数学软件——短学期课程
Matlab 第三次作业

姓名：汪利军
学号：3140105707
班级：统计 1401

2016.07.11

目 录

摘 要

现实中的数字图像在数字化和传输过程中常受到成像设备与外部环境噪声干扰等影响，称为含噪图像或噪声图像。噪声的干扰会使得图像质量下降，影响图像的视觉效果以及进一步的处理。噪声可分类为加性噪声，乘性噪声和量化噪声。经典的噪声有椒盐噪声和高斯噪声。减少数字图像中噪声的过程称为图像去噪。主流的去噪方法有传统滤波方法 (如均值滤波，中值滤波等)，小波方法，偏微分方程方法。但寻找有效的图像去噪方法仍然是个严峻的挑战，许多图像去噪方法往往在对图像进行一定假设后效果会非常好，但是没有一个普适的、一般的图像去噪方法 [1]。

数字图像处理技术是随着计算机技术发展而开拓出来的一个新的应用领域，汇聚了光学、电子学、数学、摄影技术、计算机技术等学科的众多方面。它把图像转换成一个数据矩阵，在计算机上对其进行处理。可以预计，随着计算机规模和速度的大幅度提高，数字图像处理技术的发展前途和应用领域将更加广阔。传统的图像去噪恢复方法有空间域滤波和频率域滤波两类方法。传统的线性去噪方法虽然可以达到去除噪声提高图像质量的目的，但是它已不能适合更高图像质量的要求，比如说在某些后续处理当中，要求原图像要有很好的边缘信息，但是经线性滤波去噪后在去除噪声的同时也平滑模糊了图像的边缘特征。

刚刚接触图片去噪这个领域，通过老师上课介绍以及自学，采用了均值滤波、中值滤波、维纳滤波、PeronaMalik 等方法对带有噪声的 lena 图进行处理，并且将去噪后的图片与原图进行比较，同时将去噪结果与 matlab 自带的一些滤波函数处理结果进行比较。

同时采用 PSNR、二范数以及标准差等标准来衡量去噪图片的质量，并且基于这些标准，给出不同情况下最好的去噪效果图。

1 研究背景

一张灰度图片可以编码为一个元素为灰度值的矩阵, 坐标点 (i,j) 的灰度值为 $u(i,j)$。限制照片清晰度的可以分为两大类，一类是模糊 (污点), 另外一类是噪声。传统的图像去噪恢复方法有空间域滤波和频率域滤波两类方法。传统的线性去噪方法虽然可以达到去除噪声提高图像质量的目的，但是它已不能适合更高图像质量的要求，比如说在某些后续处理当中，要求原图像要有很好的边缘信息，但是经线性滤波去噪后在去除噪声的同时也平滑模糊了图像的边缘特征。变分法的引入给计算机视觉和图像图形处理领域的研究提供了一个有力的工具。1989 年，Mumford 和 Shah 提出了用有界变差函数表示灰度图像。1992 年，Rudin,Osher 和 Fatemi 等人在 Mumford 和 Shah 提出的模型基础上得到了基于全变分范数的去噪模型。全变分去噪已经成功地应用在图像恢复领域，成为偏微分方程在图像处理方面的典型。

基于泛函分析和微分几何的全变分去噪模型在去除噪声的同时能有效地保持图像边缘特征，成功地运用在许多图像复原问题中，是目前图像去噪算法中最成功的方法之一。

2 评价标准

那如何衡量去噪的效果呢，经过查阅文献，我采用了以下几种标准。

2.1 PSNR

PSNR 是用来衡量经过处理后的图片品质，PSNR 值越高则处理后的图片更接近原图片 [[2]]。PSNR 是通过均方误差 (MSE) 定义的。计算公式如下

$$\boldsymbol{MSE} = \frac{\mathbf{1}}{\boldsymbol{mn}} \sum_{\boldsymbol{i=0}}^{\boldsymbol{m-1}} \sum_{\boldsymbol{j=0}}^{\boldsymbol{n-1}} \Big[\boldsymbol{I(i,j) - K(i,j)} \Big] \tag{1}$$

则 PSNR(单位为 dB)

$$\begin{aligned} \boldsymbol{PSNR} &= \mathbf{10} \times \boldsymbol{log_{10}}\Big(\frac{\boldsymbol{MAX_I^2}}{\boldsymbol{MSE}}\Big) && (2)\\ &= \mathbf{20} \cdot \boldsymbol{log_{10}}\Big(\frac{\boldsymbol{MAX_I}}{\sqrt{\boldsymbol{MSE}}}\Big) && (3)\\ & && (4) \end{aligned}$$

注意到，此时的 MAX_I 即为图片的灰度级，即 255。

2.2 二范数

一种很自然的想法便是计算两张图片差的范数即

$$
\begin{aligned}
err &= norm(img - img0, 2) && (5)\\
&= \sum_{i=1}^{m}\sum_{j=1}^{n}\Big[I(i,j) - K(i,j)\Big] && (6)
\end{aligned}
$$

注意到，因为此时 MAX_I 为常值，所以用 PSNR 与 norm 计算实际上是一样的只是 PSNR 的算法多了一个常数，为了方便计算，所以我们直接用 norm 来衡量所得去噪图片的效果。

2.3 标准差 Sd

文献 [1] 指出，人眼能够容许一定范围内的噪声，而且实验表明当图像矩阵标准差小于 60 时人眼几乎感受不到噪声点，这里 60 是指 0 到 255 范围内的灰度值，但因为在实际计算中我们经常使用 matlab 的 im2double 函数将灰度转换为双精度，即对每个灰度值除以 255。于是转换后的矩阵当其标准差小于 0.2353 时，人眼几乎感受不到噪声点。于是，在后续试验中，我们可以根据这一指标来判断去噪后图片的效果。即当去噪后的矩阵标准差小于 0.2353，我们可以认定去噪效果非常好以至于肉眼可以忽略噪声点。

