一、 实验目的

1. 熟悉 MATLAB 的软件和语言指令的使用；
2. 学习利用 MATLAB 进行连续信号的时域、频域分析；
3. 通过电子音乐合成方面的练习增进对傅里叶级数的理解。

二、 实验内容

1. 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。请用 sound 函数播放每个乐音，听一听音调是否正确。最后用这一系列乐音信号拼出《东方红》片断，注意控制每个乐音持续的时间要符合节拍，用 sound 播放你合成的音乐，听起来感觉如何？

俗话说万事开头难，由于这一题是我自己编写的第一个 MATLAB 程序，之前都是看老师的演示程序，对 MATLAB 指令并不熟练，所以这样一个很简单程序我也花了不少功夫。编写过程中，我主要还是对照着课本例题，一步一步，从生成抽样时间到编辑音调。最后在 help 的帮助下，参看了 help 中的 Hallelujah 范例，学会了 sound 函数的用法。
代码如下：

```
clear all,close all,clc;
t=linspace(0,4-1/8000,4*8000)';          %4小节每小节1s,以4分音符为1拍，每小节2拍
y=0*t;
y(t<0.5)=sin(392*2*pi*t(t<0.5));        %第一个音 '5'，下依次
y(0.5<t&t<0.75)=sin(392*2*pi*t(0.5<t&t<0.75));
y(0.75<=t&t<1)=sin(440*2*pi*t(0.75<=t&t<1));
y(1<t&t<1.5)=sin(293.66*2*pi*t(1<t&t<1.5));
y(2<=t&t<2.5)=sin(261.63*2*pi*t(2<=t&t<2.5));
y(2.5<t&t<2.75)=sin(261.63*2*pi*t(2.5<t&t<2.75));
y(2.75<=t&t<3)=sin(220*2*pi*t(2.75<=t&t<3));
y(3<t&t<3.5)=sin(293.66*2*pi*t(3<t&t<3.5));
sound(y,8000)                          %播放音乐
```

运行程序可以正确播放东方红前4小节，并且符合节拍。不过，确实存在第二题中所说的‘啪’声，并且两个同样的音之间没有分开。

在之后的编程过程中我发现，这第一题我的方法其实很笨，而且代码利用率不高。于是在后边题中我对程序进行了改进，具体的改进方法我将在后边题中说明。

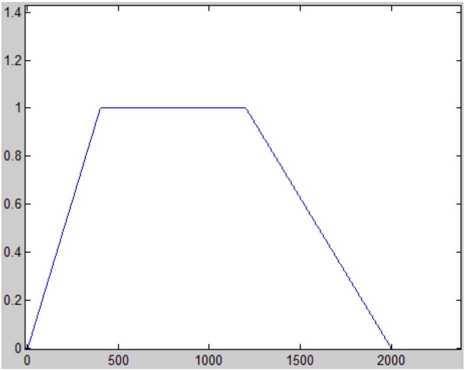
2. 你一定注意到(1)的乐曲中相邻乐音之间有“啪”的杂声，这是由于相位不连续产生了高频分量。这种噪声严重影响合成音乐的质量，丧失真实感。为了消除它，我们可以用包络修正每个乐音，以保证在乐音的邻接处信号幅度为零。此外建议用指数衰减的包络来表示 1。

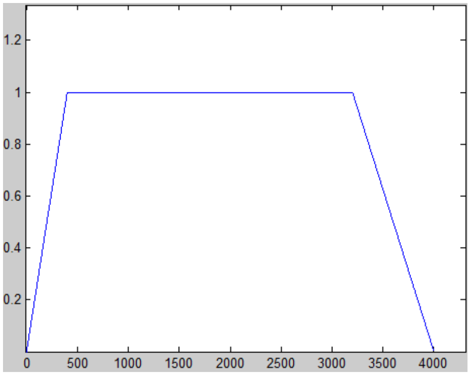
消除‘啪’声的原理是让乐音的邻接处信号幅度为零，因此我在每个音波形开始和结束时与一个从 0 到 1 递增或从 1 到 0 递减的序列相点乘，实现渐变，保证了乐音邻接处信号幅度为 0。

```
clear all, close all, clc;            %优化代码后的第二问
t1=linspace(0,0.25-1/8000,2000)’;   %修正8分音符包络
set1=ones(2000,1);
set1(t1<0.05)=20*t1(t1<0.05);
set1(t1>0.15)=10*(0.25-t1(t1>0.15));
t2=linspace(0,0.5-1/8000,4000)’;    %修正4分音符包络
set2=ones(4000,1);
set2(t2<0.05)=20*t1(t2<0.05);
set2(t2>0.4)=10*(0.5-t2(t2>0.4));
y0=zeros(4000,1);
y1=set2.*note4(392);    %G1
y2=set1.*note8(392);    %G1
y3=set1.*note8(440);    %A1
y4=set2.*note4(293.66);    %D1
y5=set2.*note4(261.63);    %C1
y6=set1.*note8(261.63);    %C1
y7=set1.*note8(220);    %A0
y8=set2.*note4(293.66);    %D1
y=[y1;y2;y3;y4;y0;y5;y6;y7;y8;y0];
sound(y,8000)

function note=note4(x)    %4分音符函数 x为频率
t=linspace(0,0.5-1/4000,4000)’;
note=sin(x*2*pi*t);

function note=note8(x)          %8分音符函数 x为频率
t=linspace(0,0.25-1/2000,2000)’;
note=sin(x*2*pi*t);
```

从代码可以看出，我在完成此题程序的编写时，对题（1）中的代码进行了优化。我先构造两个函数 note4 和 note8，根据各自的频率参数分别产生对应音调的 4 分音符和 8 分音符波形序列，然后再对各自的包络修正。最后将所有音的串起来，即得到了最终音乐。

下为包络修正序列：





3. 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。（提示：音乐播放的时间可以变化）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。（提示：视计算复杂度，不必特别精确）

此题在（2）的基础上添加一些代码即可实现，相同部分我就不重复附上，只说明后边附加的部分。
开始时我采用的是将原波形序列点数增加或减少一倍来实现的。后发现，其实只需要调整 sound 函数的采样频率即可间接实现将频率翻倍或减半的效果。

```
z1=ones(16000,1);        %升高一个八度
for m=1:16000
    z1(m)=y(2*m);
end

sound(y,8000*2);         %通过提高采样频率升高一个八度
sound(y,8000/2);         %通过降低采样频率降低一个八度
```

利用resample函数可以很方便地变更原波形的频率。它的原理是通过插值P和抽取Q，以Q/P为倍数改变

```
w=resample(y,10000,10595);    %p/q=1/1.0595 即升半音
sound(w,8000);
```

4. 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？注意谐波分量的能量要小，否则掩盖住基音反而听不清音调了。（如果选择基波幅度为 1 ， 二次谐波幅度 0:2 ， 三次谐波幅度 0:3 ，听起来像不像象风琴？）

题（2）代码不用修改，直接在 note4 和 note8 两个频率波形产生函数中加入谐波分量即可，函数名为 note4a 和 note8a。代码如下

```
function note=note4a(x)    %加谐波后的4分音符函数，  x为频率
t=linspace(0,0.5-1/4000,4000)';
note=1*sin(x*2*pi*t)+0.2*sin(2*x*2*pi*t)+0.3*sin(3*x*2*pi*t);
```

```
function note=note8a(x)    %加谐波后的8分音符函数，  x为频率
t=linspace(0,0.25-1/2000,2000)';
note=1*sin(x*2*pi*t)+0.2*sin(2*x*2*pi*t)+0.3*sin(3*x*2*pi*t);
```

加入谐波分量后声音明显感觉有立体感了。

5. 自选其它音乐合成，例如贝多芬第五交响乐的开头两小节

我选的是国歌的前奏部分，这部分既有 8 分音符，也有 4 分音符，适合做测试合成的音乐。代码如下：

```
clear all, close all, clc;           %国歌前奏部分
t1=linspace(0,0.25-1/8000,2000)';   %修正8分音符
set1=ones(2000,1);
set1(t1<0.05)=20*t1(t1<0.05);
set1(t1>0.15)=10*(0.25-t1(t1>0.15));
t2=linspace(0,0.5-1/8000,4000)';    %修正4分音符
set2=ones(4000,1);
set2(t2<0.05)=20*t1(t2<0.05);
set2(t2>0.4)=10*(0.5-t2(t2>0.4));
y0=zeros(4000,1);
y1=set1.*note8(261.63);    %C1
y2=set1.*note8(329.63);    %E1
y3=set1.*note8(392);    %G1
y4=set2.*note4(440);    %A1
y5=set2.*note4(392);    %G1
y6=set2.*note4(329.63);    %E1
y7=set2.*note4(261.63);    %C1
y8=set1.*note8(392);    %G1
y9=set1.*note8(196);    %G0
y=[y1;y2;y3;y3;y4;y5;y2;y1;y8;y8;y6;y7;y9;y9;y9;y9;y7;y0];
sound(y,8000);
```

