作业一 音乐合成

郝千越 2018011153 无 85

（系统环境：Windows10，MATLAB 版本：R2020a）

目录

1.2.1 简单的合成音乐

（1）初步合成《东方红》片段

【题目描述】

请根据《东方红》片断的简谱和“十二平均律”计算出该片断中各个乐音的频率，在 MATLAB 中生成幅度为 1、抽样频率为 8kHz 的正弦信号表示这些乐音。最后用这一系列乐音信号拼出《东方红》片断。

【主要思想】

该乐曲为 F 大调，根据乐理知识转换出各乐音相当于基准音（220Hz）相差的半音数目：

表 1 《东方红》乐音半音间隔分析

唱名	$\dot{6}$	1	2	5	6
音名	D	F	G	C	D
间隔半音数	5	8	10	15	17

按照上述间隔半音数，依次生成正弦波，联结成乐曲。

【核心代码】

```
seq=[15,15,17,10,8,8,5,10];%曲谱序列
time=[0.5,0.25,0.25,1,0.5,0.25,0.25,1];%时值(s)
inter=[0.1,0.05,0.1,0.1,0.1,0.05,0.1,0.1];%间隔时间(s)
f0=220;%基准频率
freq=8000;%采样频率
amp=1;%幅度
output=zeros(1,freq*sum(time)+freq*sum(inter));%输出向量
position=1;%辅助指针
for k=1:length(time)
    t=linspace(0,time(k)-1/freq,freq*time(k));%生成时间
    temp=amp*sin(2*pi*f0*pow2(seq(k)/12)*t);%生成乐音
    output(position:position+freq*time(k)-1)=temp;%添加乐音
    position=position+freq*time(k);%移动指针
    output(position:position+freq*inter(k)-1)=zeros(1,freq*inter(k));%添加间隔
    position=position+freq*inter(k);%移动指针
end
sound(output,freq);%播放
```

【核心代码说明】

依次生成幅度为 1、抽样频率为 8kHz 的正弦信号表示这些乐音，完整完整代码文件为 ex_1_2_1_1_a.m。

将上述单音连接成《东方红》片段完整代码文件为 ex_1_2_1_1_b.m，seq 依次存储乐谱中各个音符，time 存储各个音符的时值。考虑乐谱中有连音线，即各个音符之间间隔应当不同，因此用 inter 存储音符间隔。利用辅助指针索引 output 向量，向其中逐个插入音符，得到最终的结果 output 并播放、保存。

【运行结果】

得到音频文件“东方红 1.wav”。

【结果分析】

绘制 output 波形图并局部放大：

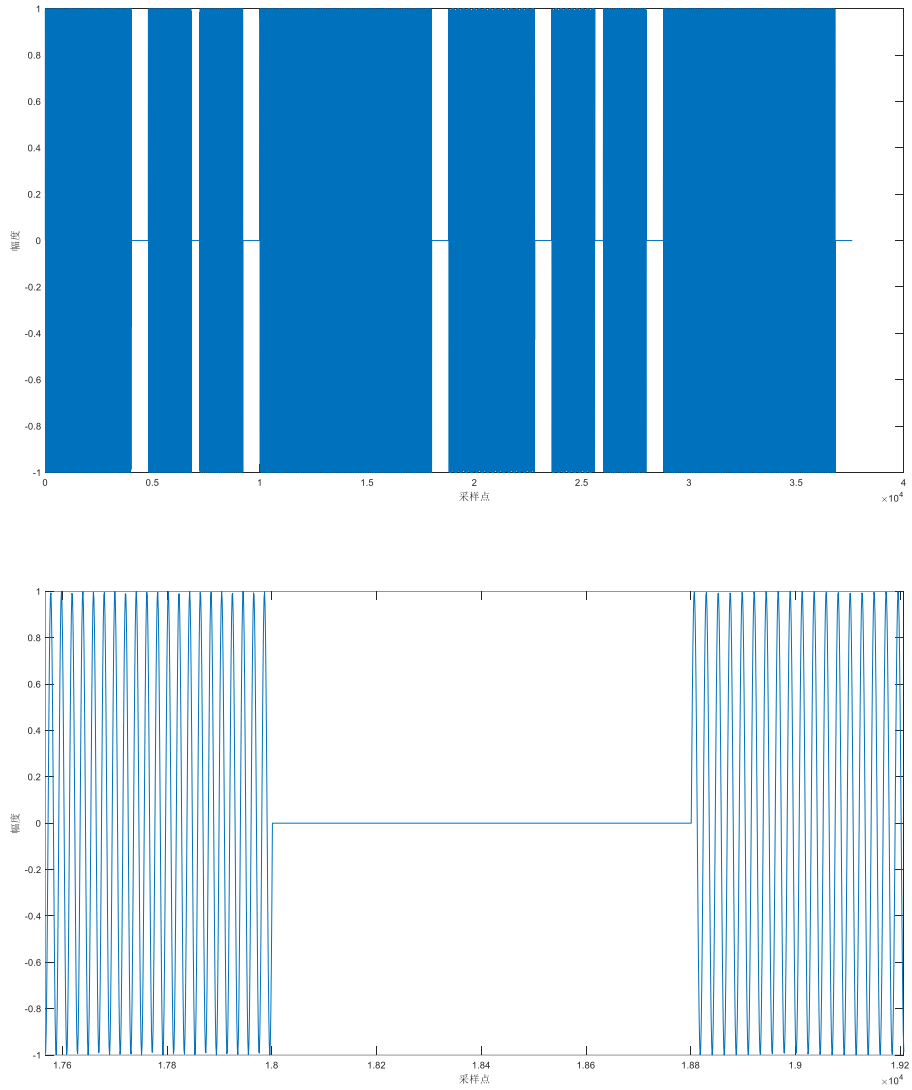


图 1 output 波形图

波形为简单的正弦波，由于没有包络，两个音符交界处产生高频分量，导致音乐中存在“啪啪”的噪声；另外由于没有谐波分量，音乐听起来很单薄，即没有丰富的音色特性。

（2）增加包络与音符迭接

【题目描述】

（1）中的乐曲中相邻乐音之间有“啪”的杂声，这是由于相位不连续产生了高频分量。这种噪声严重影响合成音乐的质量，丧失真实感。为了消除它，我们可以用包络修正每个乐音，以保证在乐音的邻接处信号幅度为零。

【主要思想】

为消除振幅跳变造成的“啪啪”噪声，给每个乐音增加包络，根据钢琴大致包络形状：

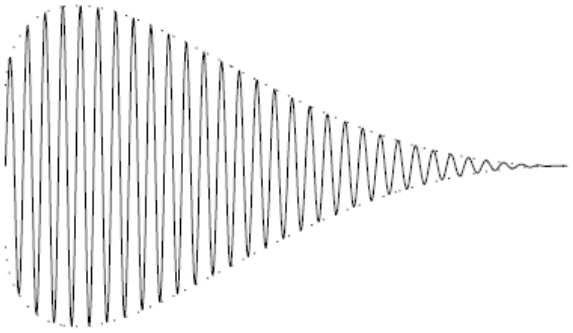


图 2 钢琴乐音包络形状

将包络用两端曲线近似，第一段为开口向下的二次函数，描述乐音开始时强度增强；第二段为指数衰减，描述乐音逐渐衰弱。表达式为：

$$y=\begin{cases}-100\left(\frac{t}{t_{Last}}\right)^2+20\left(\frac{t}{t_{Last}}\right), 0<t<0.1t_{Last}\\ 1.15652*e^{-2*\frac{t-0.1*t_{Last}}{0.9*t_{Last}+t_{Overlap}}}-0.15652, 0.1t_{Last}<t<t_{Last}+t_{Overlap}\end{cases}$$

其中t_{last}为该音符时值，$t_{Overlap}$为该音符与下一音符迭接时间。

【核心代码】

```
seq=[15,15,17,10,8,8,5,10];%曲谱序列
time=[0.5,0.25,0.25,1,0.5,0.25,0.25,1];%时值(s)
overlap=[0.2,0.3,0.2,0.2,0.2,0.3,0.2,0.2];%迭接时间(s)
f0=220;%基准频率
freq=8000;%采样频率
amp=1;%幅度
output=zeros(1,freq*(sum(time)+overlap(length(overlap))));%输出向量
position=1;%辅助指针
for k=1:length(time)
    t=linspace(0,time(k)+overlap(k)-1/freq,freq*(time(k)+overlap(k)));%生成时间
    temp=amp*sin(2*pi*f0*pow2(seq(k)/12)*t);%生成乐音
    cover=[-100*(t(1:0.1*time(k)*freq-1)/time(k)).^2+20*(t(1:0.1*time(k)*freq-1)/time(k))...
        ,1.15652*exp(-2*(t(0.1*time(k)*freq:freq*(time(k)+overlap(k)))-
0.1*time(k))/(0.9*time(k)+overlap(k)))-0.15652];%生成包络
    output(position:position+freq*(time(k)+overlap(k))-
1)=output(position:position+freq*(time(k)+overlap(k))-1)+cover.*temp;%添加乐音
    position=position+freq*time(k);%移动指针
end
output=output/max(abs(output));%归一化
sound(output,freq);%播放
```

【核心代码说明】

分别生成包络数组 cover 和正弦波数组，两者相乘得到带有包络的乐音。用带有包络的乐音，同时在乐音之间设置迭接部分，增加音乐的连贯性，重新合成东方红片段，完整代码文件为 ex_1_2_1_2.m。

【运行结果】

得到音频文件“东方红 2.wav”。

【结果分析】

增加该包络后，单个乐音波形为：

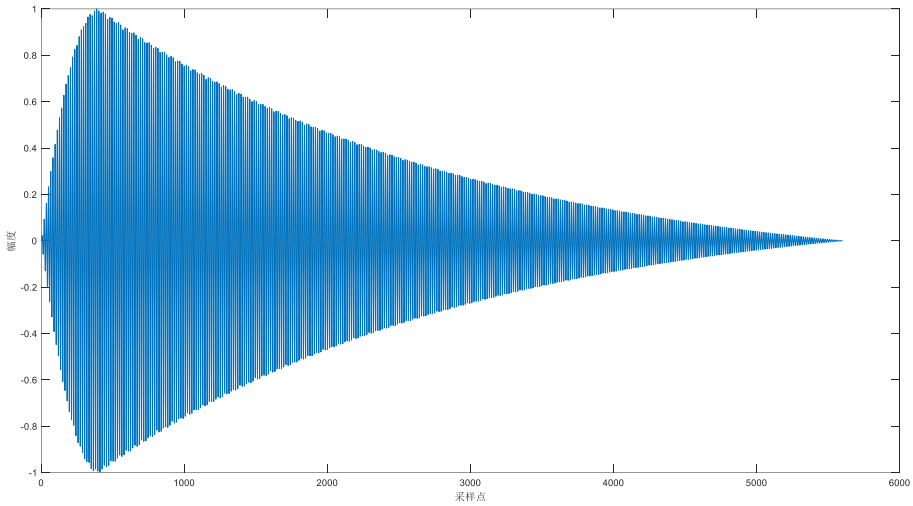


图 3 添加包络后单个乐音波形

合成音乐波形如图：

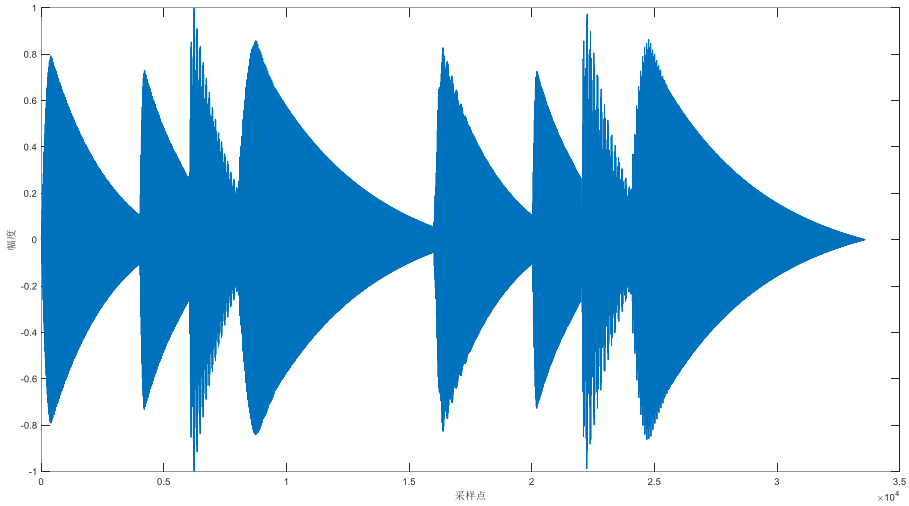


图 4 添加包络后音乐波形

“啪啪”的噪声被成功消除了。

（3）升高与降低音调

【题目描述】

请用最简单的方法将（2）中的音乐分别升高和降低一个八度。再难一些，请用 resample 函数（也可以用 interp 和 decimate 函数）将上述音乐升高半个音阶。

