MATLAB 音乐合成大作业

无15 向明义 2021012760

> 源代码及资源文件均放在 src/ 目录下，代码以 hw_x_x.m 命名，还有一些自定义函数。
>
> 生成的音频文件放在 results/ 目录下。
>
> report.md 中的图片放在 images/ 目录下。
>
> 软件版本为 MATLAB R2022b。

实验目的

- 增进对傅里叶级数的理解。
- 熟练运用 MATLAB 基本指令。
- 了解乐理和电子音乐的基本知识。

实验内容

0. 前置知识

首先要知道怎么合成do re mi。

一段时长1s的523Hz频率的正弦波，发出的就是小字二组c的do音，如果用8192Hz的采样频率，1s要采8192个点，用 t 表示时间采样点，则每个点上的音的幅度是 $\sin(2\pi ft)$。

这样一段代码可以生成一个do音：

```
Fs = 8192;
T = 0.5;
t = linspace(0, T, Fs*T);  % Fs*T是采样点数
f = 523.25;
y = sin(2*pi*f*t);
sound(y);
```

而幅度的大小就是音量的大小，如果给 y 套上一个包络，就可以得到音量随包络渐变的效果。

谐波就是把另外频率的波叠加上去。

1. 简单的合成音乐

1.1 东方红第一小节

考虑到乐曲中用到的一般不会超过选定调子内的低一组和高一组音，所以计算3*7=21个频率备用：

```
baseA = 220;
realBase = baseA * 2^(8/12);  % 1=F调
% 乐曲中用到的基本是低一组和高一组
diffs = [-12,-10,-8,-7,-5,-3,-1,0,2,4,5,7,9,11,12,14,16,17,19,21,23];
baseFreq = realBase*2.^(diffs/12);
mid = @(x) baseFreq(x+7);  % mid(1) = F
low = @(x) baseFreq(x);
high = @(x) baseFreq(x+14);
```

然后按照简谱写出需要的音调（第二列表示拍数）：

```
song = [ % 按照2拍分节
    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;
];
```

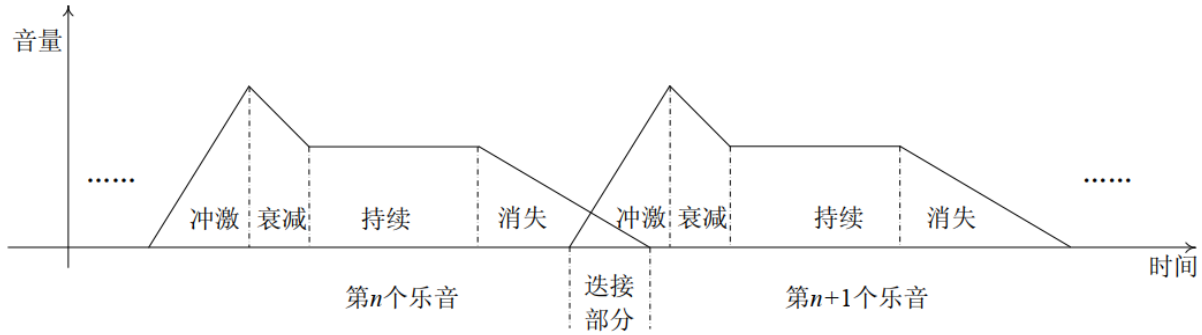
最后枚举每一行，将对应的频率和时间按照前置知识部分所述转换为采样数据拼接起来即可。