这一题比较简单，完成得比较顺利。

6. 先用 wavread 函数载入光盘中的 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？

在阅读了 help wavread 后，很轻松地实现了载入并播放。
代码如下：

```
[Y,FS]=wavread(’fmt.wav’);
sound(Y,FS);
```

与真实的吉他音乐相比，之前所做的合成音乐很不真实，因为它缺乏包络的变化(即真实乐器的振幅特点)以及频率丰富的和弦、揉弦等，而这些是真实的乐器很容易做到的。

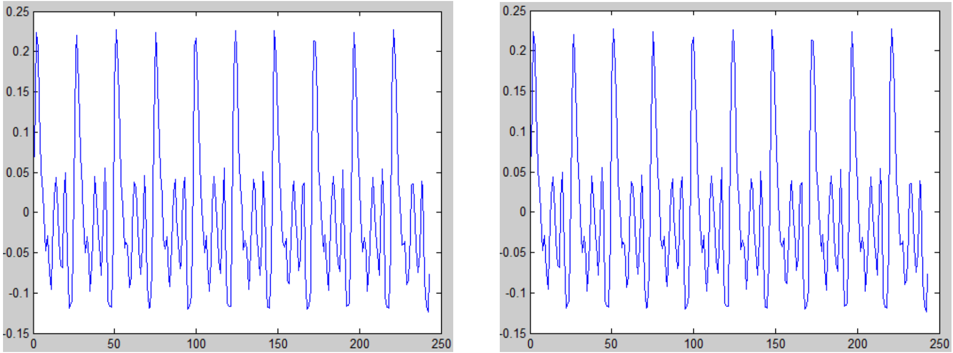
7. 你知道待处理的 wave2proc 是如何从真实值 realwave 中得到的么？这个预处理过程可以去除真实乐曲中的非线性谐波和噪声，对于正确分析音调是非常重要的。提示：从时域做，可以继续使用 resample 函数。

这一题可以说是我在做大作业中遇到的第一个坎。开始我没想明白提示中所说的 resample 函数在此题中如何使用。提示说从时域做，我的第一反应是用自相关，但后来想到这样只能检出波形而不能滤掉非线性谐波和噪声。
在仔细观察了 realwave 和 wave2proc 的波形后，我发现其处理后恰好是 10 个周期，波形近乎相同。于是我想到如果对于每个周期的波形来说，非线性谐波和噪声都是随机的。如此一来，如果将很多周期的波形叠加，最后再除以周期数，那么所得的取“平均”后的周期波形，非线性谐波和噪声的影响因抵消而减小，而有价值的波形本身不受影响。
这样，resample 函数就派上了用场，可以利用它将 realwave 抽样点变为 10 倍，即 10 个周期，每个周期的抽样点数量都为整数 243。如此只要将 realwave 每次向左错开 1 个周期然后相加，一共进行 10 次，即可得到取“平均”后的波形。再利用 resample 函数将抽样频率还原即可。代码如下：

```
clear all, close all, clc;
load(’guitar.mat’);
y=resample(realwave,10,1);      %用10倍采样频率采样
z=y;
x=zeros(2430,1);
for m=1:10                                %一共10个周期
    z=[z(244:2430);z(1:243)];     %左移一个周期
    x=x+z;                        %累加
```

```
end
x=x/10;              %求平均
y=resample(x,1,10);          %还原回采样频率
```

运行后发现所得波形 y 与 wave2proc 一模一样，分毫不差，当时成就感油然而生。可能我采用的方法与老师的方法相同，才能得到如此一致的结果。y 和 wave2proc 的波形如下：

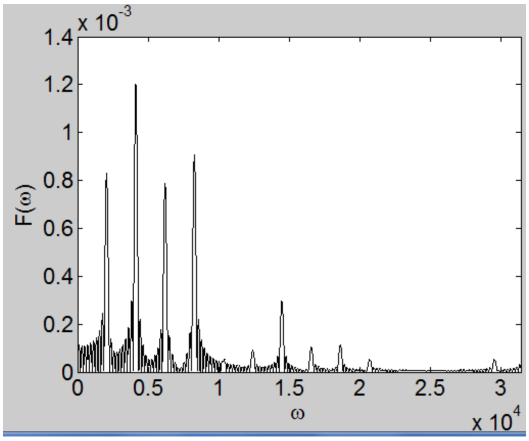


8. 这段音乐的基频是多少？是哪个音调？请用傅里叶级数或者变换的方法分析它的谐波分量分别是什么。提示：简单的方法是近似取出一个周期求傅里叶级数但这样明显不准确，因为你应该已经发现基音周期不是整数（这里不允许使用 resample 函数）。复杂些的方法是对整个信号求傅里叶变换（回忆周期性信号的傅里叶变换），但你可能发现无论你如何提高频域的分辨率，也得不到精确的包络（应该近似于冲激函数而不是 sinc 函数），可选的方法是增加时域的数据量，即再把时域信号重复若干次，看看这样是否效果好多了？请解释之。

首先在不增加时域数据量的情况下，采用教材的矩阵计算法，对整个信号求傅里叶变换，代码如下：

```
clear all, close all, clc;    %未增加时域数据量
load('guitar.mat');

N = 243;
t = linspace(0,(N-1)/8000,N);
f = wave2proc;
OMG = 10000*pi;
K = 20000;
omg = linspace(-OMG,OMG-OMG/K,K)';
U = exp(-j*kron(omg,t));
F = (N-1)/8000/N*U*f;              %矩阵计算法求傅里叶变换
F = abs(F);               %取结果取模
```

```
figure;
hold on, box on;
plot(omg,F,'k-');
set(gca,'XLim',[0,10000*pi],'FontSize',16);    %画出正半轴
xlabel('\omega');
ylabel('F(\omega)');
```



此时得到变换后的图像为右图，确实得不到冲激函数。

因此需要增加时域的数据量，即再把时域信号重复若干次。因为时域周期延拓，频域采样。因此时域信号重复若干次后，频域信号接近采样结果，即冲激函数。

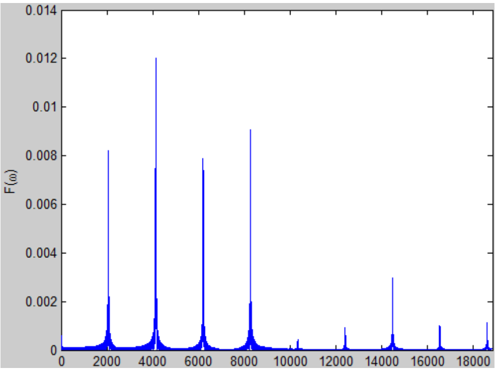
增加时域数据量后代码如下：

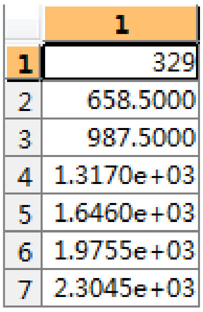
```
clear all, close all, clc;
load('guitar.mat');

N=243*10;                          %重复10次
t=linspace(0,(N-1)/8000,N);
f=[wave2proc;wave2proc;wave2proc;wave2proc;wave2proc;
    wave2proc;wave2proc;wave2proc;wave2proc;wave2proc];
OMG=6000*pi;
K=6000;
omg=linspace(0,OMG-OMG/K,K)';
U=exp(-j*kron(omg,t));
F=(N-1)/8000/N*U*f;
F=abs(F);      %取绝对值

[X(1),Y(1)]=maxp(0,2500,omg,F);     %基波的频率和幅度
[X(2),Y(2)]=maxp(2500,5000,omg,F);     %二次谐波的频率和幅度，
下为三至七次谐波的频率和幅度
[X(3),Y(3)]=maxp(5000,7500,omg,F);
[X(4),Y(4)]=maxp(7500,10000,omg,F);
[X(5),Y(5)]=maxp(10000,12000,omg,F);
[X(6),Y(6)]=maxp(12000,14000,omg,F);
[X(7),Y(7)]=maxp(14000,16000,omg,F);
X=X';
Y=Y';
```

```
figure;
hold on, box on;
plot(omg,F);
set(gca,’XLim’,[0,6000*pi]);
xlabel(’\omega’);
ylabel(’F(\omega)’);

function [X,Y]=maxp(L,R,omg,F)    %返回[L,R]区间内，F取最大值时的omg和F
[Y,x]=max(F(omg>L&omg<R));
z=omg(omg>L&omg<R);
X=z(x)/2/pi;     %将omg角频率换算为频率
```

