

数字图像处理及应用 第3次作业

组号: 16 小组成员: 冯坤龙 郝锦阳 朱从庆 辛梓阳 徐振良

A

Part I Exercises

Ex 4.12 Consider a checkerboard image in which each square is 0.5×0.5 mm. Assuming that the image extends infinitely in both coordinate directions, what is the minimum sampling rate (in samples/mm) required to avoid aliasing?

Answer:

Taking the horizontal direction as an example, we can know that the period of a checkerboard image is equal to $T = 1$ mm by viewing two squares, one black and one white, as a period. According to Nyquist sampling theorem, a period has at least two sampling points. Therefore, to avoid aliasing the minimum sampling rate is $\frac{2}{T}$ (samples/mm) = 2(samples/mm).

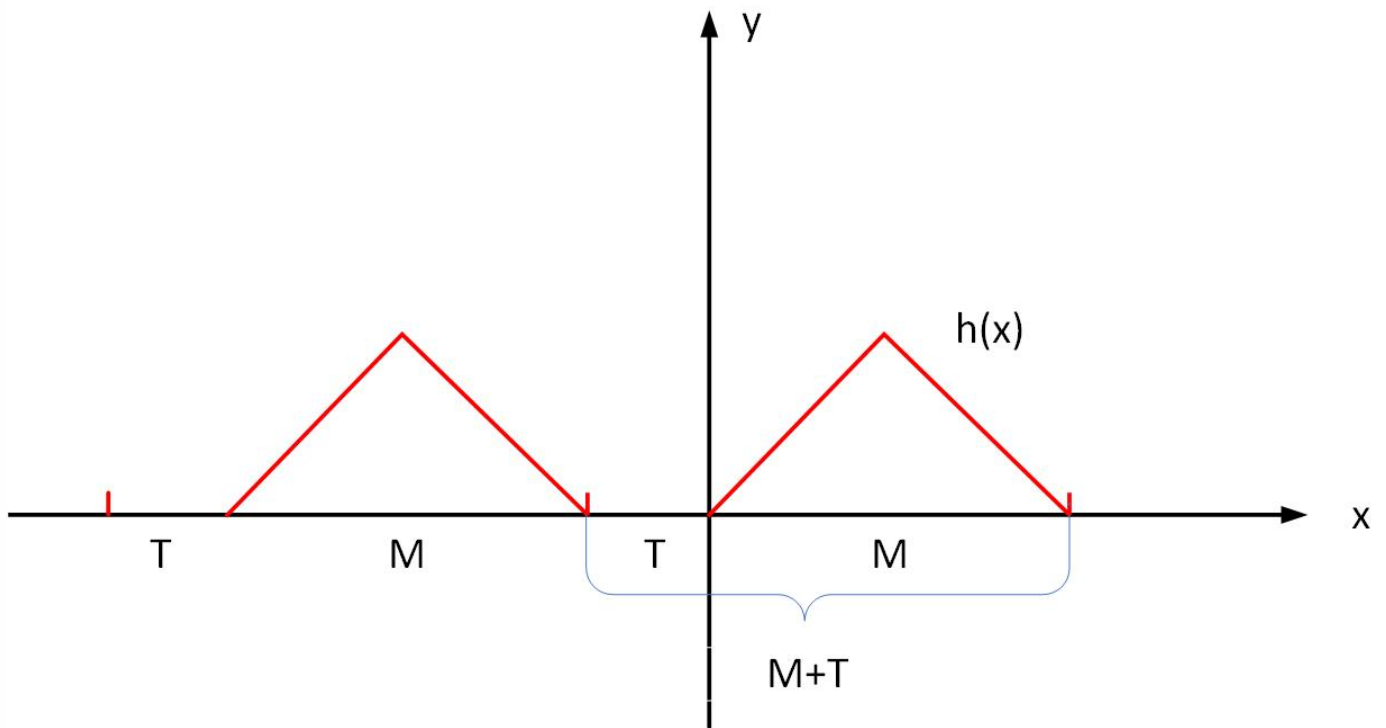
Ex 4.21 The need for image padding when filtering in the frequency domain was discussed in Section 4.6.6. We showed in that section that images needed to be padded by appending zeros to the ends of rows and columns in the image (see the following image on the left). Do you think it would make a difference if we centered the image and surrounded it by a border of zeros instead (see image on the right), but without changing the total number of zeros used? Explain.



