In [ ]:

```python
import numpy as np
import skimage.data as data
import matplotlib.pyplot as plt
```

9.1)

5x5 lik bir matrisi

```
h = [0,1,0 :
     1,1,1 :
     0,1,0]
```

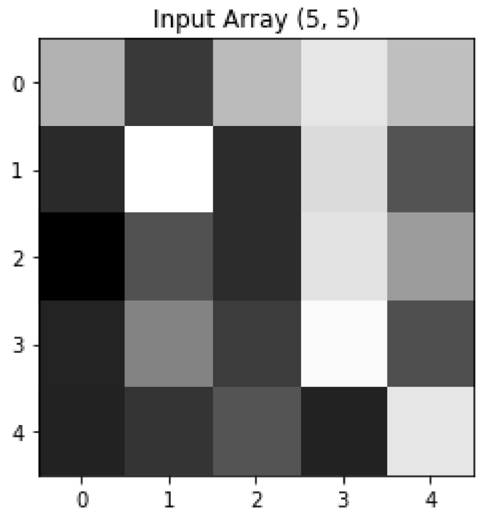
matrisi ile 2 boyutlu konvolüsyonunu:

a) spatial domain'de hesaplayın

In [ ]:

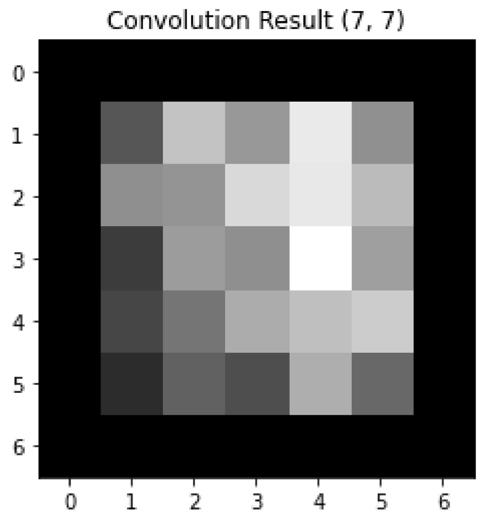
```python
def conv2d(img, kernel):
    """
    2D convolution
    """
    # Padding
    pad = (kernel.shape[0] - 1) // 2
    img = np.pad(img, pad_width=pad, mode='constant', constant_values=0)

    # Output image
    img_conv = np.zeros_like(img)

    # Loop over every pixel of the image
    for i in range(pad, img.shape[0] - pad):
        for j in range(pad, img.shape[1] - pad):
            # Convolution operation
            img_conv[i, j] = np.sum(img[i - pad:i + pad + 1, j - pad:j + pad + 1] * kernel)

    return img_conv
```

In [ ]:
```python
input_array = np.random.randint(0, 255, size=(5, 5))
kernel = np.array([
                   [0, 1, 0],
                   [1, 1, 1],
                   [0, 1, 0]
                ])

plt.imshow(input_array, cmap='gray')
plt.title(f'Input Array {input_array.shape}')
plt.show()
```



In [ ]:
```python
conv_result = conv2d(input_array, kernel)
plt.imshow(conv_result, cmap='gray')
plt.title(f'Convolution Result {conv_result.shape}')
```

Out[ ]:
```
Text(0.5, 1.0, 'Convolution Result (7, 7)')
```



b) frekans bölgesinde hesaplayın.

fft ve ifft dışında konvolüsyonu kendiniz yazın. sonuçları karşılaştırın. Sıfır eklemeyi unutmayın.

In [ ]:

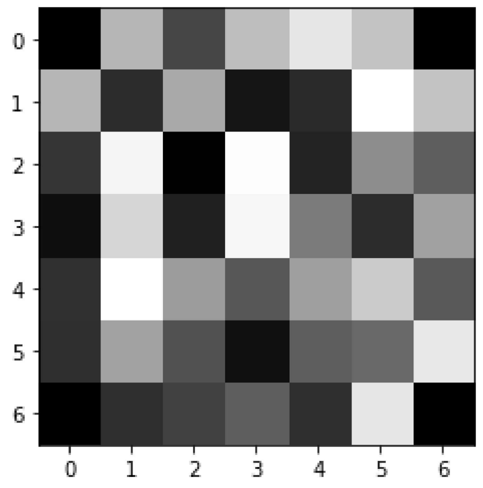
```
def fourier_conv2d(img, kernel):

    size = np.array(input_array.shape) + np.array(kernel.shape) - 1

    fsize = 2 ** np.ceil(np.log2(size)).astype(int)
    fslice = tuple([slice(0, int(sz)) for sz in size])

    input_array_f = np.fft.fft2(input_array , fsize)
    kernel_f = np.fft.fft2(kernel , fsize)
    result = np.fft.ifft2(input_array_f*kernel_f)[fslice].copy()
    return np.array(result.real, dtype=np.uint8)
```

