

# 图像处理与可视化 Homework-5 报告

林子开 21307110161

2023 年 12 月 16 日

## 目录

<b>1 主要 Python 文件简介</b>	<b>1</b>
<b>2 噪声的生成</b>	<b>1</b>
2.1 白噪声的生成 . . . . .	1
2.2 瑞利噪声的生成 . . . . .	2
2.3 向测试图像添加噪声 . . . . .	3
<b>3 最佳陷波滤波器</b>	<b>6</b>
3.1 最佳陷波滤波器的原理 . . . . .	6
3.2 Python 代码实现 . . . . .	6
3.3 滤波效果展示 . . . . .	9

## 1 主要 Python 文件简介

由于实现本次作业的代码文件比较复杂，下面进行简要说明。

- `white_noise_generator.py` 文件能够生成白噪声。白噪声的频谱为 1，相位角服从均匀分布或正态分布，允许用户从中选择。
- `Rayleigh_noise_generator.py` 文件能够生成瑞利噪声。
- `1_adding_noise.py` 文件向测试用图分别添加不同强度白噪声或瑞利噪声。
- `2_best_notch_filter.py` 文件实现了最佳陷波滤波器，以上次作业带斜纹的噪声的脑膜图为测试用图。

## 2 噪声的生成

### 2.1 白噪声的生成

**噪声原理介绍** 在 `white_noise_generator.py` 文件中，函数 `generator_w` 实现了白噪声的生成。在该函数中，白噪声的频谱均取常数 1；相位角有两种选择：(1) 服从  $[0, 2\pi]$  上的均匀分布，(2) 服从  $\mathcal{N}(0, \pi)$  的正态分布。对频域进行傅里叶逆变换后，对时域取实部，并且进行归一化： $\tilde{n}(i, j) = n(i, j)/n_{max}$ ，其中  $n_{max} = \max_{i,j} |n(i, j)|$ 。

白噪声生成代码 生成白噪声的 Python 代码如下:

Listing 1: Python codes to generate white noise

```
1 """能够生成白噪声: 频谱为均值, 相位角为随机变量, 返回的噪声会进行归一化"""
2
3 import numpy as np
4 from math import pi
5
6 def generator_w(m: int, n: int, rv_type: str='uniform') -> np.array:
7     """
8         generator of white noise
9         :param: m is the height of your figure
10        :param: n is the width of your figure
11        :param: const is the constant of the spectrum, default 1
12        :param: rv_type is the type of R.V. of phase in the spectrum, it is allowed to be
13        'uniform' or 'Gauss'(with miu = 0 and sigma = pi), default 'uniform'
14        :return: an array of size m*n, containing real white noise, normalized to [-1,1]
15    """
16    const = 1
17    noise_spectrum = const * np.ones((m,n))
18    if rv_type == 'uniform':
19        noise_phase = 1j * np.random.uniform(0,2*pi,(m,n))
20        noise_phase = np.exp(noise_phase)
21    elif rv_type == 'Gauss':
22        noise_phase = 1j * np.random.normal(0,pi,(m,n))
23        noise_phase = np.exp(noise_phase)
24
25    # IDFT, 并且只取实部
26    noise = np.fft.ifft2(noise_spectrum*noise_phase).real
27    # 归一化到 [-1,1] 区间
28    noise = noise/np. max(np. abs(noise))
29    return noise
```

## 2.2 瑞利噪声的生成

**噪声原理介绍** 在 `Rayleigh_noise_generator.py` 文件中, 函数 `generator_r` 实现了瑞利噪声的生成。该函数产生服从分布  $p(z) = \frac{2}{b}(z-a) \exp\left\{-\frac{(z-a)^2}{b}\right\}$ ,  $z \geq a$  的瑞利噪声。与产生白噪声的函数类似, 产生瑞利噪声函数在返回噪声时也会进行归一化:  $\tilde{n}(i,j) = n(i,j)/n_{max}$ , 其中  $n_{max} = \max_{i,j} |n(i,j)|$ 。

瑞利噪声生成代码 生成瑞利噪声的 Python 代码如下:

Listing 2: Python codes to generate Rayleigh noise

```
1 """能够生成瑞利噪声, 返回的噪声会进行归一化"""
2
3 import numpy as np
4
5 def generator_r(m: int, n: int, a: float = -1, b: float = 10) -> np.array:
6     """
7         generator of white noise
8         :param: m is the height of your figure
```

```

9     :param: n is the width of your figure
10    :param: a is a constant add to the Rayleigh r.v., default -1
11    :param: b is the scale of the pdf of Rayleigh distribution, default 10
12    :return: an array of size m*n, containing real white noise, normalized to [-1,1]
13    """
14
15    noise = a + np.random.rayleigh(scale=b,size=(m,n))
16    noise = noise/np. max(np. abs(noise))
17    return noise

```

## 2.3 向测试图像添加噪声

添加噪声的细节说明 添加噪声的数学公式为

$$\hat{f}(x, y) = f(x, y) + c\eta(x, y)$$

其中， $\eta(x, y)$  分别选取相位角服从均匀分布的白噪声，相位角服从正态分布的白噪声，以及瑞利噪声。三类噪声均提前进行了归一化处理。系数  $c$  则代表了噪声强度，在本次实验中，分别取  $c = 50, 100, 150$ 。在添加噪声后，对于超出  $[0, 255]$  区间的灰度值，统一进行截断处理。

添加噪声的代码实现 向图像中添加噪声的 Python 代码如下：

Listing 3: Python codes to add noise

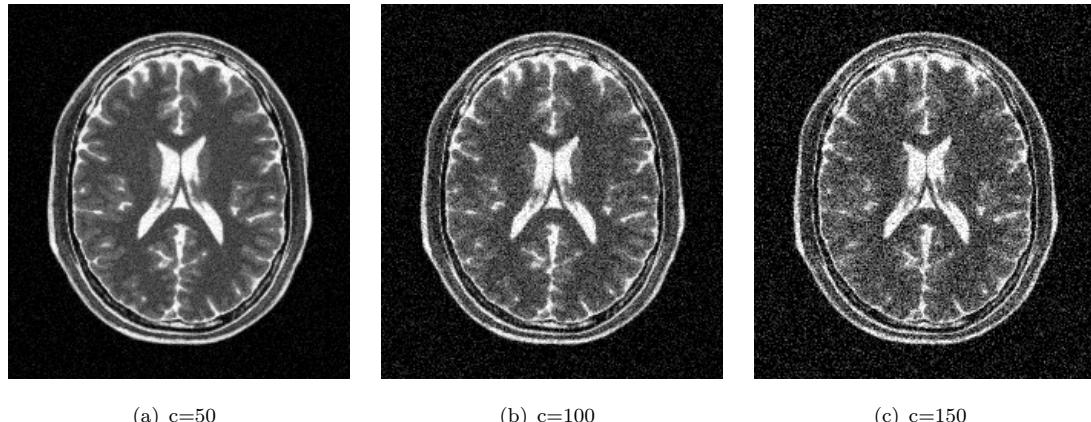
```

1 import numpy as np
2 from PIL import Image
3 from white_noise_generator import generator_w
4 from Rayleigh_noise_generator import generator_r
5
6
7 def truncate(M:np.array) -> np.array:
8     """将超出[0,255]区间的灰度值进行截断，并返回uint8类型"""
9     M = np.clip(M,0,255) # 截断
10    M = np.uint8(M) # 无符号整数可以表示0~255
11    return M
12
13
14 def main():
15     for fileName in ['大脑图像.png','心脏图像.png']:
16         im_raw = Image. open(fileName)
17         im = im_raw.convert('L')
18         pixels = np.array(im)
19         m = pixels.shape[0]
20         n = pixels.shape[1]
21         for c in [50,100,150]:
22             pixels_w_u_noise = pixels + c*generator_w(m,n,rv_type='uniform')
23             im1 = Image.fromarray(truncate(pixels_w_u_noise))
24             im1.save('adding_noise_result/'+fileName[0:4]+' c='+str(c)+'_uniform white noise.png')
25             pixels_w_g_noise = pixels + c*generator_w(m,n,rv_type='Gauss')
26             im2 = Image.fromarray(truncate(pixels_w_g_noise))
27             im2.save('adding_noise_result/'+fileName[0:4]+' c='+str(c)+'_Gauss white noise.png')

```

```
28     pixels_r_noise = pixels + c*generator_r(m,n)
29     im3 = Image.fromarray(truncate(pixels_r_noise))
30     im3.save('adding_noise_result/'+fileName[0:4]+' c='+ str(c) +' Rayleigh noise.png')
31
32
33 if __name__ == '__main__':
34     main()
```

添加噪声后的效果 以下是对大脑图像添加不同类型、不同强度噪声后的效果

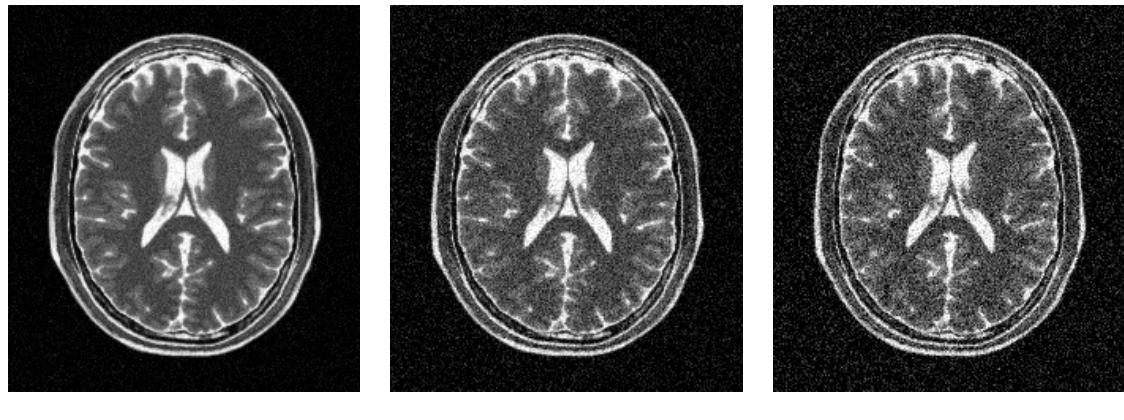


(a)  $c=50$

(b)  $c=100$

(c)  $c=150$

图 1: 向大脑图像添加相位角服从均匀分布的白噪声



(a)  $c=50$

(b)  $c=100$

(c)  $c=150$

图 2: 向大脑图像添加相位角服从  $\mathcal{N}(0, \pi)$  正态分布的白噪声

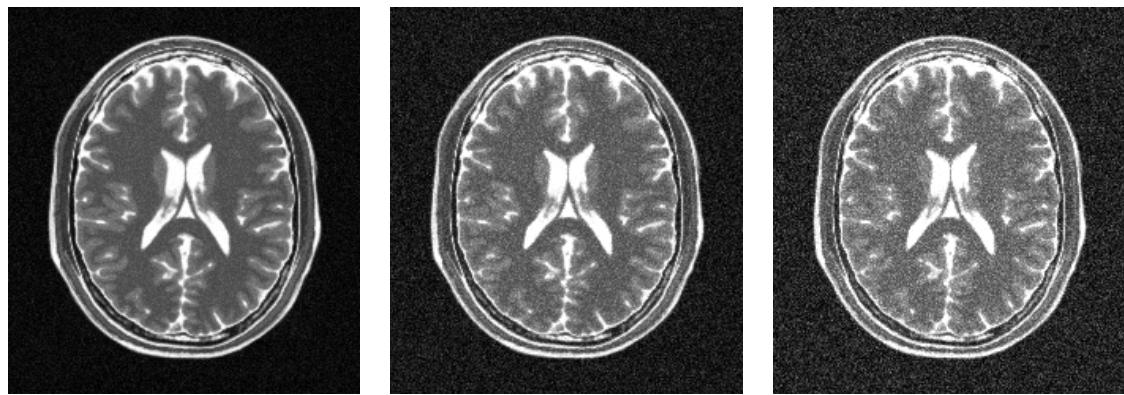
(a)  $c=50$ (b)  $c=100$ (c)  $c=150$ 

图 3: 向大脑图像添加瑞利噪声

以下是对心脏图像添加不同类型、不同强度噪声后的效果

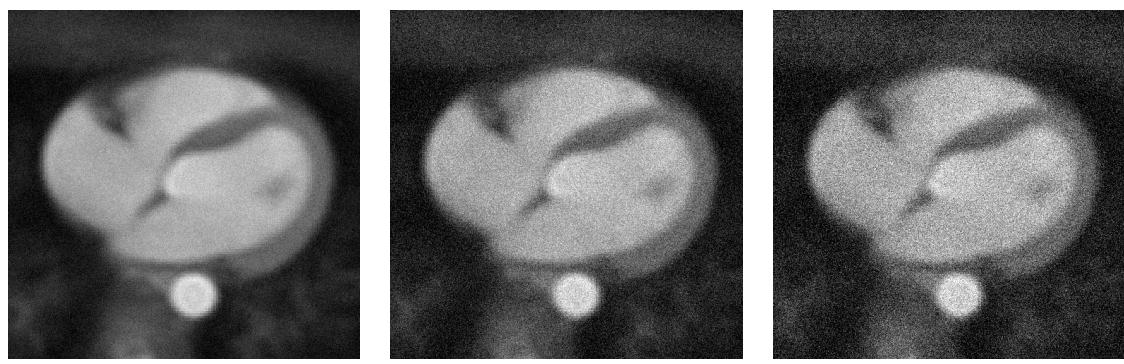
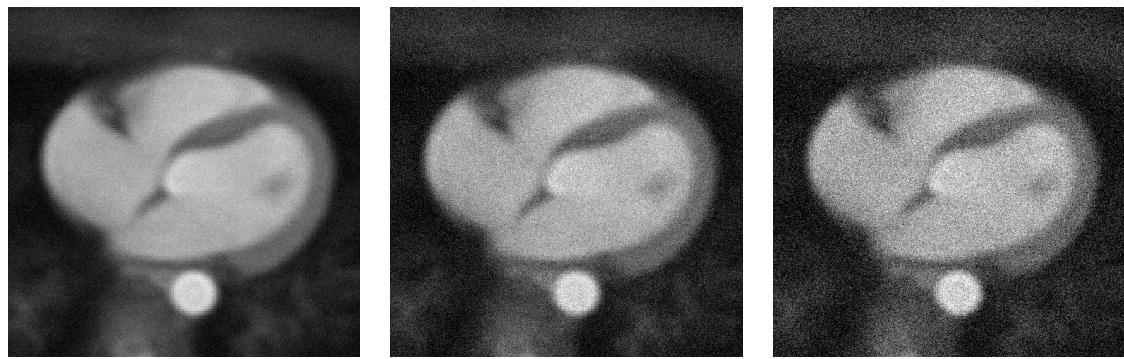
(a)  $c=50$ (b)  $c=100$ (c)  $c=150$ 

图 4: 向心脏图像添加相位角服从均匀分布的白噪声

(a)  $c=50$ (b)  $c=100$ (c)  $c=150$ 图 5: 向心脏图像添加相位角服从  $\mathcal{N}(0, \pi)$  正态分布的白噪声

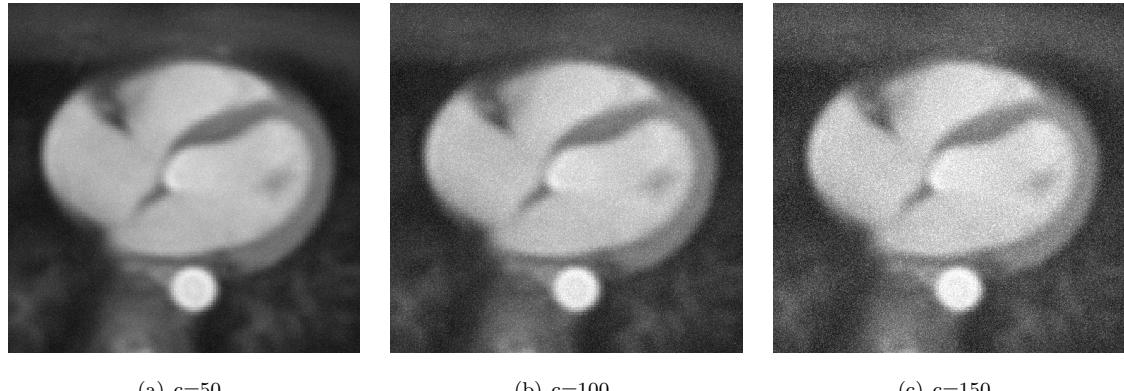
(a)  $c=50$ (b)  $c=100$ (c)  $c=150$ 

图 6: 向心脏图像添加瑞利噪声

### 3 最佳陷波滤波器

#### 3.1 最佳陷波滤波器的原理

首先观察带噪声图像的频域图像，构建频域滤波器，得到噪声的频域函数  $N(u, v)$ 。对  $N(u, v)$  进行傅里叶逆变换，得到噪声的时域函数  $\eta(x, y) = \mathcal{F}^{-1}\{N(u, v)\}$ 。

进行时域滤波时，需要对含噪声的图像进行如下去噪操作：

$$\hat{f}(x, y) = g(x, y) - w(x, y)\eta(x, y)$$

其中  $w(x, y)$  为调制函数，表达式可以简写为：

$$w(x, y) = \frac{\bar{g}\bar{\eta} - \bar{g}\bar{\eta}}{\bar{\eta}^2 - \bar{\eta}^2}$$

$g(x, y)$  和  $\eta(x, y)$  分别是以  $(x, y)$  为中心的邻域上的原图灰度值和噪声灰度值。不同的邻域大小可能产生不同的  $w(x, y)$ 。

#### 3.2 Python 代码实现

实现最佳陷波滤波器的 Python 代码如下所示。

Listing 4: Python codes of optimal notch filter

```

1 """实现了最佳陷波滤波器"""
2
3 import numpy as np
4 from PIL import Image
5 from numba import njit # 加速器
6
7 # 辅助函数
8 def truncate(M:np.array) -> np.array:
9     """将超出[0,255]区间的灰度值进行截断，并返回uint8类型"""
10    M = np.clip(M,0,255) # 截断
11    M = np.uint8(M) # 无符号整数可以表示0~255
12    return M
13
14 # 辅助函数

```

```

15 def rescale(M:np.array) -> np.array:
16     """线性映射到[0,255]区间，并返回uint8类型"""
17     # m,n = M.shape
18     largest = np. max(M)
19     least = np. min(M)
20     M = 255 * (M - least)/(largest - least)
21     M = np.uint8(M) # 无符号整数可以表示0~255
22     return M
23
24 # 辅助函数
25 def logRescale(M:np.array) -> np.array:
26     """先取模得到频谱，然后取对数，并映射到[0,255]区间，将映射后的频谱转为uint8返回"""
27     # m,n = M.shape
28     M = np. abs(M) # 取模得到频谱
29     M = np.log(M + 1) # 取对数
30     largest = np. max(M)
31     least = np. min(M)
32     M = 255 * (M - least)/(largest - least)
33     M = np.uint8(M) # 无符号整数可以表示0~255
34     return M
35
36 # 辅助函数
37 def get_freq_spectrum(pixels:np.array):
38     """得到频域"""
39     m,n = pixels.shape
40     # padding
41     P = np.zeros((2*m,2*n))
42     P[0:m,0:n] = pixels
43     # DFT and centerization
44     F = np.fft.fft2(P)
45     F = np.fft.fftshift(F)
46     return F
47
48 @njit
49 def best_notch_filter(pixels,noise,patchSize: int):
50     """最佳陷波滤波器，要求patchSize为奇数"""
51     X,Y = pixels.shape
52     half = patchSize//2
53     w = np.zeros((X,Y))
54     for i in range(half,X-half): # 不考虑边缘区域
55         for j in range(half,Y-half):
56             g = pixels[i-half:i+half+1,j-half:j+half+1]
57             n = noise[i-half:i+half+1,j-half:j+half+1]
58             gn = g*n
59             n2 = n*n
60             n_mean = np.average(n)
61             n2_mean = np.average(n2)
62             gn_mean = np.average(gn)
63             g_mean = np.average(g)
64             w[i,j] = (gn_mean - g_mean*n_mean)/(n2_mean - n_mean + 0.001)
65             # print(w[i,j])
66             # time.sleep(0.05)

```

```

67     # print(w)
68     return pixels - w*noise
69     # return pixels - noise
70
71
72 def main():
73     im_raw = Image. open("带噪声的脑膜图.png")
74     im = im_raw.convert('L')
75     pixels = np.array(im)
76
77     # 寻找噪声模式（展示带噪声图像的频谱）
78     F = get_freq_spectrum(pixels)
79     F_log = logRescale(F)
80     im_ori_freq_spectrum = Image.fromarray(F_log)
81     im_ori_freq_spectrum.save('notch_filter_result/'+'原图的频谱.png')
82
83
84     # 构建针对噪声的频域滤波器（展示并保存频域滤波器）
85     m,n = pixels.shape
86     H = np.ones((2*m,2*n))
87     H[m-90:m+90,:] = 0
88     H[:,n-90:n+90] = 0
89
90     H_rescale = rescale(H)
91     im_H = Image.fromarray(H_rescale)
92     im_H.save('notch_filter_result/'+'噪声滤波器示意图.png')
93
94
95     # 展示噪声的干扰模式（噪声的频谱以及噪声的空间图像）
96     noise_freq = F * H
97     noise_freq_log = logRescale(noise_freq)
98     im_noise_freq_log = Image.fromarray(noise_freq_log)
99     im_noise_freq_log.save('notch_filter_result/'+'噪声的频域图像.png')
100
101    noise_freq = np.fft.ifftshift(noise_freq) # decenterization
102    noise = np.fft.ifft2(noise_freq).real # IDFT and remain the real part
103    noise = noise[0:m,0:n] # remain the left top part
104    noise_rescale = rescale(noise)
105    im_noise = Image.fromarray(noise_rescale)
106    im_noise.save('notch_filter_result/'+"噪声图像.png")
107
108
109    # 使用最佳陷波滤波器进行空间过滤
110    for patchSize in [3,5,7,9,11,13]:
111        filtered_figure = best_notch_filter(pixels,noise,patchSize)
112
113        # 保存滤波后的图像
114        # filtered_figure = rescale(filtered_figure)
115        filtered_figure = truncate(filtered_figure)
116        # filtered_figure = global_equalization(filtered_figure)
117        im = Image.fromarray(filtered_figure)
118        im.save('notch_filter_result/'+'滤波后的图像 patchSize=' + str(patchSize) + '.png')

```

```

119
120
121 if __name__ == '__main__':
122     main()

```

### 3.3 滤波效果展示

选取 Homework-4 所提供的带噪声的脑膜图作为测试图，如图7所示。

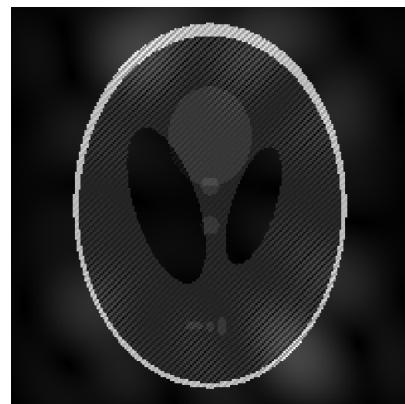
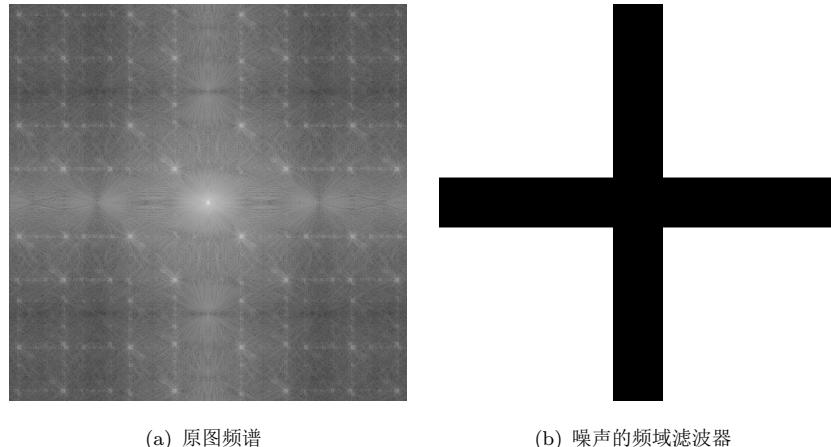


图 7: 测试用途: 带噪声的脑膜图

首先根据频谱，构建相应的频域滤波器，提取噪声的频域函数。原图的频谱如图8(a)所示。根据原图频谱，构建相应的频域滤波器，如图8(b)所示。

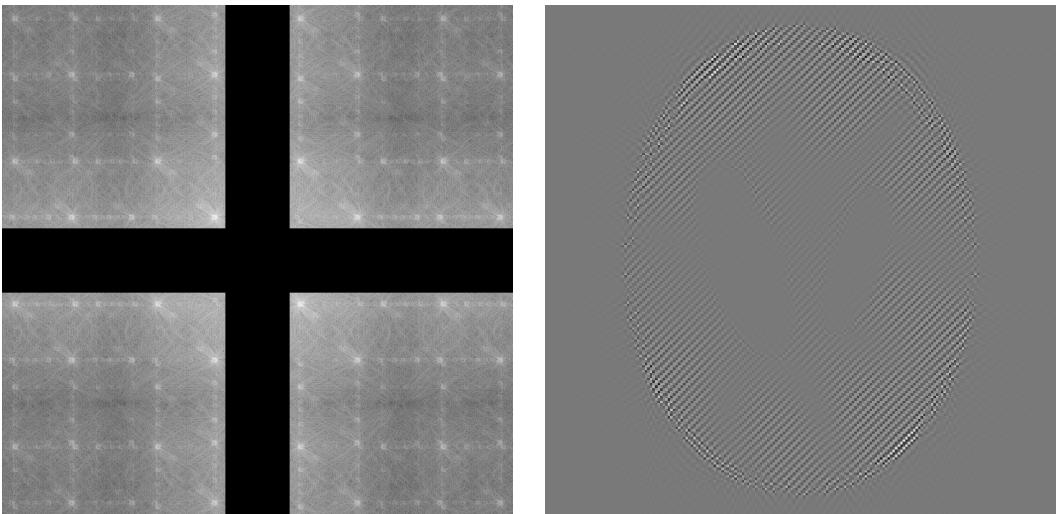


(a) 原图频谱

(b) 噪声的频域滤波器

图 8: 原图频谱和噪声的频域滤波器

用该频域滤波器对原图频谱进行频域滤波，得到噪声的频域函数  $N(u, v)$ ，如图9(a)所示。由噪声的频域图像，得到噪声的时域函数  $\eta(x, y)$ ，如图9(b)所示。

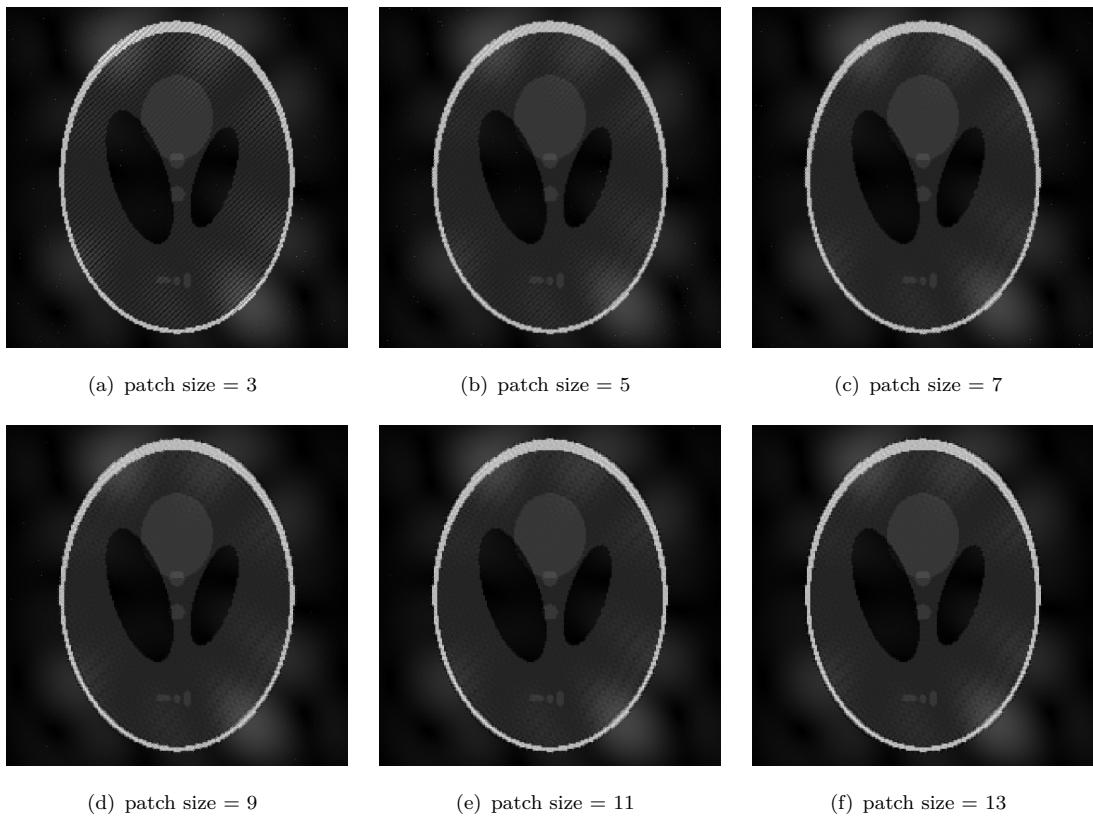


(a) 噪声的频域图像

(b) 噪声的时域图像

图 9: 噪声的频域和时域图像

尝试不同的邻域大小（要求为奇数）的滤波效果如下：



(a) patch size = 3

(b) patch size = 5

(c) patch size = 7

(d) patch size = 9

(e) patch size = 11

(f) patch size = 13

图 10: 基于不同邻域大小的最佳陷波滤波器的滤波效果

可以看出，当邻域大小  $\text{patchSize}=7$  时，滤波效果已经比较好；当继续增加  $\text{patchSize}$ ，滤波效果不再有显著改变。因此，在本次实验中，对于该测试用图，邻域大小取 7 是比较合适的。