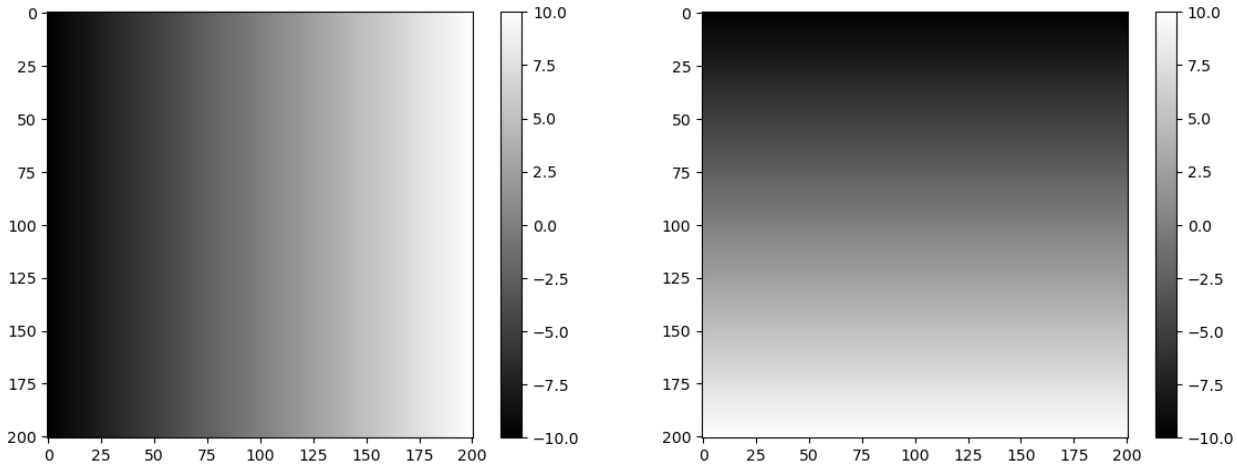
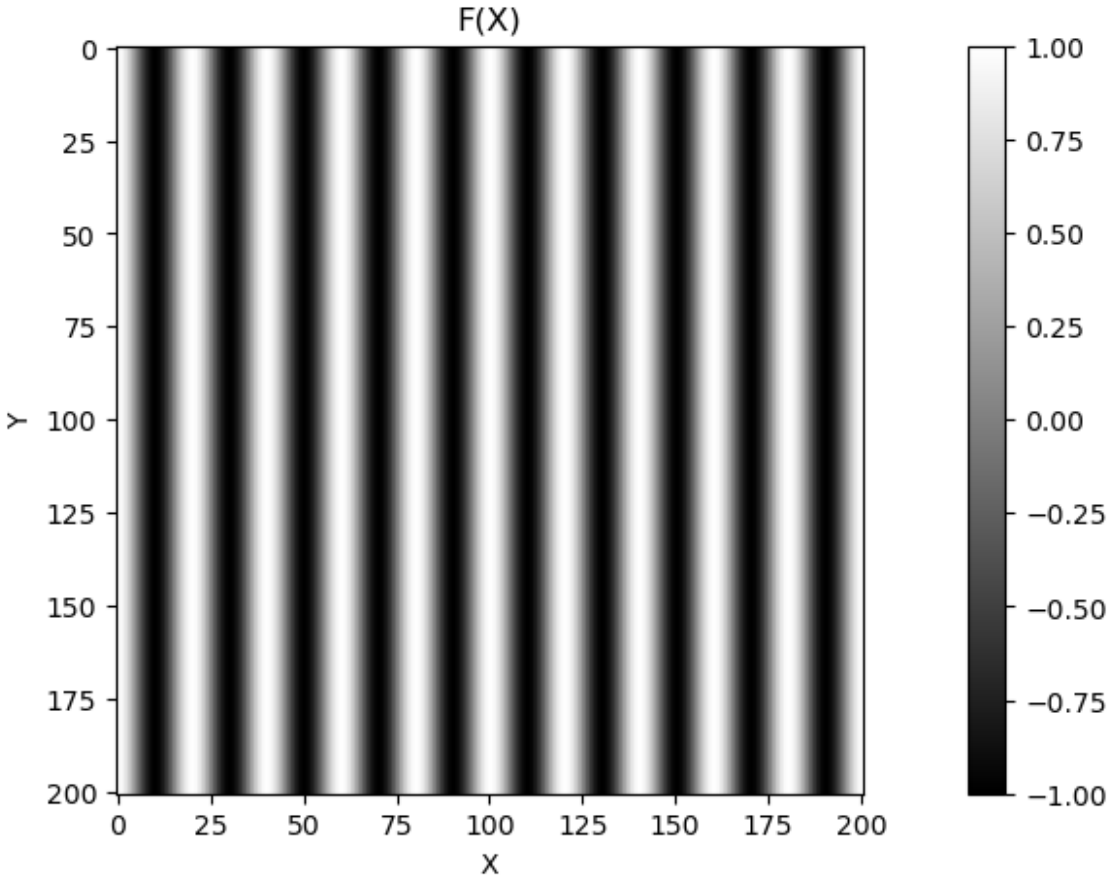
```python
# CODE BLOCK 1
import numpy as np
import matplotlib.pyplot as plt

# Define a range of x values, Here x ranges from -10 mm to 10 mm with
201 samples inbetween (including the endpoints), at 0.1 mm increment
xr = np.linspace(-10, 10, 201)

# Define a range of y values. The range of y is the same as x.
yr = np.linspace(-10, 10, 201)

# Create a 2D grid of x & y values
[X, Y] = np.meshgrid(xr, yr)

print(xr.shape, yr.shape, X.shape, Y.shape)
print(X)
```

```
(201,) (201,) (201, 201) (201, 201)
[[-10.   -9.9  -9.8 ...   9.8   9.9  10. ]
 [-10.   -9.9  -9.8 ...   9.8   9.9  10. ]
 [-10.   -9.9  -9.8 ...   9.8   9.9  10. ]
 ...
 [-10.   -9.9  -9.8 ...   9.8   9.9  10. ]
 [-10.   -9.9  -9.8 ...   9.8   9.9  10. ]
 [-10.   -9.9  -9.8 ...   9.8   9.9  10. ]]
```

```python
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

x = ax1.imshow(X, cmap='gray')
fig.colorbar(x, ax=ax1)
y = ax2.imshow(Y, cmap='gray')
fig.colorbar(y, ax=ax2)
plt.show()
```



```
F = np.cos(np.pi*X)

fig, (ax1) = plt.subplots(1, 1, figsize=(14, 5))

x = ax1.imshow(F, cmap='gray')
fig.colorbar(x, ax=ax1)
ax1.set_title('F(X)')
ax1.set_xlabel('X')
ax1.set_ylabel('Y')
plt.show()
```




```
from numpy.fft import fft2, fftshift, ifftshift

fftshifted = fftshift(fft2(ifftshift(F)))
print(fftshifted.shape)

origin = [100,100]
```

```
(201, 201)
```