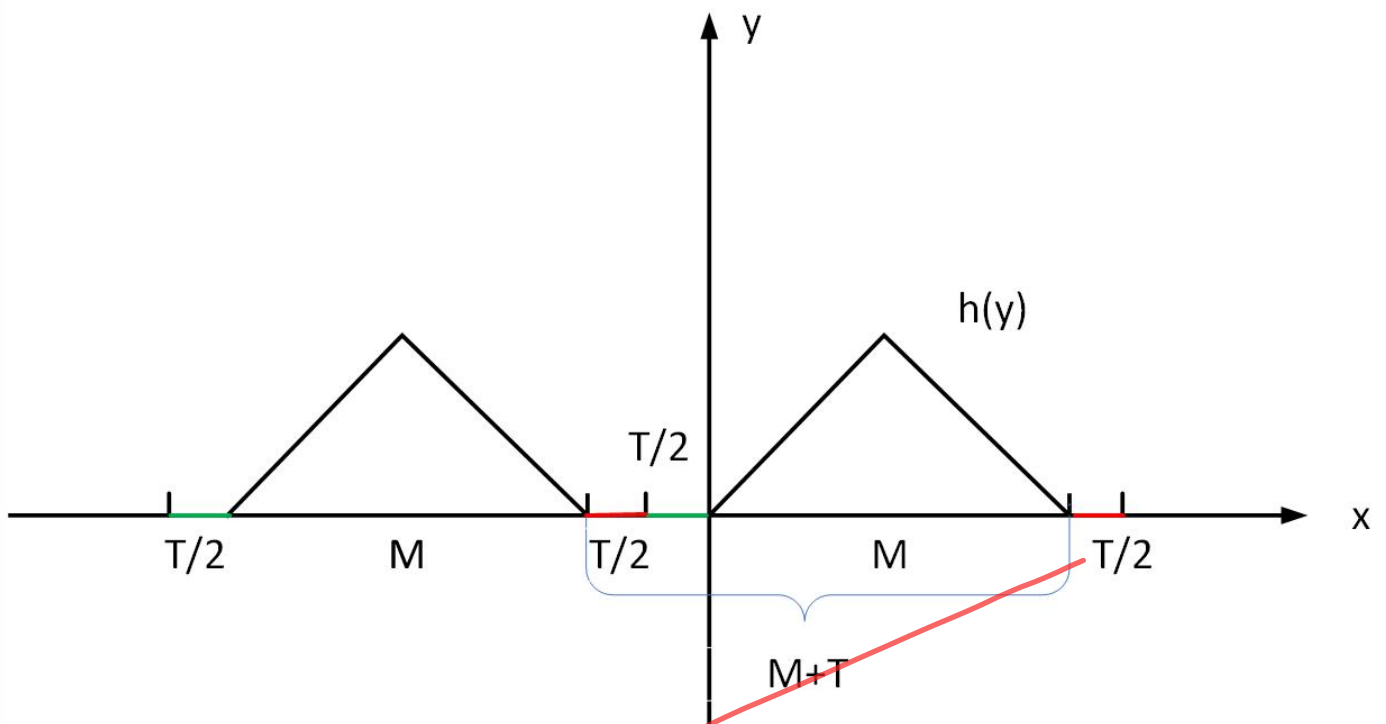
Answer:

It is known that complementing 0 for an image $f(x, y)$ of size $A \times B$ is to prevent errors caused by multiplying $h(x, y)$ of size $C \times D$ with $f(x, y)$ of the previous period when convolution is flipped, so it is necessary to complement the image size to $(A + C, B + D)$ to ensure sufficient space for convolution. For this problem, Take a one-dimensional function as an example, let the period of $m(x, y)$ be M , the period of $t(x, y)$ be T , and the period after taking the complement of 0 be $M + T$

As shown in the figure, if you fill the end with 0, you can leave $M + T$ space for the convolution



If 0 is filled on both sides, the resulting image is the same as that obtained by filling 0 at the end, leaving a space of size $M + T$ to satisfy the convolution condition



So expanding to two dimensions and using different kinds of padding gives you the same result

Ex 4.26

(a) Show that the Laplacian of a continuous function $f(t, z)$ of continuous variables t and z satisfies the following Fourier transform pair [see Eq. (3.6-3) for a definition of the Laplacian]:

$$\nabla^2 f(t, z) \iff -4\pi^2(\mu^2 + v^2)F(\mu, v)$$

(b) The preceding closed form expression is valid only for continuous variables. However, it can be the basis for implementing the Laplacian in the discrete frequency domain using the $M \times N$ filter

$$H(u, v) = -4\pi^2(u^2 + v^2)$$

for $u = 0, 1, 2, \dots, M - 1$ and $v = 0, 1, 2, \dots, N - 1$. Explain how you would implement this filter.

(c) As you saw in Example 4.20, the Laplacian result in the frequency domain was similar to the result of using a spatial mask with a center coefficient equal to -8. Explain the reason why the frequency domain result was not similar instead to the result of using a spatial mask with a center coefficient of -4. See Section 3.6.2 regarding the Laplacian in the spatial domain.

Answer:

(a) We know that the definition of the Laplace operator is

$$\nabla^2 f(t, z) = \partial^2 f(t, z) / \partial t^2 + \partial^2 f(t, z) / \partial z^2$$

By the property of the Fourier transform

$$\begin{aligned} \nabla^2 f(t, z) &\iff (j2\pi u)^2 F(u, v) + (j2\pi v)^2 F(u, v) \\ &= -4\pi^2(u^2 + v^2)F(u, v) = H(u, v)F(u, v) \end{aligned}$$

(b) The filter can be taken points and discretized

$$H_0(u, v) = -4\pi^2(u^2 + v^2)$$

$$u = 0, 1, 2 \dots M - 1$$

$$v = 0, 1, 2 \dots N - 1$$

Then the filter is centralized

$$H_1(u, v) = H_0(u - M/2, v - N/2)$$

$$\nabla^2 f(t, z) \iff H_1(u, v)F(u, v)$$

$$u = 0, 1, 2 \dots M - 1$$

$$v = 0, 1, 2 \dots N - 1$$

(c) Because the Laplacian operator in the frequency domain is similar to the spatial template with a center coefficient of -8, it also has a sharpening effect in the diagonal direction, so it is different from the spatial template with a center coefficient of -4

Ex 4.28 Based on Eq. (3.6-4), one approach for approximating a discrete derivative in 2-D is based on computing differences of the form $f(x + 1, y) + f(x - 1, y) - 2f(x, y)$ and $f(x, y + 1) + f(x, y - 1) - 2f(x, y)$.

