ST. FRANCIS INSTITUTE OF TECHNOLOGY

Mount Poinsur, S.V.P. Road, Borivali (W), Mumbai - 400 103.

CLASS / BATCH: TE-ExTCA /TA-3 SEM: VI

NAME: Om Kadam ROLL NO: 41

SUBJECT: IPMV

EXPERIMENT NO. : 04

TITLE: To perform Image enhancement in frequency domain

DATE OF PERFORMANCE:

DATE OF SUBMISSION:

Correction Parameters	Marks Allotted	Correction Parameters	Marks Allotted
Submission on time [10%]		Experimental Result [30%]	
Conclusion / Result Analysis/ Discussion [30%]		Post Experiment Exercise [20%]	
Presentation of Write-Up [10%]			

TOTAL MARKS:

FACULTY SIGNATURE (WITH DATE)

Roll No:

Experiment –04: To perform Image Enhancement in Frequency Domain

Date:

1. **Aim:** To perform Image Enhancement (Ideal Low Pass Filter and Butterworth High Pass Filter) in Frequency Domain

2. **Requirements:** Python

3. **Pre-Experiment Exercise**
   3.1 Brief Theory

Filtering in the frequency domain is a common image and signal processing technique. It can smooth, sharpen, de-blur, and restore some images. Essentially, filtering is equal to convolving a function with a specific filter function. So one possibility to convolve two functions could be to transform them to the frequency domain, multiply them there and transform them back to spatial domain. The filtering procedure is summarized in Fig. 1.

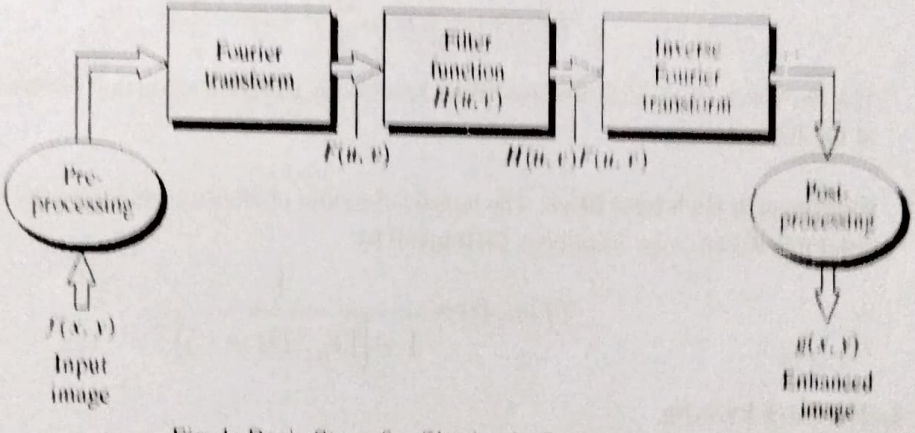


Fig. 1: Basic Steps for filtering in Frequency domain

Basic steps of filtering in the frequency domain:

1. Multiply the input image $f(x, y)$ by $(-1)^{(x+y)}$ to center the transform.
2. Compute $F(u, v)$, the DFT of the input image from (1).
3. Multiply $F(u, v)$ by a *filter* function $H(u, v)$.
4. Compute the inverse DFT of the result in (3).
5. Obtain the real part (better take the magnitude) of the result in (4).
6. Multiply the result in (5) by $(-1)^{(x+y)}$.

The ***Discrete Fourier Transform*** (DFT) is the most important discrete transform, used to perform Fourier analysis in many practical applications. In image processing, the samples can be the values of pixels along a row or column of a raster image. The 2D Discrete Fourier Transform is given by the equation:

$$F(u,v)=\frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}f(x,y)e^{-j2\pi(ux/M+vy/N)}$$

and its inverse:

$$f(x,y)=\frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1}F(u,v)e^{j2\pi(ux/M+vy/N)}$$

Ideal Low Pass Filter: The simplest low pass filter is the ideal low pass. It suppresses all frequencies higher than the cut-off frequency D0 and leaves smaller frequencies unchanged.

$$H(u,v)=\begin{cases}1 \text{ if } D(u,v)\le D_0\\ 0 \text{ if } D(u,v)>D_0\end{cases}$$

D0 is called the *cutoff frequency* (nonnegative quantity), and D(u,v) is the distance from point (u,v) to the frequency rectangle.

$$D(u,v)=\sqrt{(u-\frac{M}{2})^2+(v-\frac{N}{2})^2}$$

The drawback of the ideal low pass filter function is a ringing effect that occurs along the edges of the filtered image.

Butterworth High pass filter: The transfer function of Butterworth high pass filter (BHPF) of order n and with *cutoff frequency* D0 is given by

$$H(u,v)=\frac{1}{1+\left[D_0/D(u,v)\right]^{2n}}$$

4. Laboratory Exercise

4.1 Algorithm:

1. Take a gray scale image as an input.
2. Apply basic steps of filtering in frequency domain for ideal low pass filter and Butterworth high pass filter.
3. Display all intermediate (Fourier transform image, Filter function H(u,v), Filtered output) images.

5. Post-Experiment Exercise

5.1 Conclusion:

This experiment successfully showcased the efficacy of frequency domain techniques for image enhancement. Fourier transformation facilitated selective enhancement of specific frequency components, resulting in improved image quality and detail preservation.

5.2 Questions:

1. Apply DFT and IDFT transform on given image :

$$f(x,y) = \begin{matrix} 2 & 2 & 2 & 1 \\ 2 & 4 & 4 & 2 \\ 2 & 4 & 4 & 2 \\ 2 & 2 & 2 & 2 \end{matrix}$$

2. What is the difference between Ideal, Butterworth and Gaussian smoothening filters?

D) $f(x,y) = \begin{bmatrix} 2 & 2 & 2 & 1 \\ 2 & 4 & 4 & 2 \\ 2 & 4 & 4 & 2 \\ 2 & 2 & 2 & 2 \end{bmatrix}$

Applying DFT,

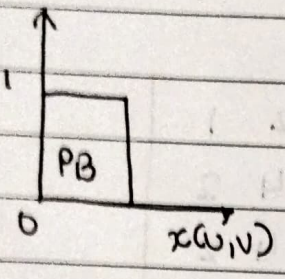
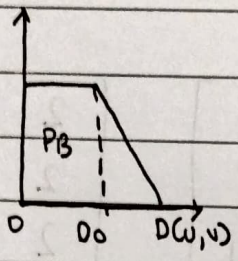
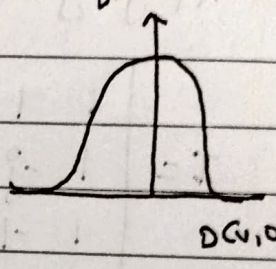
$$I = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 2 & 2 & 2 & 1 \\ 2 & 4 & 4 & 2 \\ 2 & 4 & 4 & 2 \\ 2 & 2 & 2 & 2 \end{bmatrix}$$

$$= \begin{bmatrix} 8 & 12 & 12 & 7 \\ 0 & -2-2j & -2-2j & -1 \\ 0 & 0 & 0 & -1 \\ 0 & -2+2j & -2+2j & -1 \end{bmatrix}$$

Applying IDFT,

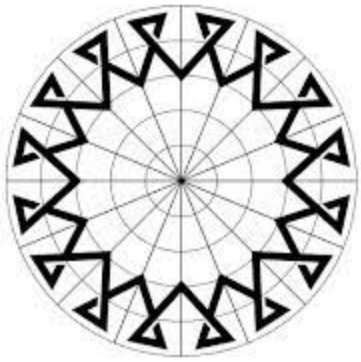
$$A = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix} \begin{bmatrix} 8 & 12 & 12 & 7 \\ 0 & -2-2j & -2-2j & -1 \\ 0 & 0 & 0 & -1 \\ 0 & -2+2j & -2+2j & -1 \end{bmatrix}$$

$$= \frac{1}{4} \begin{bmatrix} 8 & 8 & 8 & 4 \\ 8 & 16 & 16 & 8 \\ 8 & 16 & 16 & 8 \\ 8 & 8 & 8 & 8 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 2 & 1 \\ 2 & 4 & 4 & 2 \\ 2 & 4 & 4 & 2 \\ 2 & 2 & 2 & 2 \end{bmatrix}$$

