Assignment 3

Birk Nøhr Dissing
Jakob Damgaard Olsen
Ian Pascal Møller

February 27, 2023

1. Fourier Transform - Theory

1.1

A Fourier series is a decomposition of periodic signal into a sum of sines and cosines, while the Fourier Transform is a transform that decomposes non-periodic function into the frequencies that it is made up of. The main difference between the Fourier series and transform is that the Fourier series handles periodic signal in a discrete manner while the Fourier Transform can handle non-periodic signal in an infinite interval.

1.2

Any function can be written as an addition of an even and odd function. For complex functions this becomes

$$f(x) = (\epsilon(x) + o(x)) + i(\hat{\epsilon}(x) + \hat{o}(x)) \tag{1}$$

Where ϵ denotes the even part and o denotes the odd part. The $\hat{}$ denotes that it is for the imaginary part of the function. Using Eulers equation the continuous Furier transform can be written as

$$F(u) = \int_{-\infty}^{\infty} f(x)e^{-i2\pi xu}dx = \int_{-\infty}^{\infty} f(x) \cdot (\cos(2\pi xu) - i \cdot \sin(2\pi xu))dx \tag{2}$$

Using how $f(x)$ can be written in terms of even and odd functions the Fourier transform can be written as

$$F(u) = \int_{-\infty}^{\infty} ((\epsilon(x) + o(x)) + i(\hat{\epsilon}(x) + \hat{o}(x))) \cdot (\cos(2\pi xu) - i \cdot \sin(2\pi xu))dx \tag{3}$$

Using the fact that an integral from $-a$ to a is 0 for an odd function and that cos and sin are even and odd respectively the Fourier transform can be reduced to

$$F(u) = \int_{-\infty}^{\infty} (\epsilon(x) + i\hat{\epsilon}(x)) \cdot \cos(2\pi xu)dx - i\int_{-\infty}^{\infty} (o(x) + i\hat{o}(x)) \cdot \sin(2\pi xu)dx \tag{4}$$

When the function is real and even the Fourier transform reduces to

$$f(x) = \epsilon(x) \tag{5}$$

$$F(u) = \int_{-\infty}^{\infty} \epsilon(x) \cdot \cos(2\pi xu)dx \tag{6}$$

which means that the Fourier transform of an even and real function also is even and real.

1.3

We need to solve the equation

$$\int_{-\infty}^{\infty}(\delta(x-d)+\delta(x+d))e^{-i2\pi kx}\,dx = \int_{-\infty}^{\infty}\delta(x-d)e^{-i2\pi kx}\,dx + \int_{-\infty}^{\infty}\delta(x+d)e^{-i2\pi kx}\,dx \quad (7)$$

Using that the fourier transform of a single delta function is

$$\int_{-\infty}^{\infty}\delta(x+d)e^{-i2\pi kx}\,dx = e^{-i2\pi kd} \textbf{ and } \int_{-\infty}^{\infty}\delta(x-d)e^{-i2\pi kx}\,dx = e^{i2\pi kd}$$

We get that

$$\int_{-\infty}^{\infty}\delta(x-d)e^{-i2\pi x}\,dx + \int_{-\infty}^{\infty}\delta(x+d)e^{-i2\pi x}\,dx = e^{i2\pi d} + e^{i2\pi d} \quad (8)$$

We can now expand this using Euler's formula $e^{ix} = cos(x) + i \cdot sin(x)$ to get

$$e^{i2\pi kd} + e^{i2\pi kd} = cos(2\pi kd) + i \cdot sin(2\pi kd) + cos(-2\pi kd) + i \cdot sin(-2\pi kd) \rightarrow \quad (9)$$

$$cos(2\pi kd)+i\cdot sin(2\pi kd)+cos(2\pi kd)-i\cdot sin(-2\pi kd) = 2cos(2\pi kd)+i\cdot sin(2\pi kd)-i\cdot sin(2\pi kd)$$

Which can be simplified to our final result

$$\int_{-\infty}^{\infty}(\delta(\mathbf{x-d})+\delta(\mathbf{x+d}))\mathbf{e^{-i2\pi kx}}\,\mathbf{dx} = \mathbf{2cos(2\pi kd)} \quad (10)$$

1.4

i.

We are given the following box function:

$$b_a(x) = \begin{cases} \frac{1}{a} & \text{if} |x| \leq \frac{a}{2} \\ 0 & \text{otherwise} \end{cases} \quad (11)$$

The following integral needs to be evaluated:

$$\int_{-\infty}^{\infty} b_a(x)dx \quad (12)$$

The integral can be rewritten and evaluated in the following way as the box function is 0 when $|x| > \frac{a}{2}$.

$$\int_{-\infty}^{\infty} b_a(x)dx = \int_{-\frac{a}{2}}^{\frac{a}{2}} b_a(x)dx = \frac{1}{a}\cdot\left(\frac{a}{2}\right) - \frac{1}{a}\cdot\left(\frac{-a}{2}\right) = \frac{1}{2} - \frac{-1}{2} = 1 \quad (13)$$

The integral is therefore the following:

$$\int_{-\infty}^{\infty} b_a(x)dx = 1 \quad (14)$$

ii.

The Fourier transform of the box function $b_a(x)$ is:

$$B_a(k) = \int_{-\infty}^{\infty} b_a(x) e^{-i2\pi xk} dx = \int_{-\infty}^{\infty} b_a(x) \cdot (\cos(2\pi xk) - i \cdot \sin(2\pi xk)) dx = \tag{15}$$

$$\int_{-\infty}^{\infty} b_a(x) \cdot \cos(2\pi xk)) dx + \int_{-\infty}^{\infty} b_a(x) \cdot (-i \cdot \sin(2\pi xk)) dx$$

$b_a(x)$ is an even function which means the second integral is 0 as $\sin(x)$ is odd.

$$B_a(k) = \int_{-\infty}^{\infty} b_a(x) \cdot \cos(2\pi xk)) dx = \int_{-\frac{a}{2}}^{\frac{a}{2}} b_a(x) \cdot \cos(2\pi xk)) dx \tag{16}$$

The integral is then solved.

$$B_a(k) = \int_{\frac{-a}{2}}^{\frac{a}{2}} b_a(x) \cdot \cos(2\pi xk)) dx = \tag{17}$$

$$\left[b_a(x) \cdot \frac{\sin(2\pi xk)}{2\pi k} \right]_{\frac{-a}{2}}^{\frac{a}{2}} - \int_{\frac{-a}{2}}^{\frac{a}{2}} \frac{db_a(x)}{dx} \cdot \frac{\sin(2\pi xk)}{2\pi k} dx$$

As $\frac{db_a(x)}{dx} = 0$ in the interval $|x| \leq \frac{a}{2}$ the integral becomes 0.

$$B_a(k) = \left[b_a(x) \cdot \frac{\sin(2\pi xk)}{2\pi k} \right]_{\frac{-a}{2}}^{\frac{a}{2}} = \frac{1}{a} \cdot \left(\frac{\sin\left(2\pi \frac{a}{2} k\right)}{2\pi k} - \frac{\sin\left(2\pi \frac{-a}{2} k\right)}{2\pi k} \right) = \tag{18}$$

$$\frac{1}{a} \cdot \left(\frac{\sin(\pi ak)}{2\pi k} + \frac{\sin(\pi ak)}{2\pi k} \right) = \frac{1}{k\pi a} \sin(k\pi a)$$

Using the function $\text{sinc}(x) = \frac{\sin(x)}{x}$ $B_a(k)$ can be rewritten as:

$$B_a(k) = \text{sinc}(k\pi a) \tag{19}$$

iii.

The following limit is found:

$$\lim_{a \to 0} B_a(k) = \lim_{a \to 0} \text{sinc}(k\pi a) \tag{20}$$

When a goes to 0 $k\pi a$ also goes to 0 which means the limit can be written the following way.

$$\lim_{a \to 0} B_a(k) = \lim_{x \to 0} \text{sinc}(x) = 1 \tag{21}$$

The limit of $B_a(k)$ when a goes to 0 is therefore 1.

iv.

The sinus in $B_a(k)$ can be written in terms of a constant and k.

$$B_a(k) = \frac{1}{k\pi a} \sin(\omega k) \quad \textbf{where} \omega = \pi a \tag{22}$$

When a is small ω is also small. This means that the changes in k affects the value of $\sin(\omega k)$ little. $B_a(k)$ is therefore broad in the frequency domain when a is small. When a is large the box function is wide in the x space domain. In the frequency domain ω is large when a is large. $B_a(k)$ is therefore narrow in the frequency domain when a is small.

2. Fourier Transform - Practice

2.1

A function which uses nested for loops to do convolution between a filter and images was created.

```
def apply_filter(image, filter):
    cop_image = np.copy(image)
    out_image = np.zeros(image.shape)
    row = (filter.shape[0]-1)//2
    column = (filter.shape[1]-1)//2
    for i in range(cop_image.shape[0]-filter.shape[0]):
        for j in range(cop_image.shape[1]-filter.shape[1]):
            out_image[i+row,j+column] = np.sum(cop_image[i:i+
               filter.shape[0],j:j+filter.shape[1]]*filter)
    return out_image
```

The function does not create any padding at the borders of the image and the output image is therefore cropped compared to the input. Only pixels which can be calculated with the kernel is used in the output image.

A function was also created which uses the Fourier transformation to do convolution.

```
def fft_convolve(image, filter):
    cop_image = np.copy(image)
    prod = fft2(image, image.shape) * fft2(filter, image.shape)
    ifftprod = ifft2(prod).real
    convImg = np.roll(ifftprod, (-((filter.shape[0] - 1)//2), -((
       filter.shape[1] - 1)//2)), axis=(0, 1))
    return convImg
```

The way the convolution was calculated in the Fourier domain, was by first converting the image and kernel into Fourier space. Afterwards the convolution can be calculated by taking the product of the image and kernel in the Fourier domain. This product can then be transformed back into our original domain by doing an inverse Fourier transform. To then make the values align with the ones used in methods such as when doing cyclic convolution with Scipy's convolve2d, one has to roll the values of the axis which is done in the last line before we return the final convolution.

The image used was "cameraman.tif" and the filter is a NxN mean kernel. In figure 1 a 3x3 and 7x7 kernel was applied to the image using both nested for loop and Fourier transformation. The difference in the images created by the different function is also shown. The two functions result in identical images for both kernel sizes except for the borders where the fact that the nested for loop crops the image which the Fourier transformation function does not. In figure 2 a 5x5 mean kernel was applied to different image sizes. Again except for at the borders of the image, where the nested for loops crops the original image, the results of the two functions are the same.

Figure 1: Variuous kernel sizes of a mean filter was applied to the image using nested for loops and Fourier transformation. The difference between the two functions is shown and they result in the same image.

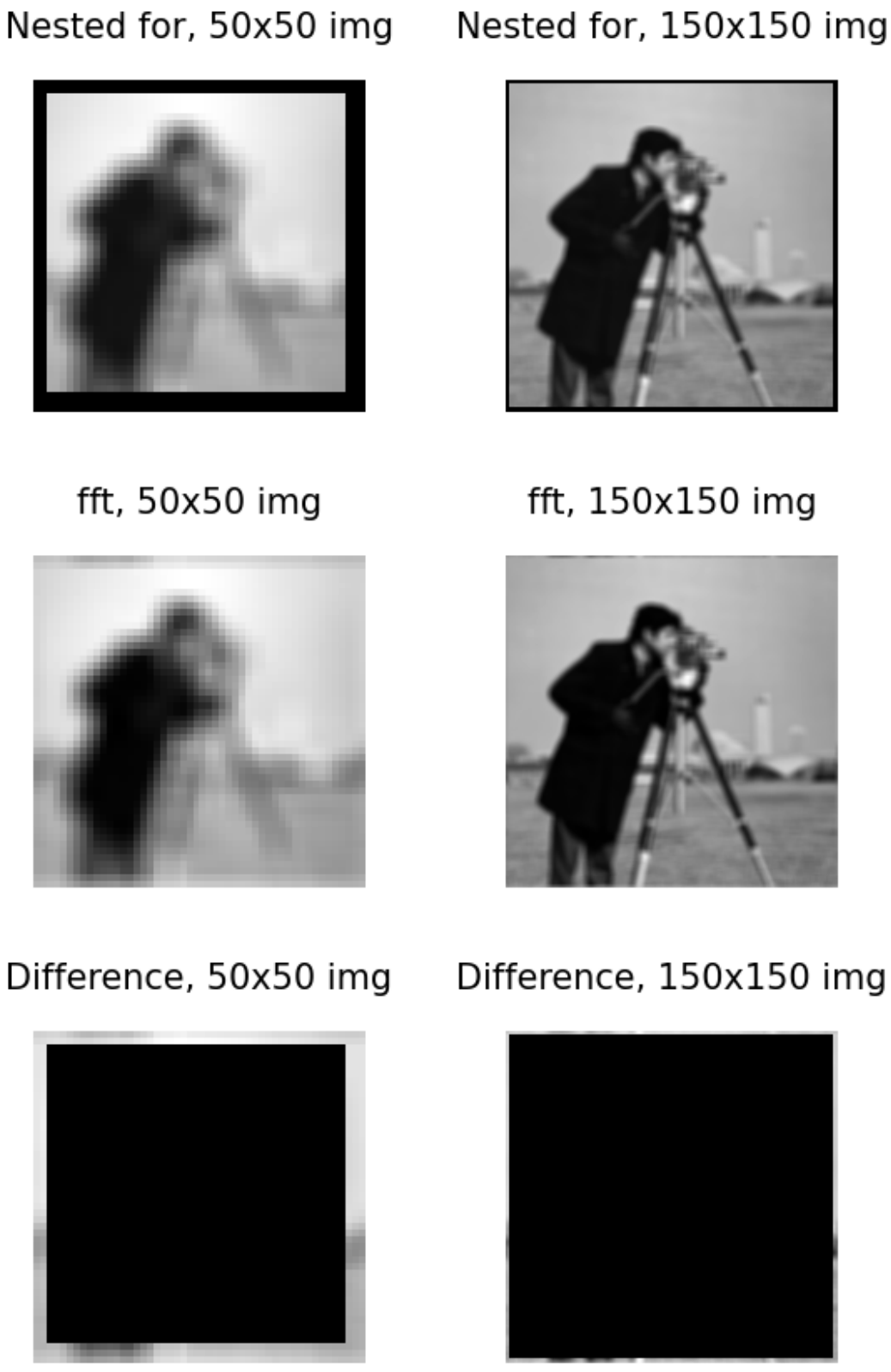

Figure 2: A 5x5 mean kernel was applied to images with varying sizes. The difference in the image from the two functions is shown where it can be seen that the functions result in the same image.

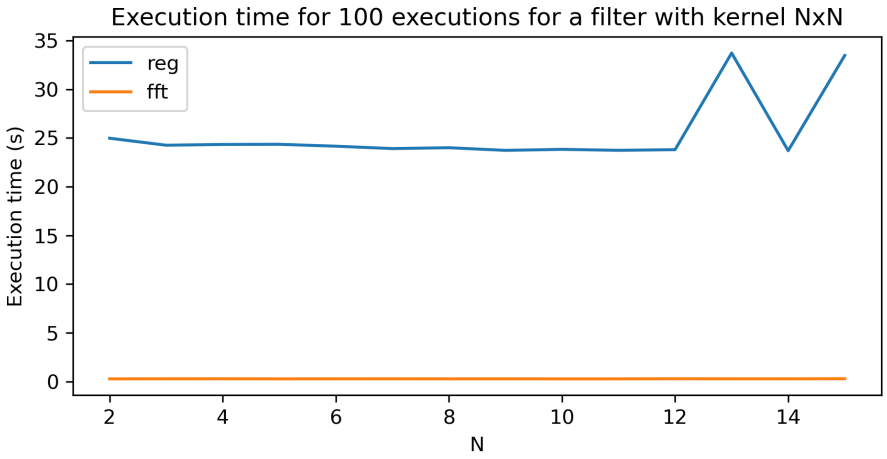

Figure 3: The time it takes to do 100 executions of one of the functions. The "reg" is the nested for loop function while "fft" is the Fourier transformation function. The image size is constant at 256x256 pixels while the kernel size increases from 2x2 to 15x15. The kernel is a mean filter.

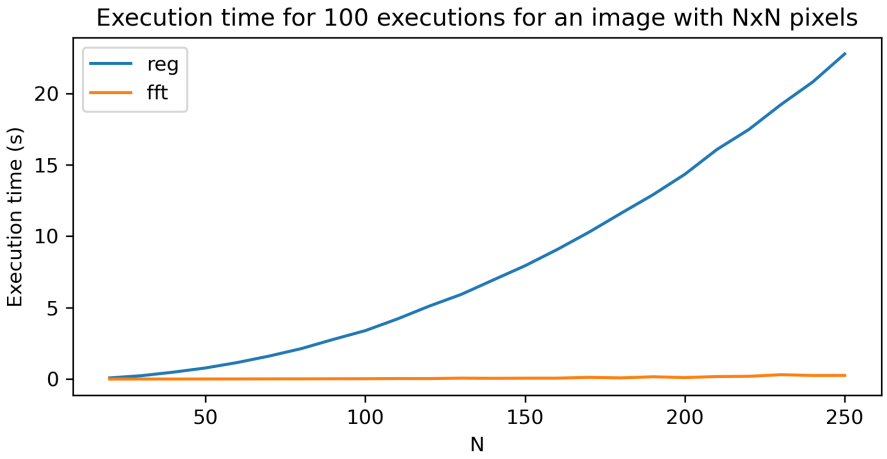

Figure 4: The time it takes to do 100 executions of one of the functions. The "reg" is the nested for loop function while "fft" is the Fourier transformation function. The kernel is a 5x5 mean filter for all executions with the images size being NxN pixels.

In figure 3 the execution time of the two functions for various kernel sizes can be seen for a 256x256 pixel image. It is obvious that the Fourier transformation function is much faster than the nested for loop function. The execution time of both functions does not seem to depend on the size of the kernel. For the nested for loop function this can be explained by the fact that the image is cropped. This means that when the kernel is larger there is fewer pixels the kernel needs to be applied to. The execution time therefore does not scale with N^2 which is otherwise expected for a general nested for loop convolution. In figure 4 the execution time for various image sizes with a constant 5x5 mean kernel can be seen. Here it can be seen that at low pixel values the execution time for the two functions are similar. The execution time for the Fourier transformation function does not depend on the image size while the nested for loop function does with what seems to be

N^2 relation. For large images the Fourier transformation is therefore much faster than the nested for loop, while their performances are similar for small images.

2.2

The code used to add the planar wave was the following, where a cosine wave was created and added to the input image.

```
1 def waveAdd(image,a,v,w):
2     x, y = np.meshgrid(np.arange(image.shape[1]), np.arange(image.
          shape[0]))
3     cos_wave = a * np.cos(v * x + w * y)
4     img = image + cos_wave
5     return img
```

Using the values $a = 50$, $v = 0.5$, and $w = 0.5$ the cosine noise was added and can be seen in figure 5.

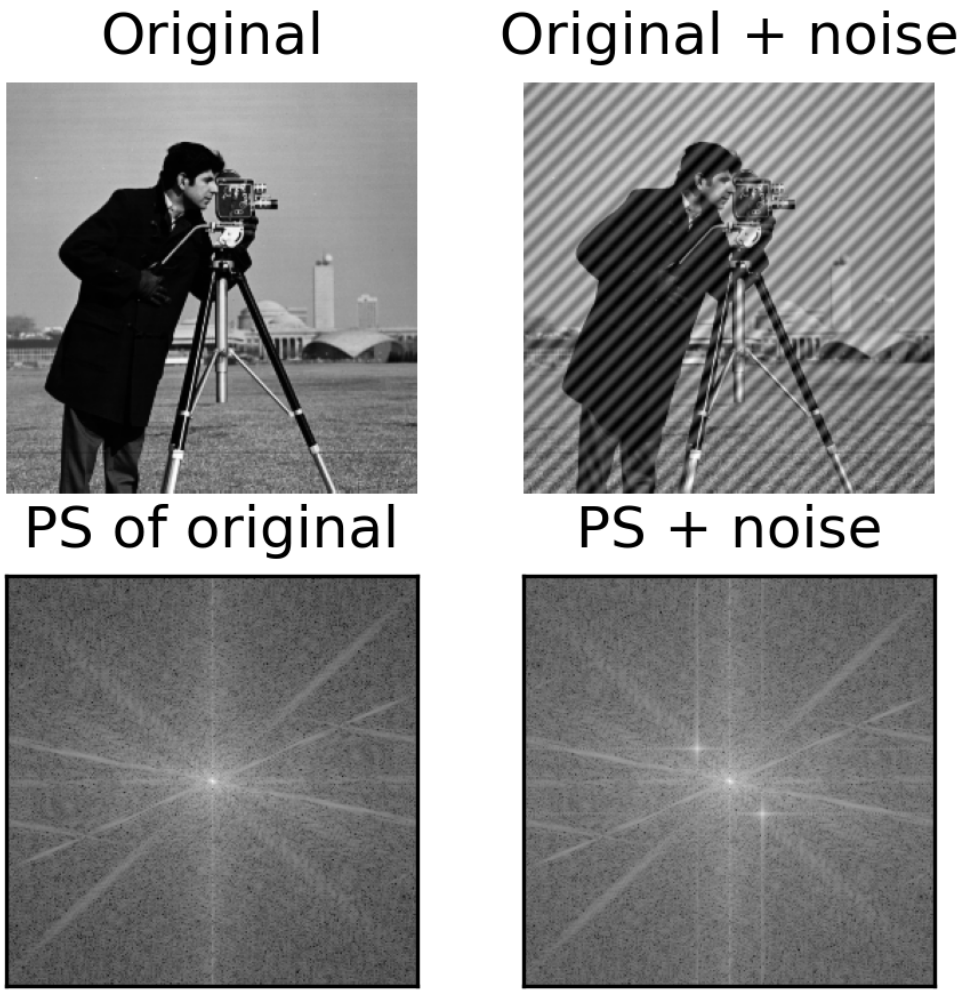

Figure 5: Comparing the original image of "cameraman.tif" to one with a cosine wave added onto it. The top plots show the image itself while the bottom plots show the power spectrum of each image.

From figure 5 we see that when adding the cosine wave to the image, this results in the image getting waves on it. When analyzing this using the power spectrum we see that there now is two bright spots which correspond to this new planar wave that has been added onto the image.
To remove this wave again a handcrafted filter was made by finding which pixels were the center of the bright spots. These were found to be [107,107] and [148,148]. The following code was then used to filter out the planar wave.

```
def filterFunc(fft,pixel1,pixel2):
    N = fftB.shape[0]
    x,y = np.meshgrid(np.arange(N), np.arange(N))
    a1, a2 = 0.005, 0.005
    F1 = 1 - np.exp(-a1*(x-pixel1[0])**2-a2*(y-pixel1[1])**2)
    F2 = 1 - np.exp(-a1*(x-pixel2[0])**2-a2*(y-pixel2[1])**2)
    Z = F1*F2
```

```
    imgFs = fft*Z
    imgF = ifftshift(imgFs)
    imgF = ifft2(imgF)
    return imgF, Z
```

This function takes in the fft of an image, the pixel position of the added noise, and then creates two Gaussian's at the center of these pixels as a filter which gets multiplied onto the fft. The constructed Gaussian's are 0 at the center and goes towards 1 when getting further away from the center. The result of applying this filter can be seen in figure 6. We can see that the filter indeed did remove some of the cosine wave, but not everything. We are able to still see the waves at the edge of the image, and also the edges between the man and his surroundings.

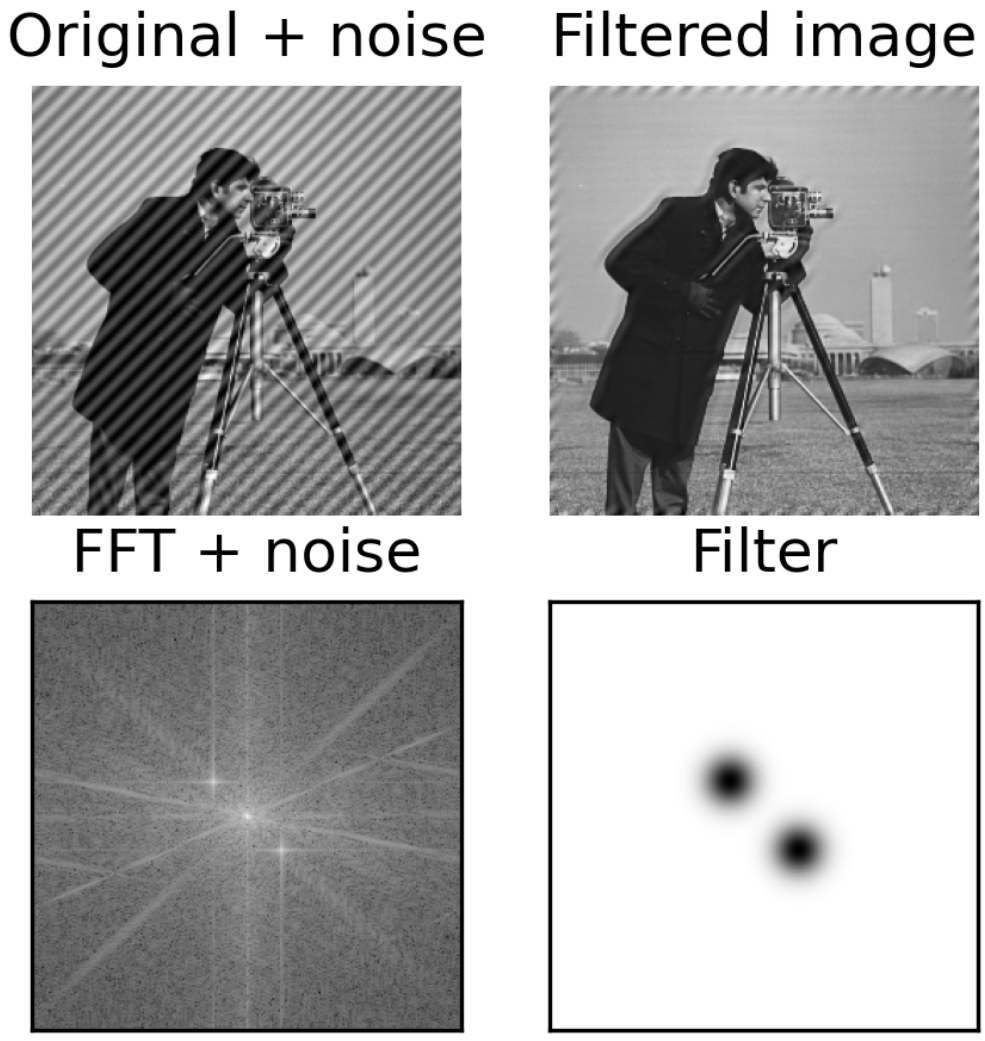

Figure 6: Showing the result of filtering the original image. The top plots shows a comparison of the image before and after filtering, while the bottom left plot shows the FFT of the original image with noise, and the bottom right plot shows the filter Z.