PHY407
Lab #5: Computing Fourier Transforms

Q1: Nicholas Pavanel, Q2: Hayley Agler

October 2020

Q1

a)

See Figures 1, and 2. Figure 1 shows the number of sunspots per month for every month since January 1749. Note that the origin for the x-axis of Figure 1 is the year 1749, $(x(0) = 1749$, in which data on the number of sunspots started being recorded. Focusing on a small area of the x-axis of Figure 1, approximately one period of the sun-spot axis, provides an estimate of a 150 month peak-to-peak period. I choose to use a peak-to-peak period for the added simplicity when determining where a period begins and ends, that is, at the peak.

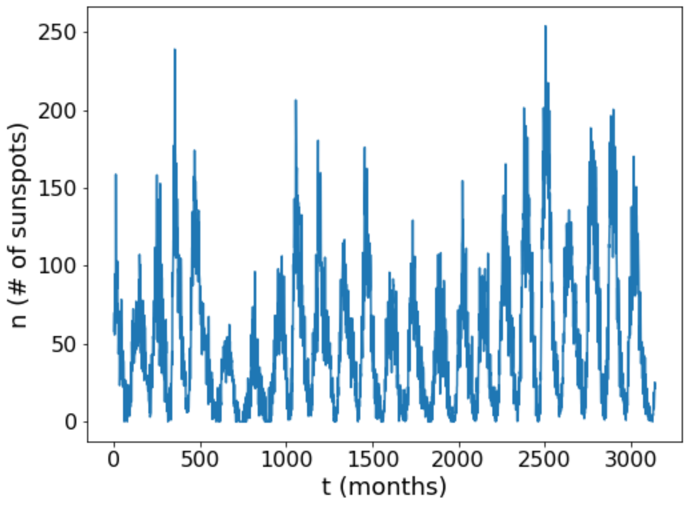

Figure 1: This plot show the number of sunspots observed per month since January 1794. Notice that there exists a periodicity to the number of sunspots that are observed in a given month. Thus, the 'sunspot cycle', refers to the periodic, time-dependant nature of the amount of sunspots that the sun has at a given time.

Figure 2 shows the power spectrum of the sunspot-data plotted in Figure 1. The power spectrum was created by taking a discrete Fourier transform (DFT) of the sunspot data. The program that was used to take the DFT was from page 5 of the class notes. The power spectrum plotted in Figure 2 is really a plot of the DFT coefficients $c_k = \Sigma_{n=1}^{N-1} y_n exp(-i\frac{2\pi kn}{N})$. Power spectrum's are useful because of the way they are created. Since power spectrum's are created by plotting Fourier coefficients, they show how much of the signal in question is represented by the individual waves of the Fourier transformation. Figure 2 plots the k^{th} coefficient against its magnitude. The location of the peak in the power spectrum was at $k = 23$ by sorting the data. The peak at $k = 23$ tells us that a large portion of the sun-spot cycle is contained within one sine term: $f_{k=23}(x) = c_{23} sin(\frac{2\pi 23x}{L})$ where L is the total number of months that the sun-spot data encompasses.