【主要思想】

利用改变采样与播放的关系可以实现音调的升高与降低。

保持原采样不变（8000Hz），播放和保存时按照 4000Hz 的采样频率，则时间延长一倍，音调降低一个八度；播放和保存时按照 16000Hz 的采样频率，则时间缩短为一半，音调升高一个八度。

使用 resample 函数对原音乐重采样，由于$2^{1/12} \approx 1.05946 \approx \frac{26487}{25000}$，按照$\frac{26487}{25000}$重采样并按照原采样率播放、保存，可以得到降半音的效果；按照$\frac{25000}{26487}$重采样并按照原采样率播放、保存，可以得到升半音的效果。

【核心代码】

```
x=audioread('东方红 2.wav').';%载入合成好的音乐
freq=8000;%原采样频率
sound(x,0.5*freq);%慢速播放降八度
pause(8);
sound(x,2*freq);%快速播放升八度
audiowrite('东方红_升八度.wav',x,2*freq);
pause(2);
y1=resample(x,26487,25000);%升高采样率，降半音
sound(y1,freq);
pause(4);
audiowrite('东方红_降半音.wav',y1,freq);
y2=resample(x,25000,26487);%降低采样率，升半音
sound(y2,freq);
pause(4);
audiowrite('东方红_升半音.wav',y1,freq);
```

【核心代码说明】

利用 resample 函数调整音频采样率，完整代码文件为 ex_1_2_1_3.m。

【运行结果】

得到音频文件“东方红_升八度.wav”、“东方红_降八度.wav”、“东方红_升半音.wav”、“东方红_降半音.wav”。

【结果分析】

通过调整采样率，改变了音乐播放的时间，实现了升高和降低音调的目的。

（4）增加谐波

【题目描述】

试着在（2）的音乐中增加一些谐波分量，听一听音乐是否更有“厚度”了？

【主要思想】

只是用正弦基频合成的声音音色特性很单薄，增加高次谐波分量以丰富音色，此处增加了二次谐波和三次谐波，模仿风琴的声音。

【核心代码】

```
seq=[15,15,17,10,8,8,5,10];%曲谱序列
time=[0.5,0.25,0.25,1,0.5,0.25,0.25,1];%时值(s)
overlap=[0.2,0.3,0.2,0.2,0.2,0.3,0.2,0.2];%迭接时间(s)
f0=220;%基准频率
freq=8000;%采样频率
amp=1;%幅度
output=zeros(1,freq*(sum(time)+overlap(length(overlap))));%输出向量
position=1;%辅助指针
for k=1:length(time)
    t=linspace(0,time(k)+overlap(k)-1/freq,freq*(time(k)+overlap(k)));%生成时间
    temp=amp*sin(2*pi*f0*pow2(seq(k)/12)*t);%生成乐音
    temp=temp+0.2*amp*sin(2*pi*2*f0*pow2(seq(k)/12)*t);%二次谐波
    temp=temp+0.3*amp*sin(2*pi*3*f0*pow2(seq(k)/12)*t);%三次谐波
    cover=[-100*(t(1:0.1*time(k)*freq-1)/time(k)).^2+20*(t(1:0.1*time(k)*freq-1)/time(k))...
        ,1.15652*exp(-2*(t(0.1*time(k)*freq:freq*(time(k)+overlap(k)))-
0.1*time(k))/(0.9*time(k)+overlap(k)))-0.15652];%生成包络
    output(position:position+freq*(time(k)+overlap(k))-
1)=output(position:position+freq*(time(k)+overlap(k))-1)+cover.*temp;%添加乐音
    position=position+freq*time(k);%移动指针
end
output=output/max(abs(output));%归一化
sound(output,freq);%播放
```

【核心代码说明】

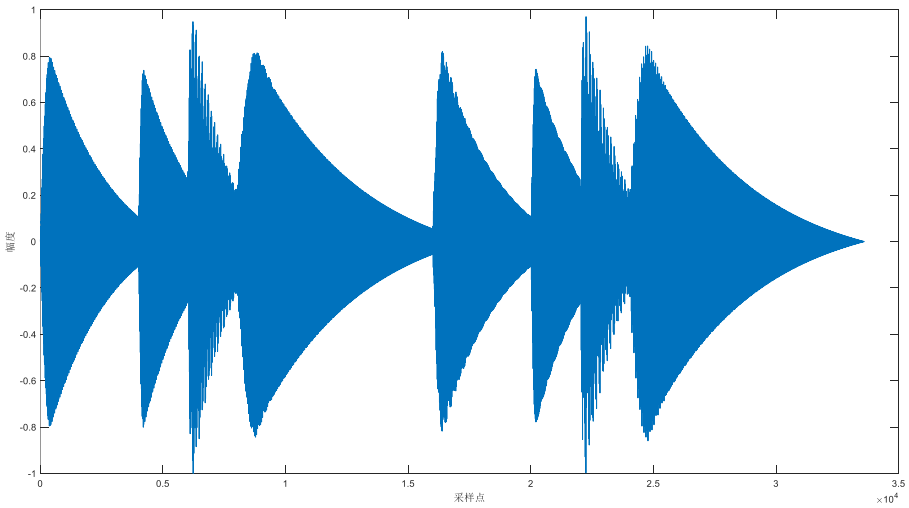
首先利用向 temp 数组中叠加各次谐波的正弦函数，叠加完成后 temp 数组与包络数组 cover 相乘得到最终的乐音，插入到 output 数组中得到最终乐曲。完整代码文件为 ex_1_2_1_4.m。

【运行结果】

得到音频文件“东方红 3.wav”。

【结果分析】

合成音乐波形及其局部放大如图：



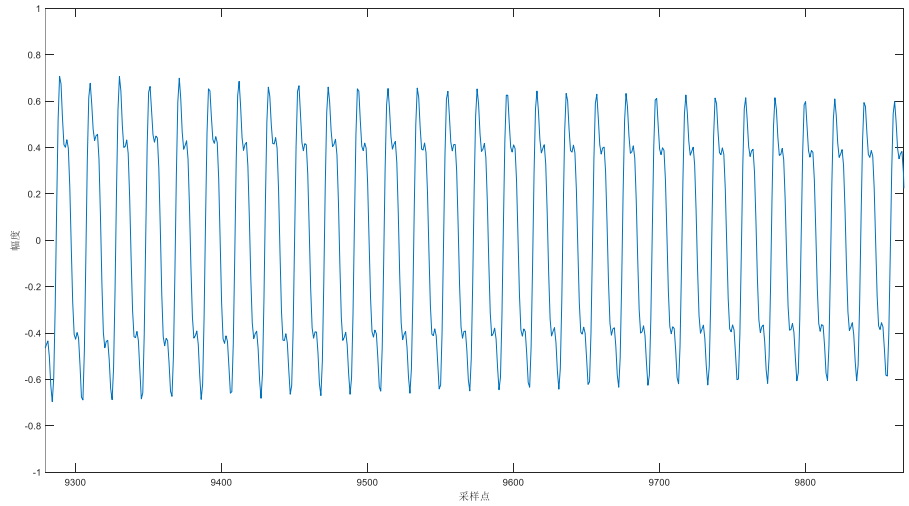


图 5 增加谐波分量后的波形图

可以听出相比于简单正弦声音，音色发生了改变，波形不再是正弦形态，音乐产生了一定的“厚度”。

（5）合成音乐

【题目描述】

自选其它音乐合成。

【主要思想】

为了尽可能真实地模仿钢琴的音色，先录制了一个由钢琴弹出的音符（已经合并为一个声道并 resample 为 8000Hz 的采样率），保存为文件“钢琴.wav”。用代码文件 ex_1_2_1_5_a.m 分析该音符，其波形为：

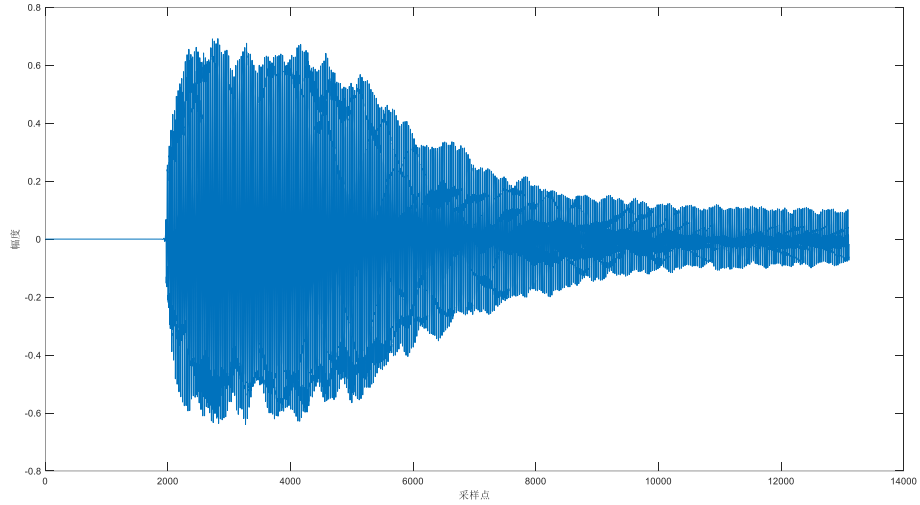


图 6 钢琴音符的波形

取其中一小段并在时域重复十次，对其使用快速傅里叶变换，得到频谱如图，由于采样率为 8000Hz，因此频谱图频率范围截止到 4000Hz[1]：

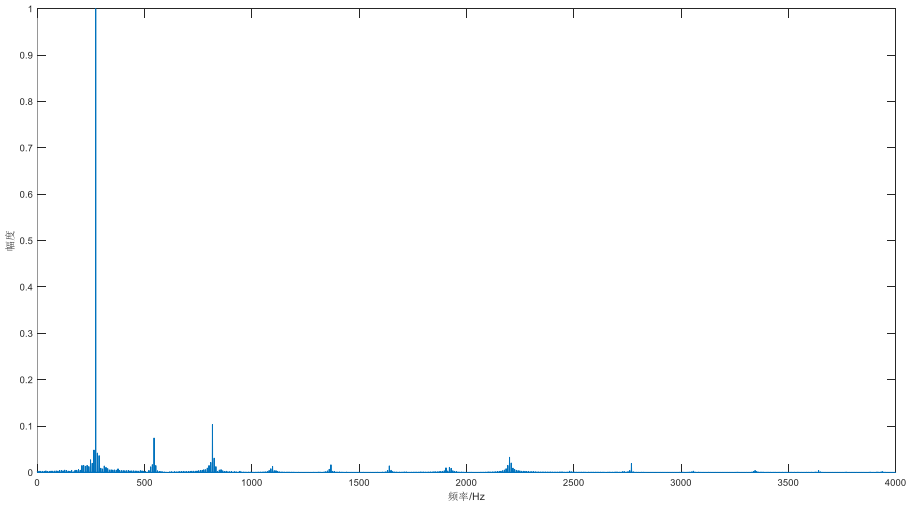


图 7 钢琴声音频谱图

从中得出各次谐波分量强度如下表：

表 2 钢琴音色各次谐波分量

谐波次数	基波	2	3	4	5	6	7	8	9	10
相对强度	1	0.074	0.103	0.013	0.016	0.014	0.011	0.033	0.001	0.019

