完全可以实现人工剔除明显错误的样本进行剔除或进行其他处理。

最后，为了减小误差，对于一个采集到了多组样本的音符，将该音符的所有样本取平均值，做为该音符的各次谐波分量。

本问题的完整代码位于 hw_1_2_2_9.m 脚本文件和 get_component.m 函数文件内。其中 hw_1_2_2_9.m 是启动的脚本，最后得到的谐波分量信息为 get_component 函数的返回值。

10. 使用 `wave2proc` 的傅里叶级数重奏《东方红》

先运行之前介绍过的的脚本 hw_1_2_2_8.m，得到各次谐波分量，储存在 `res` 中，然后将其作为参数传递给 `produce` 函数，即可得到音乐向量。播放音乐，音色确实有些类似于吉他，但是比较生硬。

本问题的完整代码位于 hw_1_2_2_10.m 中，得到的音乐文件在 hw_1_2_2_10.wav 中。

11. 使用 fmt.wav 中提取的谐波分量重奏《东方红》

注意到，我们从 fmt 提取的样本的基频都比较低（从上面的图中可以看出均低于 400Hz），而《东方红》的频率则比较高，无法直接应用提取的样本。因此，我决定将东方红低八度演奏，即频率均变为原来的二分之一，为此只需要将得到的 F 大调的标准音的频率除以 2 即可：

```
tunes = get_tunes('F');
tunes = tunes / 2;
```

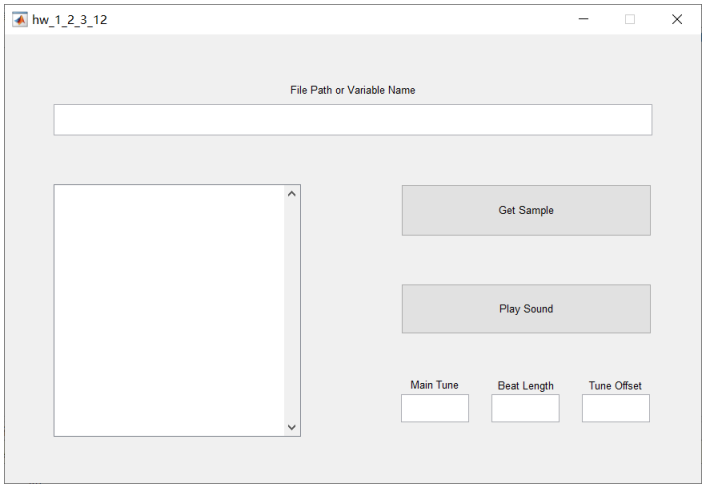
先运行脚本 hw_1_2_2_9.m，得到提取的样本。并将 `produce` 函数修改为 `produce_with_record` 函数，储存在 produce_with_record.m 文件中。修改后的函数能够实现根据给定的样本频谱来合成音乐向量，即将原来的 `harm` 参数修改为得到的谐波分量信息。