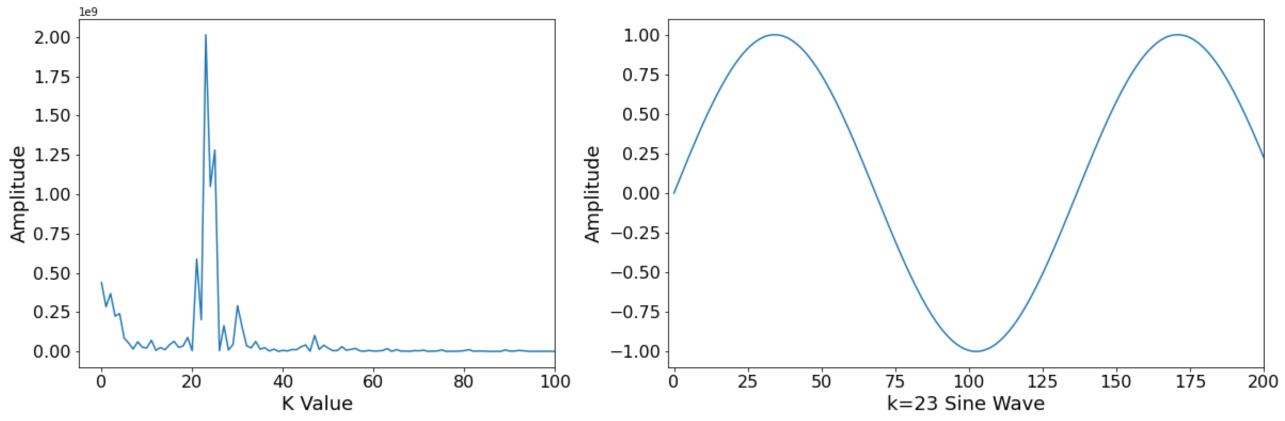

Figure 2: This plot shows the power spectrum of the sunspot data shown in 1 on left and a portion of the 23^rd sine wave of the Fourier transformation that creates the sunspot cycle, $f_{k=23}(x) = c_{23} sin(\frac{2\pi 23x}{L})$, of Figure 1 on right. On right, this plot shows us that the majority of the sunspot signal is represented by the 23^{rd} sine wave of the Fourier transformation. On left, Note that the peak-to-peak period approximately matches the peak-to-peak cycle of Figure 1, the sunspot cycle. This reaffirms Figure 2.

A portion of $f_{k=23}(x)$ is plotted in Figure 2. Note that the peak-to-peak period of $f_{k=23}(x)$ is approximately 150 months, the same as what was estimated for the sunspot cycle. The shared period between the sunspot cycle and $f_{k=23}(x)$ displays that the periodicity of the sunspot cycle is largely encapsulated by $f_{k=23}(x)$, as the single peak of the power spectrum implies.

b)

See Figure 3. Figure 3 shows the closing values of the Dow Jones Industrial Average from late 2006 until the end of 2010 in yellow, a slightly smoothed out version of the data in red, and a largely smooth out version of the data in blue. To 'smooth' the data, the following program was used:

```
def smooth(data, co_per):
    fft_data = np.fft.rfft(data) # take a Fourier transformation of the data
    cutoff_index = int(co_per * len(data)) # determine cutoff from percentage that we want t0 keep
    fft_data_subset = np.array(fft_data)
    fft_data_subset[cutoff_index:] = 0 # set all indicies past cutoff_index equal to zero
    ifft_data = np.fft.irfft(fft_data_subset) # inverse Fourier transform subset of coefficients
    return ifft_data
```

When a subset of the Fourier coefficients are used when inverse Fourier transforming the data back to normal, some of the original data of the function is lost. This can be seen in Figure 3. Notice that with 10 percent of the Fourier coefficients (red), the inverse Fourier transformation models the data quite well. With only 2 percent of the Fourier coefficients (blue), the inverse Fourier transformation only picks out the general trends of the data.

c)

See Figure 4. Figure 4 shows the closing values of the Dow Jones Industrial Average from 2004 until 2008 in yellow, a DFT smoothed out version of the data in red, and a discrete cosine transformation (DCT) in blue.

Firstly notice that that the DFT smoothing process approximates the data quite well everywhere but the ends. The two large tails at the start and the end of the DFT data result from the fact that DFT requires

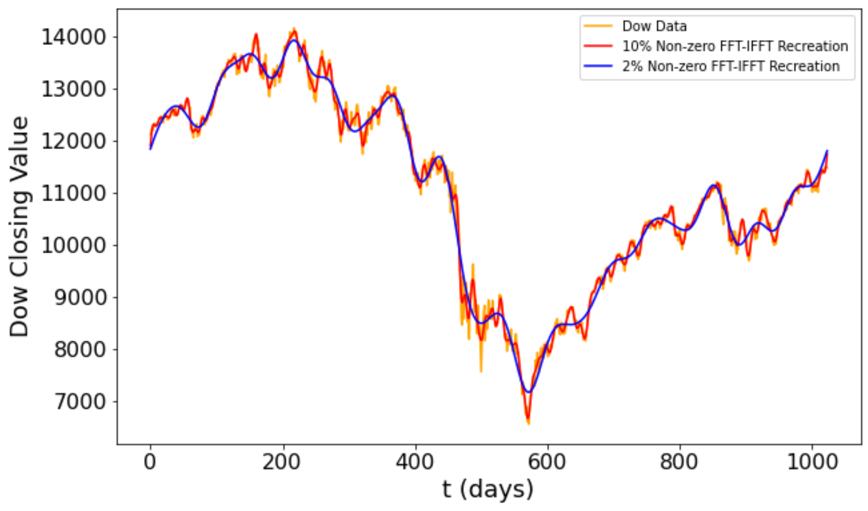

Figure 3: This figure shows an example of how data can be 'smoothed' out. By taking the Fourier transformation of the data, setting all Fourier coefficients to zero except for a small subset, and taking the inverse Fourier transformation over the modified set of Fourier coefficients, the general trends of the data can be picked out from the local fluctuations.

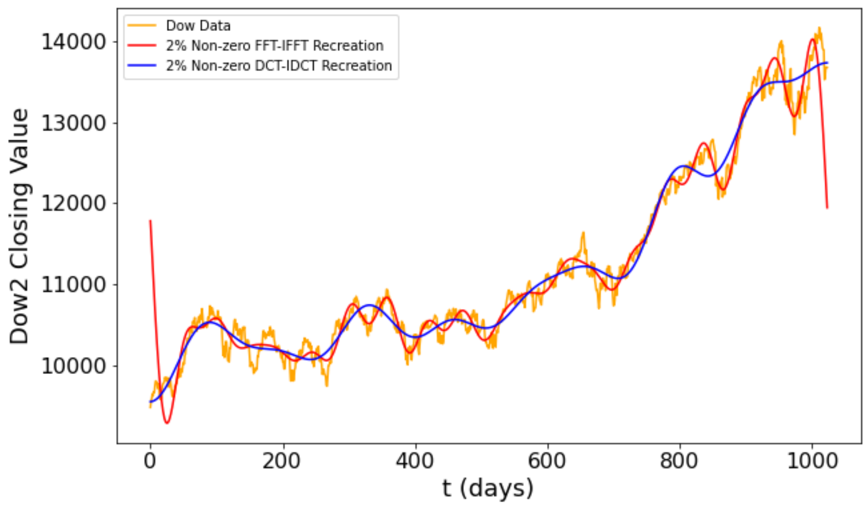

Figure 4: This figure shows a comparison between DFT smoothing and DCT smoothing. Notice that although DCT smoothing is not restricted to periodic data sets, it requires more coefficients than DFT to produce the same accuracy.

a periodic data set. That is, it requires, at the very least, a data set that starts and ends in similar places. Next notice that the DCT smoothing process does not have the tails. This is because DCT does not require a periodic data set. Finally notice that, although by the DFT and DCT smoothing routines used the same number of coefficents to return to the orignal data, DFT approximates the data better than DCT.

d)

See Figure 5 and Figure 5's description. Figure 5 shows the waveforms and the power spectrum of a piano and a trumpet when playing the same note. To determine what note was played, I use the formula: $n_{Hz} = k44100100000$ where I take k to be the dominant wave in the piano power spectrum, 44,100 to be the sampling rate of the sound and 100000 to be the total number of samples. I use the piano's power spectrum because it only requires a small number of frequencies to produce its note, thus the dominant frequency should be close to that of the note's frequency. As a result, with $k = 1191$, $n = 525.231Hz$. This is approximately the note of a fifth octave C, $C_5 = 523.251$.

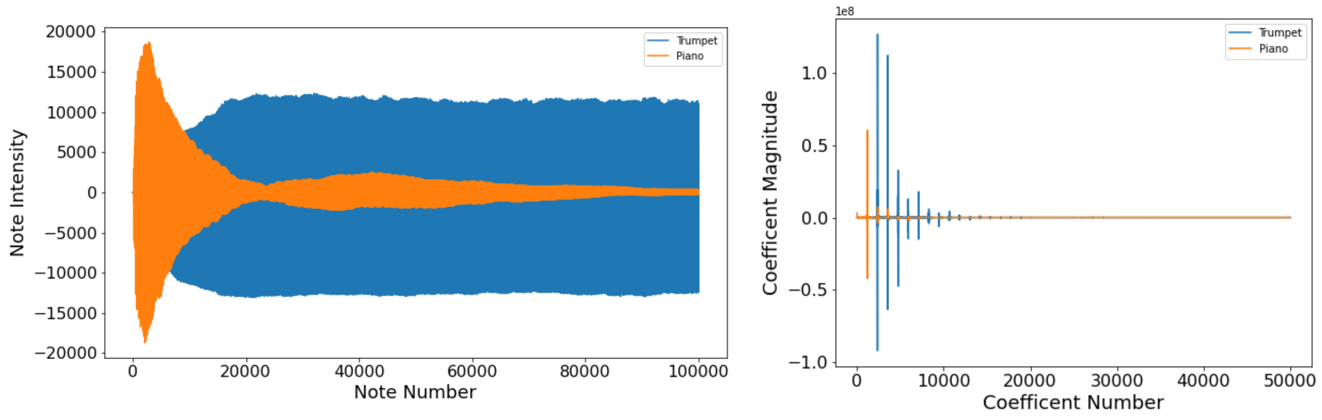

Figure 5: On left, this figure shows the waveform of a single note played by both a trumpet and a piano. The piano waveform represents the initial strike of a piano key, and then the gradual reverberations as the note tails off. The trumpet waveform represents the constant sound produced as a player plays a continuous single note. On right, this figure shows the power spectrum of notes played by the piano and trumpet. The appearance of multiple large peaks in the trumpet's power spectrum implies that the trumpet needs to use a large number of large number of frequencies to produce a single note. This is in contrast to the small number of large peaks in the piano's power spectrum, meaning that the piano does not need to use a lot of frequencies to create a single note.

Q2

a)

The file "blur.txt" was read into a 2D array and then plotted as a density plot. The resulting image can be seen in Figure 6.

b)

A grid of samples drawn from the Gaussian point spread function:

$$f(x,y) = \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \tag{1}$$

with $\sigma = 25$, then plotted in a density plot in Figure 7.

c)

The Fourier transforms of the blurred image and of the point spread function were taken using numpy.fft's rfft2 function. The former was divided by the latter to give the Fourier transform of the pure image, and

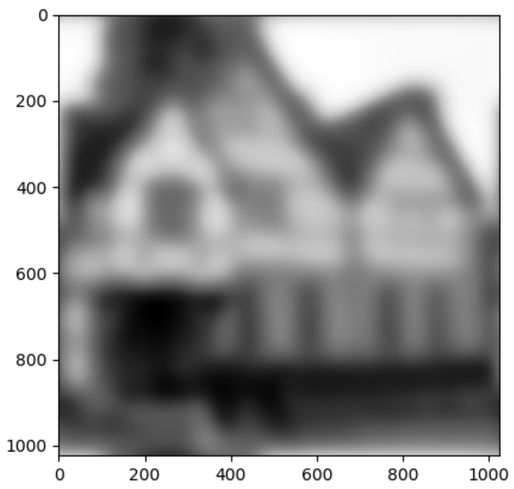

Figure 6: The blurry photo created from a 2D array of brightness values.

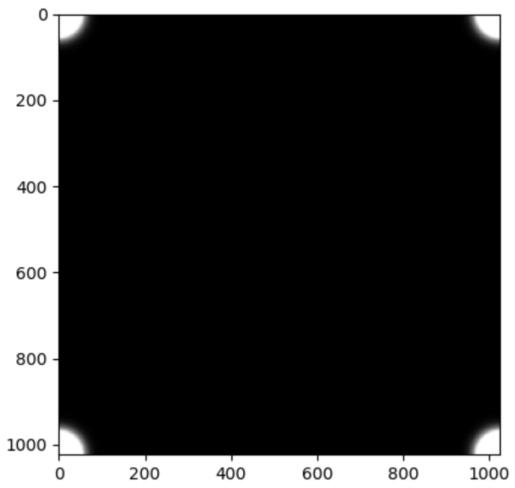

Figure 7: A density plot of the point spread function.

the inverse Fourier transform was then taken to give the pure image as seen in Figure 8. Since some of the values of the Fourier transform of the point spread function were close to 0, these values were left out of the division to avoid getting an undefined value.

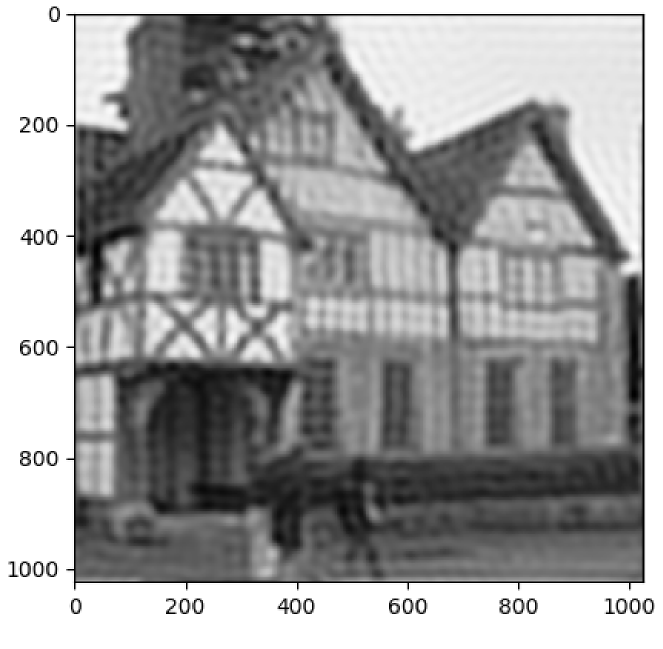

Figure 8: The final unblurred image resulting from the sharpening procedure.

Code

Q1

```
#needed imports
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt

# a)

#load in the data
sun_data = np.loadtxt('PHY407_Lab5_sunspots.txt')

#plot the data
plt.figure(figsize=[8,6])
plt.plot(sun_data[:,0],sun_data[:,1])
plt.xlabel('t (months)',size=18)
plt.ylabel('n (# of sunspots)',size=18)
plt.xticks(size=16)
plt.yticks(size=16)
#plt.savefig('1_a')
plt.show()
plt.close()

#import the discrete fourier transform function from page 5 of class notes
def dft(y):
    N = len(y)
    c = np.zeros(N//2+1, complex) # we only need the first 1/2 of the points
    for k in range(N//2+1): # let's do loops, pedagogy > speed today
        for n in range(N): # trapezoidal integration with y0=yend
            c[k] += y[n]*np.exp(-2j*np.pi*k*n/N)
    return c

#fourier transform the sunspot data with dft
fft_sun_data = dft(sun_data[:,1])

#get data ready to plot
sun_fft = np.abs(fft_sun_data[1:])**2

#plot the figure
plt.figure(figsize=[8,6])
plt.plot(sun_fft)
plt.xlabel('K Value',size=18)
plt.ylabel('Amplitude',size=18)
plt.xticks(size=16)
plt.yticks(size=16)
#plt.xlim([-5,100])
plt.xscale('log')
#plt.savefig('1_b1')
#plt.yscale('log')
plt.show()
plt.close()
```

```
#to find the approximate value of k, we may argsort
peak_k = np.argsort(sun_fft)[-1]
print('K-value of peak:',peak_k)

#find the period of the sine wave with k=23
sun_sine = sun_fft[23] * np.sin( (2 * np.pi * peak_k * sun_data[:,0]) / (len(sun_data[:,0])) )

#plot sign curve
plt.figure(figsize=[10,6])
plt.plot(sun_sine)
plt.xlim([-2,200])
plt.xticks(size=16)
plt.yticks(size=16)
plt.xlabel('k=23 Sine Wave',size=18)
plt.ylabel('Amplitude',size=18)
plt.savefig('1_b2')
plt.show()
plt.close()

# b)

#load in the data
dow_data = np.loadtxt('PHY407_Lab5_dow.txt')

#define a function that smooths a data set by taking its Fourier transformation,
#choosing to keep a subset of the Fourier coefficients (co_per = what percentage to keep),
#and inverse Fourier transforming back with those coefficients
def smooth(data, co_per):
    fft_data = np.fft.rfft(data) # take a Fourier transformation of the data
    cutoff_index = int(co_per * len(data)) # determine cutoff index from percentage that we want t0 keep
    fft_data_subset = np.array(fft_data)
    fft_data_subset[cutoff_index:] = 0 # set all indicies past cutoff_index equal to zero
    ifft_data = np.fft.irfft(fft_data_subset) # take inverse Fourier transform with a subset of coeffici
    return ifft_data

#smooth dow data with smooth
smooth_dowdata_2per = smooth(dow_data,0.02)
smooth_dowdata_10per = smooth(dow_data,0.1)

#plot dow data w ifft'd data
plt.figure(figsize=[10,6])
plt.plot(dow_data,label='Dow Data',color='orange')
plt.plot(smooth_dowdata_10per,label='10% Non-zero FFT-IFFT Recreation',color='red')
plt.plot(smooth_dowdata_2per,label='2% Non-zero FFT-IFFT Recreation',color='blue')
plt.xlabel('t (days)',size=18)
plt.ylabel('Dow Closing Value',size=18)
plt.xticks(size=16)
plt.yticks(size=16)
plt.legend()
plt.savefig('1_b3')
plt.show()
plt.close()
```

```
# c)

#load in the data
dow2_data = np.loadtxt('PHY407_Lab5_dow2.txt')

#smooth dow data 2 with smooth
smooth_dowdata2_2per = smooth(dow2_data,0.02)

#define a function that smooths a data set by taking its Fourier transformation,
#choosing to keep a subset of the Fourier coefficients (co_per = what percentage to keep),
#and inverse Fourier transforming back with those coefficients
def smooth_dct(data, co_per):
    dct_data = sp.fft.dct(data) # take a Fourier transformation of the data
    cutoff_index = int(co_per * len(data)) # determine cutoff index from percentage that we want t0 kee
    dct_data_subset = np.array(dct_data)
    dct_data_subset[cutoff_index:] = 0 # set all indicies past cutoff_index equal to zero
    idct_data = sp.fft.idct(dct_data_subset) # take inverse Fourier transform with a subset of coeffici
    return idct_data

#smooth dow data 2 with smooth_dct
smooth_dowdata2_2per_dct = smooth_dct(dow2_data,0.02)

#plot dow data w ifft'd data
plt.figure(figsize=[10,6])
plt.plot(dow2_data,label='Dow Data',color='orange')
plt.plot(smooth_dowdata2_2per,label='2% Non-zero FFT-IFFT Recreation',color='red')
plt.plot(smooth_dowdata2_2per_dct,label='2% Non-zero DCT-IDCT Recreation',color='blue')
#plt.plot(ifft_dow200_sp,label='20% Non-zero DCT-IDCT Recreation',color='black')
plt.xlabel('t (days)',size=18)
plt.ylabel('Dow2 Closing Value',size=18)
plt.xticks(size=16)
plt.yticks(size=16)
plt.legend()
#plt.savefig('1_c')
plt.show()
plt.close()

# d)

#load data
trumpet_data = np.loadtxt('PHY407_Lab5_trumpet.txt')
piano_data = np.loadtxt('PHY407_Lab5_piano.txt')

#plot raw data
plt.figure(figsize=[11,6])
plt.plot(trumpet_data,label='Trumpet')
plt.plot(piano_data,label='Piano')
plt.legend()
plt.xticks(size=16)
plt.yticks(size=16)
plt.xlabel('Note Number',size=18)
plt.ylabel('Note Intensity',size=18)
#plt.savefig('1_d1')
plt.show()
plt.close()

#calculate the Fourier coefficients with numpy rfft
fft_trumpet = np.fft.rfft(trumpet_data)
fft_piano = np.fft.rfft(piano_data)

#plot fft data
plt.figure(figsize=[8,6])
plt.plot(fft_trumpet,label='Trumpet')
plt.plot(fft_piano,label='Piano')
plt.legend()
plt.xticks(size=16)
plt.yticks(size=16)
plt.xlabel('Coefficent Number',size=18)
plt.ylabel('Coefficent Magnitude',size=18)
#plt.xlim([750,2000])
#plt.savefig('1_d2')
plt.show()
plt.close()

#determine what note is being played
n =  np.argsort(fft_piano)[-1] * (44100) / 100000
print('Note being played is', n, 'Hz')
```

Q2

```
#import necessart modules
from numpy.fft import rfft2, irfft2
import numpy as np
import matplotlib.pyplot as plt

blur=np.loadtxt("/Users/owner/407/blur.txt")

#define values for point spread function
sigma=25
rows, cols = blur.shape
gauss=np.empty([1024,1024], float)

#routine from the lab to create a 2d array of point spread values
for i in range(rows):
    ip = i
    if ip > rows/2:
        ip -= rows # bottom half of rows moved to negative values
    for j in range(cols):
        jp = j
        if jp > cols/2:
            jp -= cols # right half of columns moved to negative values

        gauss[i, j] = np.exp(-(ip**2 + jp**2)/(2.*sigma**2)) # compute gaussian

#fourier transform the blurry photo and point spread function
blur_trans=rfft2(blur)
gauss_trans=rfft2(gauss)
```

```
#define empty array for fourier trans of the division to be stored in
img_trans=np.empty(gauss_trans.shape, complex)

#divide one by the other, except when gauss_trans is small
epsilon=1e-3
for i in range(len(gauss_trans)):
    for j in range(len(gauss_trans.T)):
        if np.abs(gauss_trans[i,j])>epsilon:
            img_trans[i,j]=blur_trans[i,j]/gauss_trans[i,j]
        else:
            img_trans[i,j]=blur_trans[i,j]


#inverse transform the transform of pure image
img=irfft2(img_trans)

#%% plotting
plt.figure(1)
plt.imshow(blur, cmap="Greys_r")
plt.figure(2)
plt.imshow(gauss, cmap="Greys_r", vmax=0.1)
plt.figure(3)
plt.imshow(img, cmap="Greys_r")
```