Computer Assignment 2

CPE 261453 (Digital Image Processing)

โดย

นายเธียร สุวรรณกุล

รหัสนักศึกษา 620610176

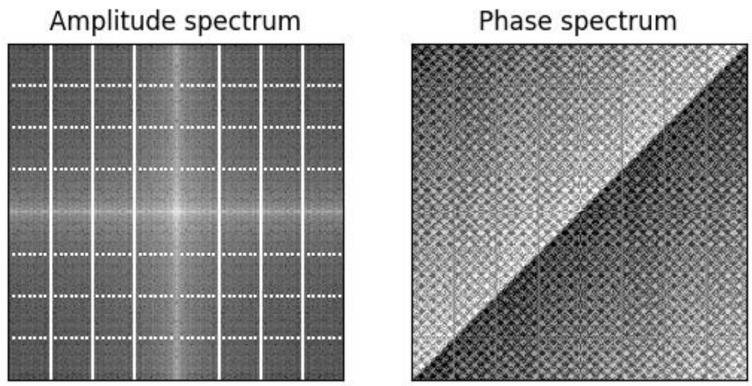
เสนอ

ผศ.ดร.ศันสนีย์ เอื้อพันธ์วิริยะกุล

คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเชียงใหม่

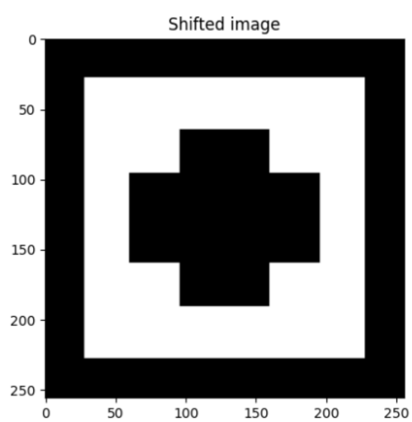
Properties of the Fourier Transform

1. Convert the Fourier Transform of the image "Cross.pgm" (200x200) as FFT requires the image to be in the form of 2n. Therefore, padding may be necessary. Display the resulting image in the form of amplitude and phase spectra.



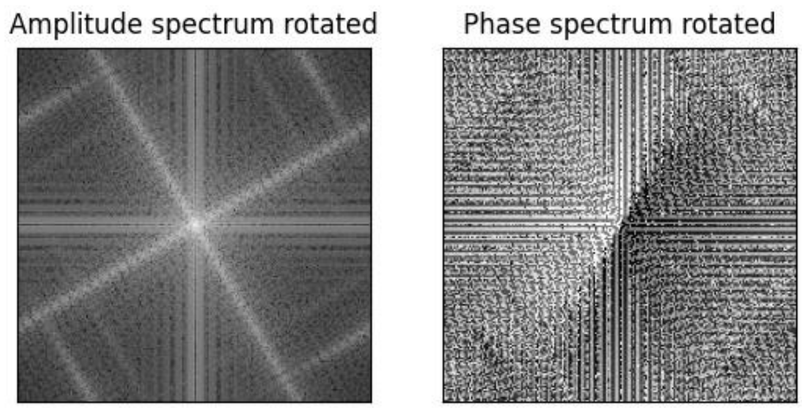
First , I’ve padded original image from 200x200 to 256x256(2^n) form, then compute using FFT and shifting for finding both spectrums.

2. Multiply the phase spectrum obtained in step 1 by a complex number to shift the resulting image by (20, 30) in the x and y axes after inverse Fourier transform.



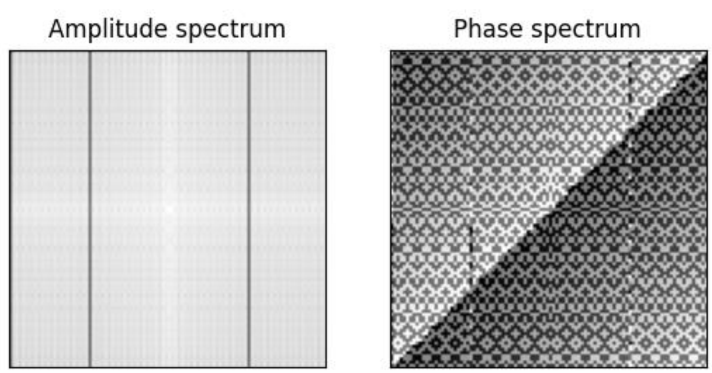
The phase spectrum is multiplied by a complex number to shift the image by (20, 30) in the x and y axes. This is done by creating a new complex array of the same size as the shifted Fourier transform, setting the (30, 20) element to 1, and taking the inverse Fourier transform of the resulting array then multiplied with the shifted Fourier transform.Then takes real part of the image and show it.

3. Rotate the image "Cross.pgm" by 30 degrees and display the result of Fourier transform. Analyze what happens.



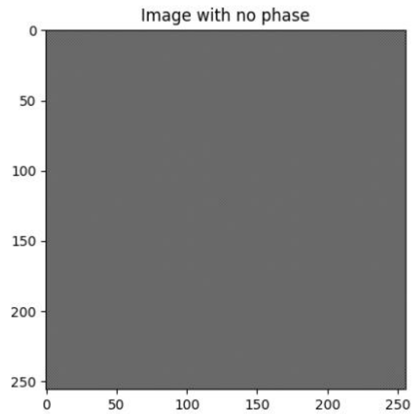
The output of this code will display the amplitude and phase spectra of the rotated image. When you rotate an image, the Fourier transform changes because the frequency components of the image are shifted. In the amplitude spectrum, you will see that the energy of the high-frequency components is spread out in different directions. In the phase spectrum, you will see that the phase values are shifted, indicating the change in the position of the frequency components.

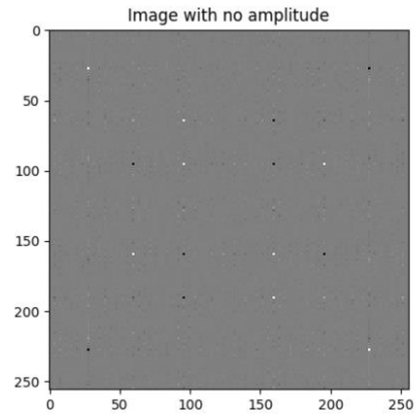
4. Down-sample "Cross.pgm" to a size of 100x100, then perform Fourier Transform to display the resulting image in the form of amplitude and phase spectra. Analyze what happens.



The resulting image will be much smaller and have a lower resolution. Downsampling can cause loss of information, and as a result, the amplitude and phase spectra will be less detailed compared to the original image.

