西安交通大学

XI'AN JIAOTONG UNIVERSITY

数字图像处理实验报告

项目名称：图像的噪声恢复

姓名： XXX

班级： XXXXXXX

学号： XXXXXXXXXX

摘　要

图像的噪声恢复是根据图像的损失函数和噪声的性质，对原始图像进行重建的过程，其应用十分广泛，相较于图像增强，图像的噪声恢复是一个相对客观的过程。

本文基于 Python 的 Scikit-image、OpenCV、Pillow 和 SciPy 等图像和信号处理有关组件，对图像分别添加了高斯噪声和椒盐噪声，并对图像进行了多种方式的滤波重建；除此以外，本文还对图像添加运动模糊后对图像进行了维纳滤波处理。

实验表明，不同的滤波方式对不同噪声的处理结果有较大差异，应适当选择滤波器，以达到对图像的合理恢复效果。

一 题目 1

1.1 题目

在 lena 测试图像上产生高斯噪声，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。

1.2 解答

1.2.1 添加高斯噪声

在这一问和第二问中，图像添加噪声的方式可在空域用以下公式表示：

$$g(x,y)=f(x,y)+\eta(x,y)$$

其中$\eta(x,y)$为添加的噪声项。

高斯噪声的概率分布函数可以表示为

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

其中z表示图像灰度，$\bar{z}$为z的均值，σ为其标准差，而σ^2则被称为方差。

以下使用 scikit-image 中的 random_noise()函数指定添加高斯噪声，并设置均值为 0.1，方差为 0.04，其结果对比如下图所示。

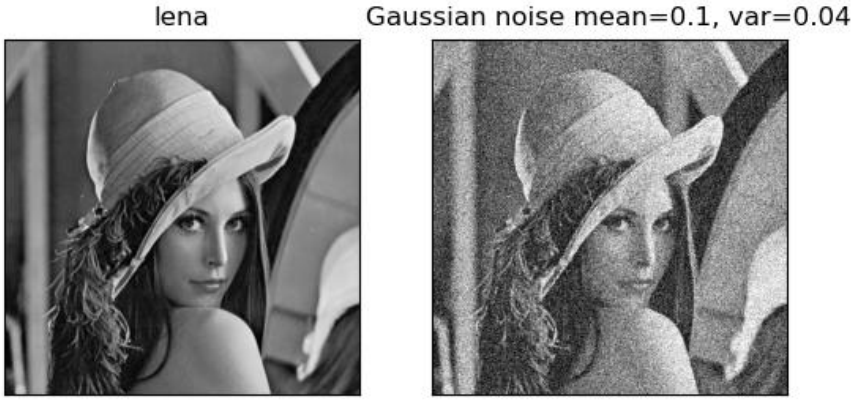

