

Problem 5

Image restoration

问题描述

Suppose a blurring degradation function as

$$H(u, v) = \frac{T}{\pi(ua + vb)} \sin[\pi(ua + vb)]e^{-j\pi(ua+vb)} \quad (1)$$

- (a) Implement a blurring filter using Eq. (1).
- (b) Blur the test image book_cover.jpg using parameters $a = b = 0.1$ and $T = 1$.
- (c) Add Gaussian noise of 0 mean and variance of 650 to the blurred image.
- (d) Restore the blurred image and the blurred noisy image using the inverse filter, Wiener deconvolution filter, respectively.
- (e) Add Gaussian noise of 0 and different variances to the blurred image and repeat (d), investigate the performance of the Wiener deconvolution filter.

算法思想

在数字图像处理领域，退化模型可以用来处理图像复原方面的问题。在本题中，我们需要先对原始图像进行混淆滤波，获取模糊后的图像，然后再添加高斯噪声，最后做复原处理。

1. 混淆退化函数

题目中给出的混淆退化函数定义如下：

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

2. 高斯噪声

高斯噪声在 Problem 4 里有涉及，请参考上次作业。

由该函数可知，进行混淆滤波处理时，图像分别在 x, y 方向做 $x(t) = at/T, y(t) = bt/T$ 的均匀线性运动。

3. 逆滤波

逆滤波函数的定义如下：

$$\hat{F}(u, v) = \frac{G(u, v)}{H(u, v)}$$

根据题意，我们应使用退化后图像的傅里叶变换来进行逆滤波处理，把得到的结果作为原始图像的傅里叶变换，并对原始图像进行傅里叶反变换，得到原始图像的复原图。

4. 傅里叶变换与反变换

高斯噪声在 Problem 3 里有涉及，请参考第三次作业。

5. 维纳滤波

维纳滤波函数的定义如下：

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

其中， $H(u, v)$ 为退化函数， S_η 为噪声的功率谱， S_f 为未退化图像的功率谱。

维纳滤波是一种平稳随机过程的最佳滤波理论，换句话说就是在滤波过程中系统的状态参数（或信号的波形参数）是稳定不变的。它将所有时刻的采样数据用来计算互相关矩阵，涉及到解维纳-霍夫方程。可以说维纳滤波仅在理论上具有意义，在实际应用中的局限性表现在：不适用于非平稳的随机过程的滤波；要用到所有时刻的采样数据，需要的数据存储容量大；解维纳-霍夫方程是要用到矩阵的求逆运算，计算量大（因为互相关矩阵的阶数很大），而且实际数据下的维纳-霍夫方程可能无解。

源码分析

高斯噪声、傅里叶变换与反变换使用的是之前作业里的代码，在此不再附上源码。

- - a. 读取原始图片并将其转为灰度图像
 - b. 使用混淆退化函数得到函数H，对原始图像进行二维傅里叶变换，得到原始图像的功率谱，并将其与H做卷积运算，得到混淆退化图像的功率谱，最后再做二维傅里叶反变换，得到混淆退化图像
 - c. 使用高斯噪声函数生成一个噪声矩阵，并对其进行二维傅里叶变换，再与退化混淆图像的功率谱相加得到添加噪声后的功率谱，再对其做二维傅里叶反变换得到添加噪声后的混淆退化图像
 - d. 对于没有加噪声的混淆退化图像功率谱做逆滤波处理，再进行二维傅里叶反变换，得到逆滤波复原图片
 - e. 对加了噪声后的混淆退化图像功率谱做维纳滤波处理，再进行二维傅里叶反变换，得到维纳滤波复原图片
 - f. 显示所有图片，并修改噪声方差再多次进行实验。

```
1 def main():
2     originalImage = np.array(Image.open('./resource/book_cover.jpg'))
3     plt.subplot(2, 3, 1)
4     plt.imshow(originalImage, cmap = plt.get_cmap('gray'))
5     plt.title('Original')
6
7     degradationFunction = BlurringDegradation.getBlurringDegradationFunction(originalImage)
8     originalFFT = np.fft.fft2(originalImage)
9     blurringDegradationImageFFT = originalFFT * degradationFunction
10    plt.subplot(2, 3, 2)
11    plt.imshow(np.real(np.fft.ifft2(blurringDegradationImageFFT)), cmap = plt.get_cmap('gray'))
12    plt.title('Blurring Degradation')
13
14    mean = 0
15    variance = 0.001
16    noiseFunction = AddNoise.addGaussianNoise(blurringDegradationImageFFT, mean, variance)
17    plt.subplot(2, 3, 3)
18    noiseFunctionFFT = np.fft.fft2(noiseFunction)
19    noiseImageFFT = blurringDegradationImageFFT + noiseFunctionFFT
20    plt.imshow(np.real(np.fft.ifft2(noiseImageFFT)), cmap = plt.get_cmap('gray'))
21    plt.title('Gaussian Noise(Mean = %s, Variance = %s)' % (mean, variance))
22
```

```

23 IF = Filter.inverseFilter(blurringDegradationImageFFT, degradationFunction)
24 plt.subplot(2, 3, 5)
25 invertFilterImage = np.real(np.fft.ifft2(IF))
26 plt.imshow(invertFilterImage, cmap = plt.get_cmap('gray'))
27 plt.title('Inverse Filter')
28
29 IFFT = Filter.WienerDeconvolutionFilter(noiseImageFFT, noiseFunctionFFT, degradation
30 Function, originalFFT)
31 resultImage = np.real(np.fft.ifft2(IFT))
32 plt.subplot(2, 3, 6)
33 plt.imshow(resultImage, cmap = plt.get_cmap('gray'))
34 plt.title('Wiener Deconvolution Filter(Mean = %s, Variance = %s)' % (mean, variance)
35 )
36
37 plt.show()