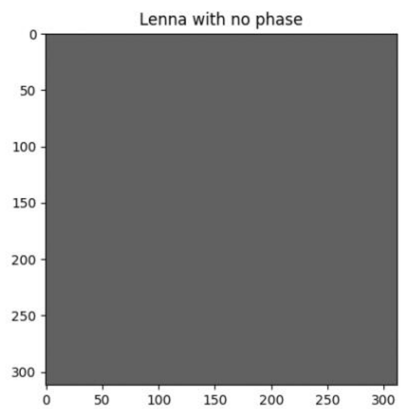
5. Use the inverse Fourier Transform of the result obtained in step 1 with:

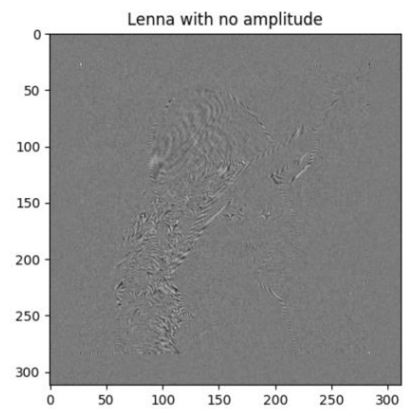




Both image, if you're zoom in closely you will see lines in no phase one , and dots in no amplitude one.

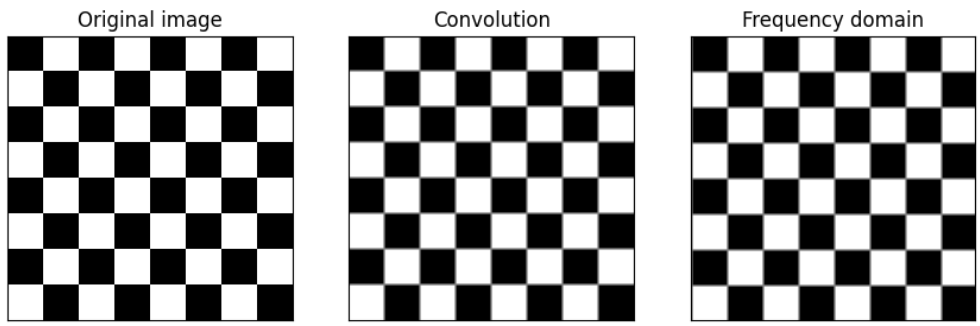
6. Perform the same experiment as in step 5 using the image "Lenna.pgm" (256x256).





In the same case of step 5 , no phase one represent lines of lady Lenna , no amplitude show all the angles of lady Lenna.

7. (a) Perform convolution on the image "Chess.pgm" (256x256) with a small-sized mask or kernel to blur the image. (b) Filter in frequency domain using Fourier transform of the kernel used in step (a) to blur the image. Compare the results obtained from both methods and analyze the similarities or differences.

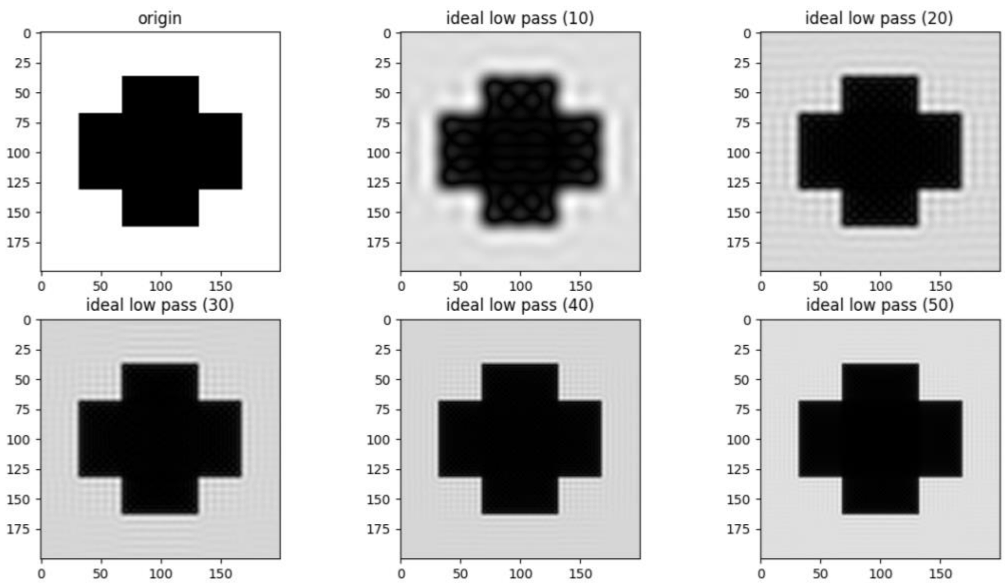


The output of the code shows that both methods of blurring the image produce similar results. The convolution method results in a slightly more blurred image compared to the frequency domain filtering method. This is expected because convolution involves convolving the image with the kernel, which has the effect of blurring the image. However, the frequency domain filtering method is more efficient and faster compared to the convolution method, especially for large kernel sizes.

Filter Design

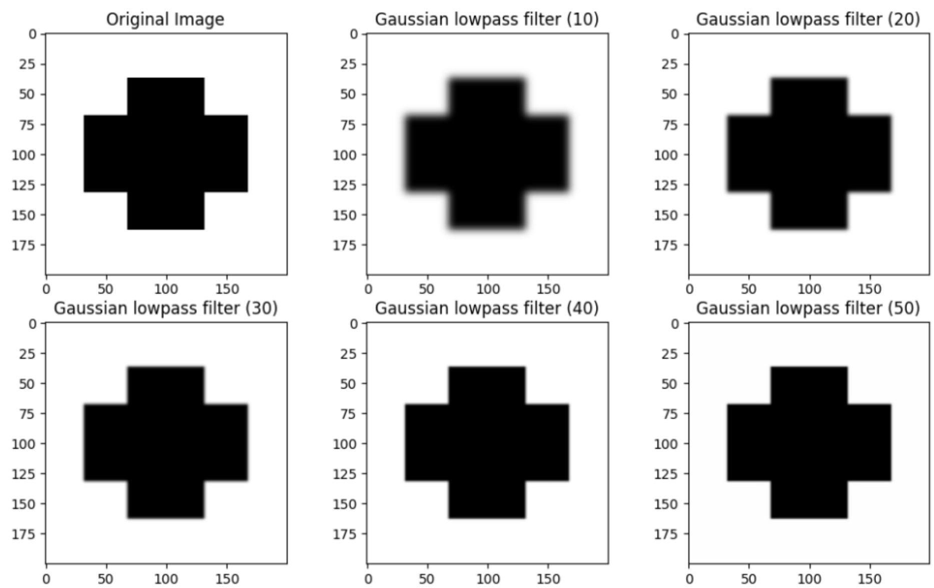
1. Apply an ideal low-pass filter to the image "Cross.pgm" by changing the cutoff frequency and study the ringing effect that occurs. Then, repeat the experiment with other non-ideal filters.