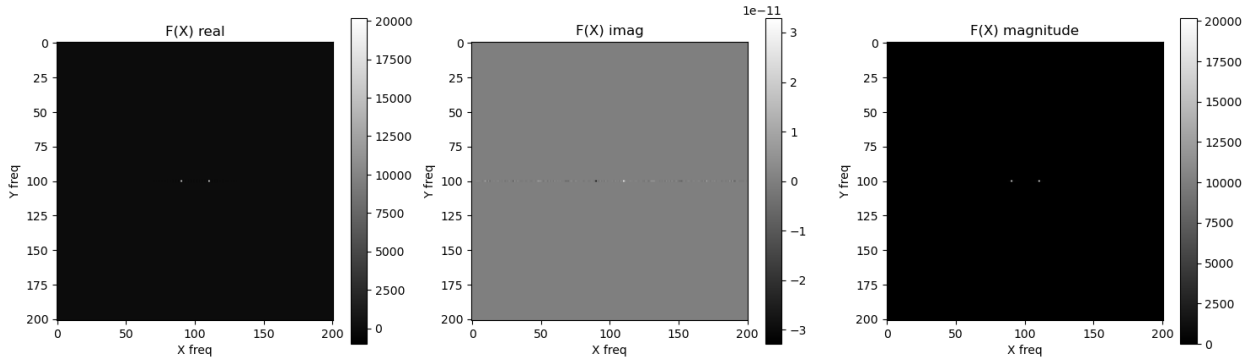
Question 1

200 samples in range of 20 (-10,10) means 10 samples per mm. 10 samples/mm

The Nyquist frequency is 1/2 the sampling frequency. = 5 samples/mm

```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(18, 5))


x = ax1.imshow(np.real(fftshifted), cmap='gray')
fig.colorbar(x, ax=ax1)
ax1.set_title('F(X) real')
ax1.set_xlabel('X freq')
ax1.set_ylabel('Y freq')

y = ax2.imshow(np.imag(fftshifted), cmap='gray')
fig.colorbar(y, ax=ax2)
ax2.set_title('F(X) imag')
ax2.set_xlabel('X freq')
ax2.set_ylabel('Y freq')

magnitude = np.imag(fftshifted)**2 + np.real(fftshifted)**2
magnitude = np.sqrt(magnitude)

print(np.max(np.real(fftshifted)), np.min(np.real(fftshifted)))
print(np.max(magnitude), np.min(magnitude))

z = ax3.imshow(magnitude, cmap='gray')
fig.colorbar(z, ax=ax3)
ax3.set_title('F(X) magnitude')
ax3.set_xlabel('X freq')
ax3.set_ylabel('Y freq')

plt.show()
```

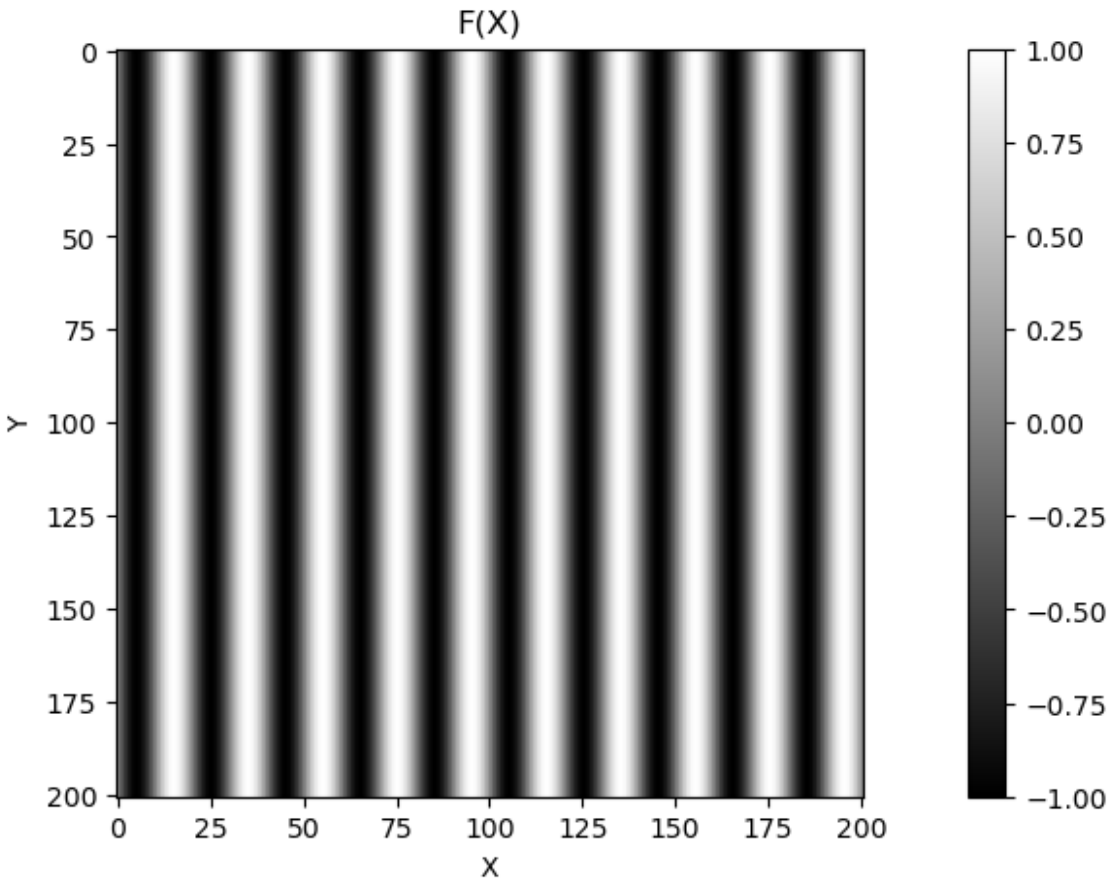
```
20168.532296092682 -1011.6109265114898
20168.532296092682 0.0
```



F(X) real plot has two bright white spots at Y freq = 100 and symmetrical around X freq = 100. The fourier transform of cosine is real and even, so the peaks are real as well. F(X) imaginary plot is roughly 0 because the fourier transform of a cosine function does not have an imaginary component. The noise we see may be a result of discretizing the cosine function The F(X) magnitude plot looks the same as the F(X) real plot. The magnitude plot is nearly identical to the real part, which is expected for a real, even signal that has a real Fourier transform.