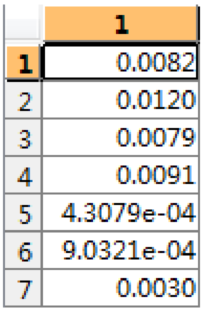
maxp函数是基于max函数的自定义函数，它的功能是根据区间的设置，返回区间内y取最大值时的x和y。并将x由角频率转化为频率。

此时得到变换后的图像和计算结果为下图，
相比之前已经有了明显改善。





	1
1	329
2	658.5000
3	987.5000
4	1.3170e+03
5	1.6460e+03
6	1.9755e+03
7	2.3045e+03



	1
1	0.0082
2	0.0120
3	0.0079
4	0.0091
5	4.3079e-04
6	9.0321e-04
7	0.0030

通过结果可知，基频为329Hz，由此可知该音音调为E1。基波幅度为0.0082，二至七次谐波幅度分别为0.0120、0.0079、0.0091、0.0004、0.0009、0.0030。

完成此题时，我在求冲激点和幅值时遇到了一点小麻烦。
首先是函数选取上。开始我想用求极值的函数，直接求出几个谐波的幅值。但后来发现傅里叶变换后并不是单调增的，而是有振荡的，所有很多值很小的点也被求了出来，于是这条路走不通了。后来就换了取区间内的最大值的方法，区间人为给定。
第二个问题是max函数的使用上。由于我是在max函数的参数上直接给定范围的，因此返回值x是针对该范围的索引值。而我却误以为返回值x直接就是整个序列的索引值。因此算出来的结果总是不对，查了很久，最后我发现某个点的x值很小，这才发现了问题所在。

9. 再次载入 fmt.wav ，现在要求你写一段程序，自动分析出这段乐曲的音调和节拍！如果你觉得太难就允许手工标定出每个音调的起止时间，再不行你就把每个音调的数据都单独保存成一个文件，然后让 MATLAB 对这些文件进行批处理。注意：不允许逐一地手工分析音调。

这一题当之无愧可说是所有题里最复杂最难的花时间最长的一道。一开始我本打算利用每个音振幅的突然增加来判断音调的起止时间，但这样做有一个问题就是，，每个点的值是振荡的，并非单调的，无法直接判断。当时没有想到其他的方法，因此就先手工标定每个音调的起止时间，并用 onenote 函数将该音调从原音乐中提取出来。由于每段序列长度不同，所以无法存为 y(:,m)的形式，代码如下：

```
clear all,close all,clc
Y=wavread('fmt.wav');
y1=onenote(Y,2336,10400);
y2=onenote(Y,14272,18000);
y3=onenote(Y,18000,21712);
y4=onenote(Y,21712,25320);
y5=onenote(Y,25320,29024);
y6=onenote(Y,29024,32640);
y7=onenote(Y,32640,36168);
y8=onenote(Y,36168,38064);
y9=onenote(Y,38064,40320);
y10=onenote(Y,40320,46400);
y11=onenote(Y,46400,56240);
y12=onenote(Y,56240,62400);
y13=onenote(Y,62400,68000);
y14=onenote(Y,68000,71760);
y15=onenote(Y,71760,75784);
y16=onenote(Y,75784,78968);
y17=onenote(Y,78968,81056);
y18=onenote(Y,81056,82856);
y19=onenote(Y,82856,84488);
y20=onenote(Y,84488,86640);
y21=onenote(Y,86640,90400);
y22=onenote(Y,90400,93984);
y23=onenote(Y,93984,98120);
y24=onenote(Y,98120,101904);
y25=onenote(Y,101904,106280);
y26=onenote(Y,106280,110160);
y27=onenote(Y,110160,119744);
y28=onenote(Y,119744,131072);
```

```
for m=1:28        %傅里叶分析
eval(['F',num2str(m),'=fouriernote(y',num2str(m),');']);
end
```

这里用到了 eval 和 num2str 函数。因为我的乐音命名是直接字母+数字（比如 y1），因此普通的循环语句不能工作。后来在网上查找资料，找到了解决办法，即先用 num2str 函数将 m 变为字符，然后用 eval 函数对整句进行运算，即可实现循环。

下为提取函数 onenote

```
function note=onenote(s,L,R) %从s中提取L、R两点间的一段数据
note=s(L:R);
```

下为傅里叶变换函数 foriernote，计算并绘图

```
function F=fouriernote(y)
N=length(y);
t=linspace(0,(N-1)/8000,N);
f=y;
OMG=4000*pi;
K=4000;
omg=linspace(0,OMG-OMG/K,K)';
U=exp(-j*kron(omg,t));
F=(N-1)/8000/N*U*f;
F=abs(F);      %取绝对值

figure;
hold on, box on;
plot(omg,real(F));
set(gca,'XLim',[0,4000*pi]);
xlabel('\omega');
ylabel('F(\omega)');
line([174.61*2*pi 174.61*2*pi], [0,max(F)], 'Color',
'red');  %在最低音和最高音处划线
line([659.25*2*pi 659.25*2*pi], [0,max(F)], 'Color', 'red');
```

以上部分可以正确的分离出每个音，求出傅里叶变换并绘图。
为了能够实现自动分析出音调和节拍，我和室友讨论了一下，室友说可以参考网上一个语音信号端点检测的方法，用短时平均能量来判断乐音的始末。于是我在网上搜索到了这个方法的代码。它是结合短时能量和短时过零率来检测语音信号端点的方法。
其代码如下:

```
function [x1,x2] = vad(x)

%幅度归一化到[-1,1]
x = double(x);
x = x / max(abs(x));

%常数设置
FrameLen = 240;
FrameInc = 80;

amp1 = 10;
amp2 = 2;
zcr1 = 10;
zcr2 = 5;

maxsilence = 8;  % 6*10ms  = 30ms
minlen  = 15;      % 15*10ms = 150ms
status  = 0;
count    = 0;
silence = 0;

%计算过零率
tmp1  = enframe(x(1:end-1), FrameLen, FrameInc);
tmp2  = enframe(x(2:end)  , FrameLen, FrameInc);
signs = (tmp1.*tmp2)<0;
diffs = (tmp1 -tmp2)>0.02;
zcr     = sum(signs.*diffs, 2);

%计算短时能量
amp = sum(abs(enframe(filter([1 -0.9375], 1, x), FrameLen, FrameInc)), 2);

%调整能量门限
amp1 = min(amp1, max(amp)/4);
amp2 = min(amp2, max(amp)/8);

%开始端点检测
x1 = 0;
x2 = 0;
for n=1:length(zcr)
    goto = 0;
    switch status
    case {0,1}                          % 0 =  静音, 1 =  可能开始
        if amp(n) > amp1              %  确信进入语音段
            x1 = max(n-count-1,1);
```

```
         status  = 2;
         silence = 0;
         count    = count + 1;
      elseif amp(n) > amp2 | ... %  可能处于语音段
              zcr(n) > zcr2
         status = 1;
         count  = count + 1;
      else                        %  静音状态
         status  = 0;
         count    = 0;
      end
   case 2,                        % 2 =  语音段
      if amp(n) > amp2 | ...      %  保持在语音段
         zcr(n) > zcr2
         count = count + 1;
      else                        %  语音将结束
         silence = silence+1;
         if silence < maxsilence %  静音还不够长，尚未结束
            count  = count + 1;
         elseif count < minlen    %  语音长度太短，认为是噪声
            status  = 0;
            silence = 0;
            count    = 0;
         else                     %  语音结束
            status  = 3;
         end
      end
   case 3,
      break;
   end
end

count = count-silence/2;
x2 = x1 + count -1;

subplot(311)
plot(x)
axis([1 length(x) -1 1])
ylabel('Speech');
line([x1*FrameInc x1*FrameInc], [-1 1], 'Color', 'red');
line([x2*FrameInc x2*FrameInc], [-1 1], 'Color', 'red');

subplot(312)
plot(amp);
```

```
axis([1 length(amp) 0 max(amp)])
ylabel('Energy');
line([x1 x1], [min(amp),max(amp)], 'Color', 'red');
line([x2 x2], [min(amp),max(amp)], 'Color', 'red');

subplot(313)
plot(zcr);
axis([1 length(zcr) 0 max(zcr)])
ylabel('ZCR');
line([x1 x1], [min(zcr),max(zcr)], 'Color', 'red');
line([x2 x2], [min(zcr),max(zcr)], 'Color', 'red');
```

代码中红色的部分是我参考或用到的。最有用处的是这一句

```
amp = sum(abs(enframe(filter([1 -0.9375], 1, x), FrameLen, FrameInc)), 2);
```