Ideal low-pass filter



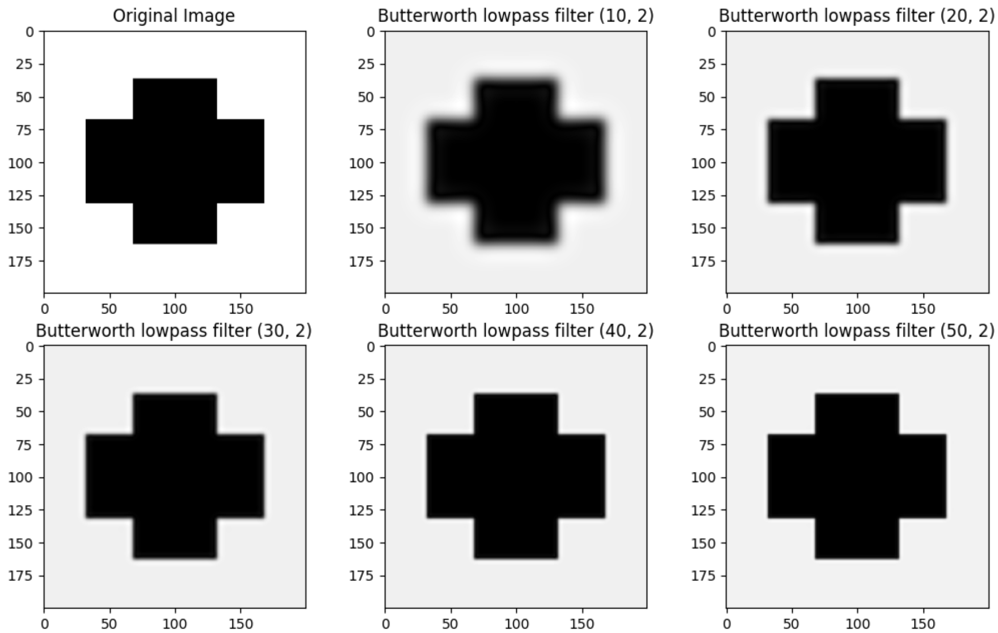
As you can see, it has a ringing effect. According to my research, this effect occurs when there is an abrupt transition from a low-frequency region to a high-frequency region. The output shows that the images become sharper when the cutoff is increased.

Gaussian low-pass filter



In the same way as ideal, the result obtained using a higher cutoff shows that the output image becomes more blurred, and the higher the cutoff, the clearer the output. Moreover, there is no ringing effect in this filtering process

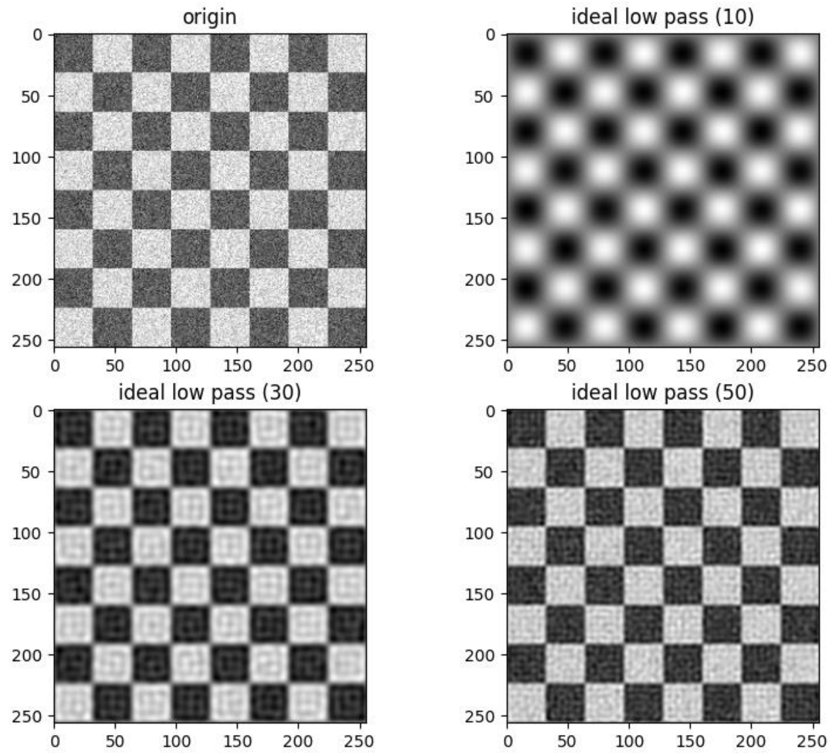
Butterworth low-pass filter



The result is similar to Gaussian low-pass filter which has no ringing effects, but the background is getting more grey.

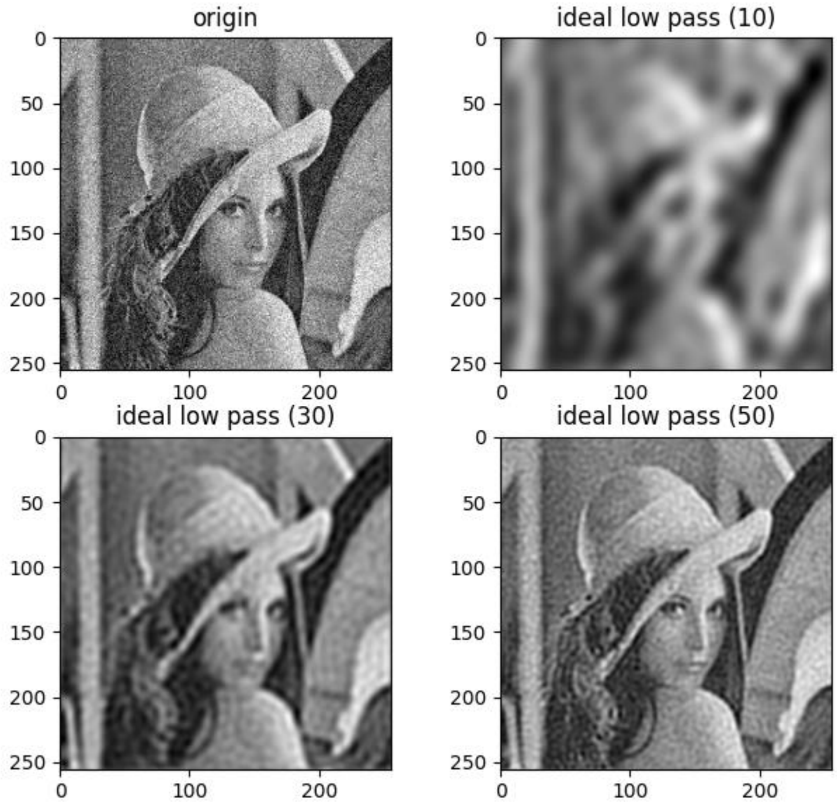
2. Apply multiple types of low-pass filters, including different variable values and filter sizes, to the images "Chess_noise.pgm" and "Lenna_noise.pgm" to reduce noise and compare the results with a median filter that changes in size. Use RMS to calculate the difference between the obtained results and the original noise-free images "Chess.pgm" and "Lenna.pgm".