根据曲谱，利用上述分析结果，合成《克罗地亚狂想曲》。

【核心代码】

```
output=zeros(1,freq*(sum(time{1})+1));%输出向量
for m=1:4
    position=1;%辅助指针
    for k=1:length(time{m})
        t=linspace(0,time{m}(k)+overlap{m}(k)-1/freq,freq*(time{m}(k)+overlap{m}(k)));%生成时间
        if seq{m}(k)~=-100%-100 表示休止符
            temp=amp{m}(k)*sin(2*pi*f0*pow2(seq{m}(k)/12)*t);%生成乐音
            temp=temp+0.074*amp{m}(k)*sin(2*pi*2*f0*pow2(seq{m}(k)/12)*t);%高次谐波
            temp=temp+0.103*amp{m}(k)*sin(2*pi*3*f0*pow2(seq{m}(k)/12)*t);
            temp=temp+0.013*amp{m}(k)*sin(2*pi*4*f0*pow2(seq{m}(k)/12)*t);
            temp=temp+0.016*amp{m}(k)*sin(2*pi*5*f0*pow2(seq{m}(k)/12)*t);
            temp=temp+0.014*amp{m}(k)*sin(2*pi*6*f0*pow2(seq{m}(k)/12)*t);
            temp=temp+0.011*amp{m}(k)*sin(2*pi*7*f0*pow2(seq{m}(k)/12)*t);
            temp=temp+0.033*amp{m}(k)*sin(2*pi*8*f0*pow2(seq{m}(k)/12)*t);
            temp=temp+0.001*amp{m}(k)*sin(2*pi*9*f0*pow2(seq{m}(k)/12)*t);
            temp=temp+0.019*amp{m}(k)*sin(2*pi*10*f0*pow2(seq{m}(k)/12)*t);
        else
            temp=zeros(1,length(t));
        end
```

[1] 后面的频谱图同样设置

```
        cover=[-100*(t(1:0.1*time{m}(k)*freq-1)/time{m}(k)).^2+20*(t(1:0.1*time{m}(k)*freq-
1)/time{m}(k))...
            ,1.15652*exp(-2*(t(0.1*time{m}(k)*freq:freq*(time{m}(k)+overlap{m}(k)))-
0.1*time{m}(k))/(0.9*time{m}(k)+overlap{m}(k)))-0.15652];%生成包络
        output(position:position+freq*(time{m}(k)+overlap{m}(k))-
1)=output(position:position+freq*(time{m}(k)+overlap{m}(k))-1)+cover.*temp;%添加乐音
        position=position+freq*time{m}(k);%移动指针
    end
end
output=output/max(abs(output));%归一化
sound(output,freq);%播放
```

【核心代码说明】

完整代码文件为 ex_1_2_1_5_b.m。主要使用了以下技术：

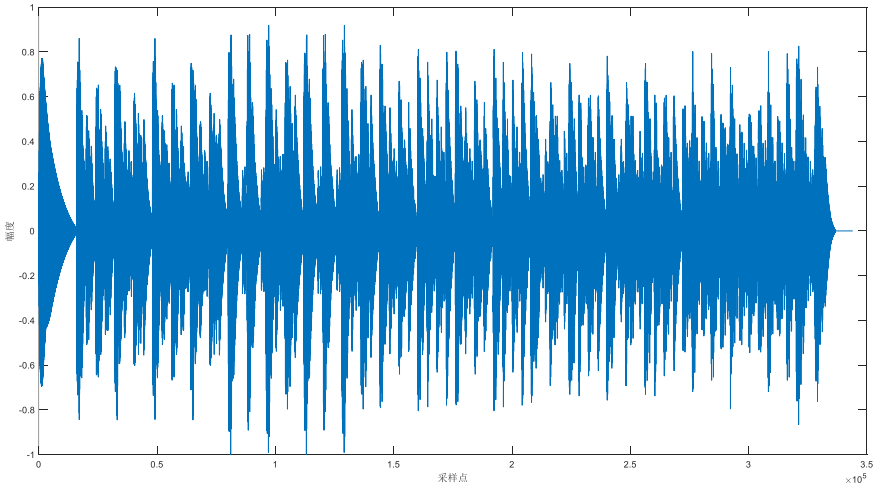
- 使用 4 组轨道，分别合成高音部分、低音部分等，最后进行叠加；
- 设置幅度向量 amp，每个音符赋予不同的强度，实现 4/4 拍“强-弱-次强-弱”的强弱关系；
- 设置迭接时间向量 overlap，每个音符之间赋予不同的迭接时间，实现曲谱中连音线效果；
- 预先为 output 向量分配空间，使用辅助指针向其中循环叠加音符，提升合成速度。

【运行结果】

得到音频文件“克罗地亚狂想曲.wav”。

【结果分析】

结果波形及局部放大如图。



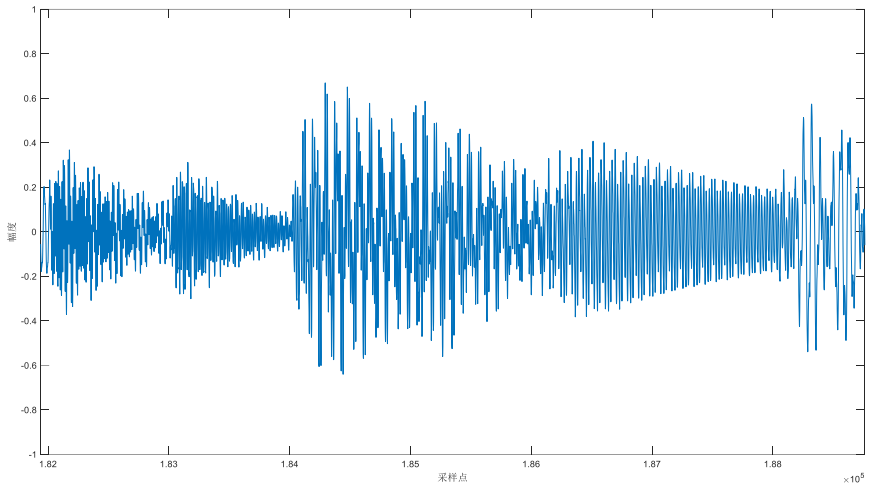


图 8 合成音乐波形

可以看出增加了谐波分量以及多个音符同时奏响叠加后，波形非简单的正弦波而具有复杂的周期振动模式，这种模式中蕴含了和弦、音色的特性。

1.2.2 用傅里叶级数分析音乐

（6）分析 fmt.wav 文件中的音乐

【题目描述】

载入 fmt.wav 文件，播放出来听听效果如何？是否比刚才的合成音乐真实多了？

【主要思想】

fmt.wav 文件记录了吉他演奏的真实乐曲，听上去比合成的音乐更有真实感，从时域和频域分析这段音乐，观察该乐曲与合成乐曲的区别。

【核心代码】

```
y=audioread('fmt.wav');%读取音乐文件
sound(y,8000);%播放
plot(y);%绘制波形
xlabel('采样点');
ylabel('幅度');
[t,omg,FT,IFT]=prefourier([0,0.5-0.5/4000],4000,[-10000,10000],50000);
F=FT*y(5001:9000);
plot(omg,abs(F));
xlabel('\omega')
ylabel('幅度');
```

【核心代码说明】

载入并分析 fmt.wav 中的文件，首先播放音频，再画出时域波形图，最后利用 prefourier 函数对音频中一个小片段进行傅里叶变换，得到频谱图。完整代码文件为 ex_1_2_2_6.m。

【运行结果】

运行得到时域波形图和频谱图，见“结果分析”。

【结果分析】

其波形以及局部放大如图，可以看出音乐的波形非常复杂，呈现大量不同频率的谐波叠加，同时混合有一定的噪声。

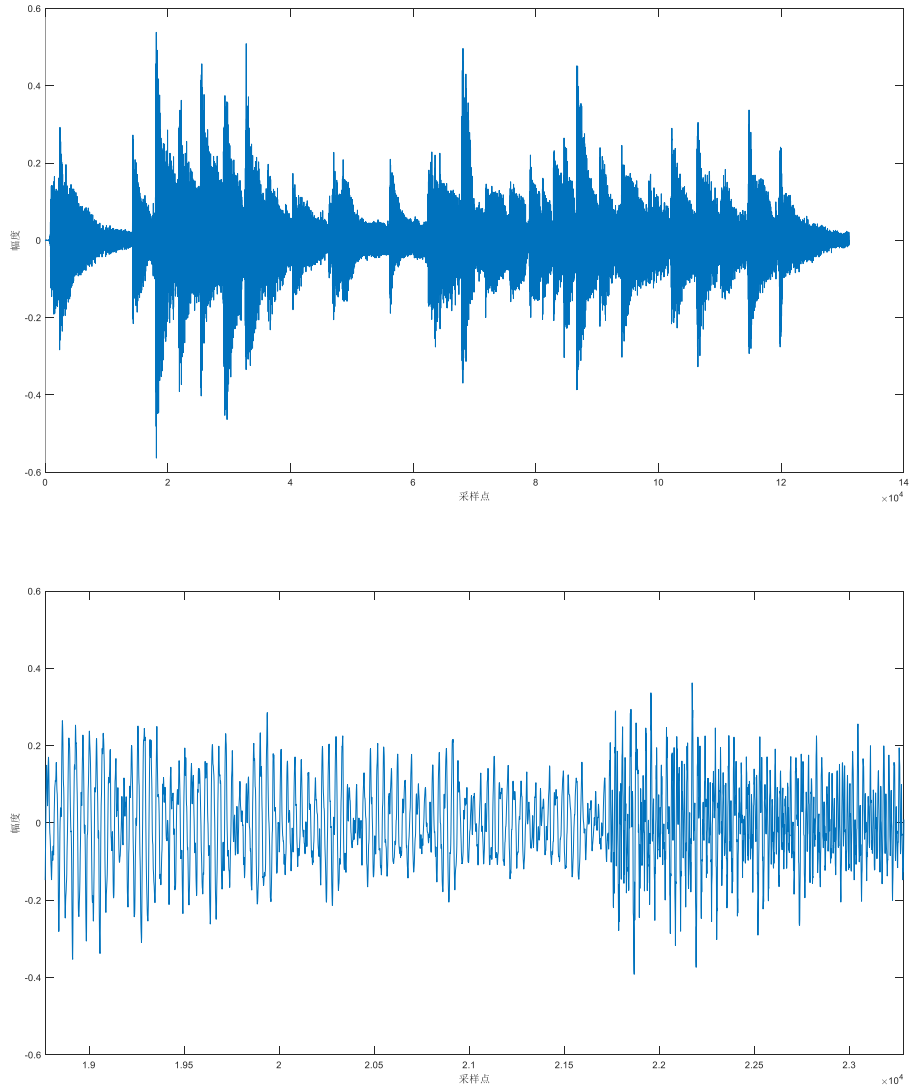


图 9 fmt.wav 中音乐波形

截取一个小片段做傅里叶变换，频谱结果如图（取绝对值）：

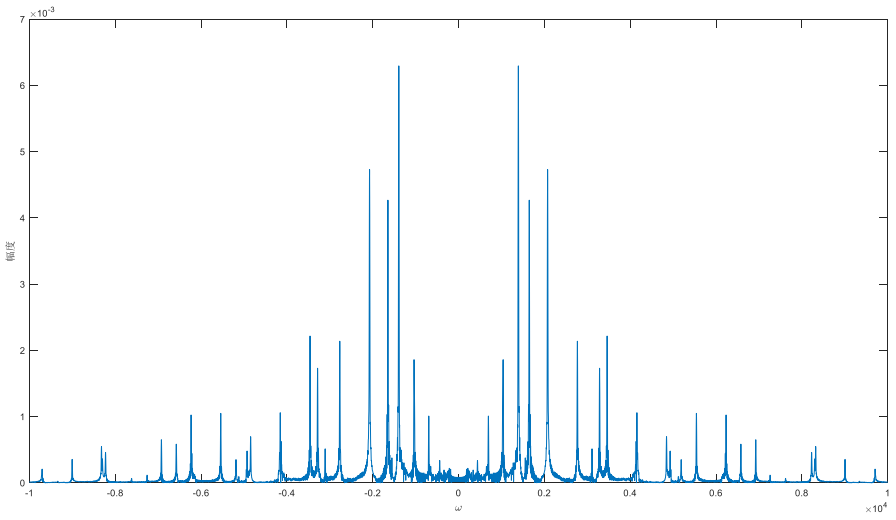


图 10 fmt.wav 中音乐的频谱

可以观察到，选取的片段为和弦，基音包含多个频率，且每个基音频率强度又不同，同时频谱中有异常复杂的谐波分量，也含有不是谐波分量的各种噪声，正是这种复杂的频谱结构，使得音乐有丰富的音色，听起来比合成的音乐真实多了。

（7）预处理 realwave

【题目描述】

你知道待处理的 wave2proc 是如何从真实值 realwave 中得到的么？这个预处理过程可以去除真实乐曲中的非线性谐波和噪声，对于正确分析音调是非常重要的。提示：从时域做，可以继续使用 resample 函数。

【主要思想】

根据 realwave 的波形可以看出这是从原始音乐中取 10 个周期得到的，但是由于噪声和非线性谐波的存在，每个周期大致形状相同但是波形并不完全重合。由于噪声多具有随机性，可以近似认为噪声的均值为 0，由此想到通过对 10 个周期取平均消去噪声。

