Assignment 3

Bjarke Heldbo, Toke Reines & Frederik Christiansen
ku-id: rvs299 , bkx591 & nzt365

February 28, 2022

Contents

1 Theory **2**

2 Practice **4**

1 Theory

(a) There is a very nice answer to this question from the following math.stackexchange post [1]. The essence of that answer is that a Fourier series is used to represent a periodic function by a discrete sum of complex exponentials, and the Fourier transform is then used to represent a general, nonperiodic function by a continuous superposition or integral of complex exponentials. We can therefore look at the Fourier transform as the limit of the Fourier series of a function, when the period approaches to infinity. Meaning that the limits of integration change from one period to $(-\infty, \infty)$.

(b) Let $f : \mathbb{R} \to \mathbb{R}$ be an integrable and even function, and denote its Fourier transform by $\hat{f}$, i.e

$$\hat{f}(\omega) = \int_{\mathbb{R}} f(x) e^{-i\omega x} \mathrm{d}x.$$

Use the substitution $z = -x$ and let $\omega = 2\pi k$. From this we get that

$$\begin{aligned}
\overline{\hat{f}(\omega)} &= \hat{f}(-\omega) \\
&= \int_{-\infty}^{\infty} f(x) e^{i\omega x} \mathrm{d}x \\
&= \int_{-\infty}^{\infty} f(-x) e^{i\omega x} \mathrm{d}x \\
&= \int_{-\infty}^{\infty} f(z) e^{-i\omega z} \mathrm{d}z \\
&= \hat{f}(\omega).
\end{aligned}$$

This proofs the desired result.

(c) Using the sifting property of the Dirac delta function [2] we get that

$$\begin{aligned}
\hat{\delta}(x-d) - \hat{\delta}(x+d) &= \int_{\mathbb{R}} \delta(x-d) e^{-i\omega x} \mathrm{d}x + \int_{\mathbb{R}} \delta(x+d) e^{-i\omega x} \mathrm{d}x \\
&= e^{-i\omega d} - e^{i\omega d} \\
&= 2\cos(2\pi k d).
\end{aligned}$$

In the last equality we used Eulers formula [3]

[1] https://math.stackexchange.com/questions/221137/what-is-the-difference-between-fourier-series-and-fourier-transformation

[2] https://mathworld.wolfram.com/SiftingProperty.html

[3] https://en.wikipedia.org/wiki/Euler%27s_formula#Relationship_to_trigonometry

(d) i. A direct caluclation gives

$$\begin{aligned}\int_{\mathbb{R}} b_a(x)\mathrm{d}x &= \frac{1}{a}\int_{-a/2}^{a/2} 1\mathrm{d}x \\ &= \frac{1}{a}\left(\frac{a}{2}+\frac{a}{2}\right) \\ &= 1.\end{aligned}$$

ii. The continuous Fourier transform using the definition of the Fourier transform given in Bracewell Chapter 2 (system 1) is

$$F(k) = \int_{\mathbb{R}} f(x)e^{-i\omega x}\mathrm{d}x.$$

Using the fact that b_a is an even function (see the slides from 22-02-2022 page 4) and $\omega = 2\pi k$ we get that

$$\begin{aligned}B_a(k) &= \int_{\mathbb{R}} f(x)e^{-i\omega x}\mathrm{d}x \\ &= \frac{1}{a}\int_{-a/2}^{a/2}\cos(\omega x)\mathrm{d}x \\ &= \frac{1}{a}\left[\frac{1}{\omega}\sin(\omega x)\right]_{x=-a/2}^{x=a/2} \\ &= \frac{1}{a}\left(\frac{1}{2\pi k}\sin(\pi a k)+\frac{1}{2\pi k}\sin(\pi a k)\right) \\ &= \frac{\sin(\pi a k)}{\pi a k} \\ &= \mathrm{sinc}(\pi a k).\end{aligned}$$

is $B_a(k) = \frac{1}{ak\pi}\sin(ak\pi)$. This gives the desired expression of $B_a(k)$.

iii. Using L'Hôpital's rule yields the desired result. We have that

$$\lim_{a\to 0} B_a(k) = \lim_{a\to 0} B_a(k) = \lim_{a\to 0}\frac{\sin(ak\pi)}{ak\pi} = \lim_{a\to 0}\frac{k\pi\cos(ak\pi)}{k\pi} = \lim_{a\to 0}\cos(ak\pi) = 1.$$

iv. When a is near zero, then b_a is narrow in the x space domain by definition of b_a. Conversely, b_a is wide in the space domain for large values of a.

