

课程 Matlab 高级编程与工程应用

实验一 音乐合成

姓名：黎佳维

学号：2022010540

日期：2024.7

一、简单的音乐合成

(1)

根据十二平均律计算乐音频率¹:

名称	C3	C#3	D3	D#3	E3	F3	F#3	G3	G#3	A3	A#3	B3
频率/Hz	130.81	138.59	146.83	155.56	164.81	174.61	185	196	207.65	220	233.08	246.94
名称	C4	C#4	D4	D#4	E4	F4	F#4	G4	G#4	A4	A#4	B4
频率/Hz	261.63	277.18	293.66	311.13	329.63	349.23	369.99	392	415.3	440	466.16	493.88
名称	C5	C#5	D5	D#5	E5	F5	F#5	G5	G#5	A5	A#5	B5
频率/Hz	523.25	554.37	587.33	622.25	659.25	698.46	739.99	783.99	830.61	880	932.33	987.77

在 code1_1.m 中，首先以幅度 1、抽样频率 8kHz 的正弦信号测试了《东方红》歌曲中使用到的各个乐音；接着使用 mfreq 和 beats 两个数组存储了音乐的乐谱（一拍为 0.5s），转换为正弦信号写入到 music 数组中按顺序播放得到音乐，存储到 result1_1.wav 文件中。

在实际编写过程中，符号“5”与“1”之间差的倍数并不是 $2^{(4/12)}$ ，而是 $2^{(7/12)}$ ，这是因为中间存在半音，我在第一次编写时忽略了这一点因此产出的音乐音调不对，检查后更正了错误。

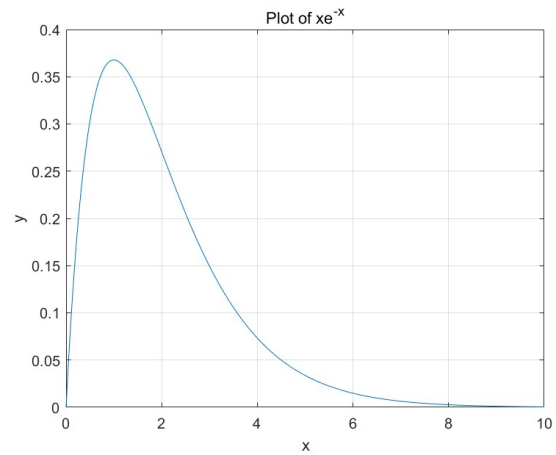
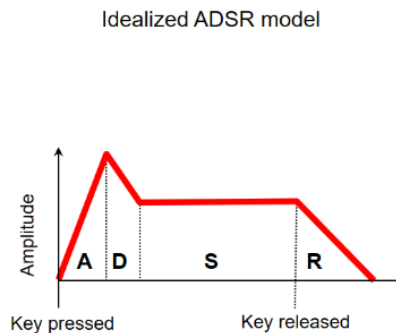
关键代码（code1_1.m）:

```
% test the sounds
t = linspace(0,beat,sampfreq*beat);
y = sin(2*pi*sound1*t);
sound(y,sampfreq);
pause(0.8);
...
music = [];
for i=1:8
    t = linspace(0,beats(i),sampfreq*beats(i));
    y = sin(2*pi*mfreq(i)*t);
    music = [music, y];
end
...
sound(music,sampfreq);
```

¹ 本报告采用科学音调记号法，即中央 C（261.63Hz）为 C4，标准音（440Hz）为 A4。

(2)

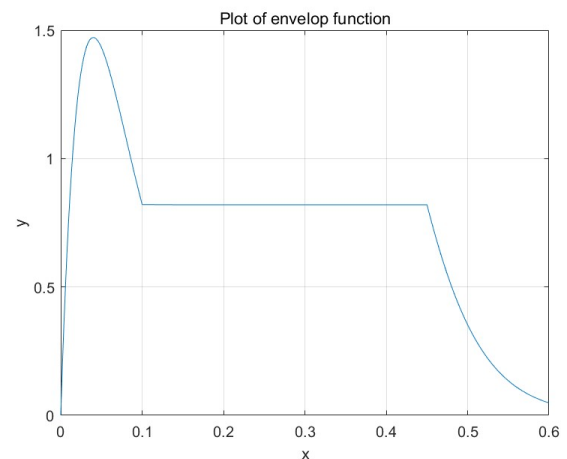
采用 $y = ax * e^{bx^c}$ 与 $y = d$ （其中 a, b, c, d 为参数）两个函数组成分段函数来拟合 ADSR 模型²：下图展示了 $y = ax * e^{bx^c}$ 函数，其与 ADR 三个阶段性质相似；而 $y = d$ 函数与 S 阶段性质一致。



使用 code1_2_test.m，调整参数（在代码中可见其具体值），得到如图所示的包络函数，近似拟合 ADSR 模型，且满足音乐在乐音的邻接处信号幅度近似为零：

关键代码（code1_2_test.m）：

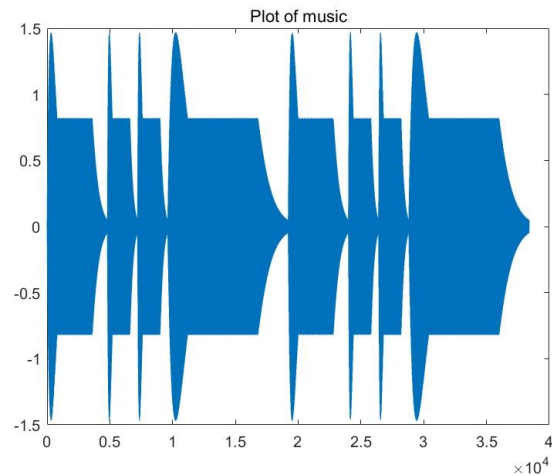
```
%parameters
a=100;
b=10;
c=2.5;
d=0.0082;
%func1
x1 = linspace(0, 0.1, 1000);
y1 = x1 .* (exp(-b*x1) .^c);
%func2
x2 = linspace(0.1, 0.45, 1000);
dd = linspace(d, d, 1000);
y2 = dd;
%func3
x3 = linspace(0.45, 0.6, 1000);
y3 = (x3-0.35) .* (exp(-b*(x3-0.35)) .^c);
%combination
x=[x1 x2 x3];
y=a*[y1 y2 y3];
```