由于 realwave 含有 243 个采样点，无法等分为 10 份取平均，因此先使用 resample 函数进行插值，获得 10 倍数目即 2430 个采样点，这时每个周期含有 243 个采样点。对 10 个周期取平均、做周期延拓，得到处理后的数据，再使用 resample 函数恢复 243 个采样点。

【核心代码】

```
load('Guitar.mat');%载入数据
y=resample(realwave,2430,243);%插值增加采样点
y_bar=zeros(243,1);
for k=1:10
    y_bar=y_bar+y(243*k-242:243*k);%取 10 个周期平均
end
y_bar=y_bar/10;%取平均
for k=1:10
    y(243*k-242:243*k)=y_bar;%周期延拓
end
y=resample(y,243,2430);%恢复 243 个采样点
error=y-wave2proc;%计算误差
```

【核心代码说明】

用 resample 函数对 realwave 插值，新建 y_bar 数组，向其中累计 10 个周期的数据并取平均。对 y_bar 做周期延拓，恢复 243 个采样点，最后计算得到的结果与给出的 wav2proc 的误差。完整代码文件为 ex_1_2_2_7.m。

【运行结果】

得到计算结果与给出的 wav2proc 的误差，见“结果分析”。

【结果分析】

计算处理后得到的波形与给出的 wav2proc 的差，绘制如图：

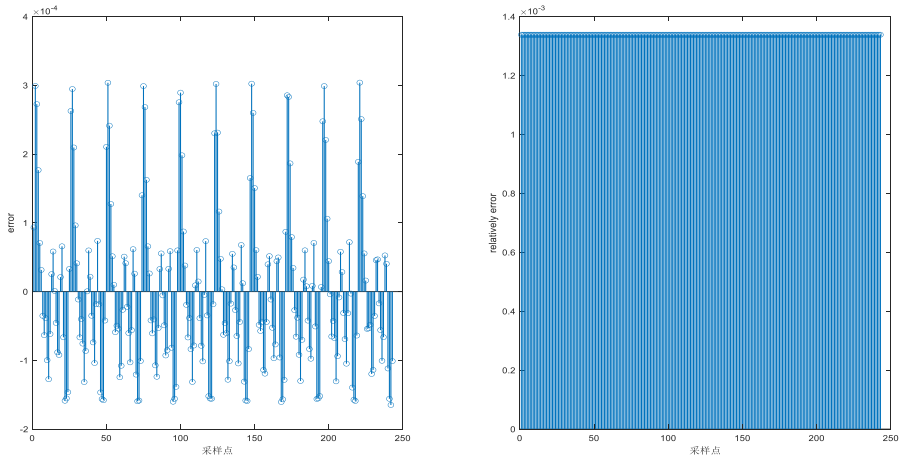


图 11 数据处理误差

可以发现相对误差恒定为一个常数（且该常数很小），按照上述方法处理后的数据与给出的理论值只在幅度上有微小拉伸，不影响后续频域的分析。

（8）分析基音与谐波

【题目描述】

这段音乐的基频是多少？是哪个音调？请用傅里叶级数或者变换的方法分析它的谐波分量分别是什么。可选的方法是增加时域的数据量，即再把时域信号重复若干次，看看这样是否效果好多了？请解释之。

【主要思想】

首先计算这段声音的基音周期，音频采样率为 8kHz，共有 243 个采样点，对应 10 个周期，由此周期T为

$$T = \frac{243}{8000} \times \frac{1}{10} = 0.0030375s$$

频率f为

$$f = \frac{1}{T} \approx 329.22Hz$$

对应 E4[2]音，下面用两种不同方法分析各阶谐波分量的强度。

方法一：傅里叶级数分析

【核心代码】

```
load('Guitar.mat');%载入数据并近似取一个周期
f=wave2proc(1:24);

T=0.0030375;%基音周期
omg=2*pi/T;%基频
N=24;%时域抽样点数
t=linspace(-T/2,T/2-T/N,N);%生成抽样时间
k=[-100:100].';%谐波次数
```

[2] 按照钢琴键的记法，中央 C 记为 C4

```
F=1/N*exp(-j*omg*k*t)*f;%求傅里叶级数
a0=F(101);
ak=F(102:201)+F(100:-1:1);%由指数形式转为三角形式
a=[a0;ak];
b=[0;1j*(F(102:201)-F(100:-1:1))];
stem(abs(a+1j*b)/max(abs(a+1j*b)));%归一化并画图像
xlabel('k');
ylabel('幅度');
```

【核心代码说明】

取出一个周期的信号求傅里叶级数（前 24 个采样点），先求出指数形式的傅里叶级数，再根据指数形式傅里叶级数与三角形式傅里叶级数的关系得到三角形式傅里叶级数。最后画出傅里叶级数图像。完整代码文件为 ex_1_2_2_8_a.m。

【运行结果】

得到傅里叶级数图像和数值大小，见“结果分析”。

【结果分析】

前 13 个傅里叶系数如图（画出的是$\sqrt{a_n^2+b_n^2}$）：

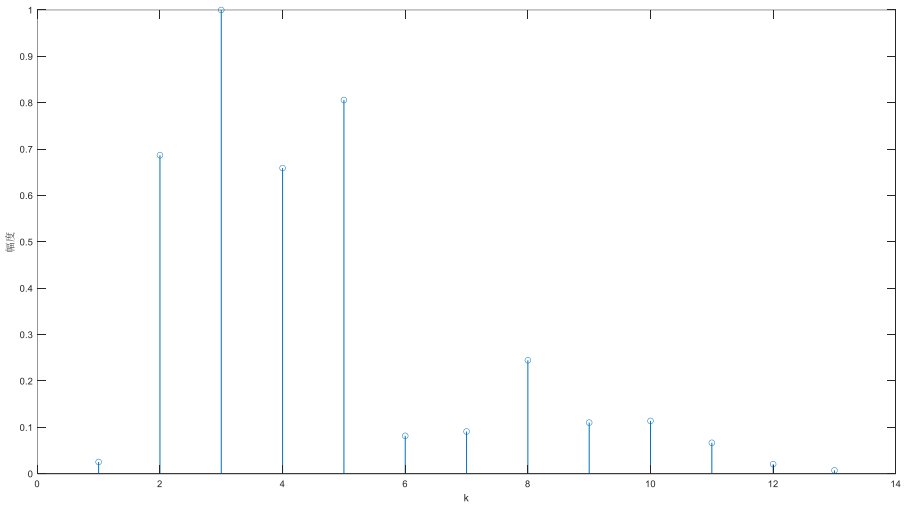


图 12 一个周期信号的前 13 个傅里叶系数

根据傅里叶系数，直流分量、基波和前 11 次谐波分量强度为：

表 3 傅里叶系数

谐波次数	直流	基波	2	3	4	5	6
相对强度	0.0253	0.6867	1.0000	0.6591	0.8058	0.0817	0.0910
谐波次数	7	8	9	10	11	12	
相对强度	0.2446	0.1102	0.1138	0.0666	0.0206	0.0072	

之所以选取前 13 个系数是由于周期中只有 24 个采样点，因此最多可以允许 12 次谐波的存在。时域抽样即在时域乘上冲激串，在频域与冲激串卷积，因而 13 个之后的系数呈现周期性变化，如图：

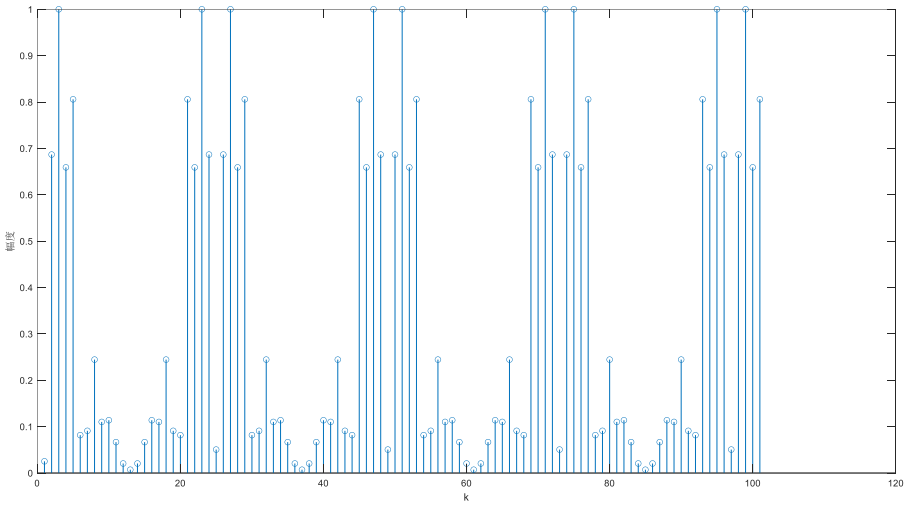


图 13 傅里叶级数周期性变化

由于 10 个周期中只有 243 个采样点，因此一个周期并非整数个采样点，取 24 个采样点作为一个周期只是一种近似处理，因此这种分析方式得到的谐波强度是不准确的。下面将采用傅里叶变换的方法得到更加准确的结果。

方法二：傅里叶变换分析

【核心代码】

```
load('Guitar.mat');%载入数据
f=zeros(10*length(wave2proc),1);
for k=1:10
    f(243*k-242:243*k)=wave2proc;%重复 10 次
end

[t,omg,FT,IFT]=prefourier([0,0.30375-0.30375/2430],2430,[-25000,25000],100000);
F=FT*f;
plot(omg/2/pi,abs(F)/max(abs(F)));%归一化并画图
xlabel('频率/Hz');
ylabel('幅度');
xlim([0,4000]);
```

【核心代码说明】

将原本 10 个周期的波形重复 10 次，再利用 prefourier 函数生成傅里叶变换矩阵，对重复 10 次后的波形做傅里叶变换，画出频谱图。完整代码文件为 ex_1_2_2_8_b.m。

【运行结果】

得到 wave2proc 的频谱图，见“结果分析”。

【结果分析】

首先使用预设函数 prefourier 进行傅里叶变换，此函数可以任意设置频域采样间隔，得到高分辨率的频域结果，直接对这段含有 10 个周期的音频作傅里叶变换，得到频谱如下图：

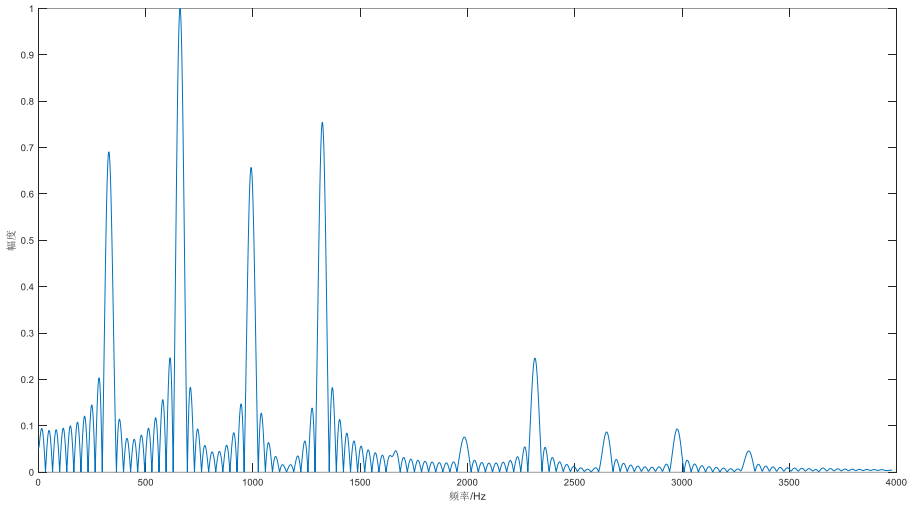


图 14 直接傅里叶变换结果

可以看出每个峰值处的谱线形状不是δ函数，而是Sa函数（这里取了模）。这是由于 10 个周期的序列是有限长度的，相当于无限长周期序列时域上与矩形窗函数相乘，频域上与矩形窗函数的傅里叶变换，即Sa函数做卷积。无限长周期序列频谱为若干δ函数，与Sa函数做卷积后即得到上图结果。

为提高频谱的精度，将原本 10 个周期音频重复 10 次后再做傅里叶变换。将数据重复 10 次相当于时域矩形窗函数宽度扩大 10 倍，其傅里叶变换对应的Sa函数变“瘦”了，因此可以更好地趋近δ函数，提高了精度，其结果如下图。