图1　添加高斯噪声（均值 0.1，方差 0.04）

注意到高斯噪声的效果与均值和方差这两个参数有着极为重要的关系，当均值增大时，图像越亮，更偏白色；而方差越大，图像则越模糊，颗粒感更重。

如下两图分别展示了当增加高斯噪声的均值和方差时结果的不同。

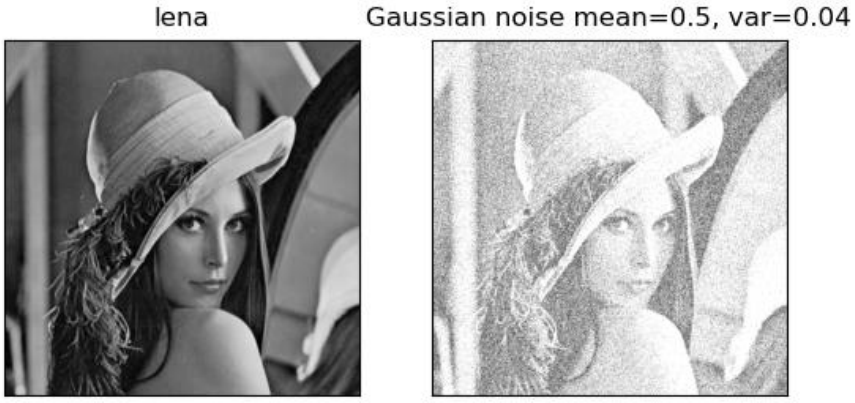

图2　添加高斯噪声（均值 0.5，方差 0.04）

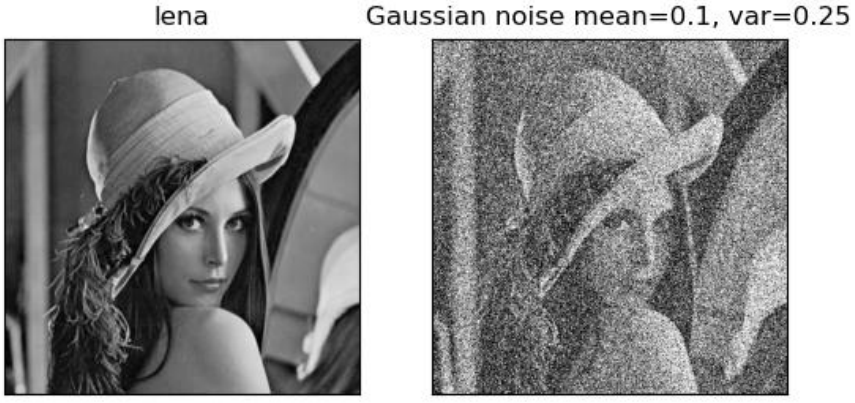

图3　添加高斯噪声（均值 0.1，方差 0.25）

1.2.2 滤波

以下为使用高斯空域滤波器对添加高斯噪声的图像滤波的结果。

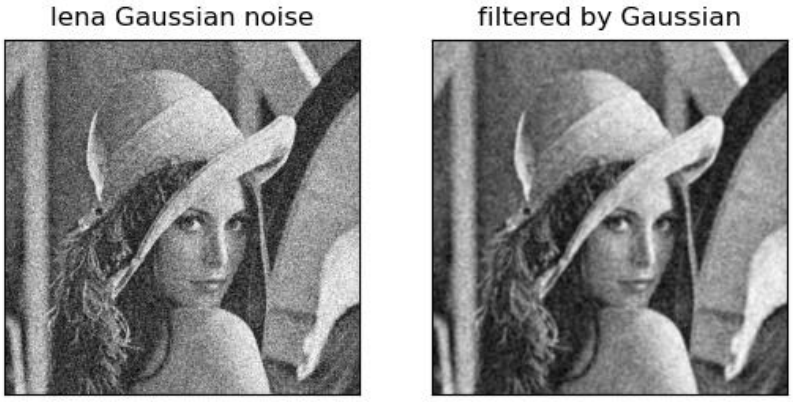

图4　高斯空域滤波器处理高斯噪声

以下为使用算术平均滤波器对高斯噪声的滤波结果。

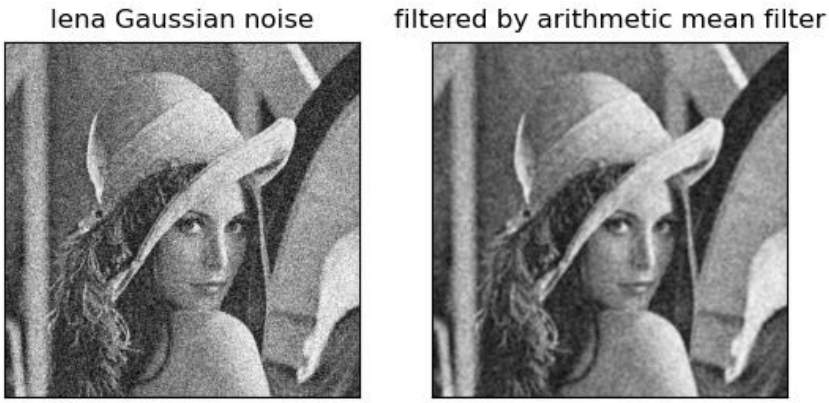

图5　算术平均滤波器处理高斯噪声

以下为使用中值滤波器对添加高斯噪声的图片滤波的结果。

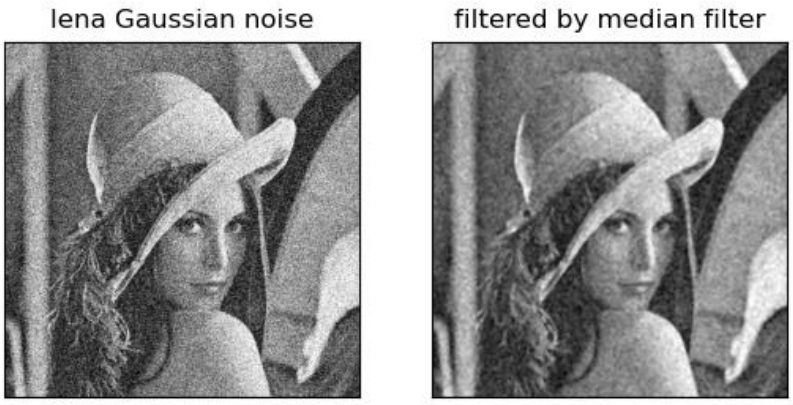

图6　中值滤波器处理高斯噪声

综上可见，高斯滤波器处理高斯噪声的能力略优于算术平均滤波器和中值滤波器，但其缺点是对图像产生了模糊效果，可能需要后续锐化处理。

注意到对于几何均值滤波器有如下滤波函数

$$\hat{f}(x,y)=\sqrt[mn]{\prod_{(s,t)\in S_{xy}} g(s,t)}$$

又根据对数函数的性质：

$$\ln\left((x_1\cdot x_2\cdot\cdots\cdot x_n)^{\frac{1}{n}}\right)=\frac{\ln x_1+\ln x_2+\cdots+\ln x_n}{n}$$

