Lab 3

1. Abstract

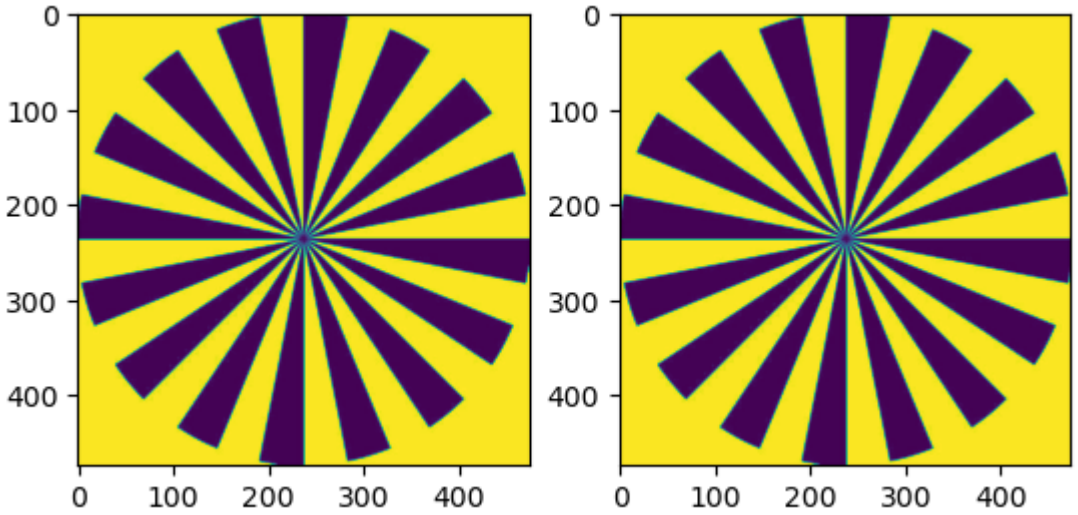
This report explores the application of Fourier optics in simulating various optical systems, with a focus on the analysis and manipulation of optical signals through filters. Techniques such as low-pass, high-pass, and phase contrast filtering are applied to enhance or suppress certain image features. The Siemens star pattern is used as a test image for analyzing system performance. Additionally, optical transfer functions (OTFs) and modulation transfer functions (MTFs) are examined to evaluate image quality through different apertures, including circular, rectangular, and vortex phase filters. The Wiener deconvolution method is used to attempt image recovery in the presence of noise, with its effectiveness demonstrated through an example.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from skimage import io, color
```

2. Filters

In [2]:
```
im = io.imread("Siemens.jpeg")
im = color.rgb2gray(im)

plt.figure()
plt.subplot(1,2,1)
plt.imshow(im)
plt.subplot(1,2,2)
plt.imshow(np.angle(np.exp(1j*im)))
```

Out[2]:
```
<matplotlib.image.AxesImage at 0x265c2587690>
```



A Siemen star is a common pattern used in imaging and optical testing, particularly to measure the resolving power of cameras, lenses, and optical systems. It consists of a circular arrangement of alternating black and white radial lines that converge toward the center.

In [3]:
```python
# Fourier Transform
def ft2(x):
    return np.fft.fftshift(np.fft.fft2(np.fft.ifftshift(x)))

# Inverse Fourier Transform
def ift2(x):
    return np.fft.fftshift(np.fft.ifft2(np.fft.ifftshift(x)))
```

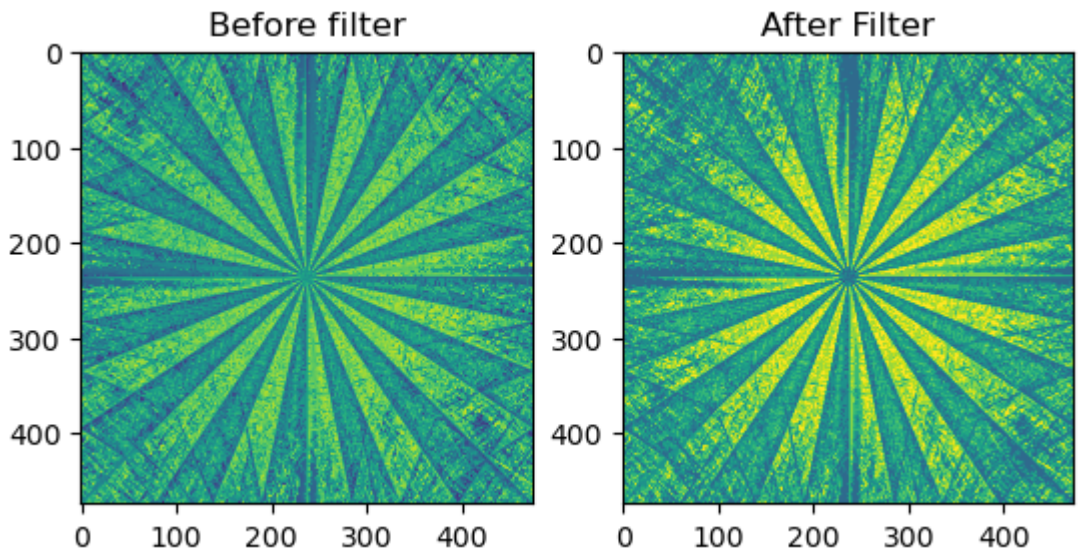
In [4]:
```python
# Constants
N = im.shape[0]
p = 5e-6
x0 = np.linspace(-N/2,  N/2, N, endpoint = True)
x, y = np.meshgrid(x0, x0)
x = x*p
y = y*p

f0 = x0/N
fx, fy = np.meshgrid(f0, f0)
fx = fx/p
```

```
fy = fy/p

f = 5e-2
wavelength = 0.5e-6
k = 2*np.pi/wavelength
```

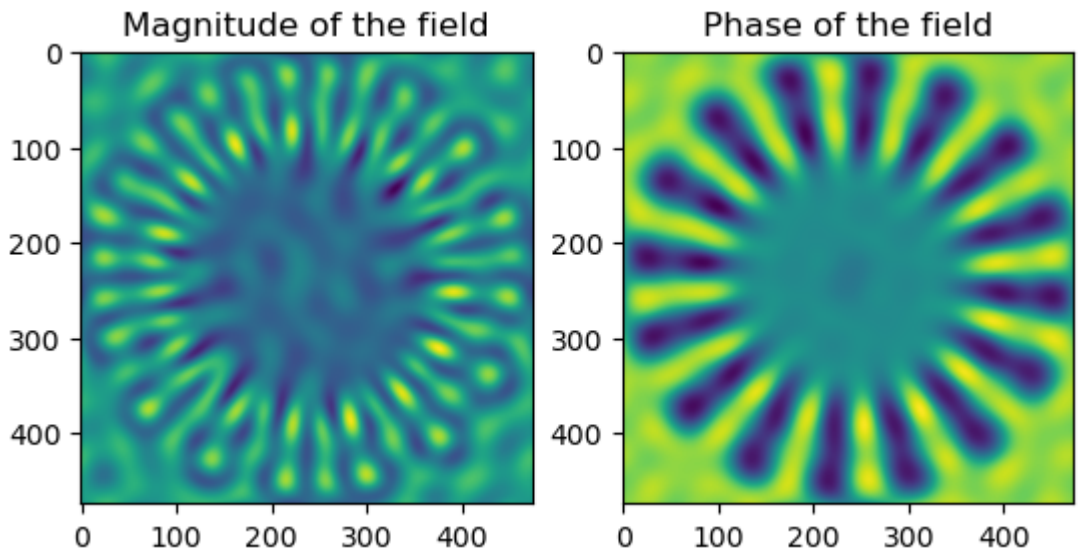
2.1. Low Pass Filter

The code simulates the propagation of a phase object through a 4f system with a low pass filter at the fourier plane.

In [5]:
```
#First lens
radius1 = N*p/2 # Defines the physical radius of the lens
circ1 = np.zeros((N, N))
circ1[x**2 + y**2 <= radius1**2] = 1
phase1 = k*(x**2 + y**2)/(2*f) # Quadratic phase of the lens
lens1 = np.exp(-1j * phase1) # Complex field of the lens
#plt.imshow(np.angle(lens1))

u1 = np.exp(1j*im)# Object at the input plane
U1 = ft2(u1) # Object at the focus of lens 1

# Low pass Filter
lp_radius = 10/(N*p)
lp_filter = np.zeros((N, N))
lp_filter[fx**2 + fy**2 <= lp_radius**2] = 1
#plt.imshow(lp_filter)

# Applying the low pass filter to U1
U2 = U1*lp_filter

plt.figure()
plt.subplot(1,2,1)
plt.imshow(np.angle(U1))
plt.title("Before filter")
plt.subplot(1,2,2)
plt.imshow(np.angle(U2))
plt.title("After Filter")
plt.show()
```



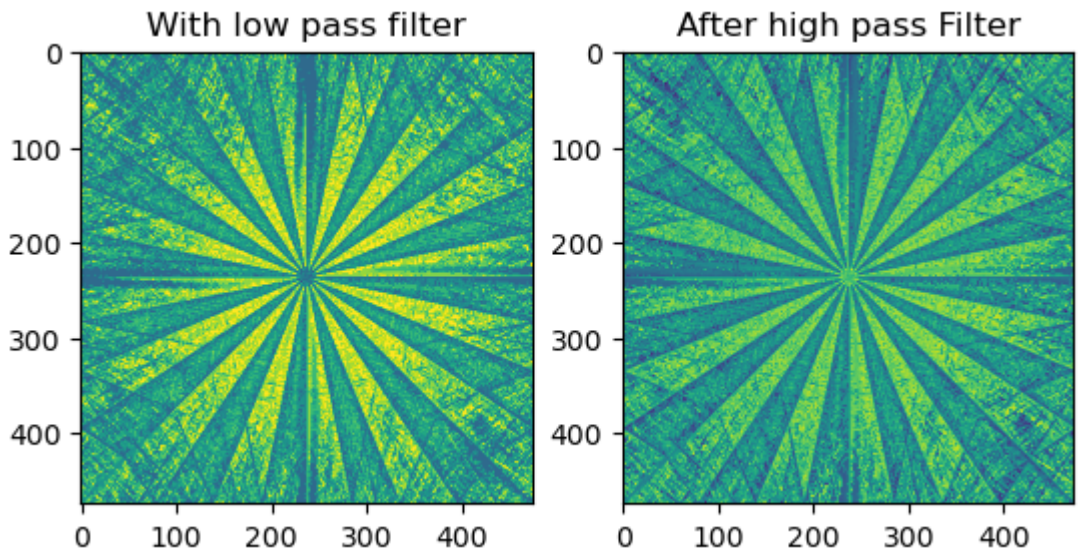
In [6]:
```
# Field after second lens
u2 = ft2(U2)