3 去噪方法

3.1 中值滤波

中值滤波法是一种非线性平滑技术，它将每一像素点的灰度值设置为该点，某领域窗口内的所有像素点灰度值的中值。中值滤波是基于排序统计理论的一种能有效抑制噪声的非线性信号处理技术，中值滤波的基本原理是把数字图像或数字序列中的一点的值用该点的一个邻域中各点值得中值代替，让周围的像素值接近的真实值，从而消除孤立的噪声点。处理方法是对每个像素点的领域中的点按照像素值的大小进行排序，生成单调的数据序列，然后对于每一点，若其偏离该中值较大，则取该中值，否则保持不变。

3.2 均值滤波

均值滤波是典型的线性滤波算法，是指在图像上对目标像素给一个模板，该模板包括了其周围的临近像素，然后用模板中的全体像素的平均值来代替像素值。

3.3 维纳滤波

维纳滤波对每一个像素点计算均值与方差

$$\mu = \frac{1}{NM} \sum_{n_1,n_2 \in \eta} a(n_1, n_2) \tag{7}$$

和

$$\sigma^2 = \frac{1}{NM} \sum_{n_1,n_2 \in \eta} a^2(n_1, n_2) - \mu^2 \tag{8}$$

其中，η 是图片矩阵中每个像素点的邻域。于是根据这些信息维纳滤波用如下估计

$$b(n_1, n_2) = \mu + \frac{\sigma^2 - \nu^2}{\sigma^2}(a(n_1, n_2) - \mu) \tag{9}$$

其中，v^2 是噪音方差。具体 wiener 滤波代码见附录

3.4 统计二维滤波

统计二维滤波可以直接通过 matlab 内置的函数 $ordfilt2(A, order, domain)$ 来实现。其中 $order$ 和总体灰度的均值有关，$domain$ 是模板大小。在具体操作过程中，为了方便编程求解，我们假设 domain 为方形区域，然后对不同参数进行比较分析，选择去噪效果最好的一组参数作为我们的去噪方法。

3.5 各向异性滤波 (AF)

根据文献 [1] 可以得到各向异性的滤波

$$AF_h u(x) = \int G_h(t) u(x + t\frac{Du(x)^{\perp}}{|Du(x)|}) dt \tag{10}$$

其中，$G_h(t) = \frac{1}{\sqrt{2\pi}h} exp\{-\frac{t^2}{2h^2}\}$ 是方差为 h^2 的一维正态分布。

3.6 PDE

将图像类比为物理过程，于是可以得到 PDE。

$$min \quad \int_{\Omega} |\nabla I|^2 d\Omega \tag{11}$$

$$\begin{cases} \frac{\partial I}{\partial t} = \triangle I \\ I(t_0) = I_0 \qquad \frac{\partial I}{\partial n} = 0 \end{cases} \tag{12}$$

3.7 PeronaMalik

$$\begin{cases} \frac{\partial I}{\partial t} = \nabla \cdot (\rho(\nabla I)\nabla I) \\ I(t_0) = I_0 \qquad \frac{\partial I}{\partial n} = 0 \end{cases} \tag{13}$$

3.8 非线性全变差

$$\begin{cases} \frac{\partial I}{\partial t} = \nabla \cdot (\frac{\nabla I}{|\nabla I|} + \lambda(I - I_0)) \\ I(t_0) = I_0 \qquad \frac{\partial I}{\partial n} = 0 \end{cases} \tag{14}$$

在实际操作中，我使用了均值滤波、中值滤波、维纳滤波、AF 滤波以及 matlab 内置的一些滤波函数。

4 实验结果

4.1 subplot 的使用

首先通过 doc subplot 命令学习了 subplot 的用法，然后编写代码得出四个子图组合成的图片 (1)

Figure 1: 四个子图

4.2 椒盐噪声的去噪

椒盐噪音中的噪声十分分散，并且噪声点的值都非常极端，要么全黑，要么全白，这时候用均值法就不能得到很好的效果，但是中值法却避开了这个问题，能对椒盐噪音进行很好的去噪。了解噪声的特点和成因，能让我们更好更快地选择去噪的方法，而不是盲目地尝试。于是我对椒盐噪声进行中值滤波处理，得到结果如下。

Figure 2: 中值滤波处理椒盐噪声

同时返回误差

```
>> testMidFilter
iteration norm(img-img0) norm(img-origin)
    1.0000     0.0041     1.4463

    2.0000          0     1.1658

After 2 iterations reach err 1.1658
medfilt2 error

ans =

    1.8939
```

Figure 3: 中值滤波处理误差

可见对于椒盐噪声，我们使用中值滤波便可以很完美地进行除噪。无论是从视觉上，还是从运算结果上看，用中值处理椒盐噪声的效果效果相当好!