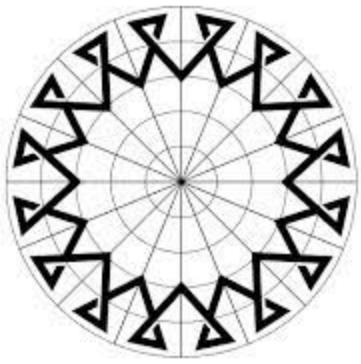
2)	Ideal	Butterworth	Gaussian
①	It has perfectly flat pair band with unity gain	It provides max flat freq response in passband	It has bell-shaped freq response in freq domain
②			
③	It has sharp cutoff	It has a more gradual transition between passband & stopband	It provides a smooth transition between pass-band & stopband.
④	It has less control over cutoff order	It has more control over cutoff	It has less control over cutoff

In [ ]:
```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
import matplotlib.pyplot as plt
from io import BytesIO
from PIL import Image
from google.colab import files
from skimage.io import imread
import matplotlib.pyplot as plt
import scipy.fftpack as fp
from math import sqrt,exp
```

In [ ]:
```
img=cv2.imread('csym.jpg')
cv2_imshow(img)
```



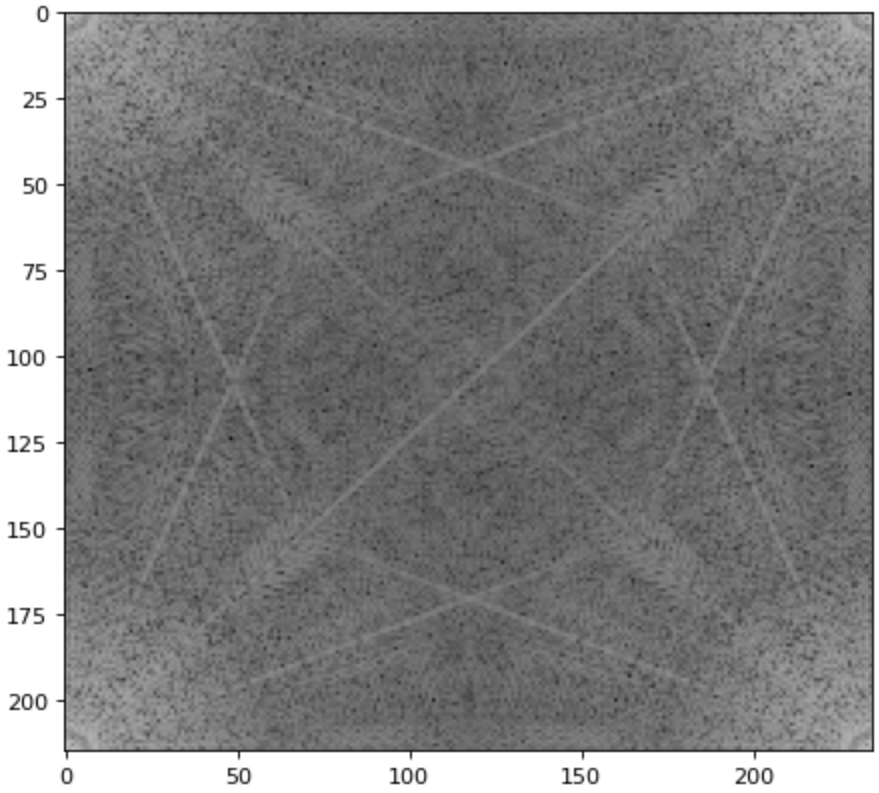
In [ ]:
```
img.shape
gray_img=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
cv2_imshow(gray_img)
[m,n]=gray_img.shape
m,n
```



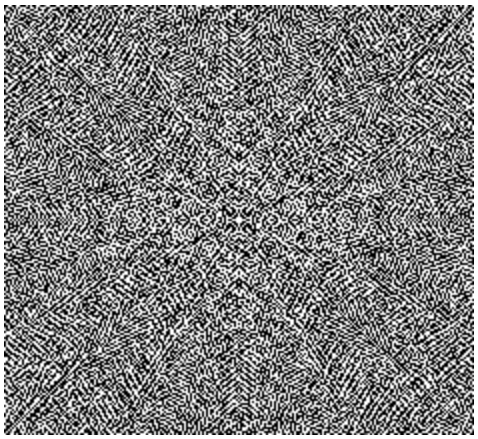
Out[ ]: (215, 235)

In [ ]:
```
img_fft=np.fft.fft2(gray_img)
plt.figure(num=None, figsize=(8, 6), dpi=80)
plt.imshow(np.log(abs(img_fft)), cmap='gray')
```

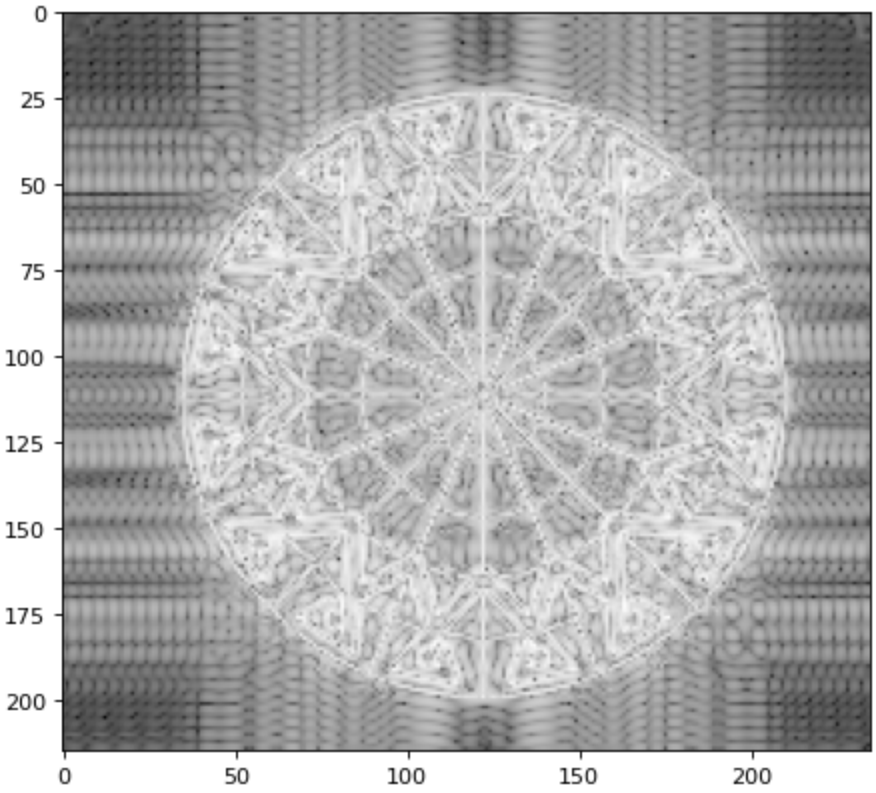
Out[ ]: <matplotlib.image.AxesImage at 0x7bfaac7fe4a0>



In [ ]:
```
img_fftshift=np.fft.fftshift(img_fft)
cv2_imshow(img_fftshift)
```

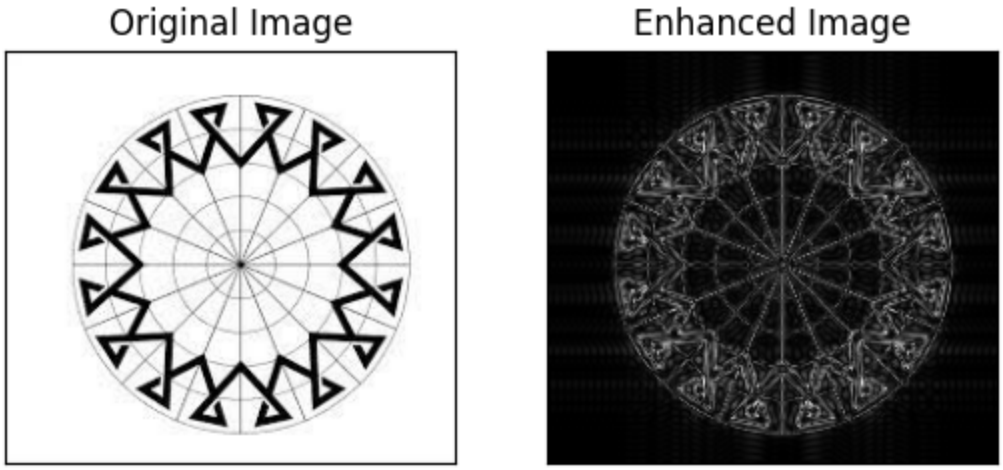


In [ ]:
```
crow, ccol = m // 2, n // 2
img_fftshift[crow - 30:crow + 30, ccol - 30:ccol + 30] = 0
img_Ifft=np.fft.ifft2(img_fftshift)
plt.figure(num=None, figsize=(8, 6), dpi=80)
plt.imshow(np.log(abs(img_Ifft)), cmap='gray');
cv2_imshow(img_Ifft)
```