In [ ]:

```
result = fourier_conv2d(input_array, kernel)
plt.imshow(result, cmap='gray')
```

Out[ ]:

```
<matplotlib.image.AxesImage at 0x7f9ba6fc4610>
```



9.2)

In [ ]:

```
def circular_kernel(img, d0, shifted=True, low_pass=True):
    """
    low pass filter
    """

    # Calculate the size of the image
    size = np.array(img.shape)
    # Calculate the size of the filter
    fsize = 2 ** np.ceil(np.log2(size)).astype(int)
    print("fsize: ", fsize)

    # Calculate the center of the filter
    center = (fsize - size) // 2 if not shifted else fsize // 2

    # Calculate the distance of the pixel from the center
    dist = lambda i,j: np.sqrt((i - center[0]) ** 2 + (j - center[1]) ** 2)

    if low_pass:
      filter_val = lambda i,j: 1 if dist(i,j) <= d0 else 0
    else:
      filter_val = lambda i,j: 1 if dist(i,j) >= d0 else 0

    # Create the filter
```

```python
    filter = np.zeros(fsize, dtype=complex)
    for i in range(fsize[0]):
        for j in range(fsize[1]):
            # Calculate the filter value

            filter[i, j] = filter_val(i, j)

    ## DEBUG
    plt.imshow(filter.real.astype(np.uint8), cmap='gray')
    plt.show()

    return filter
```

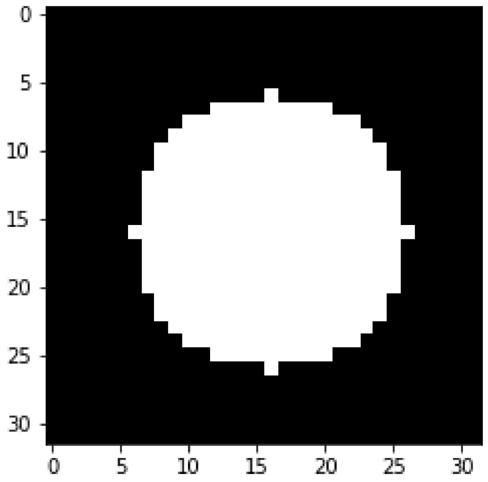
In [ ]:

```python
def apply_filter(img, filter, shift=False):
    fsize = np.array(filter.shape, dtype=int)
    # Fourier transform the image
    img_f = np.fft.fft2(img, fsize)

    if shift:
        # shift the zero frequency component to the center of the filter
        img_f = np.fft.fftshift(img_f)

    # Apply the filter
    result = img_f * filter

    if shift:
        # Shift the zero frequency component back to the top left corner
        result = np.fft.ifftshift(result)

    # Inverse Fourier transform the image
    result = np.fft.ifft2(result).real

    return np.array(result, dtype=np.uint8)
```

In [ ]:

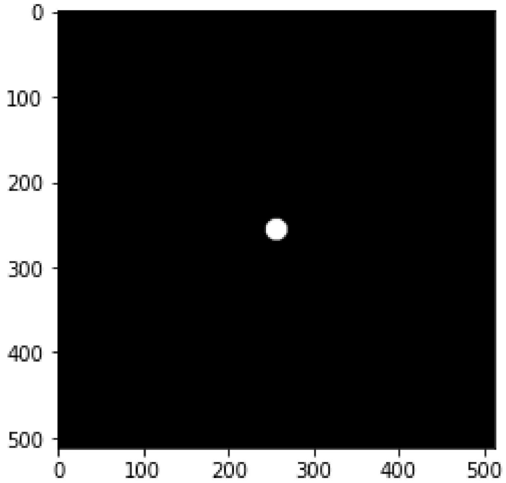
```python
img_org = data.camera()

low_pass_kernel = circular_kernel(np.random.randint(0,255, size=(20,20)), d0=10)
```

```
fsize:  [32 32]
```