(a) Find the equivalent filter, $H(u, v)$, for Eq.(3.6-6) in the frequency domain.

$$\nabla^2 f(x, y) = f(x + 1, y) + f(x - 1, y) + f(x, y + 1) + f(x, y - 1) - 4f(x, y) \quad (3.6-6)$$

(b) Show that your result is a highpass filter.

Answer:

(a) Let's assume

$$t(x, y) = f(x + 1, y) + f(x - 1, y) + f(x, y + 1) + f(x, y - 1) - 4f(x, y)$$

By the property of the Fourier transform

$$\begin{aligned} T(u, v) &= F(u, v)(e^{\frac{j2\pi u}{M}} + e^{\frac{-j2\pi u}{M}} + e^{\frac{j2\pi v}{N}} + e^{\frac{-j2\pi v}{N}} - 4) \\ &= F(u, v)H(u, v) \end{aligned}$$

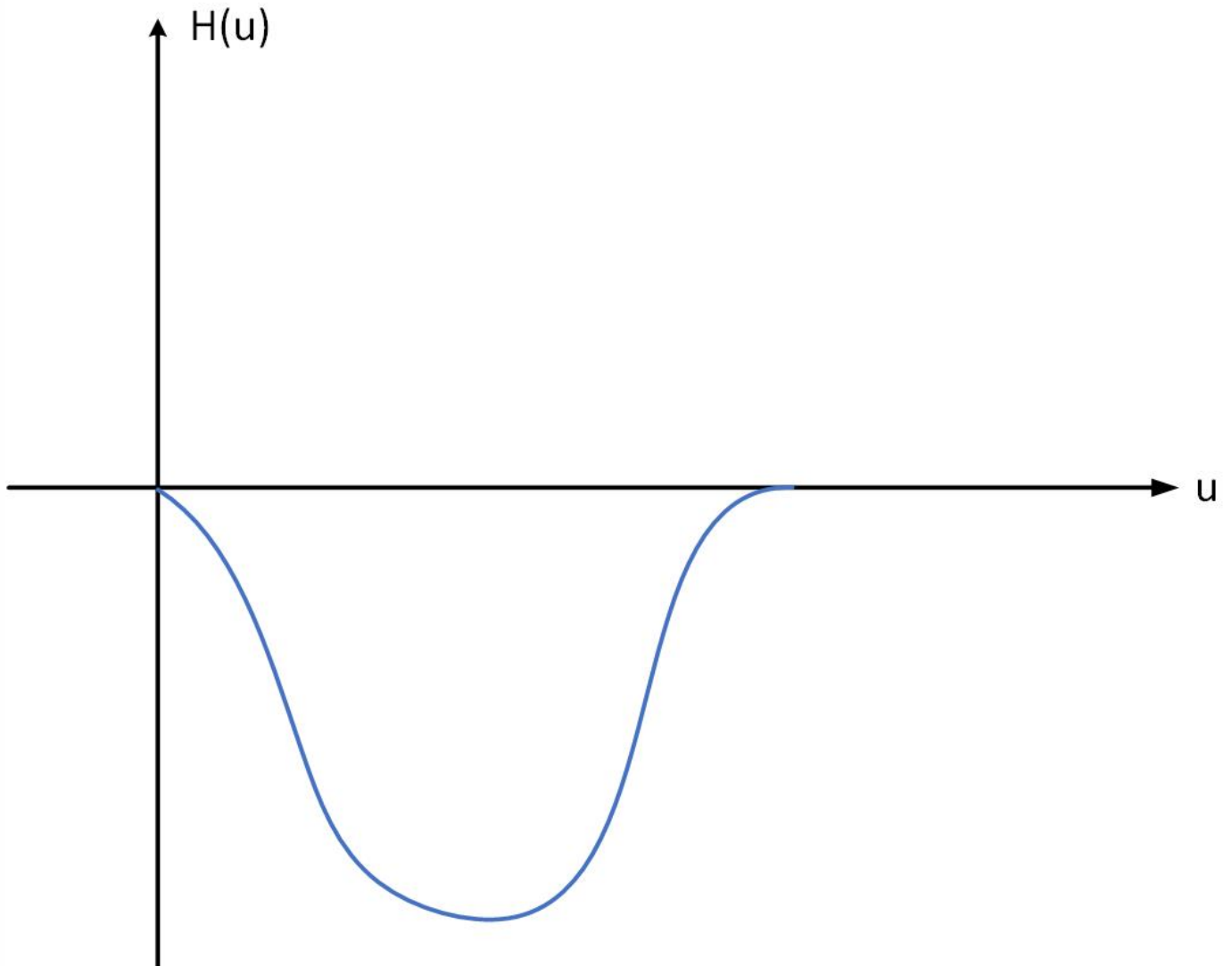
So we can get

$$H(u, v) = (e^{\frac{j2\pi u}{M}} + e^{\frac{-j2\pi u}{M}} + e^{\frac{j2\pi v}{N}} + e^{\frac{-j2\pi v}{N}} - 4)$$