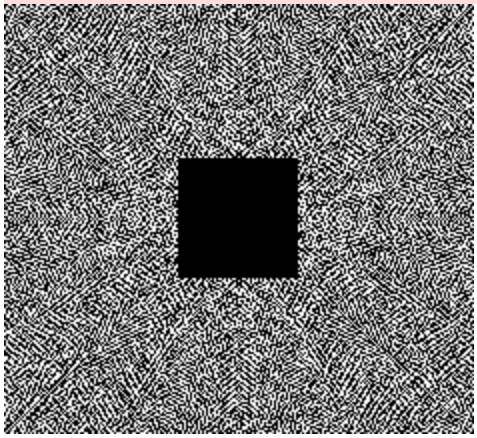
In [ ]:
```
f_ishift = np.fft.ifftshift(img_fftshift)
image_back = np.fft.ifft2(f_ishift)
image_back = np.abs(image_back)
```

In [ ]:
```
plt.subplot(1, 2, 1), plt.imshow(img, cmap='gray')
plt.title('Original Image')
plt.xticks([]), plt.yticks([])

plt.subplot(1, 2, 2), plt.imshow(image_back, cmap='gray')
plt.title('Enhanced Image')
plt.xticks([]), plt.yticks([])

plt.show()
```

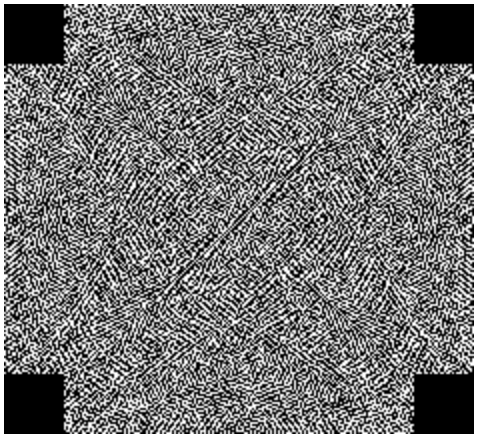


In [ ]:
```
cuttoff=5
J=np.zeros([m,n])
for x in range (m):
  for y in range (n):
    distance=np.sqrt((x-m/2)**2+(y-n/2)**2)
    if distance > cuttoff:
     J[(x,y)]=img_fftshift[(x,y)]
cv2_imshow(J)
```

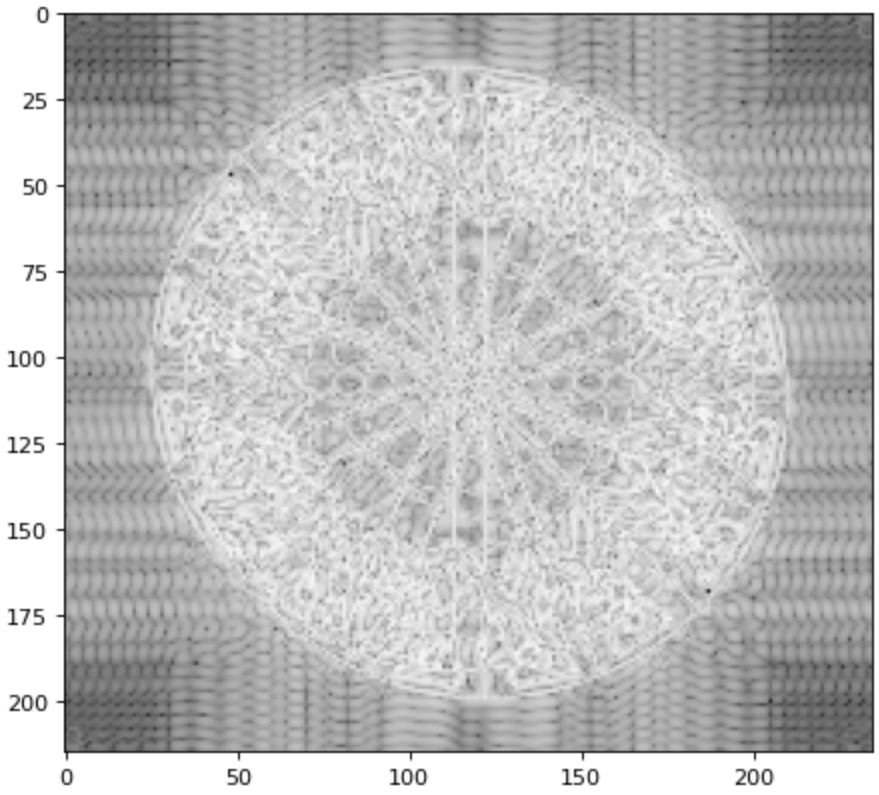
```
<ipython-input-71-041a68fabbfe>:7: ComplexWarning: Casting complex values to real di
scards the imaginary part
  J[(x,y)]=img_fftshift[(x,y)]
```



In [ ]:
```
img_Ifftshift=np.fft.ifftshift(J)
cv2_imshow(img_Ifftshift)
```



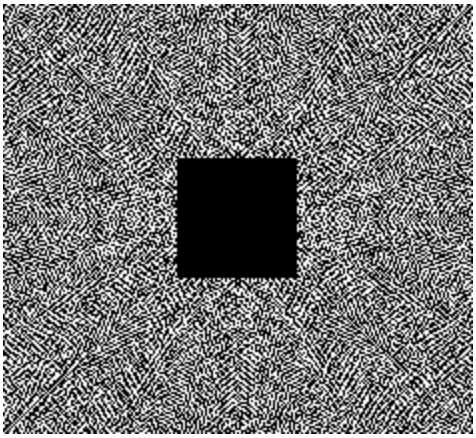
In [ ]:
```
#HIGH PASS
img_Ifft=np.fft.ifft2(img_Ifftshift)
plt.figure(num=None, figsize=(8, 6), dpi=80)
plt.imshow(np.log(abs(img_Ifft)), cmap='gray');
cv2_imshow(img_Ifft)
```