播放得到的声音，发现声音果然更像吉他了，给人的感觉也有了明显的提升。但是，播放过程中也有一些细小的杂音，推测是采样时的误差导致的。

本问题的代码位于文件 hw_1_2_3_11.m 中，得到的音乐文件为 hw_1_2_3_11.wav。

12. 图形界面操作

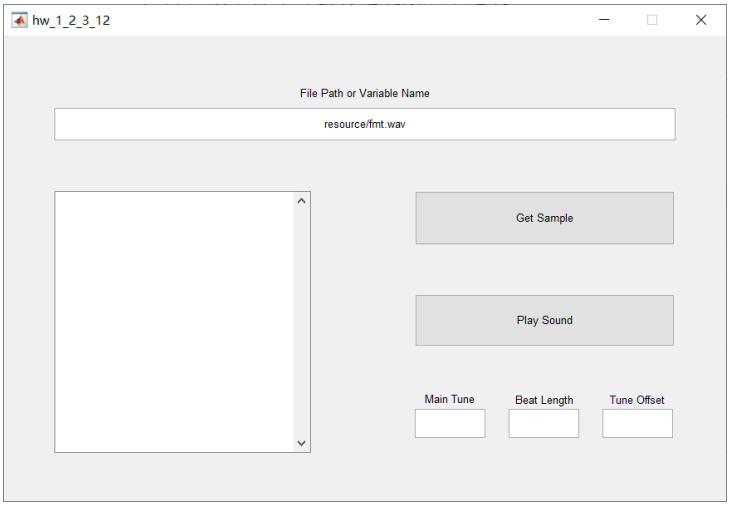
上述功能很容易封装为图形界面操作。我设计的图形界面布局如下：



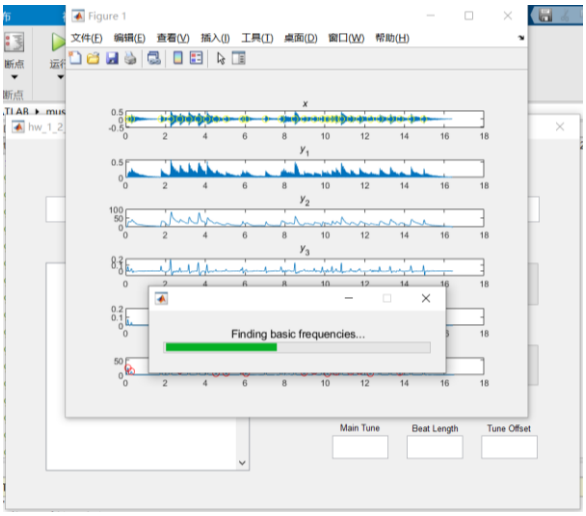
其中，最上面的输入框用来指定用于谐波信息采样的音频文件，或者储存乐谱信息的变量名；左侧的列表框用于展示采集到的谐波分量信息；“Get Sample”按钮用于开始采集谐波分量；“Play Sound”按钮用于播放音乐；按钮下方的三个输入框分别用于：指定音乐的基调（用一个大写字母表示大调）、每拍的时长（单位为秒）、是否声调或降调播放（即将频率要乘的倍数填在该输入框内）。此外在读取样本时会有进度条展示进度。

下面用图形界面演示上述功能：

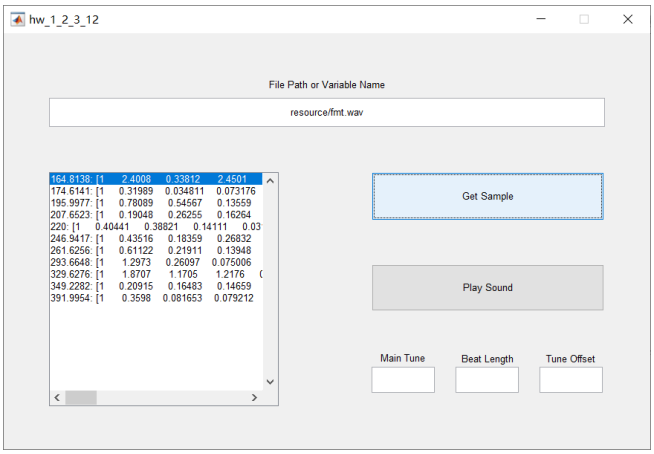
先在输入框内要读取的文件位置（本次实验为 resource/fmt.wav）：



然后点击“Get Sample”按钮开始采样，等待进度条：

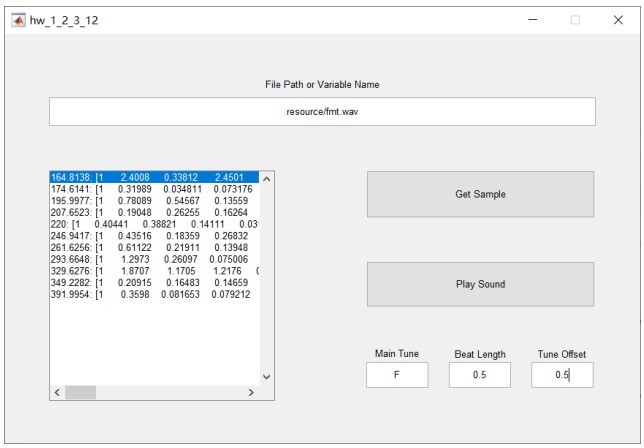


如果采样成功，则在左侧的列表框会显示采集到的样本信息：



正如图所示，列表框内最左侧的数字时基频，后面的方括号中的第几各元素就代表几次谐波分量。

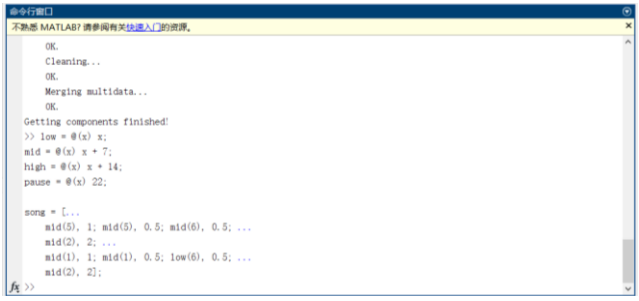
然后，设置音乐的参数。《东方红》是 F 大调，因此“Main Tune”设为“F”；每拍设置为 0.5 秒，因此“Beat Length”设为“0.5”；由于我们要降八度播放，因此频率应当除以 2，即乘 0.5，因此“Tune Offset”设为“0.5”：