plt.figure()
plt.subplot(1,2,1)
plt.imshow(np.abs(u2))
plt.title("Magnitude of the field")
plt.subplot(1,2,2)
plt.imshow(np.angle(u2))
plt.title("Phase of the field")
plt.show()
```



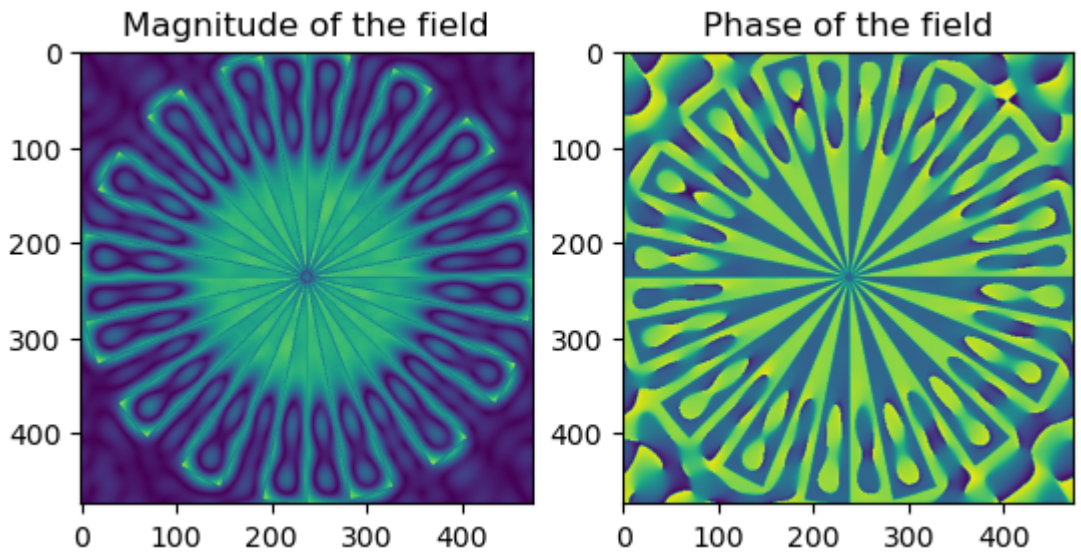
The low pass filter does not allow high frequencies to pass through leading to a blurred image. The high frequencies carry the information of edges and absence of these lead to lower contrast and overall blur.

2.2. High Pass Filter

In [7]:
```
hp_filter = 1 - lp_filter
U2_high = U1*hp_filter

plt.figure()
plt.subplot(1,2,1)
plt.imshow(np.angle(U2))
plt.title("With low pass filter")
plt.subplot(1,2,2)
plt.imshow(np.angle(U2_high))
plt.title("After high pass Filter")
plt.show()
```



In [8]:
```
u2_high = ft2(U2_high)

plt.figure()
plt.subplot(1,2,1)
plt.imshow(np.abs(u2_high))
plt.title("Magnitude of the field")
plt.subplot(1,2,2)
plt.imshow(np.angle(u2_high))
plt.title("Phase of the field")
plt.show()
```



The high pass filter passes high frequencies. This enhances fine scale changes and sharp edges.

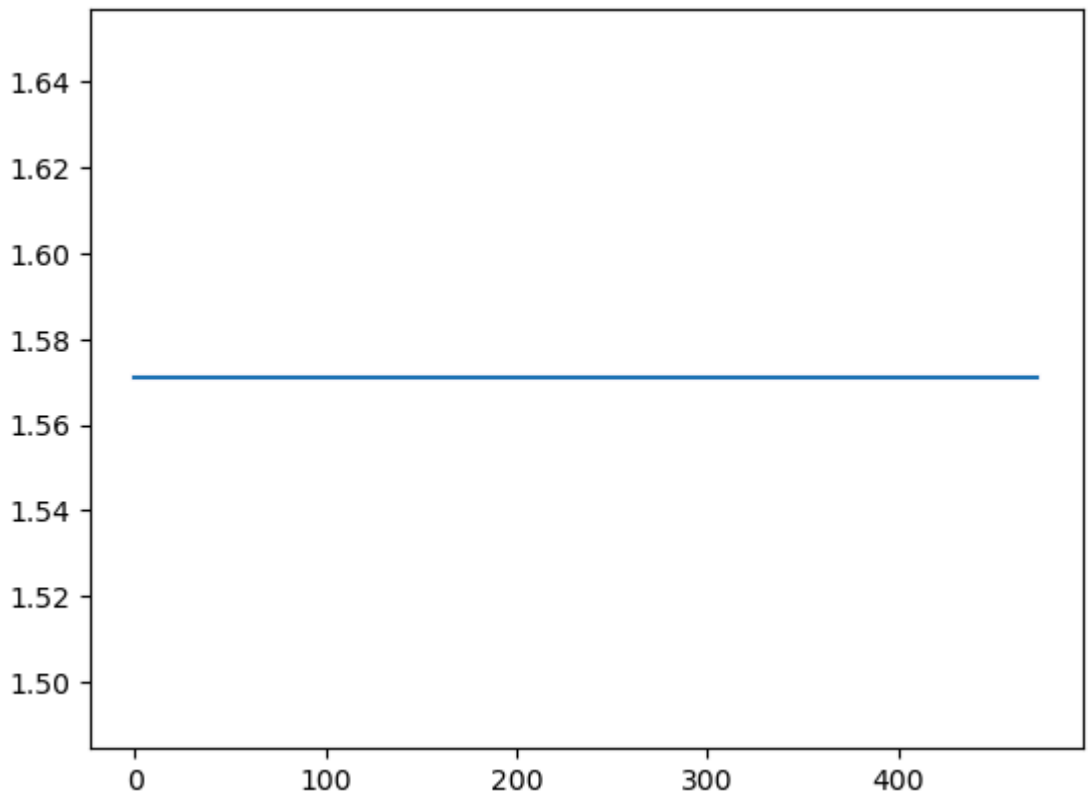
2.3. Phase Contrast Filter

A phase contrast filter is an optical device used in microscopy to enhance the contrast of transparent and nearly colorless specimens, such as living cells and tissues. The code simulates a phase contrast effect by manipulating the phase of low-frequency components while preserving the amplitude of higher frequencies.

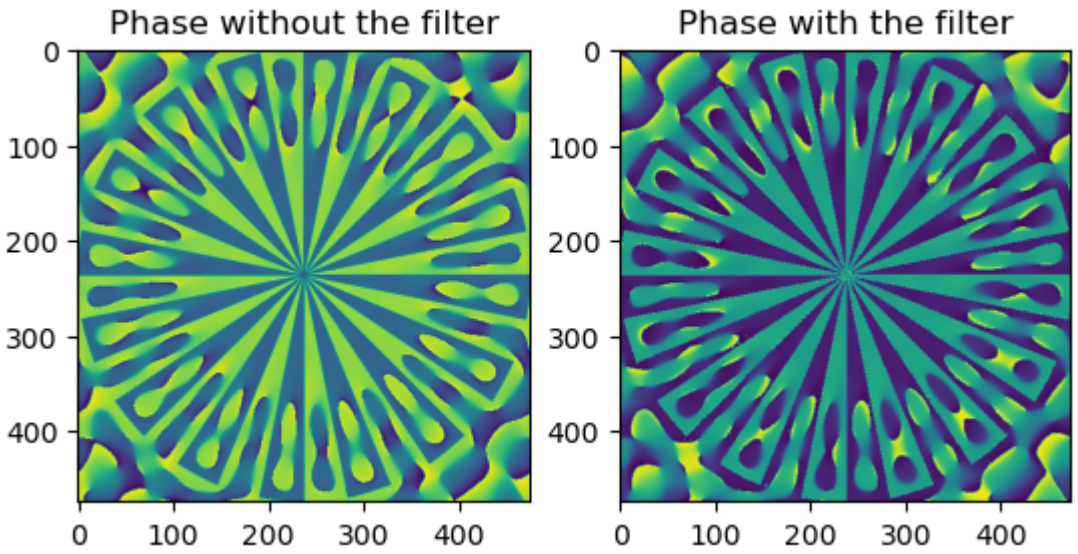
In [9]:
```
phase_filter = lp_filter
phase_filter[fx**2 + fy**2 <= (lp_radius)**2] = np.exp(1j*np.pi/2)  # Adding pi/2 phase to region between lp_radius
phase_filter[fx**2 + fy**2 > (lp_radius)**2] = 1 # No phase change outside the lp_radius
phase_contrast = phase_filter*np.exp(1j*np.pi/2) # pi/2 phase change only with the low pass filter

profile = np.angle(phase_contrast)[N//2,:]
plt.plot(profile) # Display the phase values for the filter
```

```
C:\Users\nitin negi\AppData\Local\Temp\ipykernel_6660\3140429178.py:2: ComplexWarning: Casting complex values to real discards
the imaginary part
  phase_filter[fx**2 + fy**2 <= (lp_radius)**2] = np.exp(1j*np.pi/2)  # Adding pi/2 phase to region between lp_radius
```

Out[9]: [<matplotlib.lines.Line2D at 0x265c2a1f550>]



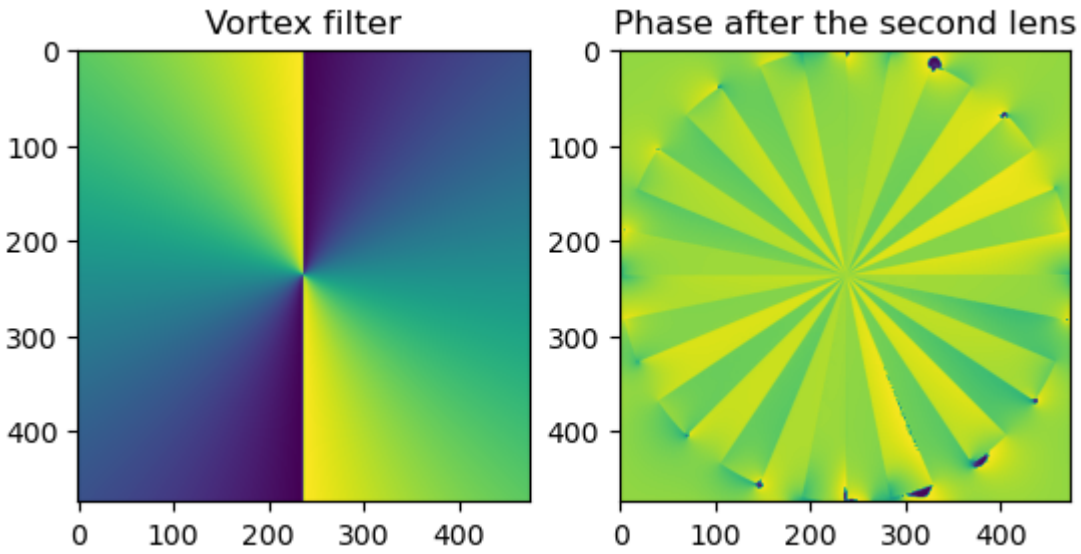
In [10]:
```
# Apply phase contrast filter
U2_phase = U1*phase_contrast
# After second lens
u2_phase = ft2(U2_phase)

plt.figure()
plt.subplot(1,2,1)
plt.imshow(np.angle(u2_high))
plt.title("Phase without the filter")
plt.subplot(1,2,2)
plt.imshow(np.angle(u2_phase))
plt.title("Phase with the filter")
plt.show()
```



2.4. Vortex phase

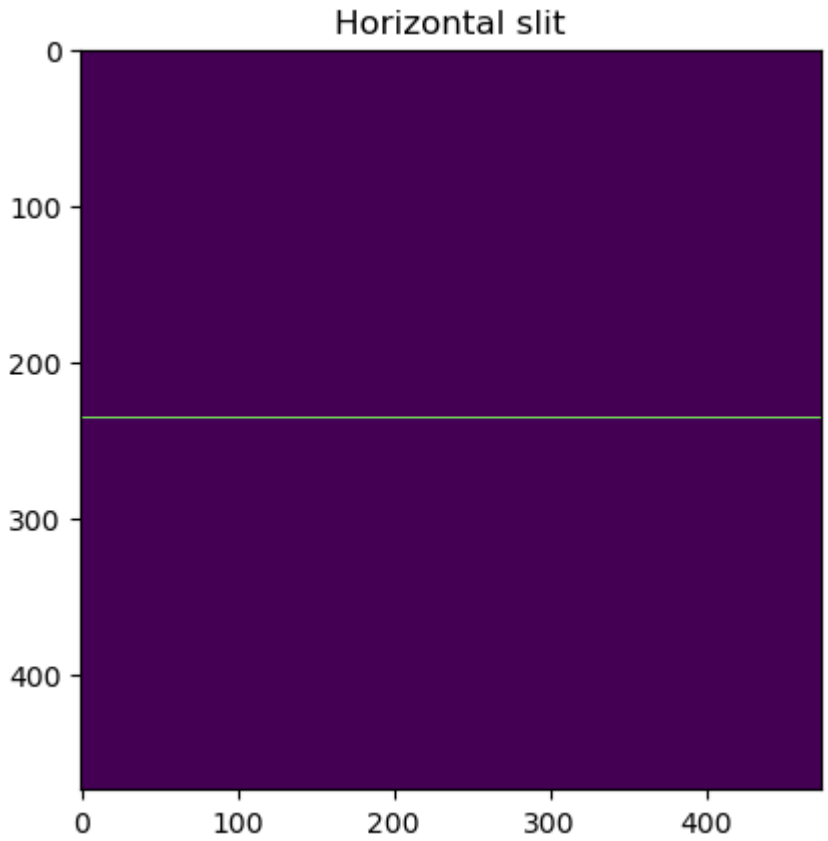
A vortex phase filter introduces a spiral phase. This results in a beam with a donut-shaped intensity profile and a phase singularity at its center. Vortex beams carry orbital angular momentum (OAM). The vortex filter is typically represented by a phase shift proportional to the angular coordinate around the center, imparting a helical structure to the beam.

In [11]:
```
phi = np.arctan2(y,x) # Angular coordinate
vortex = np.exp(1j*2*phi) # Vortex phase with l = 1