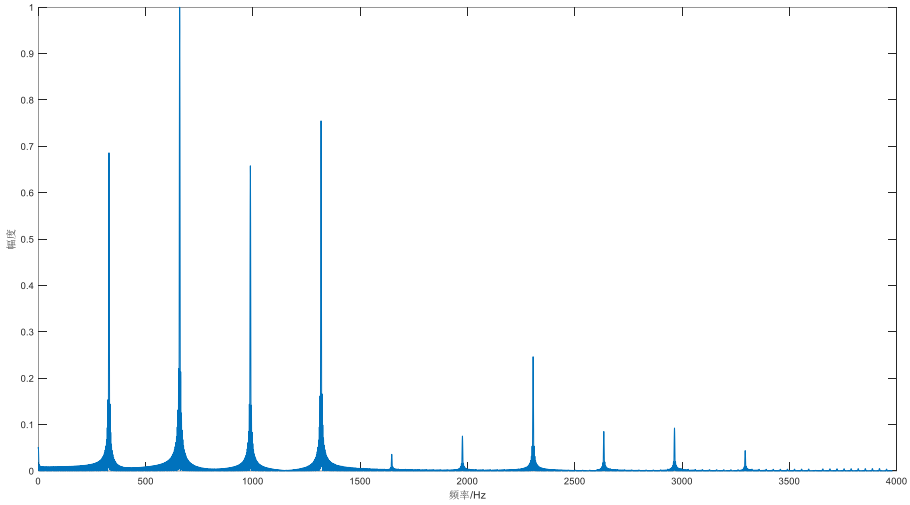


图 15 重复数据后傅里叶变换结果

使用快速傅里叶变换，可以得到下图结果，完整代码文件为 ex_1_2_2_8_c.m，可以看出峰值位置和强度是一致的，但分辨率降低了很多，这有利于使用代码自动定位峰值，编写函数位于文件 locatepeak.m，输入采样向量 X、对应的幅度 Y、幅度敏感阈值 thY，采样敏感阈值 thX，返回各峰值的位置和相对强度，定位结果同样在下图标出。

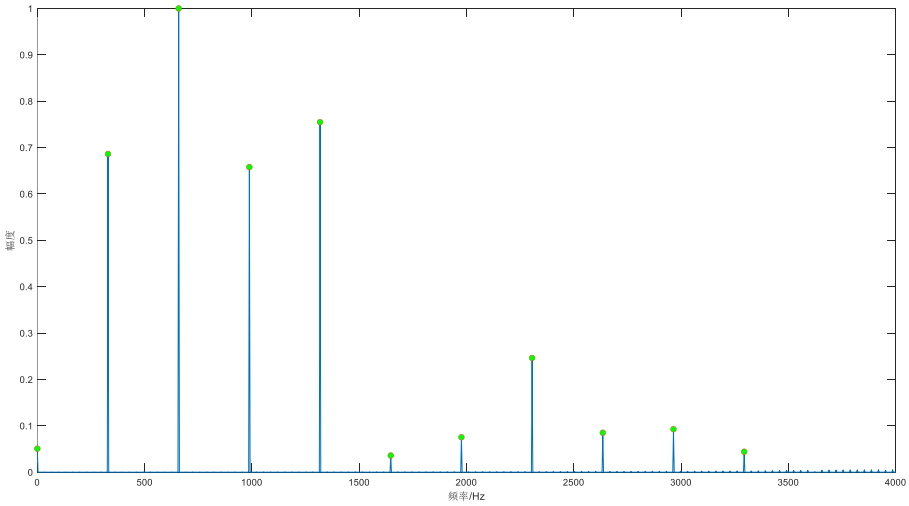


图 16 快速傅里叶变换结果

峰值定位结果为：

表 4 谐波分量强度

谐波次数	直流	基波	2	3	4	5
频率(Hz)	0	329.35	658.70	988.06	1317.41	1646.77
相对强度	0.0507	0.6863	1.0000	0.6580	0.7548	0.0359
谐波次数	6	7	8	9	10	
频率(Hz)	1976.12	2305.48	2634.83	2964.18	3293.53	
相对强度	0.0754	0.2463	0.0851	0.0927	0.0441	

谐波分量强度与之前傅里叶级数结果基本一致，但这里的结果更加精确，同时基波频率为 329.35Hz，为 E4 音，与之前结论一致。

（9）自动分析音符与节拍

【题目描述】

再次载入 fmt.wav，写一段程序，自动分析出这段乐曲的音调和节拍。

【主要思想】

按照以下两步实现对 fmt.wav 中音乐乐谱的自动提取：首先分析时域波形的包络，根据包络的变化分离每个乐音；其次对每个乐音应用类似（8）中方法，通过傅里叶变换得到其频谱，进而得出其对应音符和各次谐波分量强度。

step1：时域包络提取与音符分隔

【核心代码】

```
y=audioread('fmt.wav');
t=[1:length(y)].';%生成时间序列
peak=locatepeak(t,y,0.02,150);%调用 locatepeak 函数寻找时域峰值，确定包络
figure;
hold on;
box on;
xlabel('采样点')
ylabel('幅度')
plot(y);
plot(peak(:,1),peak(:,2),'LineWidth',2);%画出原始包络
```

```
peak=lowpass(peak,0.6);%对原始包络低通滤波
plot(peak(1:400,1),peak(1:400,2),'LineWidth',2);%画出滤波后的包络


start=locatepeak(peak(1:400,1),peak(1:400,2),0.1,1000);%定位音调起始点
time_last=zeros(length(start(:,1)),1);%各音调持续时间
for h=1:length(start(:,1))-1
    time_last(h)=(start(h+1,1)-start(h,1))/8000;
end
time_last(length(start(:,1)))=(length(y)-start(length(start(:,1))-1,1))/8000;
scatter(start(:,1),start(:,2),'MarkerFaceColor','g');%画出起始点
legend('时域波形','原始包络','低通滤波后包络','音调起始点');
```

【核心代码说明】

首先使用之前编写的 locatepeak 函数，通过寻找时域信号的局部峰值来初步确定属于信号的包络，通过调节采样敏感阈值参数 thX 可以达到较好的效果。观察发现原始包络在每个音符的其实位置存在峰值，但是有较多的毛刺，不利于分隔音符，因此对原始包络进行低通滤波，得到平滑的包络后对包络使用 locatepeak 函数，确定包络的峰值即每个音符的起始位置，同时可以得出每个音调持续的时间，即乐曲的节奏。完整代码文件为 ex_1_2_2_9.m。

【运行结果】

得到时域波形包络和各个音符起始位置图，见“结果分析”；共分析出 29 个音调，得到每个音符持续时间，储存在 time_last.mat 中。time_last.mat 为有 29 个元素的向量，每个元素表示每个音调持续的时间。

【结果分析】

时域分析结果如下，可以看出该方法能够比较准确地检测出时域波形的包络，并确定了各个音符起始的位置。

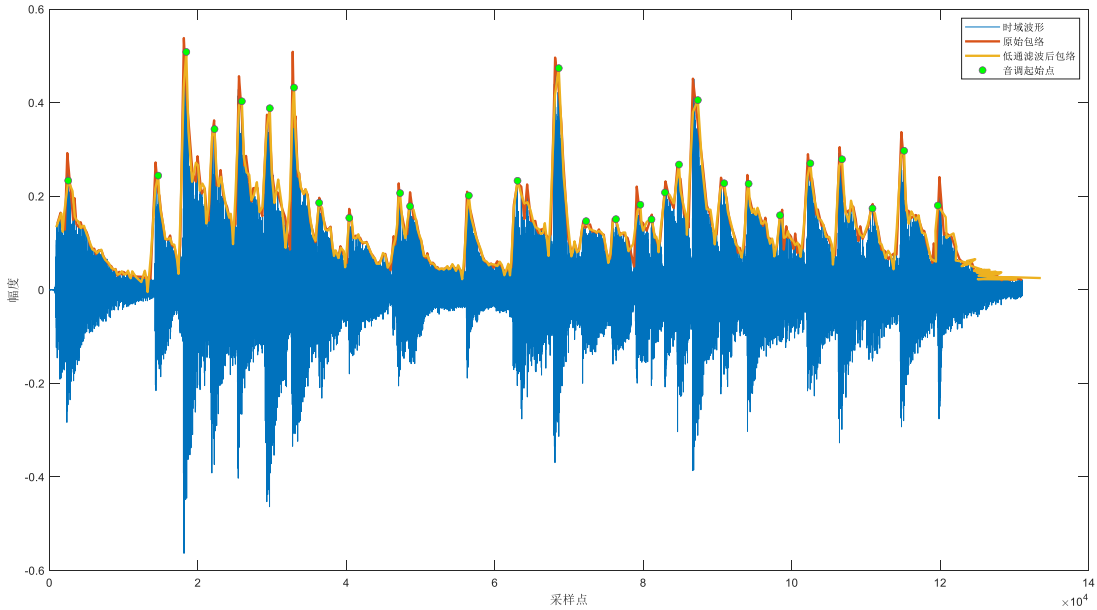


图 17 时域包络分析

step2：频域分析

【核心代码】

```
note=zeros(length(start(:,1)),4,12);
% 储存音符信息，每页表示一个音调，每行表示一个音符，第一列为基波频率，
% 第 2-11 列依次为基波、2-10 次谐波的相对强度，第 12 列为该音符相对于 220Hz 偏移的半音数
for k=1:length(start(:,1))%逐个分析音调
    sample0=y(start(k,1)+1000:start(k,1)+1999);%从中间位置取样长度为 1000 的片段
    sample=zeros(100000,1);
    for h=1:100
        sample(1000*h-999:1000*h)=sample0;%重复 100 次提高精确度
    end
    N=100000;%时域采样点数
    T=12.5;%时间范围
    Omg=2*pi*N/T;%频域范围
    omg=linspace(0,Omg,N).';%生成频域采样
    t=linspace(0,T-T/N,N).';%生成时域采样
    f1=sample.*exp(-1j*omg(1)*t);%辅助函数
    F1=T*exp(1j*omg(1)*t(1))/N*fft(f1);%快速傅里叶变换
    F_fft=F1.*exp(-1j*omg*t(1));%由 fft 结果得到频谱
    peak=locatepeak(omg,abs(F_fft),0.04,150);%定位峰值
    peak=peak(1:length(peak(:,1))/2,:);
    peak(:,1)=peak(:,1)/2/pi;%频率单位转换为 Hz

    figure;
    hold on;
    box on;
    plot(omg/2/pi,abs(F_fft));
    scatter(peak(:,1),peak(:,2),'MarkerFaceColor','g');%标记峰值
    xlabel('频率/Hz');
    ylabel('幅度');
    xlim([0,4000]);
    ylim([0,1]);

    note_temp=zeros(20,12);
    % 临时储存音符信息，每行表示一个音符，第一列为基波频率，
    % 第 2-11 列依次为基波、2-10 次谐波的相对强度，第 12 列为各阶谐波的能量和
    pointer=1;%辅助指针
    for h=1:length(peak(:,1))
        flag=0;%标记是否已经有该音符
        for m=1:20
            if note_temp(m,1)~=0 & abs(round(peak(h,1)/note_temp(m,1))*note_temp(m,1)-
peak(h,1))/peak(h,1)<=0.05
                order=round(peak(h,1)/note_temp(m,1));%计算谐波阶次
                if order<=10%仅考虑前 10 阶谐波
```

```
                    note_temp(m,order+1)=peak(h,2);%如果为已有音符的谐波，则存入谐波分量中
                end
                flag=1;
            end
        end
        if flag==0%目前没有的音符存入基波分量中
            note_temp(pointer,1)=peak(h,1);
            note_temp(pointer,2)=peak(h,2);
            pointer=pointer+1;%移动辅助指针
        end
    end
    note_temp(:,12)=sum(note_temp(:,2:11),2);%求每个音符谐波能量和，存入第 12 列
    for h=1:4%选取总能量最大、且大于一定阈值的四个音符作为最终结果
        [max_value,max_pointer]=max(note_temp(:,12));
        if max_value>=0
            note(k,h,1:11)=note_temp(max_pointer,1:11);
            note(k,h,12)=round(12*log2(note(k,h,1)/220));
            note_temp(max_pointer,:)=[];
        end
    end
end
```

【核心代码说明】

根据 step1 中对时域包络的分析，将乐曲划分成了若干音调，下面分析每个音调中含有的音符和每个音符对应的谐波分量，完整代码文件仍为 ex_1_2_2_9.m。首先取每个音调中间部分 1000 个采样点长度的片段，重复 100 次以提高快速傅里叶变换的精确度（在第 8 小题中已经说明），调用 locatepeak 函数定位每个音调频谱的峰值。