于是可以利用 OpenCV 的 boxFilter()函数，结合对数函数实现几何均值滤波器，以下为使用几何平均滤波器实现的结果。

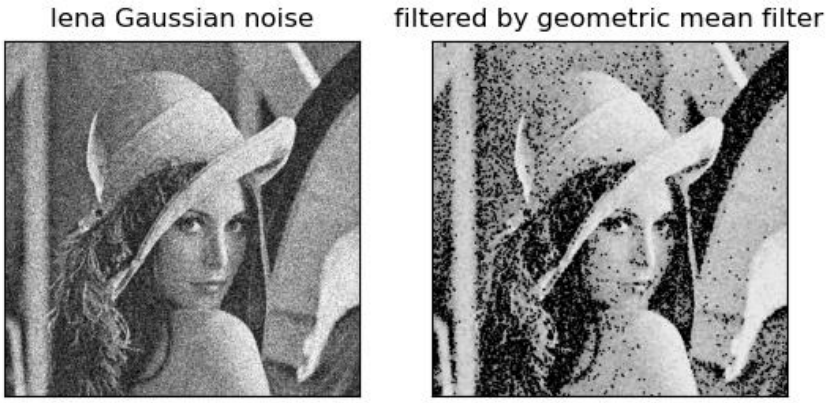

图7　几何均值滤波器处理高斯噪声

可见，几何均值滤波器对图像的处理效果并不是很好，因为在几何均值的计算公式中，存在部分 kernel 中的灰度值为 0，使得该 kernel 的计算结果为 0，因此图像的黑色噪点更为突出。

二 题目 2

2.1 题目

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，再使用反谐波分析 Q 大于 0 和小于 0 的作用。

2.2 解答

如果k代表用于表示数字图像中的强度值的比特数，那么该图像可能的强度值范围是$[0,\ 2^k-1]$，椒盐噪声的概率密度函数由下式给出：

$$p(z)=\begin{cases}P_s & \text{for } z=2^k-1\\ P_p & \text{for } z=0\\ 1-(P_s+P_p) & \text{for } z=V\end{cases}$$

其中V为满足$0<V<2^k-1$的整数值。（以上公式出自教材第四版，有别于第三版中的公式）

一个像素被盐或胡椒噪声破坏的概率为$P=P_s+P_p$，我们称P为噪声密度。例如，如果$P_s=0.02,\ P_p=0.01$，那么$P=0.03$，我们说图像中大约有 2%的像素被盐噪声破坏，1%被胡椒噪声破坏，其噪声密度为 3%，这意味着图像中大约有 3%的像素被椒盐噪声破坏。

在本问中，我们加入$P_s=P_p=0.1$的椒盐噪声，注意在 scikit-image 的 random_noise() 函数中，amount 参数代表在范围$[0,1]$上用噪声代替的图像像素的比例，即为P，而 salt_vs_pepper 参数代表在$[0,1]$范围内的盐与胡椒噪声的比例，默认值为 0.5，代表盐与胡椒噪声有相同比例。对于本题而言，我们设 amount 为 0.2，salt_vs_pepper 为 0.5，其结果如下图所示。