4.3 高斯噪声的去噪

对高斯噪声的去噪远没有椒盐噪声那么顺利得到比较好的结果，而且这方面比较棘手，于是试验了很多方法，试图得到更好的结果。结果发现采用 AF 滤波效果最好，但误差稍微还是有点大，AF 算法的代码根据文献 [1] 中的源码改编而成。另外，根据实验结果，值得说明一点的是，在我使用的 AF 算法是引用 markus 的成果. [3], 函数调用格式为

$$sol = diffusionAnisotropic(img, varargin)$$

主要有参数 $sigma, sigmaGauss, maxIter, times$, 于是在实际操作中我主要对 $sigma$ 和 $times$ 参数进行调整，并且发现 $times$ 值不能太大，$sigma$ 应该取高斯噪声的方差，但在寻找 AF 算法最优参数时，发现如果 $sigma = 0.0001$ 的效果还会略好于当 $sigma$ 取该噪声的

标准差，因为高斯噪声是均值为零方差为 0.02(注意，matlab 中的方差与一般文献中讨论的方差不一样)，进行转换应该是 $\mathbf{0.02 \times 255 = 5.1}$，具体原因还没弄清楚。

方法	二范数	标准差
中值滤波	6.1975	0.2307
均值滤波	3.7342	0.1898
维纳滤波	3.6922	0.1890
AF 滤波 (sigma=5.1)	3.1787	0.1834
AF 滤波 (sigma=0.0001)	3.0074	0.1829

Table 2: 处理高斯噪声的几种方法

Figure 4: 处理高斯噪声的几种方法

从上述运算结果即肉眼观察，可以判断采用 AF 滤波进行去噪的效果最好。

4.4 全噪声的去噪

在全噪声的去噪中，讨论了两种情况，分别是使用单一方法处理全噪声、以及使用混合方法处理全噪声

4.4.1 单一方法处理全噪声

方法	二范数
中值滤波	6.5964
均值滤波	9.6218
维纳滤波	6.9799
统计二维滤波	83.6541
高斯滤波	9.6218
AF 滤波 (sigma=5.1)	5.7530
AF 滤波 (sigma=0.0001)	5.8173