代码详见 `src/1_1.mlx`。对应音频文件在 `results/1_1.wav`。

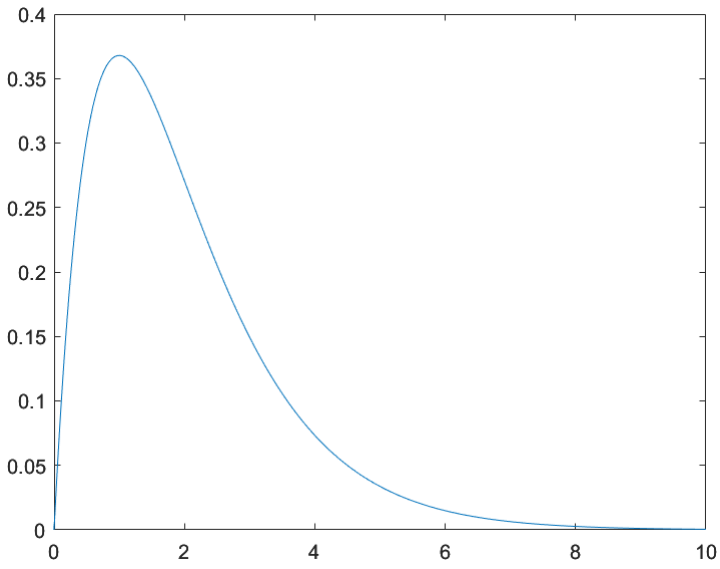
这样合成的音乐，音量大小处处相同，同时在两个音调切换处有明显的“啪”音。

1.2 加入包络

相邻乐音间的“啪”杂音是由于两个音频率不一致，在转换时产生了高频分量，为了消除它，需要给前一个音的末尾加上渐弱消音：



我们知道 xe^{-x} 的图像可以很好的满足这个特征曲线：



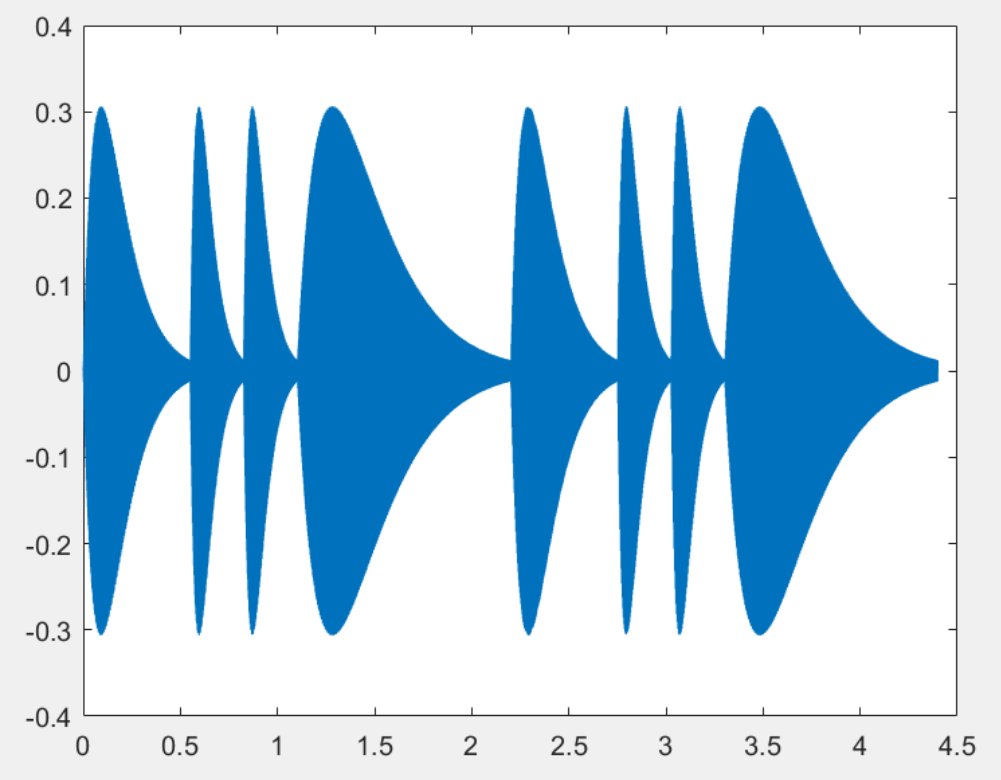
经过简单的调参，确定使用 $6xe^{-8x}$ 作为包络，使用的方法为将 t/T 作为 x 传入函数然后相乘。

此时啪声已经可以较好地消去，为了使乐音更有渐进感，将相邻两乐音做连接。具体方法为将原乐音延长一定比例，然后将后一个乐音与延长部分叠加。

关键改动如下：

```
shiftLen = 0;
for k = 1:rows
    f = song(k,1);
    T = beatTime * song(k,2) * shiftRatio;  % 这里对单音做延长
    t = linspace(0, T, Fs*T)';
    res = sin(2*pi*f*t) .* envelope(t/T);   % 这里乘上包络
    tunes = [
        tunes(1:end-shiftLen);
        tunes(end-shiftLen+1:end) + res(1:shiftLen);
        res(shiftLen+1:end);
    ]; % 拼接
    shiftLen = round(Fs * beatTime * song(k, 2) * (shiftRatio - 1));  % 取到延长部分的点数
end
```

最终波形：



1.3 改变音调

升降八度最简单的方法是在 `sound` 输出时将采样频率加倍或减半：

```
sound(tunes, Fs*2); % 升八度（频率加倍，时间减半）
```

```
sound(tunes, Fs/2); % 降八度（频率减半，时间加倍）
```