It's given by Euler's formula

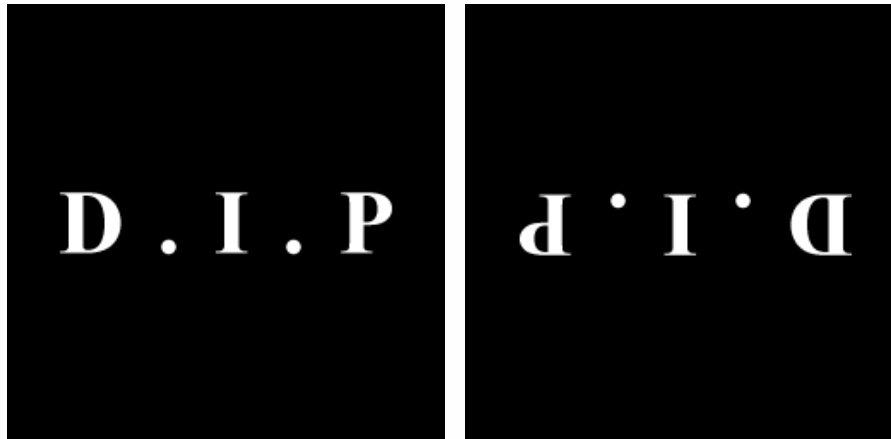
$$H(u, v) = 2[\cos(\frac{2\pi u}{M}) + \cos(\frac{2\pi v}{N}) - 2]$$

(b) You can sketch the filter roughly, using the u direction as an example



Therefore, the same conclusion can be obtained in the v direction, and it can be seen that the filter conforms to the high-pass filter

Ex 4.33 Consider the images shown. The image on the right was obtained by: (a) multiplying the image on the left by $(-1)^{x+y}$; (b) computing the DFT; (c) taking the complex conjugate of the transform; (d) computing the inverse DFT; and (e) multiplying the real part of the result by $(-1)^{x+y}$. Explain (mathematically) why the image on the right appears as it does.



Answer:

Let's assume

$$h(x, y) = f(x, y) \times (-1)^{x+y}$$

Then there is

$$h(x, y) \iff H(u, v)$$

And because $h(x, y)$ is a real function

$$h(-x, -y) \iff H^*(u, v)$$

(This is given by the symmetry of the DFT)

So we get

$$h(-x, -y) \times (-1)^{x+y} = f(-x, -y) \times (-1)^{-(x+y)} \times (-1)^{x+y} = f(-x, -y)$$

Therefore, the phenomenon shown in the figure will occur

Ex 4.37 Given an image of size $M \times N$ you are asked to perform an experiment that consists of repeatedly lowpass filtering the image using a Gaussian lowpass filter with a given cutoff frequency D_0 . You may ignore computational round-off errors. Let c_{min} denote the smallest positive number representable in the machine in which the proposed experiment will be conducted.

(a) Let K denote the number of applications of the filter. Can you predict (without doing the experiment) what the result (image) will be for a sufficiently large value of K ? If so, what is that result?

(b) Derive an expression for the *minimum* value of K that will guarantee the result that you predicted.

Answer:

(a) By Gaussian filter Expression

$$H(u, v) = e^{-\frac{D^2(u, v)}{2D_0^2}},$$

We assume that the original image is

$$f(x, y) \iff F(u, v)$$

Filter the image by once, we can get

$$T(u, v) = H(u, v)F(u, v) = e^{-\frac{D^2(u, v)}{2D_0^2}} F(u, v)$$

Filter the image by K times, we can get

$$G(u, v) = [H(u, v)]^k F(u, v) = e^{-\frac{KD^2(u, v)}{2D_0^2}} F(u, v)$$

When K is large enough, we can only remain the lowest frequency part of the image $F(0, 0)$

and its size is the average size of the entire image.

(b) by result in (a)

$$[H(u, v)]^k = e^{-\frac{KD^2(u, v)}{2D_0^2}}$$

when $K \rightarrow \infty$

$$[H(u, v)]^k = \begin{cases} 1, & (u, v) = (0, 0) \\ 0, & else \end{cases}$$

due to $Cmin$ is the smallest positive number representable in the machine and u, v are both positive integers

so we can get

$$e^{-\frac{KD^2(u, v)}{2D_0^2}} < 0.5Cmin$$

$$\Rightarrow K > \frac{-2D_0^2 \ln(0.5Cmin)}{D^2(u, v)}$$

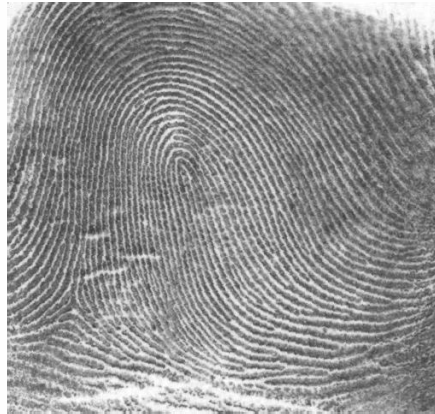
$$\Rightarrow K_{min} = \left[\frac{-2D_0^2 \ln(0.5Cmin)}{D^2(u, v)} \right]_{max}$$

$$= \frac{-2D_0^2 \ln(0.5Cmin)}{1}$$

$$= -2D_0^2 \ln(0.5Cmin)$$

Part II Programming

1. Implement Example 4.19 in page 308.



(followed by *Matlab live Scripts* or *Jupyter Scripts* and running results)