仔细研究后我发现，他的原理是先将 x 滤波，减小噪声等的影响，然后通过 enframe 函数将一段 FrameLen 长的信号作为一个整体，每次所取所取信号右移 FrameInc。因而可以反映一段信号的整体的变化，即可以将振荡的幅度大小总的变化规律体现了出来，因而可以得到近乎单调的短时平均能量。接下来就可以利用它来判断音调的始末了。

在跑过一次这段代码后，得到了短时平均能量和短时过零率的图像。我没有用网上这段代码的判断端点的双门限算法，原因有几点。首先，通过对比短时平均能量的图像和原音乐波形发现，该图像已经能够很好地将每个音分开，而短时过零率波形却在很多地方与音乐分音不一致，我猜想可能是由于原来这段代码是用来处理语音而不是音乐的。其次，这段代码的门限设置对于吉他拨弦力度不同的乐音判断效果不好，乐音幅度的大小并不一致，因此以一个确定的门限来判断显然是不合适的。

因此我采用了自己的判断方法，代码如下：

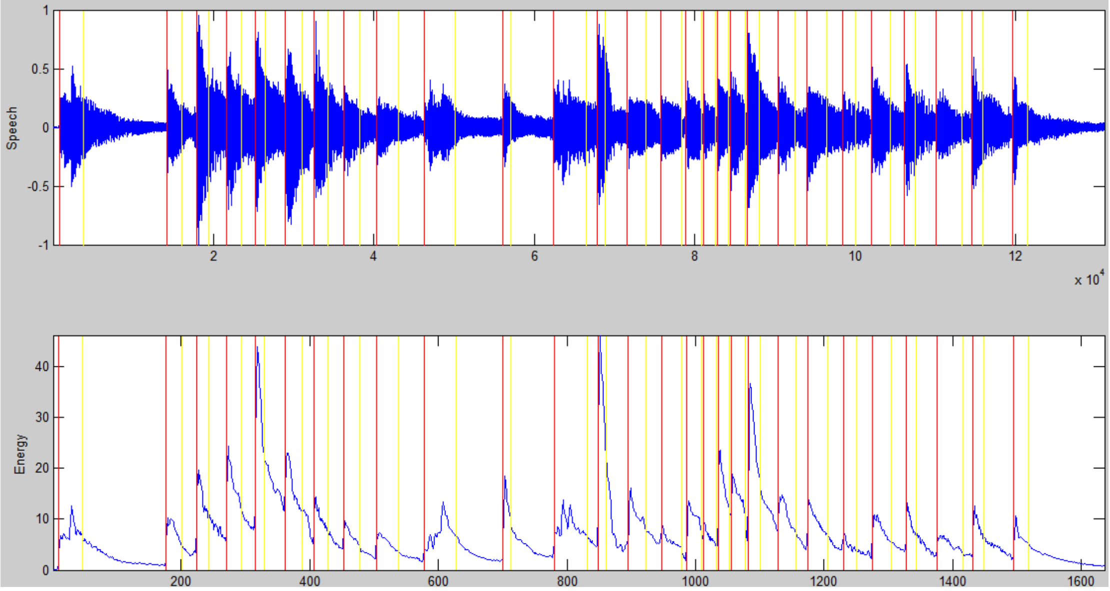
```
function [head,tail,beat] = vad(x)    %自动将音乐分段 并分析每个音拍数

%幅度归一化到[-1,1]
x = double(x);
x = x / max(abs(x));
%常数设置
FrameLen = 240;
FrameInc = 80;
status  = 0;              %初始状态为非乐音状态

%计算短时能量
amp = sum(abs(enframe(filter([1 -0.9375], 1, x), FrameLen,
FrameInc)), 2);

%开始端点检测
p = 1;           %标记乐音序号
ampmin=5;           %初始门限
ampmax=0;          %记录该音最大幅值
for n=1:length(amp)
   switch status
   case 0,                  % 0=非乐音状态
      if amp(n) > (ampmin+2)          %大于最低值
         if ampmax < amp(n)            %更新ampmax
             ampmax = amp(n);
         end
         head(p) = n;         %记录始端位置
         status  = 1;         %进入乐音状态
      else
         status  = 0;          %保持非乐音状态
         if ampmin > amp(n)         %更新ampmin
             ampmin = amp(n);
         end
      end
   case 1,                       % 1=乐音状态
       if ampmax < amp(n)       %更新ampmax并判断下一个点
           ampmax = amp(n);
       end
      if amp(n) < ampmax/2         %幅度小于最大值一半判定结束
          tail(p) = n;        %记录末端位置
          p=p+1;              %序号+1
          status=0;           %进入非乐音状态
          ampmax=0;           %重置ampmax
          ampmin=amp(n);
      end
   end
end

subplot(211)
plot(x)              %音乐图像
axis([1 length(x) -1 1])
ylabel('Speech');
for n=1:length(head)      %红线为始端，黄线为末端
    line([head(n)*FrameInc head(n)*FrameInc], [-1 1], 'Color',
'red');
    line([tail(n)*FrameInc tail(n)*FrameInc], [-1 1], 'Color',
'yellow');
```

```
end
subplot(212)
plot(amp);          %短时平均能量图像
axis([1 length(amp) 0 max(amp)])
ylabel(’Energy’);
for n=1:length(head)      %红线为始端，黄线为末端
    line([head(n) head(n)], [min(amp),max(amp)], ’Color’,
’red’);
    line([tail(n) tail(n)], [min(amp),max(amp)], ’Color’,
’yellow’);
end
head=(head*80)’;
tail=(tail*80)’;

t=zeros(length(head));         %节拍判断
for m=1:length(head)-1
    t(m)=head(m+1)-head(m);       %相邻节拍的始端间距
    if t(m)<=2800
        beat(m)=0.5;            %半拍
    elseif t(m)>2800&&t(m)<=5600
        beat(m)=1;              %一拍
    elseif t(m)>5600&&t(m)<=11000;
        beat(m)=2;              %两拍
    elseif t(m)>11000&&t(m)<=16000;
        beat(m)=3;              %三拍
    elseif t(m)>=16000 beat(m)=4;            %四拍
    end
end
beat(m+1)=0;            %最末一拍无法判断，标0
beat=beat’;
```

我的方法是，设置初始门限 ampmin，对短时平均能量图像逐个点判断，第一个过门限的点为第一个音始端，记录该音最大值 ampmax。当幅度降到 ampmax 的一半时，判定该音结束，ampmax 置零。将前一个音的结束值作为暂时的最低值（新门限），以非乐音状态中继续记录最低值 ampmin。当遇到幅值比最低值 ampmin 大 2 的点时，判定第二个音开始。如此继续下去，即可得到所有音的始末端。

此方法优点是简洁易懂，容易通过代码实现，而且准确度高。缺点是，由于是基于能量幅值的判断，因此对于和弦和揉弦这种幅值不变但频率不同的音无法检测出来。

得到的分音的图像如下（红线为始端，黄线为末端）：



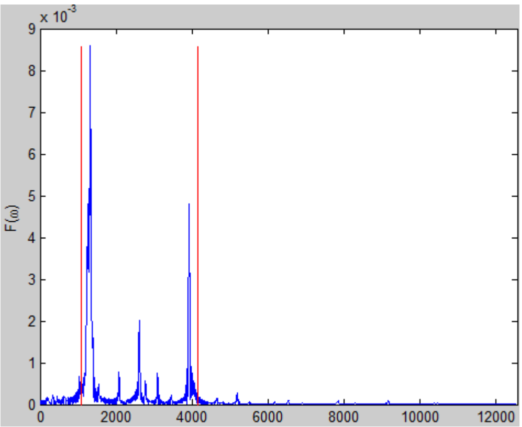
由于记录的始、末端并不是每一拍的始末，因此节拍的判断是通过判定前后两个音的始端间的距离来实现的。对照代码可以发现，不同拍数的时间区间并不是简单的正比关系，并且误差界定范围设置的比较大。这是因为该音乐的演奏速度并不是从始至终一致的，某些音弹奏的舒缓因而稍长，和弦音会比正常音稍早开始等。因此我针对这些情况做了调整处理。

在编写了自动分音的函数后，我重新编写了脚本文件，代码如下：

```
clear all,close all,clc     %优化代码后
Y=wavread('fmt.wav');
[head,tail,beat]=vad(Y);          %返回每个音的开始和结束及拍
数
for m=1:length(head)          %提取每个音
    eval(['y',num2str(m),'=onenote(Y,head(m),tail(m));']);
end
for m=1:28          %傅里叶分析

eval(['F',num2str(m),'=fouriernote(y',num2str(m),');']);
end
for m=1:28          %识别音调

eval(['note',num2str(m),'=hz2note(findf(F',num2str(m),'));'
]);
end
for m=1:28          %清除多余变量
```

```
    eval(['clear F',num2str(m),]);
    eval(['clear y',num2str(m),]);
end
```