U2_vortex = U1*vortex
u2_vortex = ft2(U2_vortex) # After the second lens

plt.figure()
plt.subplot(1,2,1)
plt.imshow(np.angle(vortex))
plt.title("Vortex filter")
plt.subplot(1,2,2)
plt.imshow(np.angle(u2_vortex))
plt.title("Phase after the second lens")
plt.show()
```

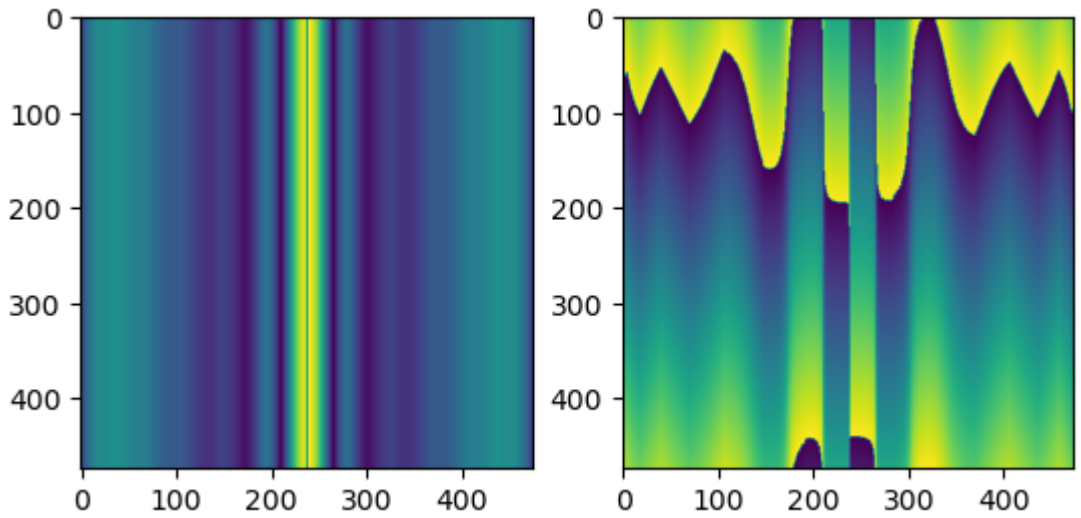


2.5. Horizontal Slit

In [12]:
```
# Horizontal Slit
h_slit = np.zeros((N, N))
slit_center = N//2
slit_half_width = p/2

upper = int(slit_center - slit_half_width)
lower = int(slit_center + slit_half_width)

h_slit[upper : lower] =1

plt.figure()
plt.imshow(h_slit)
plt.title("Horizontal slit")
plt.show()
```



In [13]:
```
# Apply the horizontal slit
U2_horizontal = U1*h_slit
u2_horizontal = ft2(U2_horizontal)

plt.figure()
plt.subplot(1,2,1)
plt.imshow(np.abs(u2_horizontal))
plt.subplot(1,2,2)
plt.imshow(np.angle(u2_horizontal))
plt.show()
```



The horizontal slit example is related to the projection slice theorem. **Projection slice theorem -** In mathematics, the projection-slice theorem in two dimensions states that the results of the following two calculations are equal:

- Take a two-dimensional function f(r), project (e.g. using the Radon transform) it onto a (one-dimensional) line, and do a Fourier transform of that projection.
- Take that same function, but do a two-dimensional Fourier transform first, and then slice it through its origin, which is parallel to the projection line.

This theorem is used, for example, in the analysis of medical CT scans where a "projection" is an x-ray image of an internal organ. The Fourier transforms of these images are seen to be slices through the Fourier transform of the 3-dimensional density of the internal organ, and these slices can be interpolated to build up a complete Fourier transform of that density. The inverse Fourier transform is then used to recover the 3-dimensional density of the object.

If f(x,y) is a 2D function, then the projection of f(x,y) onto the x axis is p(x) where

$$p(x) = \int f(x,y)dy$$

The Fourier transform of f(x,y) is

$$F(k_x, k_y) = \int\int f(x,y) e^{-2\pi i(xk_x + yk_y)} dxdy$$

The slice is then $s(k_x)$

$$\$\$s(k_x) = F(k_x, 0) = \int\int f(x,y) e^{-2\pi i x k_x} dxdy \quad (1)$$

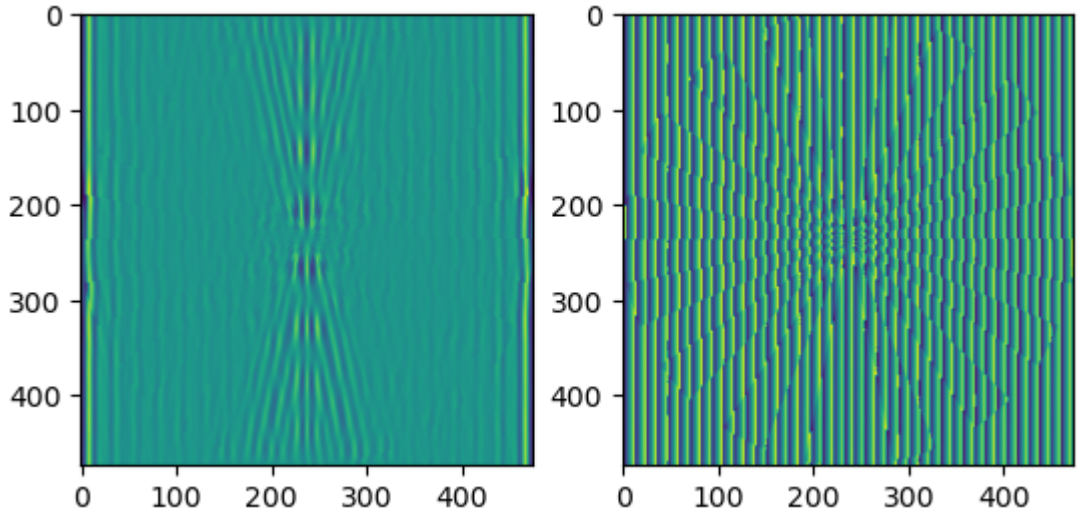
$$= \int [\int f(x,y)dy] e^{-2\pi i x k_x} dx \quad (2)$$

$$= \int p(x) e^{-2\pi i x k_x} dx\$\$ \quad (3)$$

which is just the Fourier transform of p(x).

2.6. Tilted Illumination with low pass filter

In [14]:
```
angle = np.deg2rad(0.5)
u1_tilt = np.exp(1j*im) * np.exp(-1j*k*(x*np.sin(angle)))# Object at the input plane

U1_tilt = ft2(u1_tilt) # Object at the focus of lens 1

#plt.imshow(np.angle(U1))


# Applying the low pass filter to U1
U2_tilt = U1_tilt*lp_filter
u2_tilt = ft2(U2_tilt)

plt.figure()
plt.subplot(1,2,1)
plt.imshow(np.abs(u2_tilt))

