2020

数字图像处理第六次报告

学生姓名：任泽华

班级：自动化 71

学号：2171411498

提交日期：2020-4.28

摘要 ：

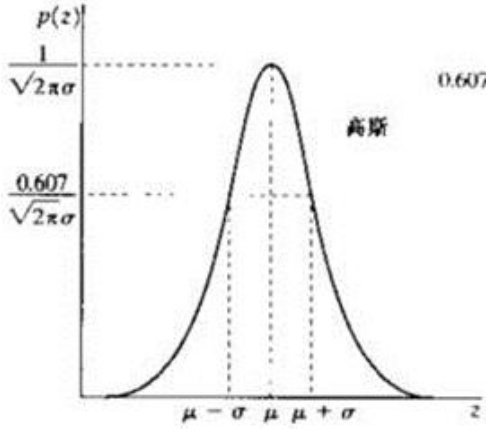
本报告主要工作：本报告软件运行环境为 MATLAB R2018b，在指定均值和方差的基础上为测试图像产生高斯噪声，用多种滤波器恢复图像，并进行了分析对比。在测试图像上加入椒盐噪声，用多种滤波器恢复图像，并进行了分析对比。得出结论：高斯噪声适用均值滤波器，椒盐噪声适用中值滤波器。推导了维纳滤波器和约束最小二乘滤波器的公式，实现了运动模糊滤波器，并分别使用逆滤波器、维纳滤波器和约束最小二乘滤波器恢复了模糊且加噪声的测试图像，对比得出结论：噪声形式已知采用维纳滤波，噪声形式未知采用约束最小二乘滤波。

本报告所有代码均为自己编写，在编写过程中主要参考了 CSDN 相关帖子与 MATLAB 官方网站。（参考文献）

一、 在测试图像上产生高斯噪声 lena 图-需能指定均值和方差；并用多种滤波器恢复图像，分析各自优缺点；

1. 高斯噪声原理

高斯噪声是指它的概率密度函数服从高斯分布（即正态分布）的一类噪声。它的概率密度服从高斯分布（正态分布）其中有 means（平均值）和 sigma（标准方差）两个参数。一个高斯随机变量 z 的 PDF 可表示为：



$$p(z) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(z-\bar{z})^2/2\sigma^2}$$

在实际实现中，对于每一个像素值为：原像素+均值+ √方差×随机数

2. 产生的不同均值方差的噪声图

在 matlab 中使用自编程序生成 lena 图的不同均值方差下高斯噪声叠加的图像。

对于均值固定为 0，分别使得方差为 200-1000

对于方差固定为 100，分别使得均值为-100-100

1.Mean=0

Original image　　var=200　　var=400

Var=600　　var=800　　var=1000

2.Var=100

mean=-100　　mean=-50　　mean=0



mean=50 mean=100

从图中可以看出，随着方差增大，图像变得模糊，随着均值的增大，图像变亮，而均值减小，图像变暗。

3. 多种滤波器原理

① 算术均值滤波器

$$\hat{f}(x,y)=\frac{1}{mn}\sum_{(s,t)\epsilon S_{xy}} g(s,t)$$

② 几何均值滤波器

$$\hat{f}(x,y)=\frac{1}{mn}\left[\prod_{(s,t)\epsilon S_{xy}} g(s,t)\right]^{\frac{1}{mn}}$$

③ 谐波均值滤波器

$$\hat{f}(x,y)=\frac{mn}{\sum_{(s,t)\epsilon S_{xy}}\frac{1}{g(s,t)}}$$

④ 逆谐波均值滤波器

$$\hat{f}(x,y)=\frac{\sum_{(s,t)\epsilon S_{xy}} g(s,t)^{\ Q+1}}{\sum_{(s,t)\epsilon S_{xy}} g(s,t)^{\ Q}}$$

⑤ 中心滤波器

$$\hat{f}(x,y)=\frac{1}{2}[min(g(s,t))+max(g(s,t))]\ \ (s,t)\epsilon S_{xy}$$

⑥ 最小值滤波

$$\hat{f}(x,y)=\ min(g(s,t))\qquad (s,t)\epsilon S_{xy}$$

⑦ 中值滤波

$$\hat{f}(x,y)=\ media(g(s,t))\qquad (s,t)\epsilon S_{xy}$$

⑧ 最大值滤波

$$\hat{f}(x,y) = max(g(s,t)) \qquad (s,t)\epsilon S_{xy}$$

⑨ alpha 修剪均值

$$\hat{f}(x,y) = \frac{1}{mn-d}\sum_{s,t)\epsilon S_{xy}} (g(s,t))$$

⑩ 自适应局部降噪

$$\hat{f}(x,y) = g(x,y) - \frac{\sigma_\eta^2}{\sigma_L^2}[g(x,y) - m_L]$$

⑪ 自适应中值

Stage A:

$A1=Z_{med}-Z_{min}$;$A2=Z_{med}-Z_{max}$

if A1>0 and A2<0,go to stage B

Else increase the window size

if window size S_{max},repeat stage A ;Else output Z_{med}

Stage B:

$B1=Z_{xy}-Z_{min}$;$B2=Z_{xy}-Z_{max}$

if B1>0 and B2<0,output Z_{xy}; Else output Z_{med}

4. 多种滤波器恢复效果 模板大小=5

Var=200　　代数均值　　几何均值

谐波均值　　反谐波均值 Q=1　　中心滤波

最小值滤波　　中值滤波　　最大值滤波

alpha 修剪均值 D=4　　自适应局部降噪　　自适应中值

5. 比较各自的优缺点

相比较而言，常规的均值滤波器，代数均值比几何均值要亮一些，因为代数均值要比几何均值大。算术均值滤波器是最简单的均值滤波器，可以去除均匀噪声和高斯噪声，但会对图像造成一定程度的模糊；与之相比几何均值滤波器能够更好的取出高斯噪声，并且能够更多的保留图像的边缘信息，但其对 0 值很敏感。而 alpha 修剪均值效果也好于普通均值法。更进一步，谐波均值要更暗而反谐波均值更亮。对于随机高斯噪声，最大最小值效果都不好，它们分别是这些滤波器中最亮和最暗的。而中值滤波器效果要好于中心滤波器。对于两种自适应滤波器，自适应局部降噪法更适用于高斯噪声，而自适应中值效果反而不好，这是因为噪声是均匀叠加在每个像素上的，用局部中值理论上并没有消除这一部分噪声。

二、 在测试图像 lena 图加入椒盐噪声（椒和盐噪声密度均是 0.1）；用学过的滤波器恢复图像；在使用反谐波分析 Q 大于 0 和小于 0 的作用；

1. 椒盐噪声原理

椒盐噪声（salt & pepper noise）是数字图像的一个常见噪声，所谓椒盐，椒就是黑，盐就是白，椒盐噪声就是在图像上随机出现黑色白色的像素。椒盐噪声是一种因为信号脉冲强度引起的噪声。椒盐噪声的 PDF 由下式给出：

$$p(z)=\begin{cases}P_a & z=a\\ P_b & z=b\\ 1-P_a-P_b & \text{其他}\end{cases}$$

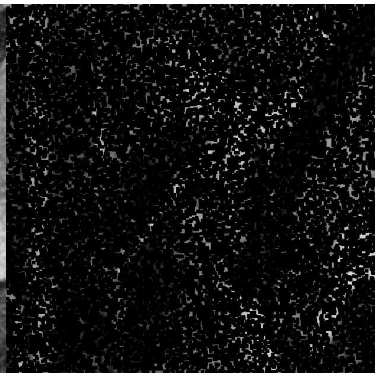
如果 $b>a$,则灰度级 b 在图像中显示为一个亮点；反之，灰度级 a 在图像中将显示为一个暗点。若P_a或P_b为零，则为单极脉冲。如果不相等的时候，尤其他们近似相等的时候，在脉冲噪声将在图像上随机产生胡椒或盐粉微粒。双极脉冲也称为椒盐噪声。此处 a=0，b=255。

