EXPERIMENT 2

Digital Image Processing

M.W. Nethmi N. Muthugala - 那娜

School of AI

W2010816010

19th November, 2022

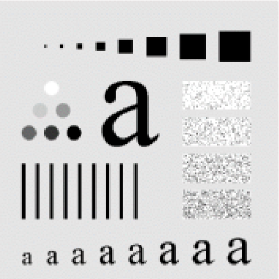
CONTENT

I. Getting to know the FFT method and its properties

1. FFT2 transform with (a) Sinusoidal wave (b) rectangle (c) square (d) triangle (d) sphere (e) objects with different directions and combinations.

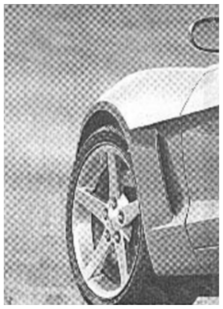
II. Filtering with the FFT method.

2. Low pass filters the img1.tif with Gaussian, Butterworth, Hard filter and try different parameters of the filters.
3. High pass filters the img1.tif with Gaussian, Butterworth, Hard filter and try different parameters of the filters.
4. Discuss the differences before and after the filtering.

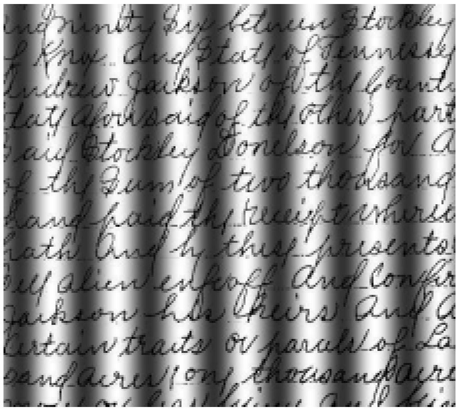


III. Compound experiment

5. Remove the dithering noise in the newspaper.png image with notch filtering the frequency spectrum



6. Try to remove the sinusoidal pattern in the img2.tif

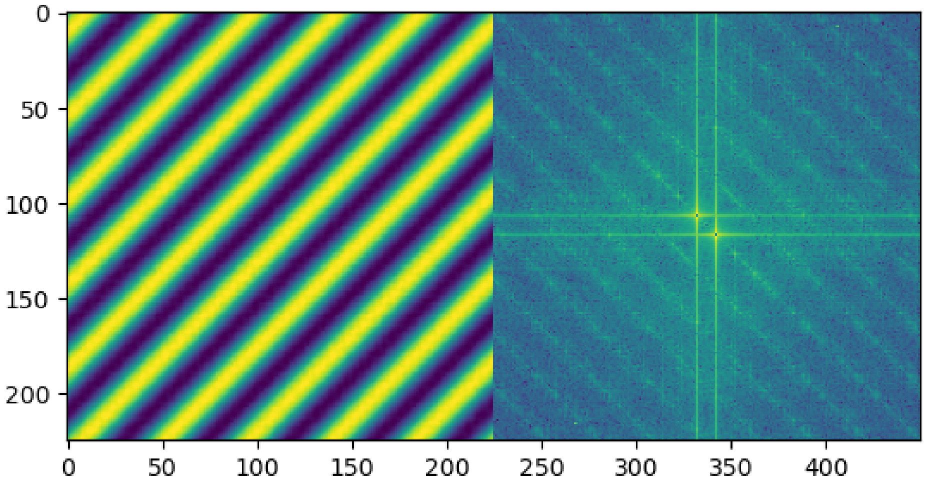

ing ninety Six between Stokley
of Knox And State of Tennessee
Andrew Jackson of the County
State aforesaid of the other part
said Stockley Donelson for A
of the Sum of two thousand
hand paid the receipt whereof
hath And by these presents
sell alien enfeoff And Confir
Jackson his heirs And A
certain traits or parcels of La
sand acres one thousand acre

I. Getting to know the FFT method and its properties

(1.1) FFT2 transformation with Sinusoidal wave

- **With Sinusoidal waves (diagonal)**

```
import cv2 as cv

import numpy as np

im = cv.imread('wave1.jfif', 0)


f = np.fft.fft2(im)

fshift = np.fft.fftshift(f)

magnitude_spectrum = 20*np.log(np.abs(fshift))

print (magnitude_spectrum)

magnitude_spectrum = np.asarray(magnitude_spectrum, dtype=np.uint8)
```



- **Comparing 2D sinusoidal waves which are different in direction**

```
import cv2 as cv
import numpy as np


im = cv.imread('wave1.jfif', 0)


f = np.fft.fft2(im)

fshift = np.fft.fftshift(f)

magnitude_spectrum = 20*np.log(np.abs(fshift))

print (magnitude_spectrum)

magnitude_spectrum = np.asarray(magnitude_spectrum, dtype=np.uint8)
```

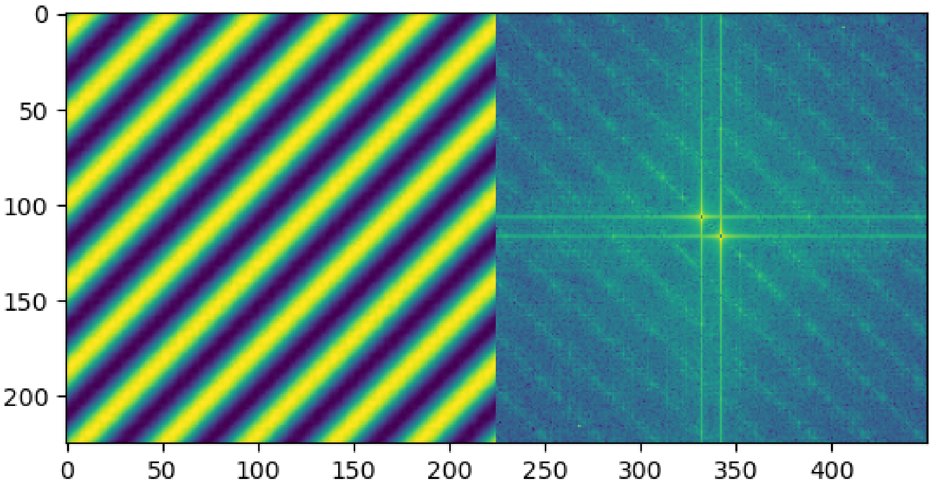


Image 1-Diagonal

```
im2 = cv.imread('wave2.jfif', 0)

fa = np.fft.fft2(im2)

fashift = np.fft.fftshift(f)

magnitude_2 = 20*np.log(np.abs(fashift))

print (magnitude_2)

magnitude_2 = np.asarray(magnitude_2, dtype=np.uint8)
```