From the expression of $B_a(k)$ found in the previous question we see that $B_a(k)$ is wide for a near zero, because the cycles from the sine wave become long. We can therefore conclude that the Fourier transformation is wide in the frequency domain, k for small a. The converse holds for large values of a. Consider for example $k \in [-1, 1]$ for different values of a, let's say $a \in \{1, 2, 3, 4\}$ then we see by plotting this that the Fourier transform becomes more wide for small a, and more narrow for larger values of a. See figure 1.

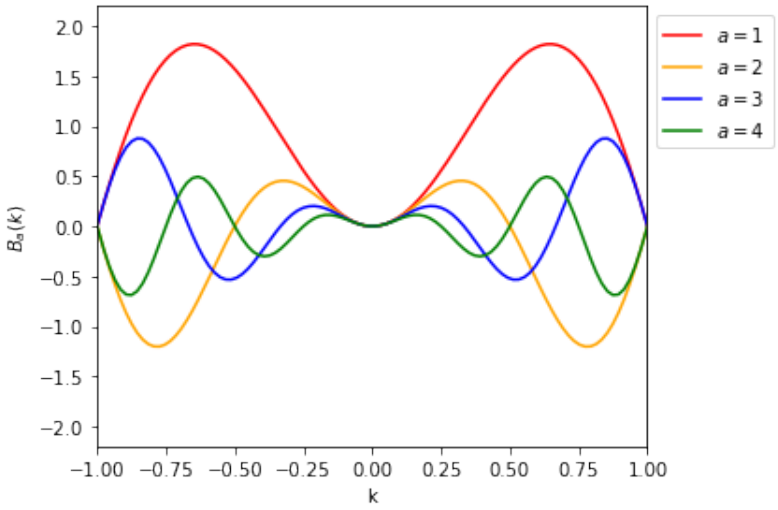

Figure 1: $B_a(k)$ for different values of a.

2 Practice

(a) We calculated the power of spectrum of *'trui.png'* by the use of *scipy.fft.fft2* and *scipy.fft.fftshift*. The implementation is given by the following code:

```
## power spectrum with shifted frequencies.
def ps(I,t):
    f = scipy.fft.fft2(I)
    fshift = scipy.fft.fftshift(f)
    if t == 1:
        return np.log(np.square(np.abs(fshift)))
    else:
        return np.square(np.abs(fshift))
```

The representation of the result can be seen in figure 2. We decided to include the power spectrum log transformed, since it provides a good visualization. See figure 2. We observe that most of the low frequency content are visualized by the white region at the center.

(b)

(c) The two functions are provided below.

```
def convolve2d(image, kernel):
    h, w = kernel.shape
    h = h // 2
    w = w // 2
    convolved_img = np.zeros(image.shape)
```