plt.subplot(1,2,2)
plt.imshow(np.angle(u2_tilt))
plt.show()
```



3. Point Spread Function

Geometrical optics in the paraxial approximation predicts that the image of a point produced by an image-forming optical system is a point. But the image of a point produced by a real optical system is not a point but a small, blurred spot. This is due to diffraction and various forms of aberrations. The blurred-spot image of a single point is called the point-spread function or the impulse response of the optical system.

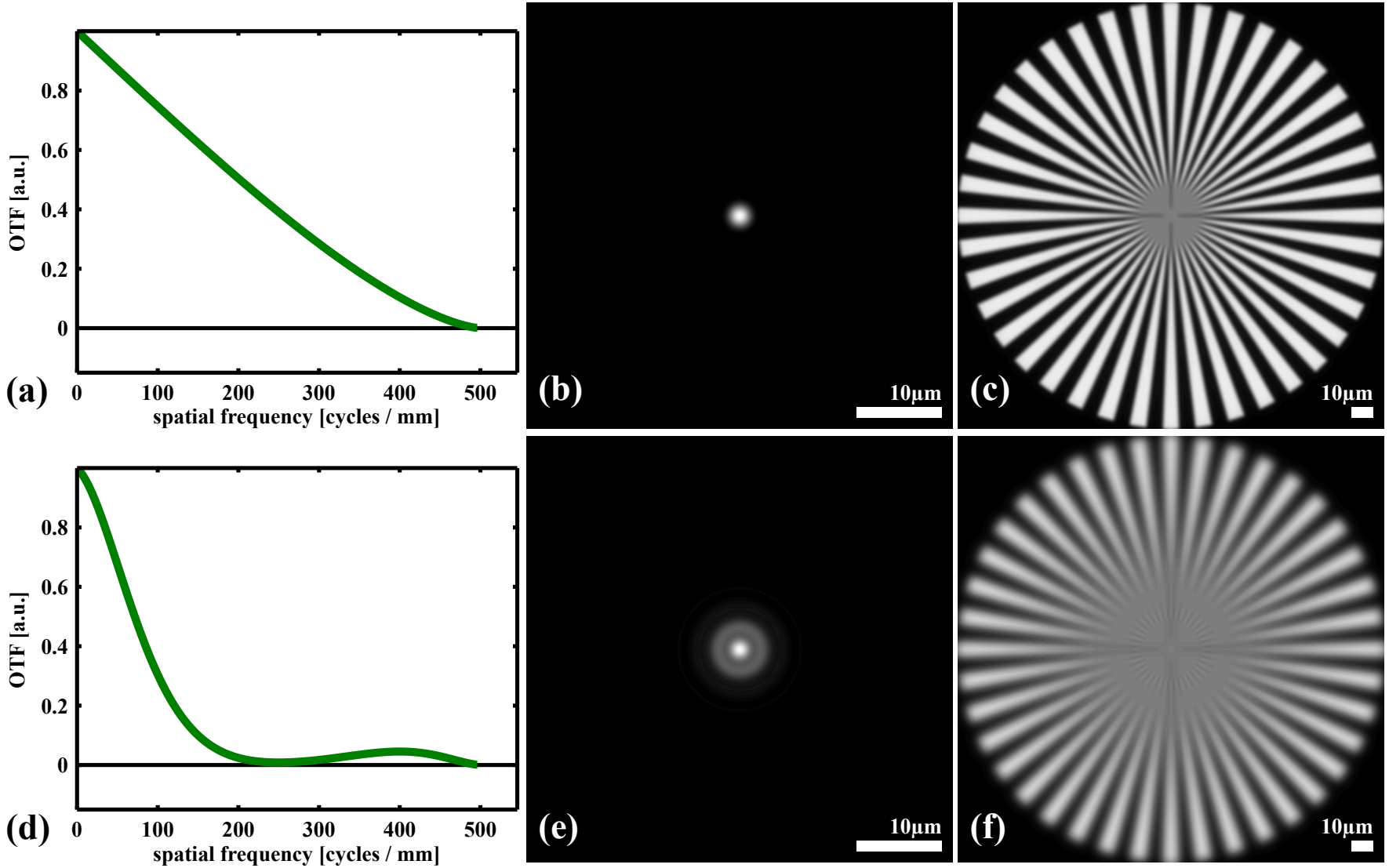
Most imaging systems can be modeled as linear shift-invariant systems. This means that the image I(x, y) formed by an object O(x, y) is the convolution of the object with the system's PSF.

$$I(x,y) = O(x,y) * |h(x,y)|^2$$

4.OTF

The OTF is defined as the Fourier transform of the point spread function. As a Fourier transform, the OTF is complex-valued; but it will be real-valued in the common case of a PSF that is symmetric about its center. The MTF is formally defined as the magnitude (absolute value) of the

complex OTF.

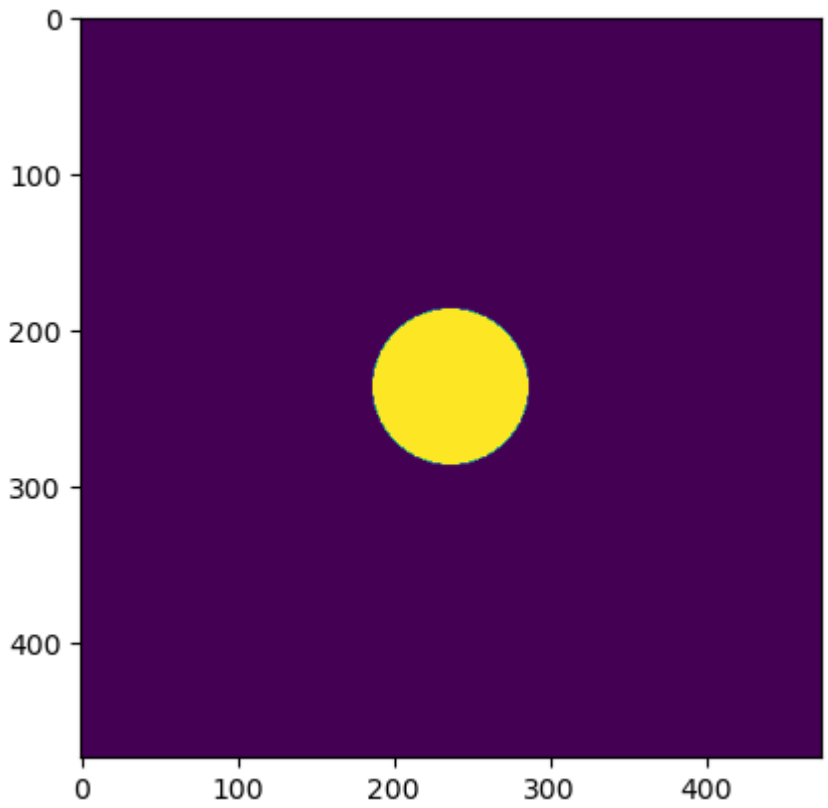


In the image the top row shows the OTF of a well focused system and corresponding image quality, while the bottom row shows an out-of-focus imaging system.

4.1. Circular aperture

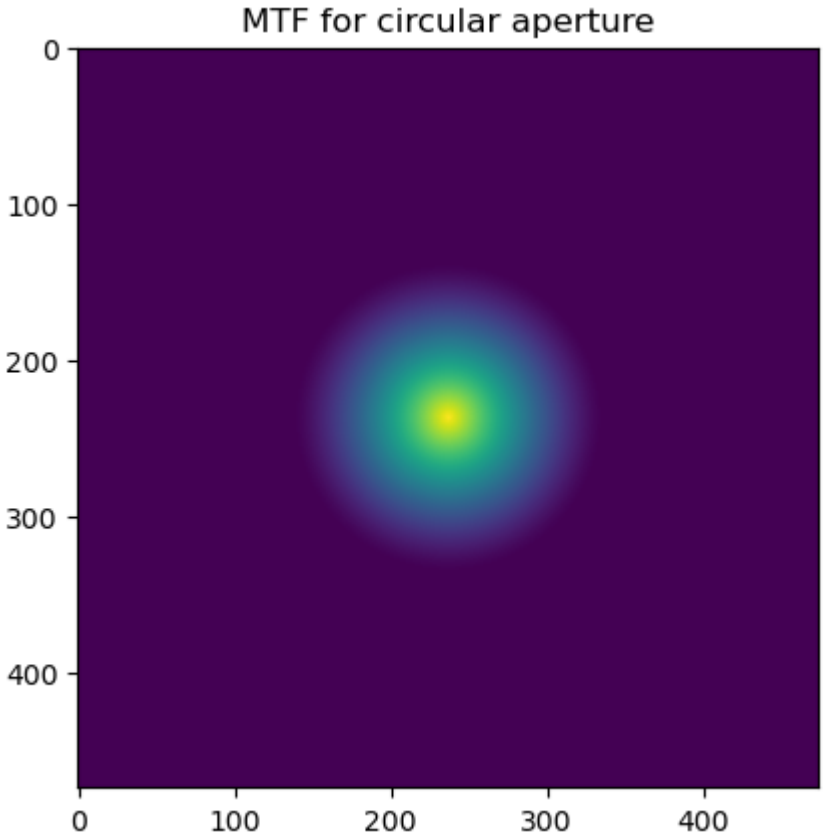
In [15]:
```
# Low pass filter
wide_lp_radius = 50/(N*p)
wide_lp_filter = np.zeros((N, N))
wide_lp_filter[fx**2 + fy**2 <= wide_lp_radius**2] = 1
plt.imshow(wide_lp_filter)
```

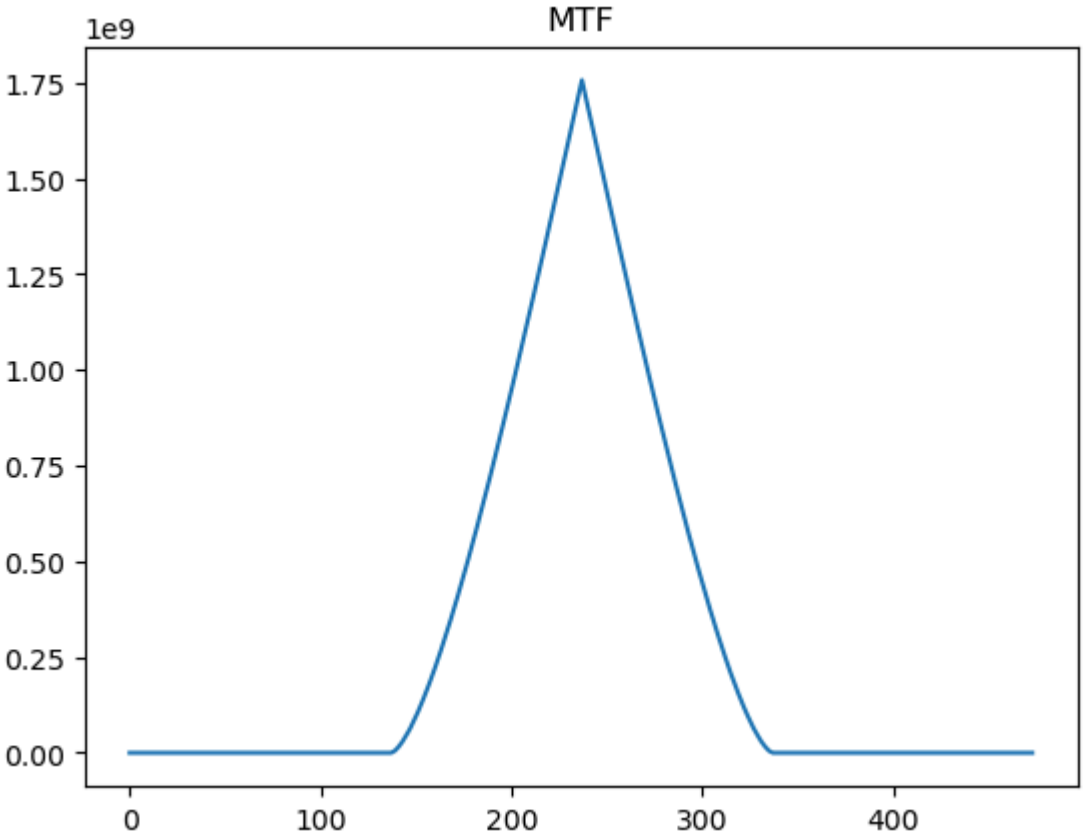
Out[15]: <matplotlib.image.AxesImage at 0x265c881a050>



In [16]:
```
H_circ = ft2(wide_lp_filter)
H_circ_abs = np.abs(H_circ)**2 # OTF
OTF_circ = ft2(H_circ_abs)

plt.figure()
plt.imshow(np.abs(OTF_circ))
```

```
plt.title("MTF for circular aperture")
plt.show()

plt.figure()
plt.plot(np.abs(OTF_circ)[N//2,:])
plt.title("MTF")
plt.show()
```



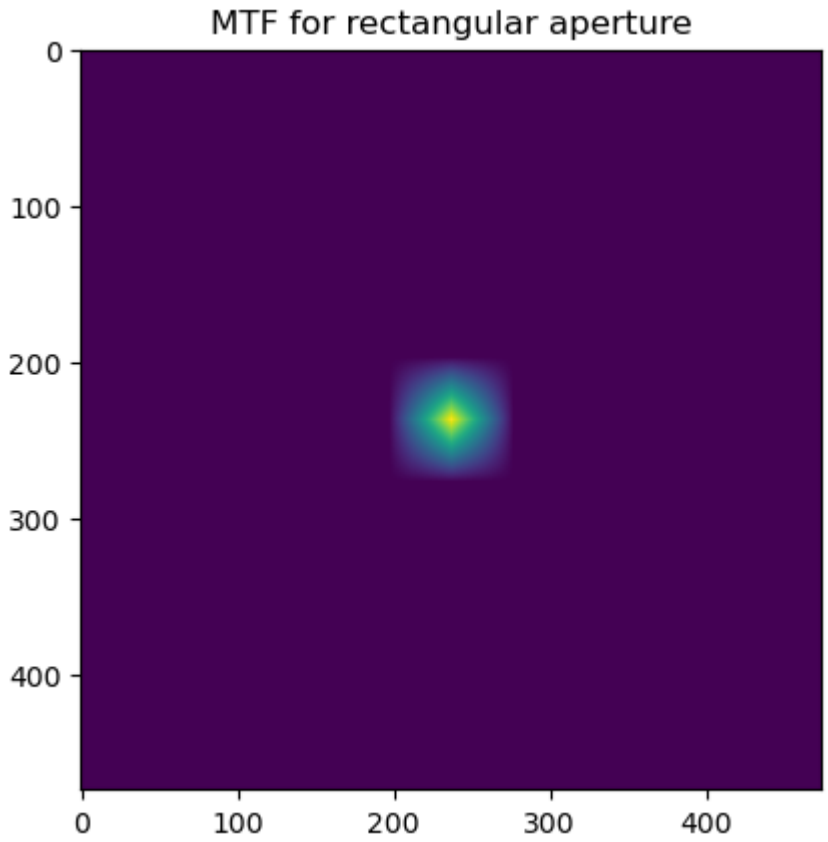


The MTF shows how different spatial frequencies are transferred through the optical system. The plot shows a peak at the center (low frequencies) and then a roll-off at higher frequencies, which is typical for a circular aperture. The sharp peak near the center suggests that low spatial frequencies (large, broad details) are transmitted well through the aperture. As you move toward higher frequencies (finer details), the modulation transfer function decreases, meaning that the system is less effective at transmitting finer details. This leads to an overall blur in the intensity distribution.

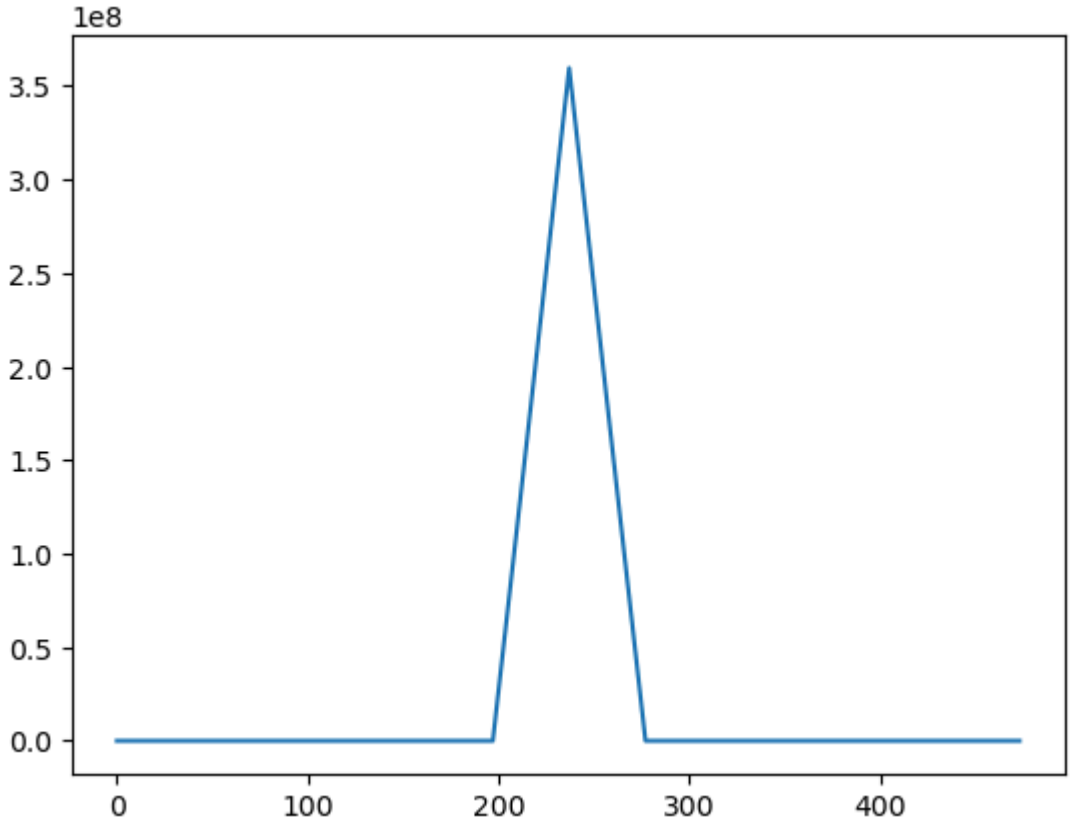
4.2. Rectangular aperture

In [17]:
```
h_rect = np.zeros((N,N))
h_rect[(np.abs(fx) <= 20/(N*p)) & (np.abs(fy) <= 20/(N*p))] = 1
```

```
H_rect = ft2(h_rect)
H_rect_abs = np.abs(H_rect)**2
OTF_rect = ft2(H_rect_abs)

plt.figure()
plt.imshow(np.abs(OTF_rect))
plt.title("MTF for rectangular aperture")
plt.show()

plt.figure()
plt.plot(OTF_rect[N//2,:])
plt.show()
```



```
C:\Users\nitin negi\anaconda3\Lib\site-packages\matplotlib\cbook.py:1699: ComplexWarning: Casting complex values to real discar
ds the imaginary part
  return math.isfinite(val)
C:\Users\nitin negi\anaconda3\Lib\site-packages\matplotlib\cbook.py:1345: ComplexWarning: Casting complex values to real discar
ds the imaginary part
  return np.asarray(x, float)
```

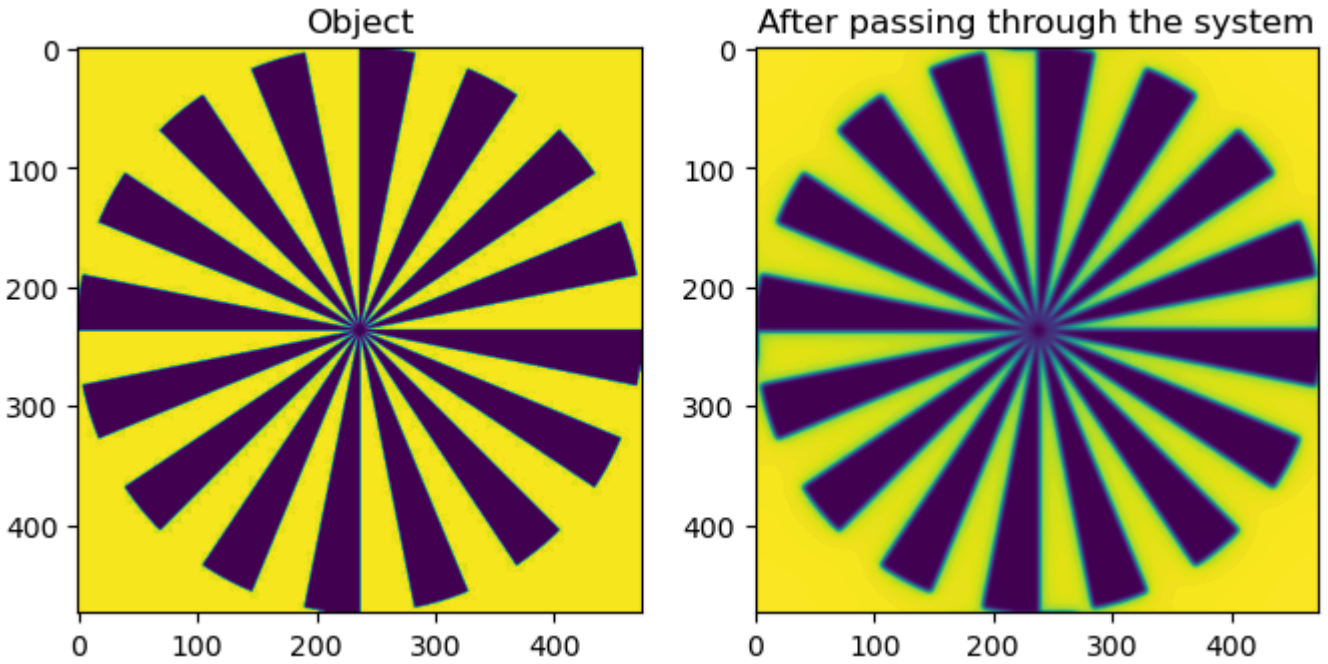


The rectangular aperture has a more abrupt transition between the spatial frequencies that are transmitted and those that are attenuated or blocked, leading to more pronounced loss of high frequencies.

4.3. Effect of OTF on an object

4.3.1 Circular aperture

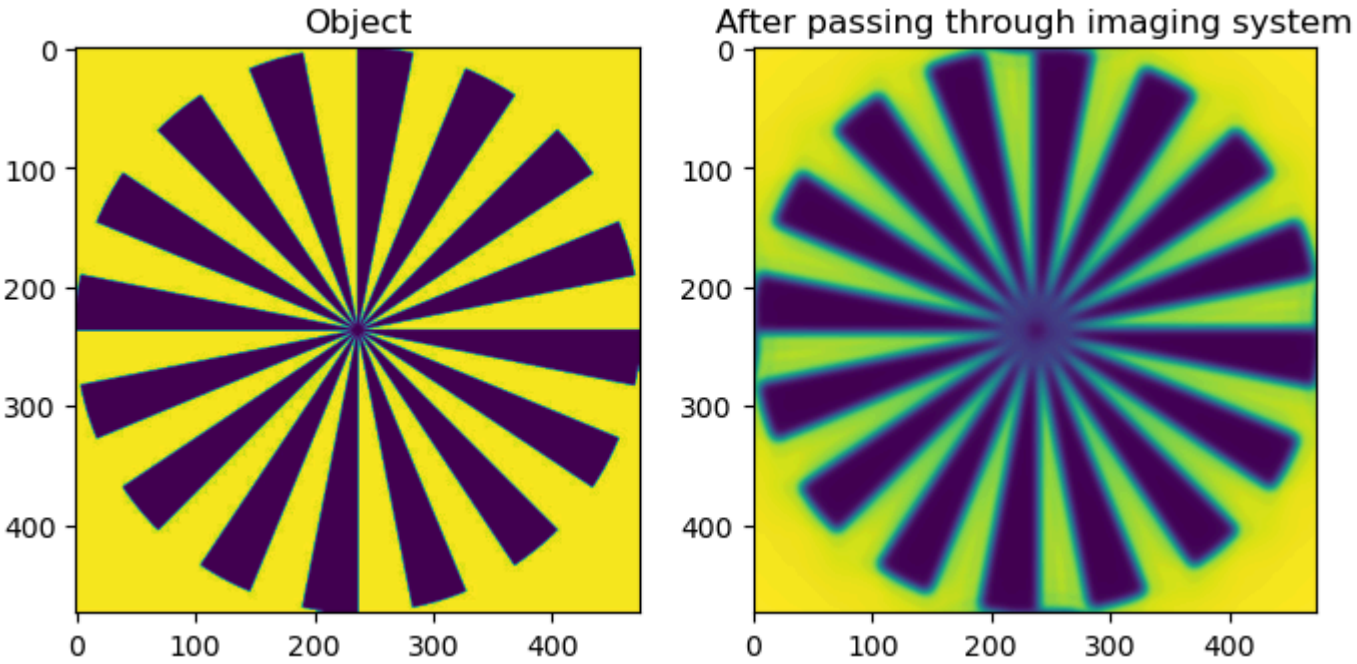
In [18]:
```
i1 = im # Siemen star
I1 = ft2(i1) # Fourier transform of the object
I1_OTF_circ = I1*OTF_circ # Multiply the object with an OTF
i1_otf_circ = ft2(I1_OTF_circ)

plt.figure(figsize = (8,5))
plt.subplot(1,2,1)
plt.imshow(np.abs(im)**2)
plt.title("Object")
plt.subplot(1,2,2)
plt.imshow(np.abs(i1_otf_circ)**2)
plt.title("After passing through the system")
plt.show()
```



The effect of the OTF can be seen in the right image. The center of the image where higher frequencies are present is blurred due to the OTF.

4.3.2 Rectangular Aperture

In [19]:
```
I1_OTF_rect = I1*OTF_rect
i1_otf_rect = ft2(I1_OTF_rect)