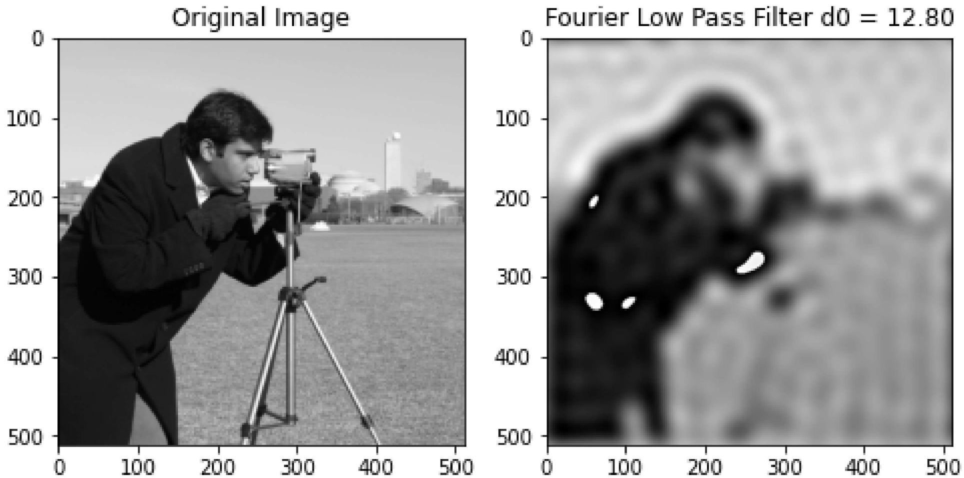


```
In [ ]: for d_coeff in np.arange(0.05, 1.0, 0.15):

            d0 = img_org.shape[0] /2 * d_coeff
            low_pass_kernel = circular_kernel(img_org, d0=d0)

            img_filtered = apply_filter(img_org, low_pass_kernel, shift=True)

            fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4))
            fig.suptitle(f'shape: {img_org.shape}')
            ax1.imshow(img_org, cmap='gray')
            ax1.set_title('Original Image')
            ax2.imshow(img_filtered, cmap='gray')
            ax2.set_title(f'Fourier Low Pass Filter d0 = {d0:.2f}')
            plt.show()
```

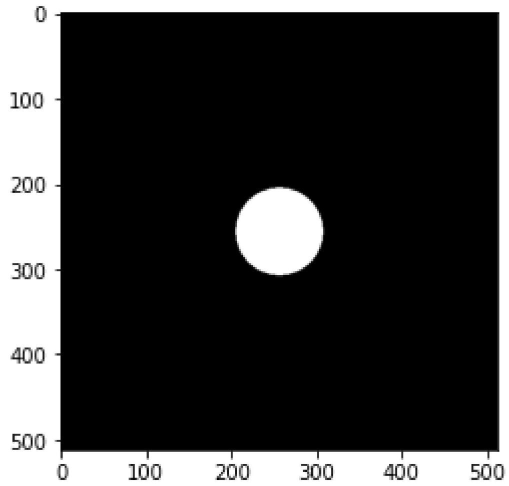
```
fsize:  [512 512]
```



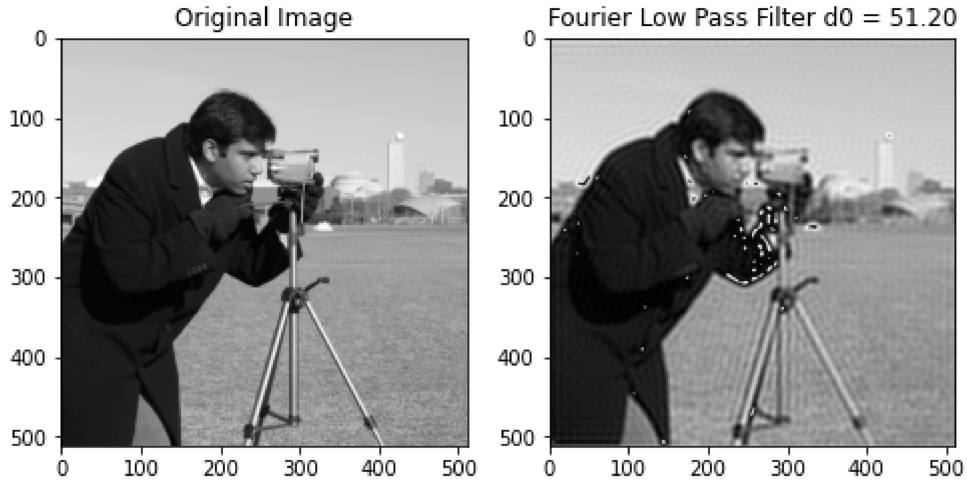
shape: (512, 512)