原图　　ρ =0.1 的椒盐噪声图

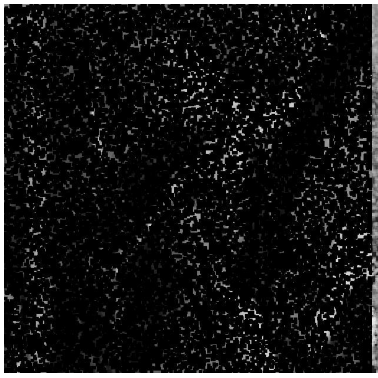
2. **多种滤波器恢复效果 模板大小**=5

椒盐噪声图

代数均值

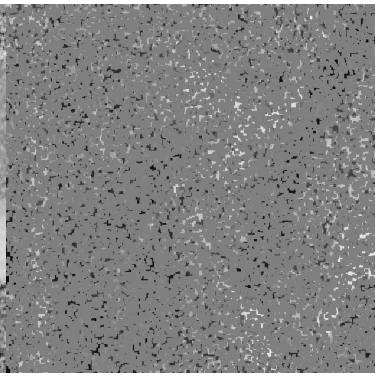


几何均值

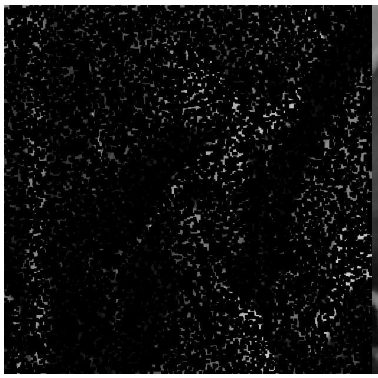


谐波均值

反谐波均值 Q=1

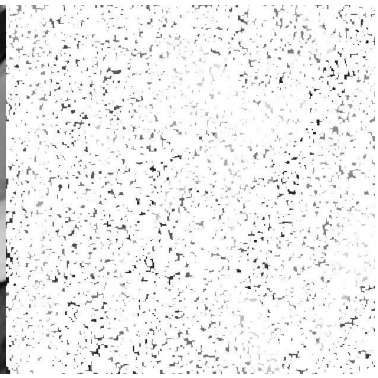


中心滤波



最小值滤波

中值滤波



最大值滤波

alpha 修剪均值 D=4　　自适应局部降噪　　自适应中值

3. 比较各自的优缺点

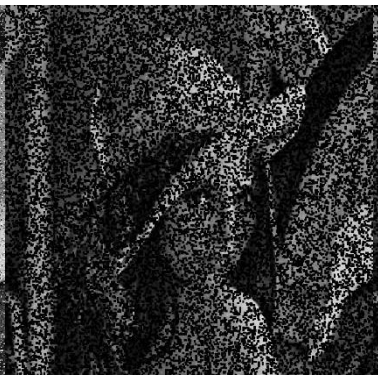
对于椒盐噪声，前面的几种滤波器大部分效果差劲，这是因为它们大部分基于局部的所有值进行的，而椒盐噪声的存在使得这种局部性被破坏，其中的噪声是影响整体的异常值。而只有中值滤波器和自适应中值滤波器效果较好。
相应地，由于普通中值滤波器每个点都是局部中值，这就造成了局部色块的现象，而自适应中值就很好地避免了这个现象。

中值滤波　　局部自适应中值

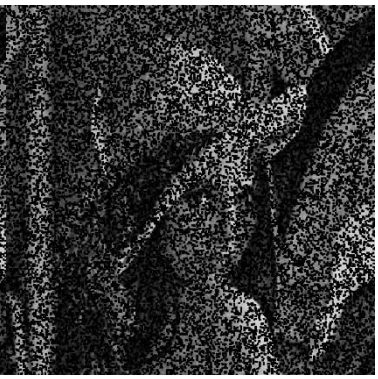
4. 分析使用反谐波分析 Q 大于 0 和小于 0 的作用

由于模板大小为 5 时反谐波滤波器过于敏感破坏了原有图片结构，所以采用大小为 3 的模板进行分析。

椒盐噪声图



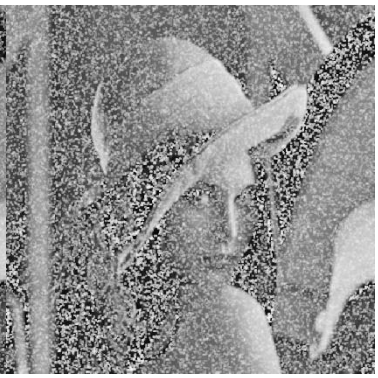
Q=-1（谐波均值）



Q=-0.5

Q=0（代数均值）

Q=0.5



Q=1

1. 当 Q 为正值时，滤波器只能消除椒噪声，

2. 当 Q 为负值时，滤波器只能消除盐噪声。

3. 当 Q=0 时，滤波器为算术均值滤波器；

4. 当 Q=-1 时，滤波器为谐波均值滤波器；

5. 结论

高斯噪声适用均值滤波器，椒盐噪声适用中值滤波器。

三、 推导维纳滤波器并实现下边要求；

6. 推导

（1）维纳滤波器

维纳滤波综合了退化函数和噪声统计特征进行复原处理的方法。该方法建立在 图像和噪声都是随机变量的基础上，目标是找到未污染图像f的一个估计$\hat{f}$ ，使它们之间的均方差误差最小。

$$e(f) = E|X(f) - \hat{X}(f)|^2$$

这里 E 是期望

如果我们替换表达式中的$\hat{X}(f)|^2$，上面可以重新组合成：

$$\begin{aligned} e(f) &= E|X(f) - G(f)Y(f)|^2 \\ &= E|X(f) - G(f)[H(f)X(f) + V(f)]|^2 \\ &= E|[1 - G(f)H(f)]X(f) - G(f)V(f)|^2 \end{aligned}$$

展开二次方，得：

$$\begin{aligned} e(f) = &[1 - G(f)H(f)][1 - G(f)H(f)]^* E|\mathrm{X(f)}|^2 \\ &-[1 - G(f)H(f)]G^*(f)E\{X(f)V^*(f)\} \\ &-G(f)[1 - G(f)H(f)]^* E\{V(f)X^*(f)\} \\ &+G(f)G^*(f)E|V(f)|^2 \end{aligned}$$

