Assignment 4

Junyi Liao 20307110289

November 4, 2022

0 Frequency Filter

In digital image processing, we can transform the digital image signal from spatial domain to frequency domain by discrete Fourier transform (DFT):

$$F(u,v) = \mathfrak{J}\{f(x,y)\} \triangleq \sum_{x=1}^{M}\sum_{y=1}^{N} f(x,y)\exp\left\{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)\right\},$$

and rebuild an image in spatial domain by its frequency representation by inverse Fourier transform:

$$f(x,y) = \mathfrak{J}^{-1}\{F(u,v)\} \triangleq \frac{1}{MN}\sum_{x=1}^{M}\sum_{y=1}^{N} F(u,v)\exp\left\{j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)\right\};$$

In implementation, to visualize the frequency representation of an image of size $M \times N$ such as its spectrum, we prefer the origin located at the center $(u_0, v_0) = (M/2, N/2)$ of the image. This can be easily achieved by multiplying a transform term $\mathrm{e}^{j2\pi(u_0 x/M + v_0 y/N)} = (-1)^{x+y}$ in DFT and IDFT:

$$F(u,v) = \mathfrak{J}\{f(x,y)(-1)^{x+y}\}, \quad f(x,y) = \mathfrak{J}^{-1}\{F(u,v)\}(-1)^{x+y};$$

In frequency domain, we can make a pointwise product of an image F with a frequency filter H, which is equivalent to the convolution operation in spatial domain, between image f and spatial filter h by the convolution theorem in 2D-DFT:

$$\mathfrak{J}\{f * h\} = F \cdot H,\ F = \mathfrak{J}\{f\},\ H = \mathfrak{J}\{h\};$$

Meanwhile, to deal with the wraparound error of the frequency filer, we preprocess the image by zero-padding and resize it to $P \times Q,\ P = 2M,\ Q = 2N$; after DFT and IDFT, the filtered image is extracted from the upper left corner.
The algorithm can be summerized as below:

Algorithm The frequency filter H is a real symmetric array of size $P \times Q$.

FILTER(f, H)

1 Initialize $f_p(x,y)$ as a zero array of size $P \times Q$, $f_p(0:M-1, 0:N-1) = f(0:M-1, 0:N-1)$
2 $F(u,v) = \mathrm{DFT}(f_p(x,y)(-1)^{x+y})$
3 $G(u,v) = H(u,v)F(u,v)$
4 $g_p(x,y) = \mathrm{Real}\{\mathrm{IDFT}(G(u,v))\}(-1)^{x+y}$
5 **return** $g_p(0:M-1, 0:N-1)$

Code We call `numpy.fft.fft2` and `numpy.fft.ifft2` to implement DFT and IDFT.

```
def dft(img):
    h, w = img.shape

    # Padding.
    img_pad = np.zeros((2 * h, 2 * w))
    img_pad[:h, :w] = img

    # Centralization.
    sign = -np.ones_like(img_pad)
    for i in range(2 * h):
    s = i % 2
    sign[i, s::2] = 1
    img_cen = img_pad * sign

    # DFT.
    return np.fft.fft2(img_cen)


def idft(img_freq):
    p, q = img_freq.shape
    # IDFT.
    img_re = np.fft.ifft2(img_freq)

    # Restore the sign.
    sign = -np.ones((p, q))
    for i in range(p):
    s = i % 2
    sign[i, s::2] = 1

    # Extract the filtered image from upper-left corner.
    img_re = np.real(img_re * sign)[:p // 2, :q // 2]
    return img_re.astype(int)


def symmetrize(f_mat):
    # Generate a centralized symmetric filter.
    h, w = f_mat.shape
    f_sym = np.zeros((2 * h, 2 * w))
    f_sym[:h, :w] = f_mat[::-1, ::-1]
    f_sym[:h, w:] = f_mat[::-1, :]
```

```
    f_sym[h:, :w] = f_mat[:, ::-1]
    f_sym[h:, w:] = f_mat
    return f_sym
```

1 Smooth and Sharpening

Relationship between frequency and spatial domain

In the DFT of an image, frequency is directly related to spatial rates of change. The low frequency component, which is located at the center of $F(u, v)$, corresponds to the slowly varying intensity component in the original image $f(x, y)$. As we move further away from the origin, the higher frequencies begin to correspond to faster and faster intensity changes in the image, such as the edges of objects in the image and some sharp characteristics.

1.1 Smooth Filter

Lowpass filter attenuates signals of high frequencies, which corresponds to the fast varying components in the original image. Thus we can use lowpass filter to smooth an image.

Ideal lowpass filter (ILPF) cuts off all frequencies outside a circle of radius from the origin. It can be expressed mathmatically as

$$H_{\mathrm{ILP}}(u, v) = I\left\{\left(u - \frac{P}{2}\right)^2 + \left(v - \frac{Q}{2}\right)^2 \leq D_0^2\right\};$$

Gaussian lowpass filter (GLPF) attentuates high frequencies according to a gaussian distribution centered at the origin. It can be expressed mathmatically as

$$D(u, v) = \sqrt{\left(u - \frac{P}{2}\right)^2 + \left(v - \frac{Q}{2}\right)^2},$$

$$H_{\mathrm{GLP}}(u, v) = \exp\left\{-\frac{D(u, v)^2}{2D_0^2}\right\};$$

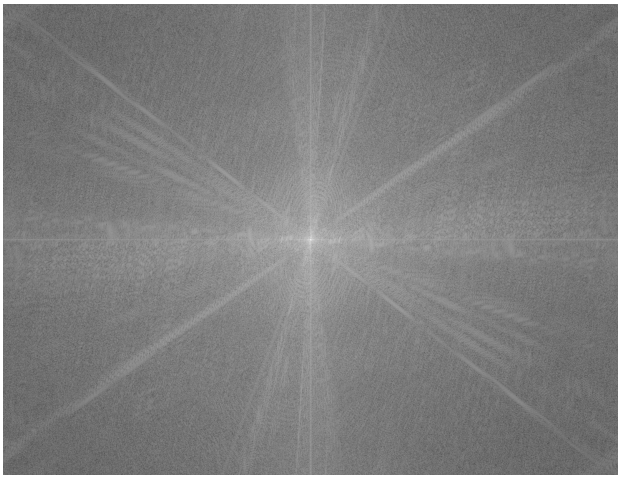
Code The code is shown below. We implement ILPF and GLPF.

```
def lowpass_ideal(p, q, radius=100):
    u_mat = np.array([np.arange(p // 2)] * (q // 2))
    v_mat = np.array([np.arange(q // 2)] * (p // 2))
    dist_sq = (u_mat ** 2).T + v_mat ** 2
    return symmetrize(dist_sq <= radius * radius)

def lowpass_gaussian(p, q, sd=100):
    u_mat = np.array([np.arange(p // 2)] * (q // 2))
    v_mat = np.array([np.arange(q // 2)] * (p // 2))
    dist_sq = (u_mat ** 2).T + v_mat ** 2
    return symmetrize(np.exp(-dist_sq / (2 * sd * sd)))
```

```
12
13 img = cv2.imread(args.img_source_path, 0)
14 h, w = img.shape
15
16 # Generate filter.
17 gauss_lp = lowpass_gaussian(2 * h, 2 * w, args.gauss_sd)
18 ideal_lp = lowpass_ideal(2 * h, 2 * w, args.radius)
19
20 # DFT.
21 img_ft = dft(img)
22
23 # Original spectrum.
24 spectrum = np.log(1 + np.abs(img_ft))  # Log-scale.
25 spectrum = spectrum / spectrum.max() * (nbins - 1) # Normalize.
26
27 # Smooth by gaussian filter.
28 img_sm1_ft = img_ft * gauss_lp
29
30 # New spectrum.
31 spectrum_sm1 = np.log(1 + np.abs(img_sm1_ft))
32 spectrum_sm1 = spectrum_sm1 / spectrum_sm1.max() * (nbins - 1)
33
34 # IDFT.
35 img_sm1 = idft(img_sm1_ft)
36
37 # Smooth by ideal filter.
38 img_sm2_ft = img_ft * ideal_lp
39
40 # New spectrum.
41 spectrum_sm2 = np.log(1 + np.abs(img_sm2_ft))
42 spectrum_sm2 = spectrum_sm2 / spectrum_sm2.max() * (nbins - 1)
43
44 # IDFT.
45 img_sm2 = idft(img_sm2_ft)
```

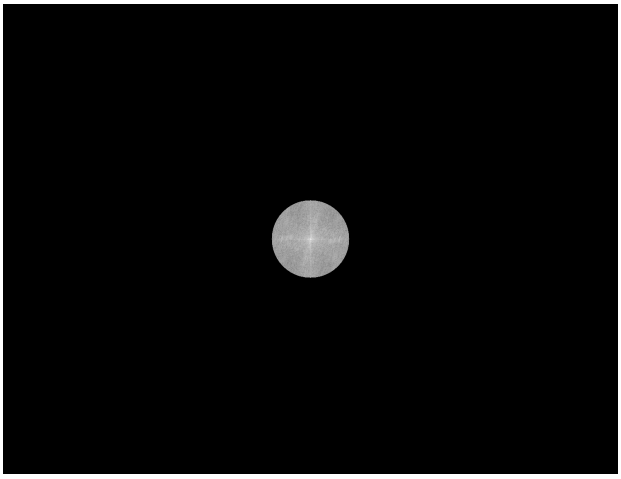
Result We test our code on image `suomi.jpg`, the result is shown below. We compared the spectrum of image before and after smoothing. It turns out that an ideal lowpass filter leads to ringing effect around objects in the image due to the vibrating property of *sinc* function. The Gaussian lowpass filter performs better, showing a blurring effect on the original image.

(a) suomi.jpg (Original)



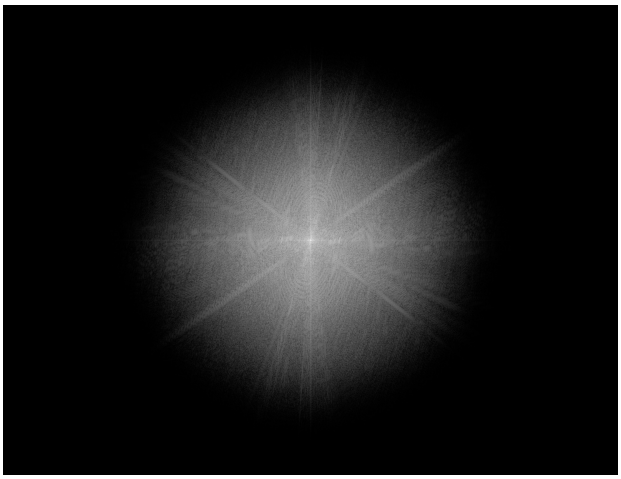
(b) Spectrum of suomi.jpg

(c) Smoothed by ideal lowpass filter ($D_0 = 100$)



(d) Lowpass spectrum (ideal)

(e) Smoothed by Gaussian lowpass filter ($D_0 = 100$)



(f) Lowpass spectrum (Gaussian)

1.2 Sharpening Filter

Highpass filter attenuates signals of low frequencies, which corresponds to the slowly varing components in the original image. This filter extracts the sharp characteristics. Similar to high boosting in the spatial domain, we can enhance the sharp characteristics in an image by adding the filtered image with certain weight k:

$$g(x,y) = \mathfrak{J}^{-1}\{[1 + kH_{\text{HP}}]F(u,v)\};$$

Thus we can sharpen an image by highpass filter.

Ideal highpass filter (IHPF) cuts off all frequencies within a circle of radius from the origin. It can be expressed mathmatically as

$$H_{\text{IHP}}(u,v) = I\left\{\left(u - \frac{P}{2}\right)^2 + \left(v - \frac{Q}{2}\right)^2 > D_0^2\right\} = 1 - H_{\text{ILP}}(u,v);$$

Gaussian highpass filter (GHPF) attentuates high frequencies according to a gaussian distribution centered at the origin. It can be expressed mathmatically as

$$H_{\text{GHP}}(u,v) = 1 - \exp\left\{-\frac{D(u,v)^2}{2D_0^2}\right\} = 1 - H_{\text{GLP}};$$

Laplacian filter in frequency domain Like the Laplacian filter in spatial domain, we can also use the Laplacian filter in frequency domain to extract the information. Suppose $F(u,v) = \mathfrak{J}\{f(x,y)\}$, it implies

$$\mathfrak{J}\left\{\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}\right\} = \left[\left(-2\pi\frac{u}{M}\right)^2 + \left(-2\pi\frac{v}{N}\right)^2\right]F(u,v),$$

Thus the Laplacian operator in frequency domain is

$$H_{\text{Laplacian}}(u,v) = 4\pi^2\left(\frac{u^2}{M^2} + \frac{v^2}{N^2}\right);$$

And we can sharpen the image by

$$g(x,y) = \mathfrak{J}^{-1}\{[1 - wH_{\text{Laplacian}}(u,v)]F(u,v)\}.$$

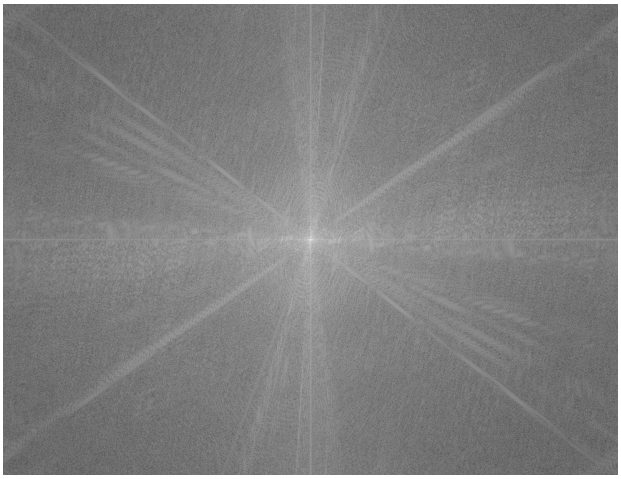
Code The code is shown below. We implement Laplacian filter and GHPF.

```
def highpass_gaussian(p, q, sd=100):
    return 1 - lowpass_gaussian(p, q, sd)

def laplacian(p, q):
    u_mat = np.array([np.arange(p // 2)] * (q // 2)) / p
    v_mat = np.array([np.arange(q // 2)] * (p // 2)) / q
    dist_sq = (u_mat ** 2).T + v_mat ** 2
    return - symmetrize(4 * np.pi**2 * dist_sq)

img = cv2.imread(args.img_source_path, 0)
h, w = img.shape
```

```
12
13 # Generate filters.
14 laplace_f = laplacian(2 * h, 2 * w)
15 gauss_hp = highpass_gaussian(2 * h, 2 * w, args.gauss_sd)
16
17 # DFT.
18 img_ft = dft(img)
19
20 # Original spectrum.
21 spectrum = np.log(1 + np.abs(img_ft))  # Log-scale.
22 spectrum = spectrum / spectrum.max() * (nbins - 1)  # Normalize.
23
24 # Sharpen by Laplacian filter.
25 img_sh1_ft = img_ft * (1 - args.weight * laplace_f)
26 spectrum_sh1 = np.log(1 + np.abs(img_sh1_ft))
27 spectrum_sh1 = spectrum_sh1 / spectrum_sh1.max() * (nbins - 1)
28
29 # IDFT and cut off the pixels out of range [0,255].
30 img_sh1 = idft(img_sh1_ft)
31 img_sh1[img_sh1 >= nbins] = nbins - 1
32 img_sh1[img_sh1 < 0] = 0
33
34 # High-boost by Gaussian filter.
35 img_sh2_ft = img_ft * (1 + args.weight * gauss_hp)
36 spectrum_sh2 = np.log(1 + np.abs(img_sh2_ft))
37 spectrum_sh2 = spectrum_sh2 / spectrum_sh2.max() * (nbins - 1)
38
39 # IDFT and cut off the pixels out of range [0,255].
40 img_sh2 = idft(img_sh2_ft)
41 img_sh2[img_sh2 >= nbins] = nbins - 1
42 img_sh2[img_sh2 < 0] = 0
```

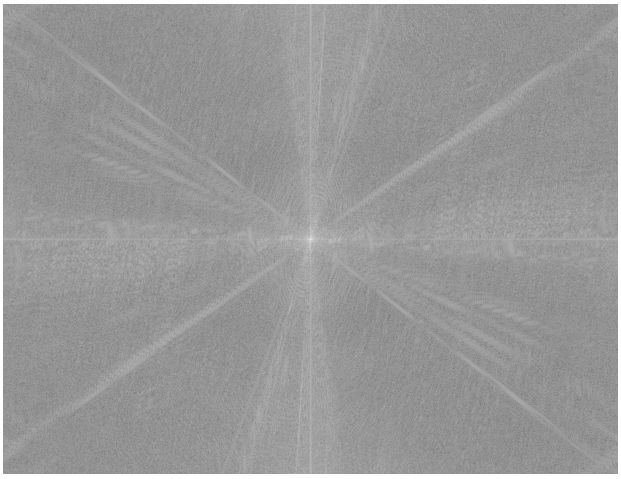
Result We test our code on images `suomi.jpg` and `moon.jpg`, the results are shown below. We compared the spectrum of image before and after smoothing. It's observed that after sharpening, signals of high frequencies located at outer part of the spectrum are enhanced. Meanwhile, it turns out that both Laplacian filter and highboosting perform well in sharpening the original image.

(g) suomi.jpg (Original)



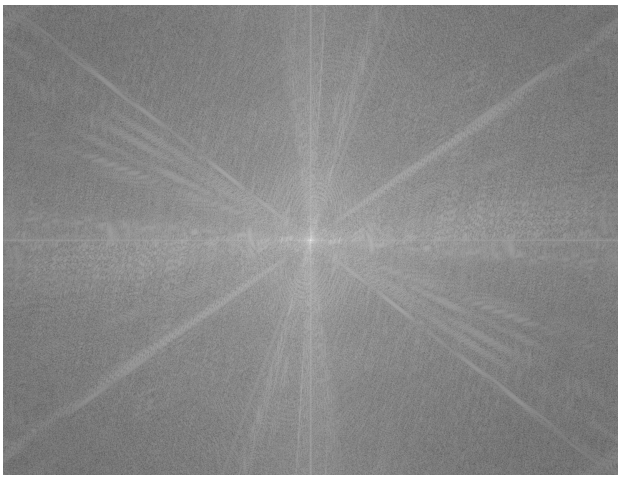
(h) Spectrum of suomi.jpg

(i) Laplacian sharpened ($w = 1$)



(j) Sharpened spectrum (Laplacian)

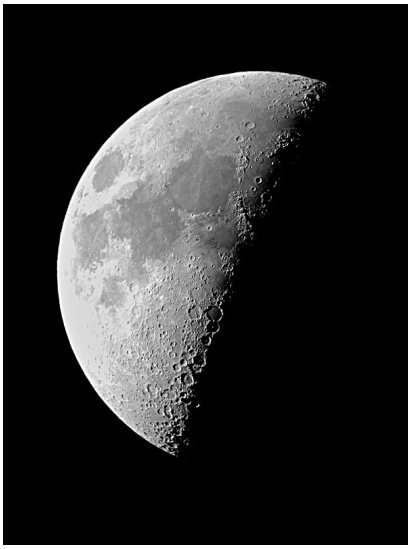
(k) Gaussian sharpened ($D_0 = 100, k = 1$)



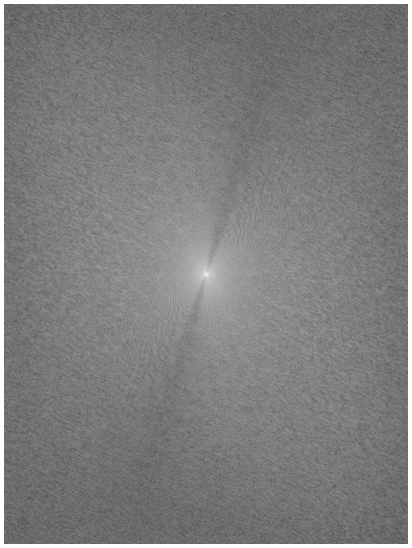
(l) Sharpened spectrum (Gaussian)

(m) suomi.jpg (Original)

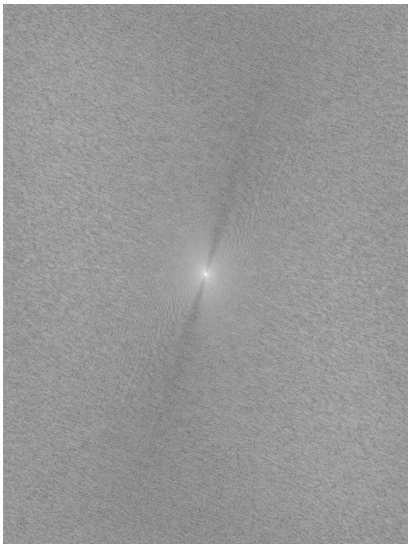
(n) Laplacian sharpened ($w = 1$)



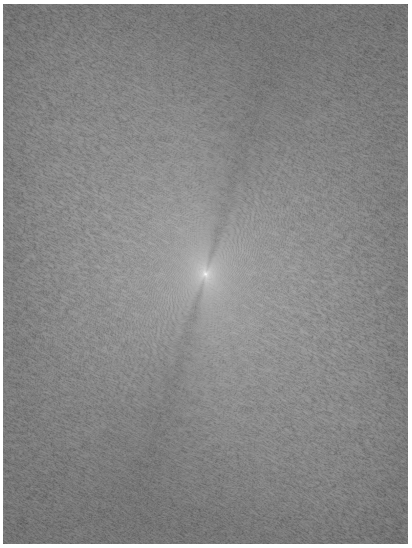
(o) Highboosted ($D_0 = 100, k = 1$)



(p) Spectrum of suomi.jpg



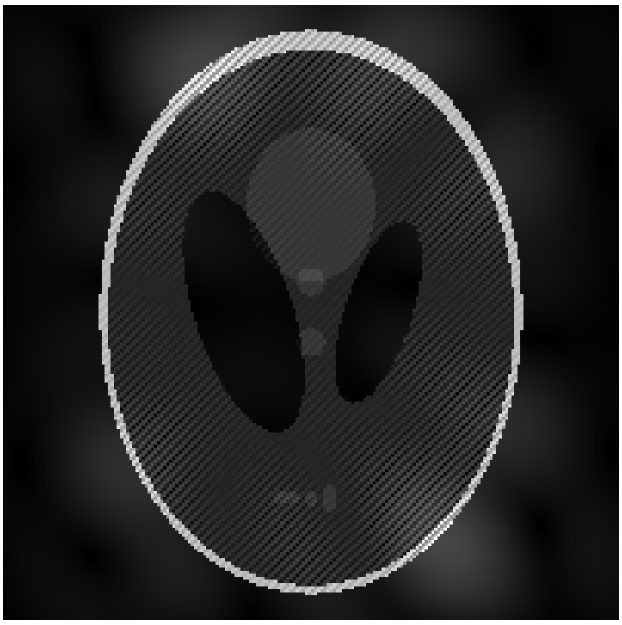
(q) Sharpened spectrum (Laplacian)



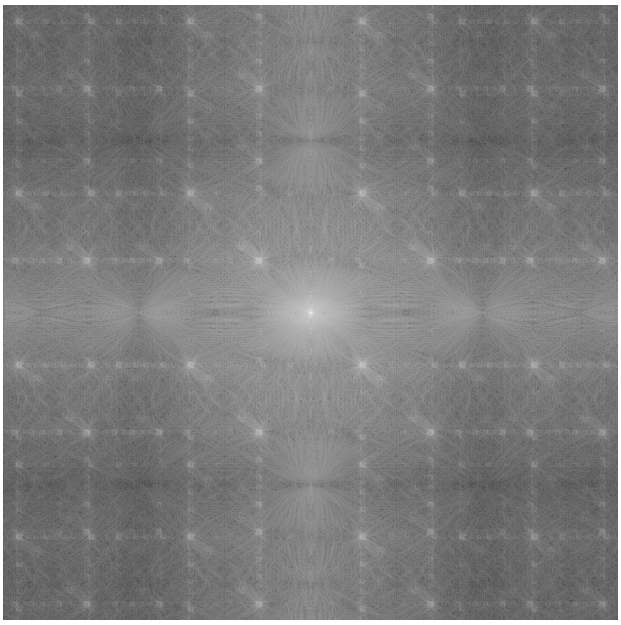
(r) Sharpened spectrum (Gaussian)

2 Reduction of Periodic Noise

Consider the image `Shepp-Logan.png` shown below, there is some periodic stripe-like noise distributed within the CT scanning result of a brain. Our goal is to reduce the stripe-like noise. As the noise in the image is periodically distributed with some regularity, it may correspond to some certain components in the frequency domain. Thus we run DFT on the image to get its spectrum for analysis. The spectrum is shown below, it has been log-scaled to enhance the contrast.



(s) Shepp-Logan.png (Original)



(t) Spectrum of Shepp-Logan.png

We notice that there are some abnormal light spots in the spectrum, which may correspond to some components in the CT photo. The stripe-like noise is distributed diagonally, and changes from top left to bottom right. The direction of signal in the frequency domain accord with that of change rate in the original image. Thus we can cut off the signals in the diagonal direction.
Here we use a notch filter to remove the light spot in the spectrum:

$$H_{\text{notch}}(u,v) = \prod_{i=1}^{k} H_i(u,v) H_{-i}(u,v),$$

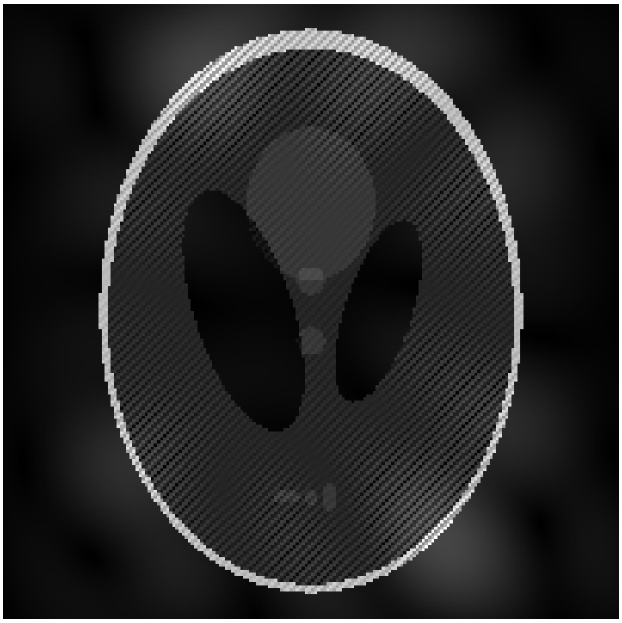
where H_i is an ideal highpass filter centered at the removed spot (u_i, v_i) with predetermined radius r, and H_{-i} centered at $(-u_i, -v_i)$, $i = 1, \cdots, k$, the origin is at the center of the image;

Code The code is shown below.

```
def point_notch(img_freq, u0, v0, radius):
    # Remove a small circle at (u0, v0) from the spectrum.
    # It's equivalent to ideal highpass filter centered at (u0, v0).
    u_mat = np.array([np.arange(img_freq.shape[0]) - u0] * (img_freq.shape[1]))
    v_mat = np.array([np.arange(img_freq.shape[1]) - v0] * (img_freq.shape[0]))
    dist_sq = (u_mat ** 2).T + v_mat ** 2
```