首先，由于乐器演奏有强弱区别，如果对幅度做归一化处理则损失了每个音调之间的强弱区分信息，因此在此处不做归一化处理。定位峰值时需要选择一个阈值，大于该阈值的认为是音乐中有效的峰值，小于该阈值则认为是噪声。阈值的设置需要经过多次尝试，如果设置过低则会出现过多峰值，干扰后续步骤（左图），如果设置过高则许多幅度较小的谐波分量被忽略，不能准确得到结果（右图）。

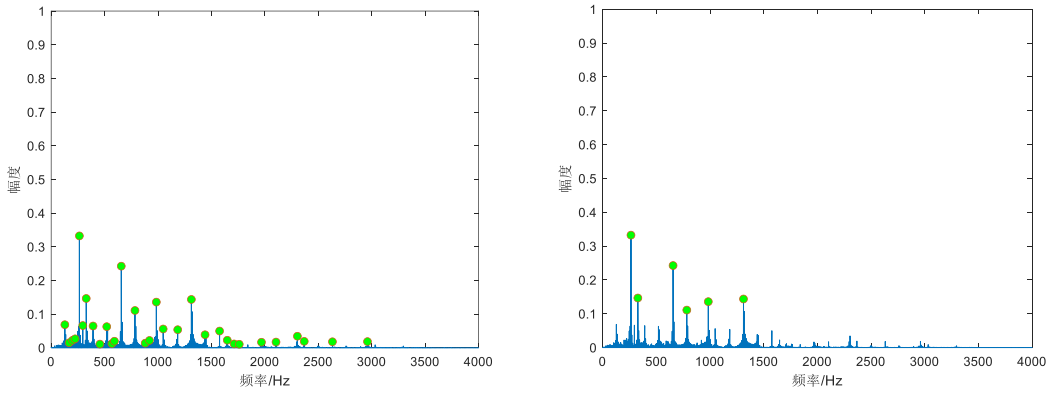


图 18 峰值检测阈值设置不当结果

选择合理的阈值后，比较明显的峰值都被检测出来，而幅度很小的则被忽略，结果如下（此处仅画出一个音调的频谱图，运行程序可以看到所有音调的分析结果）

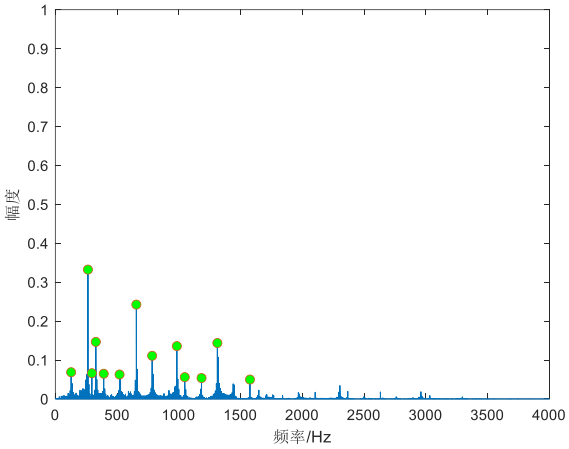


图 19 合适阈值下的检测结果

将检测到的峰值进行分组，如果某峰值频率为已有峰值的整数倍，则分为已有峰值的谐波；若频率不是任何已有峰值的整数倍，则记录为新的基波分量。

将所有峰值全部分组后，结合吉他的演奏方法（无名指、中指、食指分别弹拨#1、#2、#3 弦，拇指弹拨#4、#5、#6 三根弦），正常情况同时不会超过四个音符奏响，故将每个音调中基波和各次谐波总能量最大的四个取出，作为该音调所含有的音符。

【运行结果】

得到 step1 中分离出的 29 个音调的频率信息，结果保存为 note.mat。note.mat 为 29*4*12 的矩阵，储存音符信息，每页表示一个音调，每行表示一个音符，第一列为基波频率，第 2-11 列依次为基波、2-10 次谐波的相对强度，第 12 列为该音符相对于 220Hz 偏移的半音数，如第 12 列数值为-Inf，则表示该音符为“空”。

【结果分析】

step1、2 结果汇总如下（音符 1、2、3、4 按照能量递减排序）：

表 5 吉他音乐分析结果

序号	持续时间(s)	音符 1	音符 2	音符 3	音符 4
1	1.516	A3	E3	C4	——
2	0.473	B3	E3	A3	——
3	0.475	A2	E3	B3	——
4	0.461	D4	A2	B3	——
5	0.471	E4	A3	B3	——
6	0.408	G3	A3	E4	D4
7	0.422	A3	F3	E4	——
8	0.508	F3	D4	A3	B3
9	0.856	D4	F3	——	——
10	0.168	E3	^{b}A3	B3	——
11	0.990	^{b}E3	B3	^{b}A3	——
12	0.820	E3	——	——	——
13	0.694	A2	E3	C4	G3
14	0.461	^{b}E3	A3	B3	^{b}D5
15	0.503	A3	E4	^{b}D5	——

16	0.411	A3	E4	bD5	bA4
17	0.188	C3	G3	F4	——
18	0.227	C3	bE4	bA3	F4
19	0.236	C3	E4	bA2	D4
20	0.317	bE4	D3	G3	B3
21	0.443	C3	E4	D4	——
22	0.412	C3	E4	D4	——
23	0.529	D3	B3	A3	——
24	0.511	D3	C4	A3	E4
25	0.533	F3	B3	——	——
26	0.513	A3	F3	B3	——
27	0.532	E3	B3	A3	——
28	0.568	A3	E3	——	——
29	1.993	bA3	E3	——	——

由于吉他演奏中每个音调之间存在比较多的重叠部分，每一个音调中大都包含了上一个音调的音符。如果下一个音调中有与上一个音调相同的音符（上表中多处存在这种现象），则很难区分该相同音符来自于上一音调的残留还是曲谱中连续两个音调都弹奏该音符。尝试一种办法是通过能量区分两种音符：残留的音符经过衰减能量较低，新弹奏出的音符能量较高。但是不同音符弹奏力度差异很大，这种方法仍然存在一定缺陷。此方法能量阈值在代码第 83 行可以调节，此处暂时置为 0，不影响分析各个音符的谐波分量。

通过对乐曲中音调的分离，得到部分主要音符各次谐波的关系，此处强度已做归一化处理：

表 6 主要音符谐波强度

音名		C3	D3	E3	F3	G3	A3
基频(Hz)		128	144	168	176	192	224
相对强度	基波	0.3749	0.3529	0.2354	1.0000	0.4201	1.0000
	2 次	0.9828	1.0000	1.0000	0.2729	1.0000	0.3214
	3 次	1.0000	0.3729	0	0	0.4718	0.2104
	4 次	0	0.6826	0.4056	0.1483	0	0
	5 次	0.8740	0	0	0	0	0
	6 次	0.4855	0.1600	0.3381	0.1732	0.0960	0
	7 次	0	0	0	0	0.1324	0
	8 次	0	0	0	0.1242	0	0
	9 次	0	0	0	0	0	0
	10 次	0	0	0	0	0	0
音名		B3	C4	D4	E4	F4	bD5
基频(Hz)		248	264	296	328	352	552
相对强度	基波	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	2 次	0.3020	0.7272	0.7573	0.5074	0.2148	0
	3 次	0	0.5067	0.2488	0.7650	0	0
	4 次	0.1486	0	0.1438	0.8049	0.2260	0
	5 次	0	0	0.1306	0	0	0
	6 次	0.1068	0	0.1447	0	0	0

	7 次	0	0	0	0.3316	0	0
	8 次	0	0	0	0	0	0
	9 次	0	0	0.1110	0	0	0
	10 次	0	0	0	0	0	0

根据这里分析出的不同音符各次谐波分量的相对关系，可以比较逼真地合成出用这把吉他（音准与标准音略有差异）演奏的乐曲，具体操作间下一小节。

1.2.3 基于傅里叶级数的合成音乐

（10）用傅里叶级数合成《东方红》

【题目描述】

用（8）计算出来的傅里叶级数再次完成第（4）题，听一听是否像演奏 fmt.wav 的吉他演奏出来的？

【主要思想】

此处近似认为吉他每个音调的谐波分量强度都相同，利用（8）得到的傅里叶级数设置谐波强度，可以得到接近吉他演奏的乐曲。

【核心代码】

```
seq=[15,15,17,10,8,8,5,10];%曲谱序列
time=[0.5,0.25,0.25,1,0.5,0.25,0.25,1];%时值(s)
overlap=[0.2,0.3,0.2,0.2,0.2,0.3,0.2,0.2];%迭接时间(s)
f0=220;%基准频率
freq=8000;%采样频率
amp=1;%幅度
output=zeros(1,freq*(sum(time)+overlap(length(overlap))));%输出向量
position=1;%辅助指针
for k=1:length(time)
    t=linspace(0,time(k)+overlap(k)-1/freq,freq*(time(k)+overlap(k)));%生成时间
    temp=0.6863*amp*sin(2*pi*f0*pow2(seq(k)/12)*t);%生成乐音
    temp=temp+1.0000*amp*sin(2*pi*2*f0*pow2(seq(k)/12)*t);%高次谐波
    temp=temp+0.6580*amp*sin(2*pi*3*f0*pow2(seq(k)/12)*t);
    temp=temp+0.7548*amp*sin(2*pi*4*f0*pow2(seq(k)/12)*t);
    temp=temp+0.0359*amp*sin(2*pi*5*f0*pow2(seq(k)/12)*t);
    temp=temp+0.0754*amp*sin(2*pi*6*f0*pow2(seq(k)/12)*t);
    temp=temp+0.2463*amp*sin(2*pi*7*f0*pow2(seq(k)/12)*t);
    temp=temp+0.0851*amp*sin(2*pi*8*f0*pow2(seq(k)/12)*t);
    temp=temp+0.0957*amp*sin(2*pi*9*f0*pow2(seq(k)/12)*t);
    temp=temp+0.0441*amp*sin(2*pi*10*f0*pow2(seq(k)/12)*t);
    cover=[-100*(t(1:0.1*time(k)*freq-1)/time(k)).^2+20*(t(1:0.1*time(k)*freq-1)/time(k))...
        ,1.15652*exp(-2*(t(0.1*time(k)*freq:freq*(time(k)+overlap(k)))-
0.1*time(k))/(0.9*time(k)+overlap(k)))-0.15652];%生成包络
    output(position:position+freq*(time(k)+overlap(k))-
1)=output(position:position+freq*(time(k)+overlap(k))-1)+cover.*temp;%添加乐音
    position=position+freq*time(k);%移动指针
end
output=output/max(abs(output));%归一化
```

```
sound(output,freq);%播放
```

【核心代码说明】

首先利用向 temp 数组中叠加各次谐波的正弦函数，叠加完成后 temp 数组与包络数组 cover 相乘得到最终的乐音，插入到 output 数组中得到最终乐曲。完整代码文件为 ex_1_2_1_10.m。