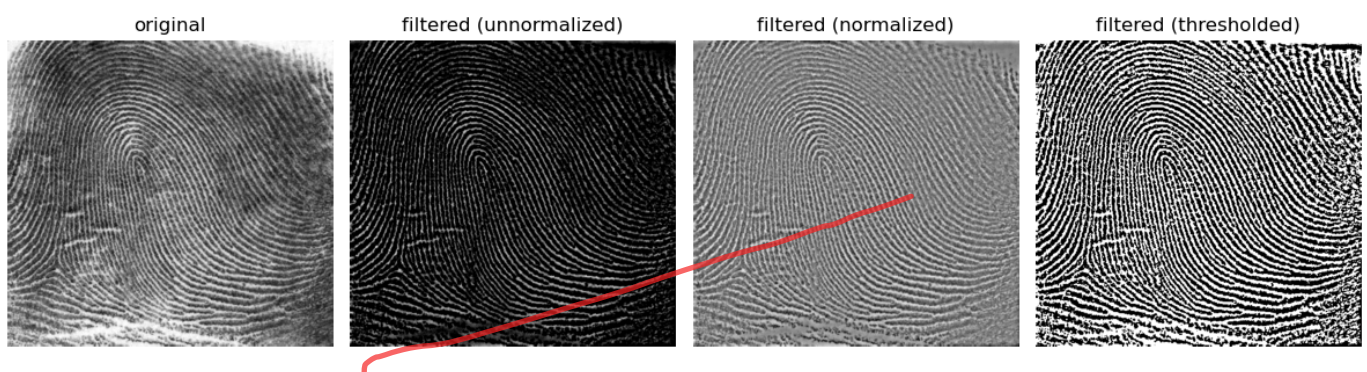
```
1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5
6 def butterHP(D, n, shape):
7     """
8     :param D: cutoff frequency
9     :param n: order
10    :param shape: (rows, columns)
11    :return: array H with the same shape
12    """
13    H = np.zeros(shape, dtype=np.float64)
14    row_c, col_c = np.floor(shape[0] / 2), np.floor(shape[1] / 2)
15    for u in range(shape[0]):
16        for v in range(shape[1]):
17            d = np.sqrt(((u - row_c) ** 2) + ((v - col_c) ** 2))
18            H[u, v] = 1 / (1 + np.power(D / (0.00000000001 + d), 2 * n))
19    return H
20
21
22 # read the original image
23 original_img = cv2.imread("../images/Fig0457(a)(thumb_print).png", flags=0)
24
25 # calculate the 2D DFT of the original image using `np.fft.fft2()`
26 original_frq = np.fft.fft2(original_img)
27
28 # centralize DFT
29 centred_original_frq = np.fft.fftshift(original_frq)
30
```

```

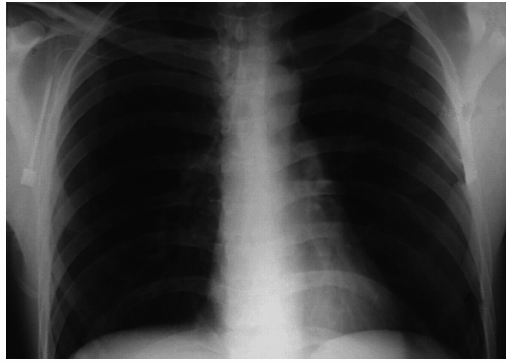
31 # construct a Butterworth Highpass Filter
32 BHPF = butterHP(D=25, n=4, shape=original_img.shape)
33
34 # apply the Butterworth Highpass Filter
35 centred_filtered_freq = centred_original_freq * BHPF
36
37 # inverse the 2D DFT
38 filtered_freq = np.fft.ifftshift(centred_filtered_freq)
39 filtered_img = np.real(np.fft.ifft2(filtered_freq))
40 filtered_img_cutoff = filtered_img.copy()
41 filtered_img_cutoff[filtered_img_cutoff < 0] = 0
42
43 # threshold the filtered image setting negative values to 0, positive values to 1
44 threshold_img = filtered_img.copy()
45 threshold_img[threshold_img < 0] = 0
46 threshold_img[threshold_img > 0] = 1
47
48 # display the results
49 plt.figure("Butterworth Highpass Filtering", (12, 4))
50 plt.subplot(141), plt.imshow(original_img, 'gray'), plt.title("original"),
    plt.axis('off')
51 plt.subplot(142), plt.imshow(filtered_img_cutoff, 'gray'), plt.title("filtered
    (unnormalized)"), plt.axis('off')
52 plt.subplot(143), plt.imshow(filtered_img, 'gray'), plt.title("filtered
    (normalized)"), plt.axis('off')
53 plt.subplot(144), plt.imshow(threshold_img, 'gray'), plt.title("filtered
    (thresholded)"), plt.axis('off')
54 plt.suptitle(f"Running Results of Example 4.19 Butterworth Highpass Filtering")
55
56 plt.tight_layout()
57 plt.show()
58