plt.figure(figsize = (8,5))
plt.subplot(1,2,1)
plt.imshow(np.abs(im)**2)
plt.title("Object")
plt.subplot(1,2,2)
plt.imshow(np.abs(i1_otf_rect)**2)
plt.title("After passing through imaging system")
plt.show()
```

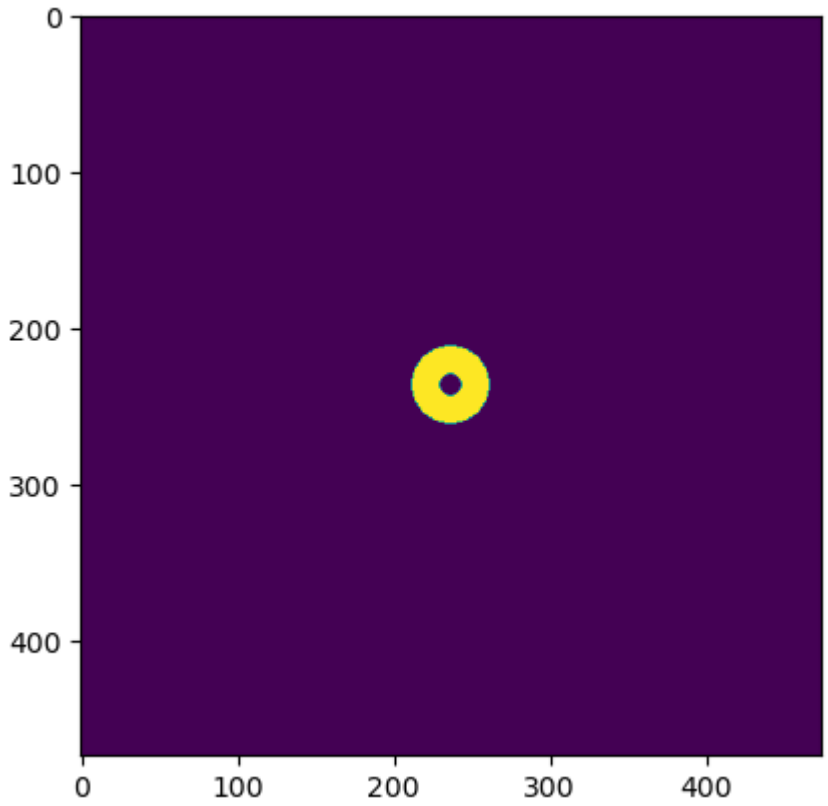


The effect of a rectangular aperture is similar to the circular aperture, but the blur is more prominent which is the result of sharp fall-off in the higher frequency region.

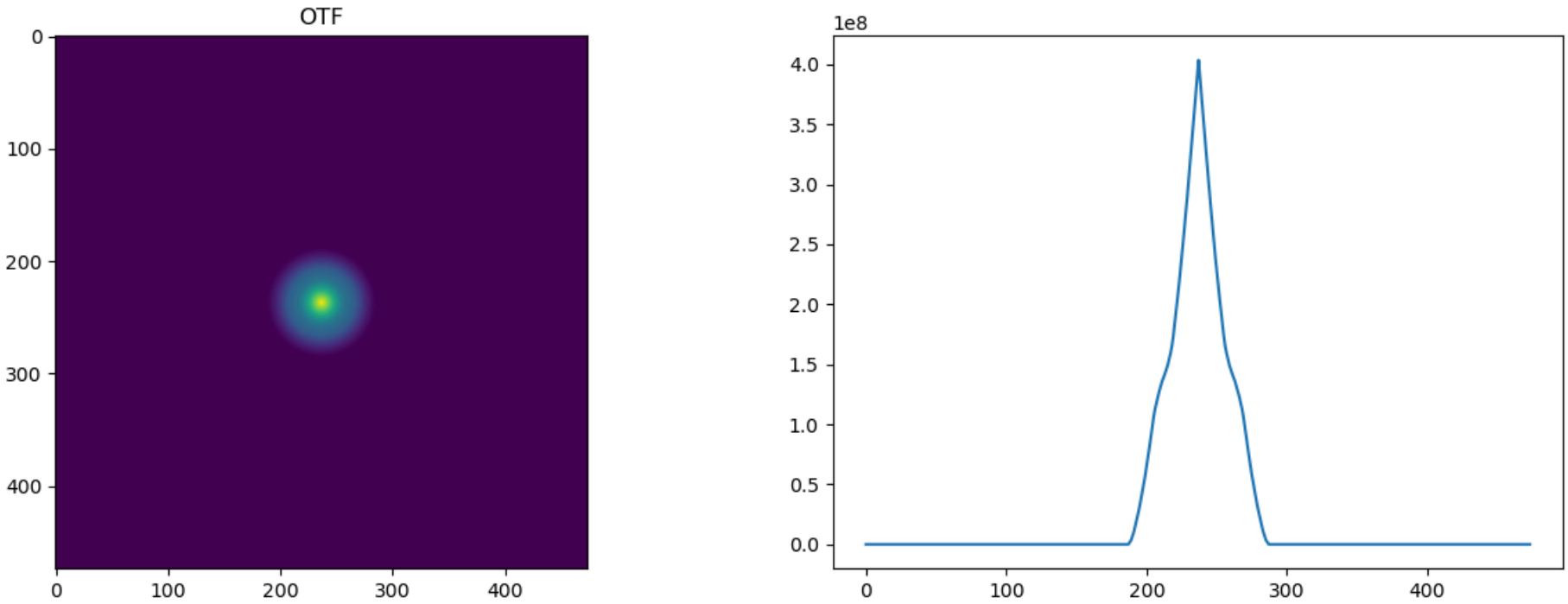
4.3.3. Annular Aperture

In [20]:
```
rho = np.sqrt(fx**2 + fy**2)
rho_0 = 1e4

width = 1000

annular_ring = np.zeros_like(rho)
annular_ring[(rho >= rho_0/3 - width/2) & (rho <= rho_0 + width/2)] = 1

plt.imshow(annular_ring)
```

Out[20]:
```
<matplotlib.image.AxesImage at 0x265ca978fd0>
```



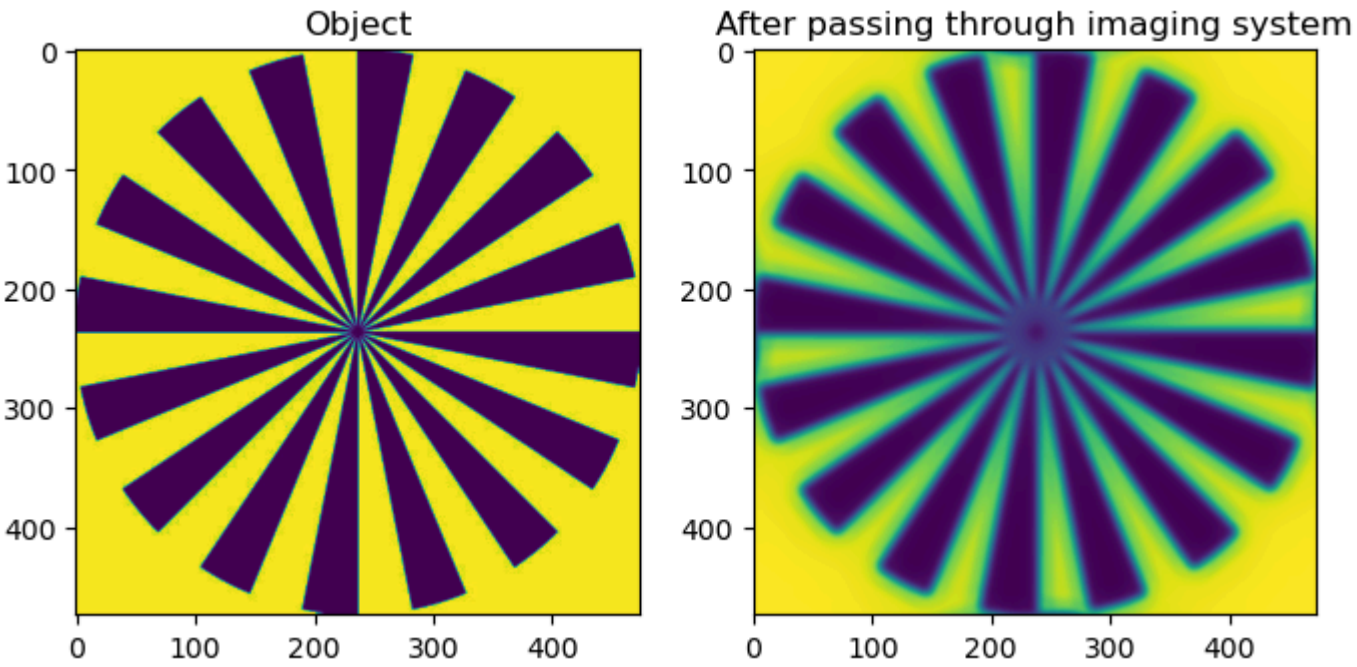
In [21]:
```
H_annular = ft2(annular_ring)
H_annular_abs = np.abs(H_annular)**2
OTF_annular = ft2(H_annular_abs)

plt.figure(figsize = (15,5))
plt.subplot(1,2,1)
plt.imshow(np.abs(OTF_annular))
plt.title("OTF")

plt.subplot(1,2,2)
plt.plot(OTF_annular[N//2,:])
plt.show()
```



The curve shows that the high spatial frequency response of the sysem is improved.

In [22]:
```
I1_OTF_annular = I1*OTF_annular
i1_otf_annular = ft2(I1_OTF_annular)
```

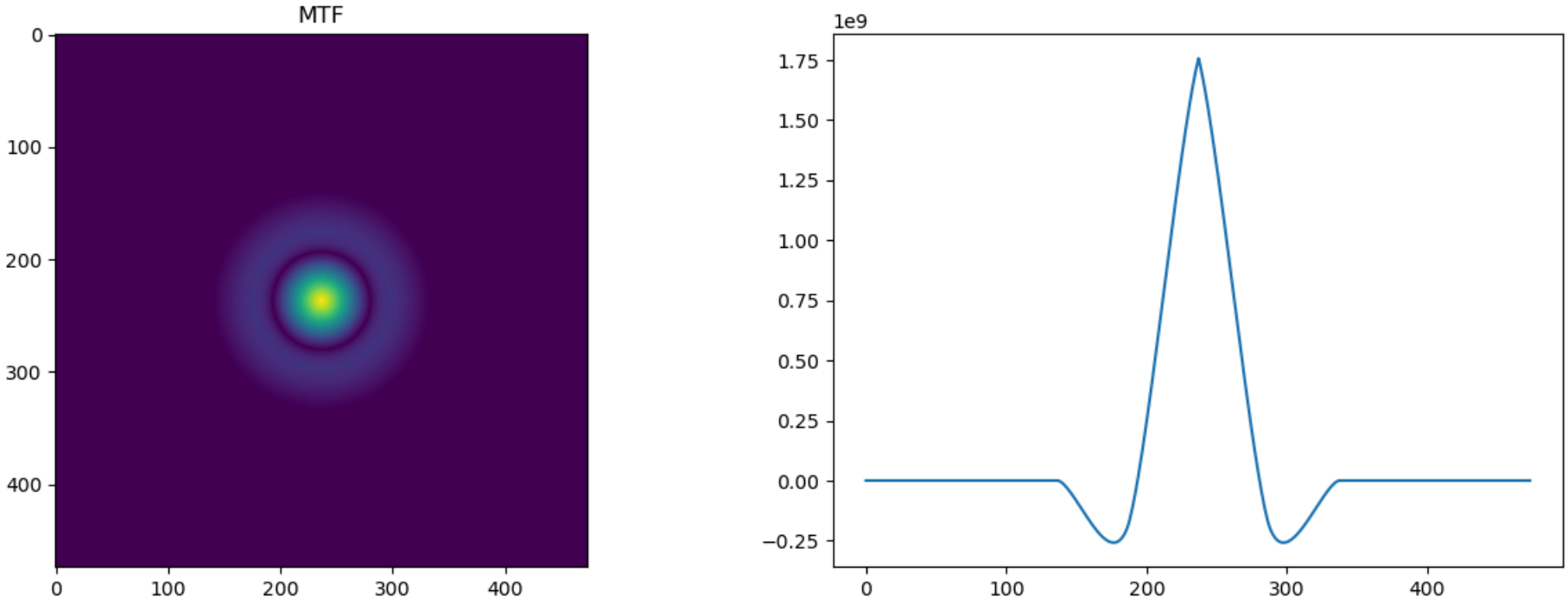
```
plt.figure(figsize = (8, 5))
plt.subplot(1,2,1)
plt.imshow(np.abs(im)**2)
plt.title("Object")
plt.subplot(1,2,2)
plt.imshow(np.abs(i1_otf_annular)**2)
plt.title("After passing through imaging system")
plt.show()
```



4.3.4. Aperture with a vortex phase

In [23]:
```
phi = np.arctan2(y,x)
vortex = np.exp(1j*phi)*wide_lp_filter