² https://www.audiolabs-erlangen.de/resources/MIR/FMP/C1/C1S3_Timbre.html#Envelope-and-ADSR-Model

将包络函数“嵌套”到原来的乐音上，得到新的输出，存储到 result1_2.wav 文件中。这时乐音之间“啪”的杂声消除了，同时音调有了起伏，更自然了。该部分代码的关键点在于生成 envelop 函数并使用.*进行数组的相乘。

将波形输出，可见包络成功“嵌套”到了每个乐音的输出上，与原来相比，存在起伏同时在邻接处信号幅度近似为零，更自然了。



关键代码 (code1_2.m):

```
%parameters
a=100;
b=10;
c=2.5;
d=0.0082;
music = [];
for i=1:8
    t = linspace(0,beat*beats(i),sampfreq*beat*beats(i));
    y = sin(2*pi*mfreq(i)*t);
    %envelop function
    %func1
    x1 = linspace(0, beat/6, sampfreq*beat*beats(i)/6);
    y1 = x1 .* (exp(-b*x1) .^c);
    %func2
    x2 = linspace(beat/6, beat*4.5/6, sampfreq*beat*beats(i)*3.5/6);
    dd = linspace(d, d, sampfreq*beat*beats(i)*3.5/6);
    y2 = dd;
    %func3
    x3 = linspace(beat*4.5/6, beat, sampfreq*beat*beats(i)*1.5/6);
    y3 = (x3-0.35) .* (exp(-b*(x3-0.35)) .^c);
    %combination
    envfunct = a*[y1 y2 y3];
    music = [music, y.* envfunct]; %multiply
end
```

(3)

升高和降低八度，分别只需要将输出的采样率*2 与/2 即可；使用 `resample` 函数，输入参数 `inter=2^(1/12)`，即升高半个音阶，新的输出存储到 `result1_3.wav` 文件中。

关键代码（`code1_3.m`）：

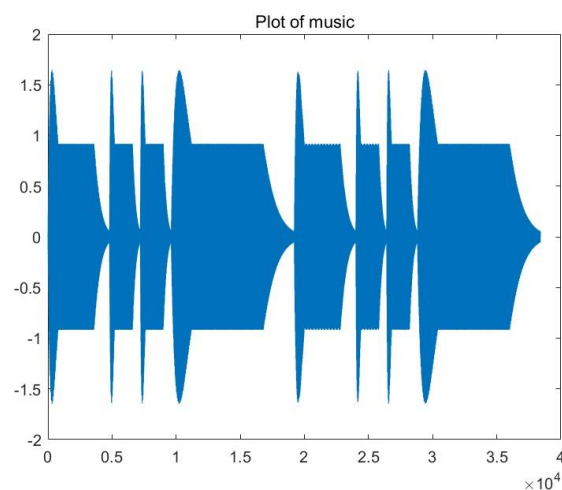
```
%sound(music,sampfreq/2);%down an octave
%sound(music,sampfreq*2);%up an octave
music = resample(music, sampfreq, round(sampfreq*inter));%+1
sound(music,sampfreq);
```

(4)

产生基波、二次谐波、三次谐波与包络相乘后加权相加即增加了谐波分量，输出其波形，于细微处和整体幅值可见其增加了谐波分量。按照 1:0.2:0.3 的比例生成了 `result1_4.wav` 文件，其音色类似风琴。

关键代码（`code1_4.m`）：

```
mag=[1 0.2 0.3];%amplitude of harmonics
z1 = sin(2*pi*mfreq(i)*t);
z2 = sin(4*pi*mfreq(i)*t);
z3 = sin(6*pi*mfreq(i)*t);
music = [music, mag(1)*z1.* envfunct+mag(2)*z2.* envfunct+mag(3)*z3.*
envfunct];%multiply
```



(5)

我选取了久石让创作的《Summer》作为编辑乐曲。相比于之前，我的程序增加了从 data1.txt 与 data2.txt 文件中读取原始乐谱、将原始乐谱翻译为频率参数（通过优化数据存储形式，直接将原始乐谱作为参数输入到函数中）的功能，增强了程序的使用便捷性，使用者可以直接输入原始唱名与节拍生成音乐（编码规则：C4=14，D4=24 以此类推，第二个数字为节拍）。

同时，程序支持一个节拍同时播放两个乐音，即“左右手同时演奏”，使得音乐更加丰富多彩。音乐存储到 result1_5.wav 文件中，《Summer》的乐谱存在 data1.txt 与 data2.txt 文件中（对应左右手）用来读取。

关键代码（code1_5.m）：

```
%read file
leflyric = [];
fid = fopen('data2.txt', 'r');
if fid == -1
    error('无法打开文件');
end
tline = fgetl(fid);
while ischar(tline)
    values = sscanf(tline, '%d %d');
    leflyric = [leflyric; values'];
    tline = fgetl(fid);
end
fclose(fid);
...
% translate
for i=1:208
    leflyric(i,2)=leflyric(i,2)/4;
    for k=1
        switch leflyric(i,k)
            case {12}
                leflyric(i,k)=-24;
            case {22}
                leflyric(i,k)=-22;
            case {32}
                leflyric(i,k)=-20;
            case {42}
                leflyric(i,k)=-19;
            case {52}
                leflyric(i,k)=-17;
            case {62}
                leflyric(i,k)=-15;
```

```

        case {72}
            leflyric(i,k)=-13;
...
lefmusic = [];
for i=1:208
    t = linspace(0,beat*leflyric(i,2),sampfreq*beat*leflyric(i,2));
    z1 = sin(2*pi*(inter^(leflyric(i,1)))*basefreq*t);
    z2 = sin(4*pi*(inter^(leflyric(i,1)))*basefreq*t);
    z3 = sin(6*pi*(inter^(leflyric(i,1)))*basefreq*t);

    %envelop function
    %func1
    x1 = linspace(0, beat/6, sampfreq*beat*leflyric(i,2)/6);
    y1 = x1 .* (exp(-b*x1) .^c);
    %func2
    x2 = linspace(beat/6, beat*4.5/6, sampfreq*beat*leflyric(i,2)*3.5/6);
    dd = linspace(d, d, sampfreq*beat*leflyric(i,2)*3.5/6);
    y2 = dd;
    %func3
    x3 = linspace(beat*4.5/6, beat, sampfreq*beat*leflyric(i,2)*1.5/6);
    y3 = (x3-0.35) .* (exp(-b*(x3-0.35)) .^c);
    %combination
    envfunct = a*[y1 y2 y3];

    lefmusic = [lefmusic, mag(1)*z1.* envfunct+mag(2)*z2.*
envfunct+mag(3)*z3.* envfunct];%multiply
end

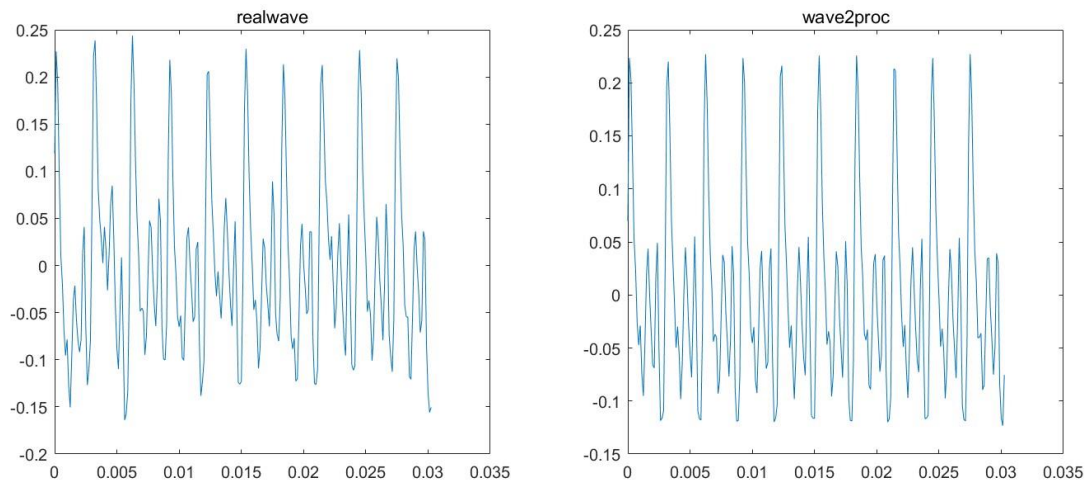
...
%play two parts at the same time
music = lefmusic + rigmusic;
sound(music,sampfreq);

```

二、用傅里叶级数分析音乐

(6)

使用如下代码将 Guitar.MAT 中的波形绘制（采样频率 8kHz），并播放 fmt.wav。此时音乐比（一、）中合成的音乐更加真实了。

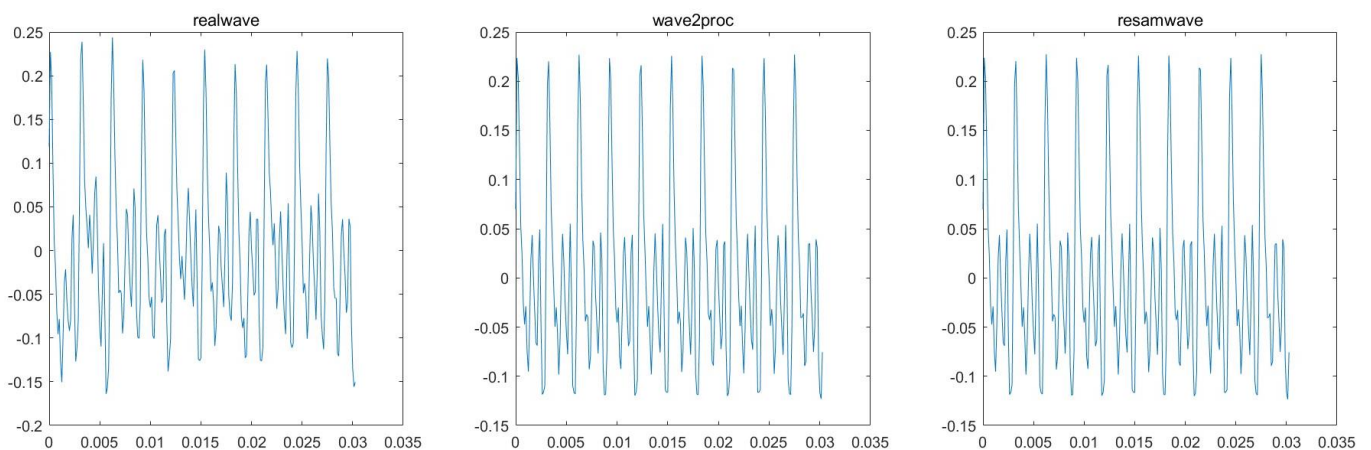


关键代码（code2_6.m）：

```
sampfreq = 8000;
%plot the waves
load Guitar.MAT
figure;
subplot(1,2,1);
plot([0:length(realwave)-1]/sampfreq, realwave);
title('realwave');
subplot(1,2,2);
plot([0:length(wave2proc)-1]/sampfreq, wave2proc);
title('wave2proc');
%sound the music
music = audioread('fmt.wav');
sound(music, sampfreq);
```

(7)

由于真实音乐中的噪音多为随机噪音，因此在此处将噪音视为随机噪音。要消除随机噪音，对此音频应该对 `realwave` 的十个周期取平均，具体的方法是：先将 `realwave` 以十倍频率重新采样，这样在保证了精度不丢失的情况下将十个周期累加起来再除以十，再将得到的 `averawave` 写回 `resamwave` 十次，再对 `resamwave` 以十分之一频率重新采样，得到消除随机噪音且采样率始终一致的音频。下图为 `realwave`，`wave2proc` 与结果 `resamwave` 的波形对比，后面二者波形相似，可见噪音消除效果较好。