```

Running Results of Example 4.19 Butterworth Highpass Filtering



2. Implement Example 4.21 in page 311.



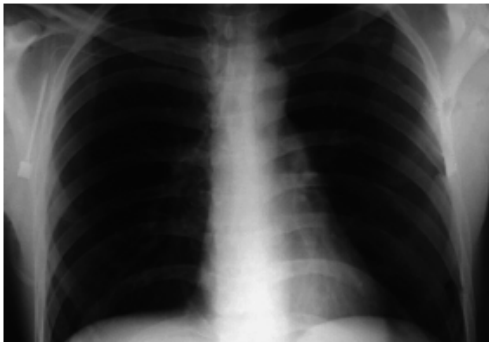
(followed by *Matlab live Scripts* or *Jupyter Scripts* and running results)

```
1 import cv2
2 import numpy
3 import matplotlib.pyplot as plt
4
5 original_image = cv2.imread("../images/Fig0459(a)(orig_chest_xray).png", flags=0)
6
7 # FFT and shift
8 original_frq = numpy.fft.fft2(original_image)
9 centred_original_frq = numpy.fft.fftshift(original_frq)
10
11 # construct a Gaussian Highpass Filter
12 rows, cols = original_image.shape # rows cols分别得到图像的行数和列数
13 row_centre, col_centre = rows // 2, cols // 2 # 得到图像中心点的坐标 (crow, cols)
14 D0 = 70
15 GHPF = numpy.zeros((rows, cols))
16 for u in range(rows):
17     for v in range(cols):
18         D2 = (u - row_centre) ** 2 + (v - col_centre) ** 2
19         GHPF[u, v] = 1 - numpy.exp(-D2 / (2 * D0 * D0))
20
21 centred_filtered_frq = centred_original_frq * GHPF
22 filtered_frq = numpy.fft.ifftshift(centred_filtered_frq)
23 filtered_img = numpy.real(numpy.fft.ifft2(filtered_frq))
24
25 # high-frequency-emphasis filtering using the same filter
26 k1 = 0.5
27 k2 = 0.75
28 centred_enhanced_filtered_freq = numpy.zeros((rows, cols),
29 dtype=numpy.complex128)
30 for u in range(rows):
31     for v in range(cols):
32         centred_enhanced_filtered_freq[u, v] = (k1 + k2 * GHPF[u, v]) *
33         centred_original_frq[u, v]
34 enhanced_filtered_frq = numpy.fft.ifftshift(centred_enhanced_filtered_freq)
35 enhanced_filtered_img = numpy.real(numpy.fft.ifft2(enhanced_filtered_frq))
36
37 # performing histogram equalization
38 equalized_img = cv2.equalizeHist(enhanced_filtered_img.astype(numpy.uint8))
39
40 # display the results
```

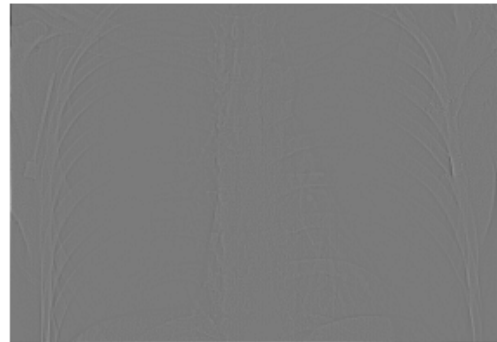
```
39 plt.figure("High-Frequency-Emphasis Filtering", (8, 8))
40 plt.subplot(221), plt.imshow(original_image, 'gray'), plt.title("original"),
    plt.axis('off')
41 plt.subplot(222), plt.imshow(filtered_img, 'gray'), plt.title("filtered"),
    plt.axis('off')
42 plt.subplot(223), plt.imshow(enhanced_filtered_img, 'gray'), plt.title("enhanced
    filtered"), plt.axis('off')
43 plt.subplot(224), plt.imshow(equalized_img, 'gray'), plt.title("equalized"),
    plt.axis('off')
44 plt.suptitle(f"Running Results of Example 4.21 High-Frequency-Emphasis
    Filtering")
45
46 plt.tight_layout()
47 plt.show()
48
```