Question 2

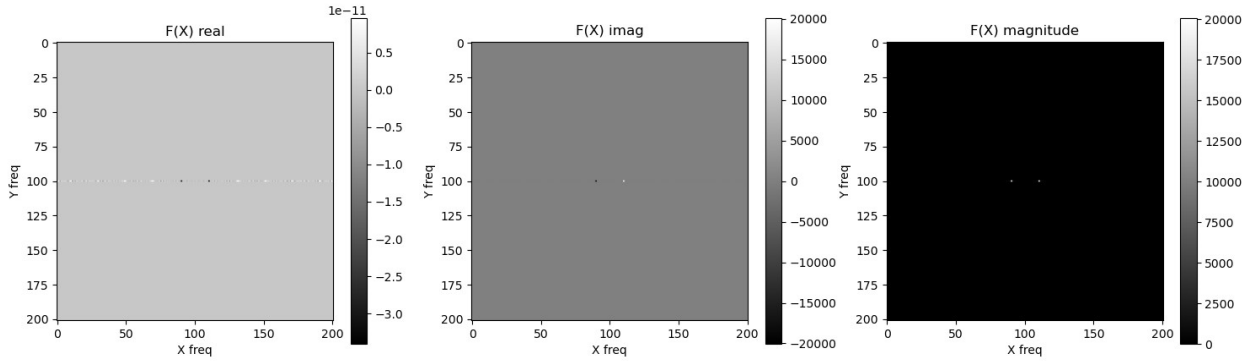
```
F = np.cos(np.pi*X + np.pi/2)

fig, (ax1) = plt.subplots(1, 1, figsize=(14, 5))

x = ax1.imshow(F, cmap='gray')
fig.colorbar(x, ax=ax1)
ax1.set_title('F(X)')
ax1.set_xlabel('X')
ax1.set_ylabel('Y')
plt.show()

fftshifted = fftshift(fft2(ifftshift(F)))
print(fftshifted.shape)
```




```
(201, 201)
```

```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(18, 5))


x = ax1.imshow(np.real(fftshifted), cmap='gray')
fig.colorbar(x, ax=ax1)
ax1.set_title('F(X) real')
ax1.set_xlabel('X freq')
ax1.set_ylabel('Y freq')

y = ax2.imshow(np.imag(fftshifted), cmap='gray')
fig.colorbar(y, ax=ax2)
ax2.set_title('F(X) imag')
ax2.set_xlabel('X freq')
ax2.set_ylabel('Y freq')

magnitude = np.imag(fftshifted)**2 + np.real(fftshifted)**2
magnitude = np.sqrt(magnitude)

z = ax3.imshow(magnitude, cmap='gray')
fig.colorbar(z, ax=ax3)
ax3.set_title('F(X) magnitude')
```

```
ax3.set_xlabel('X freq')
ax3.set_ylabel('Y freq')

plt.show()
```

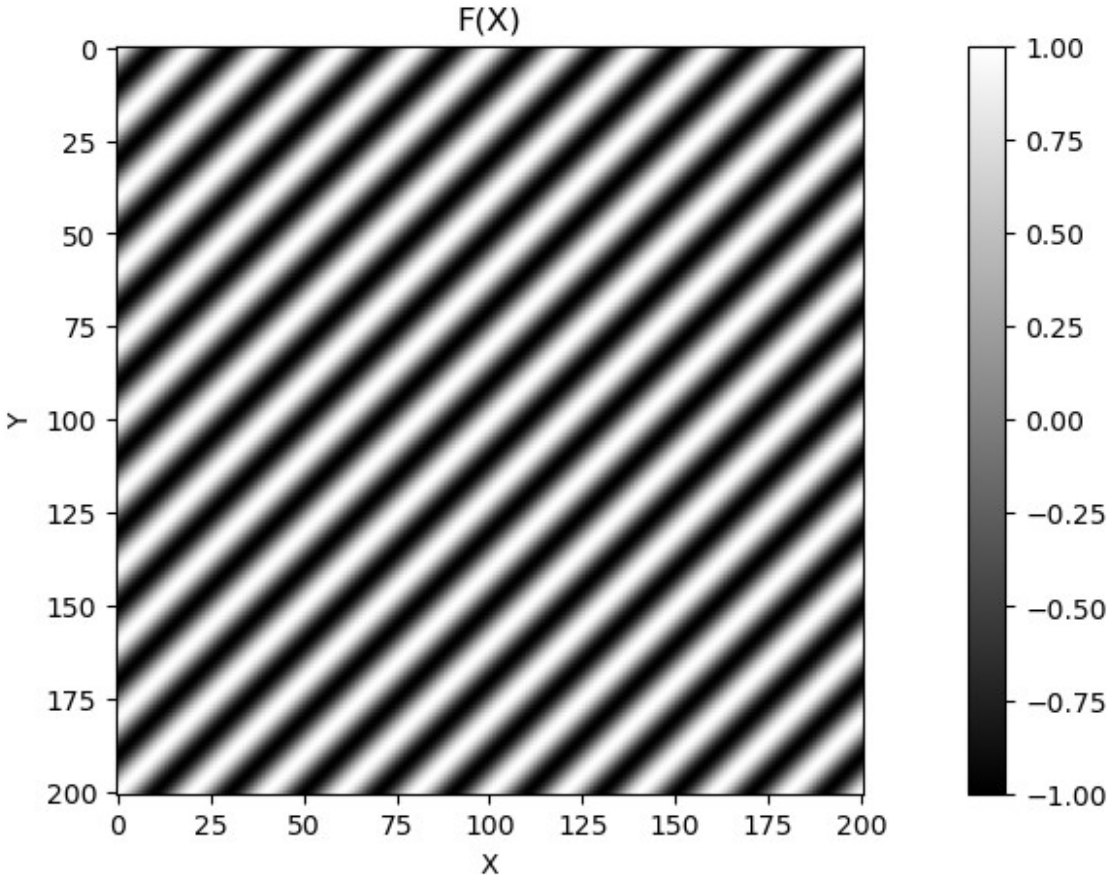


Essentially, this shift makes the F = cos(πx+2π)= − sin(πx) by the trig identity cos(θ+2π) = −sin(θ). Sine is odd so its Fourier transform is imaginary and odd. From the plot, we see that the real part = 0 because the dots are black, which makes sense since sine is odd and real. Now, the imaginary part has two symmetric peaks, but with opposite signs (one white one black), which aligns with the expectations since sine is odd-symmetric. The magnitude plot is the same as cosine plot though, since the amplitudes are the same for the sine and cosine functions.

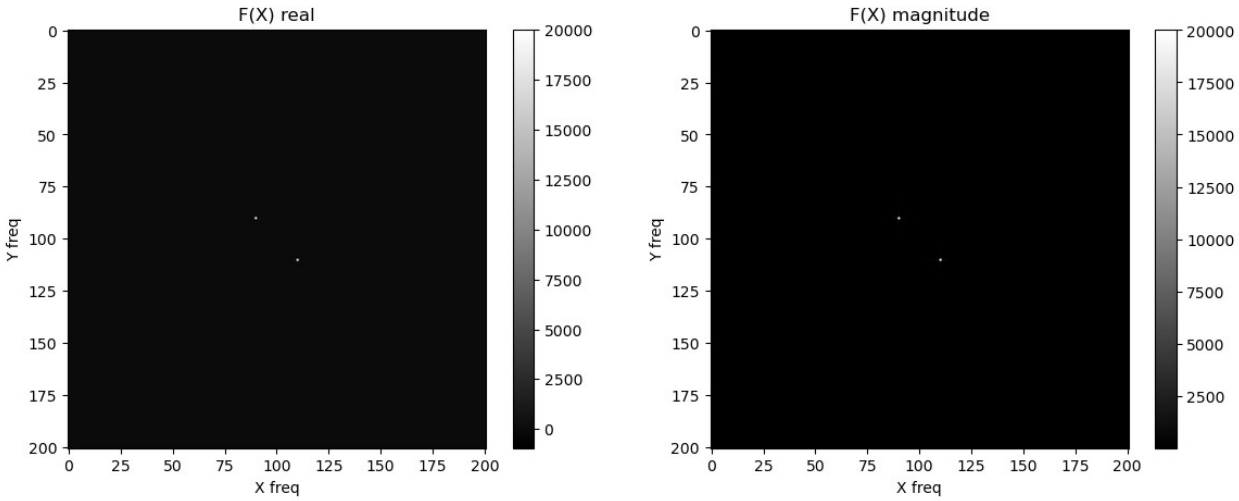
Question 3

```
F = np.cos(np.pi*X + 2*np.pi/2*Y)

fig, (ax1) = plt.subplots(1, 1, figsize=(14, 5))

x = ax1.imshow(F, cmap='gray')
fig.colorbar(x, ax=ax1)
ax1.set_title('F(X)')
ax1.set_xlabel('X')
ax1.set_ylabel('Y')
plt.show()

fftshifted = fftshift(fft2(ifftshift(F)))
print(fftshifted.shape)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))


x = ax1.imshow(np.real(fftshifted), cmap='gray')
fig.colorbar(x, ax=ax1)
ax1.set_title('F(X) real')
ax1.set_xlabel('X freq')
ax1.set_ylabel('Y freq')
```

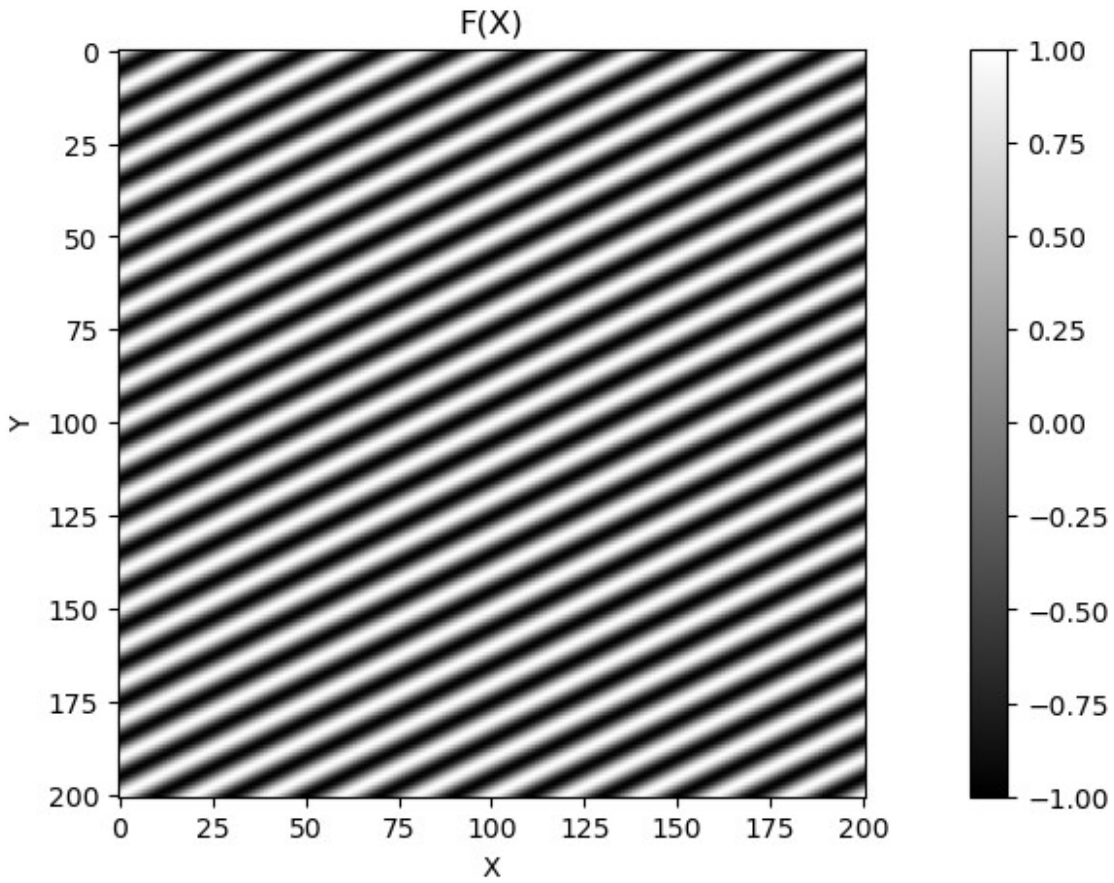
```
magnitude = np.imag(fftshifted)**2 + np.real(fftshifted)**2
magnitude = np.sqrt(magnitude)

z = ax2.imshow(magnitude, cmap='gray')
fig.colorbar(z, ax=ax2)
ax2.set_title('F(X) magnitude')
ax2.set_xlabel('X freq')
ax2.set_ylabel('Y freq')

plt.show()
```



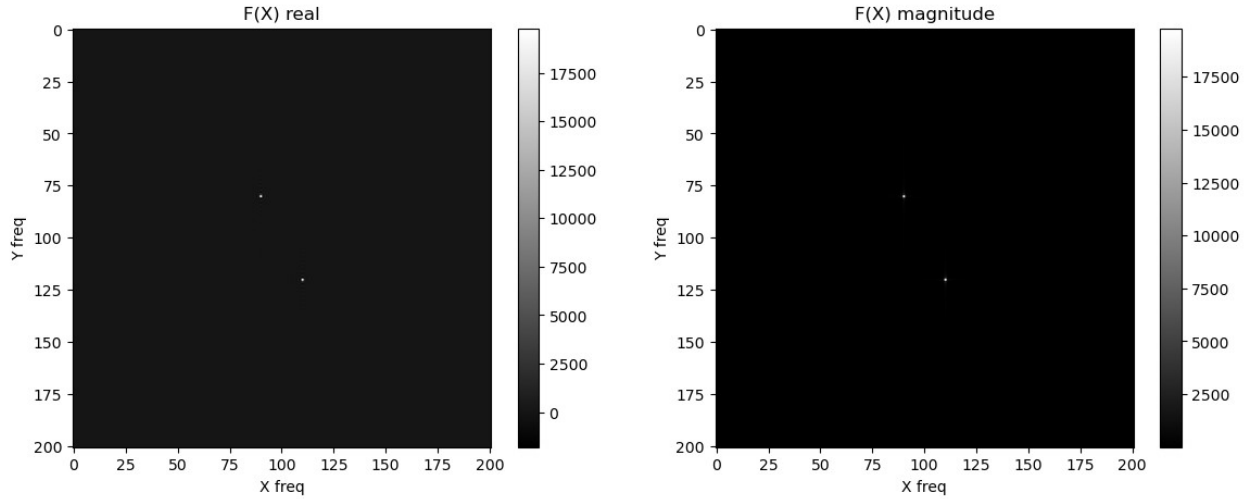

```
(201, 201)
```




```
F = np.cos(np.pi*X + 4*np.pi/2*Y)

fig, (ax1) = plt.subplots(1, 1, figsize=(14, 5))

x = ax1.imshow(F, cmap='gray')
fig.colorbar(x, ax=ax1)
ax1.set_title('F(X)')
ax1.set_xlabel('X')
ax1.set_ylabel('Y')
plt.show()

fftshifted = fftshift(fft2(ifftshift(F)))
print(fftshifted.shape)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))


x = ax1.imshow(np.real(fftshifted), cmap='gray')
fig.colorbar(x, ax=ax1)
ax1.set_title('F(X) real')
ax1.set_xlabel('X freq')
ax1.set_ylabel('Y freq')

magnitude = np.imag(fftshifted)**2 + np.real(fftshifted)**2
magnitude = np.sqrt(magnitude)

z = ax2.imshow(magnitude, cmap='gray')
fig.colorbar(z, ax=ax2)
ax2.set_title('F(X) magnitude')
ax2.set_xlabel('X freq')
ax2.set_ylabel('Y freq')

plt.show()
```




```
(201, 201)
```



In the image showing the function np.cos(np.pi*X + 2*np.pi *Y), the bands are narrower and angled less steeply compared to the plot of np.cos(np.pi*X + 4*np.pi*Y). This difference is due to the increase in frequency along the Y direction which goes from 2pi to 4pi means the wave oscillates more rapidly in Y. Since the slope of the diagonal wavefronts is determined by the ratio of the X

frequency to the Y frequency, increasing the Y frequency results in a shallower slope. In the Fourier transform plots, both functions produce two symmetric white dots, representing the positive and negative frequency components. The X positions of the dots are the same in both cases, because the X frequency remains constant at ±0.5 cycles/mm. However, the dots in the 4pi plot are further apart in the Y direction, reflecting the higher Y frequency. Doubling the Y frequency from 2pi to 4pi doubles the vertical separation between the peaks in the frequency domain.

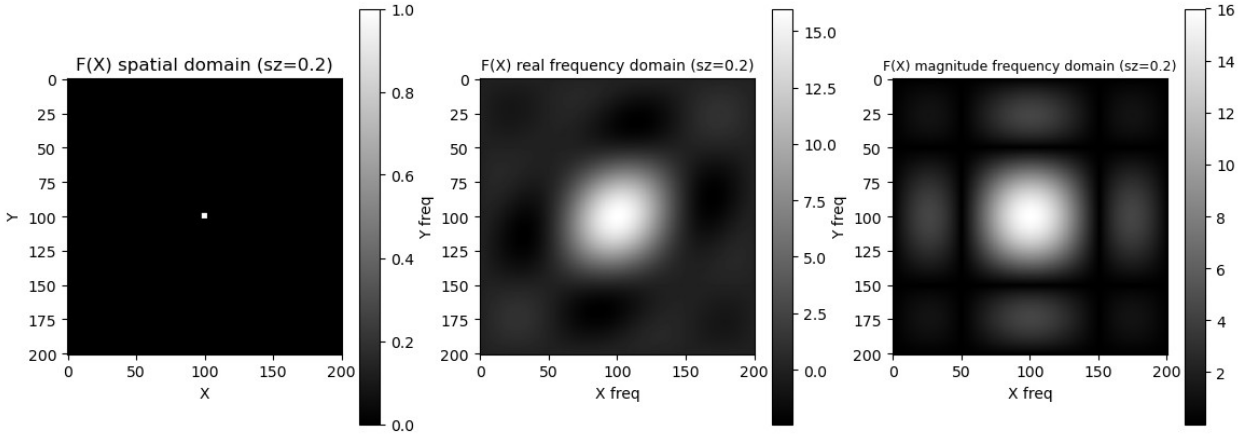
Question 4

Next consider: `(np.abs(X)<sz) * (np.abs(Y)<sz) for sz={0.2, 0.5, 1.0}`

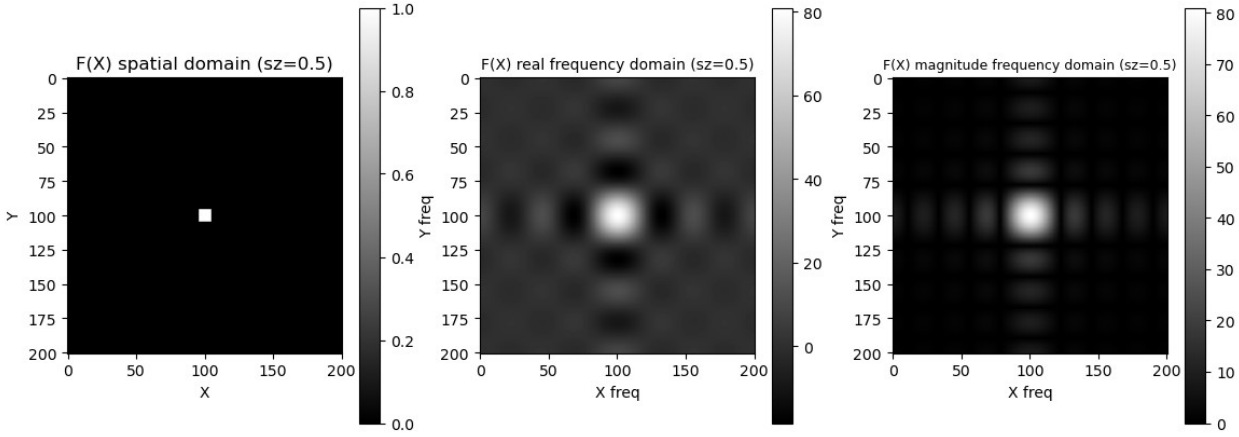
Describe the difference between these three functions.

```
# F = (np.abs(X)<sz) * (np.abs(Y)<sz) for sz={0.2, 0.5, 1.0}

for s in [0.2, 0.5, 1.0]:
    F = (np.abs(X)<s) * (np.abs(Y)<s)

    fftshifted = fftshift(fft2(ifftshift(F)))
    print(fftshifted.shape)

    fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(14, 5))

    x = ax1.imshow(F, cmap='gray')
    fig.colorbar(x, ax=ax1)
    ax1.set_title(f'F(X) spatial domain (sz={s})')
    ax1.set_xlabel('X')
    ax1.set_ylabel('Y')

    y = ax2.imshow(np.real(fftshifted), cmap='gray')
    fig.colorbar(y, ax=ax2)
    ax2.set_title(f'F(X) real frequency domain (sz={s})', fontsize=10)
    ax2.set_xlabel('X freq')
    ax2.set_ylabel('Y freq')

    magnitude = np.imag(fftshifted)**2 + np.real(fftshifted)**2
    magnitude = np.sqrt(magnitude)

    z = ax3.imshow(magnitude, cmap='gray')
    fig.colorbar(z, ax=ax3)
    ax3.set_title(f'F(X) magnitude frequency domain (sz={s})',
fontsize=9)
    ax3.set_xlabel('X freq')
    ax3.set_ylabel('Y freq')

    plt.show()
```