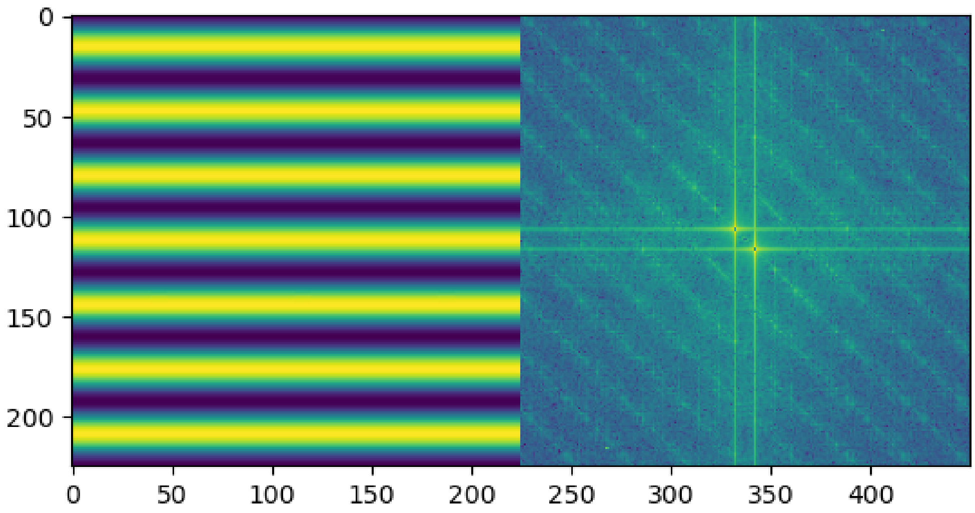


Image 2-Horizontal

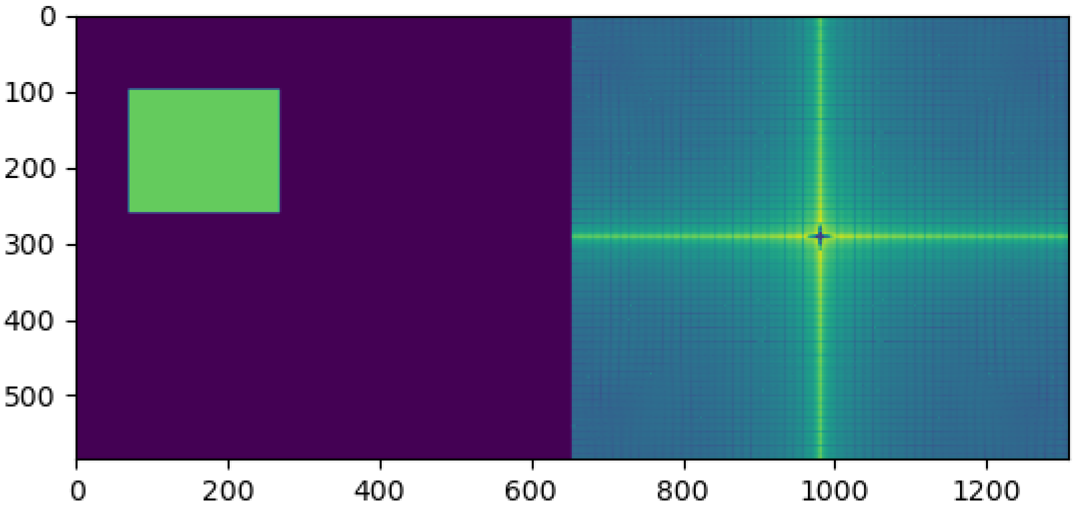
Comparing the image outputs of image 1 and 2 after adding the fourier transform - Same output

(1.2) FFT2 transformation with Rectangle

- **1 rectangle**

```
import cv2 as cv

import numpy as np


im = cv.imread('square.jpg', 0)


f = np.fft.fft2(im)

fshift = np.fft.fftshift(f)

magnitude_spectrum = 20*np.log(np.abs(fshift))

print (magnitude_spectrum)
```

```
magnitude_spectrum = np.asarray(magnitude_spectrum, dtype=np.uint8)


res = np.hstack((im, magnitude_spectrum)) #stacking images side-by-side

cv.imwrite('res.png',res)
```



- **2 rectangles but the position has changed**

```
import cv2 as cv

import numpy as np


im = cv.imread('square.jpg', 0)


f = np.fft.fft2(im)

fshift = np.fft.fftshift(f)

magnitude_spectrum = 20*np.log(np.abs(fshift))

print (magnitude_spectrum)
```

```
magnitude_spectrum = np.asarray(magnitude_spectrum, dtype=np.uint8)


res = np.hstack((im, magnitude_spectrum)) #stacking images side-by-side

cv.imwrite('res.png',res)
```