H_vortex = ft2(vortex)

H_vortex_abs = np.abs(H_vortex)**2
OTF_vortex = ft2(H_vortex_abs)
```

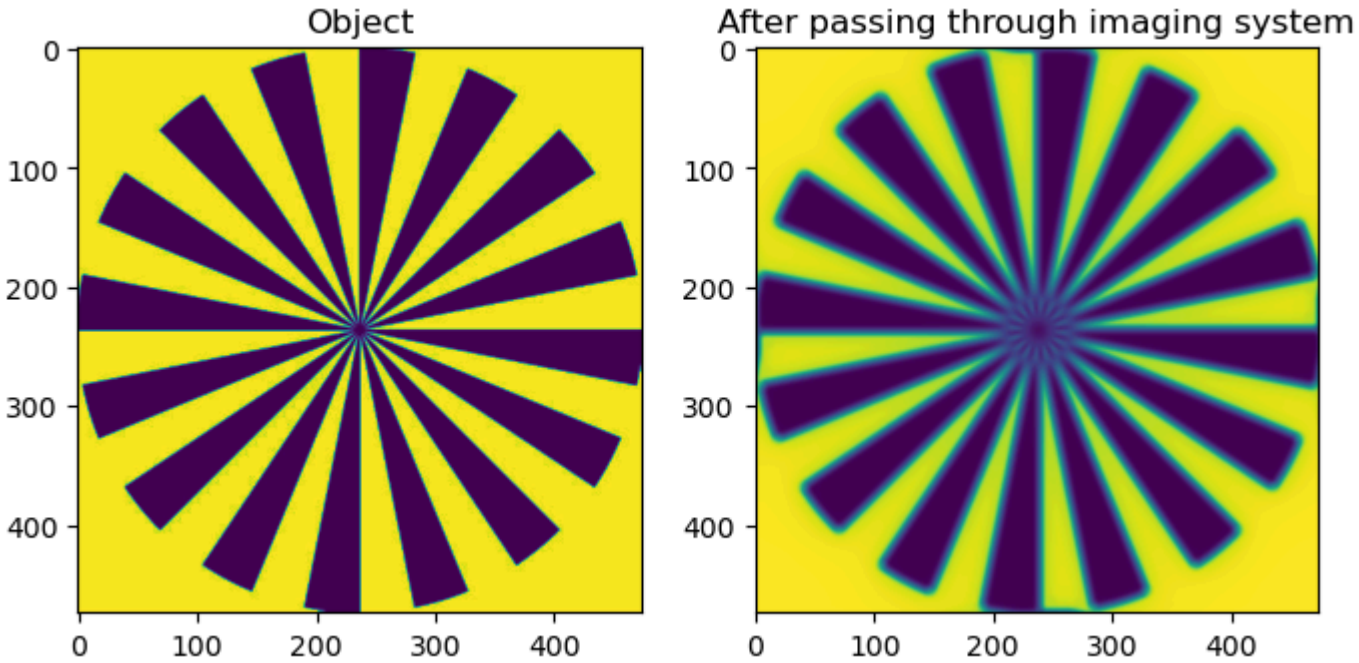
```
plt.figure(figsize = (15,5))
plt.subplot(1,2,1)
plt.imshow(np.abs(OTF_vortex))
plt.title("MTF")
plt.subplot(1,2,2)
plt.plot(OTF_vortex[N//2,:])
plt.show()
```



The negative values in the OTF curve denote a contrast reversal.

In [24]:
```
I1_OTF_vortex = I1*OTF_vortex
i1_otf_vortex = ft2(I1_OTF_vortex)

plt.figure(figsize = (8,5))
plt.subplot(1,2,1)
plt.imshow(np.abs(im)**2)
plt.title("Object")
```

```
plt.subplot(1,2,2)
plt.imshow(np.abs(i1_otf_vortex)**2)
plt.title('After passing through imaging system')
plt.show()
```

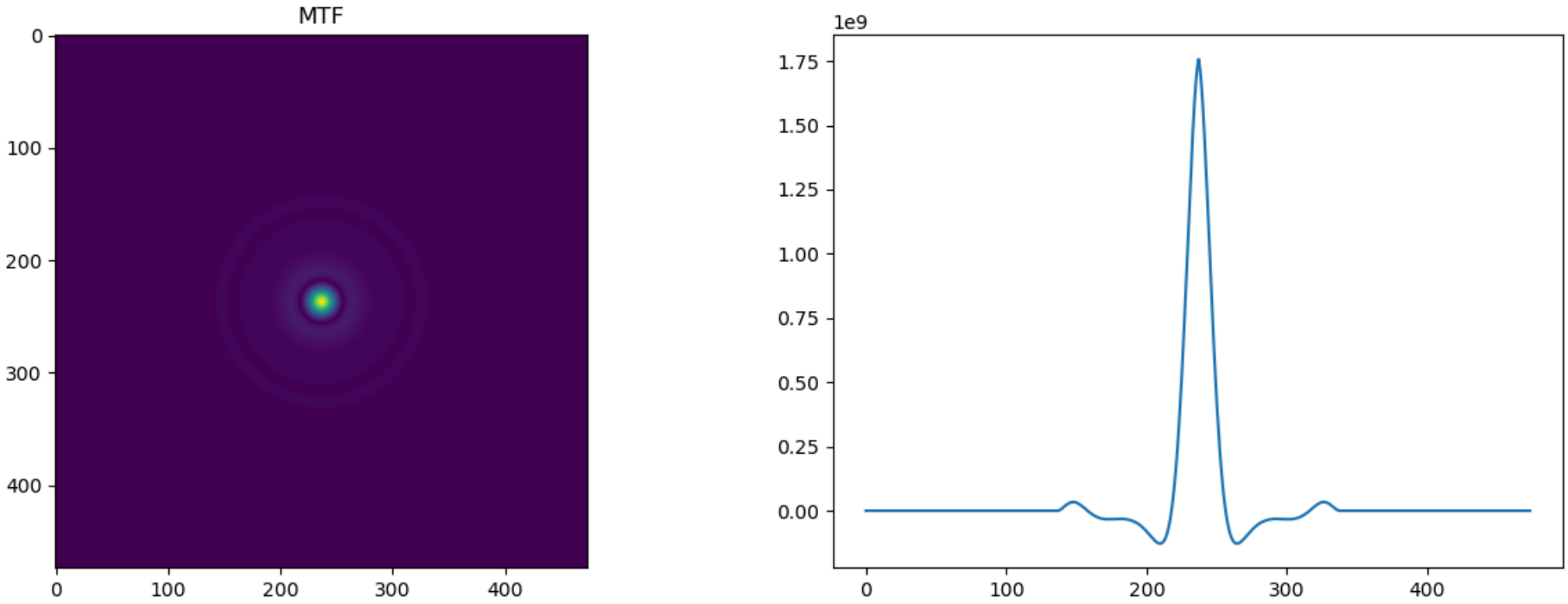


We can see that after passing through the imaging system, there's a region in the image where the bright regions become dark and vice versa. This is due to the negative values in the OTF curve showing contrast reversal.

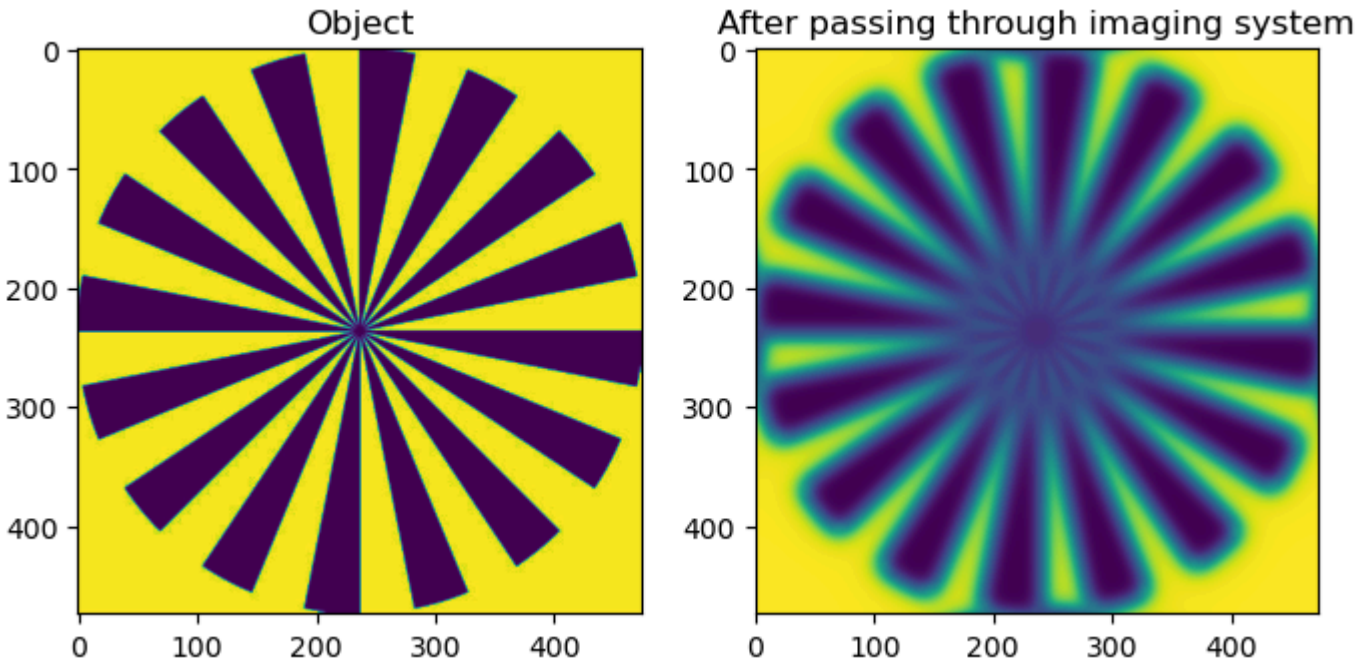
4.3.5. Aperture with quadratic phase

In [25]:
```
quadratic = np.exp(1j*2*np.pi*(fx**2 + fy**2)/wide_lp_radius**2)*wide_lp_filter