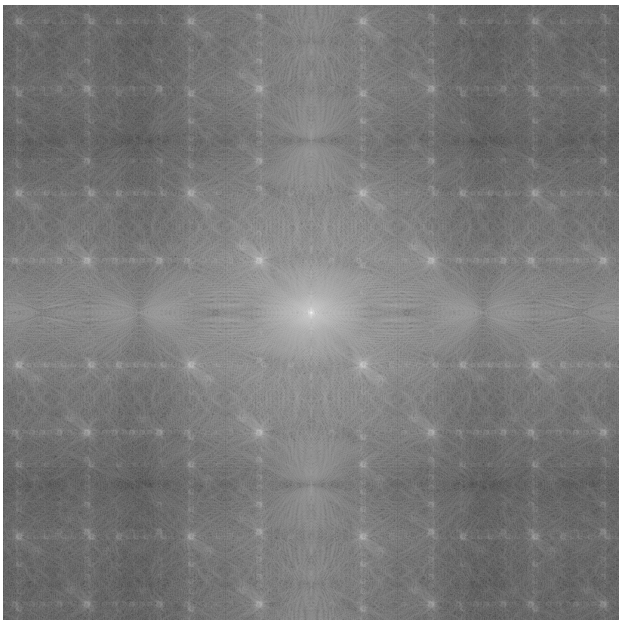
```
    return dist_sq >= radius * radius

def notch_filter(img_freq):
    h, w = img_freq.shape
    notch = np.ones_like(img_freq)
    centers = [120, 280, 520, 680]
    for uc in centers:
    for vc in centers:
    notch *= point_notch(img_freq, h // 2 - uc, w // 2 - vc, 12)
    notch *= point_notch(img_freq, h // 2 + uc, w // 2 + vc, 12)
    return img_freq * notch

img = cv2.imread(args.img_source_path, 0)

# DFT.
img_ft = dft(img)
spectrum = np.log(1 + np.abs(img_ft))
spectrum = spectrum / np.max(spectrum) * 255

# Noise reduction.
img_ft_notch = notch_filter(img_ft)
spectrum_notch = np.log(1 + np.abs(img_ft_notch))
spectrum_notch = spectrum_notch / np.max(spectrum_notch) * 255

# IDFT.
img_re = idft(img_ft_notch)
```

Result It turns out to perform well by removing the light spot in the spectrum. The result is shown below.

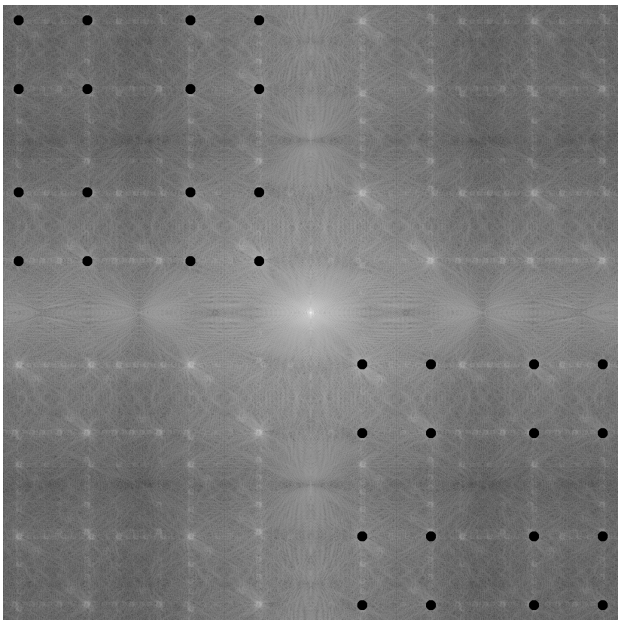


(u) Shepp-Logan.png (Original)

(v) Shepp-Logan.png after noise-rescution



(w) Spectrum of Shepp-Logan.png



(x) Notched spectrum

The spectrum before and after noise reduction is shown at the right side. We shielded the abnormal light spot along the diagonal direction from top left to bottom right. It reduces the some components at a certain change rate along the diagonal direction, including the stripe-like noise. We can observe that the stripe-like noise in the original image is reduced in a considerable scale by the notch filter, with high fidelity to the non-noise components.

3 Appendix

3.1 Libraries

We use the `numpy` library to support the large-scale computation of matrices which store the grey level information of images.
We use `opencv` library (`cv2`) to read the input images and save the output images.
We use `matplotlib` to show images before and after processing.
We also use `argparse` to read in the arguments for processing, which can be specified in the command line.

3.2 IDE

All our experiments are done on PyCharm Community Edition 2022.1.2.