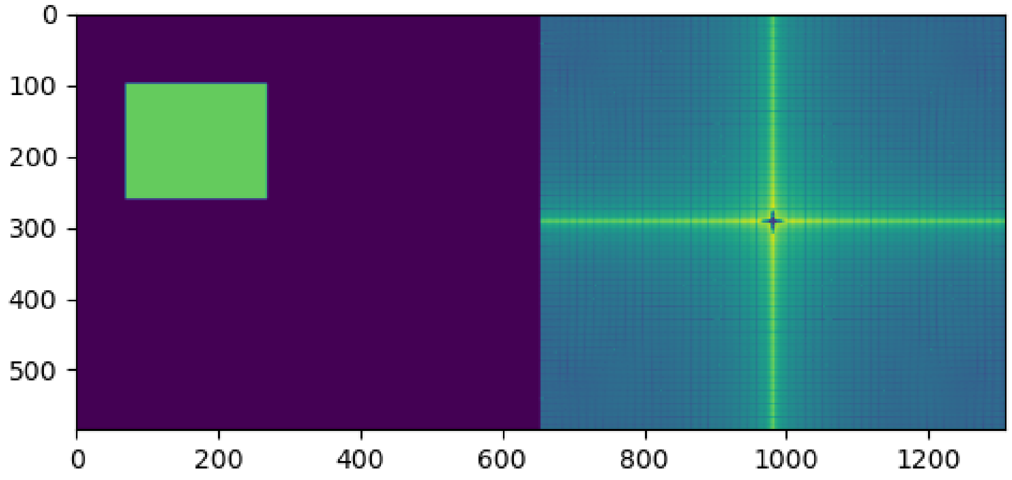


Image1-left

```
im2 = cv.imread('square2.jpg', 0)

fa = np.fft.fft2(im2)

fashift = np.fft.fftshift(fa)

magnitude_2 = 20*np.log(np.abs(fashift))

print (magnitude_2)

magnitude_2 = np.asarray(magnitude_2, dtype=np.uint8)


res2 = np.hstack((im2, magnitude_2)) #stacking images side-by-side

cv.imwrite('res2.png',res2)
```

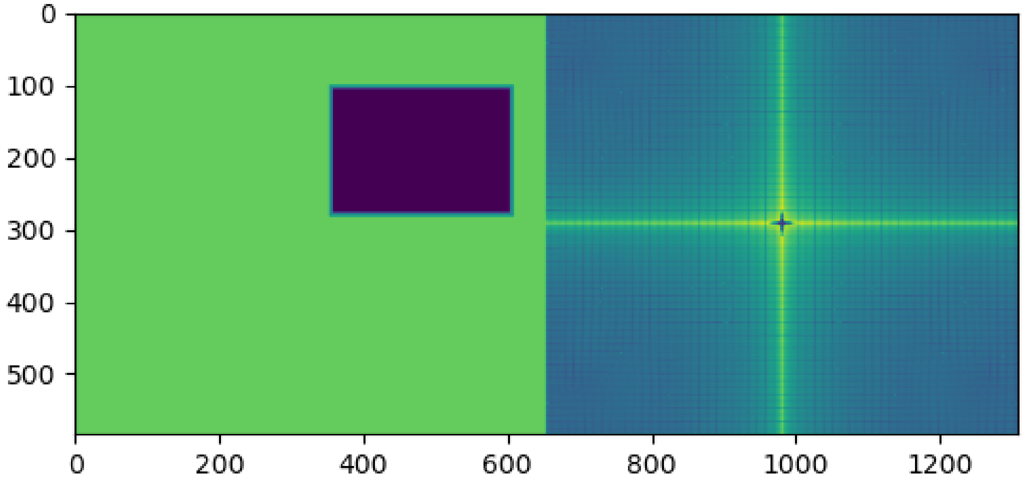


Image2-right

Comparing the image outputs of image 1 and 2 after adding the fourier transform - Same output

- **2 rectangles but the angle/direction has changed**

```
import cv2 as cv

import numpy as np


im = cv.imread('square.jpg', 0)


f = np.fft.fft2(im)

fshift = np.fft.fftshift(f)

magnitude_spectrum = 20*np.log(np.abs(fshift))

print (magnitude_spectrum)

magnitude_spectrum = np.asarray(magnitude_spectrum, dtype=np.uint8)
```

```
res = np.hstack((im, magnitude_spectrum)) #stacking images side-by-side

cv.imwrite('res.png',res)
```

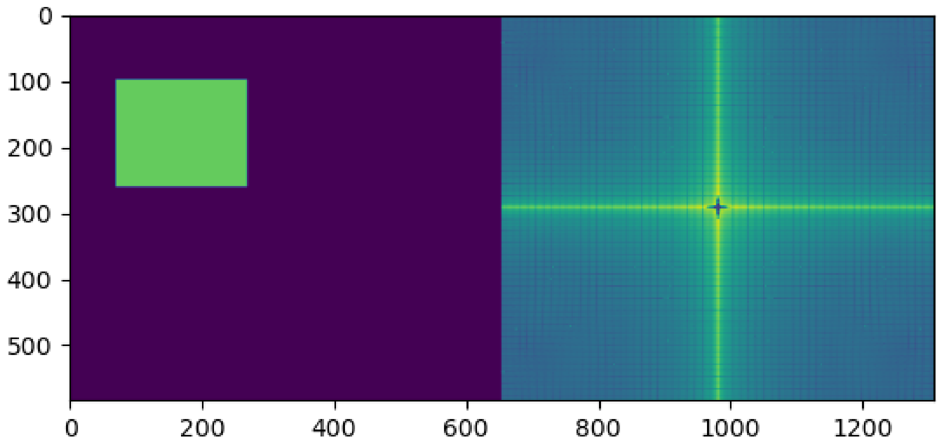


Image1-rectangle

```
im3 = cv.imread('square3.jpg', 0)

fa3 = np.fft.fft2(im2)

fa3shift = np.fft.fftshift(fa3)

magnitude_3 = 20*np.log(np.abs(fa3shift))

print (magnitude_3)

magnitude_3 = np.asarray(magnitude_3, dtype=np.uint8)


res3 = np.hstack((im3, magnitude_3)) #stacking images side-by-side

cv.imwrite('res3.png',res3)
```