傅里叶变换的结果正确，如右图(以 y28 为例,x 轴为角频率)：



脚本文件中的 Findf 函数是根据傅里叶变换的结果判断音调的函数。在仔细研究过所有音调的傅里叶变换后的图像后，我发现很多音调人耳听到的是高音，但由于低音部分同时弹奏的和弦（通常是最低音 F）的幅值很大，高于了真实的音，所以直接判断音域内所有音的最大值是不行的，可以先排除 F。

另外，有些音的二次谐波分量高于基波的幅值，因此需要判断找出的幅值最大的频率点是否是二次谐波的额频率。可以假如判断，当该频率的一半的点的幅值大于该该点的幅值的一定比例时，就判定其为二次谐波。具体代码如下：

```
function f=findf(F)
[~,f]=max(F(360:1361));
if (F(round((f+360)/2))>(F(360+f)*3/7)&&(f>360))  %判断是否为谐波
    f=(f+360)/2-360;
end
f=(f-1+360)/2;
```

再将求得的频率通过 hz2note 函数识别后，确定音调。代码如下：

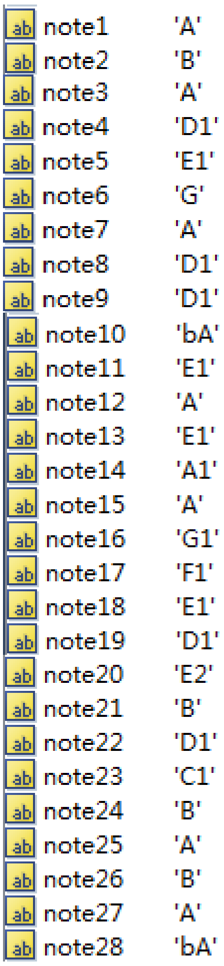
```
function note=hz2note(f)
if f>=169&&f<179.61
    note='F';
elseif f>=179.61&&f<190.5
    note='bG';
elseif f>=190.5&&f<201.7
    note='G';
elseif f>=201.7&&f<213.8
    note='bA';
elseif f>=213.8&&f<226.5
```

```
    note=’A’;
elseif f>=226.5&&f<240
    note=’bB’;
elseif f>=240&&f<254.2
    note=’B’;
elseif f>=254.2&&f<269.4
    note=’C1’;
elseif f>=269.4&&f<285.4
    note=’bD1’;
elseif f>=285.4&&f<302.4
    note=’D1’;
elseif f>=302.4&&f<320.4
    note=’bE1’;
elseif f>=320.4&&f<339.4
    note=’E1’;
elseif f>=339.4&&f<359.6
    note=’F1’;
elseif f>=359.6&&f<381
    note=’bG1’;
elseif f>=381&&f<403.65
    note=’G1’;
elseif f>=403.65&&f<427.6
    note=’bA1’;
elseif f>=427.6&&f<453.1
    note=’A1’;
elseif f>=453.1&&f<480
    note=’bB1’;
elseif f>=480&&f<508.6
    note=’B1’;
elseif f>=508.6&&f<538.8
    note=’C2’;
elseif f>=538.8&&f<570.8
    note=’bD2’;
elseif f>=570.8&&f<604.8
    note=’D2’;
elseif f>=604.8&&f<640.75
    note=’bE2’;
elseif f>=640.75&&f<=680
    note=’E2’;
end
```

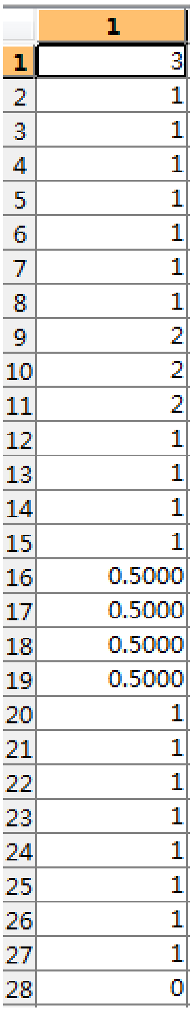
如此便实现了题目要求的所有功能。经检验，节拍的分析全部正确，有三个和弦音的音调检测有误，其余都正确。

结果如下：

音调



note1	'A'
note2	'B'
note3	'A'
note4	'D1'
note5	'E1'
note6	'G'
note7	'A'
note8	'D1'
note9	'D1'
note10	'bA'
note11	'E1'
note12	'A'
note13	'E1'
note14	'A1'
note15	'A'
note16	'G1'
note17	'F1'
note18	'E1'
note19	'D1'
note20	'E2'
note21	'B'
note22	'D1'
note23	'C1'
note24	'B'
note25	'A'
note26	'B'
note27	'A'
note28	'bA'



	1
1	3
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	2
10	2
11	2
12	1
13	1
14	1
15	1
16	0.5000
17	0.5000
18	0.5000
19	0.5000
20	1
21	1
22	1
23	1
24	1
25	1
26	1
27	1
28	0

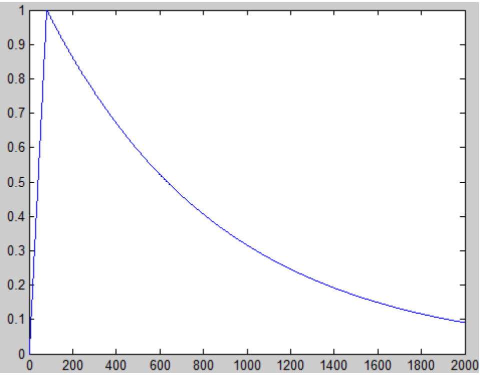
节拍

10. 用(7) 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？

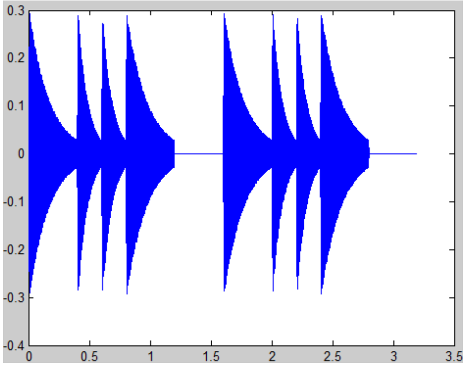
这一问在第四题的基础上，根据第 8 题的结果修改谐波分量幅值即可完成。但试听后发现并不理想。
由于吉他是弹奏型乐器，通过上一题对包络的观察，我发现它的包络波形上升过程较快，下降过程类似指数型衰减。因此我在第四题基础上修改了包络。代码如下：

```
clear all, close all, clc;           %加入二、三、四、七次谐波
t1=linspace(0,0.25-1/8000,2000)';   %修正8分音符
set1=ones(2000,1);
set1(t1<0.01)=100*t1(t1<0.01);
set1(t1>0.01)=exp(-10*(t1(t1>0.01)-0.01));
t2=linspace(0,0.5-1/8000,4000)';    %修正4分音符
set2=ones(4000,1);
set2(t2<0.01)=100*t1(t2<0.01);
```

```
set2(t2>0.01)=exp(-4.8*(t2(t2>0.01)-0.01));
y0=zeros(4000,1);
y1=set2.*note4a(392);          %加入谐波后的4分音符
y2=set1.*note8a(392);          %加入谐波后的8分音符
y3=set1.*note8a(440);
y4=set2.*note4a(293.66);
y5=set2.*note4a(261.63);
y6=set1.*note8a(261.63);
y7=set1.*note8a(220);
y8=set2.*note4a(293.66);
y=[y1;y2;y3;y4;y0;y5;y6;y7;y8;y0];
sound(y,8000)

function note=note4a(x)    %加谐波后的4分音符函数 x为频率
t=linspace(0,0.5-1/4000,4000)';
note=0.082*sin(x*2*pi*t)+0.120*sin(2*x*2*pi*t)+...

0.079*sin(3*x*2*pi*t)+0.091*sin(4*x*2*pi*t)+0.0297*sin(7*x*
2*pi*t);
%加入第8问分析得的谐波

function note=note8a(x)         %加谐波后的8分音符函数 x为频
率
t=linspace(0,0.25-1/2000,2000)';
note=0.082*sin(x*2*pi*t)+0.120*sin(2*x*2*pi*t)+...

0.079*sin(3*x*2*pi*t)+0.091*sin(4*x*2*pi*t)+0.0297*sin(7*x*
2*pi*t);
%加入第8问分析得的谐波
```



包络修正



音乐波形

再次试听后，声音比较接近拨弦乐器的声音，相比之前已经有了明显改善，但与吉他声音还是有一定差距。

11. 也许(9) 还不是很像，因为对于一把泛音丰富的吉他而言，不可能每个音调对应的泛音数量和幅度都相同。但是通过完成第(8) 题，你已经提取出 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。现在就来演奏一曲《东方红》吧。