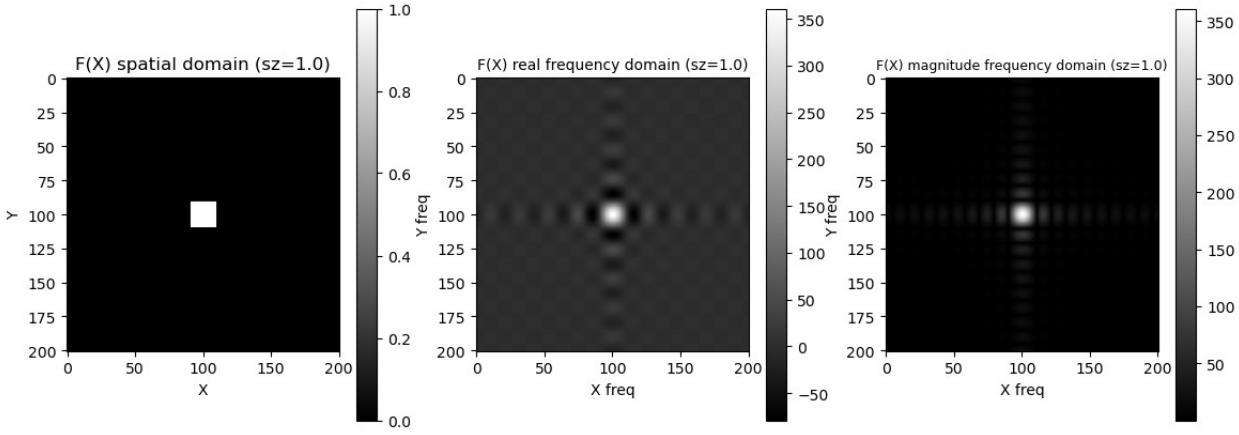
```
(201, 201)
```




```
(201, 201)
```




```
(201, 201)
```



The function is a square indicator function in the spatial domain, where the size of the square is controlled by sz. In the spatial domain, as sz increases, the size/area of the box increases, but the sharp resolution stays the same. The real frequency domain represents the real part of the Fourier transform of the spatial function. As sz increases in the real frequency domain, the spatial function extends, and the frequency content is more concentrated at the origin. Therefore, the square at the origin and the spread becomes smaller and clearer with less noise. The magnitude frequency domain represents the magnitude of the Fourier transform. As sz increases in the magnitude frequency domain, the square at the origin becomes smaller and clearer as the frequencies concentrate near the origin, creating a more compact and less noisy image.

Question 5

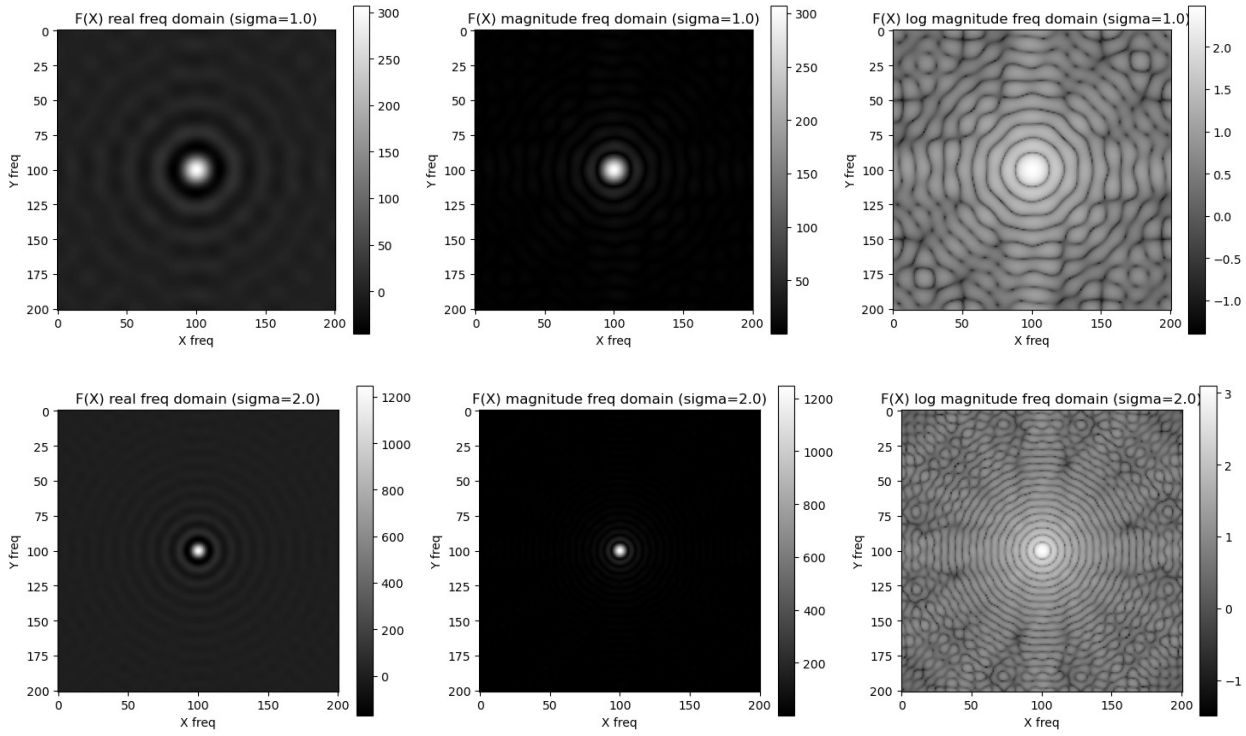
We are often interested in signal power in decibels which requires a logarithmic scaling of the frequency values. Show properly labeled image-domain and Fourier transform pairs (real, magnitude, and log10 magnitude) for the following functions:

`np.sqrt(X**2 + Y**2)<sz for sz={1.0, 2.0}`

Why might you prefer to look at log10 data?

```
for sz in [1.0, 2.0]:
    F = np.sqrt(X**2 + Y**2) < sz
    fshifted = fftshift(fft2(ifftshift(F)))

    fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(18, 5))

    x = ax1.imshow(np.real(fshifted), cmap='gray')
    fig.colorbar(x, ax=ax1)
    ax1.set_title(f'F(X) real freq domain (sigma={sz})')
    ax1.set_xlabel('X freq')
    ax1.set_ylabel('Y freq')

    magnitude = np.imag(fshifted)**2 + np.real(fshifted)**2
    magnitude = np.sqrt(magnitude)

    y = ax2.imshow(magnitude, cmap='gray')
    fig.colorbar(y, ax=ax2)
    ax2.set_title(f'F(X) magnitude freq domain (sigma={sz})')
    ax2.set_xlabel('X freq')
    ax2.set_ylabel('Y freq')

    logmag = np.log10(magnitude + 1e-10)
    z = ax3.imshow(logmag, cmap='gray')
    fig.colorbar(z, ax=ax3)
    ax3.set_title(f'F(X) log magnitude freq domain (sigma={sz})')
    ax3.set_xlabel('X freq')
```

```
    ax3.set_ylabel('Y freq')

    plt.show()
```