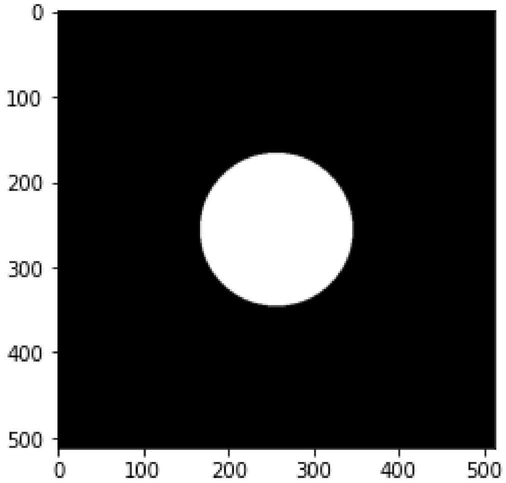

```
fsize:  [512 512]
```



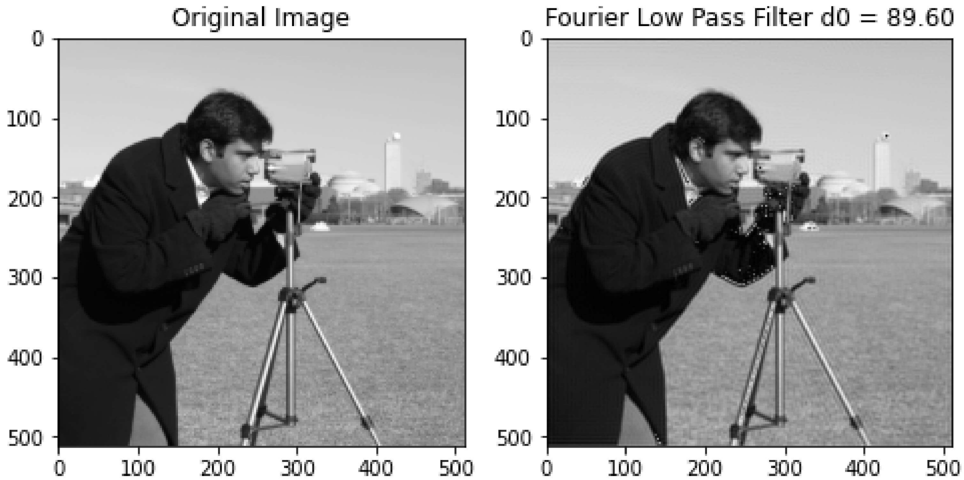
shape: (512, 512)



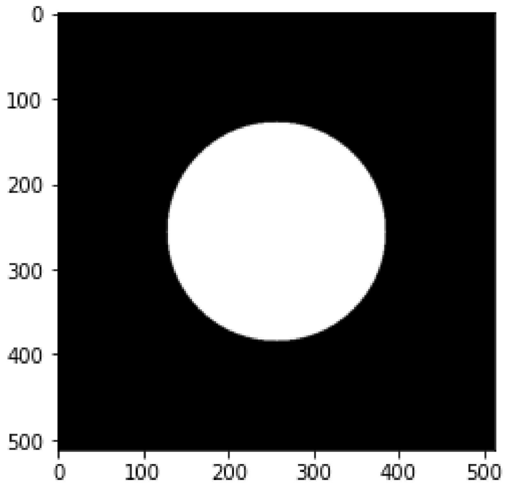

```
fsize:  [512 512]
```



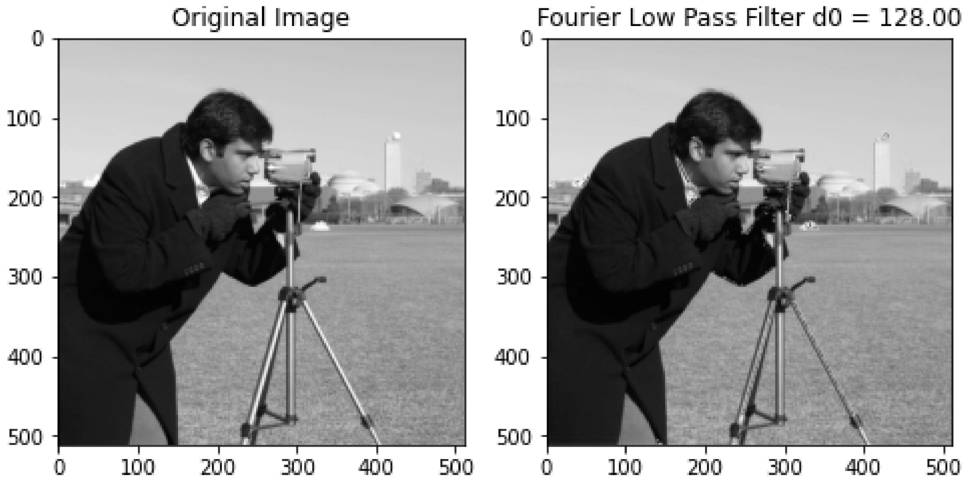
shape: (512, 512)



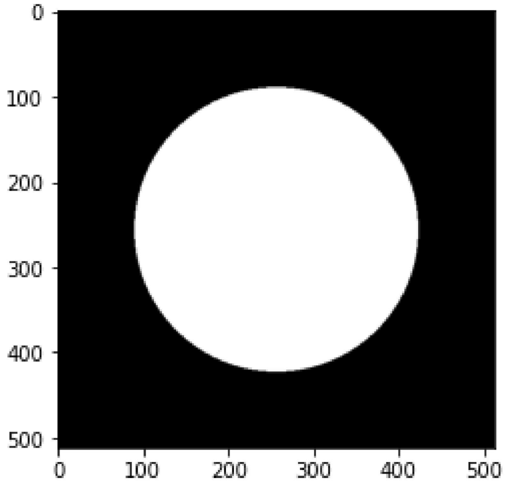

```
fsize:  [512 512]
```



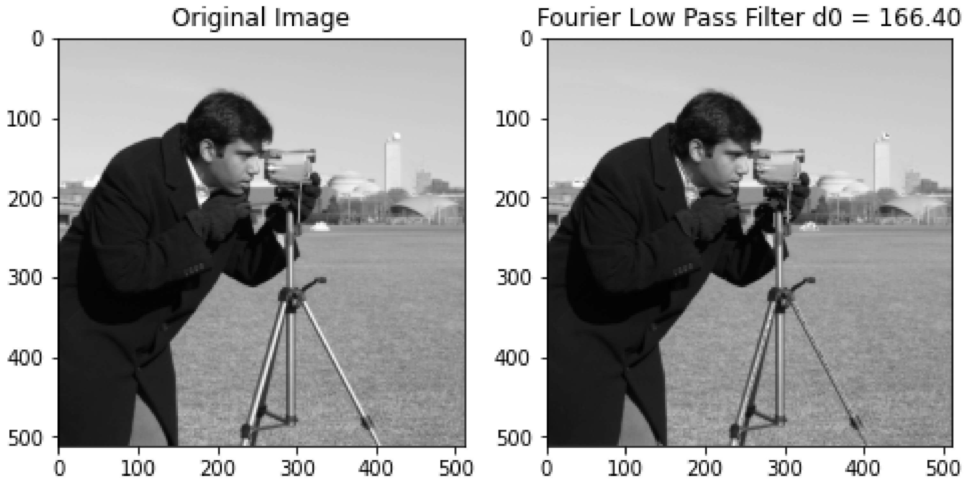
shape: (512, 512)