Table 3: 单一方法处理全噪声

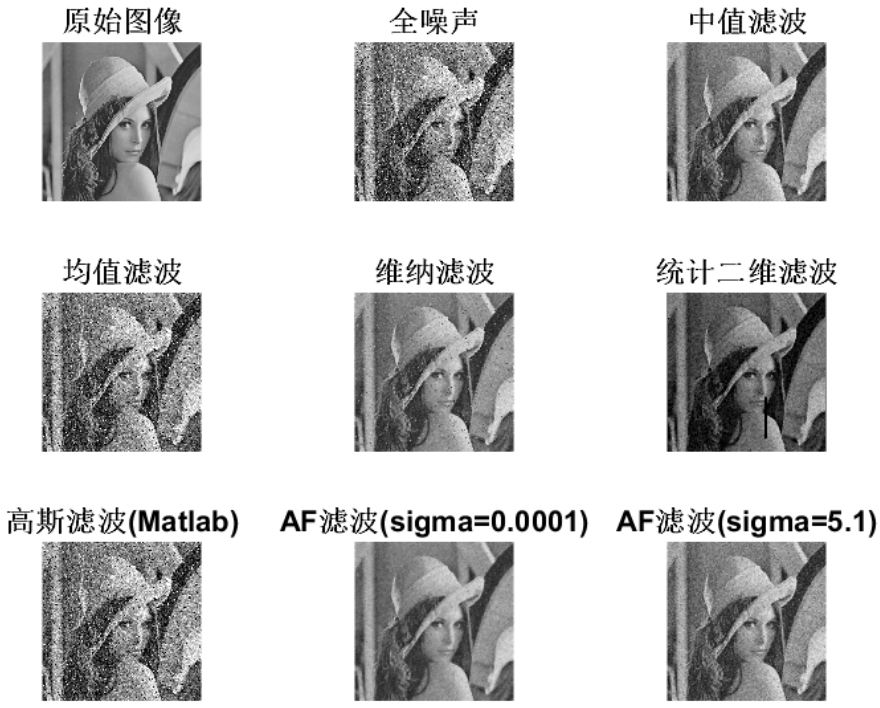

Figure 5: 单一方法处理全噪声

从上述运算结果以及肉眼观察可以判断用 AF 滤波进行去噪的效果最好。

4.4.2 混合方法处理全噪声

因为考虑到全噪声不是单一的高斯噪声，于是想组合几种方法来试试效果，但由于单一方法太多，组合多种多样，所以下面主要列出几种实验过程中效果比较好的几种组合

方法	二范数
中值滤波 + 均值滤波	6.2826
中值滤波 +AF 滤波	4.7857

Table 4: 混合方法处理全噪声

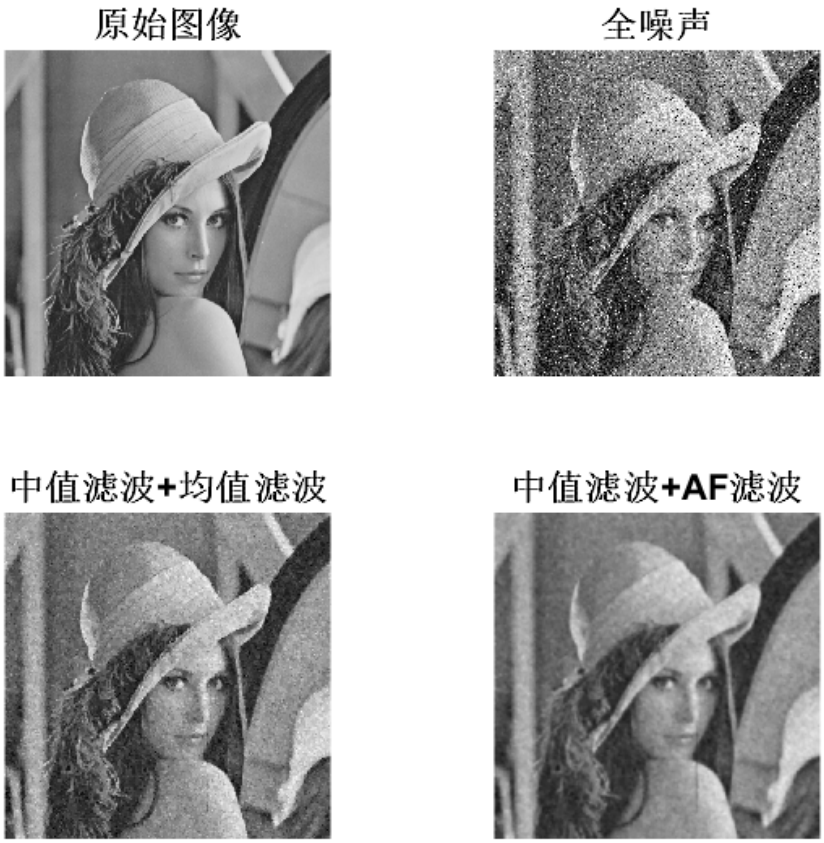

Figure 6: 混合方法处理全噪声

从上述运算结果和肉眼观察可以发现其实混合方法去噪效果也不是很好，或许也可能因为没有找到合适的组合，导致去噪效果没有单一方法去噪效果好。

5 结论

对于椒盐噪声，可以很有效地达到去噪效果，肉眼看跟原图毫无差别，此时跟原图差别二范数最低能够达到 1.1658；对于高斯噪声，尝试的那些方法去噪效果不是很好，最好的是各向异性滤波方法，跟原图偏差的二范数最低能够达到 3.0074；对于全噪声，因为有一部分是高斯噪声，所以对全噪声的去噪更加困难，尝试了一些方法，也尝试多种噪声相互结合使用，但最终效果都

不是很好，跟原图差的二范数最低能达到 4.7837.

参考文献

[1] Antoni Buades, Bartomeu Coll, Jean-Michel Morel. A review of image denoising algorithms, with a new one. SIAM Journal on Multiscale Modeling and Simulation: A SIAM Interdisciplinary Journal, 2005, 4 (2), pp.490-530. <hal-00271141>

[2] https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio

[3] http://www5.informatik.uni-erlangen.de/en/our-team/mayer-markus

附录

subplot 的代码:

```
lena512 = imread('lena512.bmp');
lena_all_noise = imread('lena_all_noise.bmp');
lena_guassian_0_02 = imread('lena_gaussian_0_02.bmp');
lena_saltpepper_05 = imread('lena_saltpepper_05.bmp');

lena512 = im2double(lena512);
lena_all_noise = im2double(lena_all_noise);
lena_guassian_0_02 = im2double(lena_guassian_0_02);
lena_saltpepper_05 = im2double(lena_saltpepper_05);
subplot(2,2,1);
imshow(lena512);
title('□’c□□');
subplot(2,2,2);
imshow(lena_all_noise);
title('Õ□□□□c□□');
subplot(2,2,3);
imshow(lena_guassian_0_02);
title('¸□'□□□□c□□');
subplot(2,2,4);
imshow(lena_saltpepper_05);
title('½·□□□□□□c□□');
```

PSNR 的计算代码:

```
function PSNR = myPSNR(origin, img)
[h,w] = size(origin);
```

```
MES = sum(sum((origin - img).^2))/(h*w);
PSNR = 20 * log10(255/sqrt(MES));
end
```

中值滤波的代码:

```
function img = myMidFilter(img0,origin)
N = size(img0, 1);
err = 100000; % a large number
threshold = 1e-3;
img = zeros(N, N);
maxIter = 10000;
errstop = 0.001;
message = ['iteration ', 'norm(img-img0) ', 'norm(img-origin)'];
disp(message);
for iter = 1:maxIter
    for i = 1:N
        for j = 1:N
            if i==1 && j==1
                tmp = sort([img0(i,j), img0(i,j+1), img0(i+1, j)]);
                img(i, j) = tmp(2);
            elseif i==1 && j==N
                tmp = sort([img0(i,j), img0(i,j-1), img0(i+1, j)]);
                img(i, j) = tmp(2);
            elseif i==N && j==1
                tmp = sort([img0(i, j), img0(i,j+1), img(i-1, j)]);
                img(i, j) = tmp(2);
            elseif i==N && j==N
                tmp = sort([img0(i,j), img0(i,j-1), img0(i-1,j)]);
                img(i, j) = tmp(2);
            elseif i==1 && j~=1 && j~=N
                tmp = sort([img0(i,j), img0(i,j+1), img0(i+1,j),
                    img0(i,j-1)]);
                img(i,j) = 0.5 * (tmp(2) + tmp(3));
            elseif i==N && j~=1 && j~=N
                tmp = sort([img0(i,j), img0(i,j+1), img0(i-1,j),
                    img0(i, j-1)]);
                img(i,j) = 0.5 * (tmp(2) + tmp(3));
            elseif j==1 && i~=1 && i~=N
                tmp = sort([img0(i,j), img0(i,j+1), img0(i-1,j),
                    img0(i, j+1)]);
                img(i,j) = 0.5 * (tmp(2) + tmp(3));
            elseif j==N && i~=1 && i~=N
                tmp = sort([img0(i,j), img0(i,j-1), img0(i-1,j),
                    img0(i+1, j)]);
                img(i,j) = 0.5 * (tmp(2) + tmp(3));
```

```
            else
                tmp = sort([img0(i,j), img0(i,j-1), img0(i-1,j),
                    img0(i+1, j), img0(i, j+1)]);
                img(i,j) = tmp(3);
            end
            if abs(img(i,j)-img0(i,j)) > threshold
                img0(i,j) = img(i,j);
            end
        end
    end
    if norm(img0 - img) < errstop || norm(img0 - img) > err
        img = img0;
        message = [iter, norm(img-img0),norm(img - origin)];
        disp(message);
        message = ['After ',num2str(iter),' iterations reach err '
            ,num2str(norm(origin - img0))];
        disp(message);
        break;
    end
    message = [iter, norm(img-img0),norm(img - origin)];
    disp(message);
    err = norm(img0 - img);
end
if iter == maxIter
    message = ['After ',num2str(iter),' iterations reach err',
        num2str((norm(origin - img))),'but it can be continue when
        you change the max iterations'];
    disp(message);
end
end
```