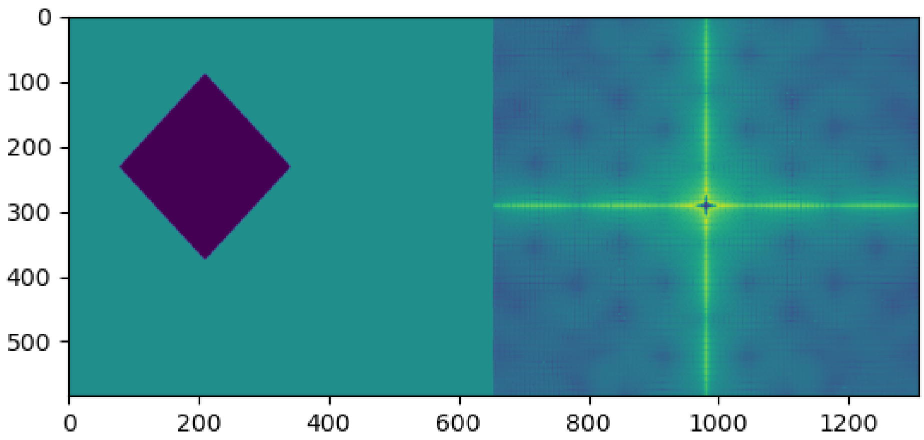


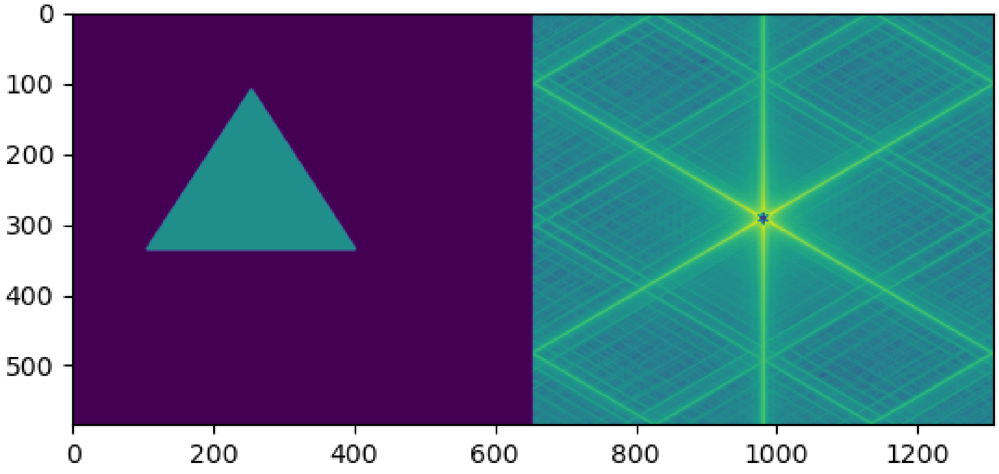
Image2- rhombus

Comparing the image outputs of image 1 and 2 after adding the fourier transform - slightly different output

(1.3) FFT2 transformation with Triangle

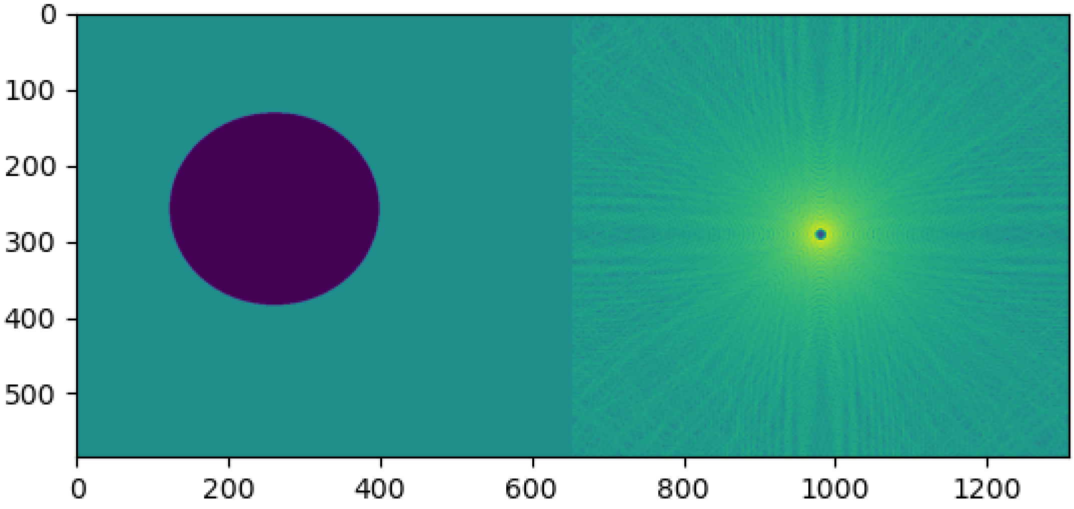
```
import cv2 as cv

import numpy as np


im = cv.imread('triangle.jpg', 0)


f = np.fft.fft2(im)

fshift = np.fft.fftshift(f)

magnitude_spectrum = 20*np.log(np.abs(fshift))

print (magnitude_spectrum)

magnitude_spectrum = np.asarray(magnitude_spectrum, dtype=np.uint8)
```

```
res = np.hstack((im, magnitude_spectrum)) #stacking images side-by-side

cv.imwrite('res.png',res)
```



(1.4) FFT2 transformation with Circle

(Same code)