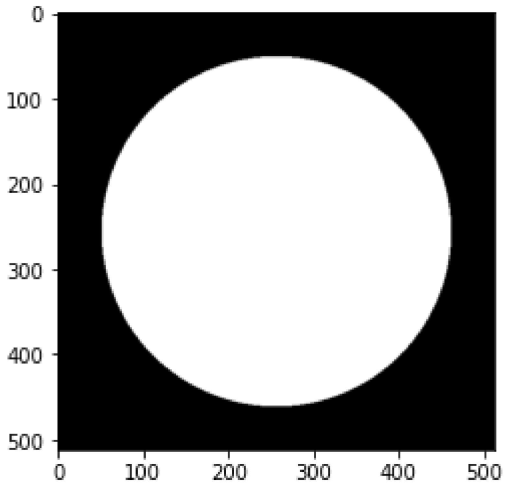

```
fsize:  [512 512]
```



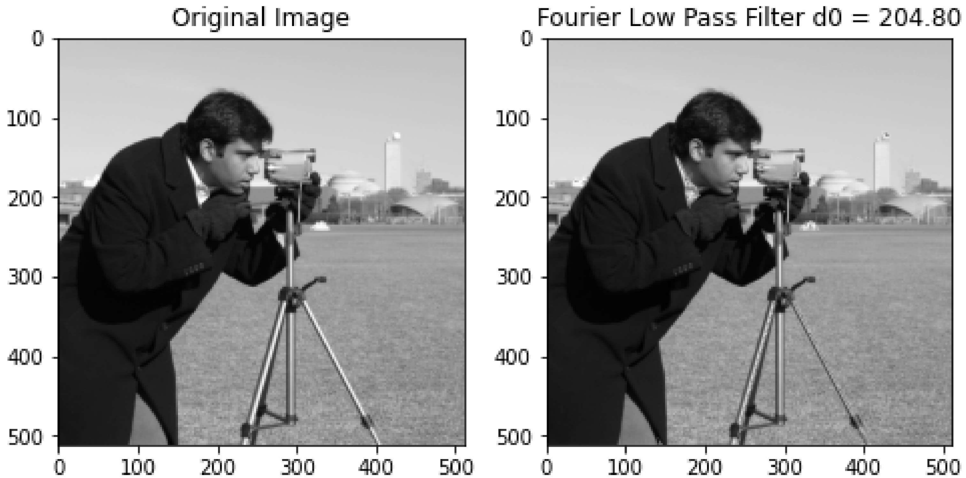
shape: (512, 512)



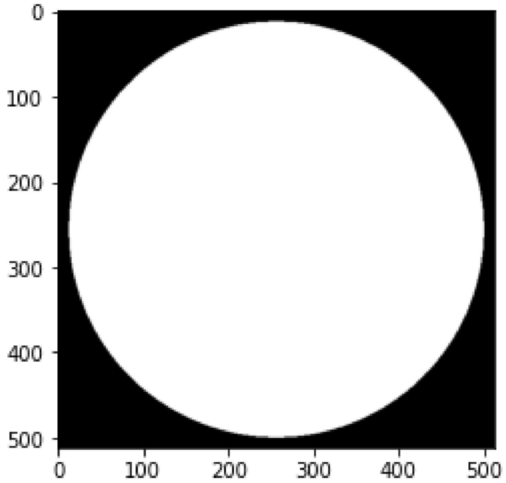

```
fsize:  [512 512]
```



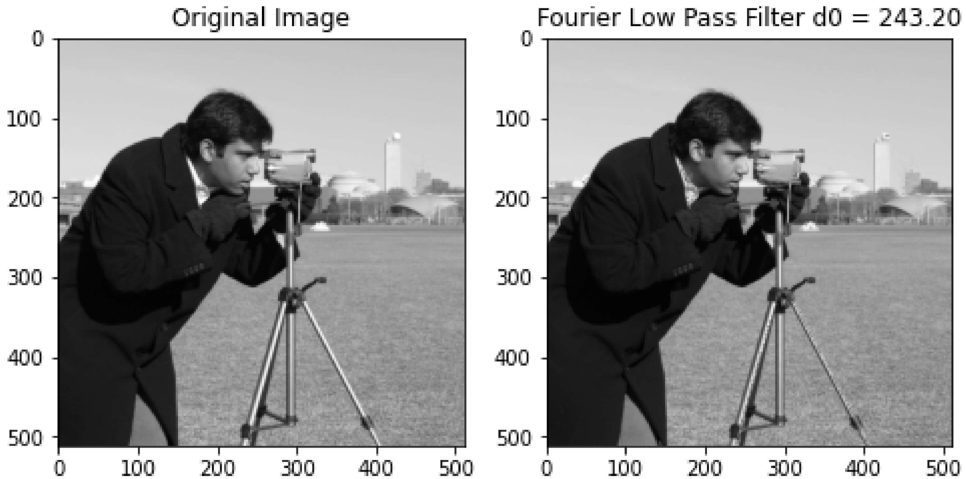
shape: (512, 512)




```
fsize:  [512 512]
```



shape: (512, 512)



In [ ]: