Signal and Image Processing
Assignment 3

Nikolaj Dybdahl Rathcke (rfq695)

February 25, 2016

1 Fourier transform - Theory

1.1 (a)

Fourier series is a way to represent periodic functions by a, potentially infinite, sum of complex exponentials (namely sine and cosine functions) of discrete frequencies. The fourier transform is the extension of this to non-periodic function as well, but it transforms the signal from the time domain into a frequency domain, by breaking it into the frequencies that exists in the signal.

1.2 (b)

The fourier transform of $f(x)$ is given by:

$$F(k_x) = \frac{1}{2\pi}\int_{-\infty}^{\infty} f(x)e^{-ik_x x}dx$$

We want to show that the transformation of $f(x)$ when it is real and even is also real and even. Euler's formula states that for any real x:

$$e^{ix} = \cos(x) + i\sin(x)$$

Since the function is real, this means $i = 0$ and we can say:

$$F(k_x) = \frac{1}{2\pi}\int_{-\infty}^{\infty} f(x)\cos(k_x x)dx$$

This means the integral over $f(x)cos(k_x x)$ must be real as it is the product of two real functions. Now, we prove that if $f(x)$ is even, then so is the transformation. If a function is even, the following holds:

$$g(x) = g(-x)$$

We want to prove this for when the fourier transform is the function g. We let $u = -x$ and $dx = -du$. We can then write:

$$\begin{aligned} F(-k_x) &= \frac{1}{2\pi}\int_{-\infty}^{\infty} f(-u)e^{-i(-k_x)x}du \\ &= \frac{1}{2\pi}\int_{-\infty}^{\infty} f(-u)e^{ik_x x}du \\ &= F(k_x) \end{aligned}$$

these two derivation prove that the fourier transformation of a real and even function is also real and even.

1.3 (c)

Again we use the fourer transform of $f(x)$ where $f(x) = \delta(x-d) + \delta(x+d)$, so

$$F(k_x) = \frac{1}{2\pi}\int_{-\infty}^{\infty} (\delta(x-d) + \delta(x+d))e^{-ik_x x}dx$$

We can split this into two integrals:

$$F(k_x) = \frac{1}{2\pi}\left(\int_{-\infty}^{\infty} \delta(x-d)e^{-ik_x x}dx + \int_{-\infty}^{\infty} \delta(x+d)e^{-ik_x x}dx\right)$$

Now the shifting property, which states that $\int_{-\infty}^{\infty} f(x)\delta(x-d)dx = f(d)$, allows us to write:

$$F(k_x) = \frac{1}{2\pi}\left(e^{-ik_x d} + e^{-ik_x(-d)}\right)$$

We can simplify this further by rewriting the complex exponentials using Euler's formula:

$$\begin{aligned}
F(k_x) &= \frac{1}{2\pi}\left(\cos(k_x d) - i\sin(k_x d) + \cos(-k_x d) - i\sin(-k_x d)\right) \\
&= \frac{1}{2\pi}\left(2\cos(k_x d) - i(\sin(-k_x d) + \sin(k_x d))\right) && \text{(As cosine is even)} \\
&= \frac{1}{2\pi}\left(2\cos(k_x d)\right) && \text{(As sine is odd)} \\
&= \frac{\cos(k_x d)}{\pi}
\end{aligned}$$

which is the derivation of the continuous fourier transform of $\delta(x-d) + \delta(x+d)$ for some d.

1.4 (d)

1.4.1 (i)

We can easily show the integral is 1. The function is a constant function taking value $1/a$ for $x = -a/2$ to $a/2$. This is, as the name implies, a box of with length $2 \cdot 2/a$ (both sides of the $y - axis$ and height $1/a$. This yields an area (which is what the integral is) of

$$2 \cdot \frac{a}{2} \cdot \frac{1}{a} = \frac{2a}{2a} = 1$$

And we have shown the integral of the box function is 1.