```

- 混淆退化函数

```

1 def functionH(u, v, a = 0.1, b = 0.1, T = 1):
2     temp = np.pi * (u * a + v * b)
3     if temp == 0:
4         return 1
5     res = T * np.sin(temp) * np.exp(np.complex(0, -temp)) / temp
6     return res
7
8 def getBlurringDegradationFunction(originalImage):
9     row, col = originalImage.shape
10    degradationFunction = np.zeros((row, col), dtype = np.complex)
11    for i in range(row):
12        for j in range(col):
13            degradationFunction[i, j] = functionH(i - row / 2, j - col / 2)
14    return degradationFunction

```

- 逆滤波函数

此处直接调用了混淆退化函数作除法。

```

1 def inverseFilter(originalImage, degradationFunction):
2     row, col = originalImage.shape
3     res = copy.copy(originalImage)
4     for i in range(row):
5         for j in range(col):
6             res[i, j] /= degradationFunction[i, j]
7     return res

```

- 维纳滤波函数

```

1 def WienerDeconvolutionFilter(G, Sn, H, Sf):
2     HConjugate = np.conjugate(H)
3     HSpectrum = np.real(H * HConjugate)
4     SnConjugate = np.conjugate(Sn)
5     SnSpectrum = np.real(Sn * SnConjugate)
6     SfConjugate = np.conjugate(Sf)
7     SfSpectrum = np.real(Sf * SfConjugate)
8     row, col = G.shape
9     F_hat = np.zeros((row, col), dtype = np.complex)
10    for u in range(row):
11        for v in range(col):
12            F_hat[u, v] = G[u, v] * HSpectrum[u, v] / ((HSpectrum[u, v] + SnSpectrum[u,
13 v]) / SfSpectrum[u, v]) * H[u, v])
14    return F_hat

```

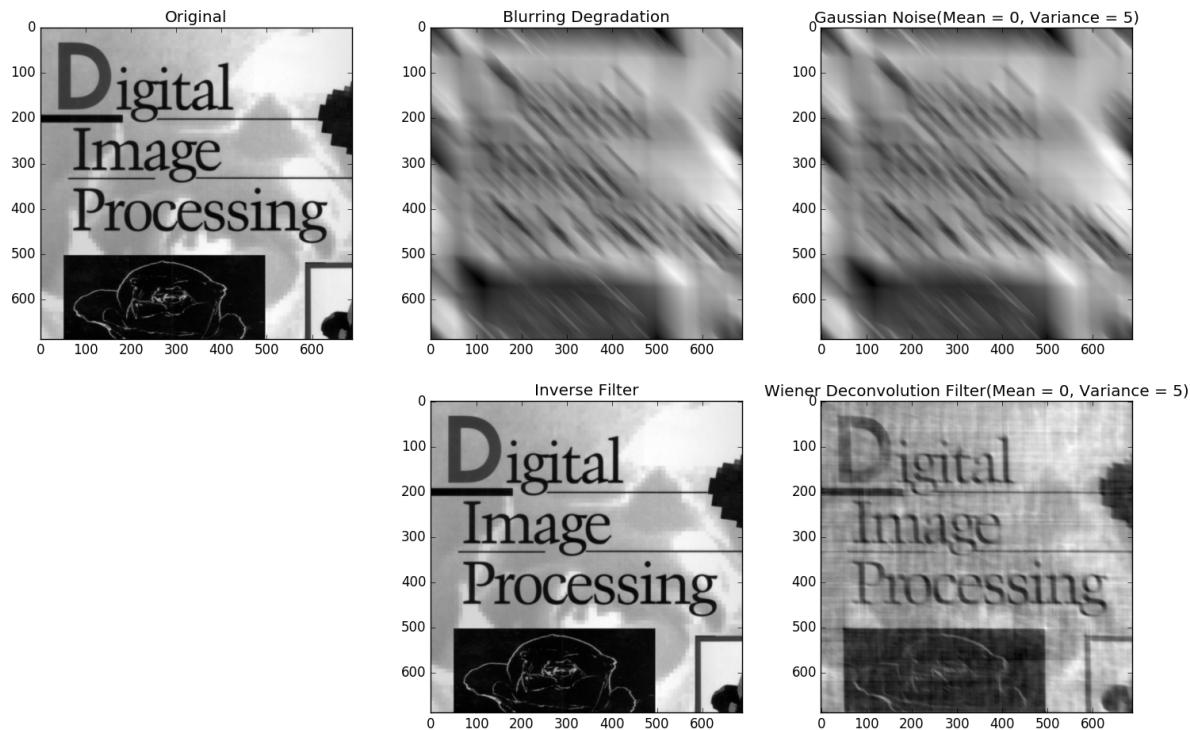
实验结果

下列图片为不同高斯噪声方差下面的实验结果。

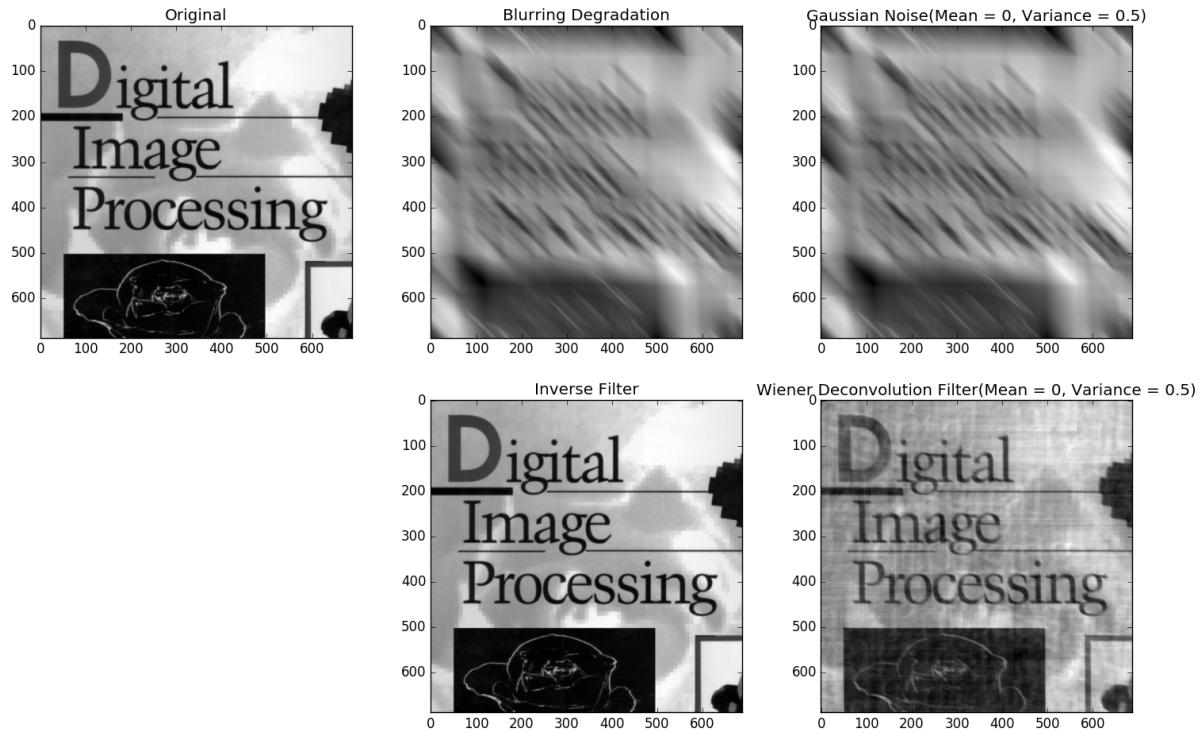
高斯噪声方差为 50 的情况



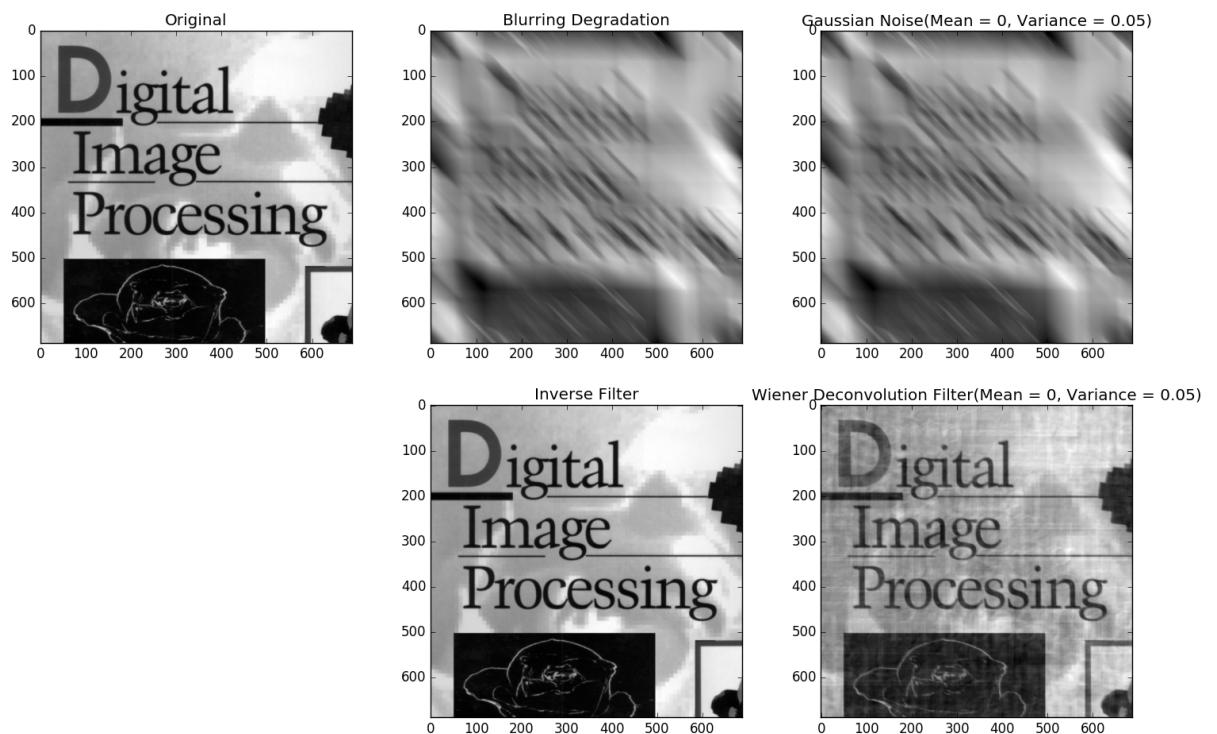
高斯噪声方差为 5 的情况



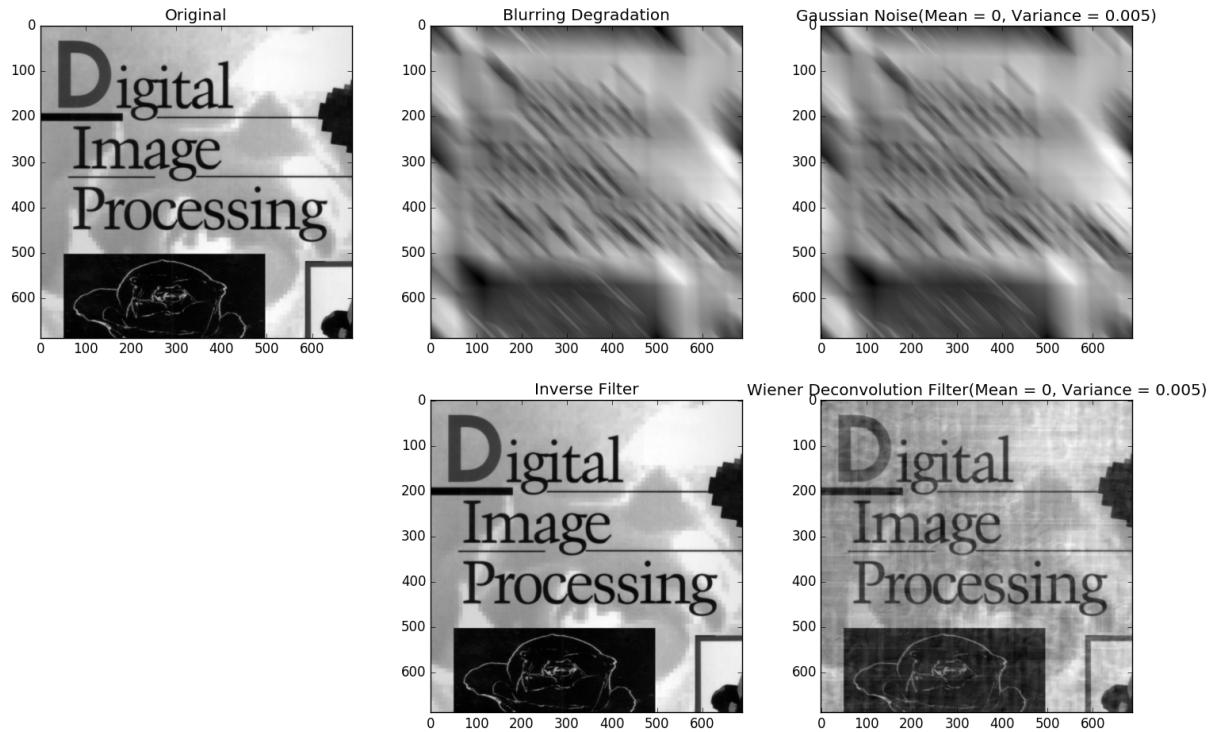
高斯噪声方差为 0.5 的情况



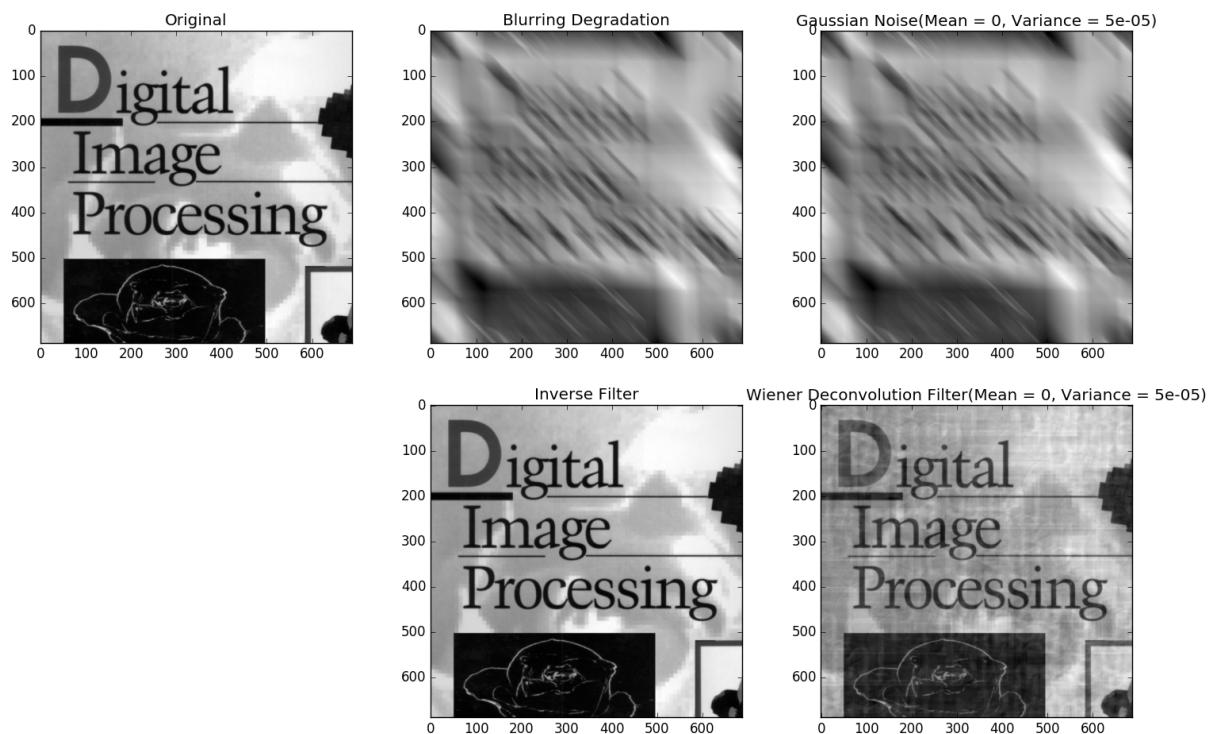
高斯噪声方差为 0.05 的情况



高斯噪声方差为 0.005 的情况



高斯噪声方差为 0.00005 的情况



由以上实验结果可以看出，逆滤波复原的效果很好，维纳滤波在较高的高斯噪声方差下很难复原出原始图片，随着方差逐渐减少，复原的效果越来越好。

由以上实验结果可以得知，维纳滤波不适用于非平稳的随机过程的滤波，对高方差的噪声十分敏感，在实际使用中有很大的局限性。