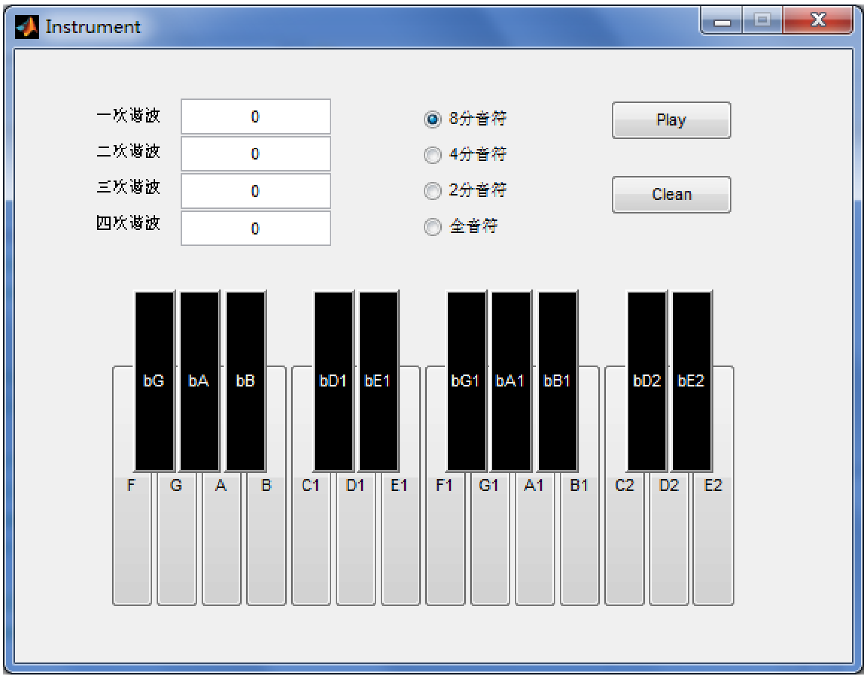
完成此题，只需修改 note4 和 note8 的代码，使其根据不同频率添加第 9 题所得的不同的谐波分量即可。

```
function note=note4b(x)      %加谐波后的4分音符函数 x为频率
t=linspace(0,0.5-1/4000,4000)';
if x==392   %G1
    note=1*sin(x*2*pi*t)+0.46*sin(2*x*2*pi*t)+...
        0.092*sin(3*x*2*pi*t)+0.066*sin(4*x*2*pi*t);
elseif x==293.66   %D1
    note=1*sin(x*2*pi*t)+0.673*sin(2*x*2*pi*t)+...
        0.153*sin(3*x*2*pi*t)+0.064*sin(4*x*2*pi*t);
elseif x==261.63   %C1
    note=1*sin(x*2*pi*t)+0.553*sin(2*x*2*pi*t)+...
        0.285*sin(3*x*2*pi*t)+0.066*sin(4*x*2*pi*t);
end

function note=note8b(x)         %加谐波后的8分音符函数 x为频
率
t=linspace(0,0.25-1/2000,2000)';
if x==392   %G1
    note=1*sin(x*2*pi*t)+0.46*sin(2*x*2*pi*t)+...
        0.092*sin(3*x*2*pi*t)+0.066*sin(4*x*2*pi*t);
elseif x==440   %A1
    note=1*sin(x*2*pi*t)+0.440*sin(2*x*2*pi*t)+...
        0.564*sin(3*x*2*pi*t)+0.048*sin(4*x*2*pi*t);
elseif x==261.63   %C1
    note=1*sin(x*2*pi*t)+0.553*sin(2*x*2*pi*t)+...
        0.285*sin(3*x*2*pi*t)+0.066*sin(4*x*2*pi*t);
elseif x==220   %A
    note=1*sin(x*2*pi*t)+0.243*sin(2*x*2*pi*t)+...
        0.270*sin(3*x*2*pi*t)+0.026*sin(4*x*2*pi*t);
end
```

试听后发现确实效果比前一题更好了。但是还是要承认，简单的谐波和包络模型，距离真实的吉他还是有一定距离。

12. 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，在学完本书后面内容之后，试着做一个图形界面把上述功能封装起来。

根据要求，我设计了如下的图形界面。该程序可以通过手动设置一至四次谐波分量改变音色，可以设置弹奏的每个音的节拍，具有播放和清空按钮，并且界面美观、简洁。



Instrument

一次谐波 0
二次谐波 0
三次谐波 0
四次谐波 0

● 8分音符
○ 4分音符
○ 2分音符
○ 全音符

在完成此题时，因为书上对于这一部分的介绍相对比较少，所遇到了不少麻烦。比如设置和访问全局变量的方法，handles 的使用，以及 set，get 函数的运用，都是在错误中一边摸索一边前进的。

在设计的过程中，有一些部分是通过上网学习掌握的，比如多个 radiobutton 的互斥的设置方法，判断文本框为空的方法等。

在设计的过程中，我注意了提高图形界面的鲁棒性，比如判断文本框为空，判断未弹奏键盘等等。

在这最后一题的完成过程中，我尝试了一些之前没有用过的方法和函数。比如在这道题中我的函数都设为了子函数，并将包络修正融入了乐音波形产生函数，并且将各种拍子的音符，通过一个参数判断，合并为了一个函数，使代码更加简洁、高效。

代码过长，见后附。

三、 实验总结

此次试验是我亲身实践的第一个 matlab 实验，它见证了我对 matlab 的认识的从无到有的过程，让我感受到了 mablab 这个软件的强大之处，也使我体会到了信号与系统与实际相结合的魅力所在。

同时它是一个对前六章知识进行总结的综合性作业，在完成作业的过程中，我不仅熟悉了 matlab 基本指令、函数的使用，也对处理连续时间信号方法等有了更深刻的认识。

最后，这次实验切实地锻炼了我的研究能力。我在实验的过程中遇到的难题后如何想办法攻克，自己如何一步一步发现问题，分析原因并找到解决办法，以及查阅 help 和相关资料的能力等等，都在这次实验中得到了锻炼。