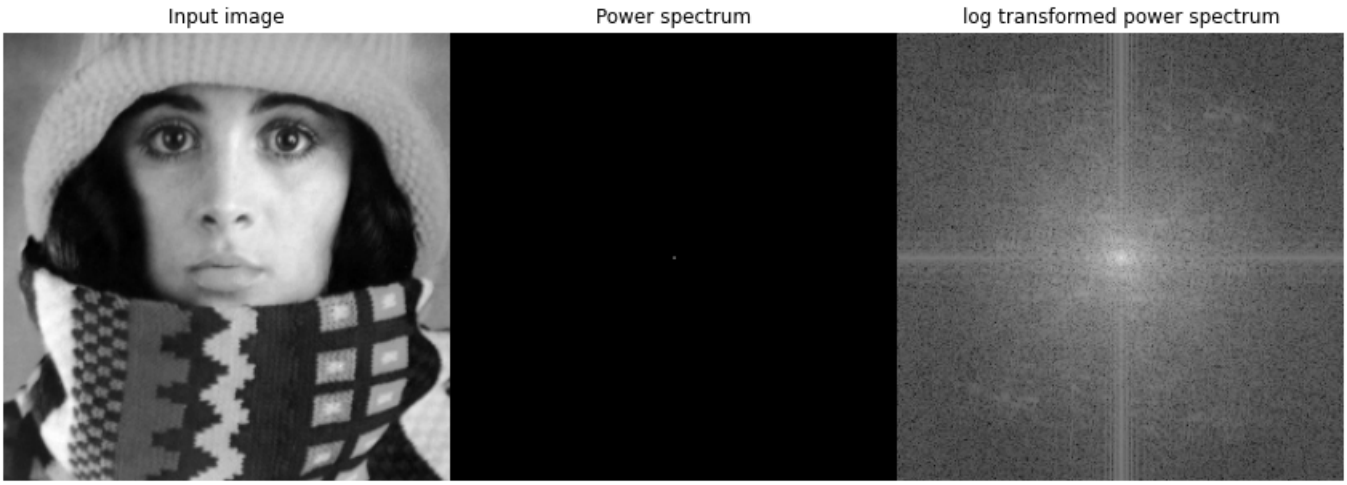

Figure 2: Representation of the power spectrum of *trui.png'*.

```
    for x in range(h, image.shape[0] - h):
        for y in range(w, image.shape[1] - w):
            sum = 0
            for m in range(kernel.shape[0]):
                for n in range(kernel.shape[1]):
                    sum += kernel[m][n] * image[x - h + m][y - w + n]
            convolved_img[x][y] = sum

    return convolved_img
```

```
def convolvefft(image, kernel):
    def _pad_kernel():
        # total amount of padding
        sz = (image.shape[0] - kernel.shape[0], image.shape[1] -
   kernel.shape[1])
        return ifftshift(np.pad(kernel, (((sz[0] + 1) // 2, sz[0] //
   2),
                                         ((sz[1] + 1) // 2, sz[1] //
   2)),
                                           'constant'))

    padded_kernel = _pad_kernel()
    image_fft = fft2(image)
    kernel_fft = fft2(padded_kernel)
    conv_fft = image_fft * kernel_fft
    conv = np.absolute(np.real(ifft2(conv_fft)))
    return conv
```

The results are illustrated in figure 3. See figure 3. Looking at the difference images in the bottom row in the above image, it is clear to see that the two approaches result in exactly the same image. The spatial convolution solution uses padding, whereas

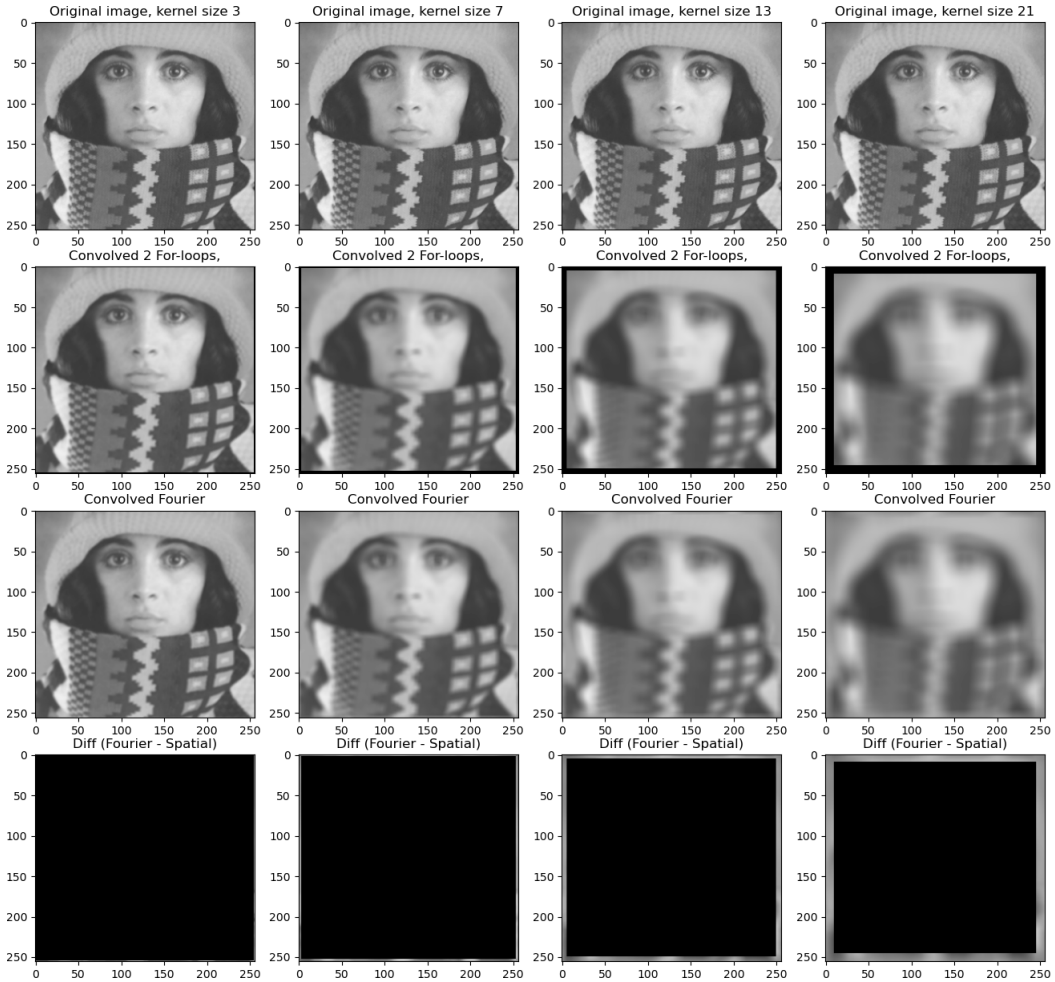

Figure 3: Mean filter as convolution in spatial domain and in frequency domain using the Fast Fourier Transform and kernel sizes [3, 7, 13, 2]

	Convolution timings for kernel-size n			
n	3	7	13	21
Spatially	0.019820ms	1.501293ms	4.67574ms	11.139721ms
Fourier	0.329953ms	0.020007ms	0.01973ms	0.020218ms

the frequency domain, living on a torus, does not result in a padded image, which results in the only difference between the two solutions being the padding. Runtime wise, kernel size does not really impact the Fourier time, but has a great impact in the two-for-loops-implementation. See the table below. A convolution in the frequency domain requires two functions/images of equal dimensions, so the kernel needs to be padded either way in the fft convolution function.

(d) We first write program, which creates the cosine wave with the form $a_0 \cos(v_0 x + w_0 y)$. An image I as float type can simply just be added to this wave. The implementation is given by

```
# Cosine wave.
def wave(I,a_0 = 1,v_0 = 1,w_0 = 1):
    x = np.arange(0,I.shape[0],1)
    y = np.arange(0,I.shape[1],1)
    X, Y = np.meshgrid(x, y)
   W = a_0 * np.cos(v_0*X + w_0*Y)
    return I + W
```

We look at three different cosine waves. Specifically, $I_1 = I + \cos(x)$, $I_2 = I + \cos(y)$ and $I_3 = I + \cos(x + y)$. Observe that we decided to set $a_0 = 1$. The visualizations of the power spectrum can be seen in figure 4. See figure 4. The low frequency content is on a horizontal line through the center for I_1, a vertical line through the center for I_2 and a diagonal line through the center for I_3.

In order to remove the planar given v_0 and w_0 waves we design a low pass filter. The idea is to remove high frequency content in the image. It will blur the image, but at the same time smooth out the planar waves. We create a mask first with high value 1 at low frequencies. Meaning that we let the low frequency content pass. The low passing filter is implemented by

```
def LPF(I,a,b):
    f = scipy.fft.fft2(I)
    fshift = scipy.fft.fftshift(f)
    rows, cols = I.shape
    crow,ccol = rows//2 , cols//2
    # Create a mask first, center square is 1, remaining all zeros
    mask = np.zeros((rows,cols),np.uint8)
    mask[crow-a:crow+a, ccol-b:ccol+b] = 1
    # Apply mask and inverse DFT
    product = fshift*mask
    f_ishift = np.fft.ifftshift(product)
```

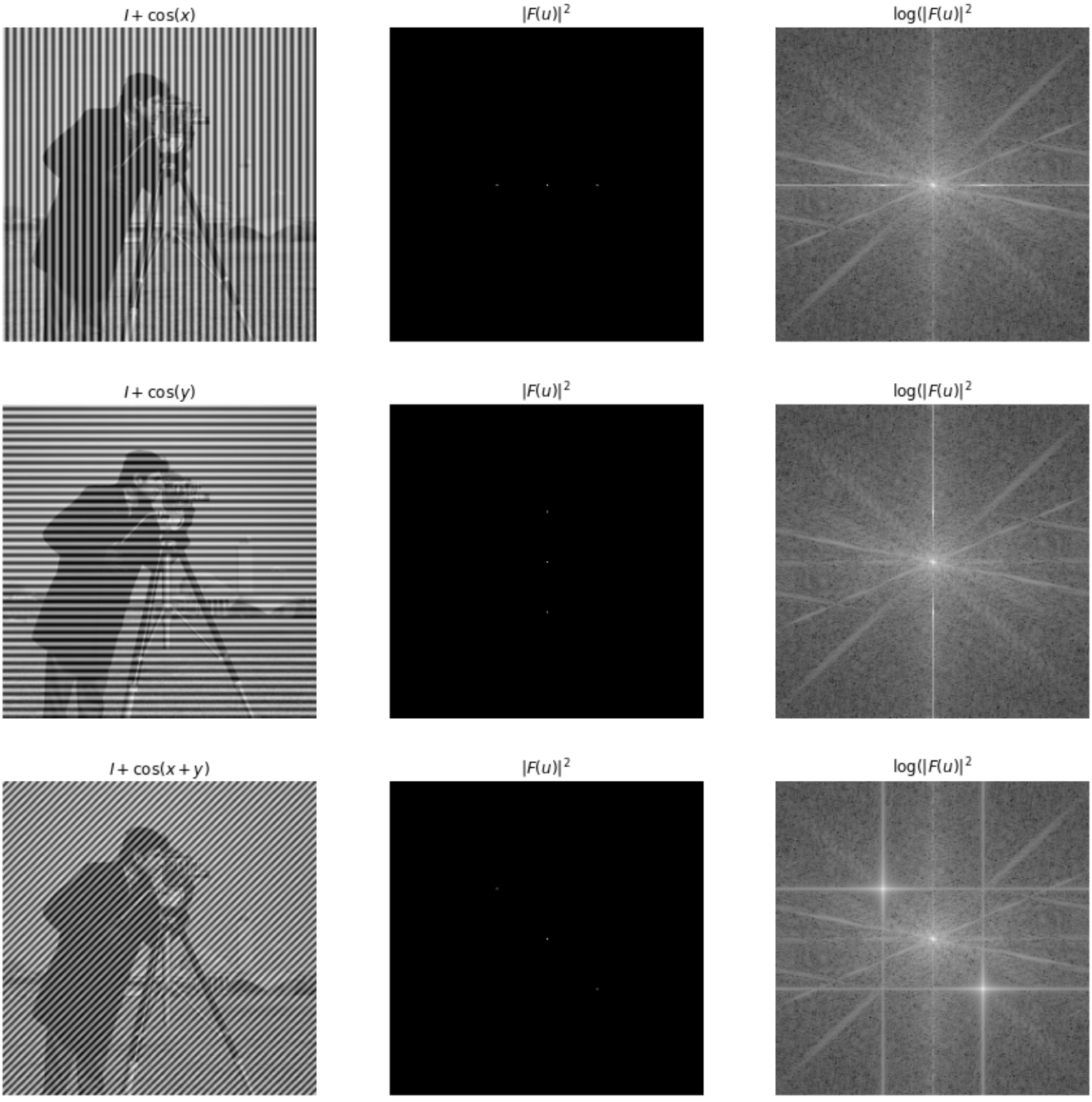

Figure 4: Representation of the power spectrum of *'cameraman.tif'* with added cosine waves.

```
    img_back = np.fft.ifft2(f_ishift)
    img_back = np.abs(img_back)

    return img_back
```

The idea of the mask is was taken from an Open-CV post[4]. If we set $a = 30$ and $b = 30$ we get three filtered images, which can be seen in figure 5. See figure 5. From the figure we see that the straight planar waves are smoothed out. The result is in all three cases a blurry image of *'cameraman.tif'*.

[4] https://docs.opencv.org/3.4/de/dbc/tutorial_py_fourier_transform.html

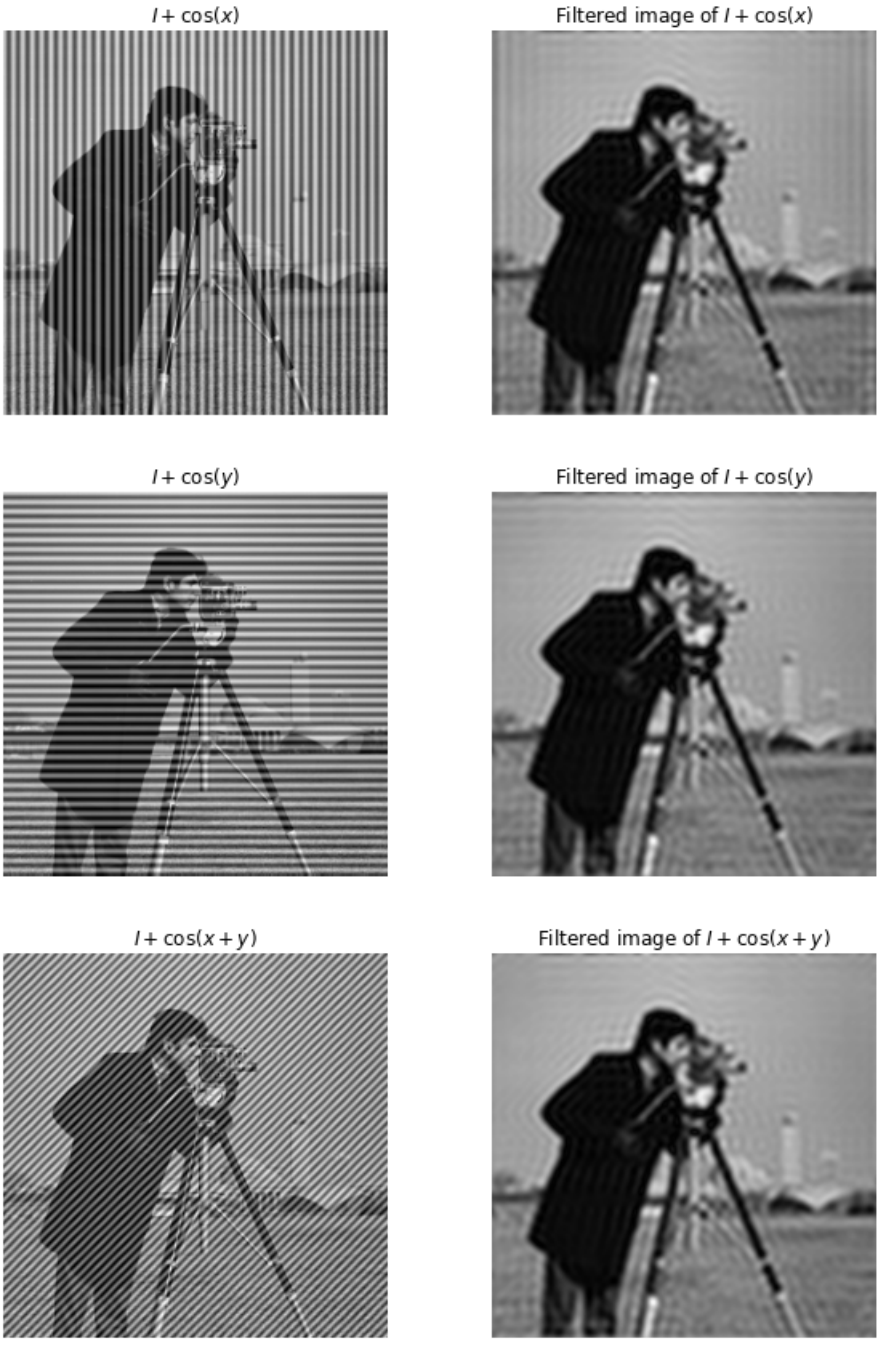

Figure 5: Filtered images.