中值滤波的测试代码:

```
subplot(2,2,1);
imshow(lena512);
title('原始图像');
subplot(2,2,2);
imshow(lena_saltpepper_05);
title('椒盐噪声');
% K1 = filter2(fspecial('average',3), lena_saltpepper_05)/255;
myimg = myMidFilter(lena_saltpepper_05, lena512);
subplot(2,2,3);
imshow(myimg);
title('中值滤波(myself)')
matimg = medfilt2(lena_saltpepper_05);
subplot(2,2,4);
imshow(matimg);
title('中值滤波(Matlab)');
disp('medfilt2 error');
```

```
norm(matimg - lena512)
```

均值滤波的代码:

```
function img = myAverageFilter(img0, origin)
N = size(img0, 1);
err = 100000; % a large number
threshold = 1e-3;
img = zeros(N, N);
maxIter = 10000;
errstop = 0.001;
for iter = 1:maxIter
    for i = 1:N
        for j = 1:N
            if i==1 && j==1
                img(i,j) = (img0(i,j)+img0(i,j+1)+img0(i+1,j))/3;
            elseif i==1 && j==N
                img(i,j) = (img0(i,j)+img0(i,j-1)+img0(i+1,j))/3;
            elseif i==N && j==1
                img(i,j) = (img0(i,j)+img0(i,j+1)+img0(i-1,j))/3;
            elseif i==N && j==N
                img(i,j) = (img0(i,j)+img0(i,j-1)+img0(i-1,j))/3;
            elseif i==1 && j~=1 && j~=N
                img(i,j) = (img0(i,j)+img0(i,j+1)+img0(i+1,j)+img0
                    (i,j-1))/4;
            elseif i==N && j~=1 && j~=N
                img(i,j) = (img0(i,j)+img0(i,j+1)+img0(i-1,j)+img0
                    (i,j-1))/4;
            elseif j==1 && i~=1 && i~=N
                img(i,j) = (img0(i,j)+img0(i,j+1)+img0(i-1,j)+img0
                    (i+1,j))/4;
            elseif j==N && i~=1 && i~=N
                img(i,j) = (img0(i,j)+img0(i,j-1)+img0(i-1,j)+img0
                    (i+1,j))/4;
            else
                img(i,j) = (img0(i,j)+img0(i,j-1)+img0(i-1,j)+img0
                    (i+1,j)+img0(i,j+1))/5;
            end
            if abs(img(i,j)-img0(i,j)) > threshold
                img(i,j) = img0(i,j);
            end
        end
    end
    message = [iter, norm(img-img0)];
    disp(message);
    if norm(img0 - img) < errstop || norm(img0 - img) >= err
        img = img0;
        message = ['After ',num2str(iter),' iterations reach err '
            ,num2str(norm(origin - img0))];
```

```
        disp(message);
        break;
    end
    err = norm(img0 - img);
end
if iter == maxIter
    message = ['After ',num2str(iter),' iterations reach err',
       num2str((norm(origin - img))),'but it can be continue when
       you change the max iterations'];
    disp(message);
end
end
```

维纳滤波的代码:

```
function [err, img1] = mywiener(img0, origin, k)
img1 = wiener2(img0, [k k]);
err = norm(origin - img1);
end
```

AF 滤波的代码: [1]

```
function sol = diffusionAnisotropic(img, varargin)
% DIFFUSIONANISOTROPIC
% Anisotropic image diffusion as defined by Joachim Weickert. A
% comprehensive introduction into this algorithm class can be
   found in:
%
% Joachim Weickert: Anisotropic Diffuion in Image Processing, ECMI
    Series,
% Teubner-Verlag, Stuttgart, Germany, 1998, available online.
%
% The implementation contains the originally proposed anisotropic
% formulation of the Perona-Malik method, with two
% edgestoping function (Perona-Malik and Tuckey), as well as
   coherence
% enhancement diffusion.
%
% The underlying PDE is solved by the lagged diffusivity method
% (C. R. Vogel, 1996, see below) using red-black Gauss-Seidel
   iteration
% steps.
%
% Parameters:
% SOL = denoisePM(IMG, VARARGIN)
% IMG: 2D image matrix
% SOL: Resulting diffused image
% VARARGIN: Optional parameters:
%   sigma: one or more parameters for the diffusion, depending on
   the
```

```
%       edge-stopping function.
%   time: Time parameter - Amount of diffusion applied
%   function: Type of Diffusion
%       Default: 'tukey' (sigma: Standart Derivation)
%       'perona' (sigma: Standart Derivation)
%       'coherence' (sigma: 3-Array: 1 = Standart Derivation,
%                                    2 = alpha in between [0,1],
%                                    3 = Diffusion weight)
%       The corresponding 'norm' options normalize the diffusion to the
%       largest structure tensor eigenvalue on the image. Faster
%       convertion and slightly different results (sigma dependency!) have
%       to be expected.
%   sigmaGauss: Standartderivation parameter(s) for pre- and postsmoothing
%       for gradient calculation. Either scalar or 2-Array
%       Default: 1
%   maxIter: Max. Iterations, Default: 200
%   initialSolution: Initial Solution
%
% The lagged diffusitivy solution of the PDE was proposed in:
% C. R. Vogel, M. E. Oman: Iterative Methods for Total Variation Denoising,
% SIAM Journal on Scientific Computing, 17(1), 1996, 227?238.
%
% Further discussion on the method can be found in:
% T. Chan, P. Mulet: On the convergence of the lagged diffusivity fixed
% point method in total variation image restoration,
% SIAM journal on numerical analysis, 36(2), 1999, 354?367.
%
% Implementation by Markus Mayer, Pattern Recognition Lab,
% University of Erlangen-Nuremberg, 2008
% This version of the Code was NOT revised, therefore use it with caution -
% if you'll find any bugs, please tell us!
%
% You may use this code as you want. I would be grateful if you would go to
% my homepage look for articles that you find worth citing in your next
% publication:
% http://www5.informatik.uni-erlangen.de/en/our-team/mayer-markus
% Thanks, Markus

Params.edgeStopFunction = 'tukey';
Params.sigma = 20;
Params.sigmaGauss = 1;
```

```
65  Params.maxIter = 200;
66  Params.time = 3;
67  Params.initialSolution = [];
68
69  % Read Optional Parameters
70  if (~isempty(varargin) && iscell(varargin{1}))
71      varargin = varargin{1};
72  end
73
74  for k = 1:2:length(varargin)
75      if (strcmp(varargin{k}, 'sigma'))
76          Params.sigma = varargin{k+1};
77      elseif (strcmp(varargin{k}, 'sigmaGauss'))
78          Params.sigmaGauss = varargin{k+1};
79      elseif (strcmp(varargin{k}, 'time'))
80          Params.time = varargin{k+1};
81      elseif (strcmp(varargin{k}, 'maxIter'))
82          Params.maxIter = varargin{k+1};
83      elseif (strcmp(varargin{k}, 'function'))
84          Params.edgeStopFunction = varargin{k+1};
85      elseif (strcmp(varargin{k}, 'initialSolution'))
86          Params.initialSolution = varargin{k+1};
87      end
88  end
89
90
91  if strcmp(Params.edgeStopFunction, 'coherence') || strcmp(Params.
        edgeStopFunction, 'normcoherence')
92      if size(Params.sigma,2) == 1
93          Params.sigma(2) = 0.001;
94          Params.sigma(3) = 1;
95      elseif size(Params.sigma,2) == 2
96          Params.sigma(3) = 1;
97      end
98  end
99
100 if strcmp(Params.edgeStopFunction, 'tukcoherence')
101     if size(Params.sigma,2) == 1
102         Params.sigma(2) = 0.01;
103         Params.sigma(3) = 0.2;
104     elseif size(Params.sigma,2) == 2
105         Params.sigma(3) = 0.2;
106     end
107 end
108
109 % Add a 1-border to the image to avoid boundary problems
110 img = [img(:,1), img, img(:,size(img,2))];
111 img = vertcat(img(1,:), img, img(size(img,1),:));
112
```

```
113 iter = 0; % Iteration counter
114
115 if numel(Params.initialSolution) == 0
116     sol = img; %Solution initialisation: original image
117 else
118     sol = [Params.initialSolution(:,1), Params.initialSolution,
        ...
119             Params.initialSolution(:,size(Params.initialSolution,2)
                )];
120     sol = vertcat(sol(1,:), sol, sol(size(sol,1),:));
121 end
122
123 % Preparing stencil matrices
124 stencilN = zeros(size(img, 1), size(img, 2));
125 stencilS = zeros(size(img, 1), size(img, 2));
126 stencilE = zeros(size(img, 1), size(img, 2));
127 stencilW = zeros(size(img, 1), size(img, 2));
128
129 stencilCO = zeros(size(img, 1), size(img, 2));
130 stencilM = zeros(size(img, 1), size(img, 2));
131
132 resimg = img;
133 resold = 1e+30; % old residual
134 resarr = [1e+30 1e+30 1e+30]; % array of the last 3 residual
    changes
135
136 % Stoping criteria: No further improvement over 3 iterations
137 % or max. iteration limit reached
138 while (sum(resarr) > 0) && (iter < Params.maxIter);
139     % Calculation of the edge-stoping function
140     if size(Params.sigmaGauss, 2) == 2
141         [Gx2, Gxy, Gy2] = structureTensor(sol, Params.sigmaGauss
            (1), Params.sigmaGauss(2));
142     else
143         [Gx2, Gxy, Gy2] = structureTensor(sol, Params.sigmaGauss
            (1), 1);
144     end
145
146     [Dy2, Dxy, Dx2] = diffusionTensor(Gx2, Gxy, Gy2, Params);
147
148     % Bringing in the timefactor
149     Dx2 =  (Dx2 ) * Params.time;
150     Dy2 =  (Dy2 ) * Params.time;
151     Dxy =  Dxy * Params.time * 2; % Weighting of diagonal
        diffusion
152
153     % stencil computation
154     stencilN(2:end-1, 2:end-1) = (Dx2(2:end-1, 2:end-1) + Dx2(1:
        end-2, 2:end-1))/2;
```

```
155     stencilS(2:end-1, 2:end-1) = (Dx2(2:end-1, 2:end-1) + Dx2(3:
            end, 2:end-1))/2;
156     stencilE(2:end-1, 2:end-1) = (Dy2(2:end-1, 2:end-1) + Dy2(2:
            end-1, 3:end))/2;
157     stencilW(2:end-1, 2:end-1) = (Dy2(2:end-1, 2:end-1) + Dy2(2:
            end-1, 1:end-2))/2;
158
159     stencilCO =  0.25 * Dxy; % One stencil for all corners
160
161     stencilM = stencilN + stencilS + stencilE + stencilW + 1; %
            Center
162
163     % Solution computation: R/B Gauss Seidel
164     sol(2:2:end-1, 2:2:end-1) = (img(2:2:end-1, 2:2:end-1) ...
165         + (stencilN(2:2:end-1, 2:2:end-1) .* sol(1:2:end-2, 2:2:
                end-1) ...
166         + stencilS(2:2:end-1, 2:2:end-1) .* sol(3:2:end, 2:2:end
                -1) ...
167         + stencilE(2:2:end-1, 2:2:end-1) .* sol(2:2:end-1, 3:2:end
                )...
168         + stencilW(2:2:end-1, 2:2:end-1) .* sol(2:2:end-1, 1:2:end
                -2) ...
169         - stencilCO(2:2:end-1, 2:2:end-1) .* sol(1:2:end-2, 3:2:
                end) ...
170         + stencilCO(2:2:end-1, 2:2:end-1) .* sol(1:2:end-2, 1:2:
                end-2) ...
171         - stencilCO(2:2:end-1, 2:2:end-1) .* sol(3:2:end, 1:2:end
                -2) ...
172         + stencilCO(2:2:end-1, 2:2:end-1) .* sol(3:2:end, 3:2:end)
                 )) ...
173         ./ stencilM(2:2:end-1, 2:2:end-1);
174
175     sol(3:2:end, 3:2:end) = (img(3:2:end, 3:2:end) ...
176         + (stencilN(3:2:end, 3:2:end) .* sol(2:2:end-1, 3:2:end)
                ...
177         + stencilS(3:2:end, 3:2:end) .* sol(4:2:end, 3:2:end) ...
178         + stencilE(3:2:end, 3:2:end) .* sol(3:2:end, 4:2:end) ...
179         + stencilW(3:2:end, 3:2:end) .* sol(3:2:end, 2:2:end-1)
                ...
180         - stencilCO(3:2:end, 3:2:end) .* sol(2:2:end-1, 4:2:end)
                ...
181         + stencilCO(3:2:end, 3:2:end) .* sol(2:2:end-1, 2:2:end-1)
                 ...
182         - stencilCO(3:2:end, 3:2:end) .* sol(4:2:end, 2:2:end-1)
                ...
183         + stencilCO(3:2:end, 3:2:end) .* sol(4:2:end, 4:2:end) ))
                ...
184         ./ stencilM(3:2:end, 3:2:end);
185
```

```
186     sol(2:2:end-1, 3:2:end) = (img(2:2:end-1, 3:2:end) ...
187         + (stencilN(2:2:end-1, 3:2:end) .* sol(1:2:end-2, 3:2:end)
                ...
188         + stencilS(2:2:end-1, 3:2:end) .* sol(3:2:end, 3:2:end)
                ...
189         + stencilE(2:2:end-1, 3:2:end) .* sol(2:2:end-1, 4:2:end)
                ...
190         + stencilW(2:2:end-1, 3:2:end) .* sol(2:2:end-1, 2:2:end
                -1) ...
191         - stencilCO(2:2:end-1, 3:2:end) .* sol(1:2:end-2, 4:2:end)
                ...
192         + stencilCO(2:2:end-1, 3:2:end) .* sol(1:2:end-2, 2:2:end
                -1) ...
193         - stencilCO(2:2:end-1, 3:2:end) .* sol(3:2:end, 2:2:end-1)
                ...
194         + stencilCO(2:2:end-1, 3:2:end) .* sol(3:2:end, 4:2:end) )
                ) ...
195         ./ stencilM(2:2:end-1, 3:2:end);
196
197     sol(3:2:end, 2:2:end-1) = (img(3:2:end, 2:2:end-1) ...
198         + (stencilN(3:2:end, 2:2:end-1) .* sol(2:2:end-1, 2:2:end
                -1) ...
199         + stencilS(3:2:end, 2:2:end-1) .* sol(4:2:end, 2:2:end-1)
                ...
200         + stencilE(3:2:end, 2:2:end-1) .* sol(3:2:end, 3:2:end)
                ...
201         + stencilW(3:2:end, 2:2:end-1) .* sol(3:2:end, 1:2:end-2)
                ...
202         - stencilCO(3:2:end, 2:2:end-1) .* sol(2:2:end-1, 3:2:end)
                ...
203         + stencilCO(3:2:end, 2:2:end-1) .* sol(2:2:end-1, 1:2:end
                -2) ...
204         - stencilCO(3:2:end, 2:2:end-1) .* sol(4:2:end, 1:2:end-2)
                ...
205         + stencilCO(3:2:end, 2:2:end-1) .* sol(4:2:end, 3:2:end) )
                ) ...
206         ./ stencilM(3:2:end, 2:2:end-1);
207
208     % Residual computation
209     resimg(2:end-1, 2:end-1) = (-(stencilN(2:end-1, 2:end-1) .*
            sol(1:end-2, 2:end-1) ...
210         + stencilS(2:end-1, 2:end-1) .* sol(3:end , 2:end-1) ...
211         + stencilE(2:end-1, 2:end-1) .* sol(2:end-1, 3:end) ...
212         + stencilW(2:end-1, 2:end-1) .* sol(2:end-1, 1:end-2) ...
213         - stencilCO(2:end-1, 2:end-1) .* sol(1:end-2, 3:end)  ...
214         + stencilCO(2:end-1, 2:end-1) .* sol(1:end-2, 1:end-2) ...
215         - stencilCO(2:end-1, 2:end-1) .* sol(3:end,   1:end-2) ...
216         + stencilCO(2:end-1, 2:end-1) .* sol(3:end, 3:end) )) ...
217         + stencilM(2:end-1, 2:end-1) .* sol(2:end-1, 2:end-1) -
```

```
            img(2:end-1, 2:end-1);

    res = sum(sum(real(resimg .^ 2)));

    resdiff = resold - res;
    resold = res;
    resarr = [resdiff resarr(1, 1:(size(resarr, 2)-1))];

    % Duplicate edges as new borders
    sol = [sol(:,2), sol(:, 2:end-1), sol(:,end-1)];
    sol = vertcat(sol(2,:), sol(2:end-1, :), sol(end-1,:));

    iter = iter + 1;
end

% Remove border
sol = sol(2:(size(sol,1)-1), 2:(size(sol,2)-1));

end

%
    ---------------------------------------------------------------------------

function [Gx2, Gxy, Gy2] = structureTensor(img, sigmaPre,
    sigmaPost)
% StructureTensor: Calculate the structure tensor

gauss1 = fspecial('gaussian', round(sigmaPre * sigmaPre + 1) ,
    sigmaPre);
smoothimg = imfilter(img, gauss1, 'symmetric');

[Gx, Gy] = gradient(smoothimg);
Gx2 = Gx .^ 2;
Gxy = Gx .* Gy;
Gy2 = Gy .^ 2;

gauss2 = fspecial('gaussian', round(sigmaPost * sigmaPost + 1) ,
    sigmaPost);
Gx2 = imfilter(Gx2, gauss2, 'symmetric');
Gy2 = imfilter(Gy2, gauss2, 'symmetric');
Gxy = imfilter(Gxy, gauss2, 'symmetric');

end

%
    ---------------------------------------------------------------------------

function [Dx2, Dxy, Dy2, kappa] = diffusionTensor(Gx2, Gxy, Gy2,
    Params)
```

```
% DiffusionTensor: Calculate Diff.Tensor out of structure tensor and
%                   parameters

% Eigenvalue calculation
temp = sqrt(((Gx2 - Gy2) .^ 2) + 4 * (Gxy .^ 2));
temp = real(temp);

lambda1 = (Gx2 + Gy2 + temp) * 0.5;
lambda2 = (Gx2 + Gy2 - temp) * 0.5;

% Eigenvector calculation
teta = 0.5 * atan2(- 2 * Gxy, Gy2 - Gx2);
cosT = cos(teta);
sinT = sin(teta);

v1x = cosT;
v1y = sinT;
v2x = -sinT;
v2y = cosT;

% Different edge-stoppers
if strcmp(Params.edgeStopFunction, 'tuckey')
    lambda1 = zeros(size(lambda2, 1), size(lambda2, 2));
    lambda1 = lambda1 + 1;
    lambda2 = tukeyEdgeStop(lambda2, Params.sigma);
elseif strcmp(Params.edgeStopFunction, 'normtuckey')
    lambda1 = zeros(size(lambda2, 1), size(lambda2, 2));
    lambda1 = lambda1 + 1;
    lambda2 = lambda2 ./ (max(max(lambda2)));
    lambda2 = tukeyEdgeStop(lambda2, Params.sigma);
    lambda2 = lambda2 ./ (max(max(lambda2)));
elseif strcmp(Params.edgeStopFunction, 'perona')
    lambda1 = zeros(size(lambda2, 1), size(lambda2, 2));
    lambda1 = lambda1 + 1;
    lambda2 = lambda2 ./ (max(max(lambda2)));
    lambda2 = peronaEdgeStop(lambda2, Params.sigma);
elseif strcmp(Params.edgeStopFunction, 'coherence')
    kappa = (lambda1 - lambda2) .^ (2 * Params.sigma(3));
    kappa(kappa <= 0) = 1e-20;
    lambda1 = Params.sigma(2) + (1 - Params.sigma(2)) * exp(-
        Params.sigma(1) ./ kappa);
    lambda2 = Params.sigma(2);
elseif strcmp(Params.edgeStopFunction, 'normcoherence')
    kappa = (lambda1 - lambda2) .^ (2 * Params.sigma(3));
    kappa = kappa ./ max(max(kappa));
    kappa(kappa <= 0) = 1e-20;
    lambda1 = Params.sigma(2) + (1 - Params.sigma(2)) * exp(-
        Params.sigma(1) ./ kappa);
```

```
304     lambda1 = lambda1 ./ max(max(lambda1));
305     lambda2 = Params.sigma(2);
306 elseif strcmp(Params.edgeStopFunction, 'tukcoherence')
307     kappa = (lambda1 - lambda2) .^ 2;
308     kappa = kappa ./ max(max(kappa));
309     kappa(kappa <= 0) = 1e-20;
310     lambda1 = (1 - Params.sigma(3)) * exp(- Params.sigma(1) ./
            kappa) ;
311     lambda1 = lambda1 ./ max(max(lambda1));
312 
313     lambda2 = lambda2 ./ (max(max(lambda2)));
314     lambda2 = Params.sigma(3) .* tukeyEdgeStop(lambda2, Params.
            sigma(2)) ;
315     lambda2 = lambda2 ./ (max(max(lambda2)));
316 end
317 
318 % Diffusion Tensor out of new eigenvalues and the eigenvectors
319 Dx2 = lambda1 .* (v1x .^ 2) + lambda2 .* (v2x .^ 2);
320 Dxy = lambda1 .* v1x .* v1y + lambda2 .* v2x .* v2y;
321 Dy2 = lambda1 .* (v1y .^ 2) + lambda2 .* (v2y .^ 2);
322 
323 end
324 
325 %
        ------------------------------------------------------------------------

326 % Tuckey edge-stoping function
327 function lambda2 = tukeyEdgeStop(lambda, sigma)
328 lambda2 = 1 - (lambda ./ sigma) .^ 2;
329 lambda2(lambda2 < 0) = 0;
330 lambda2 = lambda2 .* lambda2;
331 
332 end
333 
334 %
        ------------------------------------------------------------------------

335 % Perona-Malik edge-stoping function
336 function lambda2 = peronaEdgeStop(lambda, sigma)
337 lambda2 = exp(- lambda .* lambda / (2 * sigma * sigma));
338 lambda2 = lambda2 ./ (max(max(lambda2)));
339 
340 end
```