【运行结果】

得到音频文件“东方红 4.wav”。

【结果分析】

合成乐曲的波形以及局部放大如下：

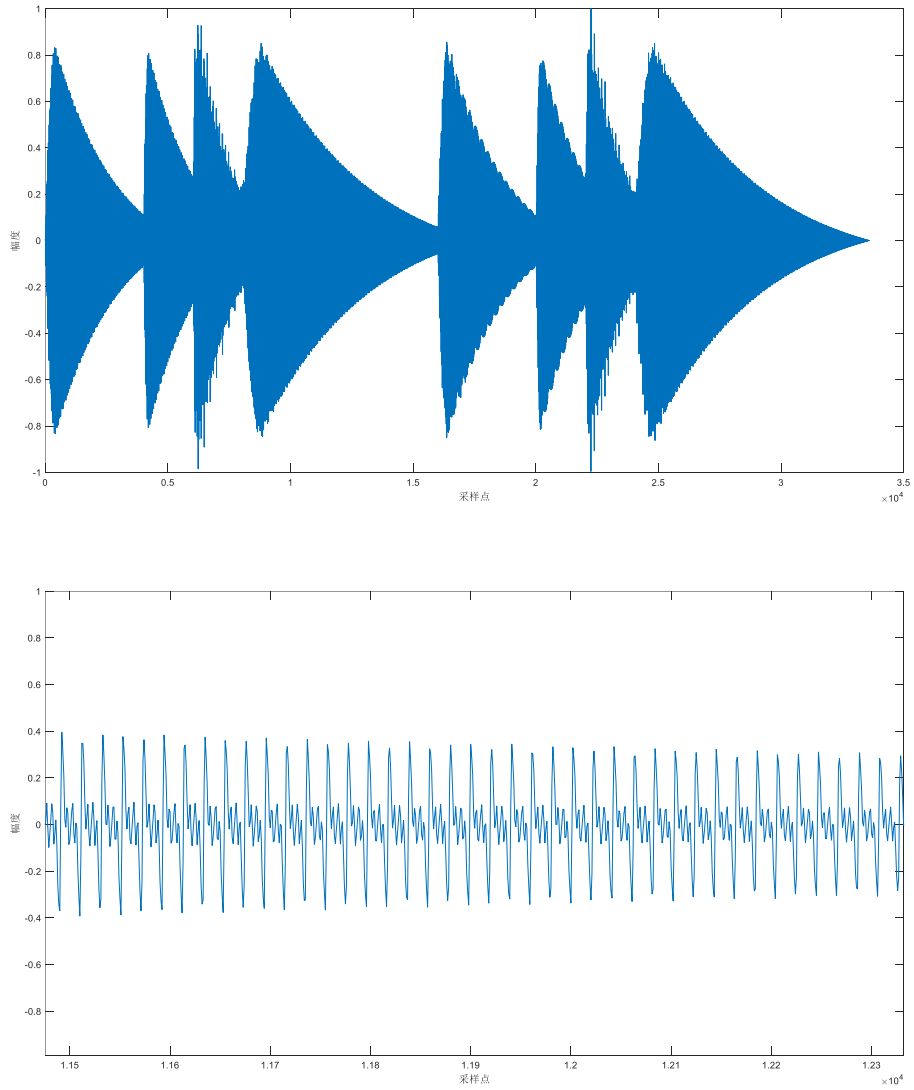


图 20 按照傅里叶级数合成的音乐波形

可以看出更多的谐波分量使得波形比（4）小题中复杂了许多，因而这段乐曲听起来与 fmt.wav 中的吉他更加相似，但仍有一定的区别。这是由于对于吉他来说，不同音符的各阶谐波分量的相对强度并不相同，因此（11）小题中将根据（9）中得到的不同音符谐波情况合成，应当能得到更加接近 fmt.wav 中的吉他的音色。

（11）用真实谐波信息合成《东方红》

【题目描述】

通过完成第（9）题，已经提取出 fmt.wav 中的很多音调，或者说，掌握了每个音调对应的傅里叶级数，大致了解了这把吉他的特征。据此演奏一曲《东方红》吧。

【主要思想】

根据（9）中结果，《东方红》前 4 小节音符及其谐波强度如下表（fmt.wav 中未出现的音符则用临近的音代替）

表 7 主要音符谐波强度

音名		C5	D5	D4	F4	G4
近似音		^{b}D5	^{b}D5	D4	F4	F4
相对强度	基波	1.0000	1.0000	1.0000	1.0000	1.0000
	2 次	0	0	0.7573	0.2148	0.2148
	3 次	0	0	0.2488	0	0
	4 次	0	0	0.1438	0.2260	0.2260
	5 次	0	0	0.1306	0	0
	6 次	0	0	0.1447	0	0
	7 次	0	0	0	0	0
	8 次	0	0	0	0	0
	9 次	0	0	0.1110	0	0
	10 次	0	0	0	0	0

利用这些不同音调的谐波分量数据，可以更加真实地实现吉他不同音调泛音不同的效果，合成出更接近真实吉他演奏的声音。

【核心代码】

```
seq=[3,3,5,-2,-4,-4,-7,-2];%曲谱序列
time=[0.5,0.25,0.25,1,0.5,0.25,0.25,1];%时值(s)
overlap=[0.2,0.3,0.2,0.2,0.2,0.3,0.2,0.2];%迭接时间(s)
harmonic=[1.0000,0.7272,0.5067,0,0,0,0,0,0,0;
    1.0000,0.7272,0.5067,0,0,0,0,0,0,0;
    1.0000,0.7573,0.2488,0.1438,0.1306,0.1447,0,0,0.1110,0;
    0.4201,1.0000,0.4718,0,0,0.0960,0.1324,0,0,0;
    1.0000,0.2729,0,0.1483,0,0.1732,0,0.1242,0,0;
    1.0000,0.2729,0,0.1483,0,0.1732,0,0.1242,0,0;
    0.3529,1.0000,0.3729,0.6826,0,0.1600,0,0,0,0;
    0.4201,1.0000,0.4718,0,0,0.0960,0.1324,0,0,0];%各音符谐波分量
f0=220;%基准频率
freq=8000;%采样频率
amp=1;%幅度
output=zeros(1,freq*(sum(time)+overlap(length(overlap))));%输出向量
position=1;%辅助指针
for k=1:length(time)
    t=linspace(0,time(k)+overlap(k)-1/freq,freq*(time(k)+overlap(k)));%生成时间
    temp=harmonic(k,1)*amp*sin(2*pi*f0*pow2(seq(k)/12)*t);%生成乐音
    temp=temp+harmonic(k,2)*amp*sin(2*pi*2*f0*pow2(seq(k)/12)*t);%高次谐波
    temp=temp+harmonic(k,3)*amp*sin(2*pi*3*f0*pow2(seq(k)/12)*t);
```

```
    temp=temp+harmonic(k,4)*amp*sin(2*pi*4*f0*pow2(seq(k)/12)*t);
    temp=temp+harmonic(k,5)*amp*sin(2*pi*5*f0*pow2(seq(k)/12)*t);
    temp=temp+harmonic(k,6)*amp*sin(2*pi*6*f0*pow2(seq(k)/12)*t);
    temp=temp+harmonic(k,7)*amp*sin(2*pi*7*f0*pow2(seq(k)/12)*t);
    temp=temp+harmonic(k,8)*amp*sin(2*pi*8*f0*pow2(seq(k)/12)*t);
    temp=temp+harmonic(k,9)*amp*sin(2*pi*9*f0*pow2(seq(k)/12)*t);
    temp=temp+harmonic(k,10)*amp*sin(2*pi*10*f0*pow2(seq(k)/12)*t);
    cover=[-100*(t(1:0.1*time(k)*freq-1)/time(k)).^2+20*(t(1:0.1*time(k)*freq-1)/time(k))...
        ,1.15652*exp(-2*(t(0.1*time(k)*freq:freq*(time(k)+overlap(k)))-
0.1*time(k))/(0.9*time(k)+overlap(k)))-0.15652];%生成包络
    output(position:position+freq*(time(k)+overlap(k))-
1)=output(position:position+freq*(time(k)+overlap(k))-1)+cover.*temp;%添加乐音
    position=position+freq*time(k);%移动指针
end
output=output/max(abs(output));%归一化
sound(output,freq);%播放
```

【核心代码说明】

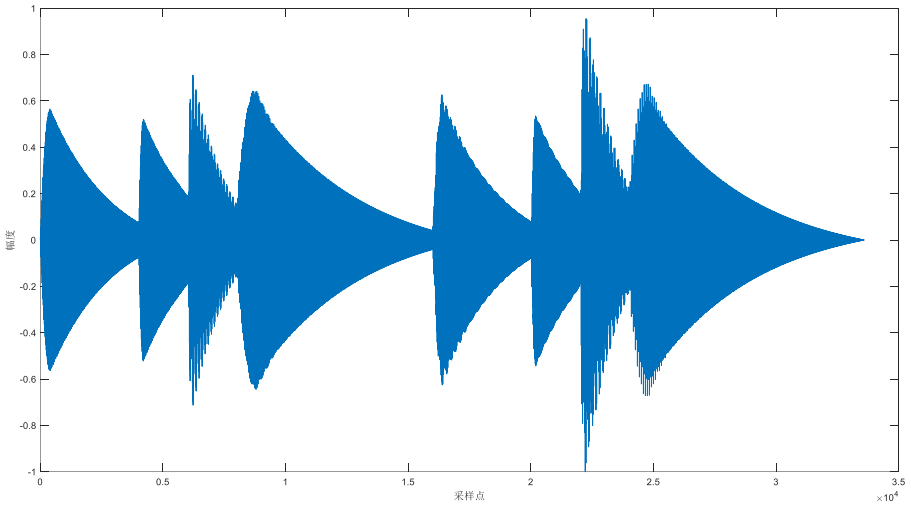
将《东方红》中每个音符的各次谐波强度一次写在 harmonic 数组中，首先利用向 temp 数组中根据 harmonic 数组中的谐波强度叠加各次谐波的正弦函数，叠加完成后 temp 数组与包络数组 cover 相乘得到最终的乐音，插入到 output 数组中得到最终乐曲。完整代码文件为 ex_1_2_3_11_a.m。

【运行结果】

得到音频文件“东方红 5.wav”

【结果分析】

结果保存为“东方红 5.wav”，这一段乐曲听起来与 fmt.wav 中吉他演奏的比较相似，其波形及局部放大为：



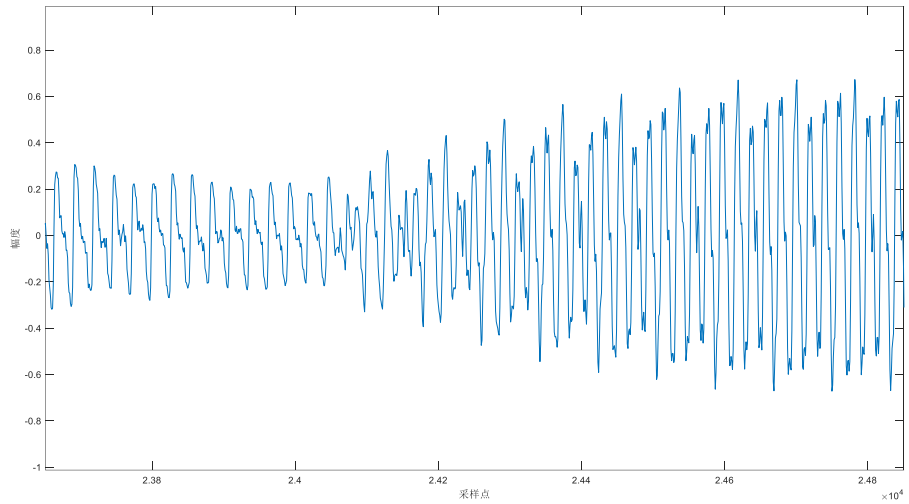


图 21 按照真实谐波合成的音乐波形

可以看出不同的音符有不同的谐波分量，因而波形并不相同，这种特性赋予了乐器丰富的音色。

由于 fmt.wav 中音调较低而《东方红》音调较高，较多的音符使用了附近音符的近似，如果将《东方红》降低一个八度，则可以更好地利用 fmt.wav 中得到的谐波数据，合成更加真实的音乐，完整代码文件为 ex_1_2_3_11_b.m，结果保存为“东方红 6.wav”。

（12）音乐合成 GUI

【题目描述】

现在只要你掌握了某乐器足够多的演奏资料，就可以合成出该乐器演奏的任何音乐，试着做一个图形界面把上述功能封装起来。

【主要思想】

设计一个钢琴键盘的界面，通过点击不同的琴键弹出不同的音符，模拟钢琴的演奏。同时设置输入框，可以设置各次谐波分量，从而实现奏出不同乐器的音色的功能。同时在界面中显示当前奏响的琴键音名，并设置文本框提示当前运行状态。

【核心代码 1】

```
t=linspace(0,overlap+0.5-1/8000,8000*(overlap+0.5));%生成时间采样
out=zeros(1,8000*(overlap+0.5));%输出序列
out=out+har(1)*sin(2*pi*220*pow2(-7/12)*t);%基波
out=out+har(2)*sin(2*pi*2*220*pow2(-7/12)*t);%二次谐波
out=out+har(3)*sin(2*pi*3*220*pow2(-7/12)*t);%三次谐波
out=out+har(4)*sin(2*pi*4*220*pow2(-7/12)*t);%四次谐波
out=out+har(5)*sin(2*pi*5*220*pow2(-7/12)*t);%五次谐波
cover=[-100*(t(1:0.1*0.5*8000-1)/0.5).^2+20*(t(1:0.1*0.5*8000-1)/0.5)...
        ,1.15652*exp(-2*(t(0.1*0.5*8000:8000*(0.5+overlap))-0.1*0.5)/(0.9*0.5+overlap))-0.15652];%生成包络
out=out.*cover;%叠加包络
handles.text6.String='D3';%显示音符
handles.text5.String=['D3','音符已加入序列'];
sound(out,8000);%播放
notes=[notes,-7];
```

```
pause(0.5+overlap)
handles.text6.String='';%取消显示音符
handles.text5.String='请继续演奏';
```

【核心代码说明 1】

代码实现单个琴键按下时发出对应的声音，并将音符插入序列中，更新界面相关提示。har 数组中存放设置的各次谐波分量强度，依次将各次谐波累加到 out 中并与包络数组 cover 对应元素相乘。out 数组计算完成后播放该音符，更新界面上的标签，并将该音符存入 notes 数组中以备最终整体播放

【核心代码 2】

```
f0=220;%基准频率
freq=8000;%采样频率
output=zeros(1,freq*(0.5*length(notes)+overlap));%输出向量
position=1;%辅助指针
for k=1:length(notes)
    t=linspace(0,0.5+overlap-1/freq,freq*(0.5+overlap));%生成时间
    temp=har(1)*sin(2*pi*f0*pow2(notes(k)/12)*t);%基波
    temp=temp+har(2)*sin(2*pi*2*f0*pow2(notes(k)/12)*t);%二次谐波
    temp=temp+har(3)*sin(2*pi*3*f0*pow2(notes(k)/12)*t);%三次谐波
    temp=temp+har(4)*sin(2*pi*4*f0*pow2(notes(k)/12)*t);%四次谐波
    temp=temp+har(5)*sin(2*pi*5*f0*pow2(notes(k)/12)*t);%五次谐波
    cover=[-100*(t(1:0.1*0.5*freq-1)/0.5).^2+20*(t(1:0.1*0.5*freq-1)/0.5)...
        ,1.15652*exp(-2*(t(0.1*0.5*freq:freq*(0.5+overlap))-0.1*0.5)/(0.9*0.5+overlap))-0.15652];% 生 成
包络
    output(position:position+freq*(0.5+overlap)-1)=output(position:position+freq*(0.5+overlap)-
1)+cover.*temp;%添加乐音
    position=position+freq*0.5;%移动指针
end
sound(output,freq);%播放
output=output/max(abs(output));
audiowrite('data/my_music.wav',output,freq)%保存
handles.text5.String='已保存到文件“my_music.wav”';
pause(0.5*length(notes)+overlap)
notes=[];
handles.text5.String='音符序列已清空，可以继续新的演奏';
```

【核心代码说明 2】

演奏完成后，上述代码将 notes 中的音符及其谐波分量依次插入 temp 数组，与包络数组 cover 对应元素相乘后存入 output 数组。最后播放并保存 output 数组，清空 notes 数组并更新界面各处提示。

【运行结果】

运行代码，界面如下：

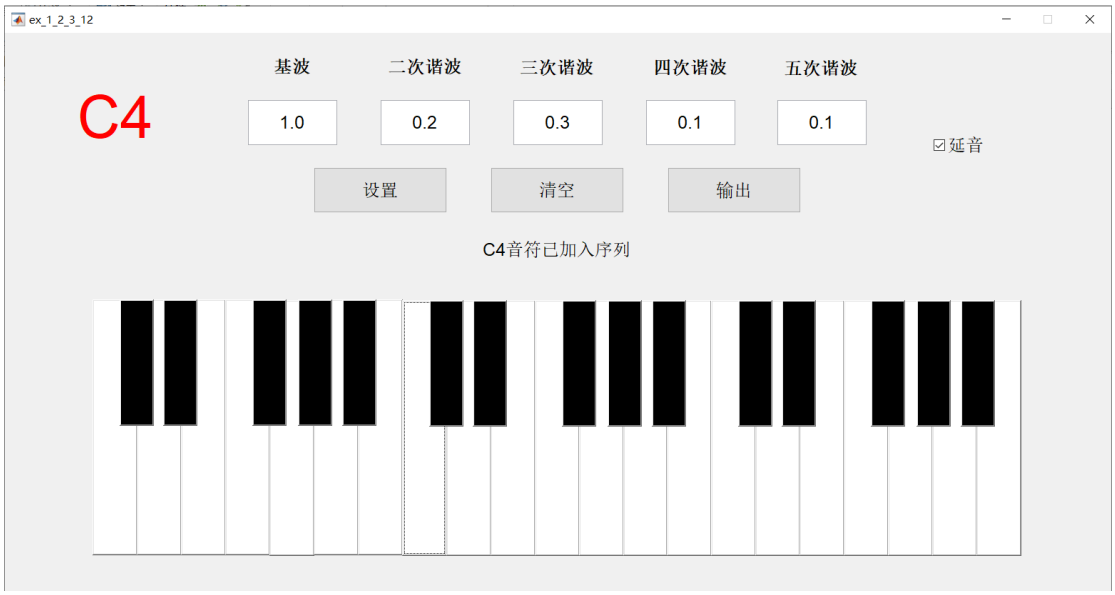


图 22 音乐合成 GUI

首先设置各阶谐波分量强度，可以勾选延音选项，设置好后点击“设置”按钮，文本框中提示“设置完成，可以演奏”。

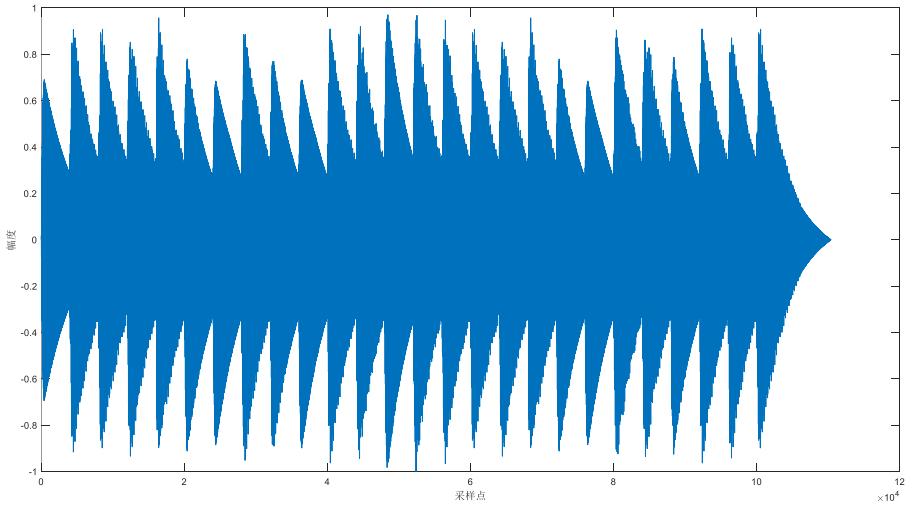
这时点击对应琴键（由于界面空间有限，只设置了 3 组琴键），会奏出相应的声音，同时左上角红色字体显示出所弹奏的音名；弹奏的同时将音符存入 notes 向量，同时文本框中提示“音符已经加入序列”。

弹奏中如果想要清除已有音符序列，点击“清空”按钮，notes 向量被置为空，文本框中提示“音符序列已清空”。如果弹奏完成，可以点击“输出”按钮，这时音符序列会整体连续播放一遍（这里所有音符均被设置为四分音符），同时文本框提示“已保存至文件 my_music.wav”。此处音符之间的迭接、音符的包络处理均使用 1.2.1（2）中的做法。

用该界面弹奏合成了《玛丽有只小羔羊》片段，结果保存为文件“my_music.wav”。

【结果分析】

生成的音乐波形和局部放大如图：



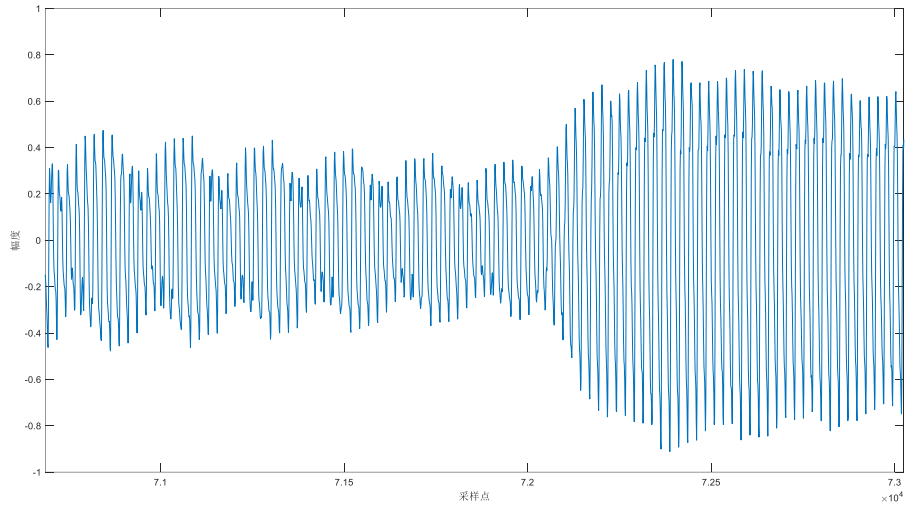


图 23 用 GUI 合成的音乐波形

根据得到的音乐和波形，可以确认设计的 GUI 音乐合成器能够合成具有较复杂的谐波的音乐片段。

实验总结

本实验通过 MATLAB 编程，进行了音乐的分析、处理与合成，实验内容丰富且颇具趣味。首先，通过基本乐理知识的学习和对于音乐片段的处理分析，我认识到音调由频率决定、不同音符的频率遵从一定数学关系；不同乐器多样的音色则由丰富的谐波分量决定，包络形状在音乐的听觉体验中也有重要作用。在此基础上，利用信号与系统的方法，处理分析了吉他的音乐片段，并合成了多种有不同特性的《东方红》乐曲，最后还设计了音乐合成 GUI。

在本实验中，我掌握了 MATLAB 编程的基本技术，同时复习了信号与系统的一些基本方法，在日后还应当继续学习编程中各类优化手段，以编写出高效而简洁的程序；同时，也要继续学习和运用时域、频域综合方法处理信号，解决各类工程问题。

附：文件清单

附表 1 文件清单

文件名	说明
实验报告.pdf	实验报告（本文档）
code\	
ex_1_2_1_1_a.m	1.2.1（1）播放单音
ex_1_2_1_1_b.m	1.2.1（1）初步合成《东方红》
ex_1_2_1_2.m	1.2.1（2）增加包络与音符迭接
ex_1_2_1_3.m	1.2.1（3）升高与降低音调
ex_1_2_1_4.m	1.2.1（4）增加高次谐波分量
ex_1_2_1_5_a.m	1.2.1（5）分析钢琴声音谐波
ex_1_2_1_5_b.m	1.2.1（5）合成《克罗地亚狂想曲》
ex_1_2_1_6.m	1.2.2（6）分析 fmt.wav 文件中的音乐
ex_1_2_1_7.m	1.2.2（7）预处理 realwave
ex_1_2_1_8_a.m	1.2.2（8）分析基音与频率——傅里叶级数方法
ex_1_2_1_8_b.m	1.2.2（8）分析基音与频率——傅里叶变换方法

ex_1_2_1_8_c.m	1.2.2（8）分析基音与频率——快速傅里叶变换方法
ex_1_2_1_9.m	1.2.2（9）自动提取乐谱
ex_1_2_1_10.m	1.2.3（10）用傅里叶级数合成《东方红》
ex_1_2_1_11_a.m	1.2.3（11）用真实谐波合成《东方红》
ex_1_2_1_11_b.m	1.2.3（11）用真实谐波合成降八度的《东方红》
ex_1_2_1_12.m	1.2.3（12）音乐合成 GUI 代码
ex_1_2_1_12.fig	1.2.3（12）音乐合成 GUI 图窗
locatepeak.m	函数：定位峰值
prefourier.m	函数：计算傅里叶变换矩阵
code\data\	
东方红 1.wav	1.2.1（1）初步合成东方红结果
东方红 2.wav	1.2.1（2）增加包络与音符迭接结果
东方红_升八度.wav	1.2.1（3）升高与降低音调结果
东方红_降八度.wav	
东方红_升半音.wav	
东方红_降半音.wav	
东方红 3.wav	1.2.1（4）增加高次谐波分量结果
钢琴.wav	1.2.1（5）中用于分析钢琴声音谐波的钢琴声
克罗地亚狂想曲.wav	1.2.1（5）合成《克罗地亚狂想曲》结果
note.mat	1.2.2（9）自动提取乐谱，记录乐谱音符
time_last.mat	1.2.2（9）自动提取乐谱，记录乐谱节奏
东方红 4.wav	1.2.3（10）用傅里叶级数合成《东方红》结果
东方红 5.wav	1.2.3（11）用真实谐波合成《东方红》结果
东方红 6.wav	1.2.3（11）用真实谐波合成降八度的《东方红》结果
my_music.wav	1.2.3（12）用音乐合成 GUI 合成音乐结果
fmt.wav	待分析的吉他音乐片段
Guitar.mat	存储 realwave 和 wave2proc