H_quad = ft2(quadratic)
H_quad_abs = np.abs(H_quad)**2
OTF_quad = ft2(H_quad_abs)

plt.figure(figsize = (15,5))
plt.subplot(1,2,1)
```

```
plt.imshow((np.abs(OTF_quad)))
plt.title("MTF")
plt.subplot(1,2,2)
plt.plot(OTF_quad[N//2,:])
plt.show()
```



In [26]:
```
I1_OTF_quad = I1*OTF_quad
i1_otf_quad = ft2(I1_OTF_quad)

plt.figure(figsize = (8,5))
plt.subplot(1,2,1)
plt.imshow(np.abs(im)**2)
plt.title("Object")
plt.subplot(1,2,2)
plt.imshow(np.abs(i1_otf_quad)**2)
plt.title("After passing through imaging system")
plt.show()
```



The image after the system shows contrast reversal for the regions where the OTF value is negative.

5. Deconvolution using Wiener Filter

Wiener filter, denoted by g(with Fourier transform G), applies to the observation y such that:

$$\hat{x} = g * y$$

$$\hat{X} = GY$$

This filter is established in the statistical framework: the image x and the noise b are considered to be random variables. They are also assumed to be statistically independent. As a result, the observation y and the estimate $\hat{x}$ are also random variables.

The Wiener filter is given by

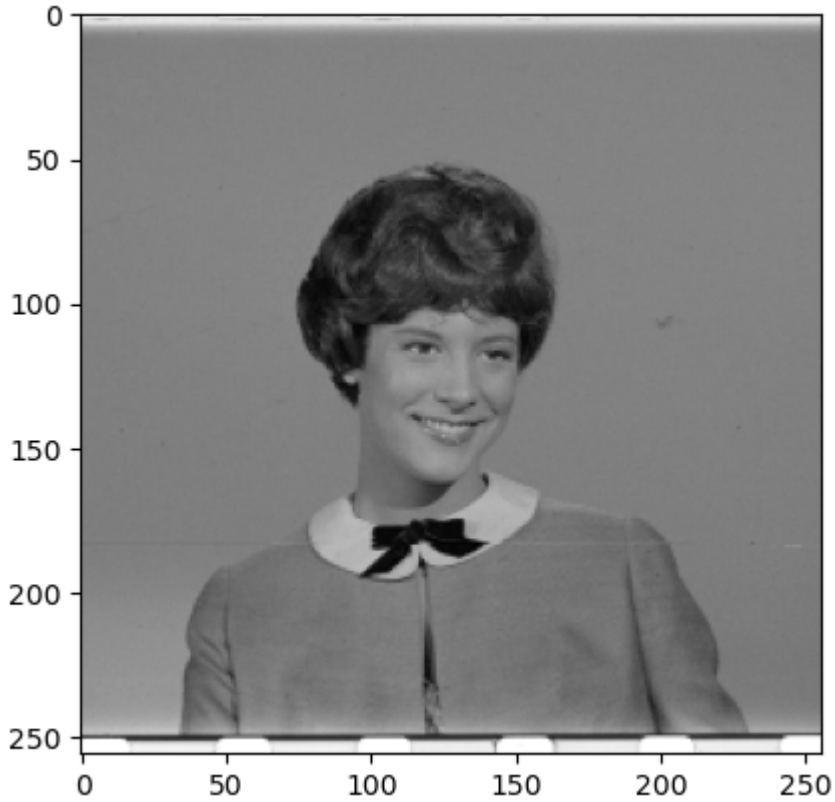
$$G = \frac{H^* S_x}{|H|^2 S_x + Sb}$$

where S_x and S_b are the power spectral densities. The deconvolved image is the inverse transform of GY:

$$\hat{x} = \mathscr{F}^{-1}[\frac{H^*}{|H|^2 + \frac{Sb}{S_x}} Y]$$

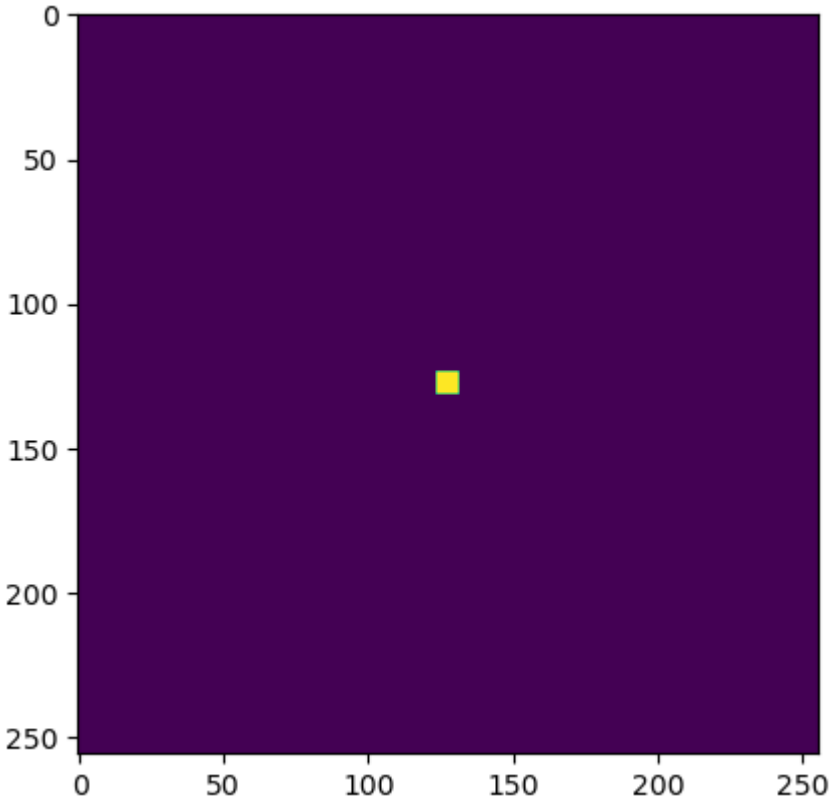
In [27]:
```
im1 = io.imread("4.1.03.tiff")

if im1.ndim == 3:
    im1 = color.rgb2gray(im1)

plt.imshow(im1, cmap ="gray")
```

Out[27]:
```
<matplotlib.image.AxesImage at 0x265c54d4b10>
```



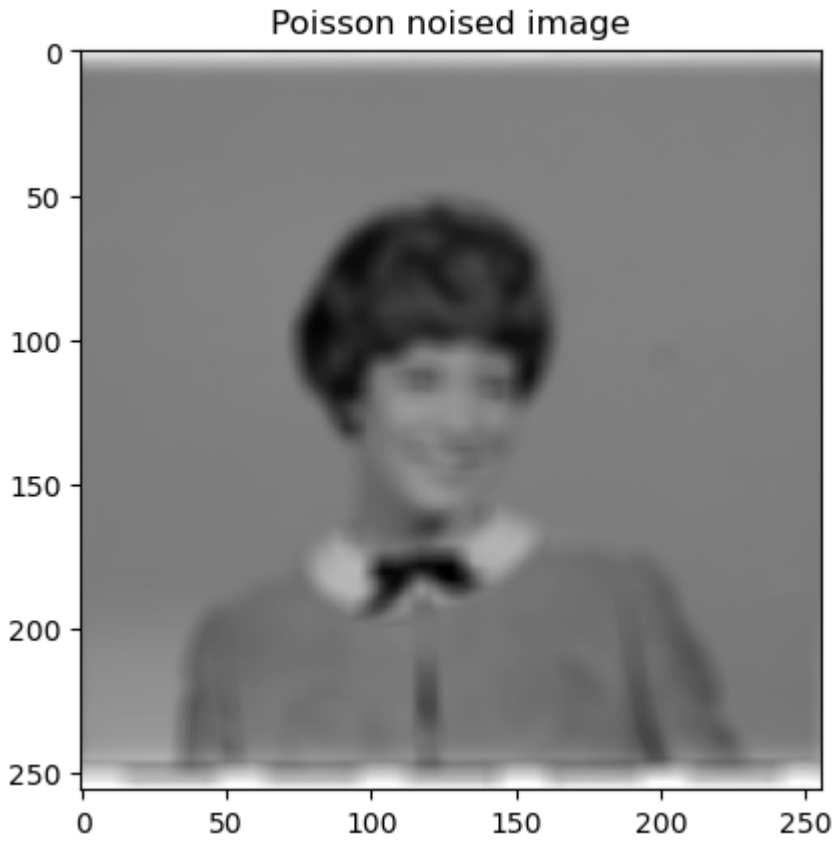
In [28]:
```
# Define a square kernel
square = np.zeros((256, 256))
square[124:132,124:132] = 1

plt.imshow(square)
```

Out[28]: <matplotlib.image.AxesImage at 0x265c547b850>

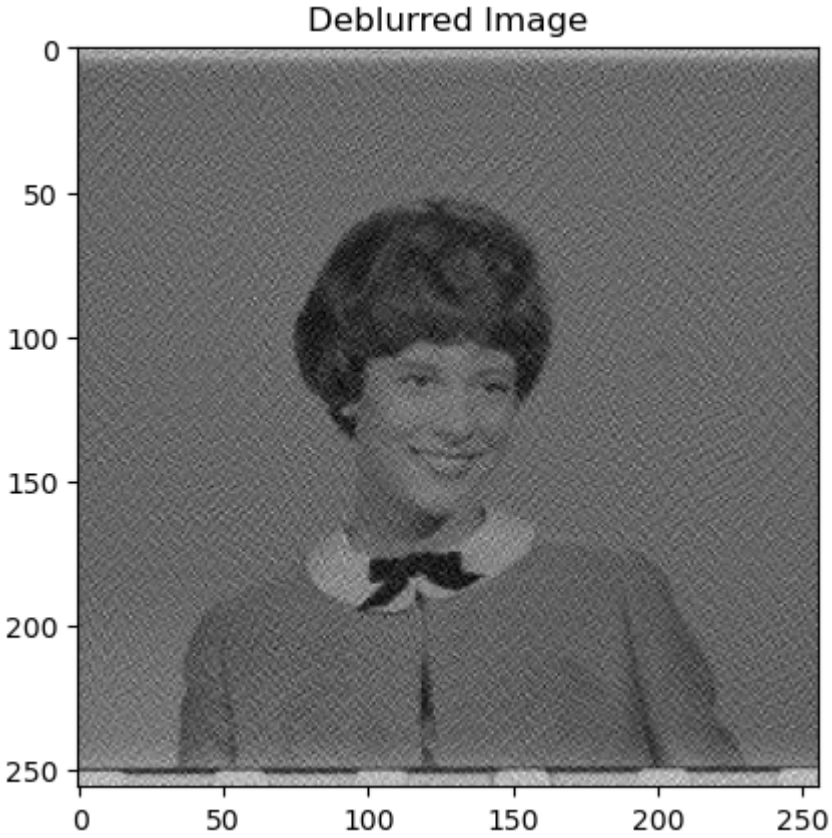


In [29]:
```
# Applying psf on the image
im1_fft = ft2(im1)
psf = ft2(square)
im1_psf = im1_fft*psf
im1_ifft = ift2(im1_psf)
im1_ifft = im1_ifft/np.max(im1_ifft)

plt.figure()
plt.imshow(np.abs(im1_ifft)**2, cmap = "gray")
plt.title("Blurred image")
plt.show()
```

In [30]:
```
# Adding poisson noise to the image with defined photon counts
im1_mean = 1e6*np.abs(im1_ifft)/np.abs(np.mean(im1_ifft)) # photon count = 1e6
im1_noised = np.random.poisson(im1_mean)

plt.figure()
plt.imshow(im1_noised, cmap = "gray")
plt.title("Poisson noised image")
plt.show()
```



In [31]:
```
#Wiener Deconvolution
wiener = np.conjugate(psf)/((np.abs(psf)**2)+ 1/np.sqrt(1e6)) #Wiener Filter
noised_fft = ft2(im1_noised)

deblur = ift2(noised_fft*wiener) # Deblurred Image

plt.figure()
plt.imshow(np.abs(deblur), cmap = "gray")
plt.title("Deblurred Image")
plt.show()
```



The result of wiener filter is presented. There are some artifacts in the deconvolved image, but the recovery is there.

6. Conclusion

This lab demonstrated the role of Fourier optics in analyzing and manipulating optical systems. Through the use of various filters, such as low-pass, high-pass, and phase contrast, we observed how different spatial frequencies contribute to image clarity. The results of this study indicate that low-pass filters tend to blur images by removing high-frequency components, while high-pass filters enhance edge definition by emphasizing these frequencies. The optical transfer functions provided insight into how different apertures affect spatial frequency transmission, with notable contrast differences when using vortex and quadratic phase filters. Finally, Wiener deconvolution proved to be an

effective method for reducing noise and recovering image details, though some artifacts persisted. Overall, the experiments provided a comprehensive understanding of the strengths and limitations of optical filtering and its impact on image quality.

In [ ]: