Signal and image processing
Assignment 2

Marcus Hansen - tkz347
Casper Bresdahl - Whs715

March 1, 2021

Exercise 1

a

The Fourier series is defined as: $\lim N \to \infty \sum_{n=-N}^{N} c_n e^{\frac{i2\pi nx}{T}}$. A Fourier series is a sum of infinite (countable) many terms of a periodic function which makes the sinoid function approximate the periodic function.
The Fourier transform is defined as: $\int_{-\infty}^{\infty} f(x) exp(i2\pi xs)$ds. A Fourier transform takes a continuous and square differentiable function and maps the function into a new space that depends on spatial/temporal frequencies.

b

Given a real and even function $f_{even}(x)$, we can find its Fourier transform $F(u)$:

$$F(u) = \int_{-\infty}^{\infty} f_{even}(x) cos(-2\pi ux))dx$$

Using the inverse Fourier transform we can go back to $f_{even}(x)$, and we can thus show:

$$\begin{aligned} \int_{-\infty}^{\infty} F(u) cos(2\pi ux)dx = f_{even}(x) \quad & \textbf{Inverse Fourier transform} \\ = f_{even}(-x) \quad & \textbf{f is even} \end{aligned}$$

We can now take the Fourier transform of $f_{even}(-x)$ and find:

$$\int_{-\infty}^{\infty} f_{even}(-x) cos(-2\pi u(-x)))dx = F(-u)$$

Which proves the continuous Fourier transform of a real and even function is real and even, i.e. $F(u) = F(-u)$.

c

$$\begin{aligned}
F\{\delta(x-d)+\delta(x+d)\} &= F\{\delta(x-d)\}+F\{\delta(x+d)\} \quad \text{(Addition theorem)}\\
&= exp(-2\pi iad)F(s)+exp(2\pi iads)F(s) \quad \text{(Shift theorem)}\\
&= exp(-2\pi iad)+exp(2\pi ids)\\
&= cos(2\pi ds)-i\cdot sin(2\pi ds)+cos(2\pi ds)+i\cdot sin(2\pi ds)\\
&= 2\cos(2\pi ds)
\end{aligned}$$

And thus we find the Fourier transform of $\delta(x-d)+\delta(x+d)$ is $2\cos(2\pi ds)$.

d, i

Because $b_a(x)$ is 0 everywhere else than in the interval $-\frac{a}{2}$ to $\frac{a}{2}$ we can show:

$$\begin{aligned}
\int_{-\infty}^{\infty} b_a(x)dx &= \int_{-\frac{a}{2}}^{\frac{a}{2}} \frac{1}{a}dx\\
&= \left[\frac{x}{a}\right]_{-a/2}^{a/2}\\
&= \frac{a}{2}\cdot\frac{1}{a}-\left(-\frac{a}{2}\cdot\frac{1}{a}\right)\\
&= \frac{1}{2}+\frac{1}{2}\\
&= 1
\end{aligned}$$

And thus we have $\int_{-\infty}^{\infty} b_a(x)dx = 1$.

d, ii

Taking the Fourier transform of $b_a(x)$ we find:

$$\begin{aligned}
\int_{-\infty}^{\infty} b_a(x)e^{-i2\pi xk}dx &= \int_{-\frac{a}{2}}^{\frac{a}{2}} \frac{1}{a}e^{-i2\pi xk}dx\\
&= \frac{i\left(e^{-i\pi ak}-e^{i\pi ak}\right)}{2\pi ak}\\
&= \frac{i\left(cos(\pi ak)-i\cdot sin(\pi ak)-(cos(\pi ak)+i\cdot sin(\pi ak))\right)}{2\pi ak}\\
&= \frac{-ii\cdot sin(\pi ak)-ii\cdot sin(\pi ak)}{2\pi ak}\\
&= \frac{sin(\pi ak)+sin(\pi ak)}{2\pi ak}\\
&= \frac{sin(\pi ak)}{\pi ak}\\
&= sinc(ak\pi)\\
&= B_a(k)
\end{aligned}$$

Thus we have shown the Fourier transform of $b_a(x)$ is $B_a(x) = \frac{1}{ak\pi}sin(ak\pi)$

d, iii

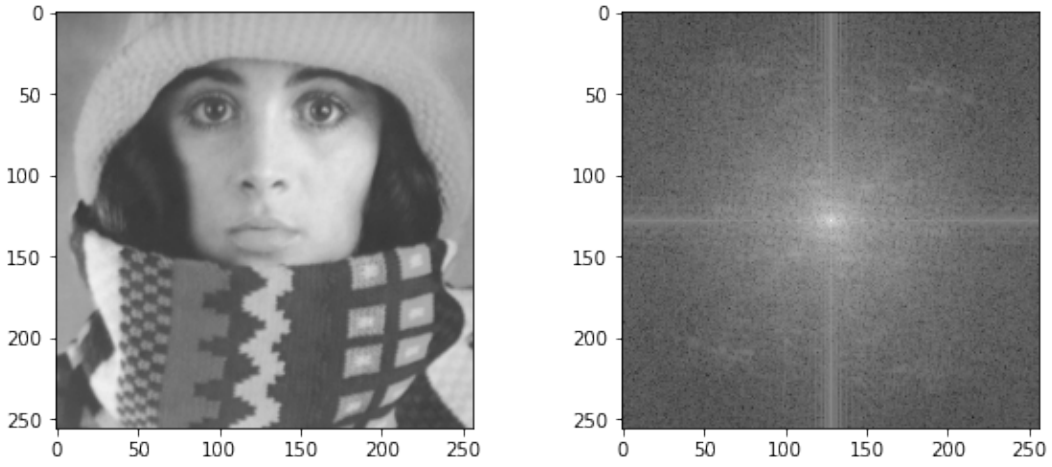
If we first look at $ak\pi$ from $B_a(k)$ we see that $\lim_{a\to 0} ak\pi = 0$. Which then implies $\lim_{a\to 0} \frac{\sin(\pi ak)}{\pi ak} = 1$. As $\frac{\sin(\pi ak)}{=} sinc(ak\pi) = B_a(x)$ we have thus shown $\lim_{a\to 0} B_a(x) = 1$.
For $a \to 0$ the *box* function turns into the $\delta(x)$ function for which the Fourier transform is 1. We have just shown $\lim_{a\to 0} sinc(x) = 1$ which implies that the transform of the $\delta(x)$ is 1. This thus proves the first entry in table 3.2.

d, iv

When a is near zero, the Fourier transform becomes wide and the peaks appear less frequently. This is due to a smaller part of the transform being caught in the integral since we go from $a/2$ to $-a/2$ and we have our limit $\lim_{a\to 0} B_a(k) = 1$, this means that the function must get larger to accommodate in the area for this requirement. The reverse happens when a becomes large. The frequency of peaks increases as a grows. I.e. the distance between peaks decreases as a grows therein showing that a controls the frequency of the function.

Exercise 2

a



(a) Trui Image (b) Power spectrum of trui image

Figure 1

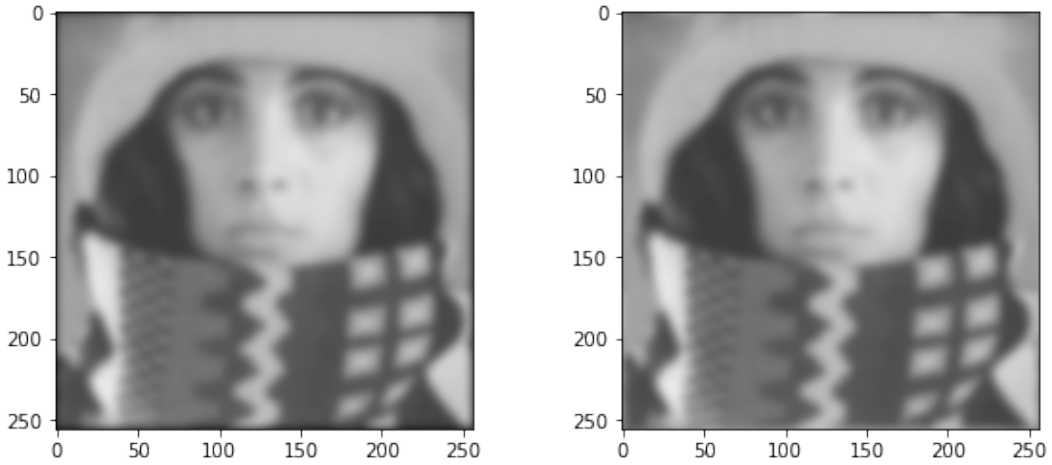
We see in figure 1b a high concentration of low frequencies in the center and from here we have a cross like beam emanating. The center is where the overall structure of the image is located, while the beam's might be the frequencies encoding the pattern seen on the hood in the image.

```
C = np.copy(A) # Copy image
fft = scipy.fft.fft2(C)
F = np.absolute(fft)
powerF = 8*np.log(1+F*F)
shiftedPowF = scipy.fft.fftshift(powerF)
showG(shiftedPowF)
```

Figure 2: Code used to convert image to power spectrum

After having applied the power spectrum to the trui image. We were having problems with the resulting spectrum being so incredible dark. To solve this we would multiple by a scalar that brought out the details of the spectrum. In our case that was 8.

b

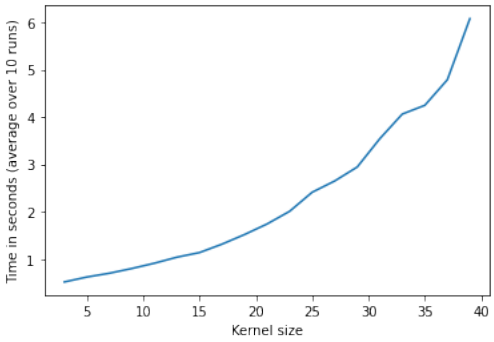


(a) Classical for loop approach

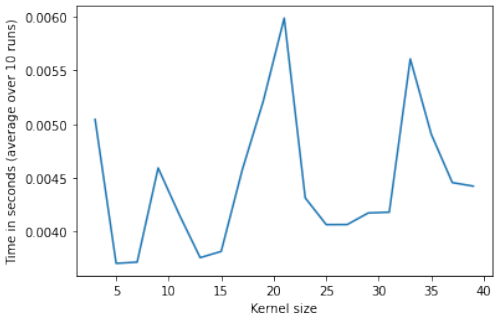
(b) Fast Fourier Transform

Figure 3

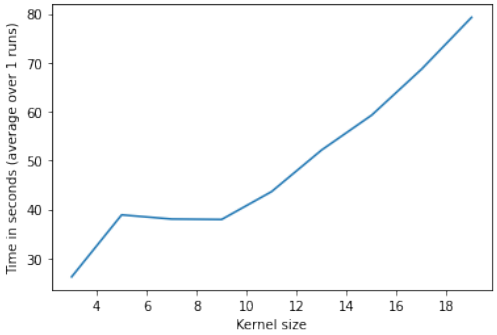
The results of applying both convolutions is the same, as seen in Figure 3, except for the padding having leaked into the image when using the loop approach.



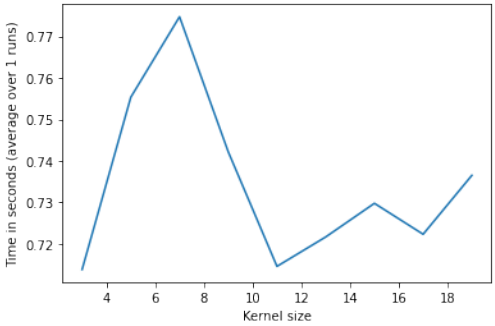
(a) Classical for loop of Gaussian kernel on trui image. Average over 10 runs per kernel size.



(b) Fast Fourier Transform on turi, averaged over 10 runs per kernel size.



(c) Classical for loop of Gaussian kernel on fruit image. Only run 1 time per kernel size due to very slow speed.



(d) Fast Fourier Transform on fruit, run only once per kernel size for consistency with the classical apprach.

Figure 4

In Figure 4 we see that even though the results are practically the same, the difference in speed of the convolution is massive. Where the Fast Fourier Transform only takes a couple of milliseconds to complete its convolution it takes many seconds for the looping approach to complete its convolution. We also see that the simple looping approach scales linearly (maybe exponentially) with the size of the kernel, while the Fast Fourier Transform has almost no change when the kernel size is increased.

```
def conv(img1, k):
    n,m = img1.shape
    out = np.zeros((n,m))
    x,y = k.shape
    offset = x//2
    pim = np.pad(img1,offset,mode='constant')
    for i in range(n):
        for j in range(m):
            A = pim[(i+offset) - (y//2):(i+offset) + (y//2+1),(j+offset) - (x//2):(j+offset) + (x//2+1)]
            A = A * k
            out[i,j] = np.sum(A)
    return out
```

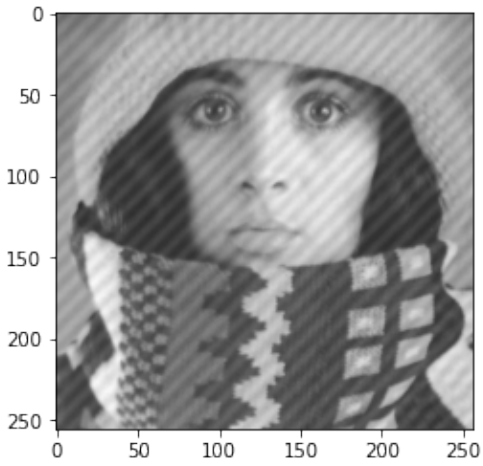
Figure 5: Code for applying guassian kernel to image with for loop

```python
def ftconv(img,k):
    sz = (img.shape[0] - k.shape[0], img.shape[1] - k.shape[1])
    k = np.pad(k, (((sz[0]+1)//2, sz[0]//2), ((sz[1]+1)//2,sz[1]//2)))
    k = scipy.fft.ifftshift(k) # Inverse shift of kernel for some reason is needed.
    fft = scipy.fft.fft2(img)
    fft_filtered = fft * scipy.fft.fft2(k)
    f = scipy.fft.ifft2(fft_filtered)
    return np.real(f)
```

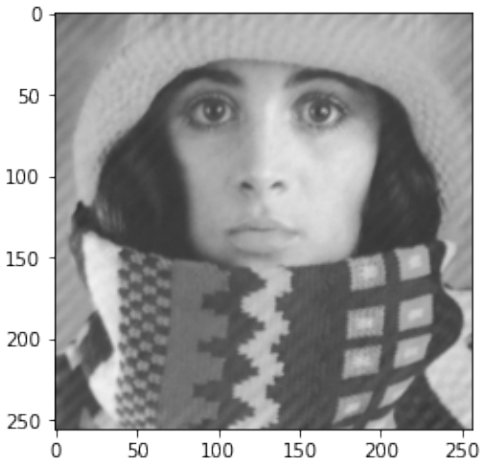
Figure 6: Code for applying Fast Fourier Transform to image

The main issue we ran into when implementing the fast Fourier transform function, was that we needed to do the inverse shift of the convolution before multiplying it with the transformed image. We are not fully sure, but we believe that this is mainly due to the kernel needing to be represented in the same space as the image.

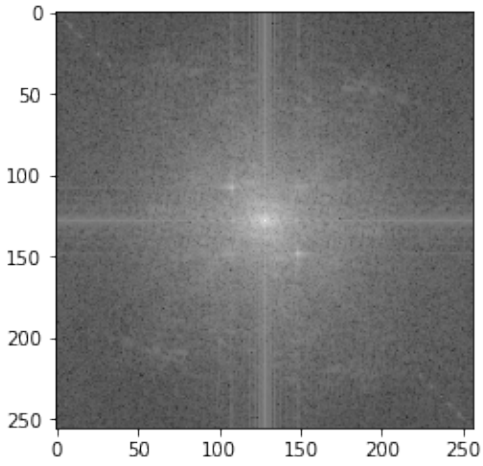
c



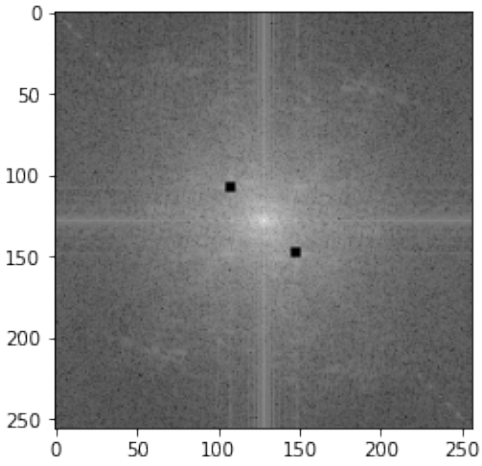
(a) Wave noise applied to trui with $a = 10$, $w = 0.4$ and $v = 0.8$.



(b) Trui image after having applied function to remove noice.

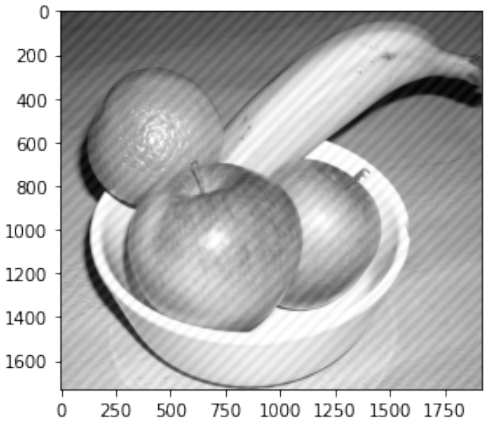


(c) Power spectrum of trui with wave noise

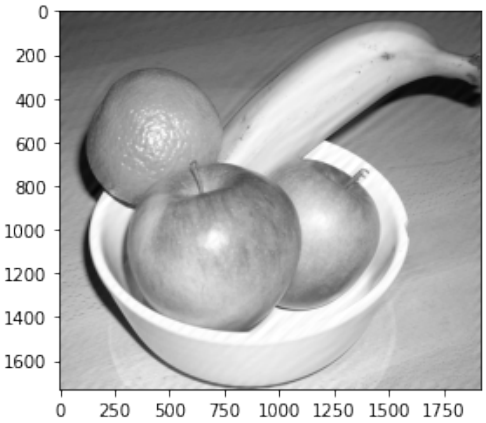


(d) Power spectrum of trui filtered out wave noise

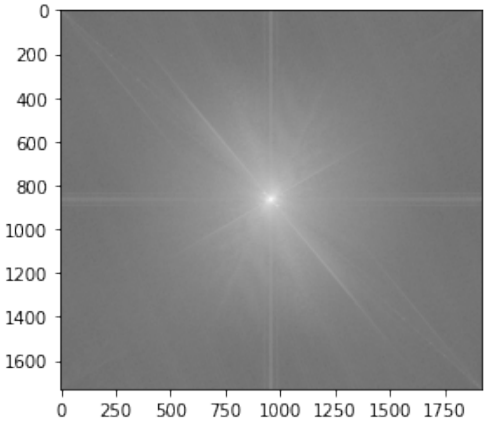
Figure 7



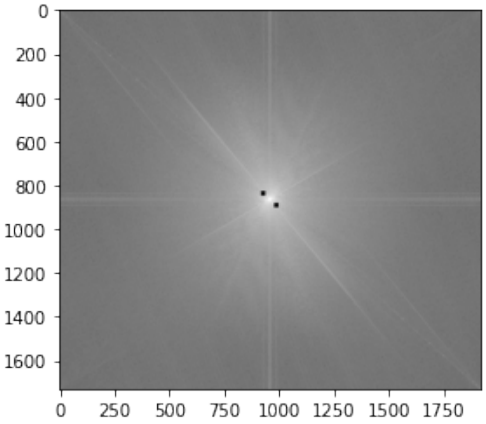
(a) Wave noise applied to fruit with $a = 10$, $w = 0.1$ and $v = 0.1$.



(b) Fruit image after having applied function to remove noice.



(c) Power spectrum of fruit with wave noise



(d) Power spectrum of fruit filtered out wave noise

Figure 8

Given the function:

$$f(x) = Acos(2\pi u_0 x)$$

We find the Fourier transform:

$$F(u) = \frac{A}{2}\delta(u - u_0) + \frac{A}{2}\delta(u + u_0)$$

When we add $f(x)$ to all pixels in an image, and we then look at its Fourier spectrum we see specularities (due to the dirac delta function) at $-u_0$ and u_0 as these values shifts the dirac delta functions.
This generalises to this exercise as we use $f(x) = a_0 cos(v_0 x + w_0 y)$. To find the location of the specularities in the Fourier spectrum we thus need to divide v_0 and w_0 by 2π (as we need the u_0 'part' of v_0 and w_0). This gives us a small number which we then can multiply by the number of pixels along the corresponding axis in the image, and thus find the 'plus' coordinate for the first specularity. The other specularity is found by subtracting the size of the image and the coordinate. A concrete example would be results of Figure 4a, where we add $40 \cdot cos(0.4x + 0.8y)$.

The image is $255x255$ pixels. We find the x coordinates as $(0.4/2\pi) \cdot 255 = 32$ and at $255 - 32 = 223$. Likewise we can compute the y coordinates. We then simply draw a black box over the specularities and use the inverse Fourier transform to get an almost noise free image.

```
def removeWaveNoise(img, w, v, offset=10):
    n,m = img.shape

    fft = scipy.fft.fft2(img)

    fft_show = fftviz(fft)

    x = int((v/(2*np.pi)) * m)
    y = int((w/(2*np.pi)) * n)

    fft[max(y-offset, 0):min(m,y+offset),max(x-offset,0):min(x+offset,n)] = 0
    fft[max(n-y-offset,0):min(n,n-y+offset),max(0,m-x-offset):min(m,m-x+offset)] = 0

    fft_show = fftviz(fft)

    return fft_show
```

Figure 9: Code for removing wave noice from image