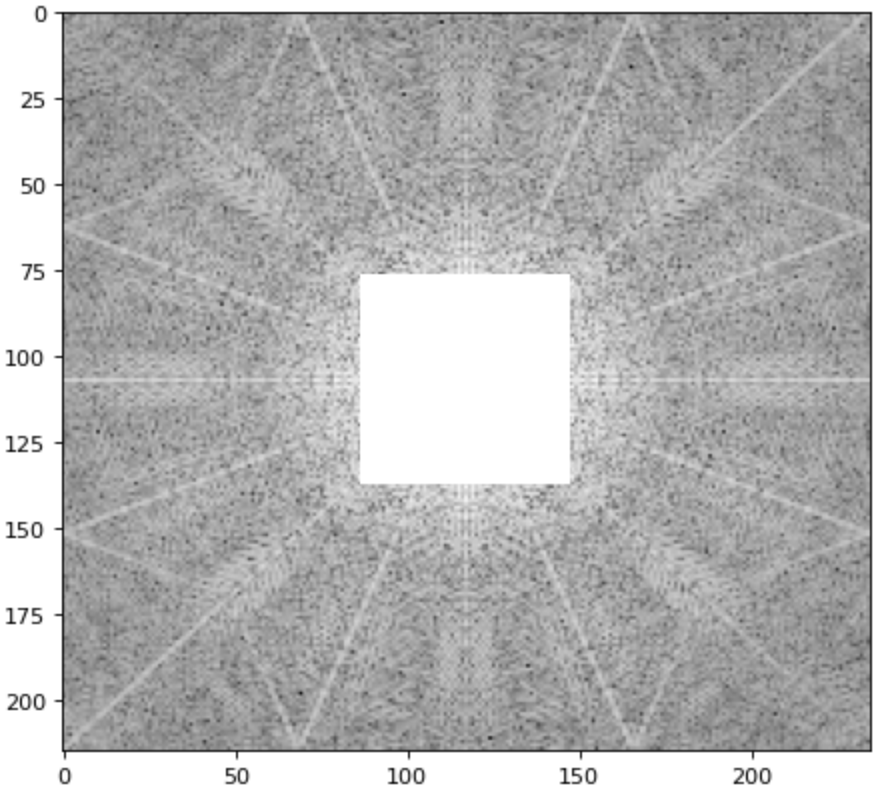


In [ ]:
```
m, n = img.shape[:2]

cutoff = 5

center_x, center_y = m // 2, n // 2

J = np.zeros([m, n], dtype=np.complex64)

for x in range(m):
    for y in range(n):
        distance = np.sqrt((x - center_x) ** 2 + (y - center_y) ** 2)
        if distance > cutoff:
            J[x, y] = img_fftshift[x, y]
```

In [ ]:
```
# Display J without inversion
plt.figure(num=None, figsize=(8, 6), dpi=80)
plt.imshow(np.log(abs(J)), cmap='gray');
cv2_imshow(J)
```

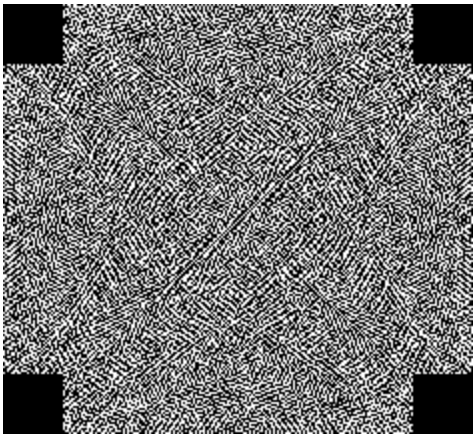
```
<ipython-input-75-5fcfb5719e0e>:3: RuntimeWarning: divide by zero encountered in log
  plt.imshow(np.log(abs(J)), cmap='gray');
```

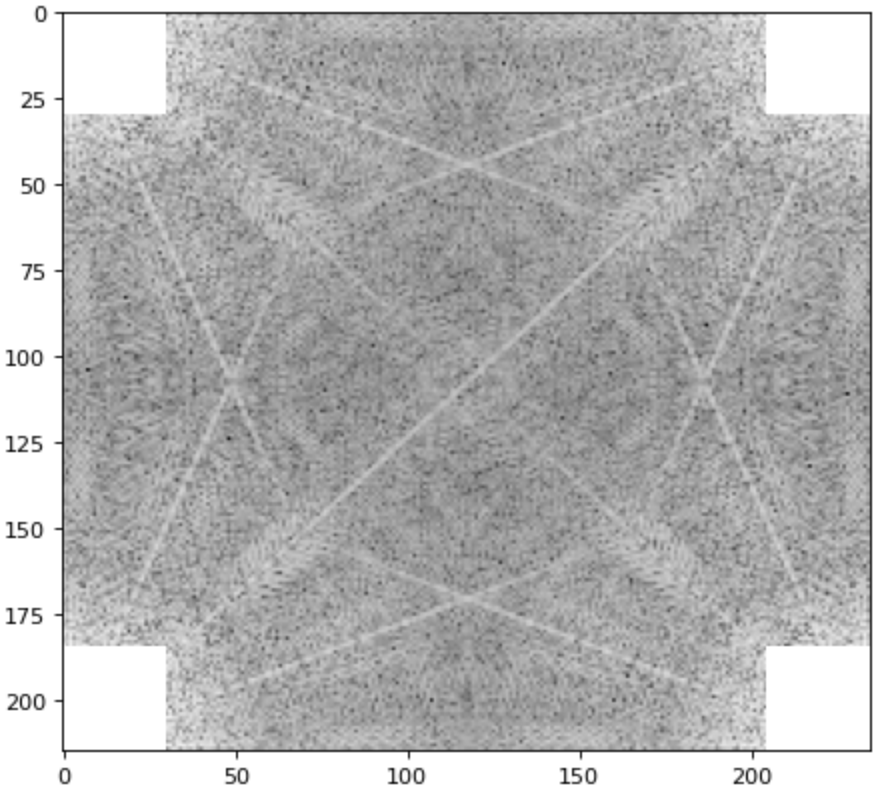




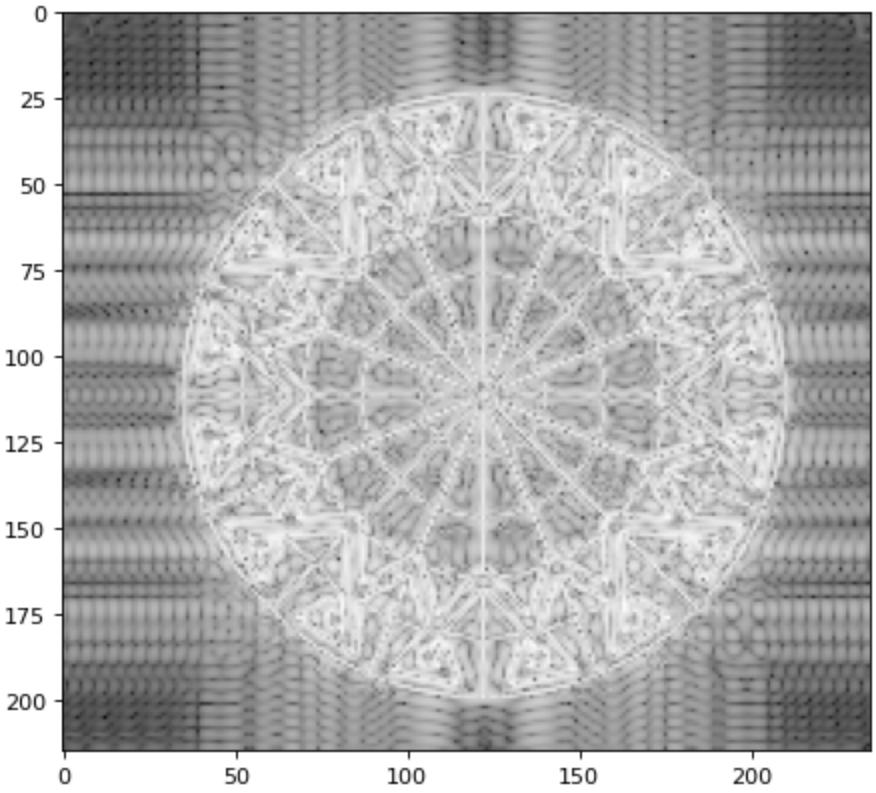
In [ ]:
```
# Perform inverse FFT shift
img_Ifftshift = np.fft.ifftshift(J)

# Display img_Ifftshift without inversion
plt.figure(num=None, figsize=(8, 6), dpi=80)
plt.imshow(np.log(abs(img_Ifftshift)), cmap='gray');
cv2_imshow(img_Ifftshift)
```

```
<ipython-input-76-b99df5fd616d>:6: RuntimeWarning: divide by zero encountered in log
  plt.imshow(np.log(abs(img_Ifftshift)), cmap='gray');
```





In [ ]:
```
# Perform inverse FFT
img_Ifft = np.fft.ifft2(img_Ifftshift)

# Display magnitude of the inverse FFT
img_Ifft=np.fft.ifft2(img_Ifftshift)
plt.figure(num=None, figsize=(8, 6), dpi=80)
plt.imshow(np.log(abs(img_Ifft)), cmap='gray');
cv2_imshow(img_Ifft)
```

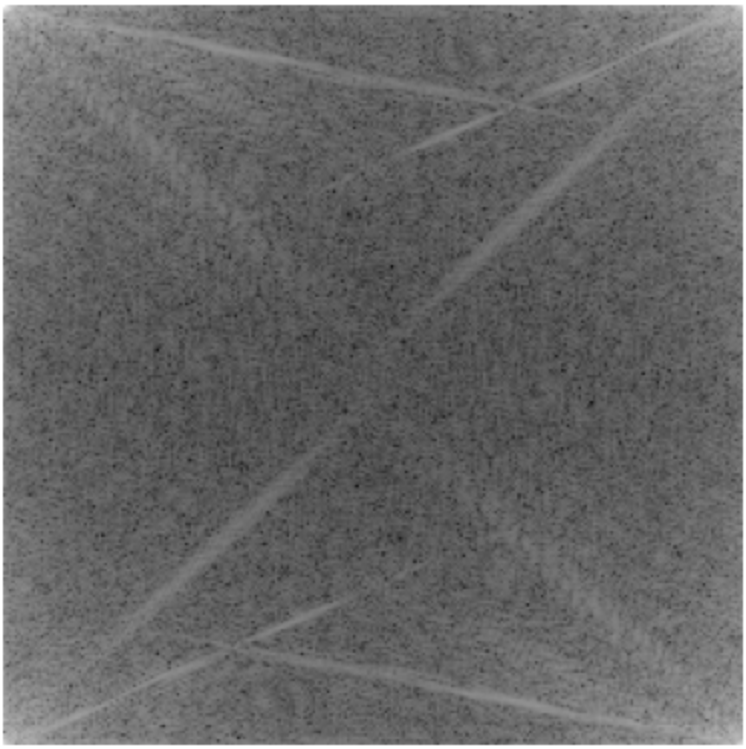


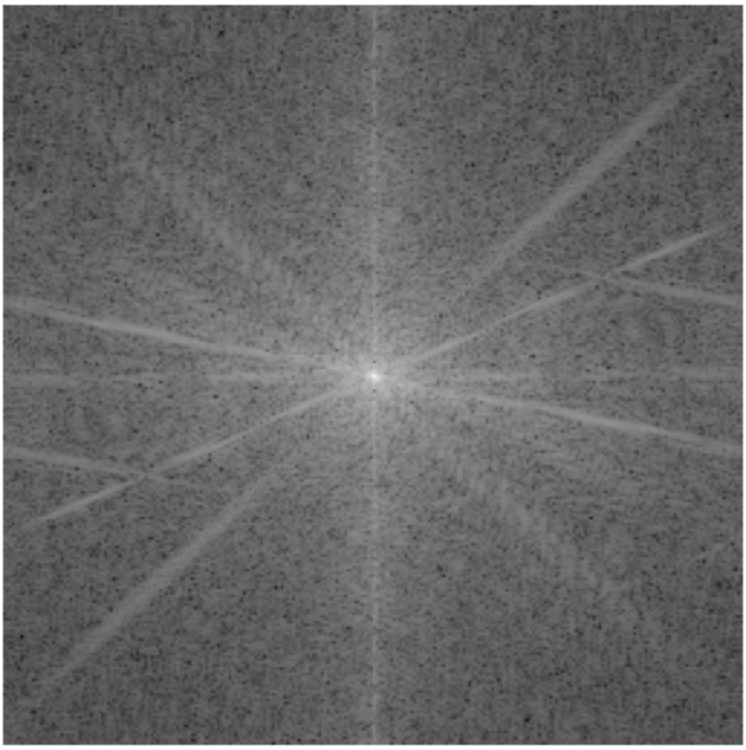
In [ ]:
```
# https://github.com/adenarayana
```

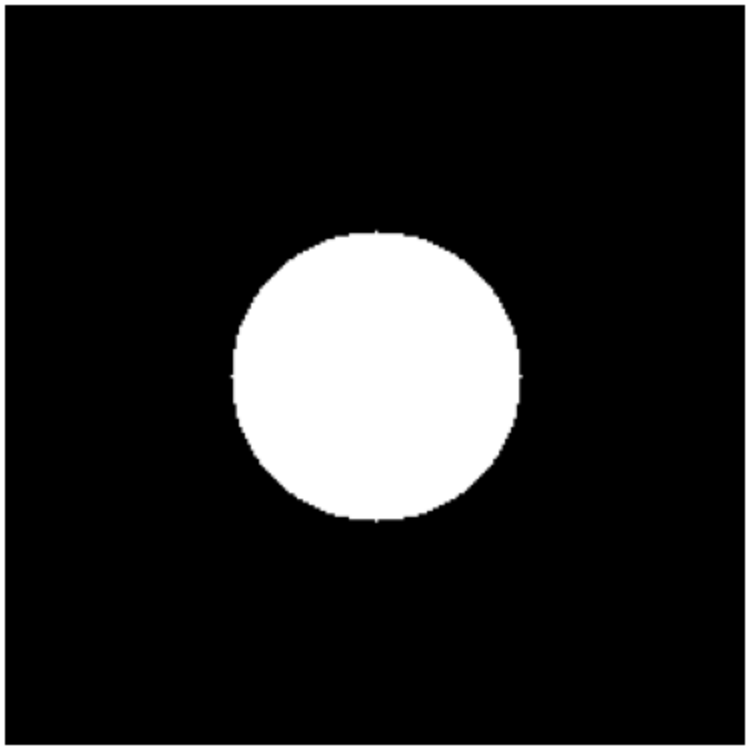
In [ ]:
```
# libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt

# original image
f = cv2.imread('cameraman.tif',0)

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

# image in frequency domain
F = np.fft.fft2(f)
plt.imshow(np.log1p(np.abs(F)),
           cmap='gray')
```

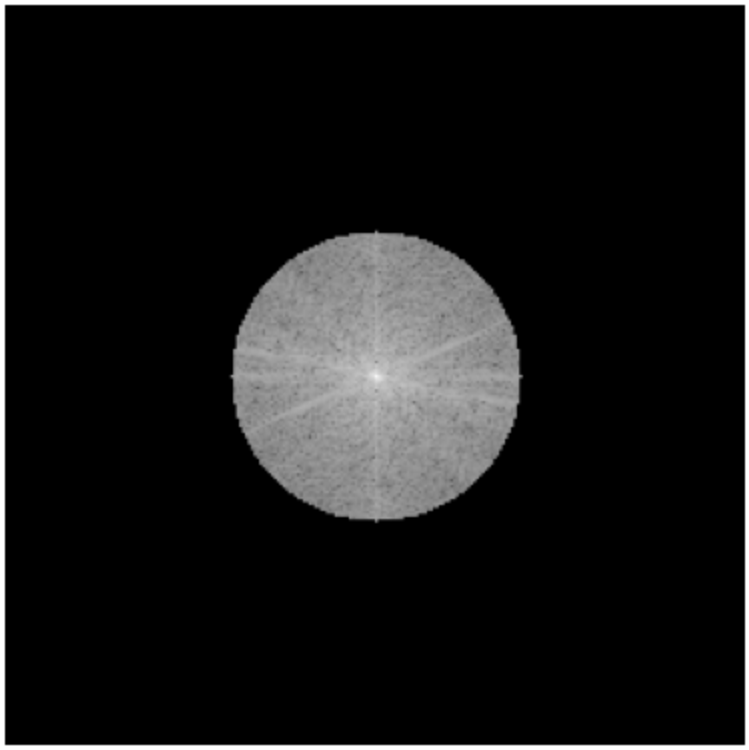
```
plt.axis('off')
plt.show()

Fshift = np.fft.fftshift(F)
plt.imshow(np.log1p(np.abs(Fshift)),
           cmap='gray')
plt.axis('off')
plt.show()

# Filter: Low pass filter
M,N = f.shape
H = np.zeros((M,N), dtype=np.float32)
D0 = 50
for u in range(M):
    for v in range(N):
        D = np.sqrt((u-M/2)**2 + (v-N/2)**2)
        if D <= D0:
            H[u,v] = 1
        else:
            H[u,v] = 0

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

# Ideal Low Pass Filtering
Gshift = Fshift * H
plt.imshow(np.log1p(np.abs(Gshift)),
           cmap='gray')
plt.axis('off')
plt.show()

# Inverse Fourier Transform
G = np.fft.ifftshift(Gshift)
plt.imshow(np.log1p(np.abs(G)),
           cmap='gray')
plt.axis('off')
plt.show()

g = np.abs(np.fft.ifft2(G))
plt.imshow(g, cmap='gray')
plt.axis('off')
plt.show()


# Filter: High pass filter
H = 1 - H

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

# Ideal High Pass Filtering
Gshift = Fshift * H
plt.imshow(np.log1p(np.abs(Gshift)),
           cmap='gray')
```