1.4.2 (ii)

We have that the fourier transform is given by:

$$F(k_x) = \frac{1}{2\pi}\int_{-\infty}^{\infty} f(x)e^{-ik_x x}dx$$

Since the function only takes values other than 0 in the interval $[-a/2, a/2]$, we can write $B(k)$ as:

$$B(k) = \frac{1}{2\pi}\int_{-a/2}^{a/2} f(x)e^{-ikx}dx$$

We can now insert the the box function:

$$\begin{aligned}
B(k) &= \frac{1}{2\pi}\int_{-a/2}^{a/2} \frac{1}{a}e^{-ikx}dx \\
&= \frac{1}{2\pi a}\int_{-a/2}^{a/2} e^{-ikx}dx \\
&= \frac{1}{2\pi a}\int_{-a/2}^{a/2} \cos(kx) - i\sin(kx)dx \\
&= \frac{1}{2\pi a}\frac{2\sin(\frac{ak}{2})}{k} && \text{(As the definite integral is } \tfrac{2\sin(\frac{ak}{2})}{k}\text{)} \\
&= \frac{1}{\pi ak}\sin\left(\frac{ak}{2}\right)
\end{aligned}$$

Which is what we wanted to show. To rewrite $B(k)$ using `sinc`, we want to produce $\frac{ak}{2}$ in the denominator. So,

$$\begin{aligned} B(k) &= \frac{1}{\pi ak}\sin\left(\frac{ak}{2}\right) \\ &= \frac{1}{\pi}\frac{\sin(\frac{ak}{2})}{ak} \\ &= \frac{1}{\pi}\frac{\sin(\frac{ak}{2})}{\frac{ak}{2}}\frac{1}{2} && \text{(Extending fraction with 1/2)} \\ &= \frac{1}{2\pi}\text{sinc}\left(\frac{ak}{2}\right) && \text{(Using sinc)} \end{aligned}$$

Which is $B(k)$ expressed with a `sinc` term.

1.4.3 (iii)

We want to show that $B(k)$ approached $\frac{1}{2\pi}$ when a approaches infinity. By definition, the sinc term approaches 1 when $\frac{ak}{2}$ approaches zero. As we let $a \to 0$, it is indeed true that the fraction approaches 0, thusi we can replace the sinc term by 1:

$$\begin{aligned} \lim_{a\to 0} B(k) &= \frac{1}{2\pi}\text{sinc}\left(\frac{ak}{2}\right) \\ &= \frac{1}{2\pi} \end{aligned}$$

Which is what we wanted to show. This proves the first entry in Table 5.2. As we let a approach 0, the box's length will also approach 0, which correspond to the delta dirac function since the integral over the function remains 1.

1.4.4 (iv)

A function and it's fourier transform cannot both have compact support as implied by the Paley-Wiener theorem. This is because the fourier transform of a function with compact support is an entire function, which in return means that the fourier transform can not have compact support.

2 Fourier Transform - Practice

2.1 (a)

The code for this question is seen below:

```
function q2_1()
    I = imread('lena.tiff');
    ps = 10*log10(fftshift(abs(fft2(I)).^2));
    subplot(1,2,1);
        imagesc(I);
        title('Original image');
        colormap gray;
    subplot(1,2,2);
        imagesc(ps);
        title('Power spectrum of the image');
        colormap gray;
end
```

The function `fft` is the fast fourier transformation in Matlab, which splits the signal into frequencies. The function `fftshift` shifts by moving the zero-frequency component to the center, which is useful four visualizing a fourier transform. Since the resulting power spectrum is pretty obscure, the logarithm is taken of the spectrum, which gives a more interpretable spectrum as the difference between high and low frequencies is clearer. Figure 1 shows the original image along with its resulting power spectrum.

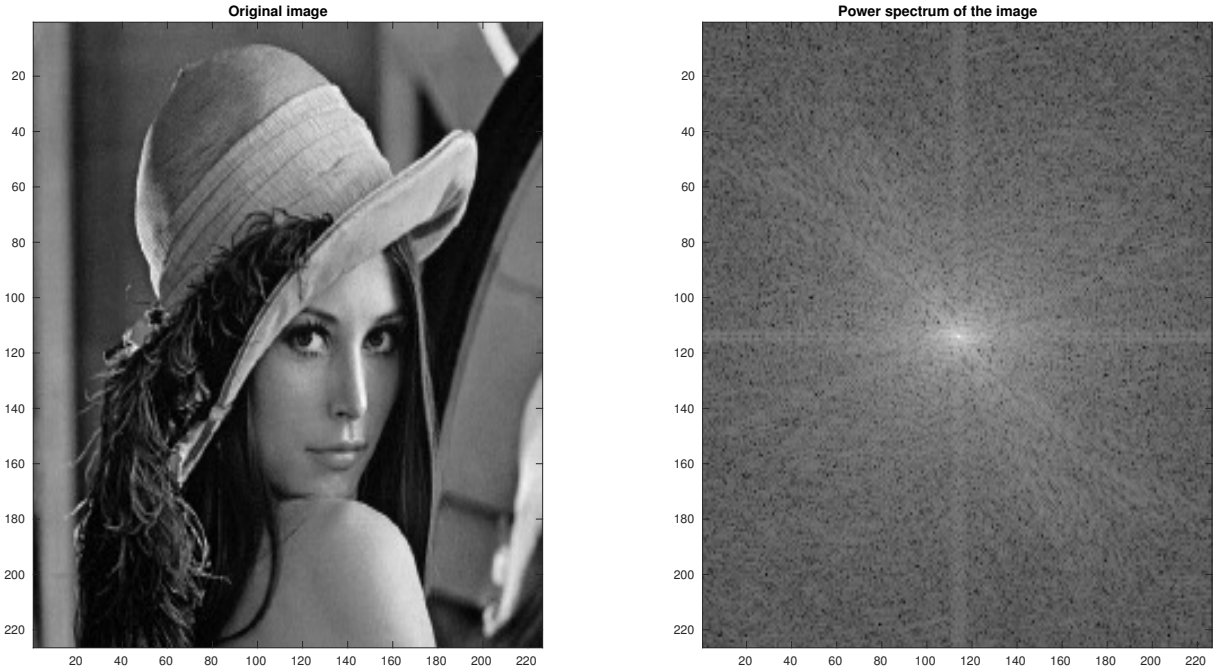

Figure 1: Figure showing the original image `lena.tiff` and the power spectrum generated from the picture.

The power spectrum is useful for determining what frequencies are dominant. A pixel's distance to the center is an indication of its frequency.
The "plus" of bright light we can see go through the center is because we shift the image, which means it is the edges we can see in the middle. The diagonal line we can see could come from the bottom of the hat. Another thing to note, is that it is very bright in the center, which means the low frequencies are very dominant in the image.

2.2 (b)

The code for this question be seen below:

```
function q2_2()
    %q2_2a()
    %q2_2b()
    q2_2c()
end

function q2_2a()
    I = imread('lena.tiff');
    h = fspecial('gaussian');
    A = nestedConv(h, I);
    B = fourConv(h, I);
    subplot(1, 3, 1);
        imagesc(I);
        colormap gray;
        title('Original image');
    subplot(1, 3, 2);
        imagesc(A);
        colormap gray;
        title('Image after convolution');
    subplot(1, 3, 3);
        imagesc(B);
        colormap gray;
        title('Image after convolution using FFT');
end

function q2_2b()
    I = imread('lena.tiff');
    fst = [];
    snd = [];
    for i = 1:7
        h = fspecial('gaussian', [i*2+1 i*2+1]);
        tic;
        A = nestedConv(h, I);
        fst(i) = toc;
        tic
```

```
        B = fourConv(h, I);
        snd(i) = toc;
    end
    subplot(1, 1, 1);
        plot([3:2:15], fst, [3:2:15], snd);
        xlabel('Window size in n*n');
        ylabel('Computation time in seconds');
        title('Nested for-loop convolution vs. convolution using FFT');
end

function q2_2c()
    I = imread('lena.tiff');
    fst = [];
    snd = [];
    for i = 1:10
        h = fspecial('gaussian');
        I2 = imresize(I, i/10);
        tic;
        A = nestedConv(h, I2);
        fst(i) = toc;
        tic
        B = fourConv(h, I2);
        snd(i) = toc;
    end
    subplot(1, 1, 1);
        plot([0.1:0.1:1], fst, [0.1:0.1:1], snd);
        xlabel('Image size in scale = n/10');
        ylabel('Computation time in seconds');
        title('Nested for-loop convolution vs. convolution using FFT');
end

function res = nestedConv(ker, img)
    x = size(img, 1);
    y = size(img, 2);
    x2 = size(ker, 1);
    y2 = size(ker, 2);
    x3 = fix(x2/2);
    y3 = fix(y2/2);
    res = zeros(x, y);
    for i = 1:x
        for j = 1:y
            acc = 0;
            for k = -x3:x3
                for l = -y3:y3
                    if (i+k > 0 && i+k < x)
                        if (j+l > 0 && j+l < y)
                            acc = acc + ker(k+x3+1, l+y3+1) * img(i+k, j+l);
                        else
                            acc = acc + ker(k+x3+1, l+y3+1) * img(i+k, j-l);
                        end
                    elseif (j+l > 0 && j+l < y)
                        acc = acc + ker(k+x3+1, l+y3+1) * img(i-k, j+l);
                    else
                        acc = acc + ker(k+x3+1, l+y3+1) * img(i-k, j-l);
                    end
                end
            end
            res(i, j) = acc;
        end
    end
end

function res = fourConv(ker, img)
    cumSize = size(img) + size(ker) - 1;
    img(cumSize(1), cumSize(2)) = 0;
    ker(cumSize(1), cumSize(2)) = 0;
    res = ifft2(fft2(img).*fft2(ker));
end
```