对原音乐进行 `resample` 重新采样，如果采样频率上升（采样点数会增多），但播放仍按原采样率播放，则相当于原音乐的频率降低了。

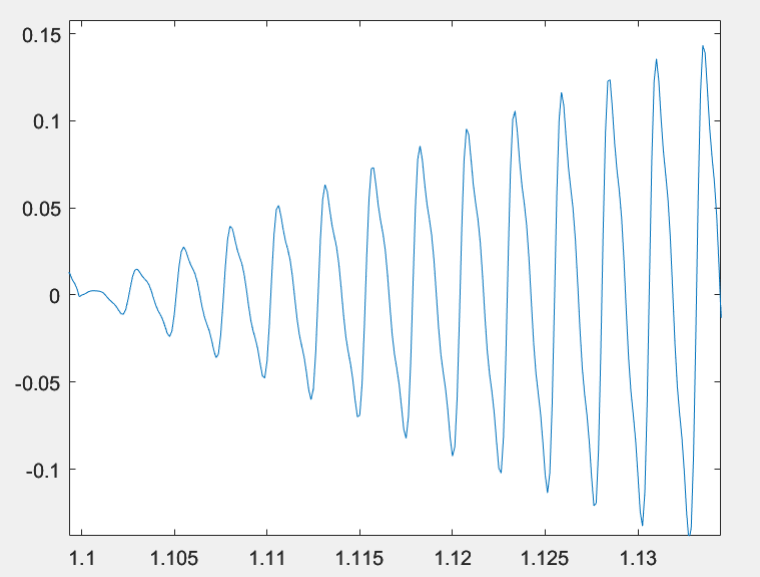
因此要升高半个音阶，则采样频率应降低 $2^{1/12}$：

```
tunes = resample(tunes, Fs, round(Fs*2^(1/12)));
```

1.4 加入谐波

谐波频率是基波的整数倍。假设基波幅度为1，设置二次谐波相对幅度为0.3，三次谐波相对幅度为0.15：

```
harmoConfs = [1, 0.3, 0.15];
res = sin(2*pi*f*t*(1:length(harmoConfs))) * harmoConfs' .* envelope(t/T) ;
```