图8　添加椒盐噪声

2.2.1 滤波

使用算术均值滤波器滤波结果如下图所示。

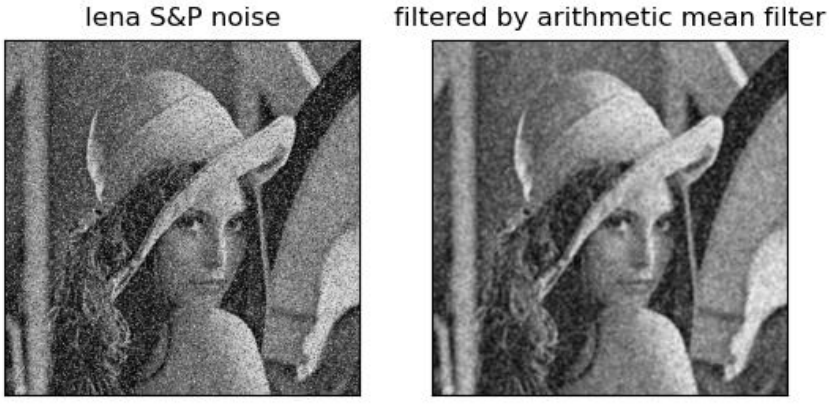

图9　算术平均滤波器处理椒盐噪声

使用中值滤波器处理椒盐噪声结果如下图所示。

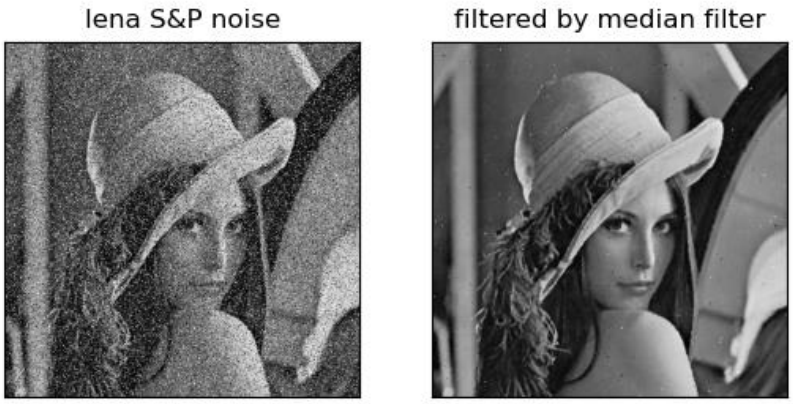

图10　中值滤波器处理椒盐噪声

由上图可见，使用中值滤波器能够最为优秀地处理椒盐噪声。

以下两图为分别使用最大值滤波器和最小值滤波器的滤波结果。

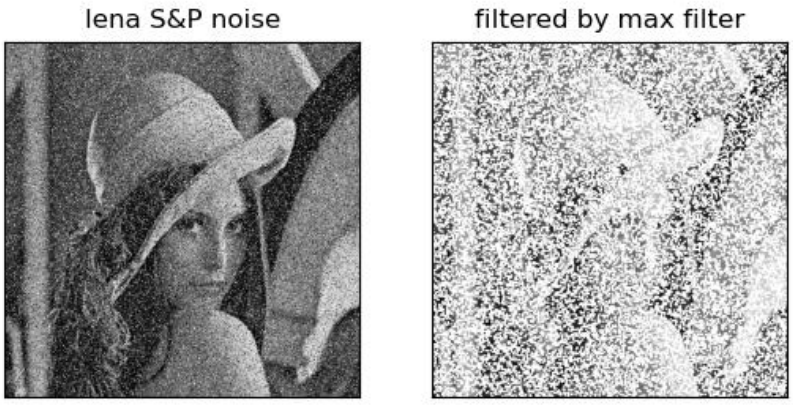

图11　最大值滤波器处理椒盐噪声

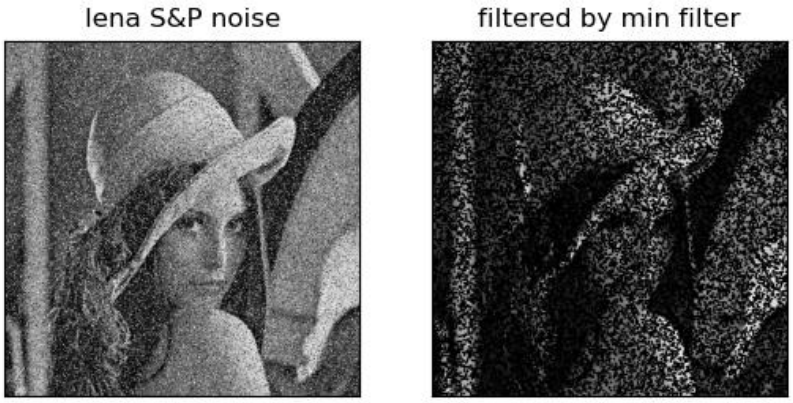

图12　最小值滤波器处理椒盐噪声

可见，使用最小值和最大值滤波器仅能处理盐和胡椒噪声中的一种，处理椒盐噪声的结果十分不理想，远不如使用中值滤波器的效果。

对于反谐波滤波器，其具有如下滤波函数：

$$\hat{f}(x,y)=\frac{\sum_{(r,c)\in S_{xy}} g(r,c)^{Q+1}}{\sum_{(r,c)\in S_{xy}} g(r,c)^{Q}}$$

其中Q为滤波器的阶数。特别地，当$Q=0$时，该滤波器即为算术平均滤波器：

$$\hat{f}(x,y)=\frac{\sum_{(r,c)\in S_{xy}} g(r,c)}{mn}$$

而当$Q=-1$时，该滤波器即为调和平均滤波器：

$$\hat{f}(x,y)=\frac{mn}{\sum_{(r,c)\in S_{xy}} \frac{1}{g(r,c)}}$$

下图分别列出了当$Q=0.5,0,-0.5$时，反谐波滤波器的处理结果。

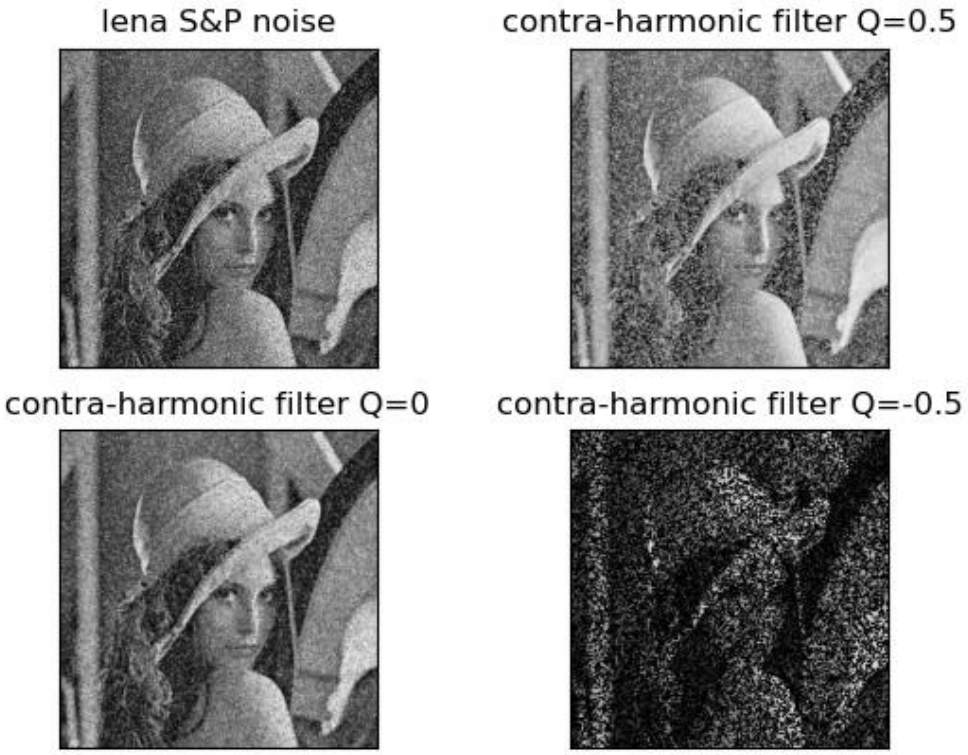

图13　反谐波滤波器处理椒盐噪声

根据实验结果可以得出如下结论：对于大于 0 的Q值，反谐波滤波器可以消除胡椒噪声；对于小于 0 的Q值，可以消除盐噪声。但反谐波滤波器不能同时做到消除胡椒噪声和盐噪声。