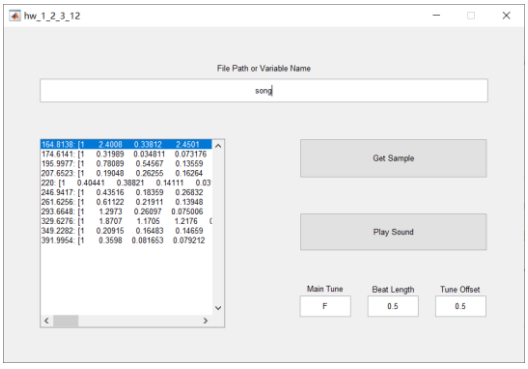
然后录入乐谱。像写脚本一样，将代表《东方红》乐谱的变量录入 MATLAB 内。方法是向命令行输入代码，我们将乐谱保存在变量 song 内：

```
low = @(x) x;
mid = @(x) x + 7;
high = @(x) x + 14;
pause = @(x) 22;


song = [...
    mid(5), 1; mid(5), 0.5; mid(6), 0.5; ...
    mid(2), 2; ...
    mid(1), 1; mid(1), 0.5; low(6), 0.5; ...
    mid(2), 2];
```



然后在最上面的输入框中输入保存乐谱的变量名，即“song”：



最后，点击“Play Sound”按钮即可播放音乐。

六、实验小结与未来展望

总体来说这次实验还是比较顺利的，我几乎独立完成了本次实验。在实验中，我对 MATLAB 工具更加熟练了，并且对《信号与系统》课程上所讲授的傅里叶

变换相关知识有了更深的理解，并亲身体会了使用 MATLAB 进行信号处理的过程。

但是实验还有很多不足之处。例如，对音频的采集还是不够准确。在我使用 GUI 界面对其他各种音乐采集并测试时，发现了一个比较严重的问题，就是采集到的谐波分量的低频段失真较为严重，尤其见于在极高次谐波例如十次谐波以上会有极高的分量。推测原因是对于一些音符来说，提取基频时错把很低频率的毛刺当做了基频分量。而由于频率很低，很容易就成为其他频率的整数倍分之一，因此容易错吧极低频率的毛刺作为基频，这是需要改进的。但是在高频段，采集的样本相对效果较好。

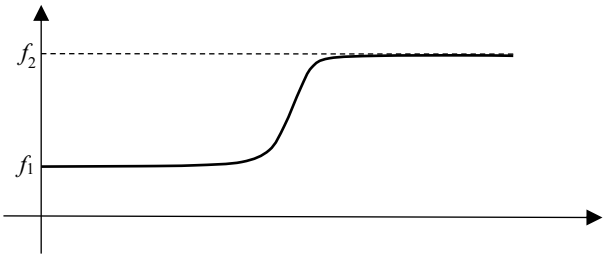
对于上面的问题，我认为一个解决方案就是我在前面所述的，在采集样本后人工对明显错误的样本进行去除或进行其他处理；另外就是改进筛选基频的算法，这点我认为还是比较难的。

此外，本实验还存在的问题是，播放音乐的方便性还需完善。目前乐谱的录入是通过代码，这样录入效率相对较低，也不是非常友好。我目前的未来改进方案是，设计一种表述乐谱的标记语言，将乐谱写在一个文本文件中，用 MATLAB 进行读取。我在以前曾经使用 C++语言设计过类似的表示音符的语言，利用下划线代表低音、“^”代表高音、方括号[]代表将其内部的音符二分，等等。但是限于我的能力，该语言尚不完善。但是可以考虑将该思想迁移到 MATLAB 中，并逐步完善。

关于乐音的包络也有改进的空间。本次实验所用的包络函数存在保持时间不长的问题，因此可以考虑更换包络函数，并且可以考虑根据不同的乐器采取不同的包络函数，甚至可以借鉴提取谐波分量的思想，从给定的样本音乐中提取包络函数，我认为这都是可行的方案。

此外还有一点，就是本次实验没有实现连音。乐谱中经常出现连音，即从一个音平滑过渡到另一个音。对此我能想到的可能方案有两个：一个方案是幅度连续：设计专门的用于连音包络函数，在需要连音的两个字符交界处使用“S”型分别将两个音符包络，一个上升一个下降实现互补；另一个方案是频率连续：将连音的所有音符统一使用一个大的包络函数，而频率的过渡在正弦函数内进行处理，以两个音连音为例，设计一个函数，自变量接收时间，因变量为频率，根据

时间，频率连续变化，初步设计该变化也是 S 型：



虽然本次 实验有很多不足之处，但是这次实验让我增长了新知识、巩固了学过的知识，提高了我的实践能力，还让我体会到了信号处理的乐趣。

最后，非常感谢老师和助教为本次实验的付出。没有老师的课上的讲解、助教热心的帮助，我也很难顺利完成本次实验，在此表示衷心的感谢。