

可视化作业5

18300290007 加兴华

注：本次作业均在matlab环境下完成

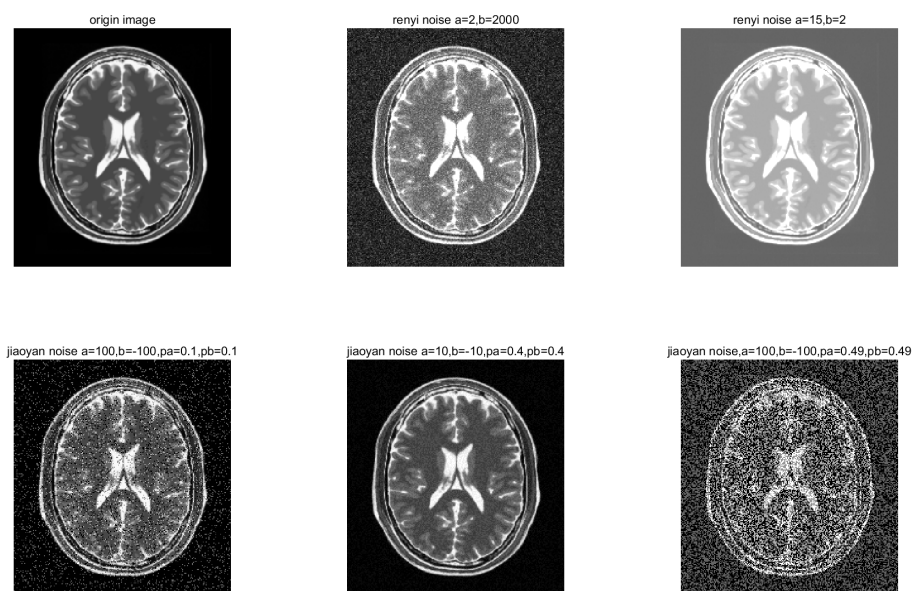
[HW5-1]

针对对**大脑**、**心脏图像**（或其他多类图像），生成其对应不同类型（瑞利噪声、椒盐噪声）的不同强度（至少2种高强度）的噪声污染图像。

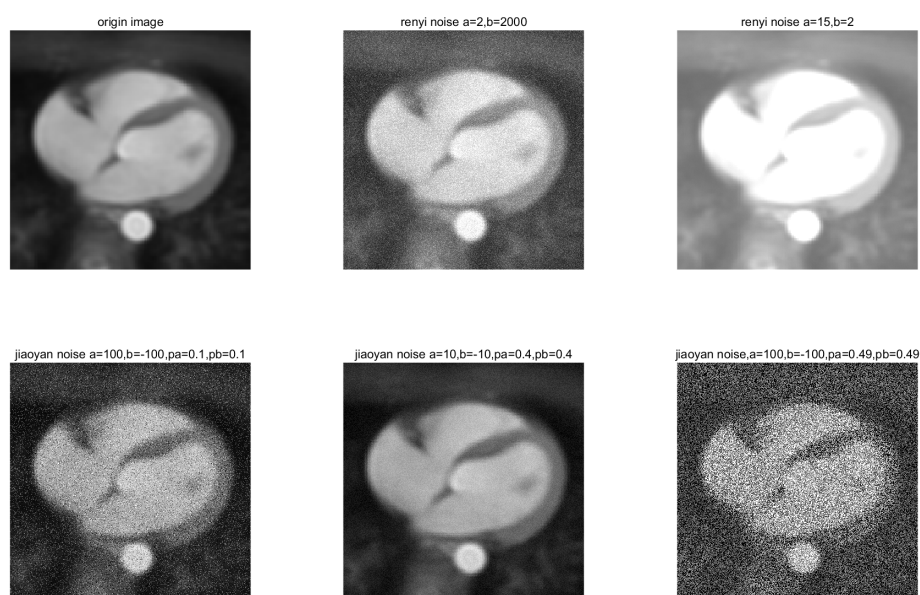
用于增加噪声的函数如下：

```
1  % 瑞利噪声
2  function x=renyi(img,a,b)
3  [m,n]=size(img);
4  noise=rand(m,n); % 生成01均匀分布的同规模矩阵
5  % 将每个元素按噪声pdf反变换等效按噪声分布生成噪声矩阵
6  noise=a+(-b.*log(1-noise)).^0.5;
7  % 叠加
8  x=img+uint8(noise);
9  end
10
11 % 椒盐噪声
12 function y=jiaoyan(img,a,b,pa,pb)
13 [m,n]=size(img);
14 noise=zeros(m,n); % 申请噪声矩阵空间
15 cdf=rand(m,n); % 生成01均匀分布的同规模矩阵
16 % 将每个元素按噪声pdf反变换等效按噪声分布生成噪声矩阵
17 noise(cdf<pa)=a;
18 noise(cdf>1-pb)=b;
19 % 叠加
20 y=uint8(double(img)+noise);
21 end
```

应用在‘大脑图像’上可获得如下结果：



应用在‘心脏图像’上可获得如下结果：



分析：

【1】瑞利噪声，噪声点的多寡取决于 b 的大小， b 越大噪声点越多；而 a 并非带来直观上的噪声， a 会产生‘灰度’意义上的噪声：使原图的亮度发生偏移。

【2】椒盐噪声， a 值决定白噪声的亮度， b 值决定黑噪声的亮度； pa 决定白噪声的密度， pb 决定黑噪声的密度。可以观察到， a 和 b 的值越大，噪声就越明显； pa 和 pb 的值越大（加起来不能超过1），噪声点就越多；另外， pa 与 pb 相等时，值小时白噪音更明显，值大时黑噪音更明显，这可能是由于人眼视觉特性导致的。

[HW5-2]

实现OSTU二类分割算法，K类均值分类的分割算法和基于高斯混合模型的分割算法，并测试一下上述高强度噪声污染的图像的分割结果：

(1)

测试二类分割，并对比三个算法结果的差异；

otsu函数实现：

```
1 function a=otsu(img)
2 % 统计图像信息到bin
3 [m,n]=size(img);
4 bin=zeros(256,1);
5 for i=1:m
6     for j=1:n
7         bin(img(i,j)+1)=bin(img(i,j)+1)+1;
8     end
9 end
10 % 初始化
11 s_max =[0,0];
12 a=zeros(m,n);
13 N=m*n; % 像素个数
14 i=0:255; %递增行向量，用于期望的向量表示
15 for threshold=1:256
16     u=0;
17     n_0 = sum(bin(1:threshold)); % 阈值以下像素数
18     n_1 = sum(bin(threshold:256)); % 阈值以上像素数
19     % 两侧频率
20     w_0 = n_0/N;
21     w_1 = n_1/N;
22     % 阈值下平均灰度
23     if(n_0>0)
24         u_0 = i(1:threshold)*bin(1:threshold)/n_0;
25     else
26         u_0=0; %考虑极端划分
27     end
28     % 阈值上平均灰度
29     if(n_1>0)
30         u_1 = i(threshold:256)*bin(threshold:256)/n_1;
31     else
32         u_1=0; %考虑极端划分
33     end
34     % 总平均灰度
35     u = w_0*u_0 + w_1*u_1;
36     % 类间方差
37     Dbet2 = w_0*((u_0-u)^2) + w_1*((u_1-u)^2);
38     % 跟先前最优比较(取绝对大者意味着右偏，二分阈值需分配给左半边)
39     if (Dbet2>s_max(2))
40         s_max=[threshold-1,Dbet2];
41     end
42 end
43 t=s_max(1);
44 for i=1:m
45     for j=1:n
46         if img(i,j)>t a(i,j)=255; end
47     end
```

```
48 end
49 end
```

k—means函数实现:

```
1 function b=mykmeans(img,k)
2 %-----
3 % 随机初始化
4 u=rand(k,1)*255;
5 u=sort(u);
6 img=double(img); % 浮点化防止后面运算报错
7 [m,n]=size(img);
8 c=zeros(m,n); % 每个像素点的类记录矩阵
9 I=ones(m,n); % 用于后续便捷表达的辅助矩阵
10 d=ones(k,1); % 每一类的迭代差距
11 err=10; % 初始差距
12 %-----
13 %正式迭代
14 while(err>0.1)
15 %根据当前每类的中心界定像素点类型
16     for i=1:m
17         for j=1:n
18             min=255;
19             for r=1:k
20                 if abs(img(i,j)-u(r))<=min
21                     min=abs(img(i,j)-u(r));
22                     c(i,j)=r-1;
23                 end
24             end
25         end
26     end
27 %计算每一类点的灰度期望作为下一迭代的类中心
28     for r=1:k
29         temp=u(r);
30         tot=sum(sum(img(c==r-1)));
31         con=sum(sum(I(c==r-1)));
32         u(r)=tot/con;
33         d(r)=abs(u(r)-temp);
34     end
35 %以最大迭代距离作为本次误差
36     err=max(d);
37 end
38 %转换成灰度图数据格式
39 b=uint8(c/(k-1)*255);
40 end
```

EM法函数实现:

```
1 function c=em(img,k)
2 % double化防止出错
3 img=double(img);
4 % 设置初始均值和方差，均值为随机在色块中选点并取整所得
5 mu=rand(k,1)*255;
```

```

6 mu=sort(mu);
7 sigma =100*ones(1,k);
8 [m,n]=size(img);
9 px = zeros(m,n,k);
10 %全局每个点为某一类的概率
11 mypi = rand(1,k);
12 mypi = mypi/sum(mypi);%将类概率归一化
13 %以迭代次数来作为停止的条件
14 stopiter = 20;
15 iter = 1;
16 while iter <= stopiter
17     %-----E-----
18     N=zeros(m,n,k);
19     for i=1:k
20         for x=1:m
21             for y=1:n
22                 N(x,y,i)=normpdf(img(x,y),mu(i),sigma(i))*mypi(i);
23             end
24         end
25     end
26     % 防止分母为0 生成一个无0近似阵
27     N2=N;
28     N2(N<0.00000001)=0.00000001;
29     for x=1:m
30         for y=1:n
31             sumk_N=sum(N2(x,y,:));
32             px(x,y,:)=N(x,y,:)/sumk_N;
33         end
34     end
35     %-----M-----
36     %更新参数集
37     sum_all_px=sum(sum(sum(px)));
38     for i=1:k
39         sum_xy_px=sum(sum(px(:,:,i)));
40         mu(i)=sum(sum(px(:,:,i).*img))/sum_xy_px;
41         sigma(i) =sqrt(sum(sum(px(:,:,i).*((img-mu(i)).^2)))/sum_xy_px);
42         mypi(i)=sum_xy_px/sum_all_px;
43     end
44     iter=iter+1;
45 end
46 c=zeros(m,n);
47 for x=1:m
48     for y=1:n
49         % 返回x, y处最大概率类型下标
50         [~,c(x,y)]=max(px(x,y,:));
51     end
52 end
53 c=uint8((c-1)/(k-1)*255);
54 end

```

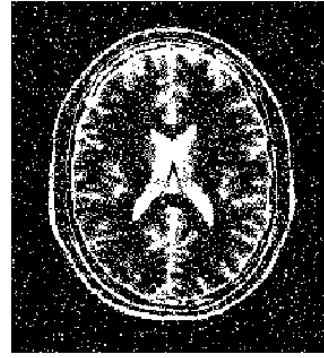
首先需要说明，由于EM法我的初始均值、方差、初始类概率均为随机生成，且迭代次数设置不多，因此EM法可能会有较大的波动性。

对于椒盐噪声，EM法的最佳还原效果比另两者好，另外两种方法差异不大：

origin



EM k=2



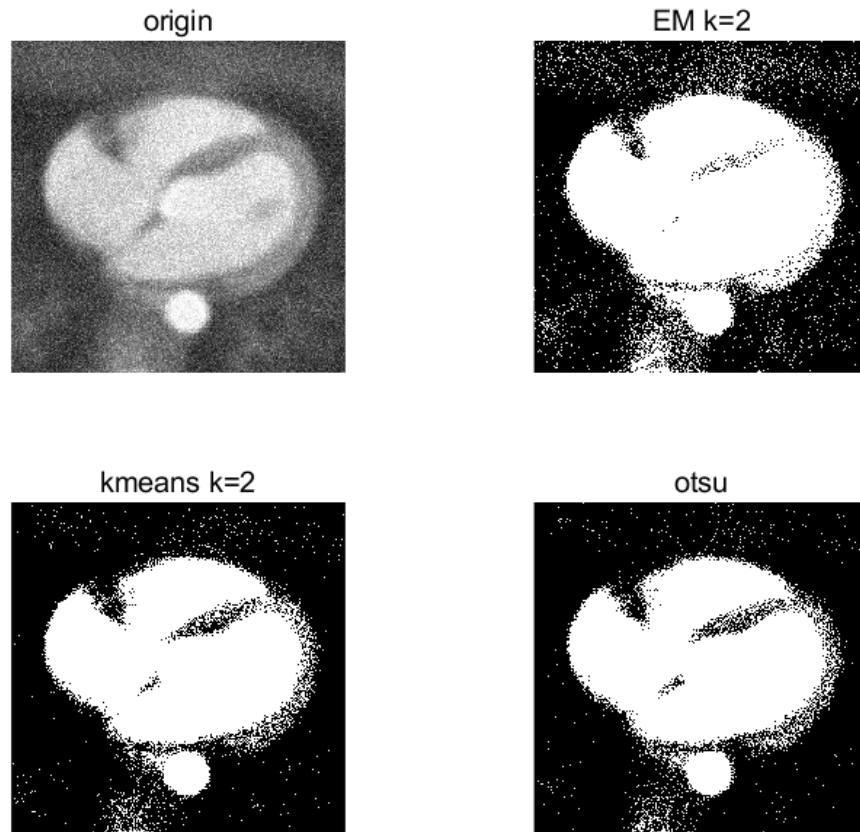
kmeans k=2



otsu



而对于瑞利噪声，经过多次运行代码后，都观察到EM法总是还原效果最差，而otsu竟然此时处理效果最好：



(2)

测试多类（大于等于三类）分割（请自己设定分割标签类别的个数），并对K类均值分类的分割算法和基于高斯混合模型的分割结果的差异；

将EM法初始均值设置为kmeans中心：

```

1  %修改成以kmeans中心作为初始均值
2  function c=em(img,k)
3  img=double(img);
4  % 设置初始均值和方差，均值为随机在各色块中选点并取整所得
5  [mu,~]=mykmeans(img,k);
6  sigma =100*ones(1,k);
7  [m,n]=size(img);
8  px = zeros(m,n,k);
9  %全局每个点为某一类的概率
10 mypi = rand(1,k);
11 mypi = mypi/sum(mypi);%将类概率归一化
12 %以迭代次数来作为停止的条件
13 stopiter = 20;
14 iter = 1;
15 while iter <= stopiter
16     %-----E-----
17     N=zeros(m,n,k);
18     for i=1:k
19         for x=1:m
20             for y=1:n
21                 N(x,y,i)=normpdf(img(x,y),mu(i),sigma(i))*mypi(i);

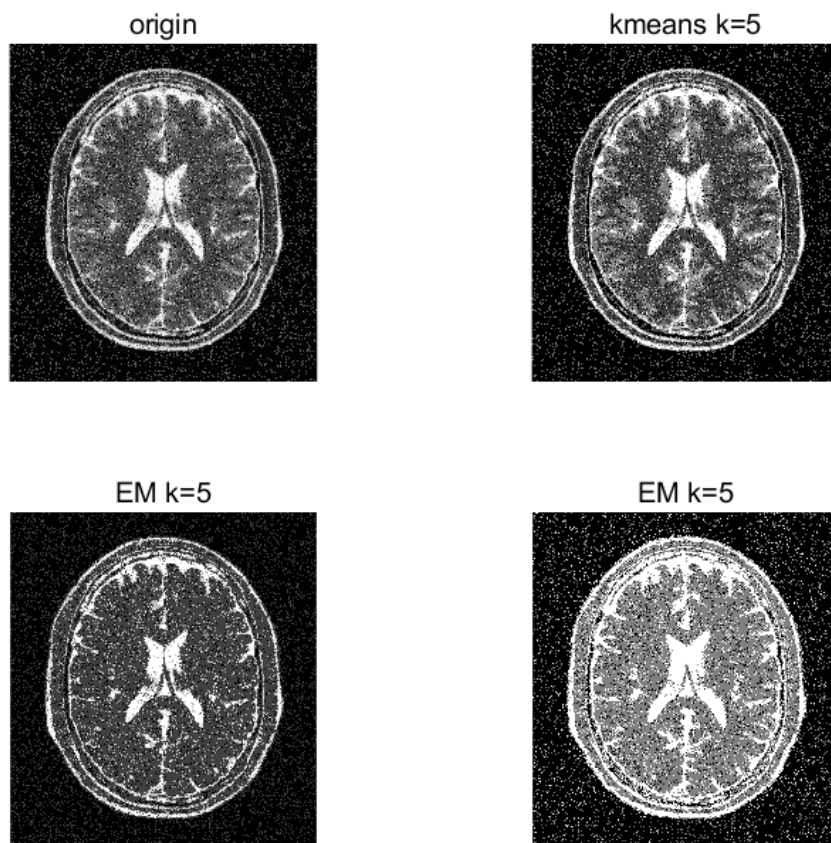
```

```

22         end
23     end
24 end
25 N2=N;
26 N2(N<0.00000001)=0.00000001;% 防止分母为0 生成一个无0近似阵
27 for x=1:m
28     for y=1:n
29         sumk_N=sum(N2(x,y,:));
30         px(x,y,:)=N(x,y,:)/sumk_N;
31     end
32 end
33 %-----M-----
34 %更新参数集
35 sum_all_px=sum(sum(sum(px)));
36 for i=1:k
37     sum_xy_px=sum(sum(px(:,:,i)));
38     mu(i)=sum(sum(px(:,:,i).*img))/sum_xy_px;
39     sigma(i) =sqrt(sum(sum(px(:,:,i).*((img-mu(i)).^2)))/sum_xy_px);
40     mypi(i)=sum_xy_px/sum_all_px;
41 end
42 iter=iter+1;
43 end
44 c=zeros(m,n);
45 for x=1:m
46     for y=1:n
47 % 返回(x, y)处最大概率类型给c
48         [~,c(x,y)]=max(px(x,y,:));
49     end
50 end
51 c=uint8((c-1)/(k-1)*255);
52 end

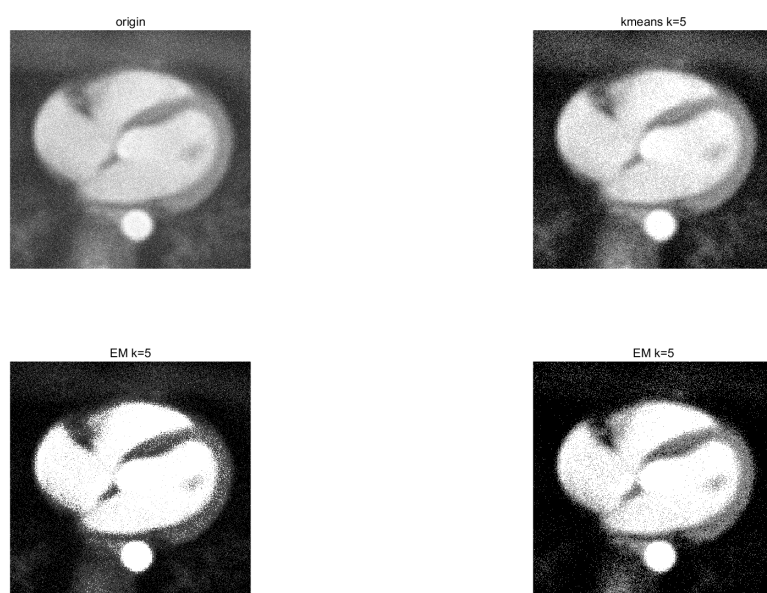
```

一次运行获得下图：



这说明，图片带椒盐噪声时，EM法的最优还原效果确实会比kmeans好，但每次运行不必然达到最优，有时甚至效果会劣化。

下面再看看面对瑞利噪声会发生什么：



观察到EM法分级的对比度普遍高于kmeans，虽然细看噪声点也随之更明显，但同时原图的轮廓也更加好辨认，有利有弊。

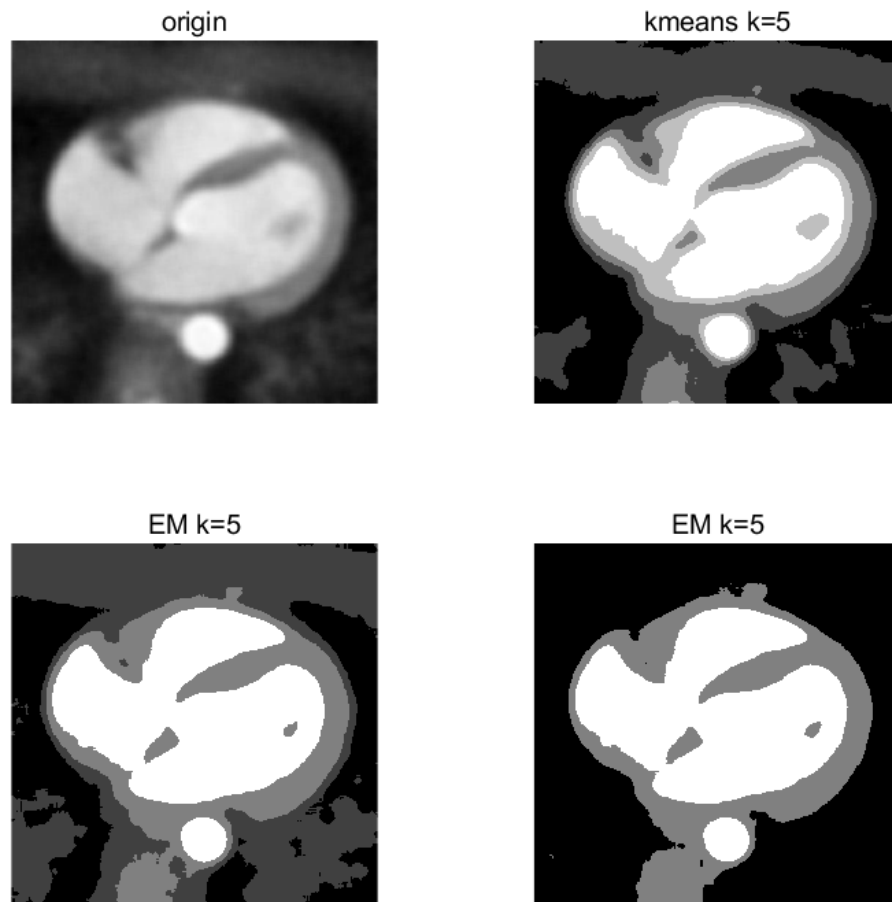
(3)

针对噪声图像，讨论为什么分割的结果不准确。用什么方法可以取得更好的分割结果。

答：

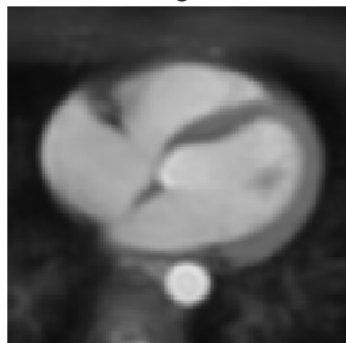
分割结果不准确其实就是图像分割操作不能很好的识别出噪声点。仔细分析，可以把噪声点看成是一阶导突增的位置，如果我们能够减缓这种突增，就能使噪声点接近未污染前状态，一方面渐小噪声图与原图聚类时的差异，另一方面也能使噪声点有更高的概率被划分到为污染前的类别。

基于此，下面我将噪音图像做了平滑处理，再进行分割，结果如下：

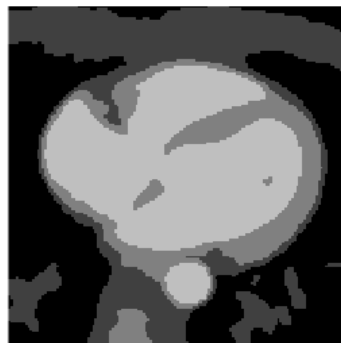


而直接对无污染原图分割结果如下：

origin



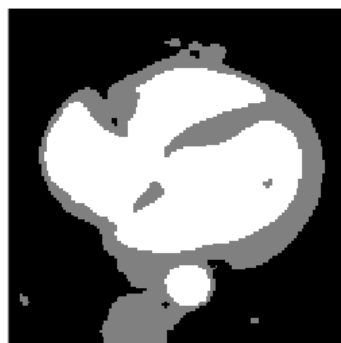
kmeans k=5



EM k=5



EM k=5



能观察到前后的右下方图几乎完全一致！说明平滑处理确实是一种有效的处理噪音的方式。