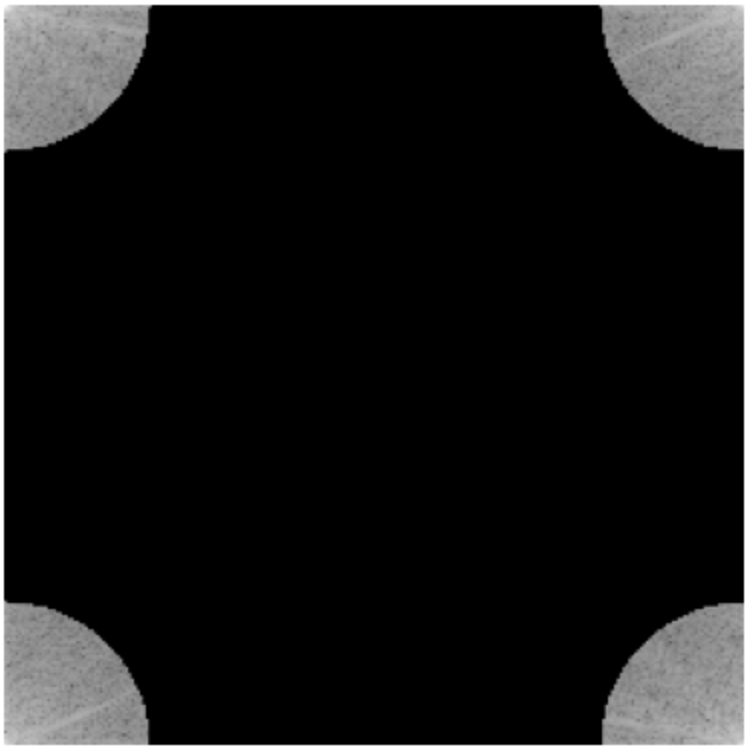
```
plt.axis('off')
plt.show()

# Inverse Fourier Transform
G = np.fft.ifftshift(Gshift)
plt.imshow(np.log1p(np.abs(G)),
           cmap='gray')
plt.axis('off')
plt.show()

g = np.abs(np.fft.ifft2(G))
plt.imshow(g, cmap='gray')
plt.axis('off')
plt.show()
```