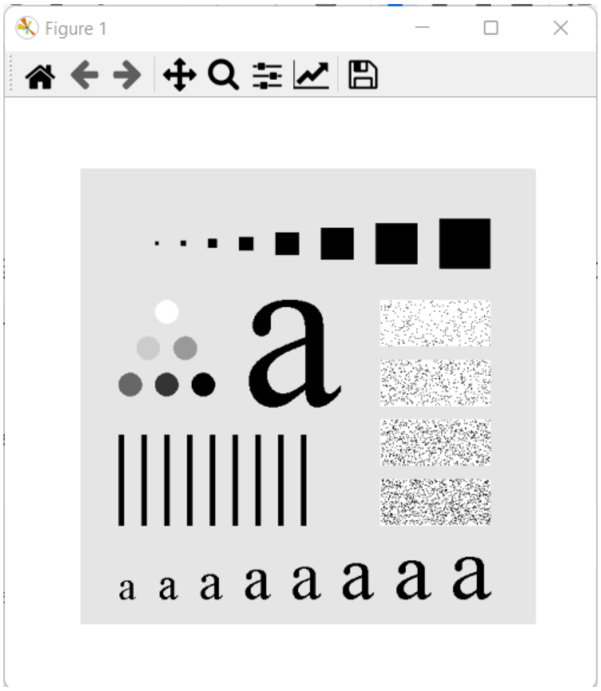


II. Filtering with the FFT method

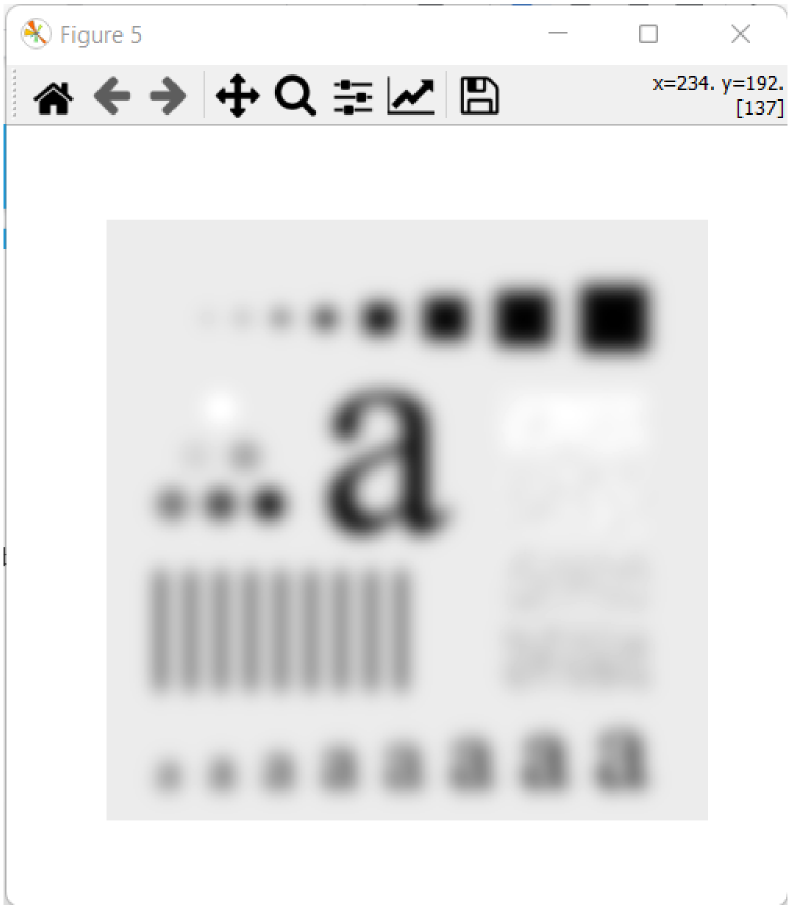
(2.1) Low pass filters the img1.tif with Gaussian, Butterworth, Hard filter and try different parameters of the filters.

- **Low pass Gaussian filter**

```
import cv2 as cv

import numpy as np

import matplotlib.pyplot as plt


# open the image f

f = cv.imread('img1.tif',0)
```



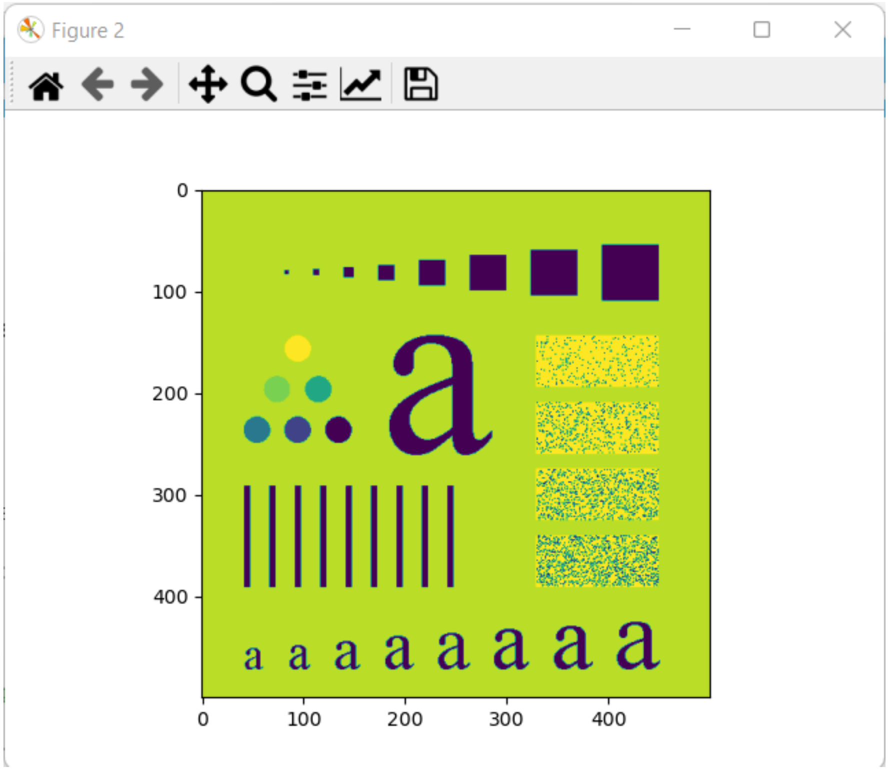
```
plt.figure(figsize=(5,5))
plt.imshow(f, cmap='gray')
plt.axis('off')
plt.show()

# transform the image into frequency domain, f --> F
F = np.fft.fft2(f)
Fshift = np.fft.fftshift(F)

plt.figure(figsize=(5,5))
plt.imshow(np.log1p(np.abs(F)), cmap='gray')
plt.axis('off')
plt.show()

plt.figure(figsize=(5,5))
plt.imshow(np.log1p(np.abs(Fshift)), cmap='gray')
plt.axis('off')
plt.show()

# Create Gaussin Filter: Low Pass Filter
M,N = f.shape
H = np.zeros((M,N), dtype=np.float32)
D0 = 10
for u in range(M):
    for v in range(N):
        D = np.sqrt((u-M/2)**2 + (v-N/2)**2)
        H[u,v] = np.exp(-D**2/(2*D0*D0))

plt.figure(figsize=(5,5))
plt.imshow(H, cmap='gray')
plt.axis('off')
plt.show()

# Image Filters
Gshift = Fshift * H
G = np.fft.ifftshift(Gshift)
g = np.abs(np.fft.ifft2(G))

plt.figure(figsize=(5,5))
plt.imshow(g, cmap='gray')
plt.axis('off')
plt.show()

plt.figure(figsize=(5,5))
plt.imshow(np.log1p(np.abs(Gshift)), cmap='gray')
plt.axis('off')
plt.show()
plt.figure(figsize=(5,5))

plt.imshow(np.log1p(np.abs(G)), cmap='gray')

plt.axis('off')

plt.show()
```



Final Output

- **Low pass Butterworth filter**

```
import cv2

import numpy as np

import matplotlib.pyplot as plt


# open the image

f = cv2.imread('img1.tif',0)
```

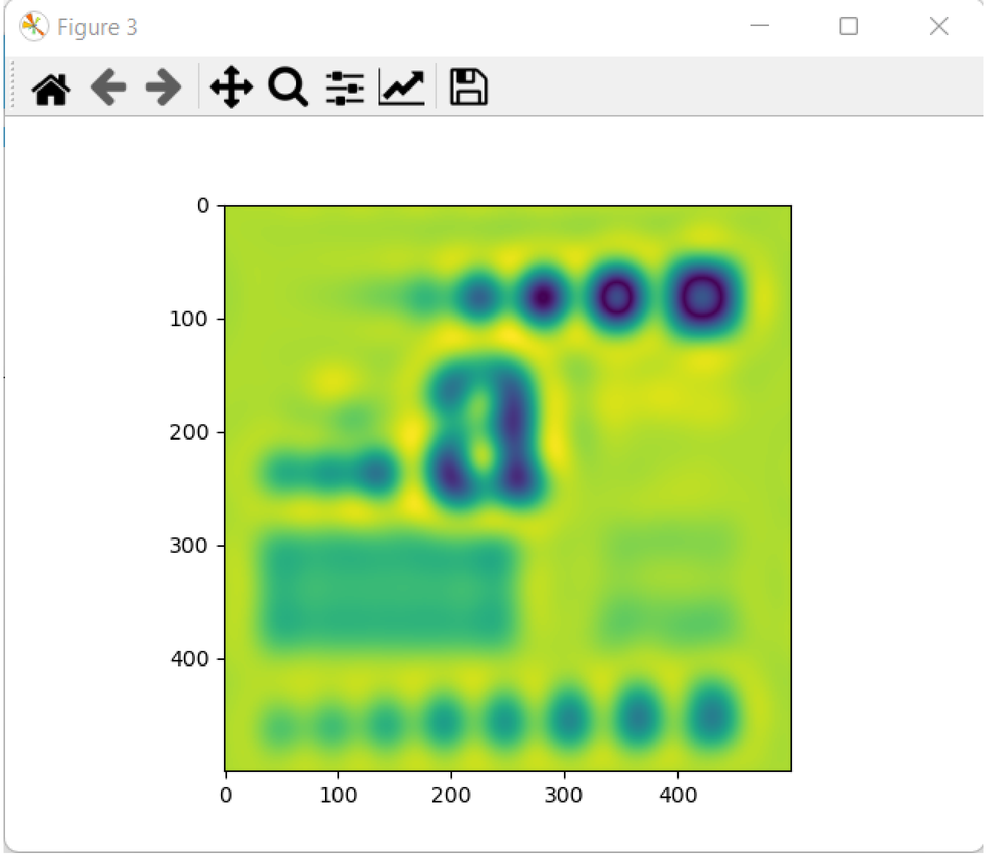


Original IMage

```
# transform image into freq. domain and shifted

F = np.fft.fft2(f)

Fshift = np.fft.fftshift(F)
```

```
plt.imshow(np.log1p(np.abs(Fshift)), cmap='gray')

plt.axis('off')

plt.show()


# Butterworth Low Pass Filter

M,N = f.shape

H = np.zeros((M,N), dtype=np.float32)

D0 = 10 # cut of frequency

n = 10 # order

for u in range(M):

    for v in range(N):

        D = np.sqrt((u-M/2)**2 + (v-N/2)**2)

        H[u,v] = 1 / (1 + (D/D0)**n)


plt.imshow(H, cmap='gray')

plt.axis('off')

plt.show()


# frequency domain image filters

Gshift = Fshift * H

G = np.fft.ifftshift(Gshift)

g = np.abs(np.fft.ifft2(G))
```

```
plt.imshow(g, cmap='gray')

plt.axis('off')

plt.show()
```

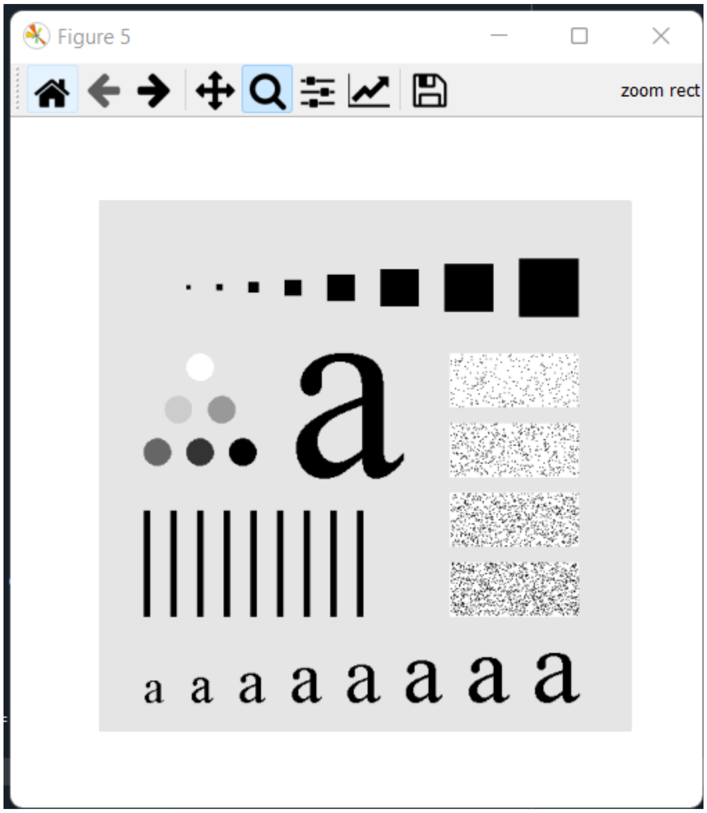


Final Output

(2.2) High pass filters the img1.tif with Gaussian, Butterworth, Hard filter and try different parameters of the filters.

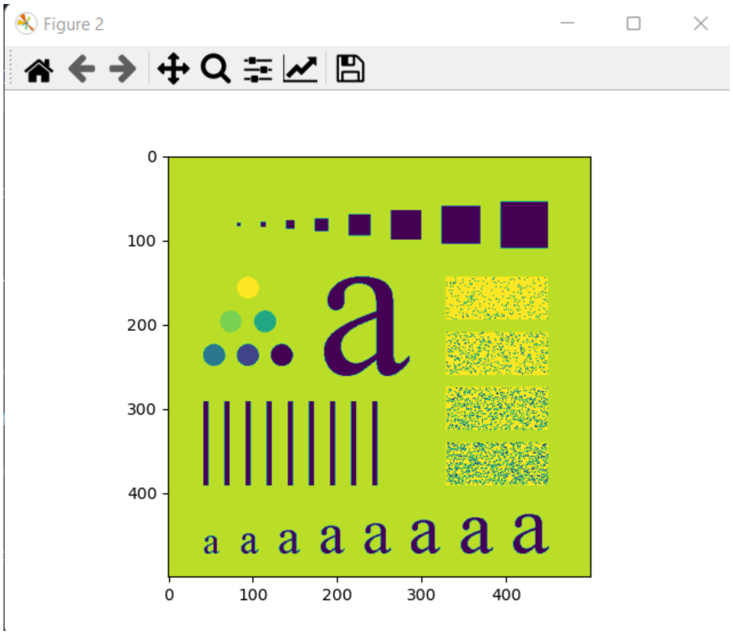
- **High pass Gaussian filter**

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

# open the image f
f = cv.imread('img1.tif',0)

plt.figure(figsize=(5,5))
plt.imshow(f, cmap='gray')
plt.axis('off')
plt.show()

# transform the image into frequency domain, f --> F
F = np.fft.fft2(f)
Fshift = np.fft.fftshift(F)

plt.figure(figsize=(5,5))
plt.imshow(np.log1p(np.abs(F)), cmap='gray')
plt.axis('off')
plt.show()
plt.figure(figsize=(5,5))

plt.imshow(np.log1p(np.abs(Fshift)), cmap='gray')

plt.axis('off')

plt.show()


# Gaussian: High pass filter

M,N = f.shape

H = np.zeros((M,N), dtype=np.float32)

D0 = 10


HPF = 1 - H


plt.figure(figsize=(5,5))

plt.imshow(HPF, cmap='gray')

plt.axis('off')

plt.show()


# Image Filters

Gshift = Fshift * HPF

G = np.fft.ifftshift(Gshift)

g = np.abs(np.fft.ifft2(G))


plt.figure(figsize=(5,5))

plt.imshow(g, cmap='gray')
plt.axis('off')

plt.show()


plt.figure(figsize=(5,5))

plt.imshow(np.log1p(np.abs(Gshift)), cmap='gray')

plt.axis('off')

plt.show()


plt.figure(figsize=(5,5))

plt.imshow(np.log1p(np.abs(G)), cmap='gray')

plt.axis('off')

plt.show()
```



- **High pass Butterworth filter**

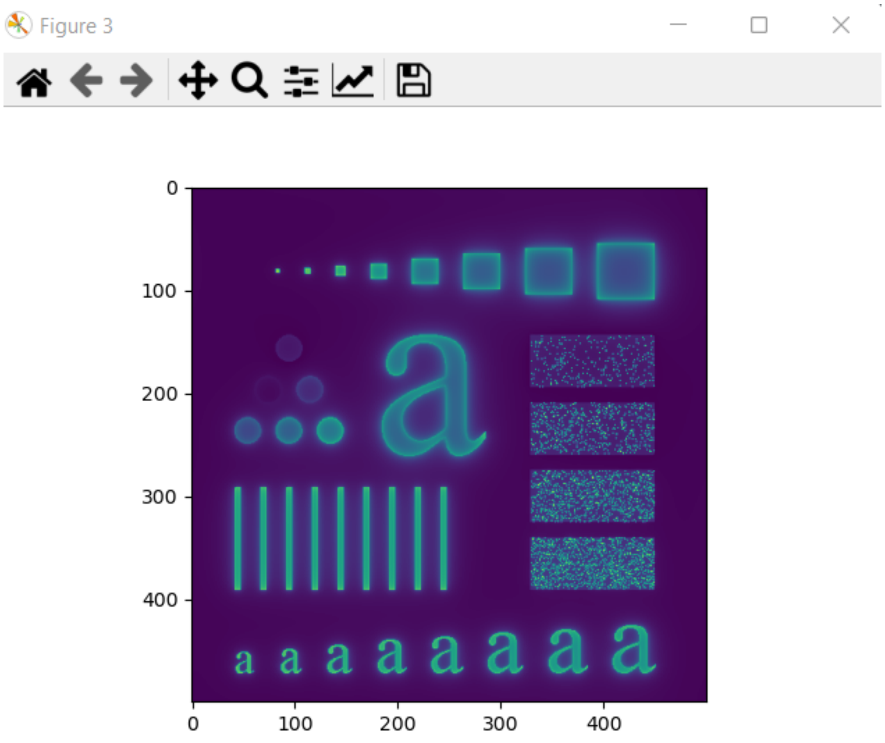
```
import cv2

import numpy as np

import matplotlib.pyplot as plt


# open the image

f = cv2.imread('img1.tif',0)
```



```
# transform image into freq. domain and shifted

F = np.fft.fft2(f)