The function `nestedConv` performs convolution with nested for loops while the function `fourConv` performs convolution with FFT. By running function `q2_2a`, we get the result in Figure 2.

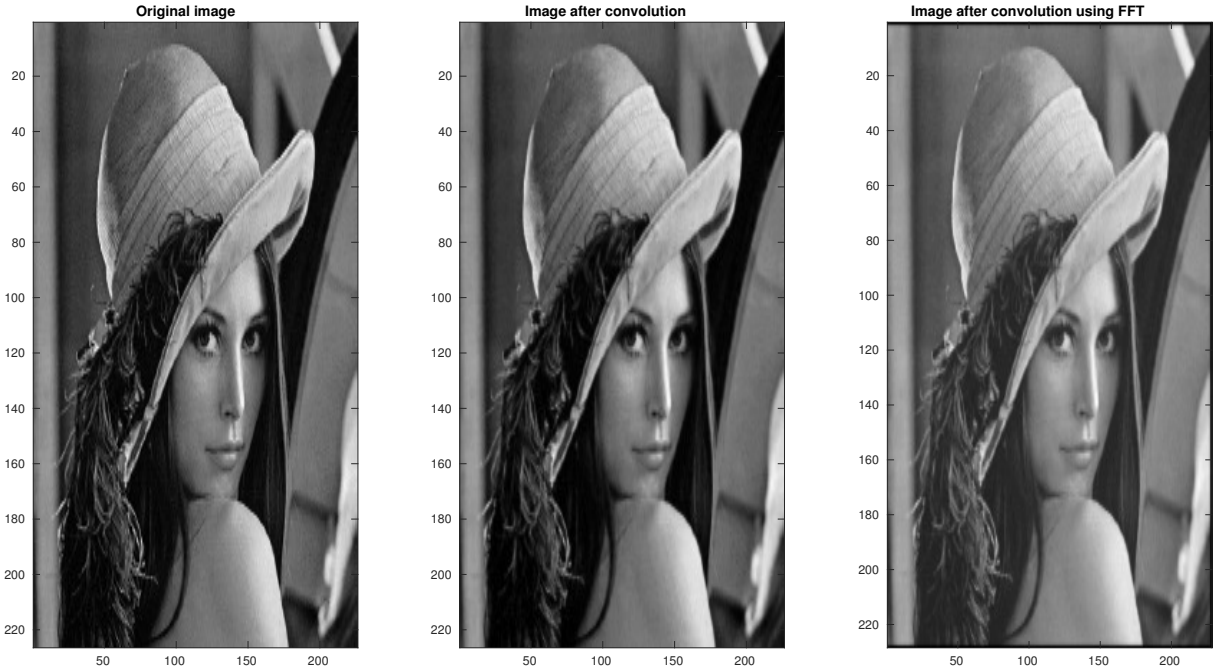

Figure 2: Figure showing the results from using convolution using nested for loops and fast fourier transform.

Both are using a gaussian filter using a window size of 3. Even though it might be a little hard to tell, the nested for loop result in the middle a little blurred as expected, but using FFT brightens the image as well.
Another result we get from running `q2_2b` is shown in Figure 4. It shows the computation time between the two methods depending on the window size of the kernel. For the window size of $n * n$, we use $n = \{3, 5, .., 15\}$.