Running Results of Example 4.21 High-Frequency-Emphasis Filtering

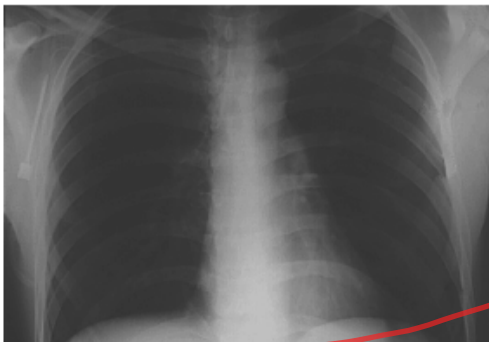
original



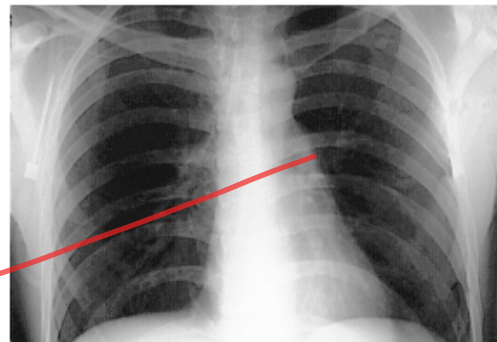
filtered



enhanced filtered



equalized



3. Implement Example 4.22 in Page 315.

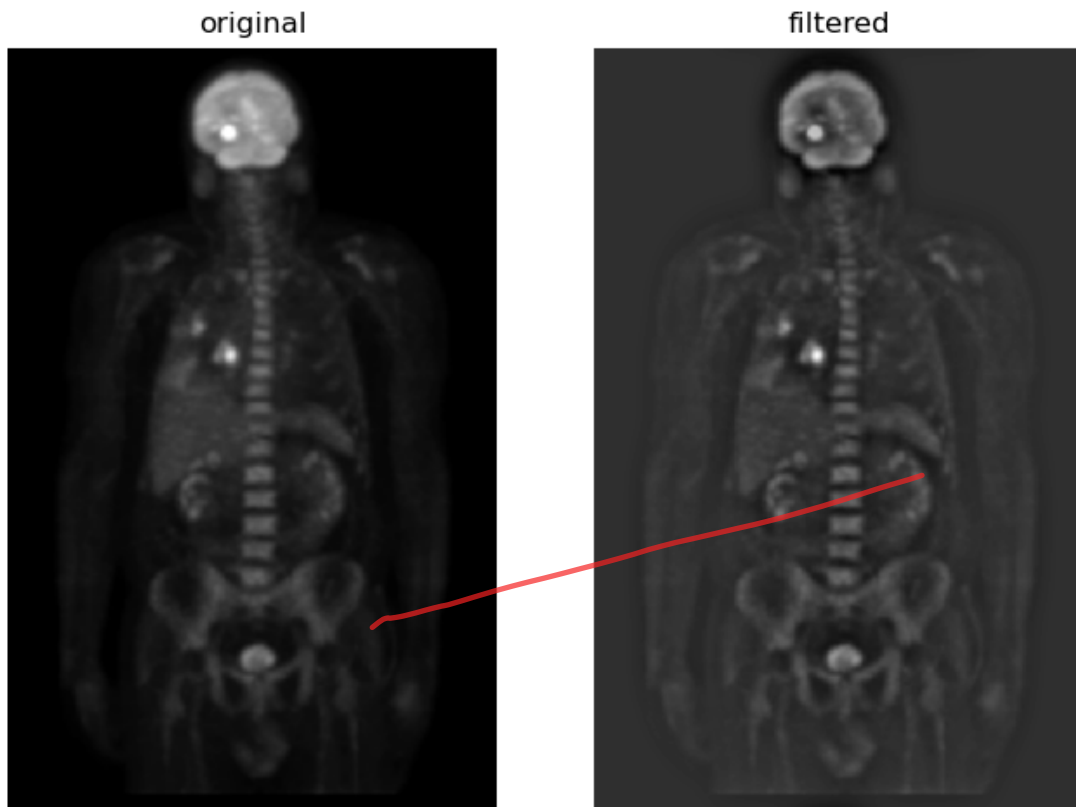


(followed by *Matlab live Scripts* or *Jupyter Scripts* and running results)

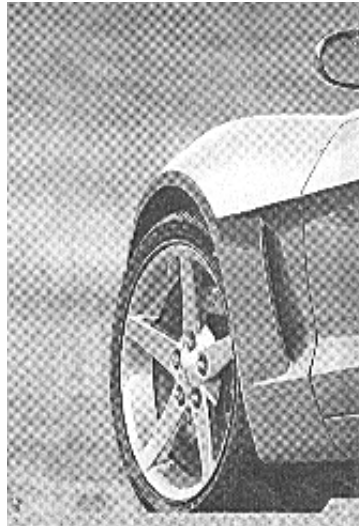
```
1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5
6 def homo_filter(gamma_L, gamma_H, c, D, shape):
7     H = np.zeros(shape, dtype=np.float64)
8     row_c, col_c = np.floor(shape[0] / 2), np.floor(shape[1] / 2)
9     D_sqr = D ** 2
10    for u in range(shape[0]):
11        for v in range(shape[1]):
12            d_sqr = ((u - row_c) ** 2) + ((v - col_c) ** 2)
13            H[u, v] = ((gamma_H - gamma_L) * (1 - np.exp(-c * d_sqr / D_sqr))) +
gamma_L
14    return H
15
16
17 # define constants
18 GAMMA_L = 0.4
19 GAMMA_H = 3.0
20 SLOPE = 5
21 CUTOFF_FREQUENCY = 20
22
23 # read the original image
24 original_img = cv2.imread("../images/Fig0462(a)(PET_image).png", flags=0)
25
26 # calculate the 2D DFT of the original image using `np.fft.fft2()`
27 original_frq = np.fft.fft2(original_img)
28
29 # centralize DFT
30 centred_original_frq = np.fft.fftshift(original_frq)
31
32 # construct a Homomorphic Filter
33 HF = homo_filter(GAMMA_L, GAMMA_H, c=SLOPE, D=CUTOFF_FREQUENCY,
shape=original_img.shape)
34
35 # apply the Homomorphic Filter
```

```
36 centred_filtered_frq = centred_original_frq * HF
37
38 # inverse the 2D DFT
39 filtered_frq = np.fft.ifftshift(centred_filtered_frq)
40 filtered_img = np.real(np.fft.ifft2(filtered_frq))
41
42 # display the results
43 plt.figure("Homomorphic Filtering", (8, 8))
44 plt.subplot(121), plt.imshow(original_img, 'gray'), plt.title("original"),
plt.axis('off')
45 plt.subplot(122), plt.imshow(filtered_img, 'gray'), plt.title("filtered"),
plt.axis('off')
46 plt.suptitle(f"Running Results of Example 4.22 Homomorphic Filtering")
47
48 plt.tight_layout()
49 plt.show()
50
```