四、 附 GUI 代码（1-11 题代码已在题中说明同时给出）：

```
function varargout = Instrument(varargin)
%INSTRUMENT M-file for Instrument.fig
%      INSTRUMENT, by itself, creates a new INSTRUMENT or raises the existing
%      singleton*.
%
%      H = INSTRUMENT returns the handle to a new INSTRUMENT or the handle to
%      the existing singleton*.
%
%      INSTRUMENT(’Property’,’Value’,...) creates a new INSTRUMENT using the
%      given property value pairs. Unrecognized properties are passed via
%      varargin to Instrument_OpeningFcn.  This calling syntax produces a
%      warning when there is an existing singleton*.
%
%      INSTRUMENT(’CALLBACK’) and INSTRUMENT(’CALLBACK’,hObject,...) call the
%      local function named CALLBACK in INSTRUMENT.M with the given input
%      arguments.
%
%      *See GUI Options on GUIDE’s Tools menu.  Choose ″GUI allows only one
%      instance to run (singleton)″.
%
% See also: GUIDE, GUIDATA, GUIHANDLES
% Edit the above text to modify the response to help Instrument

% Last Modified by GUIDE v2.5 19-Jul-2011 23:20:45

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Instrument_OpeningFcn, ...
                   'gui_OutputFcn',  @Instrument_OutputFcn, ...
                   'gui_LayoutFcn',  [], ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State,
varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Instrument is made visible.
function Instrument_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   unrecognized PropertyName/PropertyValue pairs from
the
%            command line (see VARARGIN)

% Choose default command line output for Instrument
handles.output = hObject;
handles.notenum=0;
handles.x=[];
handles.beat=[];
% Update handles structure
```

```
guidata(hObject, handles);
set(handles.radiobutton1,'value',1);

% UIWAIT makes Instrument wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Instrument_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
handles.notenum=(handles.notenum+1);
handles.x(handles.notenum)=174.61;
if get(handles.radiobutton1,'value')
    handles.beat(handles.notenum)=8;
elseif get(handles.radiobutton2,'value')
    handles.beat(handles.notenum)=4;
elseif get(handles.radiobutton3,'value')
    handles.beat(handles.notenum)=2;
elseif get(handles.radiobutton4,'value')
    handles.beat(handles.notenum)=1;
end
guidata(hObject,handles);
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
handles.notenum=handles.notenum+1;
handles.x(handles.notenum)=196;
if get(handles.radiobutton1,'value')
    handles.beat(handles.notenum)=8;
```

```
elseif get(handles.radiobutton2,'value')
    handles.beat(handles.notenum)=4;
elseif get(handles.radiobutton3,'value')
    handles.beat(handles.notenum)=2;
elseif get(handles.radiobutton4,'value')
    handles.beat(handles.notenum)=1;
end
guidata(hObject,handles);
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
handles.notenum=handles.notenum+1;
handles.x(handles.notenum)=220;
if get(handles.radiobutton1,'value')
    handles.beat(handles.notenum)=8;
elseif get(handles.radiobutton2,'value')
    handles.beat(handles.notenum)=4;
elseif get(handles.radiobutton3,'value')
    handles.beat(handles.notenum)=2;
elseif get(handles.radiobutton4,'value')
    handles.beat(handles.notenum)=1;
end
guidata(hObject,handles);
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
handles.notenum=handles.notenum+1;
handles.x(handles.notenum)=246.94;
if get(handles.radiobutton1,'value')
    handles.beat(handles.notenum)=8;
elseif get(handles.radiobutton2,'value')
    handles.beat(handles.notenum)=4;
elseif get(handles.radiobutton3,'value')
    handles.beat(handles.notenum)=2;
elseif get(handles.radiobutton4,'value')
    handles.beat(handles.notenum)=1;
```

```
end
guidata(hObject,handles);
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata, handles)
handles.notenum=handles.notenum+1;
handles.x(handles.notenum)=261.63;
if get(handles.radiobutton1,’value’)
    handles.beat(handles.notenum)=8;
elseif get(handles.radiobutton2,’value’)
    handles.beat(handles.notenum)=4;
elseif get(handles.radiobutton3,’value’)
    handles.beat(handles.notenum)=2;
elseif get(handles.radiobutton4,’value’)
    handles.beat(handles.notenum)=1;
end
guidata(hObject,handles);
% hObject    handle to pushbutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
handles.notenum=handles.notenum+1;
handles.x(handles.notenum)=440;
if get(handles.radiobutton1,’value’)
    handles.beat(handles.notenum)=8;
elseif get(handles.radiobutton2,’value’)
    handles.beat(handles.notenum)=4;
elseif get(handles.radiobutton3,’value’)
    handles.beat(handles.notenum)=2;
elseif get(handles.radiobutton4,’value’)
    handles.beat(handles.notenum)=1;
end
guidata(hObject,handles);
% hObject    handle to pushbutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --- Executes on button press in pushbutton9.
function pushbutton9_Callback(hObject, eventdata, handles)
handles.notenum=handles.notenum+1;
handles.x(handles.notenum)=392;
if get(handles.radiobutton1,'value')
    handles.beat(handles.notenum)=8;
elseif get(handles.radiobutton2,'value')
    handles.beat(handles.notenum)=4;
elseif get(handles.radiobutton3,'value')
    handles.beat(handles.notenum)=2;
elseif get(handles.radiobutton4,'value')
    handles.beat(handles.notenum)=1;
end
guidata(hObject,handles);
% hObject    handle to pushbutton9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton10.
function pushbutton10_Callback(hObject, eventdata, handles)
handles.notenum=handles.notenum+1;
handles.x(handles.notenum)=349.23;
if get(handles.radiobutton1,'value')
    handles.beat(handles.notenum)=8;
elseif get(handles.radiobutton2,'value')
    handles.beat(handles.notenum)=4;
elseif get(handles.radiobutton3,'value')
    handles.beat(handles.notenum)=2;
elseif get(handles.radiobutton4,'value')
    handles.beat(handles.notenum)=1;
end
guidata(hObject,handles);
% hObject    handle to pushbutton10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton11.
function pushbutton11_Callback(hObject, eventdata, handles)
handles.notenum=handles.notenum+1;
handles.x(handles.notenum)=329.63;
if get(handles.radiobutton1,'value')
```

```
    handles.beat(handles.notenum)=8;
elseif get(handles.radiobutton2,'value')
    handles.beat(handles.notenum)=4;
elseif get(handles.radiobutton3,'value')
    handles.beat(handles.notenum)=2;
elseif get(handles.radiobutton4,'value')
    handles.beat(handles.notenum)=1;
end
guidata(hObject,handles);
% hObject    handle to pushbutton11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton12.
function pushbutton12_Callback(hObject, eventdata, handles)
handles.notenum=handles.notenum+1;
handles.x(handles.notenum)=293.66;
if get(handles.radiobutton1,'value')
    handles.beat(handles.notenum)=8;
elseif get(handles.radiobutton2,'value')
    handles.beat(handles.notenum)=4;
elseif get(handles.radiobutton3,'value')
    handles.beat(handles.notenum)=2;
elseif get(handles.radiobutton4,'value')
    handles.beat(handles.notenum)=1;
end
guidata(hObject,handles);
% hObject    handle to pushbutton12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton13.
function pushbutton13_Callback(hObject, eventdata, handles)
handles.notenum=handles.notenum+1;
handles.x(handles.notenum)=659.25;
if get(handles.radiobutton1,'value')
    handles.beat(handles.notenum)=8;
elseif get(handles.radiobutton2,'value')
    handles.beat(handles.notenum)=4;
elseif get(handles.radiobutton3,'value')
    handles.beat(handles.notenum)=2;
elseif get(handles.radiobutton4,'value')
```

```
    handles.beat(handles.notenum)=1;
end
guidata(hObject,handles);
% hObject    handle to pushbutton13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton14.
function pushbutton14_Callback(hObject, eventdata, handles)
handles.notenum=handles.notenum+1;
handles.x(handles.notenum)=587.33;
if get(handles.radiobutton1,’value’)
    handles.beat(handles.notenum)=8;
elseif get(handles.radiobutton2,’value’)
    handles.beat(handles.notenum)=4;
elseif get(handles.radiobutton3,’value’)
    handles.beat(handles.notenum)=2;
elseif get(handles.radiobutton4,’value’)
    handles.beat(handles.notenum)=1;
end
guidata(hObject,handles);
% hObject    handle to pushbutton14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton15.
function pushbutton15_Callback(hObject, eventdata, handles)
handles.notenum=handles.notenum+1;
handles.x(handles.notenum)=523.25;
if get(handles.radiobutton1,’value’)
    handles.beat(handles.notenum)=8;
elseif get(handles.radiobutton2,’value’)
    handles.beat(handles.notenum)=4;
elseif get(handles.radiobutton3,’value’)
    handles.beat(handles.notenum)=2;
elseif get(handles.radiobutton4,’value’)
    handles.beat(handles.notenum)=1;
end
guidata(hObject,handles);
% hObject    handle to pushbutton15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --- Executes on button press in pushbutton16.
function pushbutton16_Callback(hObject, eventdata, handles)
handles.notenum=handles.notenum+1;
handles.x(handles.notenum)=493.88;
if get(handles.radiobutton1,'value')
    handles.beat(handles.notenum)=8;
elseif get(handles.radiobutton2,'value')
    handles.beat(handles.notenum)=4;
elseif get(handles.radiobutton3,'value')
    handles.beat(handles.notenum)=2;
elseif get(handles.radiobutton4,'value')
    handles.beat(handles.notenum)=1;
end
guidata(hObject,handles);
% hObject    handle to pushbutton16 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton17.
function pushbutton17_Callback(hObject, eventdata, handles)
handles.notenum=handles.notenum+1;
handles.x(handles.notenum)=184.99;
if get(handles.radiobutton1,'value')
    handles.beat(handles.notenum)=8;
elseif get(handles.radiobutton2,'value')
    handles.beat(handles.notenum)=4;
elseif get(handles.radiobutton3,'value')
    handles.beat(handles.notenum)=2;
elseif get(handles.radiobutton4,'value')
    handles.beat(handles.notenum)=1;
end
guidata(hObject,handles);
% hObject    handle to pushbutton17 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton18.
function pushbutton18_Callback(hObject, eventdata, handles)
handles.notenum=handles.notenum+1;
handles.x(handles.notenum)=207.65;
```

```
if get(handles.radiobutton1,'value')
    handles.beat(handles.notenum)=8;
elseif get(handles.radiobutton2,'value')
    handles.beat(handles.notenum)=4;
elseif get(handles.radiobutton3,'value')
    handles.beat(handles.notenum)=2;
elseif get(handles.radiobutton4,'value')
    handles.beat(handles.notenum)=1;
end
guidata(hObject,handles);
% hObject    handle to pushbutton18 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton19.
function pushbutton19_Callback(hObject, eventdata, handles)
handles.notenum=handles.notenum+1;
handles.x(handles.notenum)=233.08;
if get(handles.radiobutton1,'value')
    handles.beat(handles.notenum)=8;
elseif get(handles.radiobutton2,'value')
    handles.beat(handles.notenum)=4;
elseif get(handles.radiobutton3,'value')
    handles.beat(handles.notenum)=2;
elseif get(handles.radiobutton4,'value')
    handles.beat(handles.notenum)=1;
end
guidata(hObject,handles);
% hObject    handle to pushbutton19 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton20.
function pushbutton20_Callback(hObject, eventdata, handles)
handles.notenum=handles.notenum+1;
handles.x(handles.notenum)=369.99;
if get(handles.radiobutton1,'value')
    handles.beat(handles.notenum)=8;
elseif get(handles.radiobutton2,'value')
    handles.beat(handles.notenum)=4;
elseif get(handles.radiobutton3,'value')
    handles.beat(handles.notenum)=2;
```

```
elseif get(handles.radiobutton4,'value')
    handles.beat(handles.notenum)=1;
end
guidata(hObject,handles);
% hObject    handle to pushbutton20 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton21.
function pushbutton21_Callback(hObject, eventdata, handles)
handles.notenum=handles.notenum+1;
handles.x(handles.notenum)=415.30;
if get(handles.radiobutton1,'value')
    handles.beat(handles.notenum)=8;
elseif get(handles.radiobutton2,'value')
    handles.beat(handles.notenum)=4;
elseif get(handles.radiobutton3,'value')
    handles.beat(handles.notenum)=2;
elseif get(handles.radiobutton4,'value')
    handles.beat(handles.notenum)=1;
end
guidata(hObject,handles);
% hObject    handle to pushbutton21 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton22.
function pushbutton22_Callback(hObject, eventdata, handles)
handles.notenum=handles.notenum+1;
handles.x(handles.notenum)=466.16;
if get(handles.radiobutton1,'value')
    handles.beat(handles.notenum)=8;
elseif get(handles.radiobutton2,'value')
    handles.beat(handles.notenum)=4;
elseif get(handles.radiobutton3,'value')
    handles.beat(handles.notenum)=2;
elseif get(handles.radiobutton4,'value')
    handles.beat(handles.notenum)=1;
end
guidata(hObject,handles);
% hObject    handle to pushbutton22 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton23.
function pushbutton23_Callback(hObject, eventdata, handles)
handles.notenum=handles.notenum+1;
handles.x(handles.notenum)=277.18;
if get(handles.radiobutton1,'value')
    handles.beat(handles.notenum)=8;
elseif get(handles.radiobutton2,'value')
    handles.beat(handles.notenum)=4;
elseif get(handles.radiobutton3,'value')
    handles.beat(handles.notenum)=2;
elseif get(handles.radiobutton4,'value')
    handles.beat(handles.notenum)=1;
end
guidata(hObject,handles);
% hObject    handle to pushbutton23 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton24.
function pushbutton24_Callback(hObject, eventdata, handles)
handles.notenum=handles.notenum+1;
handles.x(handles.notenum)=311.13;
if get(handles.radiobutton1,'value')
    handles.beat(handles.notenum)=8;
elseif get(handles.radiobutton2,'value')
    handles.beat(handles.notenum)=4;
elseif get(handles.radiobutton3,'value')
    handles.beat(handles.notenum)=2;
elseif get(handles.radiobutton4,'value')
    handles.beat(handles.notenum)=1;
end
guidata(hObject,handles);
% hObject    handle to pushbutton24 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton25.
function pushbutton25_Callback(hObject, eventdata, handles)
handles.notenum=handles.notenum+1;
handles.x(handles.notenum)=554.36;
if get(handles.radiobutton1,'value')
    handles.beat(handles.notenum)=8;
elseif get(handles.radiobutton2,'value')
    handles.beat(handles.notenum)=4;
elseif get(handles.radiobutton3,'value')
    handles.beat(handles.notenum)=2;
elseif get(handles.radiobutton4,'value')
    handles.beat(handles.notenum)=1;
end
guidata(hObject,handles);
% hObject    handle to pushbutton25 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton26.
function pushbutton26_Callback(hObject, eventdata, handles)
handles.notenum=handles.notenum+1;
handles.x(handles.notenum)=622.25;
if get(handles.radiobutton1,'value')
    handles.beat(handles.notenum)=8;
elseif get(handles.radiobutton2,'value')
    handles.beat(handles.notenum)=4;
elseif get(handles.radiobutton3,'value')
    handles.beat(handles.notenum)=2;
elseif get(handles.radiobutton4,'value')
    handles.beat(handles.notenum)=1;
end
guidata(hObject,handles);
% hObject    handle to pushbutton26 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton28.
function pushbutton28_Callback(hObject, eventdata, handles)
if(handles.notenum)>=1          %已按键则演奏
    if isempty(get(handles.edit4,'String'))      %判断输入框为空则改谐波分量置0
        a1=0;
    else
        a1=str2double(get(handles.edit4,'String'));
    end
    if isempty(get(handles.edit5,’String’))
        a2=0;
    else
        a2=str2double(get(handles.edit5,’String’));
    end
    if isempty(get(handles.edit2,’String’))
        a3=0;
    else
        a3=str2double(get(handles.edit2,’String’));
    end
    if isempty(get(handles.edit3,’String’))
        a4=0;
    else
        a4=str2double(get(handles.edit3,’String’));
    end
    for m=1:handles.notenum                   %产生每个按键的波
形序列

eval([’note’,num2str(m),’=note(handles.x(m),handles.beat(m),a1,
a2,a3,a4);’]);
    end
    Y=note1;
    for m=2:handles.notenum              %合并波形
        eval([’Y=[Y;note’,num2str(m),’];’]);
    end
    sound(Y,8000);                        %播放演奏音乐
    handles.notenum=0;                    %播放后重置
    guidata(hObject,handles);
end

% hObject    handle to pushbutton28 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,’String’) returns contents of edit2 as text
%        str2double(get(hObject,’String’)) returns contents of
edit2 as a double
```

```
% --- Executes during object creation, after setting all
properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of
edit3 as a double


% --- Executes during object creation, after setting all
properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4 as text
%        str2double(get(hObject,'String')) returns contents of
edit4 as a double


% --- Executes during object creation, after setting all
properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit5_Callback(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit5 as text
%        str2double(get(hObject,'String')) returns contents of
edit5 as a double


% --- Executes during object creation, after setting all
properties.
function edit5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,’BackgroundColor’),
get(0,’defaultUicontrolBackgroundColor’))
    set(hObject,’BackgroundColor’,’white’);
end


% --- Executes on button press in radiobutton1.
function radiobutton1_Callback(hObject, eventdata, handles)
set(handles.radiobutton1,’value’,1);
set(handles.radiobutton2,’value’,0);         %使4个radiobuttion
排斥实现单选
set(handles.radiobutton3,’value’,0);
set(handles.radiobutton4,’value’,0);

% hObject    handle to radiobutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% Hint: get(hObject,’Value’) returns toggle state of radiobutton1


% --- Executes on button press in radiobutton2.
function radiobutton2_Callback(hObject, eventdata, handles)
set(handles.radiobutton1,’value’,0);
set(handles.radiobutton2,’value’,1);
set(handles.radiobutton3,’value’,0);
set(handles.radiobutton4,’value’,0);
% hObject    handle to radiobutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,’Value’) returns toggle state of radiobutton2


% --- Executes on button press in radiobutton3.
function radiobutton3_Callback(hObject, eventdata, handles)
set(handles.radiobutton1,’value’,0);
set(handles.radiobutton2,’value’,0);
set(handles.radiobutton3,'value',1);
set(handles.radiobutton4,'value',0);
% hObject    handle to radiobutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton3


% --- Executes on button press in radiobutton4.
function radiobutton4_Callback(hObject, eventdata, handles)
set(handles.radiobutton1,'value',0);
set(handles.radiobutton2,'value',0);
set(handles.radiobutton3,'value',0);
set(handles.radiobutton4,'value',1);
% hObject    handle to radiobutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton4


% --- Executes on button press in pushbutton29.
function pushbutton29_Callback(hObject, eventdata,
handles)  %clean按钮
handles.notenum=0;                           %重新输入
guidata(hObject,handles);
% hObject    handle to pushbutton29 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

function note=note(x,beat,a1,a2,a3,a4)    %根据频率、节拍和谐波分
量构建波形并修正
t=linspace(0,(2/beat)-1/(16000/beat),(16000/beat))';
set=ones((16000/beat),1);
set(t<0.05)=20*t(t<0.05);                 %修正包络上升沿
set(t>((2/beat)-0.1))=10*(2/beat-t(t>((2/beat)-0.1)));    %修正
包络下降沿
note=a1*sin(x*2*pi*t)+a2*sin(2*x*2*pi*t)+a3*sin(3*x*2*pi*t)+a4*
sin(4*x*2*pi*t);
note=set.*note;    %修正包络
```