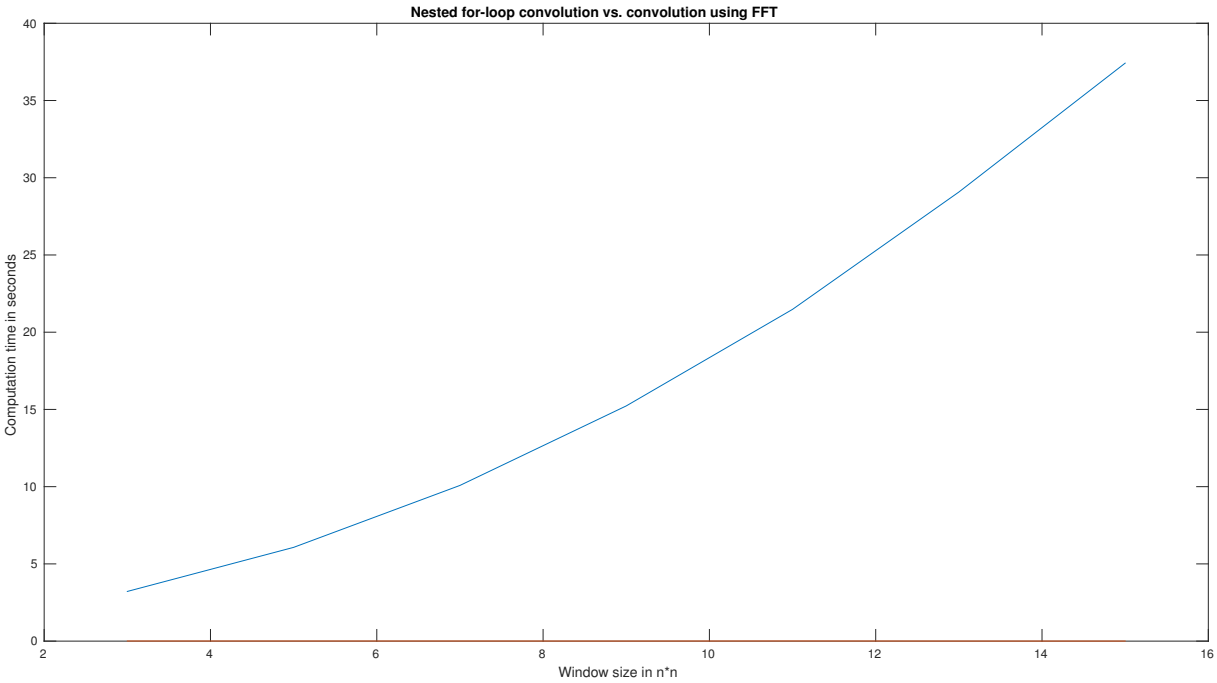

Figure 3: Figure showing the difference in computation time for the two implemented convolution methods for different window sizes of the kernel. The blue line is the nested for loop method and the red line is the FFT method.

The blue line is the nested for loop method and the red line is the FFT method. We can see that FFT is a lot faster as it is always non existant in the plot. No matter what the window size is, it has a computation time under 1 second. The nested for loop method is steadily increasing in computation time for increasing window size.
The last result in Figure **??** shows the difference in computation time for different image sizes. The image `lena.tiff` has been used with scales from 0.1 to 1 (where 1 is the normal size of the image).

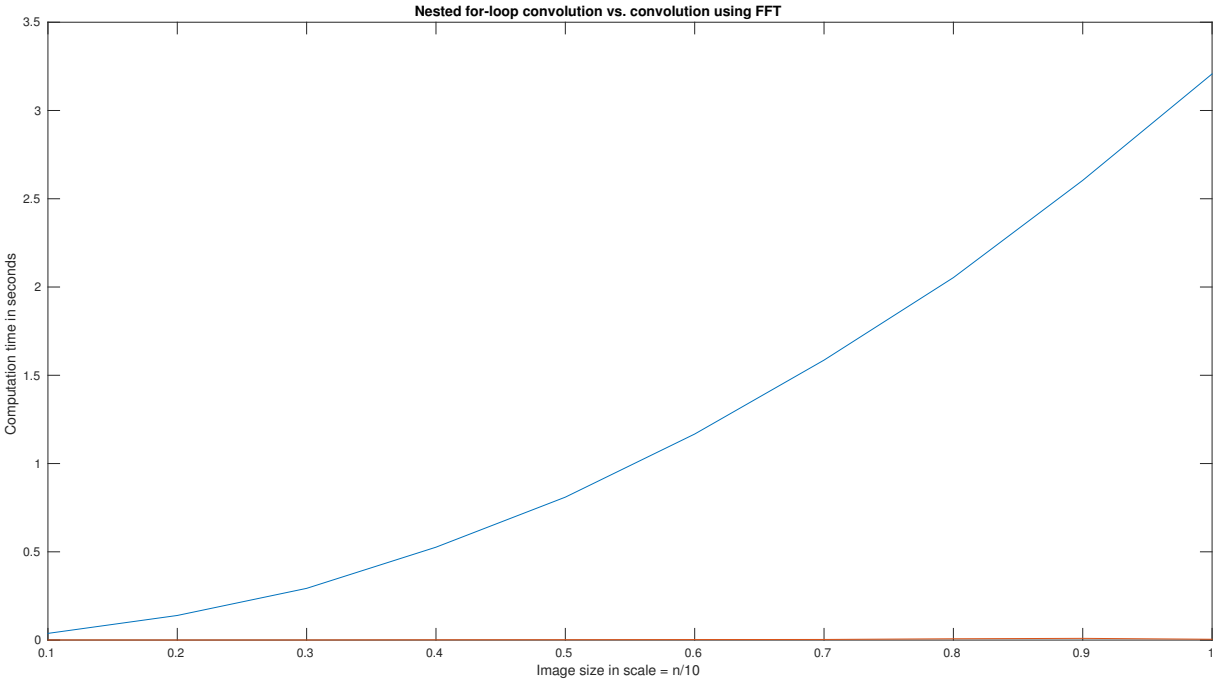

Figure 4: Figure showing the difference in computation time for the two implemented convolution methods for different image sizes. The blue line is the nested for loop method and the red line is the FFT method.

Again the blue line is the nested for loop method and the red line is the FFT method. Once again the FFT method is way faster, while the nested for loop method is increasingsly slow. So despite brightening the image a bit, it is way faster to perform FFT compared to the nested for loop method.

2.3 (c)

The code for this question is seen below:

```
function q2_3()
    I = imread('lena.tiff');
    I2 = addFunc(I, 50, 0.25, 0.25);
    subplot(2,2,1);
        imshow(I);
        title('Original image');
        colormap gray;
    subplot(2,2,2);
        imshow(I2);
        title('Image with added function');
        colormap gray;
    ps = 10*log10(fftshift(abs(fft2(I2)).^2));
    subplot(2,2,3);
        imagesc(ps);
        title('Power Spectrum of modified image');
        colormap gray;
    ps = fftshift(fft2(I2));
    ps(105, 105) = 0;
    ps(123, 123) = 0;
    subplot(2,2,4);
        imagesc(abs(real(ifft2(ps))));
        title('Image after removing cosine noise');
        colormap gray;
end

function res = addFunc(img, a, v, w)
    x = size(img, 1);
```

```
    y = size(img, 2);
    for i = 1:x
        for j = 1:y
            res(i, j) = img(i, j) + a*cos(v*i+w*j);
        end
    end
end
```

The function `addFunc` adds noise depending on parameters a, v and w. Running the code will generate 4 images that can be found in Figure 5 with parameters $a = 50$, $v, w = 0.25$.

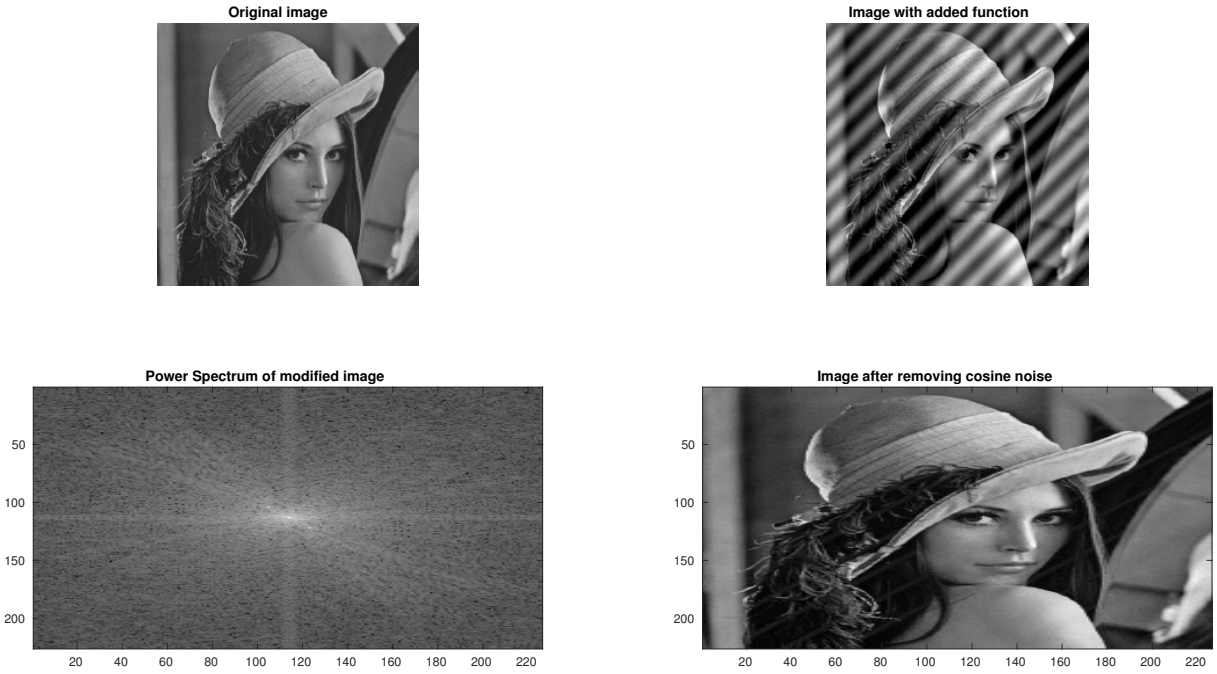

Figure 5: Figure showing the image `lena.tiff` along with the image with added cosinus noise. The two lower images is the power spectrum of the image with cosine noise and the image after an attempt to remove the noise from the image has been made.

The upper left image is the orginal one. The upper right shows the image with added cosine noise. In the lower left, we have a power spectrum of the image with cosine noise. Again the two lines that make a plus are explained by the shifting of the image. But we can also see two very bright spots near the middle, which is the signal of the cosine noise (one for negative values and one for positive values). To remove this noise, a very simple solution exists by changing pixel value for this signal. Doing this results in the image in the lower right. Even though it might seem very clean, there are actually still some grey lines going from the lower left corner to the upper right corner, which means the signal is still somewhat present in the image.

2.4 (d)

The function `scale` is implemented as below:

```
function q2_4()
    for i = 1:6
        res = scale((i-1)*3+1);
        subplot(2, 3, i);
            imshow(res);
            title(sprintf('With sigma = %d', (i-1)*3+1));
    end
end

function res = scale(arg)
    I = imread('lena.tiff');
    h = fspecial('gaussian', [arg arg], arg);
    res = imfilter(I, h);
end
```

The first function `q2_4` is responsible for plotting, while the `scale` function is implemented such that it take an argument (sigma), and performs gaussian filtering on the image `lena.tiff`. Since using a fixed

window size for the kernel would result in almost identical images, the window size is set to the value of sigma as well. This resulted in Figure 6.

Figure 6: Figure showing the difference of sigma and window size when performing gaussian filtering on the image `lena.tiff`.

As expected, the image becomes more blurry as neighbouring pixels are weighted higher, and as such, they have a larger impact on the pixel we are looking at.

2.5 (e)

We want to show that the spatial derivatives of a function $f(x)$ can be written as a multiplication of a kernel in the fourier domain. We have the fourier transform of f is given by:

$$F(k_x) = \frac{1}{2\pi} \int f(x) e^{-ik_x x} dx$$

We now have that

$$\begin{aligned} F(f')(k_x) &= \frac{1}{2\pi} \int f'(x) e^{-ik_x x} dx \\ &= \frac{1}{2\pi} \int ik_x f(x) e^{-ik_x x} dx \end{aligned}$$

Since the derivative of the exponential simply taking the constants in the power down. Since these are constants we can write:

$$\begin{aligned} F(f')(k_x) &= \frac{1}{2\pi} \int ik_x f(x) e^{-ik_x x} dx \\ &= ik_x \cdot \frac{1}{2\pi} \int f(x) e^{-ik_x x} dx \\ &= ik_x \cdot F(f)(k_x) \end{aligned}$$

which is what we wanted to show. This relation can be used when we want to find more than one order of derivatives, since it is a lot easier and faster to find derivates this way than to compute them in a traditional way every time.