Running Results of Example 4.22 Homomorphic Filtering



4. Implement Example 4.23 in Page 318.



(followed by *Matlab live Scripts* or *Jupyter Scripts* and running results)

```

1  import cv2
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5
6  def butter_notch_reject(Q, u_list, v_list, D_list, n, shape):
7      M, N = shape
8      row_c, col_c = np.floor(M / 2), np.floor(N / 2)
9      H_NR = np.ones(shape, dtype=np.float64)
10     for k in range(Q):
11         H = np.zeros(shape, dtype=np.float64)
12         H_1 = np.zeros(shape, dtype=np.float64)
13         H_2 = np.zeros(shape, dtype=np.float64)
14         for u in range(M):
15             for v in range(N):
16                 d_1 = np.sqrt((u - row_c - u_list[k]) ** 2 + (v - col_c -
v_list[k]) ** 2)
17                 d_2 = np.sqrt((u - row_c + u_list[k]) ** 2 + (v - col_c +
v_list[k]) ** 2)
18                 H_1[u, v] = 1 / (1 + np.power(D_list[k] / (0.00000000001 + d_1),
2 * n))
19                 H_2[u, v] = 1 / (1 + np.power(D_list[k] / (0.00000000001 + d_2),
2 * n))
20             H = H_1 * H_2
21             H_NR *= H
22     return H_NR
23
24
25 # read the original image
26 original_img = cv2.imread("../images/Fig0464(a)(car_75DPI_Moire).png", flags=0)
27
28 # calculate the 2D DFT of the original image
29 original_frq = np.fft.fft2(original_img)
30 centred_original_frq = np.fft.fftshift(original_frq)
31
32 original_spectrum = np.abs(centred_original_frq)

```

```

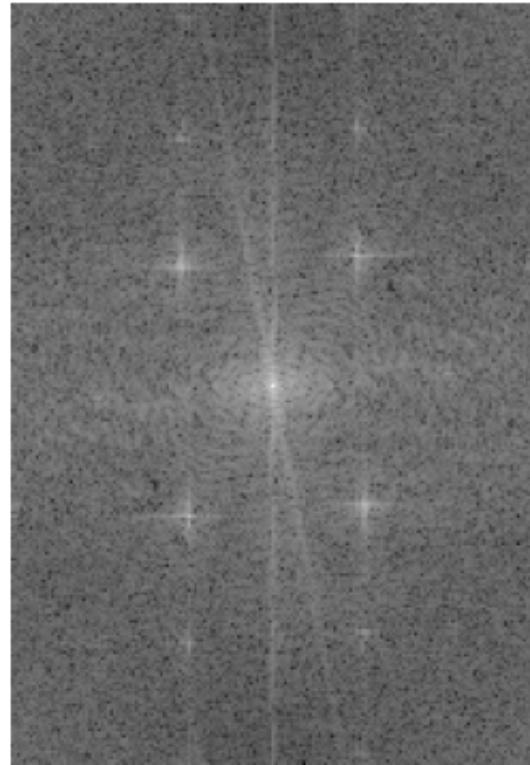
33 log_spectrum = np.log(original_spectrum)
34
35 xs = [55, 55, 57, 58]
36 ys = [44, 86, 166, 207]
37 u = []
38 v = []
39 row_c, col_c = np.floor(original_img.shape[0] / 2),
    np.floor(original_img.shape[1] / 2)
40 for x in xs:
41     v.append(col_c - x)
42 for y in ys:
43     u.append(row_c - y)
44
45 D = [9 for i in range(4)]
46 BNRf = butter_notch_reject(4, u, v, D, n=4, shape=original_img.shape)
47 centred_filtered_frq = centred_original_frq * BNRf
48 filtered_spectrum = log_spectrum * BNRf
49 log_filtered_spectrum = np.log(filtered_spectrum)
50
51 # inverse the 2D DFT
52 filtered_frq = np.fft.ifftshift(centred_filtered_frq)
53 filtered_img = np.real(np.fft.ifft2(filtered_frq))
54
55 # display the results
56 plt.figure("Notch Reject Filtering", (6, 8))
57 plt.subplot(221), plt.imshow(original_img, 'gray'), plt.title("original image"),
    plt.axis('off')
58 plt.subplot(222), plt.imshow(log_spectrum, 'gray'), plt.title("original
    spectrum"), plt.axis('off')
59 plt.subplot(223), plt.imshow(filtered_spectrum, 'gray'), plt.title("filtered
    spectrum"), plt.axis('off')
60 plt.subplot(224), plt.imshow(filtered_img, 'gray'), plt.title("filtered image"),
    plt.axis('off')
61
62 plt.tight_layout()
63 plt.show()
64

```

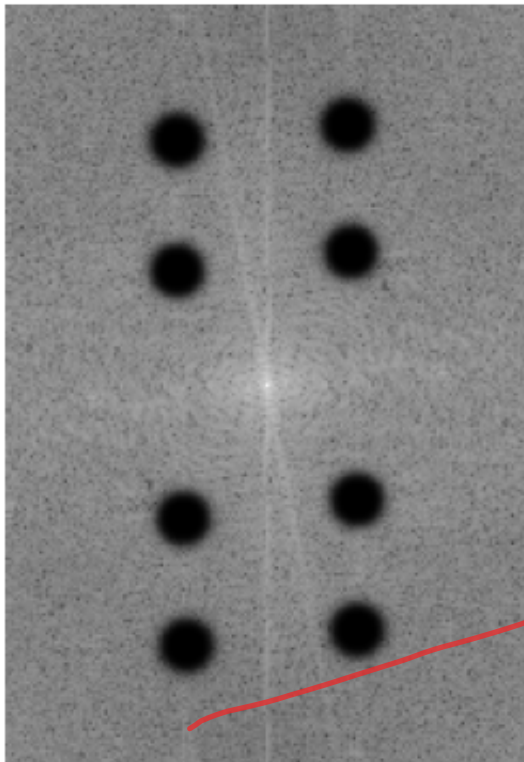

original image



original spectrum



filtered spectrum



filtered image