Fshift = np.fft.fftshift(F)


plt.imshow(np.log1p(np.abs(Fshift)), cmap='gray')

plt.axis('off')

plt.show()
```

```
# Butterworth High Pass Filter
M,N = f.shape
H = np.zeros((M,N), dtype=np.float32)
D0 = 10 # cut of frequency
n = 10 # order

HPF = np.zeros((M,N), dtype=np.float32)
D0 = 10
n = 1
for u in range(M):
    for v in range(N):
        D = np.sqrt((u-M/2)**2 + (v-N/2)**2)
        HPF[u,v] = 1 / (1 + (D0/D)**n)

plt.imshow(HPF, cmap='gray')
plt.axis('off')
plt.show()

# frequency domain image filters
Gshift = Fshift * HPF
G = np.fft.ifftshift(Gshift)
g = np.abs(np.fft.ifft2(G))
```

```
plt.imshow(g, cmap='gray')

plt.axis('off')

plt.show()
```



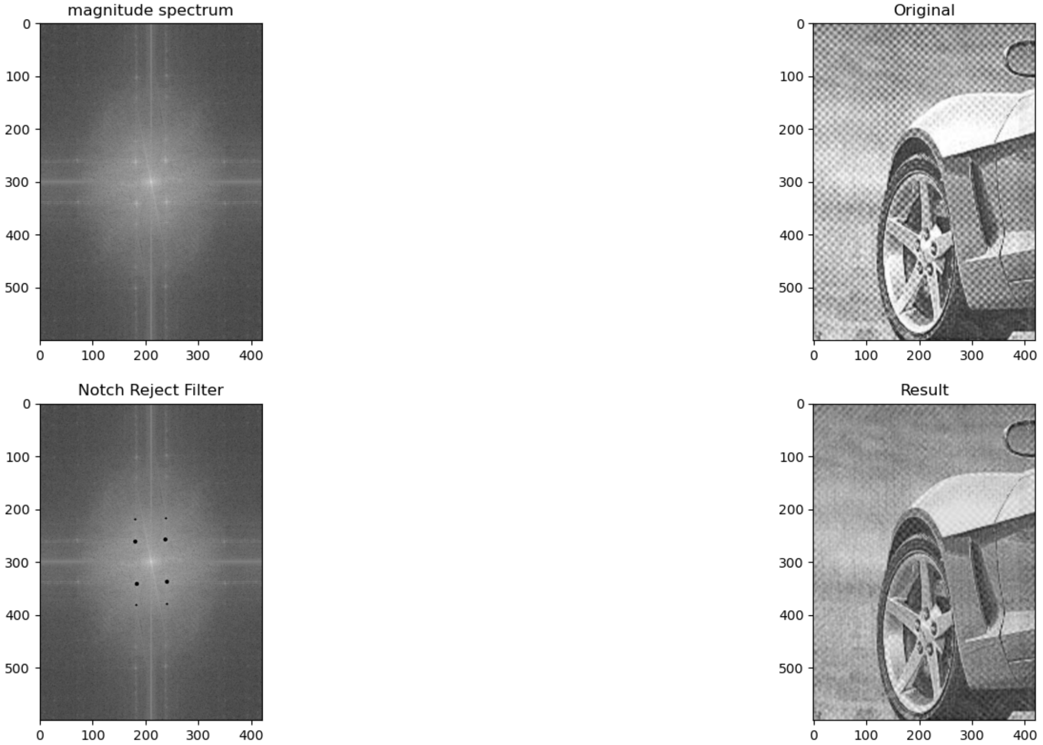
Final Output

III. Compound experiment

(3.1) Remove the dithering noise in the img4.png image with notch filtering the frequency spectrum

```
import cv2

import numpy as np

import matplotlib.pyplot as plt




def notch_reject_filter(shape, d0=9, u_k=0, v_k=0):

    P, Q = shape

    # Initialize filter with zeros

    H = np.zeros((P, Q))


    # Traverse through filter

    for u in range(0, P):

        for v in range(0, Q):

            # Get euclidean distance from point D(u,v) to the center

            D_uv = np.sqrt((u - P / 2 + u_k) ** 2 + (v - Q / 2 + v_k) ** 2)

            D_muv = np.sqrt((u - P / 2 - u_k) ** 2 + (v - Q / 2 - v_k) ** 2)


            if D_uv <= d0 or D_muv <= d0:

                H[u, v] = 0.0

            else:
                H[u, v] = 1.0

    return H

img = cv2.imread('img4.png', 0)

f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)
phase_spectrumR = np.angle(fshift)
magnitude_spectrum = 20*np.log(np.abs(fshift))

img_shape = img.shape

H1 = notch_reject_filter(img_shape, 4, 38, 30)
H2 = notch_reject_filter(img_shape, 4, -42, 27)
H3 = notch_reject_filter(img_shape, 2, 80, 30)
H4 = notch_reject_filter(img_shape, 2, -82, 28)

NotchFilter = H1*H2*H3*H4
NotchRejectCenter = fshift * NotchFilter
NotchReject = np.fft.ifftshift(NotchRejectCenter)
inverse_NotchReject = np.fft.ifft2(NotchReject)  # Compute the inverse DFT of the result
```

```
Result = np.abs(inverse_NotchReject)

plt.subplot(222)
plt.imshow(img, cmap='gray')
plt.title('Original')

plt.subplot(221)
plt.imshow(magnitude_spectrum, cmap='gray')
plt.title('magnitude spectrum')

plt.subplot(223)
plt.imshow(magnitude_spectrum*NotchFilter, "gray")
plt.title("Notch Reject Filter")

plt.subplot(224)
plt.imshow(Result, "gray")
plt.title("Result")
plt.show()
```


magnitude spectrum
Original
Notch Reject Filter
Result
0
100
200
300
400
500
0
100
200
300
400

Image

(3.2) Remove the sinusoidal pattern in the img2.tif

(Apologies, I actually couldn't find a proper way to do this task)