2.6 (f)

To generalize on the derivation we made in (2.5), we can see that the derivative of order n can be found by just multiplying with the constants n times, i.e. we can find the derivative of order 2 by $ik_x^2 \cdot F(f)(k_x)$ in the derivation made above. When we want to take the derivative of an image, we are not looking at a function $f(x)$, but a function $f(x, y)$. The derivation we made above still applies, which means what we want to calculate is actually

$$ik_x \cdot ik_y F(f(k_x, k_y))$$

For order 1, and

$$(ik_x)^n \cdot (ik_y)^m F(f(k_x, k_y))$$

For two given order n and m. This calculation has been implemented in Matlab as seen below:

```
function q2_6()
    I = imread('lena.tiff');
    for i = 0:2
        for j = 0:2
            A = deriveImg(I, i, j);
            subplot(3, 3, 3*i+j+1);
            imagesc(A);
            colormap gray;
            title(sprintf('With n = %d and m = %d', i, j));
        end
    end
end

function res = deriveImg(img, n, m)
    ft = fft2(img);
    x = size(ft, 1);
    y = size(ft, 2);
    for i = 1:x
        for j = 1:x
            ftd(i, j)= (1j * i).^n *(1j * j).^m * ft(i, j);
        end
    end
    res = real(ifft2(ftd));
end
```

Note that $1j$ correspond to the imaginary part in Matlab. The function `deriveImg` is the function that makes the above calculation, while `q2_6` simply make the plot with different parameters. The image `lena.tiff` has been used and the derivative of the image with varying derivative orders can be seen in Figure 7.

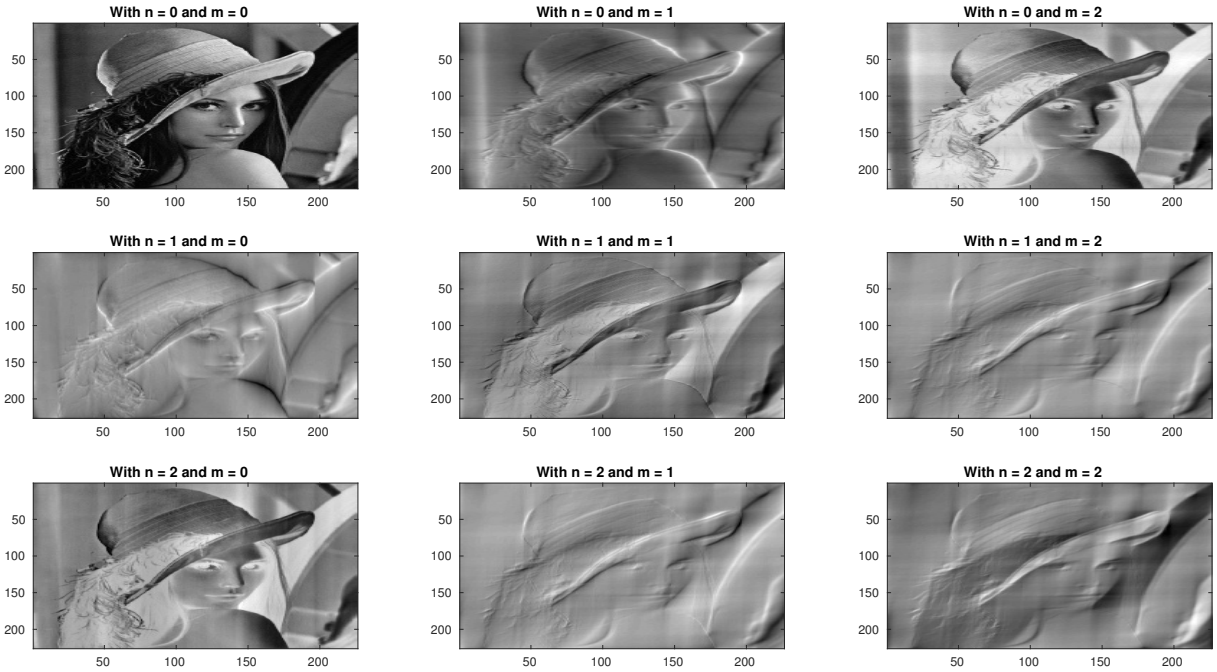

Figure 7: Figure showing the derivative of the image `lena.tiff` with different derivative orders.

We can see that a difference in the order of the two derivatives seem to negate the image. It is hard to say anything about the rest, but the higher the order, the plainer it looks. However, it seems that it can be more similar if we increase the parameters by the last image.