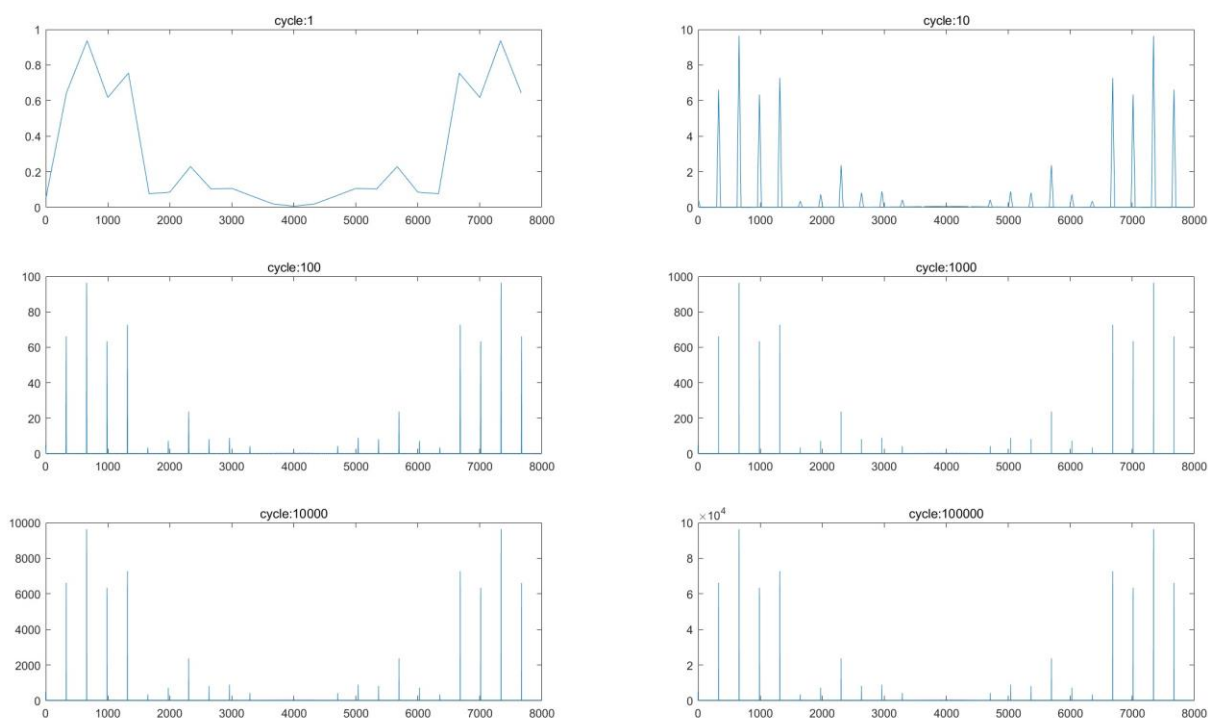
关键代码 (code2_7.m):

```
%eliminate the noise
%resample
resamwave = resample(realwave, 10, 1)/10;
averawave = zeros(length(resamwave)/10,1);
%average
for i = 0:9
    averawave = averawave +
resamwave(i*length(resamwave)/10+1:(i+1)*length(resamwave)/10);
end
%copy
resamwave = repmat(averawave, 10, 1);
%reresample
resamwave = resample(resamwave, 1, 10);
subplot(1,3,3);
plot([0:length(resamwave)-1]/sampfreq, resamwave);
title('resamwave');
```

(8)

要寻找信号的基频，需要对信号进行傅里叶变换。在 matlab 中使用 fft 对信号做快速傅里叶变换。下图给出了不同 cycle 情况下的傅里叶变化结果，此处采用的是 wave2proc 音频，cycle=10 即为该音频本身，cycle=1 为取第一个周期的波形，cycle=10 为复制其十次，以此类推。

可见，随着 cycle（即时域的数据量）的不断增多，频域在基波及谐波频率上近似冲激函数。这是因为在时域数据量较少时，信号类似于过了一个“时域上的滤波器”，在频域上则是与 sinc 函数做卷积，因此如果不扩充时域数据量，频域上始终接近 sinc 函数；而扩充了时域数据量后，信号的周期性体现更加明显，也因此体现在频率图上，不断近似为多个频点上周期信号的叠加，若扩充时域到无限处，则可得到周期信号的傅里叶级数。



在此基础上，选择扩充至 `cycle=1000` 的信号进行处理，观察图可以发现基频近似在 0~500 段中，得到基频为 329.2181Hz，即为 E4。以该频率为基准，扫描谐波的分量，由于每周期的采样点数为 243/10 个，因此最大取到十二次谐波。根据下左图可见（第一列：谐波阶次；第二列：频率；第三列：相对幅值），该音乐基波、二、三、四次谐波较强。

而采用在其他各个频段内搜索最大幅值的方法对信号频谱进行分析，将输出列表于下右图。可以发现实际上在十一、十二次谐波上，最大幅值频点并不是基频的整数倍，结合频谱图来看，推测其原因是受采样率、信号时域有限、信号的更高次谐波频谱等因素的影响。

```
>> code2_8
basefreq = 329.2181
```

0	0	0.07
1.00	329.22	1.00
2.00	658.44	1.46
3.00	987.65	0.96
4.00	1316.87	1.10
5.00	1646.09	0.05
6.00	1975.31	0.11
7.00	2304.53	0.36
8.00	2633.74	0.12
9.00	2962.96	0.14
10.00	3292.18	0.06
11.00	3621.40	0.00
12.00	3950.62	0.01

0	0	0.07
1.00	329.22	1.00
2.00	658.44	1.46
3.00	987.65	0.96
4.00	1316.87	1.10
5.00	1646.09	0.05
6.00	1975.31	0.11
7.00	2304.53	0.36
8.00	2633.74	0.12
9.00	2962.96	0.14
10.00	3292.18	0.06
11.00	3654.32	0.01
12.00	3983.54	0.01

关键代码（code2_8.m）：

```
%cycle=1000
%cycle=1000
wave = repmat(wave2proc, 100, 1);
res = fft(wave);
subplot(3,2,4);
plot([0:length(res)-1]*sampfreq/length(res),abs(res));
title('cycle:1000');

%find the base frequency
left = 0;
right = 500*length(res)/sampfreq;%look up range
[value, freq] = max(abs(res((left + 1):round(right))));
basefreq = (freq-1)/length(res)*sampfreq;
```

```

disp(['basefreq = ', num2str(basefreq)]);

%find the harmonics frequencies by multiplying
for i=0:12
    testfreq = freq*i - i + 1; %transformed frequency
    testvalue = abs(res(testfreq));
    disp([i,basefreq*i,testvalue/value]);
end

%find the harmonics frequencies by maxing
for i =0:12
    left = 333*(i-1)*length(res)/sampfreq;
    if i<=1
        left=1;
    end
    right = 333*(i)*length(res)/sampfreq+1;%look up range
    [testvalue, freq] = max(abs(res(round(left):round(right))));
    testfreq = (freq+round(left)-2)/length(res)*sampfreq;
    disp([i,testfreq,testvalue/value]);
end

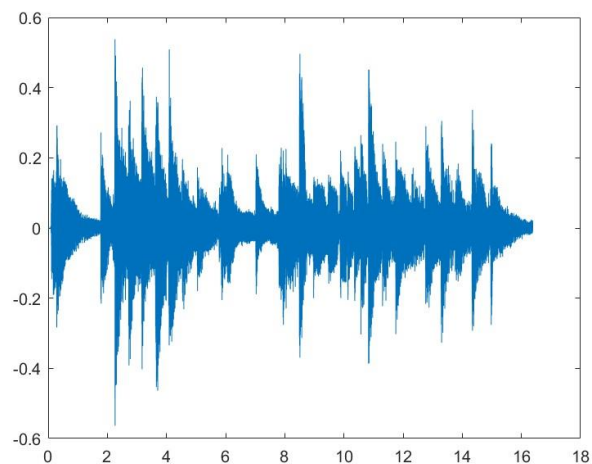
```

(9)

该部分由多次的尝试实验、分析与结果呈现组成，需要完成的任务整体上是先划分节拍，后寻找音调，划分节拍的部分在实验的基础上部分思路选择有所参考³。

划分节拍：

由 `audioread` 读出 'fmt.wav' 的 wave，可见该乐曲中每一个节拍对应一个先急升后相对缓降的波形。要划分出每一个节拍来，首先想到的是将单个节拍这样的升降模型的特性“放大出来”，由此引申出：



³ <https://github.com/zhangzw16/Project-for-Signals-and-Systems-2021/blob/main/Project2021.pdf>

code2_9_test1.m (关键部分):

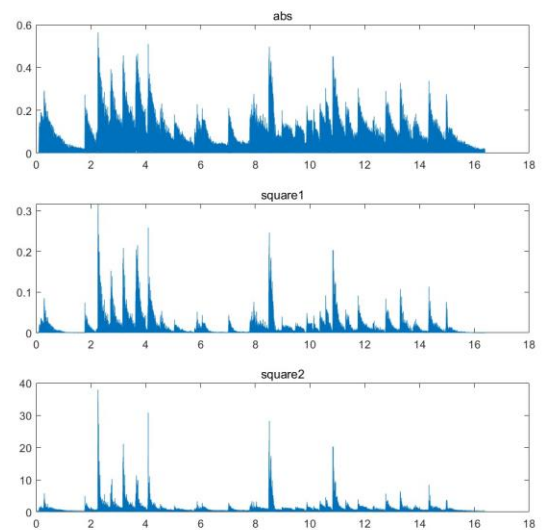
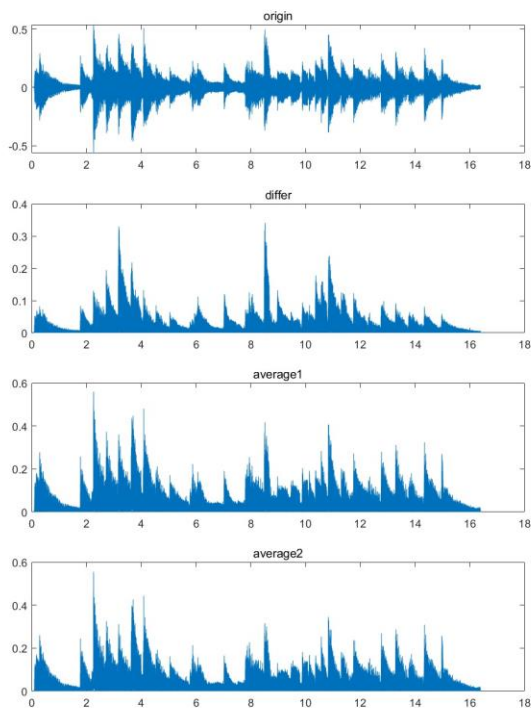
```
%1
wave = audioread('fmt.wav');
subplot(5,3,1);
plot([0:length(wave)-1]/sampfreq, wave);

%2
wave1 = abs(wave);
subplot(5,3,2);
plot([0:length(wave)-1]/sampfreq, wave1);
...

```

该处尝试了 6 中方法，对应下图中分别是：

- 1.origin (原始)
- 2.abs
- 3.differ (差分)
- 4.square1 (平方)
- 5.average1 (前后共 3 位平均)
- 6.square2 (+0.9 后十次方)
- 7.average2 (前后共 5 位平均)

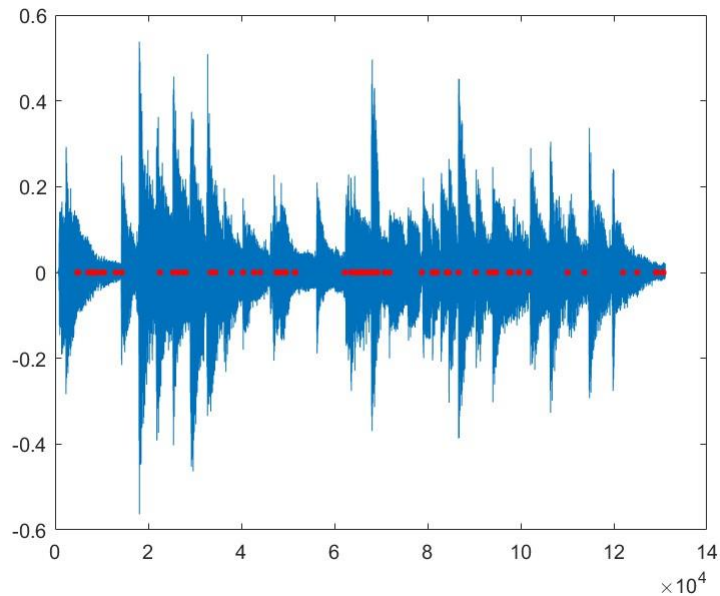


由上图可见，提取单个节拍特性的 4、6 方法较好，从理论上分析，平方能够放大大于 1 的值，缩小小于 1 的值；而方法 6 将此特性利用到了极致，10 次方使得放缩程度更加明显，+0.9 是不让信息损失过多（考虑到后续过程的划分，过小的值都将被忽略）。由于方法 6 在一些“较小”的节拍上损失过大，因此选择方法 4 作为后续分析的基础（本质上方法 4 也是方法 6 参数变换的版本）。

拥有了波形特征基础，实际上就可以开始寻找节拍了。我首先尝试了这样一种方法：从节拍包络函数的特性出发，若某处的值大于其前面 n 个数平均值的 p 倍且大于其后面 m 个数的平均值的 q 倍，则认为此处为节拍，其中这四个参数可调，并且寻找数的范围也存在很大的调整范围。此处采用了代码中所示的参数进行实验，得到如下结果。

code2_9_test2.m（关键部分）：

```
...
for i = 101:length(wave1)-100
    if(wave1(i)>wave1(i-1)&&wave1(i)>wave1(i+1))
        x=0;
        for i1 = 1:100
            x = x+wave1(i-i1);
        end
        x=x/100;
        if(wave1(i)>x*9)%bigger than 9 times of the average of the former
100 values
            y=0;
            for i2 = 1:100
                y = y+wave1(i+i2);
            end
            y=y/100;
            if(wave1(i)>y*4)%bigger than 4 times of the average of the
following 100 values
                flag(j) = i;
            end
        end
    end
end
...
```



可见，该方法得出的结果并不理想，由于噪音的存在，以及节拍存在混叠部分，使得结果过于稠密。尽管此方法还存在调参优化的空间，但本质上无法克服节拍混叠的问题，因此我决定从本质上改进方法。直接从该波形上进行提取的缺点就是存在噪声导致毛刺过多，直接使用含有噪音的单点的值存在极大的误差，因此尝试提取其包络，因为包络更为连续，提取性质更为可行，由此引申出（窗函数的选择参考了⁴⁵）：

对应下图五行

1. rectwin
2. triang
3. chebwin
4. barthannwin
5. gausswin

code2_9_test3.m（关键部分）：

```
...
%fine the most suitable envelop
sampfreq = 8000;

wave = audioread('fmt.wav');

wave1 = wave .^ 2;

window1 = rectwin(1000);
```

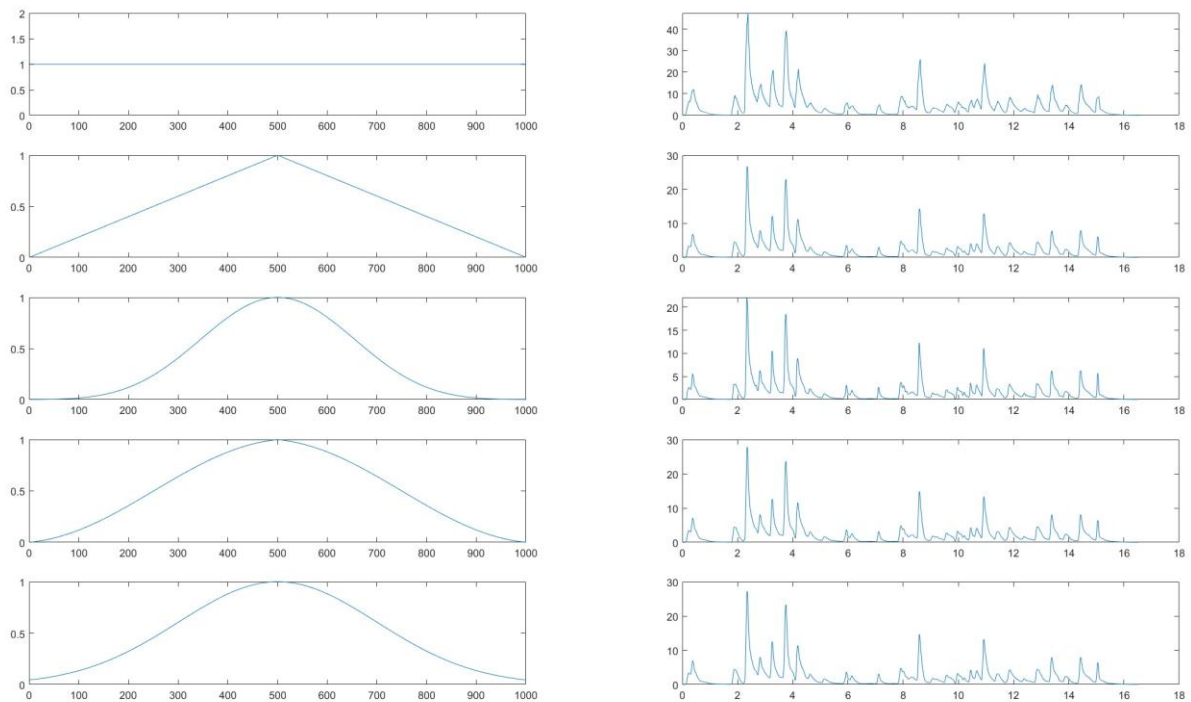
⁴ <http://www.ece.northwestern.edu/local-apps/matlabhelp/toolbox/signal/barthannwin.html>

⁵ https://ww2.mathworks.cn/help/signal/ug/windows_zh_CN.html#brbq7fh

```

subplot(5,2,1);
plot(window1);
wave2 = conv(window1, wave1);
subplot(5,2,2);
...

```



不同的窗函数得到包络提取的效果相近，选取 `triang` 作为窗函数为基础进行下一步实验。下图所示的最上方图即为 `test3` 实验 `triang` 作为窗函数提取出的包络。观察包络的性质可知，提取的目标即为各个先升后降的“山峰”部分。比较自然的想法是，对其进行差分或者求导，差分能够体现变化的性质，而求导则能找到极值点。于是在以下实验中对两种方法进行了尝试，差分得到的结果即中间的波形，求导得到的波形即最下方的波形。

`code2_9_test4.m`（关键部分）：

```

...
% differ way
while(i <= length(wave3))
    if(wave3(i) >= 0.001)
        j = i;
        while(wave3(j) >= 0.001)
            j = j + 1;

```

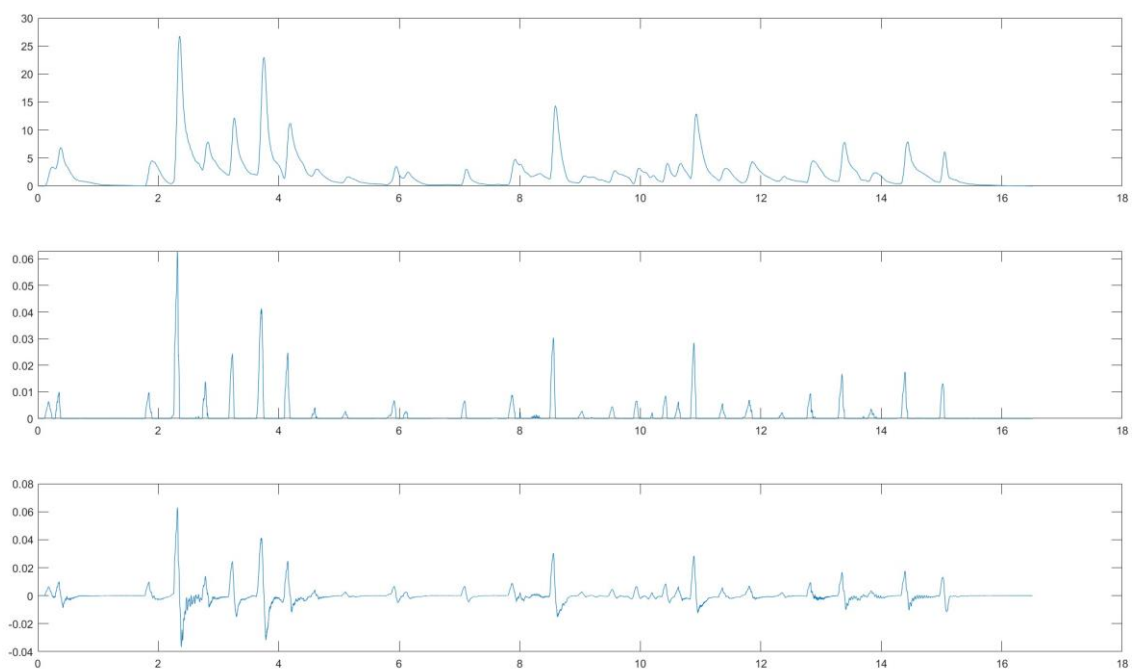


```

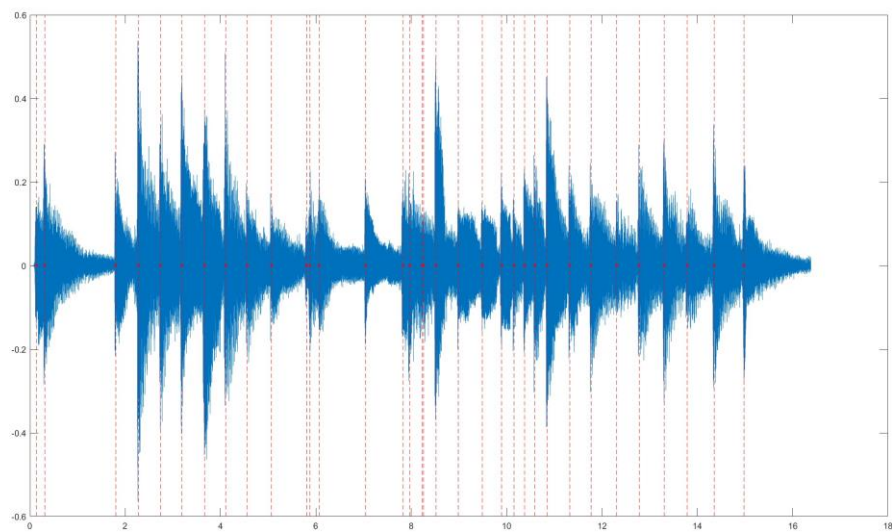
end
[~,x0(k)]=max(wave3(i:j,1));
x0(k) = x0(k) + i - 330;%parameter to offset the influence of
differential func
k = k + 1;
i = j;
end
i = i + 100;
end

%%derivation way
wave4 = diff(wave2);
window1 = gausswin(1000);
wave5 = conv(window1, wave4);
wave6 = diff(wave5);
x1 = zeros(length(wave4),1);
i1 = 1;
for i = 1: length(wave4)
    if(wave5(i) == 0)
        if(wave6(i) <= -0.000000001)
            x1(i1) = i;
            i1 = i1 + 1;
        end
    end
end
end
...

```



对于求导的方式，因为要求二次导以判断极值点类型，因此对于导数进行了再次求包络后再求导的优化，尽管如此，但由于包络中仍然存在毛刺，使用完全数学解析的方法依然行不通。从如图中也可以看出，小的抖动部分也满足极值点的条件，因此判断的条件设置的太紧就会导致节拍过少，而过松就会导致节拍过多。而是用差分的方法相比于求导，对函数的优良性质要求轻松很多，因此如中间的波形所示其提取更为清晰。采取全局搜索的办法，寻找出定义域 $[i, j]$ 上的最大值即为节拍点，其中 $\text{wave}(i)$ 、 $\text{wave}(j)$ 均大于设定的参数即可。找到后对于节拍具体的位置进行了一定的修正，这是因为差分后最大值位置统一向右偏移。最后标注出提取的节拍处如图所示。



分析音调：

该部分的大体方法取自上题，即对每个部分进行时域重复扩充后采用 `fft`，在从频域上寻找特性。在频域上，先找到左半部分的最大值对应坐标，认定其为 n 次谐波，再以此为基础搜索使其整数分之一频率附近的幅值大小，若与其幅度大小差别不太大（由设置的参数决定），则找到的最小的频率即为基准频率，再以此为基准，采用上题 `max` 寻找谐波的方法寻找谐波频率与幅值，随后输出谐波的频率和相对幅值到矩阵 `harmofrequ` 与 `harmoalue` 中，基准频率同时存入向量 `basefrequn` 中。

第(9)题完整的代码整理在 `code2_9` 中。

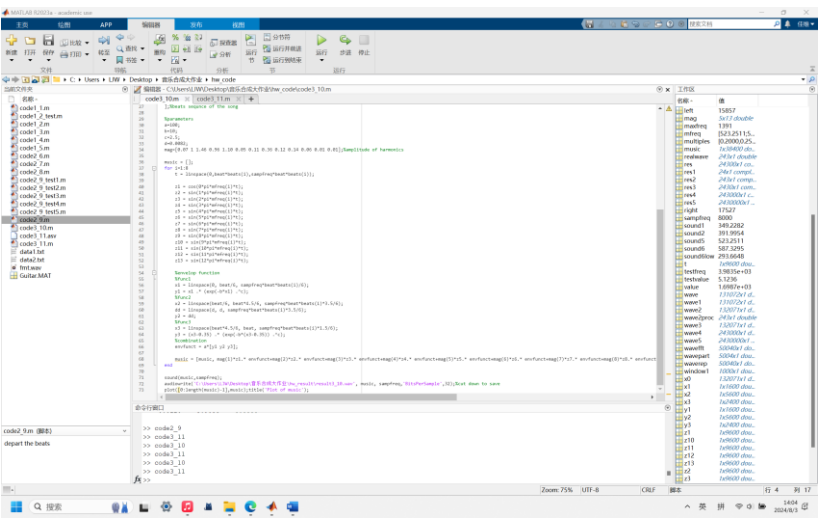
code2_9_test5.m (关键部分):

```
...
while(x0(i)>0)
    left = x0(i-1);
    right = x0(i);
    wavepart = wave(left:right); %the part searching now, avoid the first
and last x0 = 0
    waverep = repmat(wavepart,10,1);
    wavefft = fft(waverep);
    wavefft = abs(wavefft);
    plot(wavefft);
    %find the basefreq
    [value,maxfreq]=max(wavefft(1:length(wavefft)/2,1));
    multiples = [1/5, 1/4, 1/3, 1/2, 1];
    freqs = round(maxfreq * multiples);
    base = 1;
    j = 1;
    basevalue = 0;
    while(basevalue/value<0.1)
        [basevalue,basefreq]=max(wavefft((freqs(j)-10):(freqs(j)+10),1));
        basefreq = basefreq + freqs(j) - 10 - 1;
        j = j + 1;
    end
    %find the harmonies
    for ii = 1 : 13
        left = round(0.95*basefreq*(ii-1));
        if ii<=1
            left=1;
        end
        right = round(1.05*basefreq*(ii-1));%look up range
        if ii<=1
            right=10;
        end
        [harmovalue(i-1,ii), harmofrequ(i-1,ii)] = max(wavefft(left:right));
        harmofrequ(i-1,ii) = (harmofrequ(i-1,ii)+left-
2)/length(wavefft)*sampfreq;
        harmovalue(i-1,ii) = harmovalue(i-1,ii)/basevalue;
    end
    basefrequn(i-1)=harmofrequ(i-1,2);
    i = i + 1;
end...
```

三、基于傅里叶级数的合成音乐

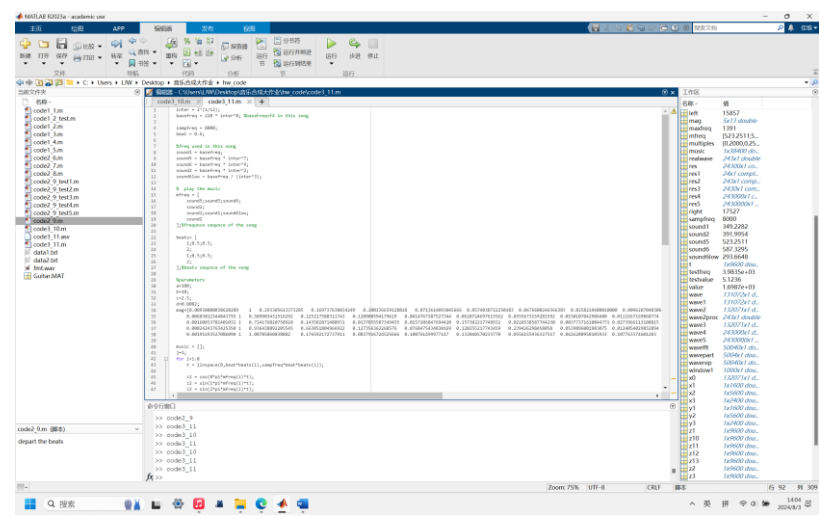
(10)

该处使用(8)中计算的参数再次合成了(4)中的音乐，存放在文件 **result3_10.wav** 中，其音色介于风琴与吉他之间。其代码在 4 的基础上进行了扩充，引入了更多的谐波与幅度参数。



(11)

基于上一题，将所有演奏的音调频率对应的谐波参数由(9)中提取后写入 **mag** 数组中，生成波形时分配不同的泛音数量和幅度，存放在文件 **result3_11.wav** 中。其音色与吉他更接近了，但仍有一定差距，可能的原因是我所使用的包络与吉他音色的包络仍有一定出入。



四、实验总结

本次实验是基于信号与系统理论课后的 matlab 实操，在对音乐的分析与合成的过程中我对于信号与系统的理解更为加深了。所谓信号，我们在本实验中处理的对象“音乐”即作为一种常见而颇具美感的信号，基础的音调、不同乐器独特的音色，就对应着信号中的频率、谐波；而在实验中所设计的划分节拍器、音调分析器以及分析中使用到的各种工具，就对应着系统中的窗函数、滤波器等。音乐与音乐处理技术很好地将信号与系统以一种有趣的方式呈现了出来。

在具体实操上，我对于 matlab 数组、封装函数、绘图等功能更为熟悉了，也体会到了其处理信号的强大之处。从 fft 操作，到包络中的各种包络函数调用、conv 操作，matlab 具有很好的工程处理能力；而 audio 方面的操作，也使其具有工程基础上进行创作的能力。

最后，在设计 envelop 函数、创作自己的乐曲，当然尤其在第（9）题，使我体会到了自主地查阅资料、进行实验寻找最优解地 research 过程的乐趣，虽然本次大作业可能还没有 research 的高度，而只是一个小的 project，但探索的过程也是我收获颇丰。