Ideal low-pass filters



"Chess_ideal"

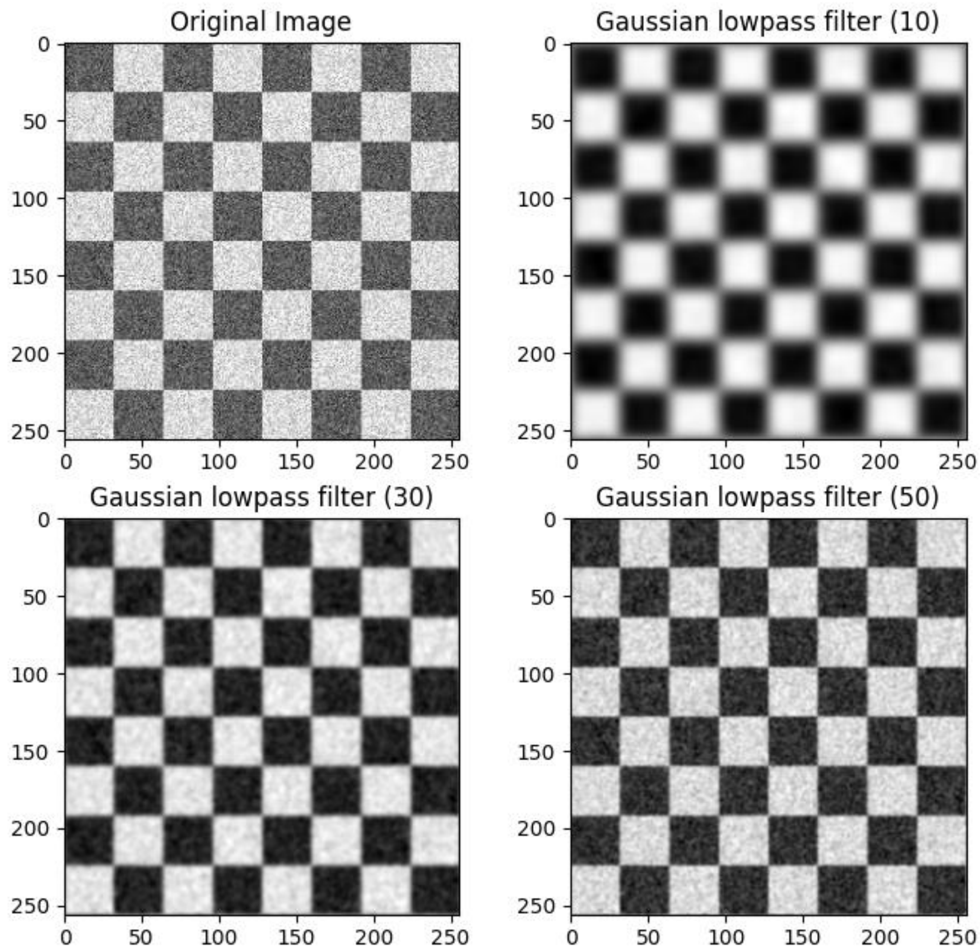
RMS :
Cutoff 10 = 35.08439517831798
Cutoff 30 = 18.7654860854012
Cutoff 50 = 14.585513105155083



"Lenna_ideal"

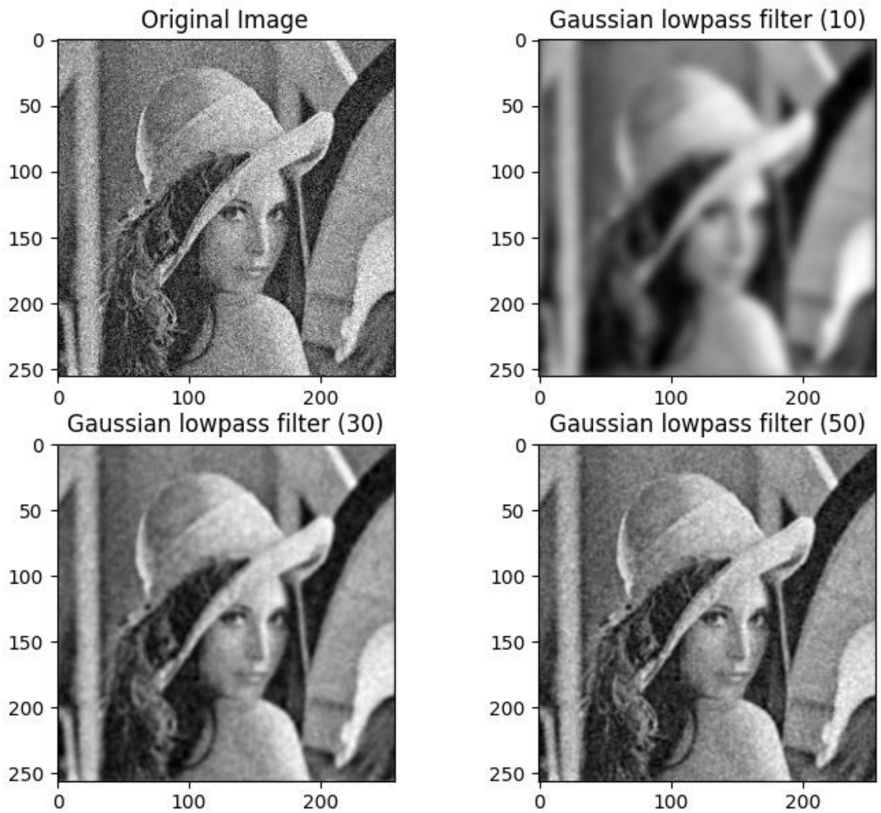
RMS :
Cutoff 10 = 24.176516786853693
Cutoff 30 = 13.170301722027345
Cutoff 50 = 9.129816740817663

Gaussian low-pass filters



"Chest_gaussian"

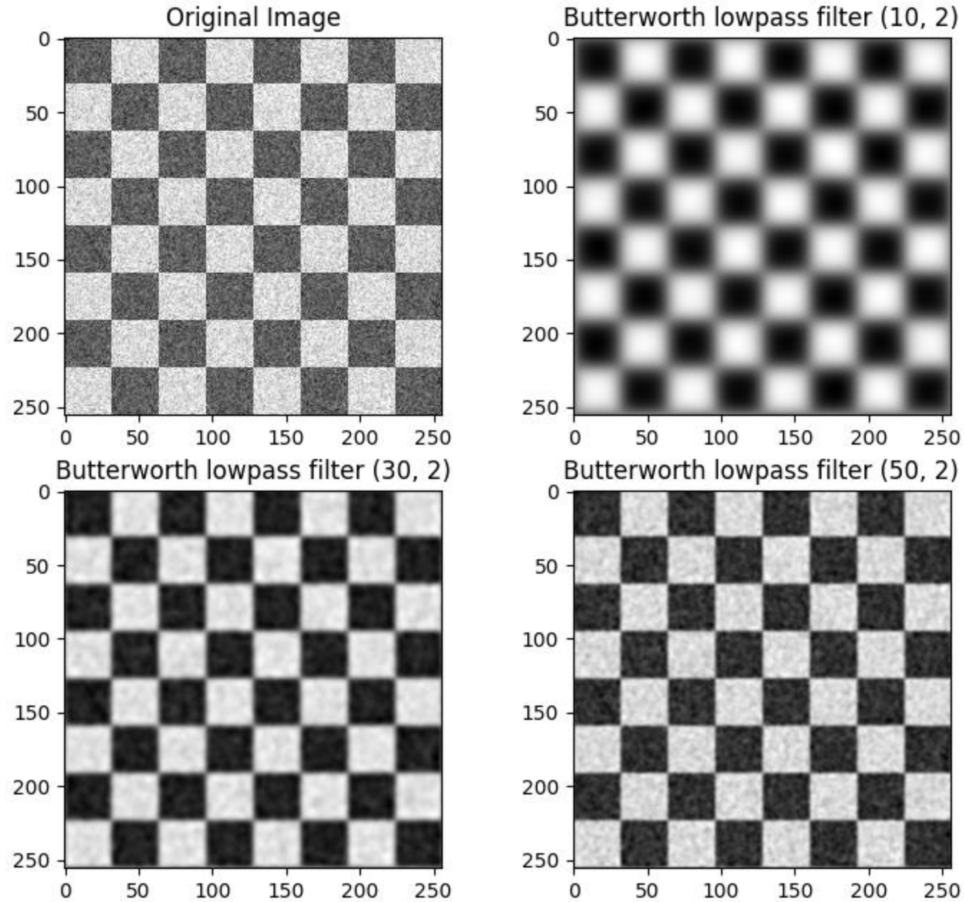
RMS :
Cutoff 10 = 29.135975742469135
Cutoff 30 = 16.05019137183371
Cutoff 50 = 11.04439319371518



"Lenna_gaussian"

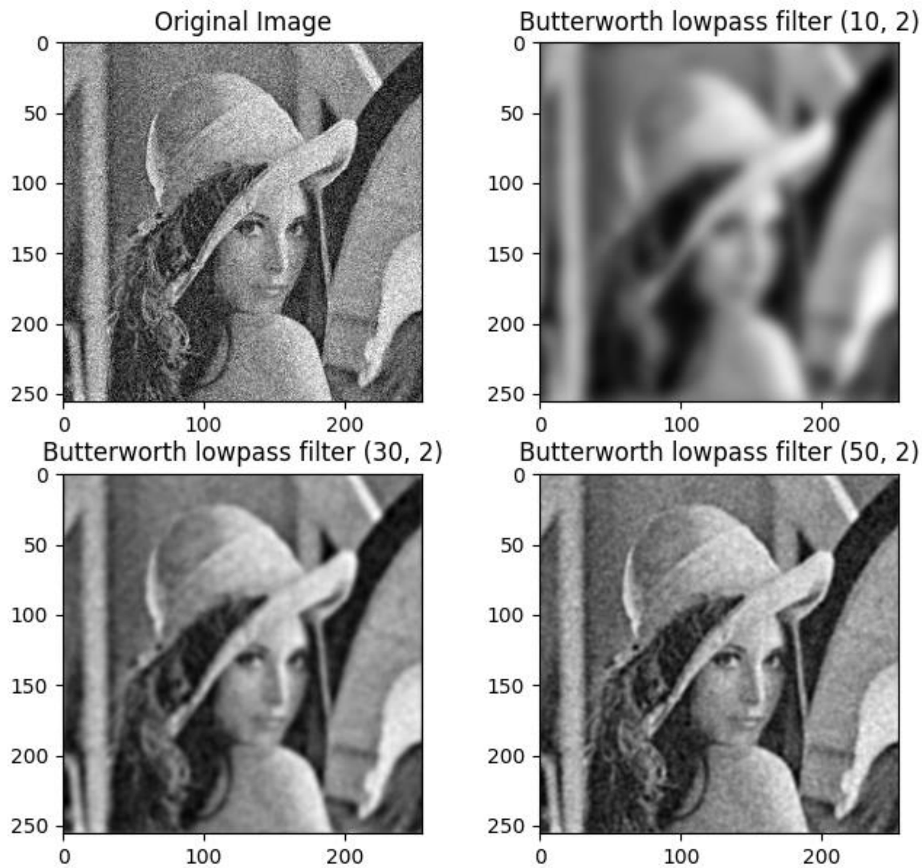
RMS :
Cutoff 10 = 20.637031540529573
Cutoff 30 = 10.693985794239518
Cutoff 50 = 6.95758706317405

Butterworth low-pass filters



"Chess_butterworth"

RMS :
Cutoff 10 = 31.028256261373077
Cutoff 30 = 17.340243872394346
Cutoff 50 = 12.418389589565052



“Lenna_butterworth”

RMS :
Cutoff 10 = 22.093371517384828
Cutoff 30 = 11.725088788971139
Cutoff 50 = 7.864665007120877

Conclusion : After all of experiment I’ve found that Gaussian low-pass filter made the lowest RMS error from the original image.

Code (All implemented in Python)

Properties of the Fourier Transform

1.1

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Read the image
ima = cv2.imread('Cross.pgm', cv2.IMREAD_GRAYSCALE)

# Pad the image to 256x256 (closet 2^n)
padimage = cv2.copyMakeBorder(ima, 28, 28, 28, 28, cv2.BORDER_CONSTANT, value=0)

# Compute the Fourier transform
imagefft = np.fft.fft2(np.float32(padimage))

# Shift the Fourier transform
shiftedfft = np.fft.fftshift(imagefft)

# Compute the magnitude and phase spectra
amp = np.log(np.abs(shiftedfft))
phase = np.angle(shiftedfft)

# Display the results
plt.subplot(121), plt.imshow(amp, cmap='gray')
plt.title('Amplitude spectrum'), plt.xticks([]), plt.yticks([])
plt.subplot(122), plt.imshow(phase, cmap='gray')
plt.title('Phase spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
```

1.2

```
import cv2
import numpy as np
```

```python
from matplotlib import pyplot as plt

# Read the image
ima = cv2.imread('Cross.pgm', cv2.IMREAD_GRAYSCALE)

# Pad the image to 256x256 (closet 2^n)
padimage = cv2.copyMakeBorder(ima, 28, 28, 28, 28, cv2.BORDER_CONSTANT, value=0)

# Compute the Fourier transform
imagefft = np.fft.fft2(np.float32(padimage))

# Shift the Fourier transform
shiftedfft = np.fft.fftshift(imagefft)

# Compute the magnitude and phase spectra
amp = np.log(np.abs(shiftedfft))
phase = np.angle(shiftedfft)

# Multiply the phase spectrum by a complex number to shift the image
shift_phase = np.zeros_like(shiftedfft)
shift_phase[30, 20] = 1
shift_phase = np.exp(1j * np.angle(np.fft.ifftshift(shift_phase)))
shiftedfft *= shift_phase

# Inverse shift the Fourier transform
shift_idft = np.fft.ifftshift(shiftedfft)

# Inverse Fourier transform
shift_idft = np.fft.ifft2(shift_idft)

# Take the real part of the image
shift_idft = np.real(shift_idft)

# Display the shifted image
plt.imshow(shift_idft, cmap='gray')
plt.title('Shifted image')
plt.show()
```

1.3

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Read the image
ima = cv2.imread('Cross.pgm', cv2.IMREAD_GRAYSCALE)

# Define the rotation matrix
rows, cols = ima.shape
M = cv2.getRotationMatrix2D((cols/2,rows/2),30,1)

# Rotate the image
rotated = cv2.warpAffine(ima,M,(cols,rows))

# Compute the Fourier transform
imagefft = np.fft.fft2(np.float32(rotated))

# Shift the Fourier transform
shiftedfft = np.fft.fftshift(imagefft)

# Compute the magnitude and phase spectra
amp = np.log(np.abs(shiftedfft))
phase = np.angle(shiftedfft)

# Display the results
plt.subplot(121), plt.imshow(amp, cmap='gray')
plt.title('Amplitude spectrum rotated'), plt.xticks([]), plt.yticks([])
plt.subplot(122), plt.imshow(phase, cmap='gray')
plt.title('Phase spectrum rotated'), plt.xticks([]), plt.yticks([])
plt.show()
```

1.4

```
import cv2
```

```
import numpy as np
from matplotlib import pyplot as plt

# Read the image
ima = cv2.imread('Cross.pgm', cv2.IMREAD_GRAYSCALE)

# Downsample the image to 100x100 using OpenCV resize function
ima = cv2.resize(ima, (100, 100))

# Compute the Fourier transform
imagefft = np.fft.fft2(np.float32(ima))

# Shift the Fourier transform
shiftedfft = np.fft.fftshift(imagefft)

# Compute the magnitude and phase spectra
amp = np.log(np.abs(shiftedfft))
phase = np.angle(shiftedfft)

# Display the results
plt.subplot(121), plt.imshow(amp, cmap='gray')
plt.title('Amplitude spectrum'), plt.xticks([]), plt.yticks([])
plt.subplot(122), plt.imshow(phase, cmap='gray')
plt.title('Phase spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
```

1.5.1 & 1.6

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Read the image
ima = cv2.imread('Cross.pgm', cv2.IMREAD_GRAYSCALE)
```

```python
# Pad the image to 256x256 (closet 2^n)
padimage = cv2.copyMakeBorder(ima, 28, 28, 28, 28, cv2.BORDER_CONSTANT, value=0)

# Compute the Fourier transform
imagefft = np.fft.fft2(np.float32(padimage))

# Set the phase data to zero
shiftedfft_zero_phase = np.abs(imagefft) * np.exp(0j * np.angle(imagefft))

# Inverse shift the Fourier transform
shift_idft_zero_phase = np.fft.ifftshift(shiftedfft_zero_phase)

# Inverse Fourier transform
image_zero_phase = np.fft.ifft2(shift_idft_zero_phase)

# Take the real part of the image
image_zero_phase = np.real(image_zero_phase)

# Display the result
plt.imshow(image_zero_phase, cmap='gray')
plt.title('Image with no phase')
plt.show()
```

1.5.2 & 1.6

```python
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Read the image
ima = cv2.imread('Cross.pgm', cv2.IMREAD_GRAYSCALE)

# Pad the image to 256x256 (closet 2^n)
padimage = cv2.copyMakeBorder(ima, 28, 28, 28, 28, cv2.BORDER_CONSTANT, value=0)

# Compute the Fourier transform
imagefft = np.fft.fft2(np.float32(padimage))
```

```
# Set the amplitude data to one
shiftedfft_one_amp = np.exp(1j * np.angle(imagefft))

# Inverse shift the Fourier transform
shift_idft_one_amp = np.fft.ifftshift(shiftedfft_one_amp)

# Inverse Fourier transform
image_one_amp = np.fft.ifft2(shift_idft_one_amp)

# Take the real part of the image
image_one_amp = np.real(image_one_amp)

# Display the result
plt.imshow(image_one_amp, cmap='gray')
plt.title('Image with no amplitude')
plt.show()
```

1.7

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Read the image
ima_chess = cv2.imread('Chess.pgm', cv2.IMREAD_GRAYSCALE)

# Define a small kernel
kernel = np.ones((3,3)) / 9

# Perform convolution on the image
ima_conv = cv2.filter2D(ima_chess, -1, kernel)

# Compute the Fourier transform of the kernel
kernel_fft = np.fft.fft2(kernel, s=ima_chess.shape)

# Shift the Fourier transform
```

```
shifted_kernel_fft = np.fft.fftshift(kernel_fft)

# Compute the Fourier transform of the image
ima_fft = np.fft.fft2(np.float32(ima_chess))

# Shift the Fourier transform
shifted_ima_fft = np.fft.fftshift(ima_fft)

# Filter in frequency domain
ima_blur_fft = shifted_ima_fft * shifted_kernel_fft

# Inverse shift the Fourier transform
ima_blur_ifft = np.fft.ifftshift(ima_blur_fft)

# Inverse Fourier transform
ima_blur_ifft = np.fft.ifft2(ima_blur_ifft)

# Take the absolute value of the image
ima_blur_ifft = np.abs(ima_blur_ifft)

# Display the results
plt.subplot(131), plt.imshow(ima_chess, cmap='gray')
plt.title('Original image'), plt.xticks([]), plt.yticks([])
plt.subplot(132), plt.imshow(ima_conv, cmap='gray')
plt.title('Convolution'), plt.xticks([]), plt.yticks([])
plt.subplot(133), plt.imshow(ima_blur_ifft, cmap='gray')
plt.title('Frequency domain'), plt.xticks([]), plt.yticks([])
plt.show()
```

2.1 ideal

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

# Load image
img = cv2.imread('Cross.pgm', cv2.IMREAD_GRAYSCALE)
```

```
# Get image dimensions
X, Y = img.shape

# Compute Fourier transform
img_fft = np.fft.fft2(img)

# Display original image
plt.subplot(2, 3, 1)
plt.imshow(img, cmap='gray')
plt.title('origin')

# Compute distances in frequency domain
u = np.arange(X)
v = np.arange(Y)
u[np.where(u > X/2)] -= X
v[np.where(v > Y/2)] -= Y
U, V = np.meshgrid(u, v)
D = np.sqrt(U**2 + V**2)

# Ideal low pass filter
H10 = np.double(D <= 10)
G10 = H10 * img_fft
ideal10 = np.fft.ifft2(G10)
plt.subplot(2, 3, 2)
plt.imshow(np.abs(ideal10), cmap='gray')
plt.title('ideal low pass (10)')

H20 = np.double(D <= 20)
G20 = H20 * img_fft
ideal20 = np.fft.ifft2(G20)
plt.subplot(2, 3, 3)
plt.imshow(np.abs(ideal20), cmap='gray')
plt.title('ideal low pass (20)')

H30 = np.double(D <= 30)
G30 = H30 * img_fft
```

```
ideal30 = np.fft.ifft2(G30)
plt.subplot(2, 3, 4)
plt.imshow(np.abs(ideal30), cmap='gray')
plt.title('ideal low pass (30)')

H40 = np.double(D <= 40)
G40 = H40 * img_fft
ideal40 = np.fft.ifft2(G40)
plt.subplot(2, 3, 5)
plt.imshow(np.abs(ideal40), cmap='gray')
plt.title('ideal low pass (40)')

H50 = np.double(D <= 50)
G50 = H50 * img_fft
ideal50 = np.fft.ifft2(G50)
plt.subplot(2, 3, 6)
plt.imshow(np.abs(ideal50), cmap='gray')
plt.title('ideal low pass (50)')


plt.show()
```

2.1 Gaussian

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

# Load image
img = cv2.imread('Cross.pgm', cv2.IMREAD_GRAYSCALE)

# Get image dimensions
X, Y = img.shape

# Compute Fourier transform
img_fft = np.fft.fft2(img)
```

```
# Display original image
plt.subplot(2, 3, 1)
plt.imshow(img, cmap='gray')
plt.title('Original Image')

# Compute distances in frequency domain
u = np.arange(X)
v = np.arange(Y)
u[np.where(u > X/2)] -= X
v[np.where(v > Y/2)] -= Y
U, V = np.meshgrid(u, v)
D = np.sqrt(U**2 + V**2)

# Gaussian lowpass filter
sigma_values = [10, 20, 30, 40, 50]
for i, sigma in enumerate(sigma_values):
    H_glf = np.exp(-(D**2)/(2*sigma**2))
    G_glf = H_glf * img_fft
    glf = np.fft.ifft2(G_glf)
    plt.subplot(2, 3, i+2)
    plt.imshow(np.abs(glf), cmap='gray')
    plt.title('Gaussian lowpass filter ({})'.format(sigma))

plt.show()
```

2.1 Butterworth

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

# Load image
img = cv2.imread('Cross.pgm', cv2.IMREAD_GRAYSCALE)

# Get image dimensions
X, Y = img.shape
```

```
# Compute Fourier transform
img_fft = np.fft.fft2(img)

# Display original image
plt.subplot(2, 3, 1)
plt.imshow(img, cmap='gray')
plt.title('Original Image')

# Compute distances in frequency domain
u = np.arange(X)
v = np.arange(Y)
u[np.where(u > X/2)] -= X
v[np.where(v > Y/2)] -= Y
U, V = np.meshgrid(u, v)
D = np.sqrt(U**2 + V**2)

# Butterworth lowpass filter
cutoff = 10
n = 2
H_blf = 1 / (1 + (D / cutoff)**(2*n))
G_blf = H_blf * img_fft
blf = np.fft.ifft2(G_blf)
plt.subplot(2, 3, 2)
plt.imshow(np.abs(blf), cmap='gray')
plt.title('Butterworth lowpass filter (10, 2)')

cutoff = 20
n = 2
H_blf = 1 / (1 + (D / cutoff)**(2*n))
G_blf = H_blf * img_fft
blf = np.fft.ifft2(G_blf)
plt.subplot(2, 3, 3)
plt.imshow(np.abs(blf), cmap='gray')
plt.title('Butterworth lowpass filter (20, 2)')

cutoff = 30
n = 2
```

```
H_blf = 1 / (1 + (D / cutoff)**(2*n))
G_blf = H_blf * img_fft
blf = np.fft.ifft2(G_blf)
plt.subplot(2, 3, 4)
plt.imshow(np.abs(blf), cmap='gray')
plt.title('Butterworth lowpass filter (30, 2)')

cutoff = 40
n = 2
H_blf = 1 / (1 + (D / cutoff)**(2*n))
G_blf = H_blf * img_fft
blf = np.fft.ifft2(G_blf)
plt.subplot(2, 3, 5)
plt.imshow(np.abs(blf), cmap='gray')
plt.title('Butterworth lowpass filter (40, 2)')

cutoff = 50
n = 2
H_blf = 1 / (1 + (D / cutoff)**(2*n))
G_blf = H_blf * img_fft
blf = np.fft.ifft2(G_blf)
plt.subplot(2, 3, 6)
plt.imshow(np.abs(blf), cmap='gray')
plt.title('Butterworth lowpass filter (50, 2)')

plt.show()
```

2.2 Ideal & RMS calculated

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

# Load image
img = cv2.imread('Lenna_noise.pgm', cv2.IMREAD_GRAYSCALE)

# Get image dimensions
```

```
X, Y = img.shape

# Compute Fourier transform
img_fft = np.fft.fft2(img)

# Display original image
plt.subplot(2, 2, 1)
plt.imshow(img, cmap='gray')
plt.title('origin')

# Compute distances in frequency domain
u = np.arange(X)
v = np.arange(Y)
u[np.where(u > X/2)] -= X
v[np.where(v > Y/2)] -= Y
U, V = np.meshgrid(u, v)
D = np.sqrt(U**2 + V**2)

# Ideal low pass filter
H10 = np.double(D <= 10)
G10 = H10 * img_fft
ideal10 = np.fft.ifft2(G10)
plt.subplot(2, 2, 2)
plt.imshow(np.abs(ideal10), cmap='gray')
plt.title('ideal low pass (10)')

H30 = np.double(D <= 30)
G30 = H30 * img_fft
ideal30 = np.fft.ifft2(G30)
plt.subplot(2, 2, 3)
plt.imshow(np.abs(ideal30), cmap='gray')
plt.title('ideal low pass (30)')

H50 = np.double(D <= 50)
G50 = H50 * img_fft
ideal50 = np.fft.ifft2(G50)
plt.subplot(2, 2, 4)
```

```
plt.imshow(np.abs(ideal50), cmap='gray')
plt.title('ideal low pass (50)')



plt.show()

#RMS calculate
def compare_cutoff_levels(image_path, cutoff_freqs):
    # Load image with noise
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    # Compute Fourier transform
    img_fft = np.fft.fft2(img)

    # Compute distances in frequency domain
    X, Y = img.shape
    u = np.arange(X)
    v = np.arange(Y)
    u[np.where(u > X/2)] -= X
    v[np.where(v > Y/2)] -= Y
    U, V = np.meshgrid(u, v)
    D = np.sqrt(U**2 + V**2)

    # Calculate RMS errors for each cutoff frequency
    rms_errors = []
    for cutoff_freq in cutoff_freqs:
        H = np.double(D <= cutoff_freq)
        G = H * img_fft
        filtered_img = np.fft.ifft2(G).real

        # Compute RMS error
        error = np.sqrt(np.mean((img - filtered_img)**2))
        rms_errors.append(error)

    return rms_errors
```

```
cutoff_freqs = [10, 30, 50]
rms_errors = compare_cutoff_levels('Lenna.pgm', cutoff_freqs)
print(rms_errors)
```

2.2 Gaussian & RMS calculated

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

# Load image
img = cv2.imread('Lenna_noise.pgm', cv2.IMREAD_GRAYSCALE)

# Get image dimensions
X, Y = img.shape

# Compute Fourier transform
img_fft = np.fft.fft2(img)

# Display original image
plt.subplot(2, 2, 1)
plt.imshow(img, cmap='gray')
plt.title('Original Image')

# Compute distances in frequency domain
u = np.arange(X)
v = np.arange(Y)
u[np.where(u > X/2)] -= X
v[np.where(v > Y/2)] -= Y
U, V = np.meshgrid(u, v)
D = np.sqrt(U**2 + V**2)

# Gaussian lowpass filter
sigma_values = [10, 30, 50]
for i, sigma in enumerate(sigma_values):
    H_glf = np.exp(-(D**2)/(2*sigma**2))
    G_glf = H_glf * img_fft
```

```python
    glf = np.fft.ifft2(G_glf)
    plt.subplot(2, 2, i+2)
    plt.imshow(np.abs(glf), cmap='gray')
    plt.title('Gaussian lowpass filter ({})'.format(sigma))

plt.show()

#RMS calculate
def compare_cutoff_levels(image_path, sigma_values):
    # Load image with noise
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    # Compute Fourier transform
    img_fft = np.fft.fft2(img)

    # Compute distances in frequency domain
    X, Y = img.shape
    u = np.arange(X)
    v = np.arange(Y)
    u[np.where(u > X/2)] -= X
    v[np.where(v > Y/2)] -= Y
    U, V = np.meshgrid(u, v)
    D = np.sqrt(U**2 + V**2)

    # Calculate RMS errors for each cutoff frequency
    rms_errors = []
    for sigma in sigma_values:
        H = np.exp(-(D**2)/(2*sigma**2))
        G = H * img_fft
        filtered_img = np.fft.ifft2(G).real

        # Compute RMS error
        error = np.sqrt(np.mean((img - filtered_img)**2))
        rms_errors.append(error)

    return rms_errors
```

```
sigma_values = [10, 30, 50]
rms_errors = compare_cutoff_levels('Lenna.pgm', sigma_values)
print(rms_errors)
```

2.2 Butterworth & RMS calculated

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

# Load image
img = cv2.imread('Lenna_noise.pgm', cv2.IMREAD_GRAYSCALE)

# Get image dimensions
X, Y = img.shape

# Compute Fourier transform
img_fft = np.fft.fft2(img)

# Display original image
plt.subplot(2, 2, 1)
plt.imshow(img, cmap='gray')
plt.title('Original Image')

# Compute distances in frequency domain
u = np.arange(X)
v = np.arange(Y)
u[np.where(u > X/2)] -= X
v[np.where(v > Y/2)] -= Y
U, V = np.meshgrid(u, v)
D = np.sqrt(U**2 + V**2)

# Butterworth lowpass filter
cutoff = 10
n = 2
H_blf = 1 / (1 + (D / cutoff)**(2*n))
```

```
G_blf = H_blf * img_fft
blf = np.fft.ifft2(G_blf)
plt.subplot(2, 2, 2)
plt.imshow(np.abs(blf), cmap='gray')
plt.title('Butterworth lowpass filter (10, 2)')


cutoff = 30
n = 2
H_blf = 1 / (1 + (D / cutoff)**(2*n))
G_blf = H_blf * img_fft
blf = np.fft.ifft2(G_blf)
plt.subplot(2, 2, 3)
plt.imshow(np.abs(blf), cmap='gray')
plt.title('Butterworth lowpass filter (30, 2)')


cutoff = 50
n = 2
H_blf = 1 / (1 + (D / cutoff)**(2*n))
G_blf = H_blf * img_fft
blf = np.fft.ifft2(G_blf)
plt.subplot(2, 2, 4)
plt.imshow(np.abs(blf), cmap='gray')
plt.title('Butterworth lowpass filter (50, 2)')

plt.show()

def compare_cutoff_levels(image_path, cutoff_freqs):
  # Load image with noise
  img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

  # Compute Fourier transform
  img_fft = np.fft.fft2(img)

  # Compute distances in frequency domain
  X, Y = img.shape
```

```
    u = np.arange(X)
    v = np.arange(Y)
    u[np.where(u > X/2)] -= X
    v[np.where(v > Y/2)] -= Y
    U, V = np.meshgrid(u, v)
    D = np.sqrt(U**2 + V**2)

    # Calculate RMS errors for each cutoff frequency
    rms_errors = []
    for cutoff_freq in cutoff_freqs:
        n = 2
        H = 1 / (1 + (D / cutoff_freq)**(2*n))
        G = H * img_fft
        filtered_img = np.fft.ifft2(G).real

        # Compute RMS error
        error = np.sqrt(np.mean((img - filtered_img)**2))
        rms_errors.append(error)

    return rms_errors

cutoff_freqs = [10, 30, 50]
rms_errors = compare_cutoff_levels('Lenna.pgm', cutoff_freqs)
print(rms_errors)
```