将波形放大查看，可以发现不再是完全的单频波。

1.5 自选音乐合成

选用《传说的世界》前半部分。为了更好听，把重叠部分扩到了2倍，因此 shiftLen 部分做了一定修改。

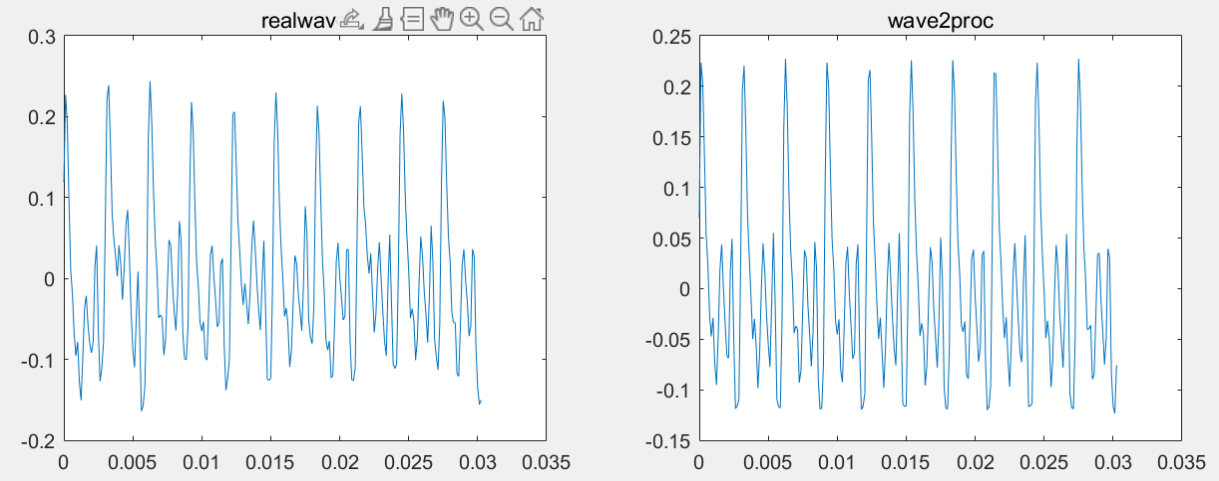
参见 `src/hw_1_5.m` 及 `results/1_5.wav`

2. 用傅里叶级数分析音乐

2.1 播放fmt.wav

使用 `audioread('fmt.wav')` 载入，再以 8K Hz采样率播放。效果明显更贴近真实音效。

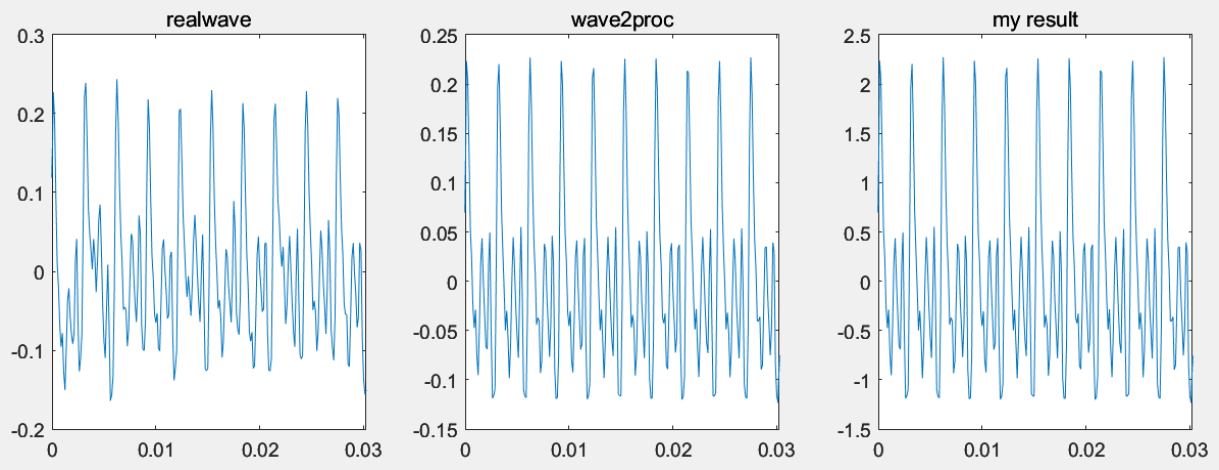
2.2 消除噪声



这是一段周期性的音频，由于噪声是偏随机的，因此多次取平均可以消除噪声。

realwave中有约10个周期，将这10个周期取平均，采用 resample 的方法，设原音频采样长度为 len，将原音频以10倍频率重新采样，这样新采样数据的每 len 个点就是原音频的一个周期，叠加之后复制10次，再以1/10频率重新采样即得到 wave2proc。

```
wav = resample(realwave, 10, 1);    % 10倍频重新采样
len = length(realwave);
res = zeros([len 1]);
for k = 1:10
    res = res + wav((k-1)*len+1:k*len);  % 周期叠加
end
res = repmat(res, 10, 1);
res = resample(res, 1, 10);
```

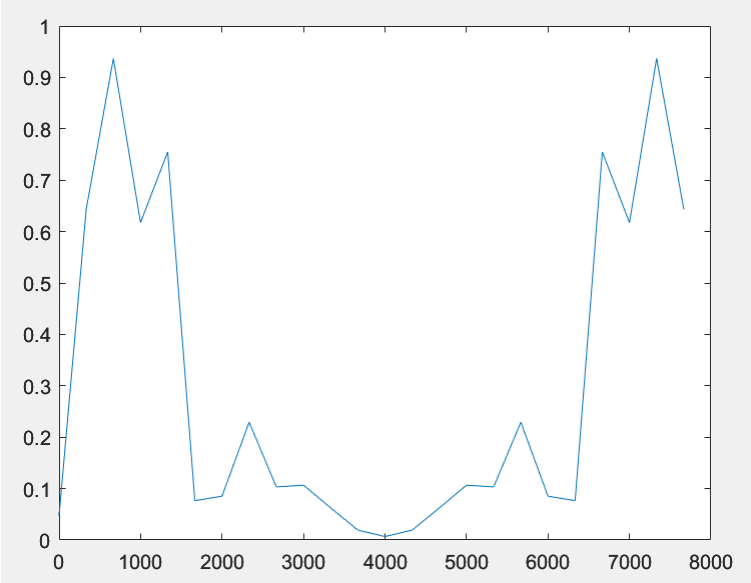


2.3 傅里叶分析谐波分量

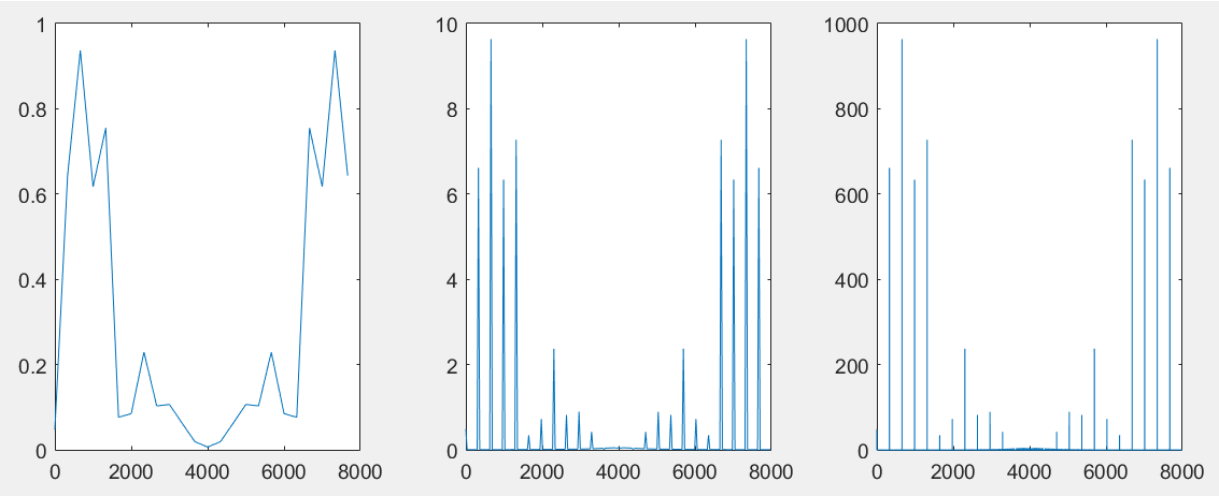
离散傅里叶变换与分量的关系：[深入理解离散傅里叶变换(DFT) - 知乎 (zhihu.com)]

```
X1 = fft(res(1:N));
plot([0:N-1]*Fs/N, abs(X1));
```

取一个周期做快速傅里叶变换，得到：



由傅里叶变换的知识可知，这样得到的是各频率分量幅度的包络。为了得到更准确的变换，我们把信号重复若干次再做变换，以下是分别取1、10、1000个周期得到的变换图：



随着时域周期数增加，频谱越来越近似冲激函数。

接下来根据变换提取各分量的频率以及幅度，由于采样频率为8000Hz，所以谐波频率不超过4000Hz。

首先观察一下，基波分量在 200Hz 到 400Hz 之间，取范围内幅度最大值为基波频率：

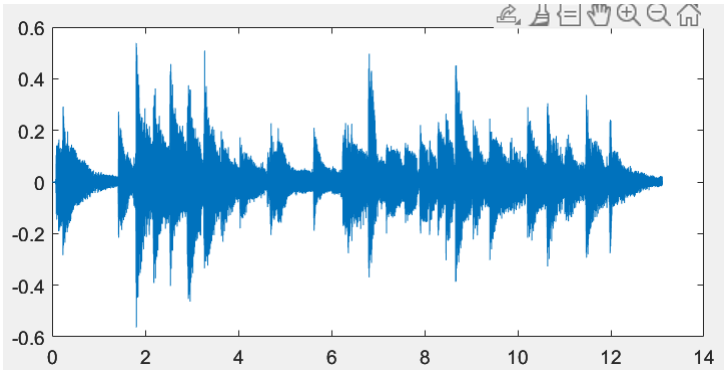
```
Rg = round([200,400]/Fs*N);
[~, base] = max(abs(X(Rg(1):Rg(2))));
base = base + Rg(1) - 2;  % 注意matlab下标是从1开始的
freq = base/N*Fs;
```

得到 $freq = 329.22\ Hz$ 。

然后按照基频的整数倍取点，得到各次谐波分量的相对强度：

```
    0    0.0739
    1    1.0000
    2    1.4572
    3    0.9587
    4    1.0999
    5    0.0523
    6    0.1099
    7    0.3589
    8    0.1240
    9    0.1351
   10    0.0643
   11    0.0019
   12    0.0058
```

2.4 自动分析乐曲音调和节拍



给出这样一段音频，根据波形自动划分它的节拍，从而得到每一个音调片段，对片段做傅里叶分析得到它的基频、谐波幅度等信息保存下来，从而可以合成与该乐器相似的曲调。

2.4.1 划分节拍

划分节拍的方式参考了谷源涛老师2021年信号与系统大作业中的节奏点划分方法：[Project-for-Signals-and-Systems-2021/Project2021.pdf at main · zhangzw16/Project-for-Signals-and-Systems-2021 (github.com)]

总体分为五步：平方、加窗、差分、半波整流、提取峰值。下面逐步阐释：

- 平方

  平方得到能量（这一步换成绝对值好像也行）

  ```
  y1 = wav.^2;
  ```

- 加窗

  这里使用matlab的barthannwin窗，窗的长度取采样率的1/10，时域做卷积，即频域相乘。加窗后得到包络。（为啥会得到包络，原理没太懂

  ```
  hann = barthannwin(round(Fs/10));
  y2 = conv(y1, hann);
  ```

- 差分

  ```
  y3 = y2(2:end) - y2(1:end-1);
  ```

- 半波整流

  只需要正的极值点。

  ```
  y4 = y3.*(y3>0);
  ```

- 提取峰值

  我们取现在图中的峰值作为该音的起始点，使用 `find_peak` 函数提取极值点进行初筛，然后过一下两个判定：

  - 极值点的幅度不能太小。选取整个音频第二大与第三大的平均值的1/29作为阈值。
  - 两个极值点不能太近。人耳能分辨的两音符间距大约0.1s，选取采样率的1/15作为阈值，若两极值点间距小于阈值，则保留幅度更大者。

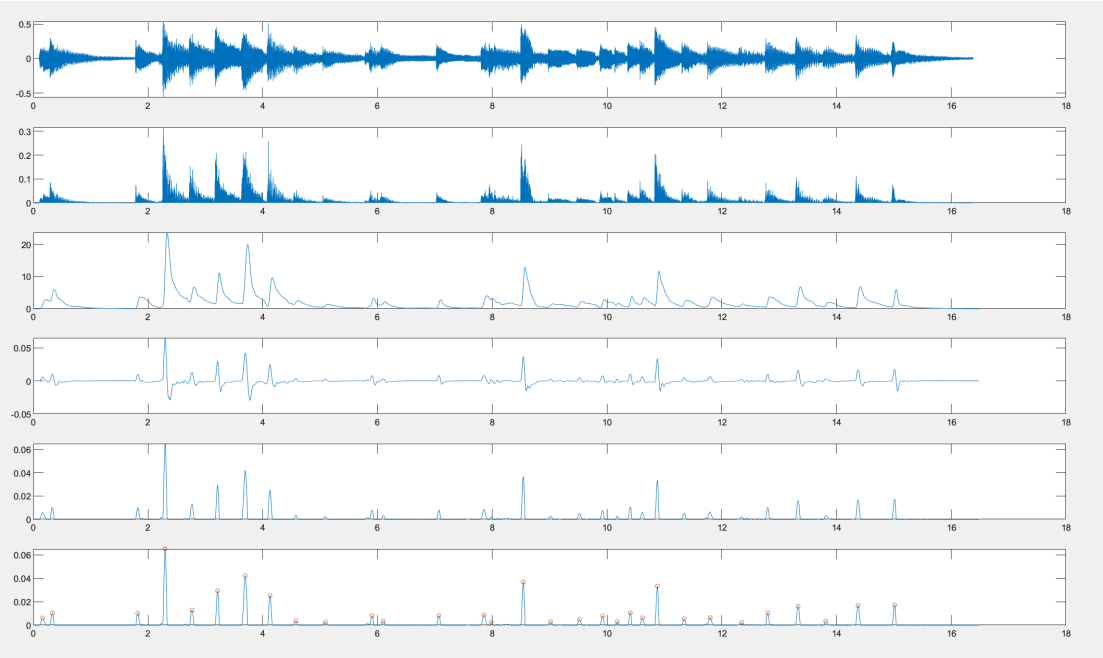
  ```
  [y5, loc] = findpeaks(y4);
  tmp = sort(y5, 'descend');
  threshold_amp = (tmp(2)+tmp(3))/2/29;
  threshold_gap = round(Fs/15);
  % disp(threshold_amp);

  loc = loc(y5>threshold_amp);
  y5 = y5(y5>threshold_amp);
  res = [loc(1)];
  last = y5(1);

  for k = 2:length(y5)
  ```

```
    if loc(k) - res(end) < threshold_gap
        if last < y5(k)
            res = [res(1:end-1), loc(k)];
            last = y5(k);
        end
    else
        res = [res, loc(k)];
        last = y5(k);
    end
end
```

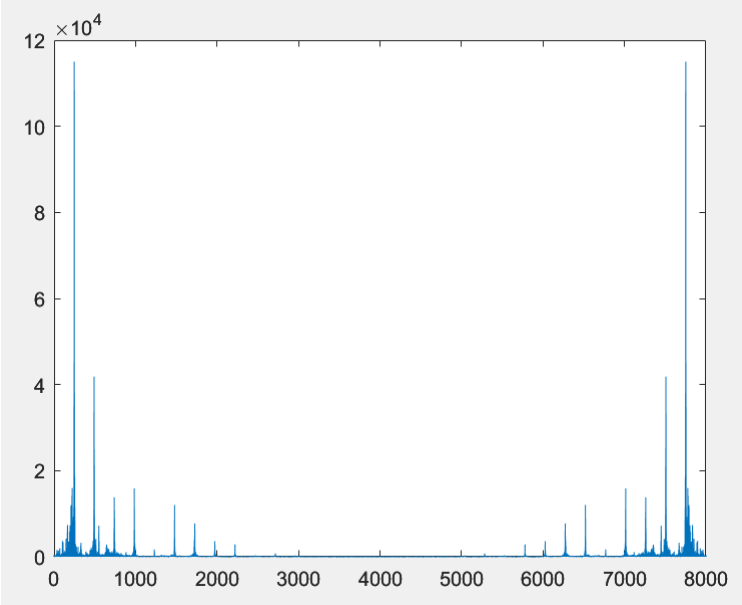
结果如下图：



2.4.2 分析音调

设一个 tunes 数组，保存 120Hz 到 1300Hz 范围内所有的标准频率（由 220Hz 乘上 $2^{[-10:30]/12}$ 得到），再设一个 harmo_amps 数组，保存每个频率的谐波相对幅度。

接下来是对分出节拍后的每一个音，提取它的基频和谐波幅度，首先片段重复1000次再做fft得到频谱：



- 寻找基频

按照这样的策略找：

1. 找出频谱中强度最大的点。
2. 基频强度不弱，以最大强度的一半为阈值，初筛，同时假定基波频率不超过最强的频率，且最强频点是基波频率的高次谐波（整数倍），只保留频率小于最强频点的部分。
3. 如果筛完的结果中存在某个频率使得最强频点很接近它的整数倍（看1,2,3,4倍就够了），则定为基频。
4. 在标准频率中找到与目前基频最接近的，定为最终结果。

```
X = X(1:end/2);
[amp, loc] = max(X);
threshold_amp = amp / 2;
ax = [1:length(X)]';
xs = ax(X>threshold_amp & ax<=loc);

for i = 1:length(xs)
    k = loc/xs(i);
    if k < 5 && abs(k/round(k)-1) < 0.05
        res = xs(i);
        break;
    end
end
baseFreq = (res-1)/N*Fs;

load instrument.mat tunes harmo_amps;
diff = abs(tunes - baseFreq);
[~, idx] = min(diff);
baseFreq = tunes(idx);
```

- 提取谐波相对幅度

  取基波的整数倍，然后在附近取幅度最大的点，附近的范围大小是基波频率的1/10。

  对不同节拍的相同基频的谐波取平均。

```
len = floor(Fs/2/baseFreq);
now_amps = zeros([1,len]);
Range = round(res/10);
for i = 1:len
    pos = (res-1)*i+1;
    now_amps(i) = max(X(pos-Range:min(pos+Range,end)));
end
now_amps = now_amps / now_amps(1);

if isempty(harmo_amps{idx})
    harmo_amps{idx} = now_amps;
else
    harmo_amps{idx} = (now_amps + harmos_amps{idx}) / 2;
end
save instrument.mat -append harmo_amps;
```

这样提取 `fmt.wav` 之后可以得到11个音调的谐波数据，然后再用邻近音调填充：

```
for i = 1:tune_num
    if isempty(harmo_amps(i))
        for j = 1:max(i-1, tune_num-i)
            if (i>j && ~isempty(harmo_amps(i-j)))
                harmo_amps(i) = harmo_amps(i-j);
                break;
            elseif (i+j<=tune_num && ~isempty(harmo_amps(i+j)))
                harmo_amps(i) = harmo_amps(i+j);
                break;
            end
        end
    end
end
```

至此，我们就获得了一段乐曲代表的所有音调的各谐波分量，它储存在谐波矩阵中。谐波矩阵相当于就代表了这段乐曲的音色特征。之后就可以利用这个谐波矩阵去演奏各种音乐了，演奏出来的音色将会与原本的乐曲音色相近。

3. 基于傅里叶级数的合成音乐

3.1 用傅里叶级数做谐波重做1.4

将 2.3 提取出的谐波相对幅度放入 1.4 的谐波系数中即可。

听起来感觉不是很像。

3.2 用 fmt.wav 演奏东方红

经过2.4提取harmo_amps保存到文件后，加载文件，找到频率对应的下标，取出对应的谐波幅度即可：

```
harmoConfs = harmo_amps{abs(tunes-f)<1};
```

听起来比 3.1 好一点，但钢琴的感觉还是比吉他重一点。

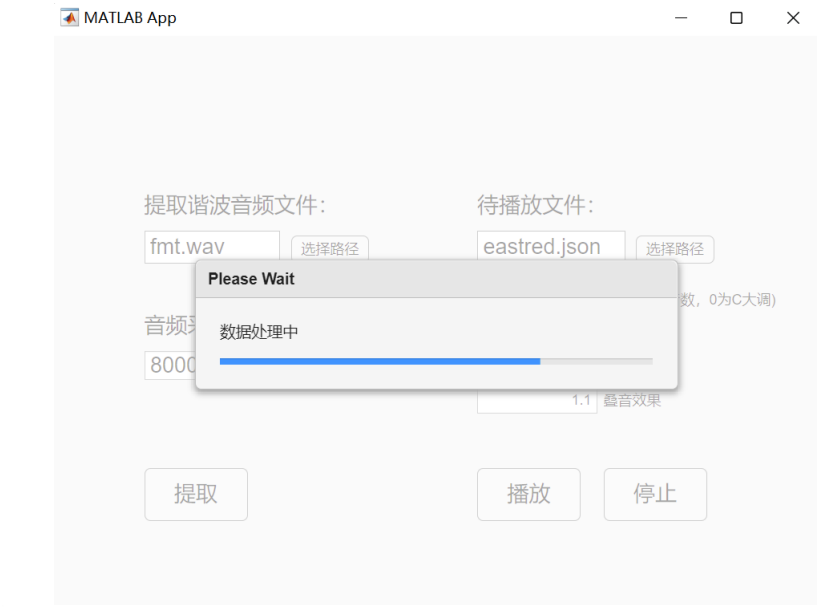
3.3 图形界面封装

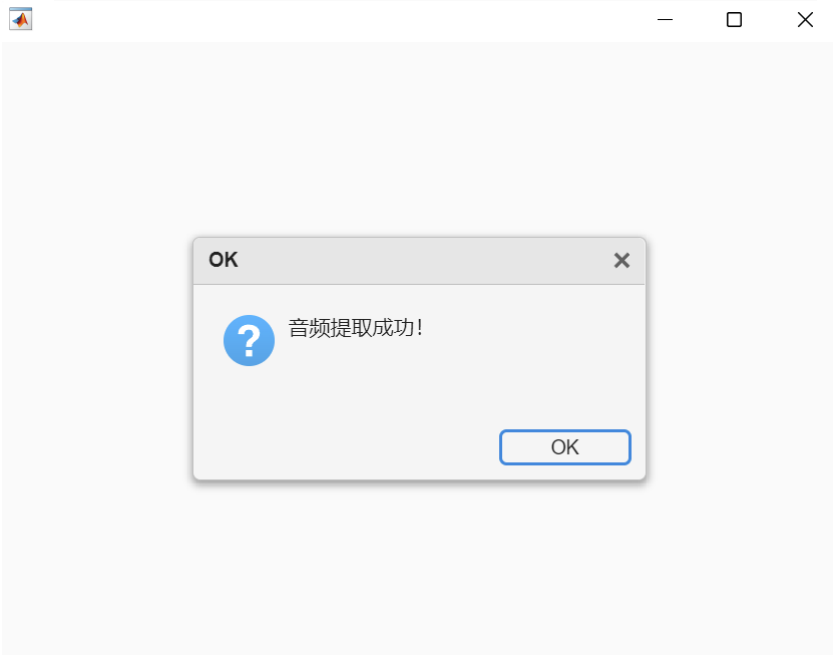
将提取音频功能写成了 `extract_harmonic` 函数，播放功能改为了由自定义的 json 文件格式播放，写做函数 `play_music`。

`json` 的每个音调的格式为：[0~2, 1~7, 节拍数]，第一个数表示低八度/正常/高八度，第二个表示do re mi，第三个为节拍数。正常do调的基频默认为C大调，可在GUI中增减。

封装为图形界面，`uigetfile` 实现路径选择，`clear sound` 实现停止播放，提取和播放分别调用函数即可。

`uiprogressdlg` 实现进度条，`uiconfirm` 实现弹窗，提取后的数据保存在 `instrument.mat` 文件中：





PS：后来发现wav的采样率可以直接`[wav, Fs] = audioread('xxx.wav')`读取，遂删去了采样率输入框。

在网上下了一个双声道的wav文件，提取之后播放了`传说的世界.json`，得到的声音更贴近真实的钢琴音了，感觉还可以接受，本来想加个伴奏的，但是发现这首歌找不到伴奏的简谱，遂作罢。

实验总结

总体来讲还是十分有趣的，从一开始的摸不着头脑，然后看PPT，看书，看网上资料，逐渐地明白音频分析的原理，然后一点点写代码，debug，画图，测参数，最后听到音乐，也算是历经考验。既温习了傅里叶变换在离散情况下的相关知识，又学到了matlab的很多使用技巧。

但是最终效果还是有很多可以改进的地方的，我想大概有这几个方面：

- 包络的改进。单纯的 xe^x 与实际钢琴的包络还是有区别，可以根据每个音调更精细地拟合，或自动提取。
- 谐波幅度的处理。合成出来有的音调听起来还是有点怪，实际片段划分做fft变换之后的频谱实际上挺杂乱的，并不是单有基频和它的整数倍这些频率，单音调的提取可能还得对片段再做一点处理。

- 乐谱的自动提取。即自动划分wav文件的节拍，提取音调和节拍数，转化为json格式保存。这个基于已有内容已经可以实现了。
- 平滑音的处理。乐谱中经常出现一个音需要平滑过渡到另一个音，而不是出现两个冲激。我尝试过在两个音的频率之间线性过渡或插值过渡，但效果都不是很好。目前还有一种思路，一种是给平滑音之间的时域加上一个横的 `s` 型包络，然后让两个音之间的频率以横、过渡、横的方式衔接，这样应该会比较符合实际。

完成这篇报告的时间是过ddl的凌晨6点，虽然没能赶上，但已经收获了一番探索的乐趣，大作业果然还是爽啊，要是有时间就更爽了。

最后感谢老师和助教为本次实验的付出，有一说一，ggg上课真是金句频出，饶有风趣，虽然大伙都说看PPT就行了，但第二节课有事没听完真是我学习生涯莫大遗憾，ε=(´ο｀*))唉。