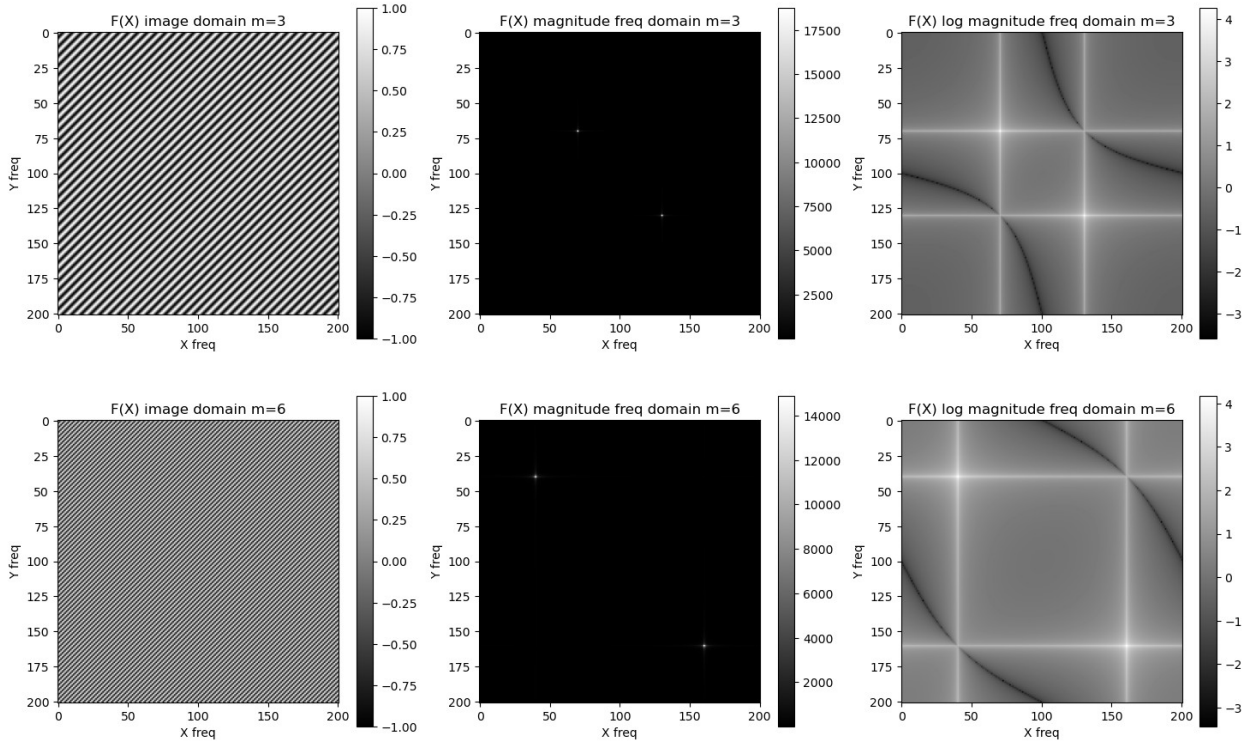


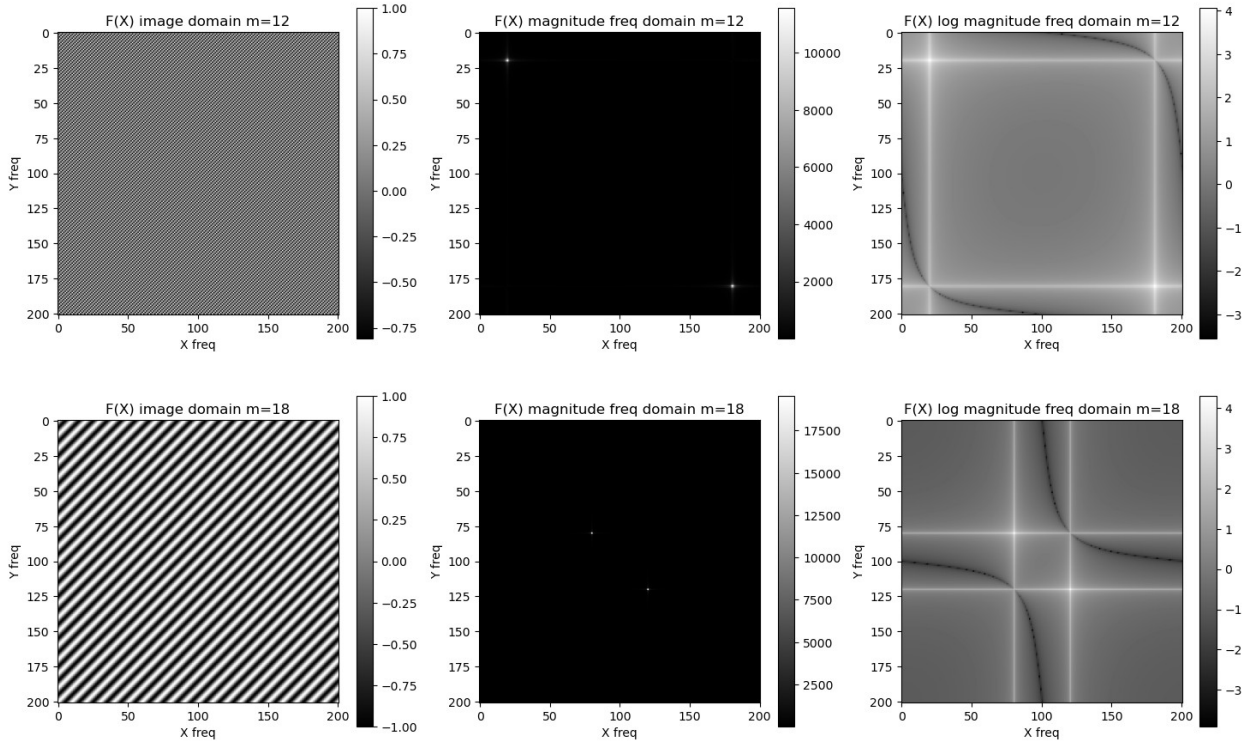
The log10 magnitude plot appears more patterned and has more even brightness because the log scaling compresses the wide range of values in the Fourier magnitude. It reduces the brightness of large values and enhances smaller ones, enhancing the contrast. In the real and magnitude frequency domain plots, a higher sigma= 2.0 results in tighter rings compared to sigma = 1.0. This is due to the inverse relationship between space and frequency. Increasing the size of the circle in the spatial domain compresses its transform in the frequency domain, pushing the rings closer together.

Question 6

```
for m in [3,6,12,18]:
    F = np.cos(np.pi*(X+Y)*m)
    fshifted = fftshift(fft2(ifftshift(F)))

    fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(18, 5))

    x = ax1.imshow(F, cmap='gray')
    fig.colorbar(x, ax=ax1)
    ax1.set_title(f'F(X) image domain m={m}')
    ax1.set_xlabel('X freq')
    ax1.set_ylabel('Y freq')
```

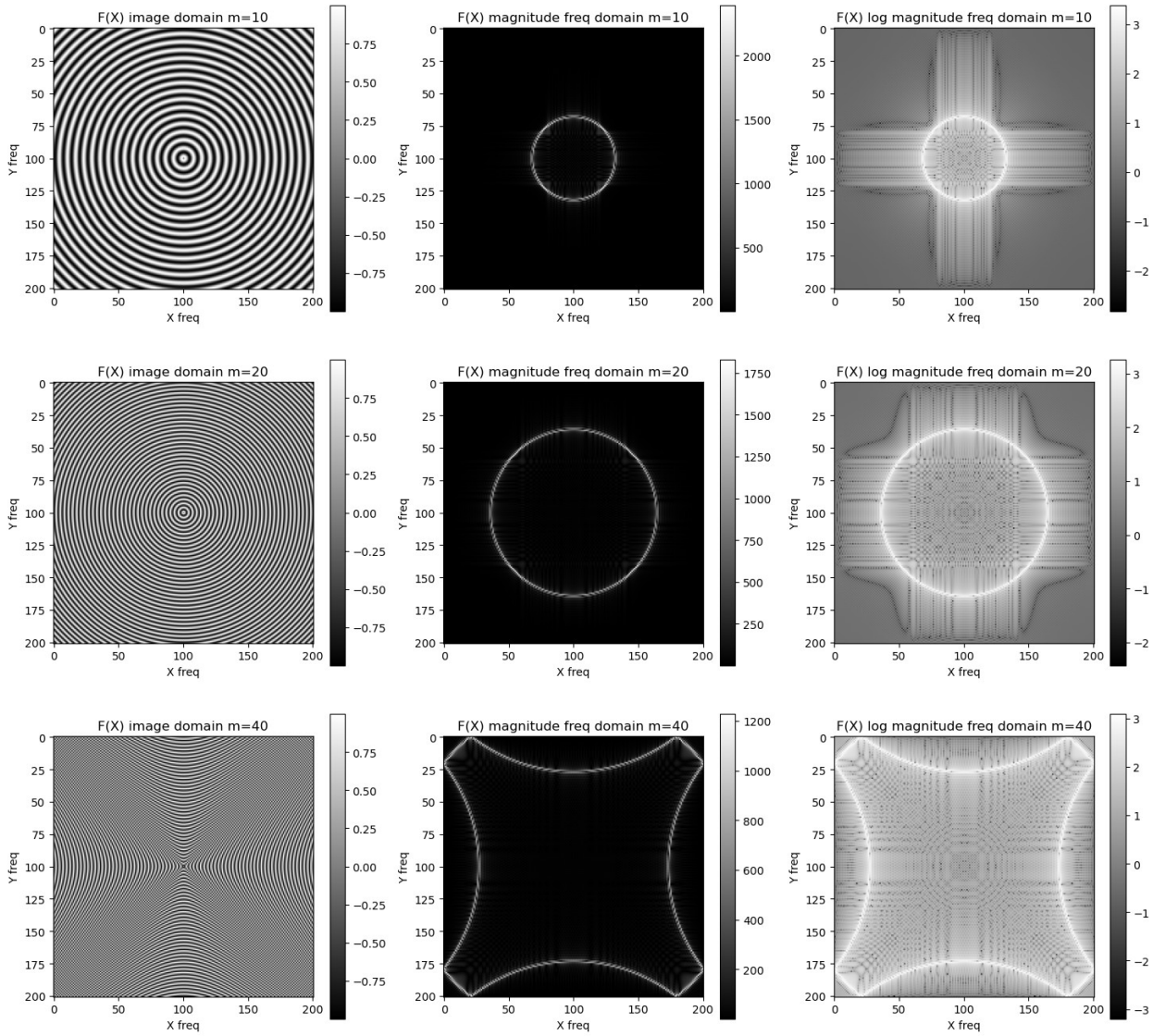
```
magnitude = np.imag(fshifted)**2 + np.real(fshifted)**2
magnitude = np.sqrt(magnitude)

y = ax2.imshow(magnitude, cmap='gray')
fig.colorbar(y, ax=ax2)
ax2.set_title(f'F(X) magnitude freq domain m={m}')
ax2.set_xlabel('X freq')
ax2.set_ylabel('Y freq')

logmag = np.log10(magnitude + 1e-10)
z = ax3.imshow(logmag, cmap='gray')
fig.colorbar(z, ax=ax3)
ax3.set_title(f'F(X) log magnitude freq domain m={m}')
ax3.set_xlabel('X freq')
ax3.set_ylabel('Y freq')

plt.show()
```






```
for m in [10, 20, 40]:
    F = np.sin(np.sqrt(X**2+Y**2)*m)
    fshifted = fftshift(fft2(ifftshift(F)))

    fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(18, 5))

    x = ax1.imshow(F, cmap='gray')
    fig.colorbar(x, ax=ax1)
    ax1.set_title(f'F(X) image domain m={m}')
    ax1.set_xlabel('X freq')
    ax1.set_ylabel('Y freq')

    magnitude = np.imag(fshifted)**2 + np.real(fshifted)**2
    magnitude = np.sqrt(magnitude)

    y = ax2.imshow(magnitude, cmap='gray')
    fig.colorbar(y, ax=ax2)
    ax2.set_title(f'F(X) magnitude freq domain m={m}')
    ax2.set_xlabel('X freq')
    ax2.set_ylabel('Y freq')

    logmag = np.log10(magnitude + 1e-10)
    z = ax3.imshow(logmag, cmap='gray')
    fig.colorbar(z, ax=ax3)
    ax3.set_title(f'F(X) log magnitude freq domain m={m}')
    ax3.set_xlabel('X freq')
    ax3.set_ylabel('Y freq')
```

```
plt.show()
```



For function 1 (np.cos(np.pi*(X+Y)*m)), the spatial domain shows diagonal waves and as m increases, the waves get closer together, indicating higher spatial frequency due to a higher frequency parameter m making the cosine function oscilate faster. For function 2 np.sin(np.sqrt(X**2+Y**2)*m), the spatial domain is of concentric circles in a wave pattern that eventually reaches high enough frequency that they become hyperbola. We see in the frequency domain that there is a bent square at the high frequency region of the graph. As you increase the frequency parameter m, the spatial pattern gets finer (more detail), and the its Fourier transform shows energy at higher frequencies, farther from the center. If your sampling rate is too low, you would miss or misrepresent these high-frequency components due to aliasing.