我们假设噪声与信号独立无关，这样：

$$E\{X(f)V^*(f)\} = E\{V(f)X^*(f)\} = 0$$

我们如下定义功率谱：

$$S(f) = E|X(f)|^2$$
$$N(f) = E|V(f)|^2$$

于是：

$$e(f) = [1 - G(f)H(f)][1 - G(f)H(f)]^* S(f) + G(f)G^*(f)N(f)$$

为了得到最小值，我们对 G(f)求导，令方程等于零：

$$\frac{\mathrm{d}(f)}{\mathrm{d}G(f)} = G^*(f)N(f) - H(f)[1 - G(f)H(f)]^* S(f) = 0$$

最终求得维纳滤波器为：

$$G(f) = \frac{1}{H(f)}\left[\frac{|H(f)|^2}{|H(f)|^2 + \frac{N(f)}{S(f)}}\right]$$
$$= \frac{1}{H(f)}\left[\frac{|H(f)|^2}{|H(f)|^2 + \frac{1}{SNR(f)}}\right]$$

这里 $H(f)$是 h 在频率域 f 的傅里叶变换。$SNR(f)=S(f)/N(f)$是信号噪声比。当噪声为零时（即信噪比趋近于无穷），方括号内各项也就等于 1，意味着此时刻维纳滤波也就简化成逆滤波过程。但是当噪声增加时，信噪比降低，方括号里面值也跟着降低。这说明，维纳滤波的带通频率依赖于信噪比。

在实践中可以将 $1/SNR(f)$替换成可以改变的值 k，从而可以动态地改变图像质量，选出最优效果。

（2）约束最小二乘滤波器

约束最小二乘滤波核心是 H 对噪声的敏感性问题。减少噪声敏感新问题的一种方法是以平滑度量的最佳复原为基础的，因此我们可以建立下列约束条件：

$$C = \sum_{0}^{M-1} \sum_{0}^{N-1} [\nabla^2 f(x,y)]^2$$

$$||G - H\hat{F}||_2^2 = ||N||_2^2$$

约束为：

F^是为退化图像的估计，N 为加性噪声，拉普拉斯算子∇2 在这里表示平滑程度。

将上式表示成矩阵形式，同时将约束项转换成拉格朗日乘子项：

$$||P\hat{F}||_2^2 - \lambda(||G - H\hat{F}||_2^2 - ||N||_2^2)$$

最小化上代价函数，对 F^求导，令其等于零：

$$P^*P\hat{F} = \lambda H^*(G - H\hat{F})$$

最后可得：

$$\hat{F} = \frac{\lambda H^* G}{\lambda H^* H + P^* P} = \left[\frac{H^*}{||H||_2^2 + \gamma||P||_2^2}\right] G$$

P 是 p 的傅里叶变换

$$p = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

γ 为可选参数。

1. 实现模糊滤波器如方程 Eq. (5.6-11).

使用书上的运动模糊公式：

$$\frac{T}{\pi(ua+vb)}\sin\left[\pi(ua+vb)\right]e^{-j\pi(ua+vb)}$$

代入即可求出 lena 图的运动模糊图像

Matlab 中使用如下代码即可实现：

```
[M,~] = size(I);
v=[-M/2:M/2-1];u=v';
A=repmat(a.*u,1,M)+repmat(b.*v,M,1);
H=T/pi./A.*sin(pi.*A).*exp(-1i*pi.*A);
H(A==0)=T;
```

其中 I 即为输入的图像，将得到的矩阵 H 和原图的二维 FFT 变换矩阵

相乘再做二维 IFFT 即可得到对应图像。

2. 模糊 lena 图像：45 度方向，T=1；

由于规定模糊方向为 45°，未规定向左下还是右下，于是采用左下。

故取参数 a=0.05，b=0.05（只要二者相等即可）T=1（题目要求）


原图　　45°模糊图

3. **再模糊的** lena **图像中增加高斯噪声，均值**= 0 **，方差**=10 pixels **以产生模糊图像；**

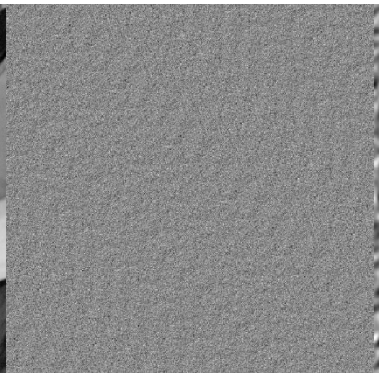
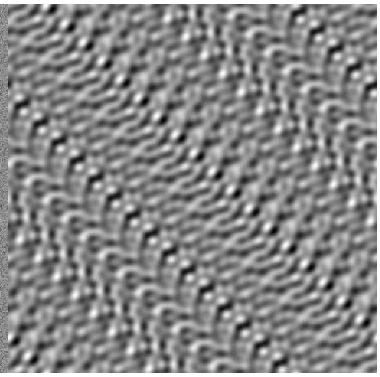
类比于前几问的方式，可是由于进行 IFFT 变换后显示图像会舍弃虚部，故在频域进行。

```
noize=sqrt(noizevar)*randn(M,M)+noizemean;
FlattenedData = noize(:)'; % 噪声归一化
MappedFlattened = mapminmax(FlattenedData, 0, 1);
noize= reshape(MappedFlattened, size(noize));
FNoise=fftshift(fft2(noise));
```


45°模糊图　　45°模糊图加噪声

4. **分别利用方程** Eq. (5.8-6)**和**(5.9-4)**，恢复图像；并分析算法的优缺点**.

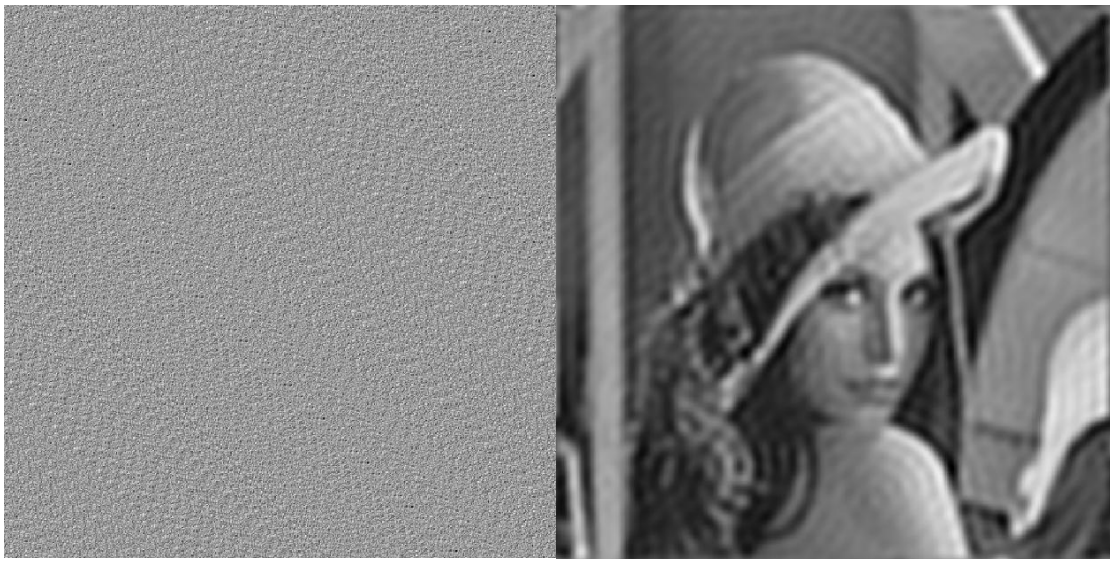
（1）逆滤波器

无噪声逆滤波　　有噪声逆滤波　　加门限逆滤波

由于 a=b=0.05 时加门限逆滤波也无法显示，令 a=b=0.02 再次尝试。

经过实验，当截至门限 d 为 33.8 时逆滤波效果最好。



完全逆滤波　　加门限逆滤波

从上图可以看出，逆滤波器效果很差，仅在完全无噪声时可以完美恢复原图像，而在有一点点噪声时就会完全失去效用。而带门限的逆滤波器在比较轻微的晃动模糊下还可以胜任，在比较剧烈的模糊下就完全不能恢复了。

（2）维纳滤波

原始定义维纳滤波　　k=0.008 维纳滤波

采用原始定义式来恢复效果不如自定义参数效果好。

实验得出 k=0.008 时约束维纳滤波的效果最好。

（3）约束最小二乘滤波

模糊噪声图　　γ=0.001 约束最小二乘滤波

实验得出 γ =0.001 时约束最小二乘滤波的效果最好。

（4）对比分析优缺点

加门限逆滤波　　维纳滤波　　约束最小二乘滤波

比较可以看出，在图像的平滑度上，维纳滤波效果更好，因为它在噪声形式已知的条件下进行，约束最小二乘虽然效果比逆滤波要好得多，但是比维纳滤波还要差一些，图片局部有很多噪点。而维纳滤波也有局限性，比如说会降低图片对比度等等。在噪声形式未知的条件下，估计出噪声的均值方差，使用约束最小二乘法反而是更好的选择。

附录

1. 参考文献

[1] Rafael C. Gonzalez（拉斐尔 C. 冈萨雷斯）,Richard E. Woods （理查德 E. 伍兹）. 数字图像处理(第三版)（英文版).北京：电子工业出版社. 2017 年.
[2] 图像去模糊（维纳滤波）CSDN
https://blog.csdn.net/bluecol/article/details/46242355
[3] 图像去模糊（约束最小二乘方滤波）CSDN
https://blog.csdn.net/bluecol/article/details/47359421?ops_request_misc=&request_id=&biz_id=102&utm_source=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-0
[4] 逆滤波和维纳滤波 CSDN
https://blog.csdn.net/weixin_41730407/article/details/80455612?ops_request_misc=%257B%2522request%255Fid%2522%253A%252215880428921919516253296 5%2522%252C%2522scm%2522%253A%252220140713.130102334.app%255Fall.%2522%257D&request_id=158804289219195162532965&biz_id=0&utm_source=distribute.pc_search_result.none-task-blog-2~all~first_rank_v2~rank_v25-1

2. 源代码

（1）生成噪声图

```
var=200;
mean=0;
k1=0.1;
k2=0.1;
img=imread('lena.bmp');
lena=double(img);
[m,n]=size(lena);
o=ones(m,n);
gauss=lena+o*mean+sqrt(var)*randn(size(lena));
figure;imshow(gauss,[0,255]);set(gca,'position',[0 0 1 1]);
```

```
set(gcf,'position',[500,280,500,500])
a1=rand(m,n)<k1;
a2=rand(m,n)<k2;
saltpepper=lena;
saltpepper(a1)=0;
saltpepper(a2)=255;
figure;imshow(saltpepper,[0,255]);set(gca,'position',[0 0 1 1]);
set(gcf,'position',[500,280,500,500])
```

（2）填充子函数

```
function pa = padding(img,modelnum) %填充函数
[r,c]=size(img);
padding=floor(modelnum/2);
pa=zeros(r+padding*2,c+padding*2);
for i=1:r        %padding 后的矩阵
    for j=1:c
        pa(i+padding,j+padding)=img(i,j);
    end
end
for i=1:padding%用原图的边缘填充 padding 矩阵的边缘
    pa(i,1+padding:c+padding)    = img(1,:);%top
    pa(i+r+padding,1+padding:c+padding)= img(r,:);%bottom
    pa(1+padding:r+padding,i)    = img(:,1);%left
    pa(1:padding,i)=pa(1+padding,i);
    pa(r+padding+1:r+2*padding,i)=pa(r+padding,i);
    pa(1+padding:r+padding,i+c+padding)= img(:,c);%right
    pa(1:padding,i+c+padding)=pa(1+padding,i+c+padding);
    pa(r+padding+1:r+2*padding,i+c+padding)=pa(r+padding,i+c+padding);
end
end
```

（3）各种滤波器集成调用函数

```
function finimg = myfilter(img,num,method,Q)
[m,n]=size(img);
```

```
padding=floor(num/2);
r=m-2*padding;c=n-2*padding;
finimg=zeros(r,c);
for i=1:r
    for j=1:c
        part=img(i:i+num-1,j:j+num-1);
        switch method
            case 1 %Arithmetic mean filter
                finimg(i,j)=sum(part(:))/num^2;
            case 2 %Geometric mean filter
                finimg(i,j)=real(prod(part(:))^(1/num/num));
            case 3 %Harmonic mean filter
                part=1./part;
                finimg(i,j)=num*num/sum(part(:));
            case 4 %Contraharmonic mean filter
                mol=part.^(Q+1);
                den=part.^Q;
                finimg(i,j)=sum(mol(:))/sum(den(:));
            case 5 %min
                finimg(i,j)=min(part(:));
            case 6 %max
                finimg(i,j)=max(part(:));
            case 7 %midian
                finimg(i,j)=median(part(:));
            case 8 %Midpoint filter
                finimg(i,j)=(min(part(:))+max(part(:)))/2;
            case 9 %Alpha-trimmed mean filter
                order=sort(part);
                finimg(i,j)=sum(order(Q/2+1:num*num-Q/2))/(num*num-Q);
            otherwise
                error('输入参数错误')
        end
    end
end
end
```

（4）两种自适应滤波器调用函数

```
function finimg = adapfilter(img,num,method,tolvar_maxnum)
if nargin==3
    tolvar_maxnum = 15;%缺省值
end
tolvar=tolvar_maxnum;
maxnum=tolvar_maxnum;
[m,n]=size(img);
pad=floor(num/2);
r=m-2*pad;c=n-2*pad;
finimg=zeros(r,c);
orgimg=img(pad+1:r+pad,pad+1:c+pad);%原图
if method==2 %自适应中值时改变 padding 矩阵
    img=padding(orgimg,maxnum);
    gap=(maxnum-num)/2; %实际模板和最大模板的差距
end
for i=1:r
    for j=1:c
        if method==1
            part=img(i:i+num-1,j:j+num-1);
        elseif method==2
            part=img(i+gap:i+num-1+gap,j+gap:j+num-1+gap);
        end
        switch method
            case 1 %Adaptive, Local Noise Reduction Filters
                locmean=sum(part(:))/num/num;
                locsqu=(part-locmean).^2;
                locvar=sum(locsqu(:))/num/num;
                finimg(i,j)=orgimg(i,j)-tolvar/locvar*(orgimg(i,j)-locmean);
            case 2 %Adaptive Median Filters
                zmin=min(part(:));
                zmax=max(part(:));
                zmed=median(part(:));
                A1=zmed-zmin;
                A2=zmed-zmax;
```

```
                    incre=1;%增加的模板圈数（基于初始模板）
                    while (~(A1>0 && A2<0))

part=img(i+gap-incre:i+num-1+gap+incre,j+gap-incre:j+num-1+gap+incre);
                        incre=incre+1;
                        if size(part,1)==maxnum
                            break
                        end
                        zmin=min(part(:));
                        zmax=max(part(:));
                        zmed=median(part(:));
                        A1=zmed-zmin;
                        A2=zmed-zmax;
                    end
                    if (A1>0 && A2<0)
                        B1=orgimg(i,j)-zmin;
                        B2=orgimg(i,j)-zmax;
                        if B1>0 && B2<0
                            finimg(i,j)=orgimg(i,j);
                        else
                            finimg(i,j)=zmed;
                        end
                    else
                        finimg(i,j)=zmed;
                    end
%                       clear part
                otherwise
                    error('输入参数错误')
            end
        end
end
end
```

（5）第一问主程序

```
clear
```

```
var=200;
mean=0;
modelnum=5;%模板大小
D=4; %修剪均值中去掉的个数
maxnum=9;%自适应中值最大模板大小
img=imread('lena.bmp');
setwindows="set(gca,'position',[0 0 1 1]);set(gcf,'position',[500,280,500,500]);";
lena=double(img);
[r,c]=size(lena);
o=ones(r,c);
gauss=lena+o*mean+sqrt(var)*randn(size(lena));
gauss(gauss<0)=0; %不加这句出现负值最后产生白点
figure;imshow(gauss,[0,255]);eval(setwindows);%模糊图像
pa=padding(gauss,modelnum);
finimg1 = myfilter(pa,modelnum,1);eval(setwindows);%代数均值
figure;imshow(finimg1,[0,255]);eval(setwindows);
finimg2 = myfilter(pa,modelnum,2);
figure;imshow(finimg2,[0,255]);eval(setwindows);%几何均值
finimg3 = myfilter(pa,modelnum,3);
figure;imshow(finimg3,[0,255]);eval(setwindows);%谐波均值
finimg4 = myfilter(pa,modelnum,4,1);
figure;imshow(finimg4,[0,255]);eval(setwindows);%反谐波均值 Q=1
finimg5 = myfilter(pa,modelnum,5);
figure;imshow(finimg5,[0,255]);eval(setwindows);%最小
finimg6 = myfilter(pa,modelnum,6);
figure;imshow(finimg6,[0,255]);eval(setwindows);%最大
finimg7 = myfilter(pa,modelnum,7);
figure;imshow(finimg7,[0,255]);eval(setwindows);%中值
finimg8 = myfilter(pa,modelnum,8);
figure;imshow(finimg8,[0,255]);eval(setwindows);%中心
finimg9 = myfilter(pa,modelnum,9,D);
figure;imshow(finimg9,[0,255]);eval(setwindows);%alpha 修剪均值
finimg10 = adapfilter(pa,modelnum,1,var);
figure;imshow(finimg10,[0,255]);eval(setwindows);%自适应局部降噪
finimg11 = adapfilter(pa,modelnum,2,maxnum);
figure;imshow(finimg11,[0,255]);eval(setwindows);%自适应中值
```

（6）第二问主程序

```
clear
k1=0.1;
k2=0.1;
modelnum=5;%模板大小
D=10; %修剪均值中去掉的个数
maxnum=23;%自适应中值最大模板大小
img=imread('lena.bmp');
setwindows="set(gca,'position',[0 0 1 1]);set(gcf,'position',[500,280,500,500]);";
lena=double(img);
[r,c]=size(lena);
o=ones(r,c);
a1=rand(r,c)<k1;
a2=rand(r,c)<k2;
saltpepper=lena;
saltpepper(a1)=0;
saltpepper(a2)=255;
figure;imshow(saltpepper,[0,255]);eval(setwindows);
pa=padding(saltpepper,modelnum);
finimg1 = myfilter(pa,modelnum,1);eval(setwindows);
figure;imshow(finimg1,[0,255]);eval(setwindows);
finimg2 = myfilter(pa,modelnum,2);
figure;imshow(finimg2,[0,255]);eval(setwindows);
finimg3 = myfilter(pa,modelnum,3);
figure;imshow(finimg3,[0,255]);eval(setwindows);
finimg4 = myfilter(pa,modelnum,4,0.1);
figure;imshow(finimg4,[0,255]);eval(setwindows);
finimg5 = myfilter(pa,modelnum,5);
figure;imshow(finimg5,[0,255]);eval(setwindows);
finimg6 = myfilter(pa,modelnum,6);
figure;imshow(finimg6,[0,255]);eval(setwindows);
finimg7 = myfilter(pa,modelnum,7);
figure;imshow(finimg7,[0,255]);eval(setwindows);
finimg8 = myfilter(pa,modelnum,8);
figure;imshow(finimg8,[0,255]);eval(setwindows);
```

```
finimg9 = myfilter(pa,modelnum,9,D);
figure;imshow(finimg9,[0,255]);eval(setwindows);
finimg10 = adapfilter(pa,modelnum,1);
figure;imshow(finimg10,[0,255]);eval(setwindows);
finimg11 = adapfilter(pa,modelnum,2,maxnum);
figure;imshow(finimg11,[0,255]);eval(setwindows);
modelnum=3;
for i=-1:0.5:1 %测试 Q 变化带来的影响
    finimg4 = myfilter(pa,modelnum,4,i);
    figure;imshow(finimg4,[0,255]);eval(setwindows);
end
```

（7）第三问

```
I=im2double(imread('lena.bmp'));% [0,1]
noizevar=10;%噪音方差均值
noizemean=0;
k1=0.008;%维纳滤波参数
k2=0.01;%最小均方滤波参数
d=33.8;%直接逆滤波半径范围
T=1;a=0.02;b=0.02;%运动模糊函数参数
setwindows='set(gca,'position',[0 0 1 1]);set(gcf,'position',[500,280,500,500]);';
[M,~] = size(I);
v=[-M/2:M/2-1];u=v';
A=repmat(a.*u,1,M)+repmat(b.*v,M,1);
H=T/pi./A.*sin(pi.*A).*exp(-1i*pi.*A);
H(A==0)=T;% replace NAN
%得运动模糊图像
F=fftshift(fft2(I));
FBlurred=F.*H;
IBlurred =real(ifft2(ifftshift(FBlurred)));
figure;imshow(IBlurred,[]);eval(setwindows);
%未加噪声直接逆滤波
FDeblurred=FBlurred./H;
IDeblurred=real(ifft2(ifftshift(FDeblurred)));
figure;imshow(IDeblurred,[]);eval(setwindows);
```

```
%加噪声
noize=sqrt(noizevar)*randn(M,M)+noizemean;
FlattenedData = noize(:)'; % 噪声归一化
MappedFlattened = mapminmax(FlattenedData, 0, 1);
noize= reshape(MappedFlattened, size(noize));
FNoise=fftshift(fft2(noise));
FBlurred_Noised=FNoise+FBlurred;
IBlurred_Noised=real(ifft2(ifftshift(FBlurred_Noised)));
figure;imshow(IBlurred_Noised,[]);eval(setwindows);
%加噪声直接逆滤波
FDeblurred0=FBlurred_Noised./H;
IDeblurred0=real(ifft2(ifftshift(FDeblurred0)));
figure;imshow(IDeblurred0,[]);eval(setwindows);
%加噪声改变滤波器截止频率
FDeblurred1=zeros(M);
for i=1:M
    for j=1:M
        if sqrt((i-M/2).^2+(j-M/2).^2)<d
            FDeblurred1(i,j)= FBlurred_Noised(i,j)./H(i,j);
        end
    end
end
IDeblurred1=real(ifft2(ifftshift(FDeblurred1)));
figure;imshow(IDeblurred1,[]);eval(setwindows);
%维纳滤波按照定义将信噪比作为参数
buf=(abs(H)).^2;
NSR=FNoise./F;
FDeblurred2=FBlurred_Noised./H.*buf./(buf+NSR);
IDeblurred2=real(ifft2(ifftshift(FDeblurred2)));
figure;imshow(IDeblurred2,[]);eval(setwindows);
%维纳滤波自选参数 k
FDeblurred3=FBlurred_Noised./H.*buf./(buf+k1);
IDeblurred3=real(ifft2(ifftshift(FDeblurred3)));
figure;imshow(IDeblurred3,[]);eval(setwindows);
%最小均方误差
P=zeros(M,M);
```

```
P(1:3,1:3)=[0 -1 0; -1 4 -1; 0 -1 0];
FP=fftshift(fft2(P));
Puf=(abs(FP)).^2;
FDeblurred4=FBlurred_Noised.*conj(H)./(buf+k2*Puf);
IDeblurred4=real(ifft2(ifftshift(FDeblurred4)));
figure;imshow(IDeblurred4,[]);eval(setwindows);
```