Properties and Applications of 2D Fourier Transform

March 7, 2019

1 Properties and Applications of 2D Fourier Transform

Kenneth V. Domingo
2015-03116
Physics 166 THV - Dr. Maricor Soriano
2nd Semester, A.Y. 2018-19
A live version of this notebook can be accessed on GitHub:
https://github.com/kvdomingo/Physics-166/blob/master/Properties%20and%20Applications%20of%202D

In [1]:
```
import numpy as np
import matplotlib.pyplot as mp
import scipy.fftpack as fft
import scipy.ndimage as img
import scipy.signal as sig
import numpy.random as rd
from PIL import Image, ImageDraw, ImageFont
from jupyterthemes import jtplot
```

In [2]:
```
jtplot.reset()
mp.rcParams["figure.figsize"] = (5, 5)
mp.rcParams["figure.dpi"] = 100
mp.rcParams["text.usetex"] = True
mp.rcParams["font.family"] = "serif"
rd.seed(314)
```

In [3]:
```
def uint8(X):
    """
    Convert input data type to 8-bit unsigned integer
    """
    return np.round(abs(X)/abs(X).max() * (2**8 - 1)).astype("uint8")

def fftcircle(Z, r, h, k, **kwargs):
    """
    Draw a circle on an existing image

    Parameters
    ----------
```

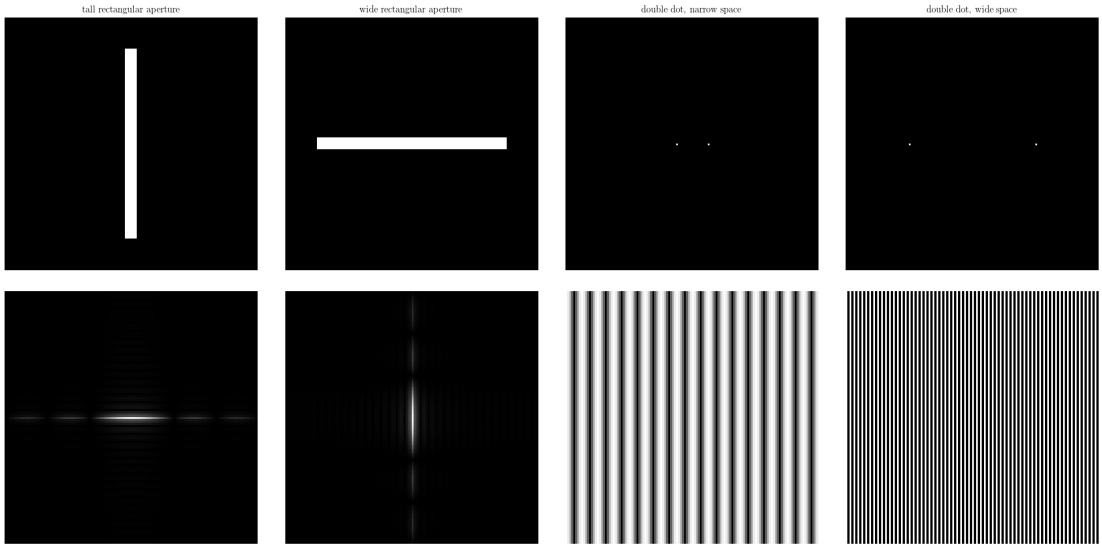
```
    Z : PIL.Image
        An instance of a blank or pre-existing PIL.Image object
    r : int
        radius of circle
    h : int
        x-location of center of circle
    k : int
        y-location of center of circle
    """
    draw = ImageDraw.Draw(Z)
    draw.ellipse((h-r, k-r, h+r, k+r), **kwargs)

def fftsquare(Z, r, h, k, **kwargs):
    """
    Draw a square on an existing image

    Parameters
    ----------
    Z : PIL.Image
        An instance of a blank or pre-existing PIL.Image object
    r : int
        half-length of a side of the square
    h : int
        x-location of center of square
    k : int
        y-location of center of square
    """
    draw = ImageDraw.Draw(Z)
    draw.rectangle((h-r, k-r, h+r, k+r), **kwargs)

def fftgauss(X, Y, mux, muy, sigma):
    """
    Generate a 2D Gaussian

    Parameters
    ----------
    X : array_like
        meshgrid of x values
    Y : array_like
        meeshgrid of y values
    mux : float
        x-location of mean
    muy : float
        y-location of mean
    sigma : float
        standard deviation of the gaussian
    """
    return np.exp(-((X - mux/len(X))**2 + (Y - muy/len(Y))**2)**2/sigma**2)
```

1.1 Activity 1. Anamorphic property of FT of different 2D patterns

In [4]:
```
N = 128
fig = mp.figure(figsize=(5*4, 5*2))
r = 0

rtall = np.zeros((N, N), "uint8")
rtall[2*N//16:14*N//16 ,N//2-3:N//2+3] = 1
ax = fig.add_subplot(241)
ax.imshow(rtall, "gray")
ax.axis("off")
ax.set_title("tall rectangular aperture")

rwide = rtall.T
ax = fig.add_subplot(242)
ax.imshow(rwide, "gray")
ax.axis("off")
ax.set_title("wide rectangular aperture")

dots = Image.new("L", (N, N), color="black")
draw = ImageDraw.Draw(dots)
draw.ellipse((7*N//16-r, N//2-r, 7*N//16+r, N//2+r), fill="white")
draw.ellipse((9*N//16-r, N//2-r, 9*N//16+r, N//2+r), fill="white")
dots = np.array(dots, "uint8")
ax = fig.add_subplot(243)
ax.imshow(dots, "gray")
ax.axis("off")
ax.set_title("double dot, narrow space")

dotspace = Image.new("L", (N, N), color="black")
draw = ImageDraw.Draw(dotspace)
draw.ellipse((N//4-r, N//2-r, N//4+r, N//2+r), fill="white")
draw.ellipse((3*N//4-r, N//2-r, 3*N//4+r, N//2+r), fill="white")
dotspace = np.array(dotspace, "uint8")
ax = fig.add_subplot(244)
ax.imshow(dotspace, "gray")
ax.axis("off")
ax.set_title("double dot, wide space")

Ftall = fft.fft2(rtall)
ax = fig.add_subplot(245)
ax.imshow(fft.fftshift(abs(Ftall)), "gray")
ax.axis("off")

Fwide = fft.fft2(rwide)
ax = fig.add_subplot(246)
ax.imshow(fft.fftshift(abs(Fwide)), "gray")
ax.axis("off")
```

```
Fdots = fft.fft2(dots)
ax = fig.add_subplot(247)
ax.imshow(fft.fftshift(abs(Fdots)), "gray")
ax.axis("off")

Fspace = fft.fft2(dotspace)
ax = fig.add_subplot(248)
ax.imshow(fft.fftshift(abs(Fspace)), "gray")
ax.axis("off")

mp.tight_layout()
mp.show()
```

```
D:\ProgramData\Anaconda3\envs\compsense\lib\site-packages\scipy\fftpack\basic.py:159: FutureWa
  z[index] = x
```

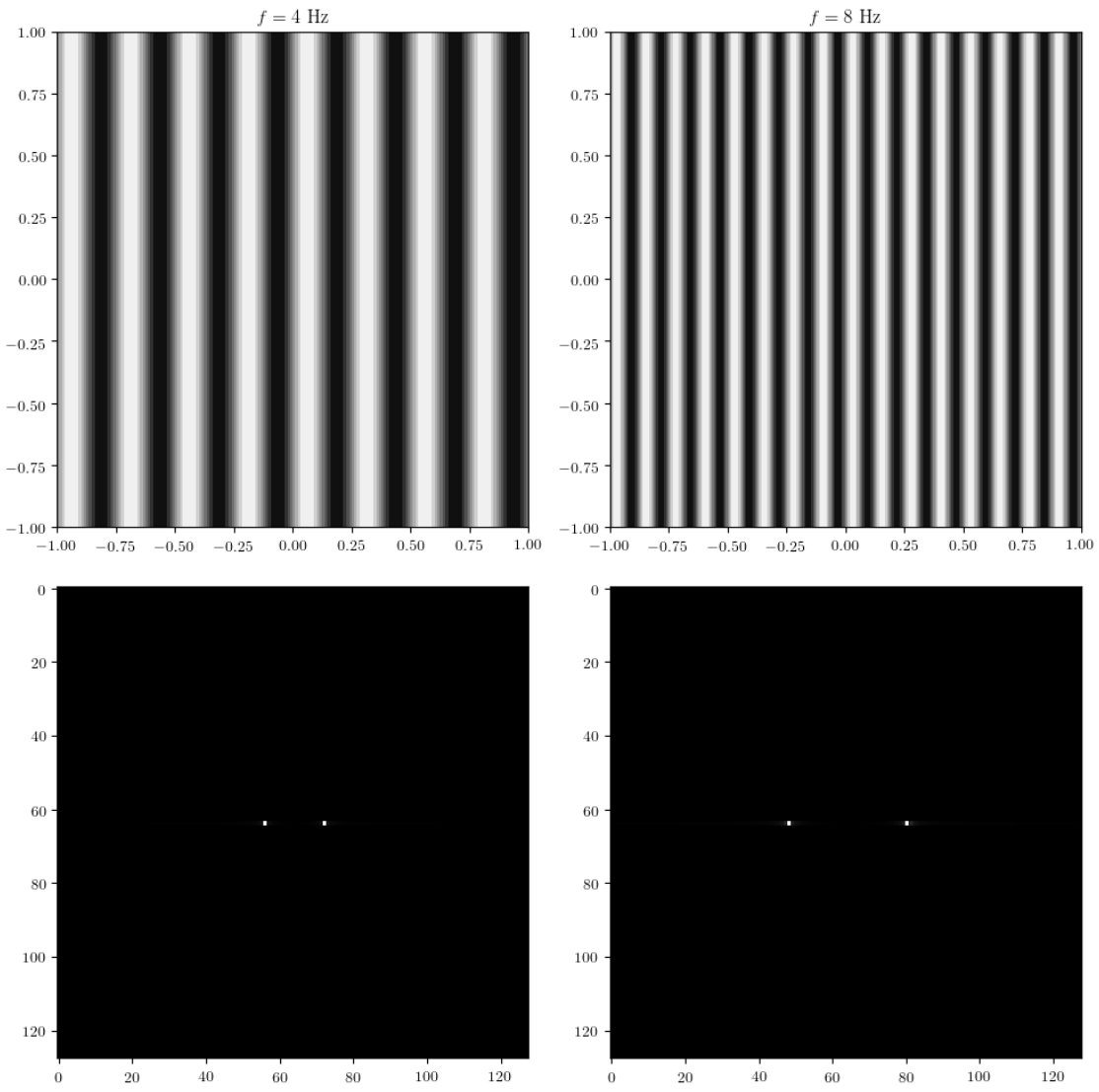


Notice that the FT of dots produces a sine wave. Pulling the dots apart produces a sine with a higher frequency. This is analogous to the FT of a temporal signal, where a pure sinusoid shows up as a two peaks in the Fourier domain, corresponding to the actual (positive) and aliased (negative) frequencies of the sinusoid. The FT of a slit can be thought of as the product of the FT of stretched dot and a line.

1.2 Activity 2. Rotation property of the FFT

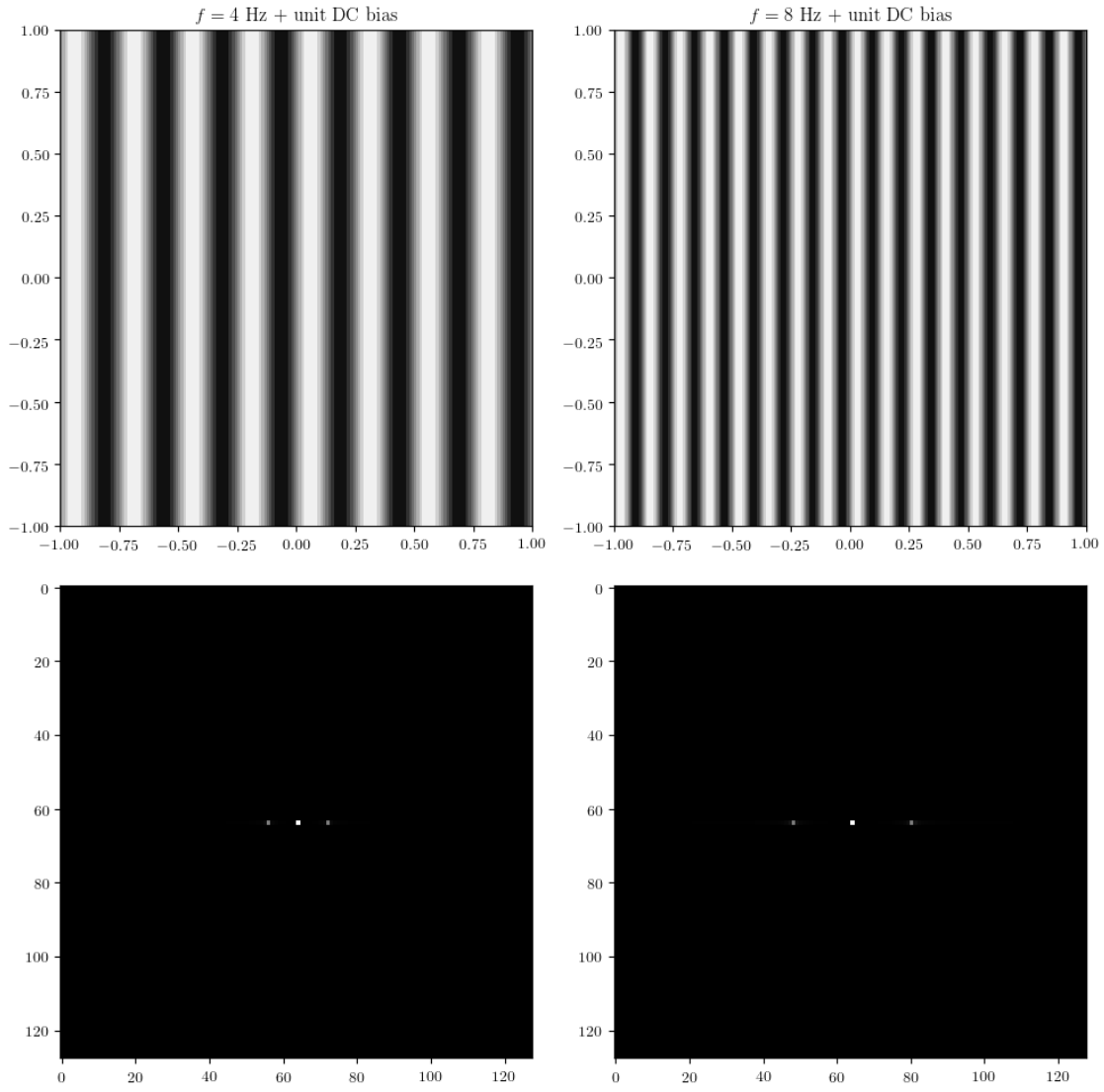
```
In [5]: N = 128
        t = np.linspace(-1, 1, N)
        X,Y = np.meshgrid(t,t)
```

```
fig = mp.figure(figsize=(5*2, 5*2))

f = 4
w = 2*np.pi*f
Z4 = np.sin(w*X)
ax = fig.add_subplot(221)
ax.contourf(X, Y, Z4, cmap="gray")
ax.set_title(r"$f = %i$ Hz"%f)

f = 8
w = 2*np.pi*f
Z8 = np.sin(w*X)
ax = fig.add_subplot(222)
ax.contourf(X, Y, Z8, cmap="gray")
ax.set_title(r"$f = %i$ Hz"%f)

FZ4 = fft.fft2(Z4)
ax = fig.add_subplot(223)
ax.imshow(fft.fftshift(abs(FZ4)), cmap="gray")

FZ8 = fft.fft2(Z8)
ax = fig.add_subplot(224)
ax.imshow(fft.fftshift(abs(FZ8)), cmap="gray")

mp.tight_layout()
mp.show()
```




```
In [6]: N = 128
        t = np.linspace(-1, 1, N)
        X,Y = np.meshgrid(t,t)

        fig = mp.figure(figsize=(5*2, 5*2))

        f = 4
        w = 2*np.pi*f
        Z4 = np.sin(w*X) + 1
        ax = fig.add_subplot(221)
        ax.contourf(X, Y, Z4, cmap="gray")
        ax.set_title(r"$f = %i$ Hz + unit DC bias"%f)
```

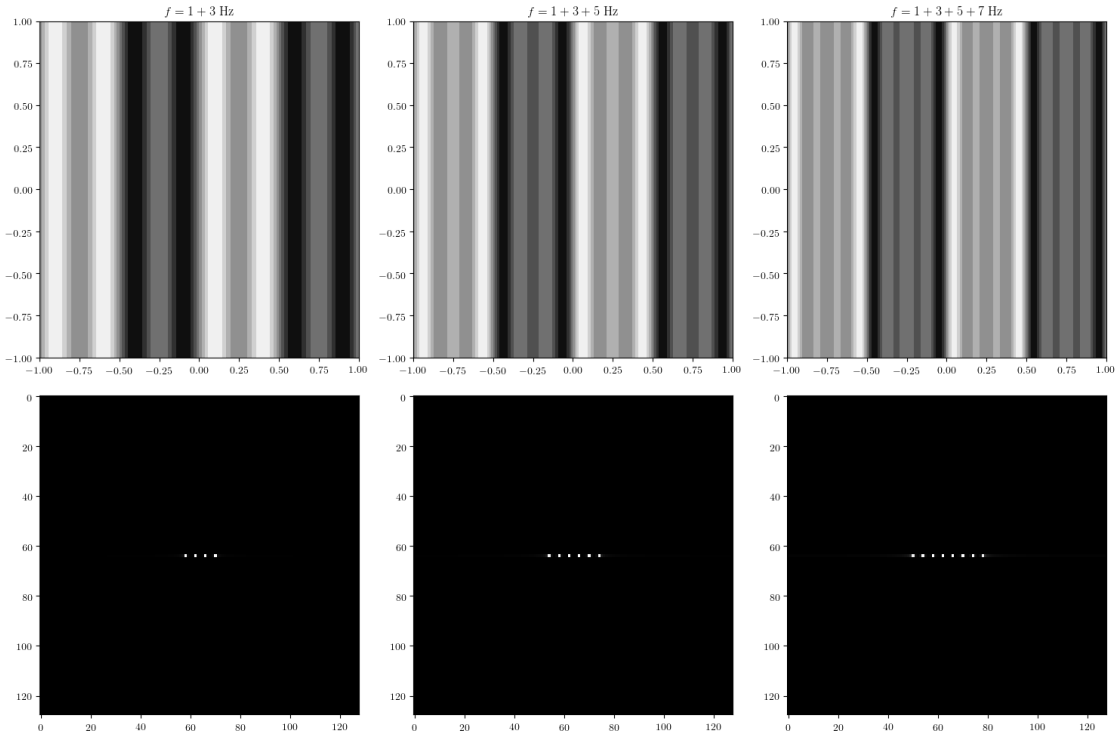
```
f = 8
w = 2*np.pi*f
Z8 = np.sin(w*X) + 1
ax = fig.add_subplot(222)
ax.contourf(X, Y, Z8, cmap="gray")
ax.set_title(r"$f = %i$ Hz + unit DC bias"%f)

FZ4 = fft.fft2(Z4)
ax = fig.add_subplot(223)
ax.imshow(fft.fftshift(abs(FZ4)), cmap="gray")

FZ8 = fft.fft2(Z8)
ax = fig.add_subplot(224)
ax.imshow(fft.fftshift(abs(FZ8)), cmap="gray")

mp.tight_layout()
mp.show()
```



A DC bias shows up as a zero-frequency component in the Fourier domain.

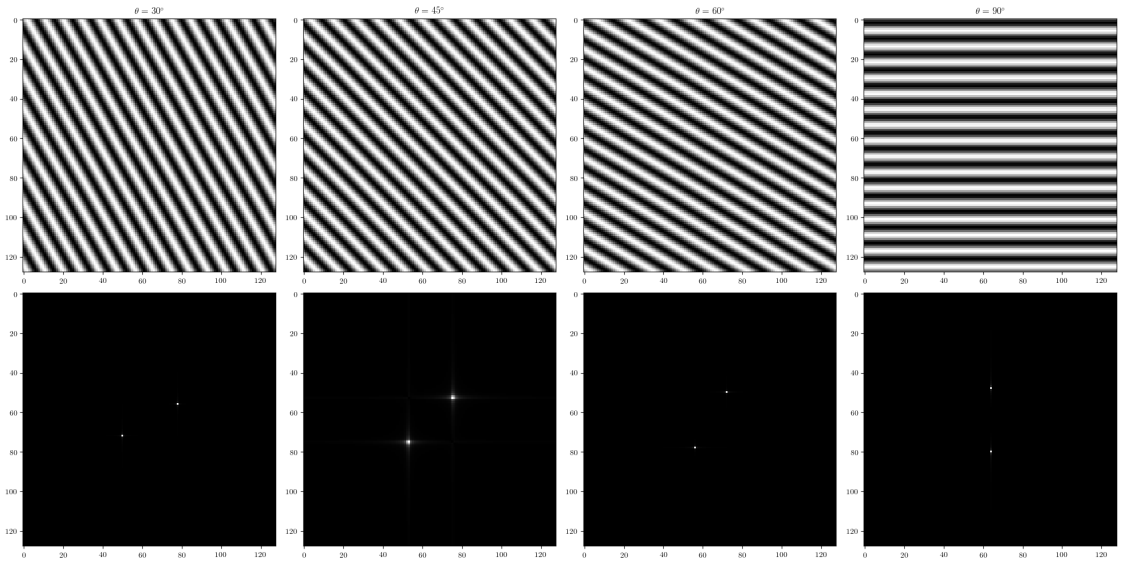
```
In [7]: N = 128
        t = np.linspace(-1, 1, N)
        X,Y = np.meshgrid(t,t)

        fig = mp.figure(figsize=(5*3, 5*2))

        Z13 = np.sin(2*np.pi*1*X) + np.sin(2*np.pi*3*X)
        ax = fig.add_subplot(231)
        ax.contourf(X, Y, Z13, cmap="gray")
        ax.set_title(r"$f = 1 + 3$ Hz")

        Z135 = np.sin(2*np.pi*1*X) + np.sin(2*np.pi*3*X) + np.sin(2*np.pi*5*X)
```

```
ax = fig.add_subplot(232)
ax.contourf(X, Y, Z135, cmap="gray")
ax.set_title(r"$f = 1 + 3 + 5$ Hz")

Z1357 = np.sin(2*np.pi*1*X) + np.sin(2*np.pi*3*X) + np.sin(2*np.pi*5*X) \
        + np.sin(2*np.pi*7*X)
ax = fig.add_subplot(233)
ax.contourf(X, Y, Z1357, cmap="gray")
ax.set_title(r"$f = 1 + 3 + 5 + 7$ Hz")

FZ13 = fft.fft2(Z13)
ax = fig.add_subplot(234)
ax.imshow(fft.fftshift(abs(FZ13)), cmap="gray")

FZ135 = fft.fft2(Z135)
ax = fig.add_subplot(235)
ax.imshow(fft.fftshift(abs(FZ135)), cmap="gray")

FZ1357 = fft.fft2(Z1357)
ax = fig.add_subplot(236)
ax.imshow(fft.fftshift(abs(FZ1357)), cmap="gray")

mp.tight_layout()
mp.show()
```



A summation of sines in the spatial domain can all be shown deconstructed in the Fourier domain.

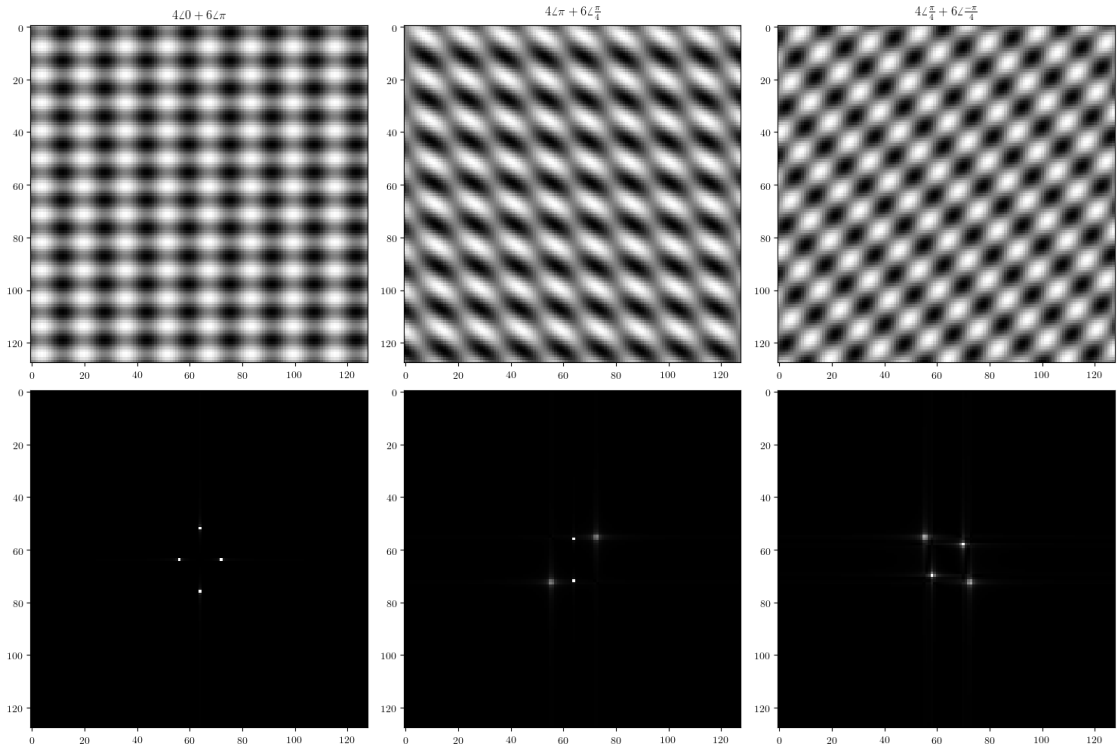
```
In [8]: fig = mp.figure(figsize=(5*4, 5*2))
        b = np.array([X, Y])
        b = np.transpose(b, axes=(1,0,2))
        thetalist = [30, 45, 60, 90]

        for i in range(4):
            ax = fig.add_subplot(2, 4, i+1)
            theta = np.radians(thetalist[i])
            R = np.array([[np.cos(theta), -np.sin(theta)],
                          [np.sin(theta), np.cos(theta)]])
            Xp, Yp = R.dot(b)
            Z8prime = np.sin(2*np.pi*8*Xp)
            ax.imshow(Z8prime, "gray")
            ax.set_title(r"$\theta = %s^\circ$"%(thetalist[i]))

            ax = fig.add_subplot(2, 4, i+5)
            FZ = fft.fft2(Z8prime)
            ax.imshow(fft.fftshift(abs(FZ)), "gray")

        mp.tight_layout()
        mp.show()
```



As the sine rotates, its FT also rotates with it, which shows that a diagonal sine is nothing but a summation of purely x-component and purely y-component sines of varying frequencies.

```
In [9]: fig = mp.figure(figsize=(5*3, 5*2))
```

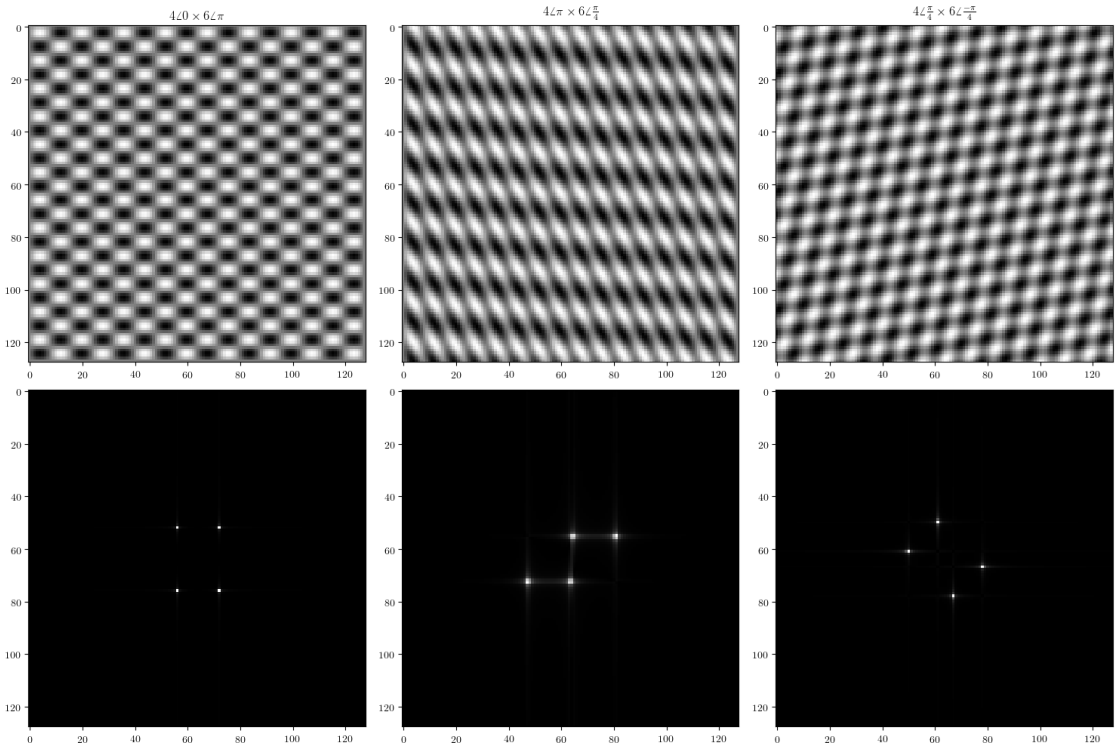
```
ax = fig.add_subplot(231)
theta = np.radians(90)
R = np.array([[np.cos(theta), -np.sin(theta)],
              [np.sin(theta), np.cos(theta)]])
Xp, Yp = R.dot(b)
Z = np.sin(2*np.pi*4*X) + np.sin(2*np.pi*6*Xp)
ax.imshow(Z, "gray")
ax.set_title(r"$4\angle 0 + 6\angle \pi$")

ax = fig.add_subplot(234)
FZ = fft.fft2(Z)
ax.imshow(fft.fftshift(abs(FZ)), "gray")

ax = fig.add_subplot(232)
theta = np.radians(45)
R = np.array([[np.cos(theta), -np.sin(theta)],
              [np.sin(theta), np.cos(theta)]])
Xp, Yp = R.dot(b)
Z = np.sin(2*np.pi*4*Y) + np.sin(2*np.pi*6*Xp)
ax.imshow(Z, "gray")
ax.set_title(r"$4\angle\pi + 6\angle\frac{\pi}{4}$")

ax = fig.add_subplot(235)
FZ = fft.fft2(Z)
ax.imshow(fft.fftshift(abs(FZ)), "gray")

ax = fig.add_subplot(233)
theta = np.radians(45)
R = np.array([[np.cos(theta), -np.sin(theta)],
              [np.sin(theta), np.cos(theta)]])
Xp, Yp = R.dot(b)
Z = np.sin(2*np.pi*4*Xp)
theta = np.radians(-45)
R = np.array([[np.cos(theta), -np.sin(theta)],
              [np.sin(theta), np.cos(theta)]])
Xp, Yp = R.dot(b)
Z += np.sin(2*np.pi*6*Xp)
ax.imshow(Z, "gray")
ax.set_title(r"$4\angle\frac{\pi}{4} + 6\angle\frac{-\pi}{4}$")

ax = fig.add_subplot(236)
FZ = fft.fft2(Z)
ax.imshow(fft.fftshift(abs(FZ)), "gray")

mp.tight_layout()
mp.show()
```



As stated before, adding sines in the spatial domain shows up deconstructed into their respective frequencies in the Fourier domain.

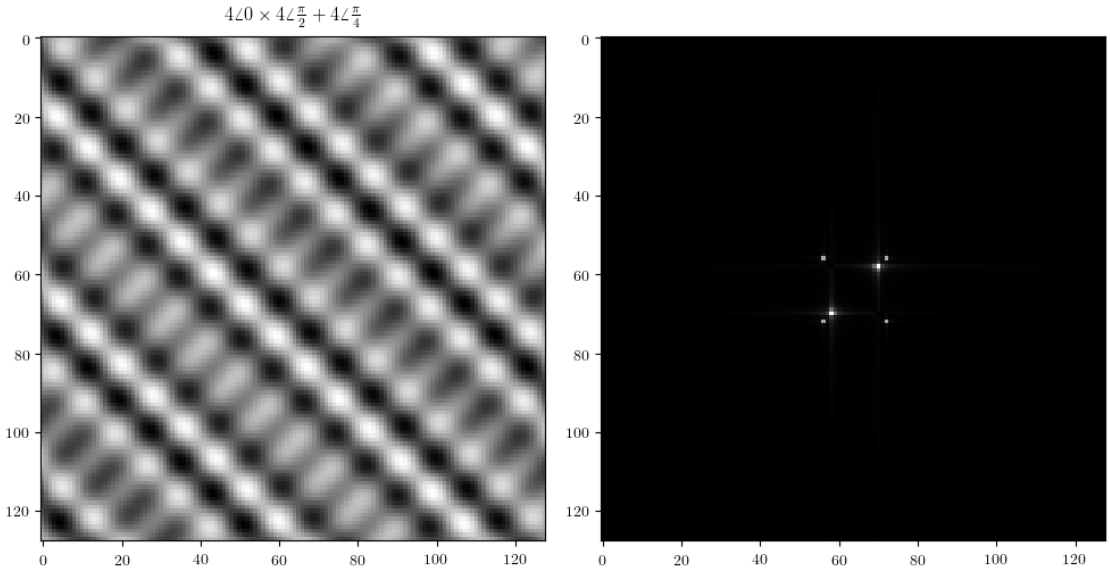
```
In [10]: fig = mp.figure(figsize=(5*3, 5*2))

         ax = fig.add_subplot(231)
         Z = np.sin(2*np.pi*4*X) * np.sin(2*np.pi*6*Y)
         ax.imshow(Z, "gray")
         ax.set_title(r"$4\angle 0 \times 6\angle \pi$")

         ax = fig.add_subplot(234)
         FZ = fft.fft2(Z)
         ax.imshow(fft.fftshift(abs(FZ)), "gray")

         ax = fig.add_subplot(232)
         theta = np.radians(45)
         R = np.array([[np.cos(theta), -np.sin(theta)],
                       [np.sin(theta), np.cos(theta)]])
         Xp, Yp = R.dot(b)
         Z = np.sin(2*np.pi*4*X) * np.sin(2*np.pi*6*Xp)
         ax.imshow(Z, "gray")
         ax.set_title(r"$4\angle\pi \times 6\angle\frac{\pi}{4}$")

         ax = fig.add_subplot(235)
```

```
FZ = fft.fft2(Z)
ax.imshow(fft.fftshift(abs(FZ)), "gray")

ax = fig.add_subplot(233)
theta = np.radians(45)
R = np.array([[np.cos(theta), -np.sin(theta)],
              [np.sin(theta), np.cos(theta)]])
Xp, Yp = R.dot(b)
Z = np.sin(2*np.pi*4*Xp)
theta = np.radians(-45)
R = np.array([[np.cos(theta), -np.sin(theta)],
              [np.sin(theta), np.cos(theta)]])
Xp, Yp = R.dot(b)
Z *= np.sin(2*np.pi*6*Xp)
ax.imshow(Z, "gray")
ax.set_title(r"$4\angle\frac{\pi}{4} \times 6\angle\frac{-\pi}{4}$")

ax = fig.add_subplot(236)
FZ = fft.fft2(Z)
ax.imshow(fft.fftshift(abs(FZ)), "gray")

mp.tight_layout()
mp.show()
```



A product of sines can be thought of as sums of sines and cosines, the latter of which can also be represented as sines with a certain phase shift. Thus, we can also deconstruct their respective frequencies in the Fourier domain.

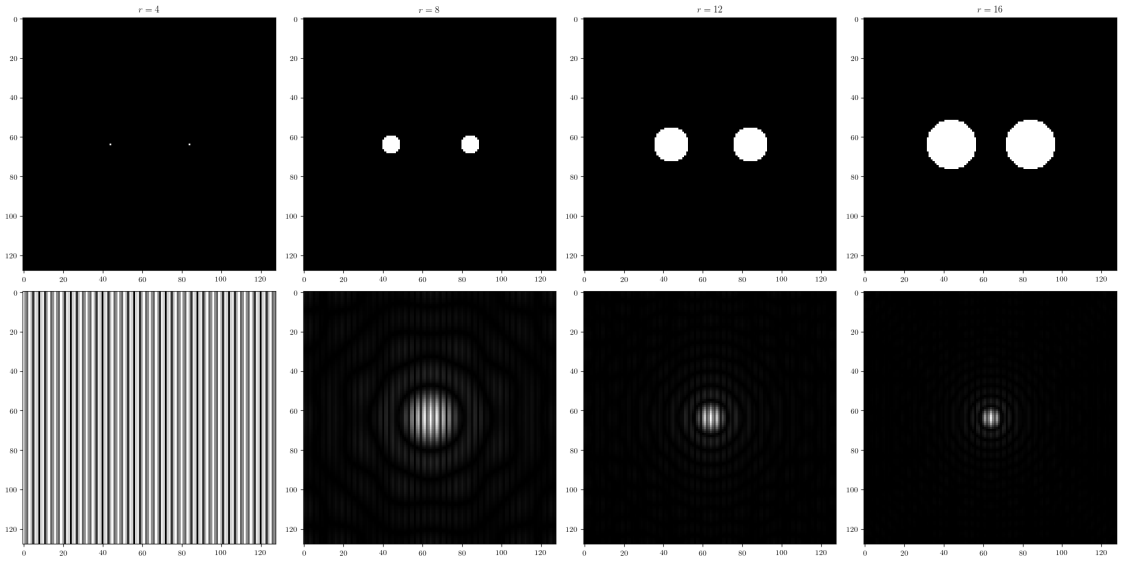
```
In [11]: fig = mp.figure(figsize=(5*2, 5))

         ax = fig.add_subplot(121)
         Z = np.sin(2*np.pi*4*X) * np.sin((2*np.pi*4*Y))
         theta = np.radians(45)
         R = np.array([[np.cos(theta), -np.sin(theta)],
                       [np.sin(theta), np.cos(theta)]])
         Xp, Yp = R.dot(b)
         Z += np.sin(2*np.pi*4*Xp)
         ax.imshow(Z, "gray")
         ax.set_title(r"$4\angle 0 \times 4\angle \
                     \frac{\pi}{2} + 4\angle\frac{\pi}{4}$")

         ax = fig.add_subplot(122)
         FZ = fft.fft2(Z)
         ax.imshow(fft.fftshift(abs(FZ)), "gray")

         mp.tight_layout()
         mp.show()
```



1.3 Activity 3. Convolution Theorem Redux

```
In [12]: N = 128
         r = 4
```

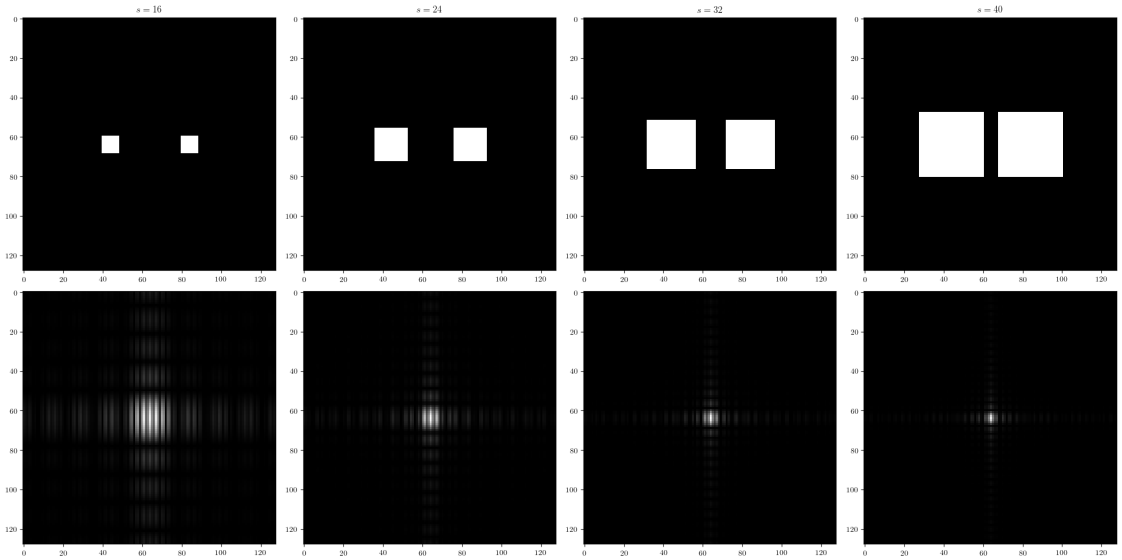
```
fig = mp.figure(figsize=(5*4, 5*2))

for i in range(4):
    Z = Image.new("L", (N, N), color="black")
    fftcircle(Z, r*i, N//2-20, N//2, fill="white")
    fftcircle(Z, r*i, N//2+20, N//2, fill="white")
    Z = np.array(Z, "uint8")

    ax = fig.add_subplot(2, 4, i+1)
    ax.imshow(Z, "gray")
    ax.set_title(r"$r = %i$"%(r*(i+1)))

    ax = fig.add_subplot(2, 4, i+5)
    FZ = fft.fft2(Z)
    ax.imshow(fft.fftshift(abs(FZ)), "gray")

mp.tight_layout()
mp.show()
```



As before, the FT of dots is a sine of corresponding frequency, and the FT of a double slit is like an enveloped sine. We can think of two circles as double slits and as usual, the FT pattern takes on the shape of the aperture or slit.

In [13]:
```
N = 128
r = 4
fig = mp.figure(figsize=(5*4, 5*2))

for i in range(1, 5):
    Z = Image.new("L", (N, N), color="black")
    fftsquare(Z, r*i, N//2-20, N//2, fill="white")
```

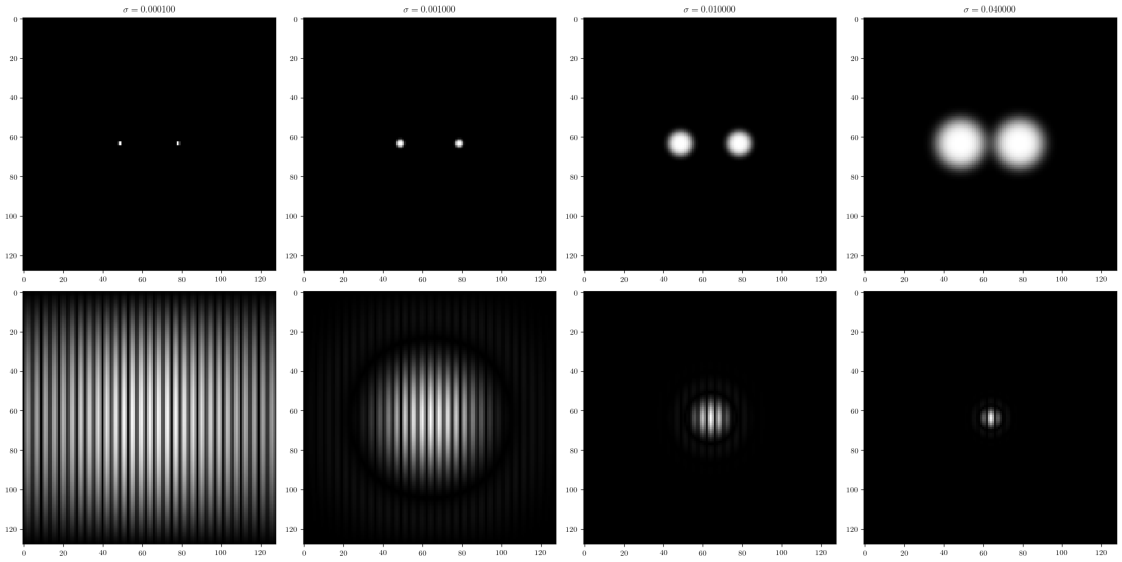
```
        fftsquare(Z, r*i, N//2+20, N//2, fill="white")
        Z = np.array(Z, "uint8")

        ax = fig.add_subplot(2, 4, i)
        ax.imshow(Z, "gray")
        ax.set_title(r"$s = %i$"%(2*r*(i+1)))

        ax = fig.add_subplot(2, 4, i+4)
        FZ = fft.fft2(Z)
        ax.imshow(fft.fftshift(abs(FZ)), "gray")

    mp.tight_layout()
    mp.show()
```



The FT pattern of a double square slit is similar to that of the rectangular double slit and again, the pattern takes on the shape of the slits.

```
In [14]: N = 128
         r = 4
         fig = mp.figure(figsize=(5*4, 5*2))
         sd = [0.0001, 0.001, 0.01, 0.04]

         for i in range(4):
             Z = fftgauss(X, Y, 30, 0, sd[i])
             Z += fftgauss(X, Y, -30, 0, sd[i])

             ax = fig.add_subplot(2, 4, i+1)
             ax.imshow(Z, "gray")
             ax.set_title(r"$\sigma = {0:f}$".format(sd[i]))
```

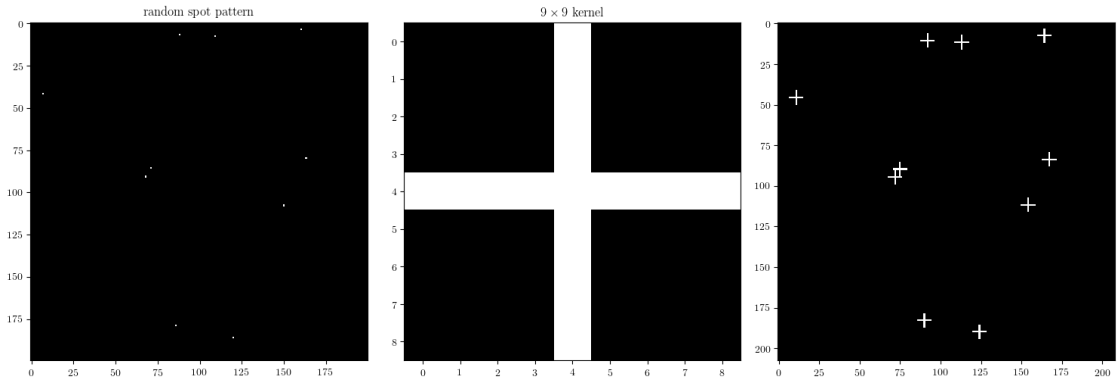
```
        ax = fig.add_subplot(2, 4, i+5)
        FZ = fft.fft2(Z)
        ax.imshow(fft.fftshift(abs(FZ)), "gray")

    mp.tight_layout()
    mp.show()
```



Previously, we saw that the FT of a gaussian is also a gaussian which grows inversely compared to the gaussian aperture in the spatial domain. Because it is now a double pattern, we see the corresponding sine pattern enveloped by the gaussian.

```
In [15]: N = 200
         A = np.zeros((N, N), "uint8")
         d = np.zeros((9,9), "uint8")
         d[9//2] = 1
         d.T[9//2] = 1
         for i in range(10):
             x,y = rd.randint(0, N, 2)
             A[x,y] = 1

         fig = mp.figure(figsize=(5*3, 5))

         ax = fig.add_subplot(131)
         ax.imshow(A, "gray")
         ax.set_title("random spot pattern")

         ax = fig.add_subplot(132)
         ax.imshow(d, "gray")
         ax.set_title(r"$9\times 9$ kernel")
```

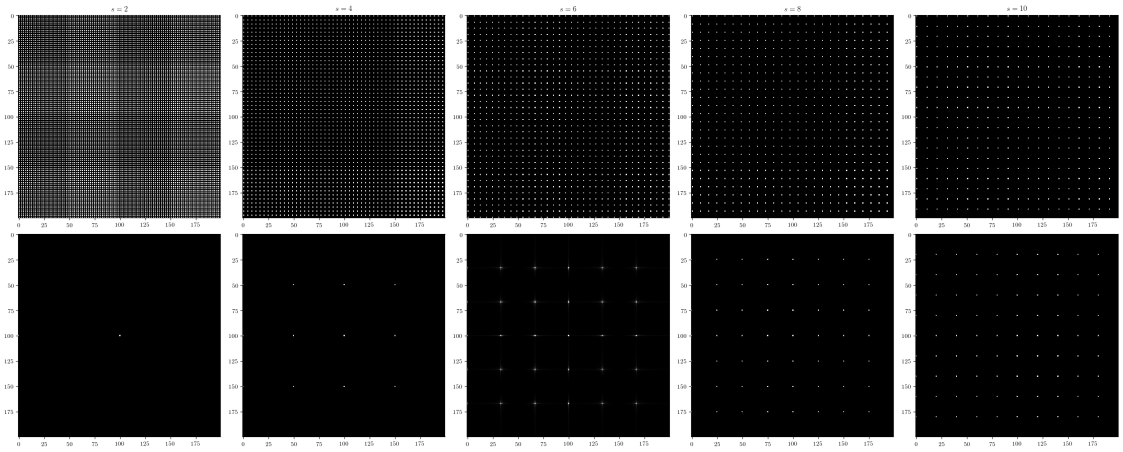
```
    ax = fig.add_subplot(133)
    y = sig.convolve2d(A, d, "full")
    ax.imshow(y, "gray")

    mp.tight_layout()
    mp.show()
```



We see that convolving a dot pattern with a cross-shaped kernel produces an image where the kernel appears wherever the dots are.

```
In [16]: N = 200
         fig = mp.figure(figsize=(5*5, 5*2))

         for i in range(5):
             Z = np.zeros((N, N), "uint8")
             Z[1::(i+1)*2, 1::(i+1)*2] = 1
             ax = fig.add_subplot(2, 5, i+1)
             ax.imshow(Z, "gray")
             ax.set_title(r"$s = %i$"%((i+1)*2))

             FZ = fft.fft2(Z)
             ax = fig.add_subplot(2, 5, i+6)
             ax.imshow(fft.fftshift(abs(FZ)), "gray")

         mp.tight_layout()
         mp.show()
```

```
D:\ProgramData\Anaconda3\envs\compsense\lib\site-packages\scipy\fftpack\basic.py:159: FutureWa
  z[index] = x
```



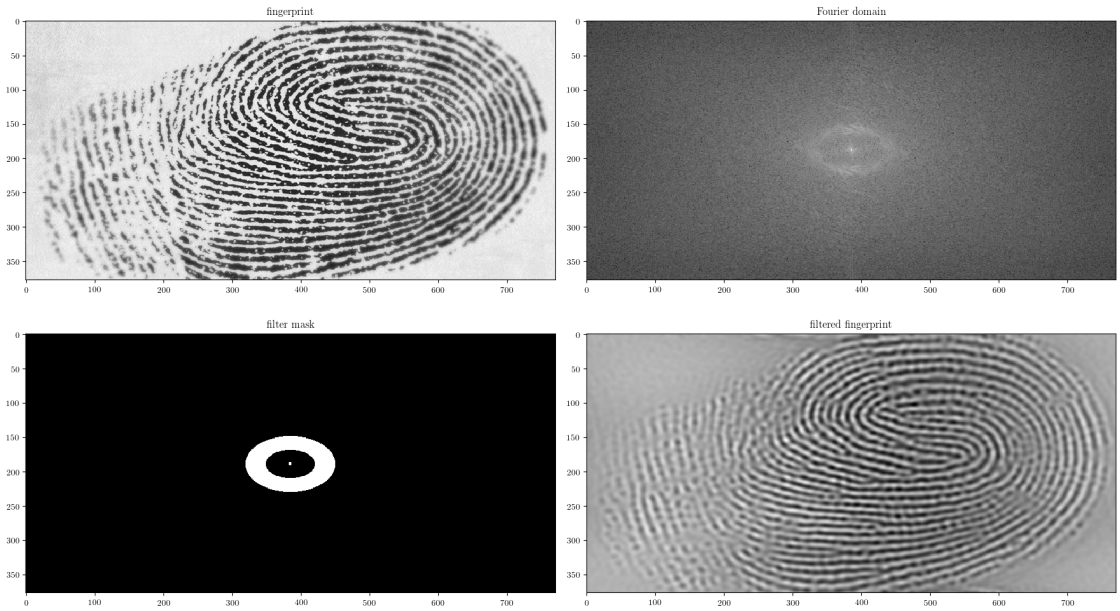
We can observe that for very tight spacing, the dot pattern approaches the appearance of a sinusoid. As the spacings become more distant, the FT pattern is also a pattern of dots with wider spacing. We can think of this as a superposition of many sines that destructively interfere in the right places to form the appearance of a dot.

1.4 Activity 4. Fingerprints: Ridge Enhancement

Fingerprint obtained from fingerprints4all.com

```
In [17]: fing = Image.open("print.jpg").convert("L")
         fing = np.array(fing, "uint8")

         fig = mp.figure(figsize=(5*16/9*2, 5*2))

         ax = fig.add_subplot(221)
         ax.imshow(fing, "gray")
         ax.set_title("fingerprint")

         ax = fig.add_subplot(222)
         FA = fft.fft2(fing)
         ax.imshow(uint8(np.log(abs(fft.fftshift(FA)))), "gray")
         ax.set_title("Fourier domain")

         ax = fig.add_subplot(223)
         mask = Image.new("L", fing.T.shape, color="black")
         draw = ImageDraw.Draw(mask)
         draw.ellipse((320, 150, 450, 230), fill="white")
         draw.ellipse((350, 170, 420, 210), fill="black")
         mask = np.array(mask, "uint8")
         mask = np.round(mask/mask.max()).astype("uint8")
         H, L = mask.shape
         mask[H//2+2-2:H//2+2+2, L//2-2:L//2+2] = 1
         ax.imshow(mask, "gray")
```

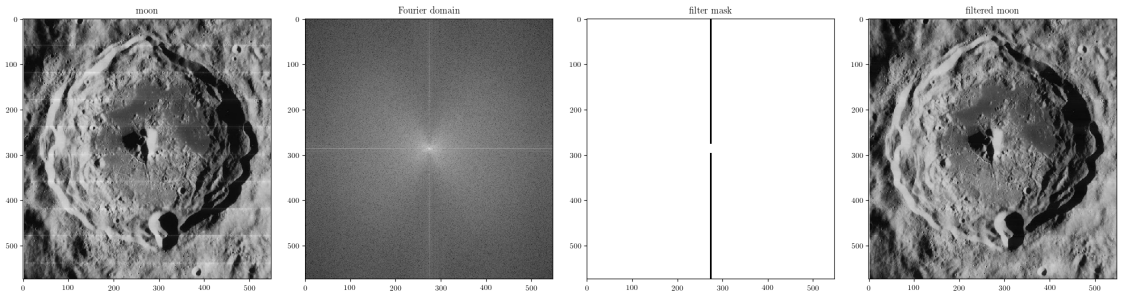
```
#ax.xaxis.set_major_locator(mp.MultipleLocator(20))
#ax.yaxis.set_major_locator(mp.MultipleLocator(20))
ax.set_title("filter mask")

ax = fig.add_subplot(224)
filt = fft.fftshift(FA) * mask
Ifilt = fft.ifft2(filt)
ax.imshow(abs(Ifilt), "gray")
ax.set_title("filtered fingerprint")

mp.tight_layout()
mp.show()
```



Displaying the FT in log scale, we see a bunch of high-intensity spots in an ellipse around the center. These are most likely the components we can safely erase since they are of relatively low frequency. Based on my prior knowledge when post-processing my own photos, the color and tone information lie in the low frequencies, while the details such as edges lie in the high frequencies. We also don't want to get rid of the components at or close to zero since it's highly likely the information that separates the actual fingerprint from the background is contained here.

```
In [18]: moon = img.imread("lunarlines.jpg", mode="L")
         fig = mp.figure(figsize=(5*4, 5))

         ax = fig.add_subplot(141)
         ax.imshow(moon, "gray")
         ax.set_title("moon")

         ax = fig.add_subplot(142)
```

```
FA = fft.fft2(moon)
ax.imshow(np.log(fft.fftshift(abs(FA))), "gray")
ax.set_title("Fourier domain")

ax = fig.add_subplot(143)
mask = np.ones_like(moon, "uint8")
H, L = moon.shape
mask.T[L//2-2 : L//2+2] = 0
mask[H//2-10 : H//2+10, L//2-10 : L//2+10] = 1
ax.imshow(mask, "gray")
ax.set_title("filter mask")

ax = fig.add_subplot(144)
filt = fft.fftshift(FA) * mask
Imoon = fft.ifft2(filt)
ax.imshow(abs(Imoon), "gray")
ax.set_title("filtered moon")

mp.tight_layout()
mp.show()
```

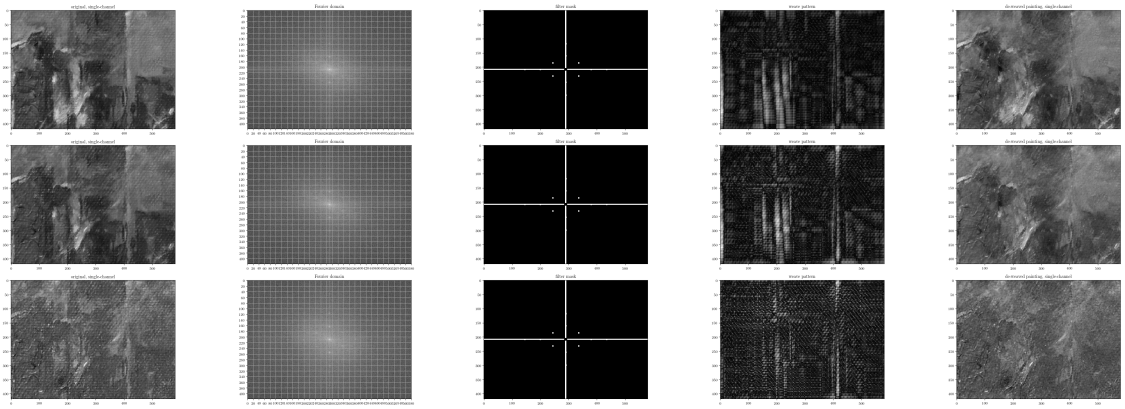


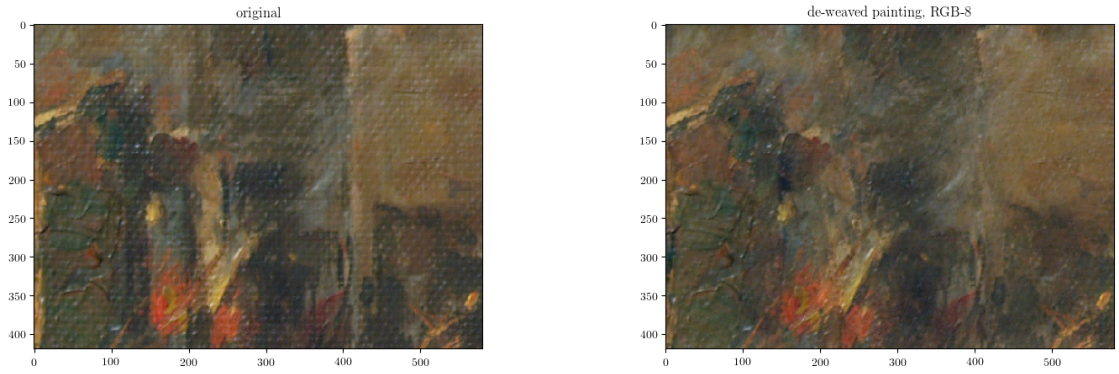
We've seen from previous examples that lines in the spatial domain are usually rotated by 90 degrees when represented in the frequency domain. Thus, we can safely mask out the vertical line in the FT (excluding of course, the frequencies at/near zero) to get rid of the horizontal lines in the original image.

```
In [19]: canvas = img.imread("canvasweave.JPG", mode="RGB")
         fig = mp.figure(figsize=(5*16/9*5, 5*3))
         IA = []

         for i in range(3):
             ax = fig.add_subplot(3, 5, i*5 + 1)
             ax.imshow(canvas[:,:,i], "gray")
             ax.set_title("original, single-channel")

             ax = fig.add_subplot(3, 5, i*5 + 2)
             FA = fft.fft2(canvas[:,:,i])
```

```
ax.imshow(np.log10(fft.fftshift(abs(FA))), "gray")
ax.set_title("Fourier domain")
ax.xaxis.set_major_locator(mp.MultipleLocator(20))
ax.yaxis.set_major_locator(mp.MultipleLocator(20))
ax.grid(True)

ax = fig.add_subplot(3, 5, i*5 + 3)
H, L = canvas[:,:,i].shape
mask = Image.new("LA", canvas[:,:,i].T.shape, color="black")
draw = ImageDraw.Draw(mask)
fftsquare(mask, 2, L//2, 254, fill="white")
fftsquare(mask, 2, L//2, 300, fill="white")
fftsquare(mask, 2, L//2, 161, fill="white")
fftsquare(mask, 2, L//2, 118, fill="white")
fftsquare(mask, 2, 336, 232, fill="white")
fftsquare(mask, 2, 244, 186, fill="white")
fftsquare(mask, 2, 244, 232, fill="white")
fftsquare(mask, 2, 336, 186, fill="white")
fftsquare(mask, 2, 200, H//2, fill="white")
fftsquare(mask, 2, 380, H//2, fill="white")
fftsquare(mask, 2, 435, H//2, fill="white")
fftsquare(mask, 2, 145, H//2, fill="white")
mask = np.array(mask, "uint8")[:,:,0]
mask = np.round(mask/mask.max()).astype("uint8")
#mask[H//2, L//2-30:L//2-5] = 1
#mask[H//2, L//2+5:L//2+30] = 1
mask[H//2-2:H//2+2, :L//2-5] = 1
mask[H//2-2:H//2+2, L//2+5:] = 1
#mask[H//2-10:H//2-5, L//2] = 1
#mask[H//2+5:H//2+10, L//2] = 1
mask[:H//2-5, L//2-2:L//2+2] = 1
mask[H//2+5:, L//2-2:L//2+2] = 1
ax.imshow(mask, "gray")
ax.set_title("filter mask")

ax = fig.add_subplot(3, 5, i*5 + 4)
filt = fft.fftshift(FA) * mask
Icanvas = fft.ifft2(filt)
ax.imshow(abs(Icanvas), "gray")
ax.set_title("weave pattern")

ax = fig.add_subplot(3, 5, i*5 + 5)
maskinv = np.array(mask, bool)
for i in range(mask.size):
    maskinv.flat[i] = not mask.flat[i]
clean = fft.fftshift(FA) * maskinv
Iclean = fft.ifft2(clean)
IA.append(abs(Iclean))
```

```
        ax.imshow(abs(Iclean), "gray")
        ax.set_title("de-weaved painting, single-channel")

    mp.tight_layout()
    mp.show()

    fig = mp.figure(figsize=(5*16/9*2, 5))

    ax = fig.add_subplot(121)
    ax.imshow(canvas)
    ax.set_title("original")

    ax = fig.add_subplot(122)
    IA = np.array(IA, "uint8").transpose(1,2,0)
    ax.imshow(IA)
    ax.set_title("de-weaved painting, RGB-8")

    mp.show()
```





The canvas weave can be thought of like a sum and/or product of many sines, as we've shown in previous examples. In the FT of the painting, we see a horizontal and vertical line running through the center, as well as notable peaks forming a diamond-like shape around the center. As before, the safest bet is to mask out the peaks on the imaginary diamond as well as the lines running through the center, excluding again the low frequency components. I have processed each channel separately and we can see in the final image that the color information was preserved excellently.