三 题目 3

3.1 题目

推导维纳滤波器，并实现以下要求：

a) 实现模糊滤波器如以下方程所示：

$$H(u,v)=\frac{T}{\pi(ua+vb)}\sin[\pi(ua+vb)]e^{-j\pi(ua+vb)}$$

b) 模糊 lena 图像：45 度方向，$T=1$；

c) 在模糊的 lena 图像中增加高斯噪声，以产生模糊图像；

d) 分别利用以下两个方程：

$$\hat{F}(u,v)=\left[\frac{1}{H(u,v)}\cdot\frac{|H(u,v)|^2}{|H(u,v)|^2+K}\right]\cdot G(u,v)$$

$$\hat{F}(u,v)=\left[\frac{H^*(u,v)}{|H(u,v)|^2+\gamma|P(u,v)|^2}\right]\cdot G(u,v)$$

恢复图像；并分析算法的优缺点。

3.2 解答

在以下的讨论中，图像的损坏过程可在空域表示为

$$g(x,y)=h(x,y)\bigstar f(x,y)+\eta(x,y)$$

其中★为卷积符号，根据傅里叶变换的卷积性质，以上过程在频域表示为

$$G(u,v)=H(u,v)F(u,v)+N(u,v)$$

在此问中，损失函数（degradation function，有时也被称为 point spread function）为

$$H(u,v)=\frac{T}{\pi(ua+vb)}\sin[\pi(ua+vb)]e^{-j\pi(ua+vb)}$$

在上式中取$T=1$，$a=0.1$，$b=0.1$（即$45°$方向）得出的模糊图像如下图所示。

图14　原图与模糊后的图像

注意到恢复图像与原图像的均方误差（Mean Squared Error, MSE）为

$$\text{MSE} = E\left[f(x,y) - \hat{f}(x,y)\right]^2$$

其中$f(x,y)$代表损坏前的原图像，$\hat{f}(x,y)$代表对损坏后的图像进行维纳滤波后的图像。

为了找到使得 MSE 最小的滤波函数，不妨设

$$e = MN\sum_x\sum_y\left|f(x,y) - \hat{f}(x,y)\right|^2$$

根据帕塞瓦尔定理，有：

$$e = MN\sum_x\sum_y\left|f(x,y) - \hat{f}(x,y)\right|^2 = \sum_u\sum_v\left|F(u,v) - \hat{F}(u,v)\right|^2$$

再带入图像损坏过程的函数，有：

$$e = \sum_u\sum_v\left|F(u,v) - [F(u,v)H(u,v) + N(u,v)]W(u,v)\right|^2$$

其中$W(u,v)$为维纳滤波函数，继续推导，得：

$$
\begin{aligned}
e &= \sum_u \sum_v |F(u,v)\,[1-H(u,v)W(u,v)]-N(u,v)W(u,v)|^2 \\
&= \sum_u \sum_v |F(u,v)\,[1-H(u,v)W(u,v)]|^2+|N(u,v)W(u,v)|^2 \\
&\sum_u \sum_v |F(u,v)|^2|1-H(u,v)W(u,v)|^2+|N(u,v)|^2|W(u,v)|^2
\end{aligned}
$$

注意，上述第二个等号的得出是基于$f(x,y)$与$\eta(x,y)$不相关的假定。

以下通过令$\frac{\partial e}{\partial W(u,v)}=0$来求得$W(u,v)$，注意到$\frac{\partial (xx^*)}{\partial x}=2x^*$，得到：

$$
\frac{\partial e}{\partial W(u,v)}=|F|^2[2(1-W^*H^*)\,(-H)]+|N|^2(2W^*)
$$

令$\frac{\partial e}{\partial W(u,v)}=0$，有：

$$
W^*(u,v)=\frac{|F(u,v)|^2H(u,v)}{|H(u,v)|^2|F(u,v)|^2+|N(u,v)|^2}
$$

则：

$$
\begin{aligned}
W(u,v) &= \frac{H^*(u,v)}{|H(u,v)|^2+\frac{|N(u,v)|^2}{|F(u,v)|^2}} = \frac{1}{H(u,v)}\frac{|H(u,v)|^2}{|H(u,v)|^2+\frac{|N(u,v)|^2}{|F(u,v)|^2}} \\
&= \frac{1}{H(u,v)}\frac{|H(u,v)|^2}{|H(u,v)|^2+\frac{1}{\mathrm{SNR}}}
\end{aligned}
$$

其中 SNR 为信噪比（Signal-to-noise ratio），以下设为K。

以下向模糊过的图像中添加高斯噪声，均值为 0，方差为 0.01，得到如下结果。

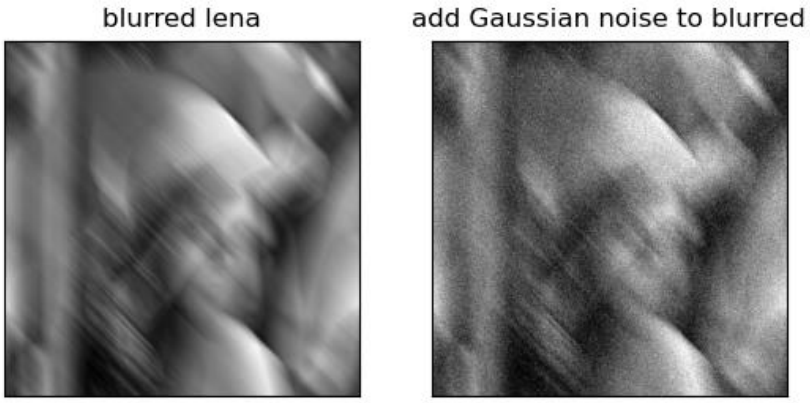

图15　向模糊后的图像添加高斯噪声

再对添加过高斯噪声的模糊图像运用维纳滤波器，取$K=0.05$，得到的滤波结果如下图所示。

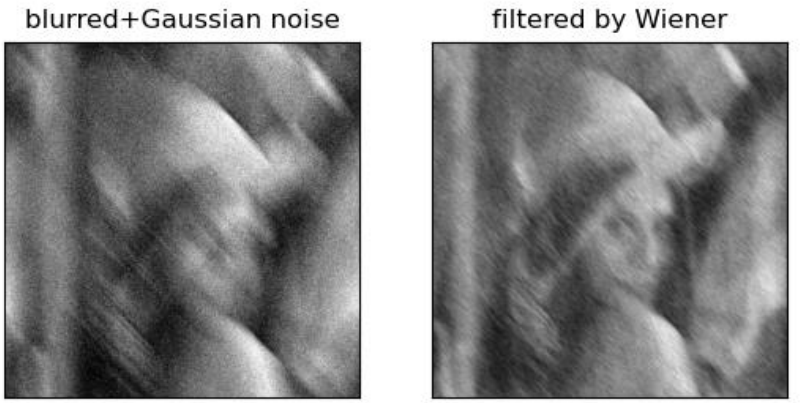

图16　维纳滤波

而约束最小二乘滤波则是基于寻找以下函数的最小值：

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

通过将原来的图像损坏的过程函数重写为向量-矩阵形式：

$$\mathbf{g}=\mathbf{H}\mathbf{f}+\boldsymbol{\eta}$$

可得以上C函数受限于如下等式：

$$\left\|\mathbf{g}-\mathbf{H}\hat{\mathbf{f}}\right\|^2=\|\boldsymbol{\eta}\|^2$$

其中$\hat{\mathbf{f}}$代表对原图的估计，则频域的滤波函数可表示为

$$\hat{F}(u,v)=\left[\frac{H^*(u,v)}{|H(u,v)|^2+\gamma|P(u,v)|^2}\right]G(u,v)$$

其中$p(x,y)=\begin{bmatrix}0&-1&0\\-1&4&-1\\0&-1&0\end{bmatrix}$为拉普拉斯核。

以下为使用最小二乘滤波器的结果：

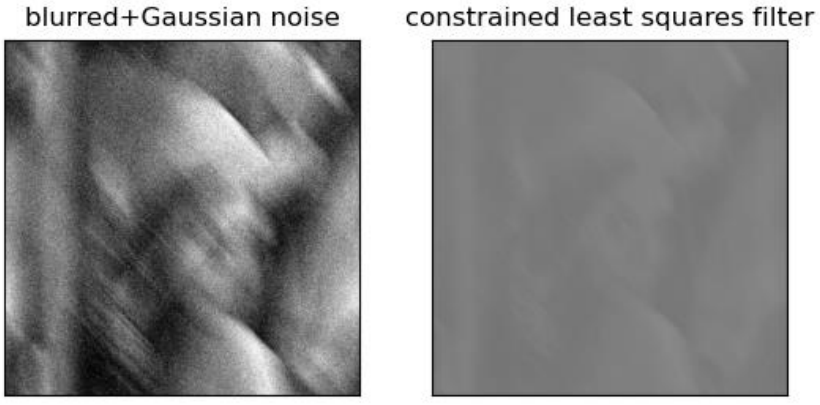

图17　使用最小二乘滤波器

可见，约束最小二乘滤波器滤波得到的图像，本应比维纳滤波器得到的图像更为清晰。理论上维纳滤波器得到的图像不能消除运动模糊的残影，而约束最小二乘方滤波的得到的图像对这些残影的去除效果十分优良，但此结果图像蒙上了薄影，效果不是很好。

四 附录：所有代码

4.1 导入有关库

```
import cv2
import matplotlib.pyplot as plt
from skimage import io
from skimage.util import img_as_float, random_noise, img_as_ubyte
from skimage.restoration import estimate_sigma
from PIL import Image, ImageFilter
from scipy.fft import fft2, ifft2, fftshift, ifftshift
import numpy as np
from numpy import pi, sin, exp
```

4.2 题目 1

```
# %% definition of functions

# 显示两幅图像的函数
def display_image(image_list, title_list):
    for i in range(2):
        plt.subplot(1, 2, i + 1)
        plt.imshow(image_list[i], 'gray')
        plt.title(title_list[i])
        plt.xticks([]), plt.yticks([])
    plt.show()


# %% read image
lena = io.imread("lena.bmp", as_gray=True)
lena = img_as_float(lena)

# %% add Gaussian noise
lenaGaussianNoise = random_noise(lena, mode='gaussian', mean=0.1, var=0.04)
images = [lena, lenaGaussianNoise]
titles = ['lena', 'Gaussian noise mean=0.1, var=0.04']
```

```
display_image(images, titles)

# Estimate the average noise standard deviation.
sigma_est = estimate_sigma(lenaGaussianNoise, channel_axis=None,
average_sigmas=True)
# Due to clipping in random_noise, the estimate will be a bit smaller than the
specified sigma.
print(f'Estimated Gaussian noise standard deviation = {sigma_est}')

# %% denoise Gaussian: Gaussian spatial filter
lenaGaussianNoiseCV = img_as_ubyte(lenaGaussianNoise)
lenaGaussian = cv2.GaussianBlur(lenaGaussianNoiseCV, ksize=(11, 11), sigmaX=1.5)
images = [lenaGaussianNoiseCV, lenaGaussian]
titles = ['lena Gaussian noise', 'filtered by Gaussian']
display_image(images, titles)

# %% denoise Gaussian: median filter
lenaMedian = cv2.medianBlur(lenaGaussianNoiseCV, ksize=5)
images = [lenaGaussianNoiseCV, lenaMedian]
titles = ['lena Gaussian noise', 'filtered by median filter']
display_image(images, titles)

# %% denoise Gaussian: arithmetic mean filter
lenaMean = cv2.blur(lenaGaussianNoiseCV, ksize=(5, 5))
images = [lenaGaussianNoiseCV, lenaMean]
titles = ['lena Gaussian noise', 'filtered by arithmetic mean filter']
display_image(images, titles)

# %% denoise Gaussian: geometric mean filter
CVF = lenaGaussianNoiseCV.astype(float)
prob_tmp = np.where(CVF > 1.0e-10, CVF, 1.0e-10)
result = np.where(CVF > 1.0e-10, np.log10(prob_tmp), -1000)
lenaGeometricMean = np.uint8(np.exp(cv2.boxFilter(result, ddepth=-1, ksize=(3,
3))))
images = [lenaGaussianNoiseCV, lenaGeometricMean]
titles = ['lena Gaussian noise', 'filtered by geometric mean filter']
display_image(images, titles)

# %% another way of geometric filter
img = lenaGaussianNoiseCV.astype(float)
rows, cols = img.shape[:2]
ksize = 5
padsize = int((ksize - 1) / 2)
pad_img = cv2.copyMakeBorder(img, *[padsize] * 4, cv2.BORDER_DEFAULT)
```

```
geomean = np.zeros_like(img)
for r in range(rows):
    for c in range(cols):
        geomean[r, c] = np.prod(pad_img[r: r + ksize, c: c + ksize]) ** (1 /
(ksize ** 2))
geomean = np.uint8(geomean)

images = [img, geomean]
titles = ['lena Gaussian noise', 'filtered by geometric mean filter']
display_image(images, titles)
```

4.3 题目 2

```
# %% add salt-and-pepper noise
lenaSPNoise = random_noise(lena, mode='s&p', amount=0.2, salt_vs_pepper=0.5)
images = [lena, lenaSPNoise]
titles = ['lena', 'salt and pepper noise']
display_image(images, titles)

# %% denoise S&P: arithmetic mean filter
lenaSPNoiseCV = img_as_ubyte(lenaSPNoise)
lenaSPMean = cv2.blur(lenaSPNoiseCV, ksize=(5, 5))
images = [lenaSPNoiseCV, lenaSPMean]
titles = ['lena S&P noise', 'filtered by arithmetic mean filter']
display_image(images, titles)

# %% denoise S&P: median filter
lenaSPMedian = cv2.medianBlur(lenaSPNoiseCV, ksize=3)
images = [lenaSPNoiseCV, lenaSPMedian]
titles = ['lena S&P noise', 'filtered by median filter']
display_image(images, titles)

# %% denoise S&P: max filter
lenaSPNoisePIL = Image.fromarray(lenaSPNoiseCV)
lenaSPMax = lenaSPNoisePIL.filter(ImageFilter.MaxFilter(size=3))
images = [lenaSPNoiseCV, lenaSPMax]
titles = ['lena S&P noise', 'filtered by max filter']
display_image(images, titles)

# %% denoise S&P: min filter
lenaSPMin = lenaSPNoisePIL.filter(ImageFilter.MinFilter(size=3))
images = [lenaSPNoiseCV, lenaSPMin]
```

```
titles = ['lena S&P noise', 'filtered by min filter']
display_image(images, titles)


# %% denoise S&P: contra-harmonic mean filter
def contra_harmonic_mean(image, size, Q):
    numerator = np.power(image, Q + 1)
    denominator = np.power(image, Q)
    kernel = np.full(size, 1.0)
    ret = cv2.filter2D(numerator, -1, kernel) / cv2.filter2D(denominator, -1,
kernel)
    return ret


lenaSPContra1 = contra_harmonic_mean(lenaSPNoise, size=(3, 3), Q=0.5)
lenaSPContra2 = contra_harmonic_mean(lenaSPNoise, size=(3, 3), Q=0)
lenaSPContra3 = contra_harmonic_mean(lenaSPNoise, size=(3, 3), Q=-0.5)
images = [lenaSPNoiseCV, lenaSPContra1, lenaSPContra2, lenaSPContra3]
titles = ['lena S&P noise', 'contra-harmonic filter Q=0.5',
          'contra-harmonic filter Q=0', 'contra-harmonic filter Q=-0.5']
for i in range(4):
    plt.subplot(2, 2, i + 1)
    plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]), plt.yticks([])
plt.show()
```

4.4 题目 3

```
# %% wiener filter

def my_wiener_filter(img, t, a, b, k):
    h = np.zeros(img.shape[:2], dtype='complex_')
    rows, columns = img.shape[:2]
    for u in range(rows):
        for v in range(columns):
            x = u - rows / 2
            y = v - columns / 2
            val = pi * (x * a + y * b)
            if val == 0:
                val = 1
            h[u, v] = (t / val) * sin(val) * exp(-1j * val)
    centered = fftshift(fft2(img))
```

```
    pass_center = np.zeros(img.shape[:2], dtype='complex_')
    for u in range(rows):
        for v in range(columns):
            pass_center[u, v] = ((1 / h[u, v]) * (abs(h[u, v])) ** 2 / ((abs(h[u,
v])) ** 2 + k)) * centered[u, v]
    passed = ifft2(ifftshift(pass_center)).real
    passed = ((passed - passed.min()) * (1 / (passed.max() - passed.min()) *
255)).astype('uint8')
    return passed


lenaWiener = my_wiener_filter(lenaGB, t=1, a=0.1, b=0.1, k=0.05)
images = [lenaGB, lenaWiener]
titles = ['blurred+Gaussian noise', 'filtered by Wiener']
display_image(images, titles)


# %% constrained least squares filter
def zero_pad(image, shape, position='corner'):
    shape = np.asarray(shape, dtype=int)
    imshape = np.asarray(image.shape, dtype=int)

    if np.alltrue(imshape == shape):
        return image

    if np.any(shape <= 0):
        raise ValueError("ZERO_PAD: null or negative shape given")

    dshape = shape - imshape
    if np.any(dshape < 0):
        raise ValueError("ZERO_PAD: target size smaller than source one")

    pad_img = np.zeros(shape, dtype=image.dtype)

    idx, idy = np.indices(imshape)

    if position == 'center':
        if np.any(dshape % 2 != 0):
            raise ValueError("ZERO_PAD: source and target shapes ""have different
parity.")
        offx, offy = dshape // 2
    else:
        offx, offy = (0, 0)
```

```
    pad_img[idx + offx, idy + offy] = image

    return pad_img


def psf2otf(psf, shape):
    if np.all(psf == 0):
        return np.zeros_like(psf)

    inshape = psf.shape
    # Pad the PSF to outsize
    psf = zero_pad(psf, shape, position='corner')

    # Circularly shift OTF so that the 'center' of the PSF is
    # [0,0] element of the array
    for axis, axis_size in enumerate(inshape):
        psf = np.roll(psf, -int(axis_size / 2), axis=axis)

    # Compute the OTF
    otf = np.fft.fft2(psf)

    n_ops = np.sum(psf.size * np.log2(psf.shape))
    otf = np.real_if_close(otf, tol=n_ops)

    return otf


LAPLACIAN = np.array([[0, -1,  0], [-1,  4, -1], [0, -1,  0]])


def my_constrained_least_squares_filter(img, t, a, b, g):
    h = np.zeros(img.shape[:2], dtype='complex_')
    rows, columns = img.shape[:2]
    for u in range(rows):
        for v in range(columns):
            x = u - rows / 2
            y = v - columns / 2
            val = pi * (x * a + y * b)
            if val == 0:
                val = 1
            h[u, v] = (t / val) * sin(val) * exp(-1j * val)
    centered = fftshift(fft2(img))
    pass_center = np.zeros(img.shape[:2], dtype='complex_')
    laplace = psf2otf(LAPLACIAN, (rows, columns))
```

```
    for u in range(rows):
        for v in range(columns):
            pass_center[u, v] = (h[u, v].conj() / ((abs(h[u, v])) ** 2 + g *
abs(laplace[u, v]) ** 2)) * centered[u, v]
    passed = ifft2(ifftshift(pass_center)).real
    passed = ((passed - passed.min()) * (1 / (passed.max() - passed.min()) *
255)).astype('uint8')
    return passed


lenaLSF = my_constrained_least_squares_filter(lenaGB, t=1, a=0.1, b=0.1,
g=0.001)
images = [lenaGB, lenaLSF]
titles = ['blurred+Gaussian noise', 'constrained least squares filter']
display_image(images, titles)
```