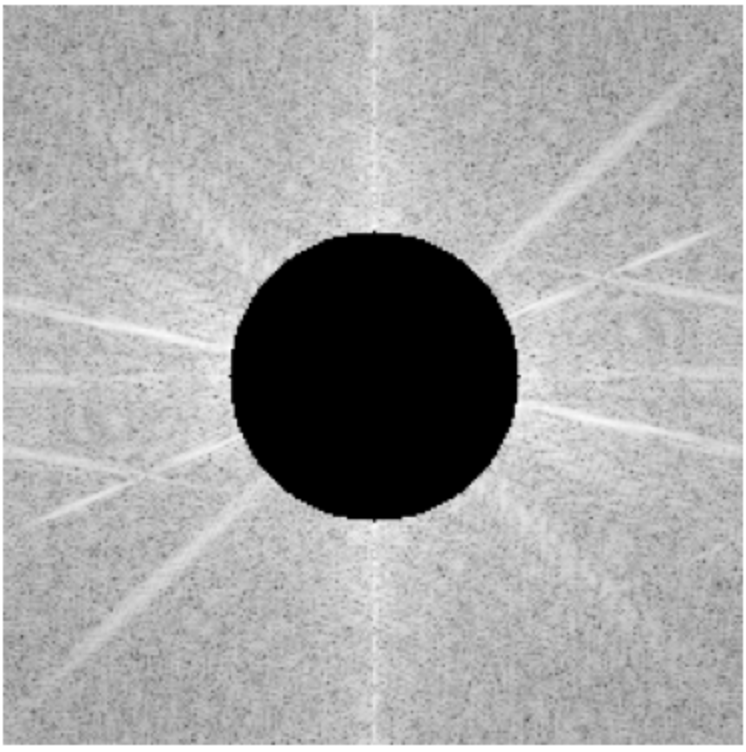


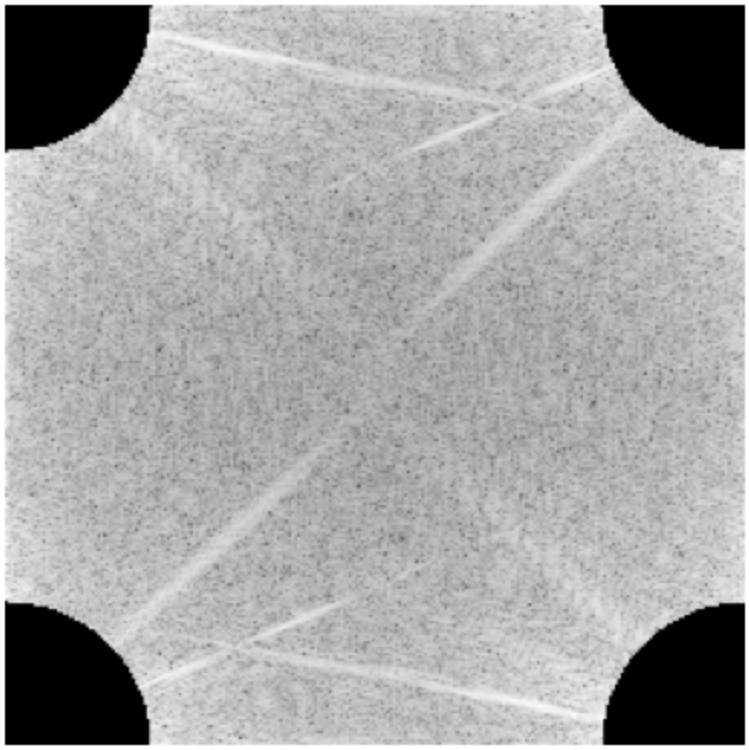








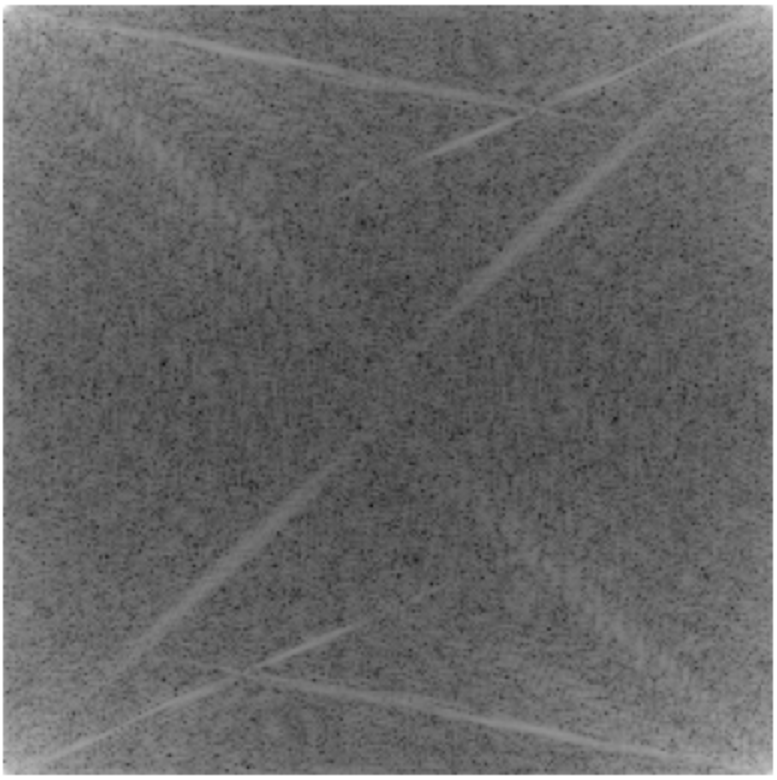


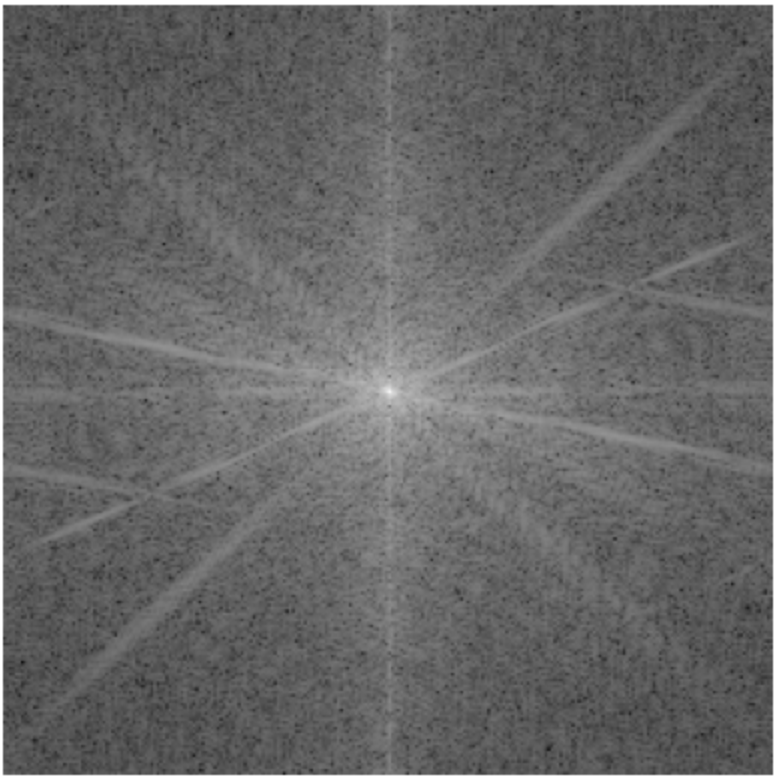


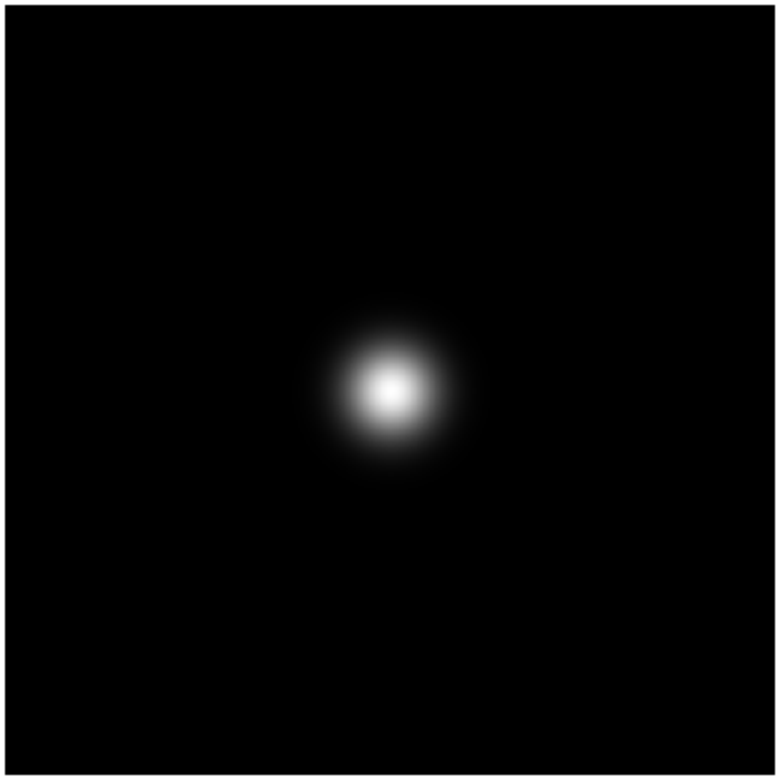
In [ ]:
```
# libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt

# open the image f
f = cv2.imread('cameraman.tif',0)

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

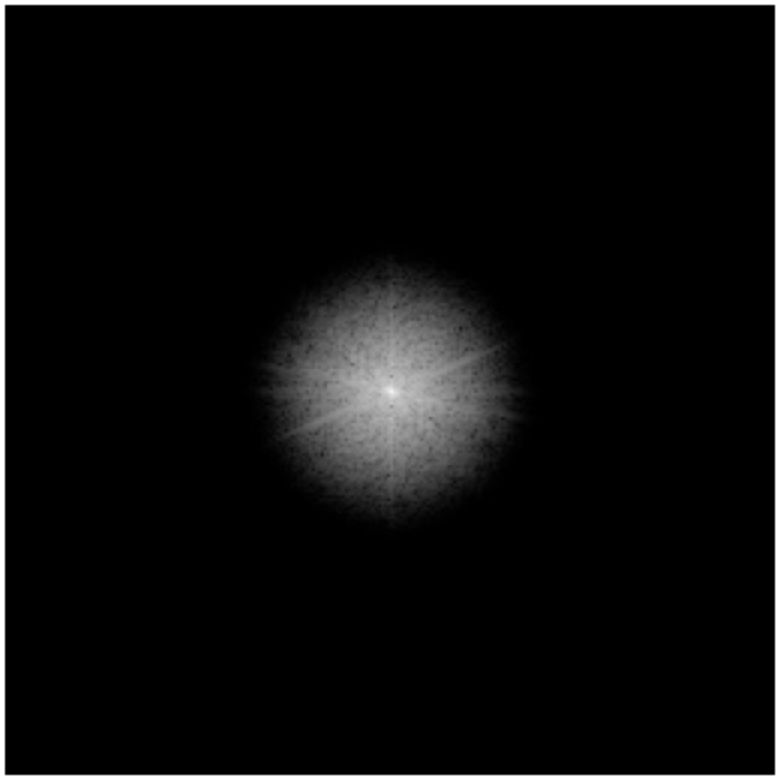
```
plt.imshow(f, cmap='gray')
plt.axis('off')
plt.show()

# 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()

# Gaussian: High pass filter
HPF = 1 - H

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

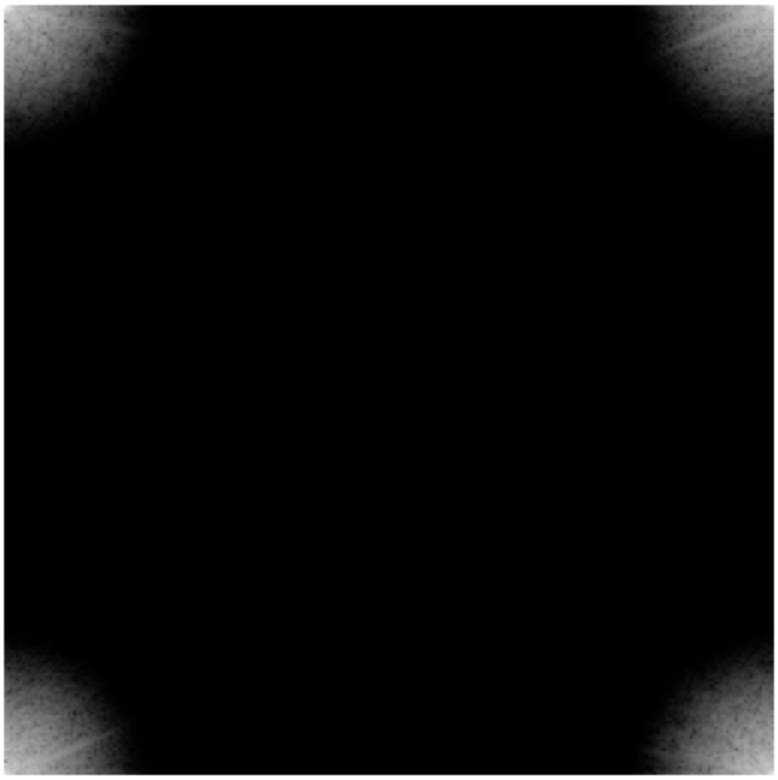
```
plt.imshow(HPF, cmap='gray')
plt.axis('off')
plt.show()

# 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()
```

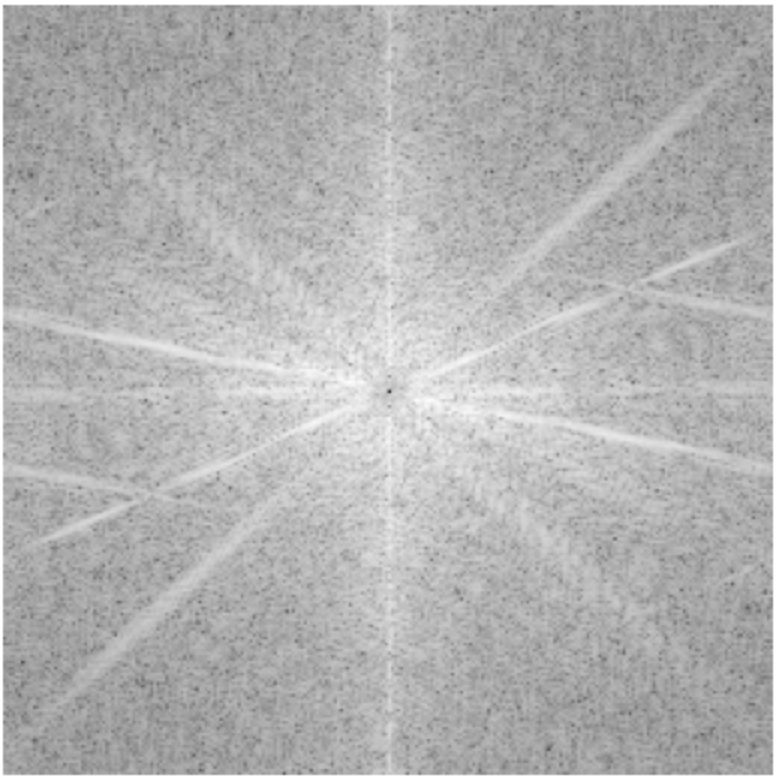


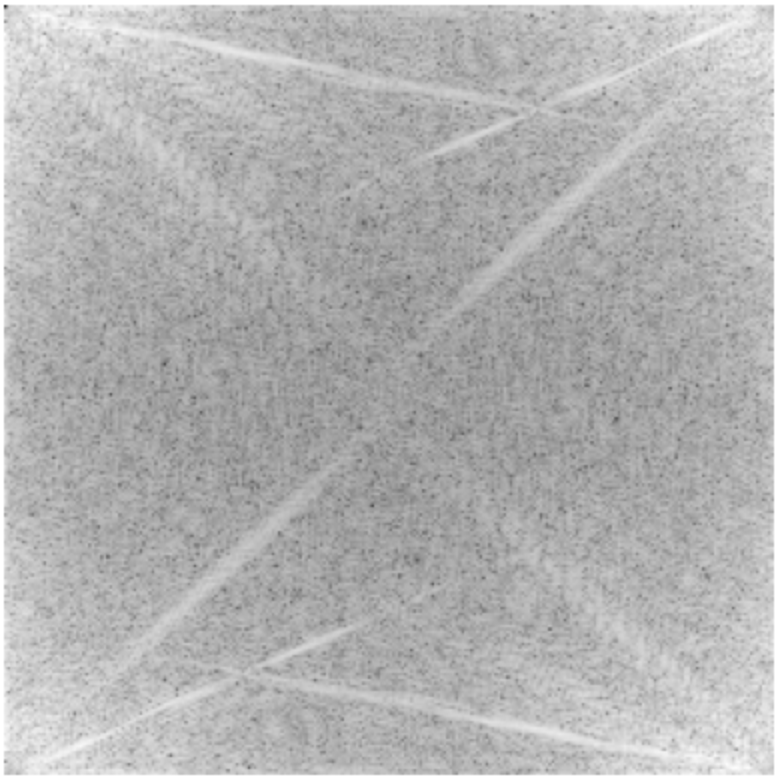






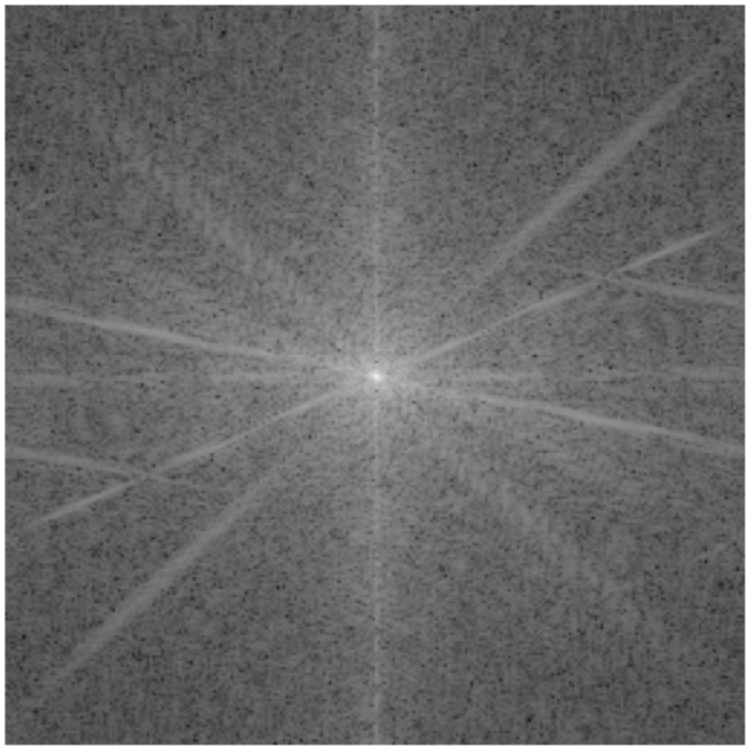


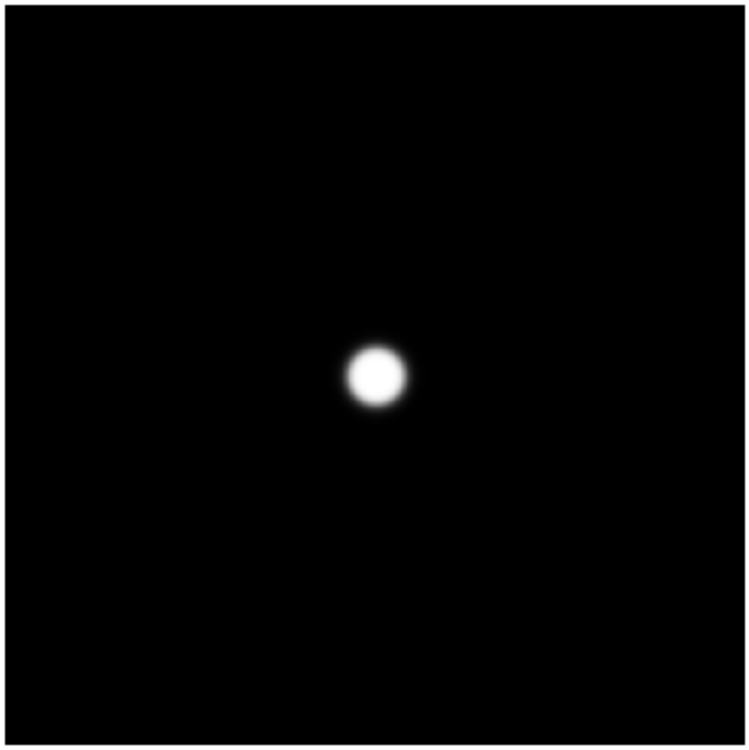




In [ ]:
```
# Libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt

# open the image
f = cv2.imread('cameraman.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()

# Butterwort 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()

# Butterworth High Pass Filter
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()
```





```
<ipython-input-81-636b48b4b227>:47: RuntimeWarning: divide by zero encountered in sc
alar divide
  HPF[u,v] = 1 / (1 + (D0/D)**n)
```

In [ ]: