Signal and image processing: Assignment 3

Mariuta Nicolae 18-09-2014

Exercise 1.a.

Fourier Series express a periodic signal as a weighted combination of sine and cosine functions that have different periods or frequencies. The Fourier Transform breaks down a nonperiodic signal into harmonic functions of continuously varying frequencies.

- Fourier Transform is like the limit of the Fourier Series of a function with the period approaches to infinity
- Fourier Series coefficients of a periodic function are sampled values of the Fourier transform of one period of the function

Exercise 1.b.

If the function is even, it means that:
$f(x) = f(-x)$ where *x* is real number. The Fourier transform of $f(x)$ is:

$$F(k) = \int_{-\infty}^{\infty} f(x)e^{-ixk}dx$$

We can replace the *x* with *–y* and if we calculate $F(-k)$ then we will have :

$$F(-k) = \int_{-\infty}^{\infty} f(-y)e^{+iyk}(-dy)$$
$$= \int_{-\infty}^{\infty} f(-y)e^{+iyk}(dy)$$

Because $f(x) = f(-x)$ it means that $F(-k)$ = $F(k)$ so the Fourier Transform of real and even function is a real number.

Exercise 1.c.

In case of function : *f(x) = δ(x − d) + δ(x + d)*, the Fourier transform is:

$$F(k) = \int_{-\infty}^{\infty} f(x)e^{-i\omega x}dx$$
$$= \int_{-\infty}^{\infty} \big(\delta(x - d) + \delta(x + d)\big)e^{-i\omega x}dx$$

$$F(k) = \int_{-\infty}^{\infty} \delta(x - d)e^{-i\omega x}dx + \int_{-\infty}^{\infty} \delta(x + d)e^{-i\omega x}dx$$
$$= e^{-i\omega d} + e^{i\omega d}$$

This is because the Fourier Transform of shifted impulse is exponential. If d = 0, then the transform of impulse function is 1 as in the Table 5.2 entry for impulse function in the course book. Now, if I turn the exponential complex numbers to polar form, then the transform becomes:

$$F(k) = \cos(-\omega d) + isin(-\omega d) + \cos(\omega d) + isin(\omega d)$$

Because of that cosines function is even(*cos(-x) = cos(x)*) and *sin(-x) = -sin(x)* the Fourier Transform becomes:

$$F(k) = \cos(\omega d) - isin(\omega d) + \cos(\omega d) + isin(\omega d) = 2\cos(\omega d)$$

I can conclude that the Fourier transform of a function represented by 2 diracs that are symmetrically shifted around the y axis is $2\cos(\omega d)$ where *d* is the distance from the ordinate.

Exercise 1.d.i

In order to demonstrate that $\int_{-\infty}^{\infty} b_a(x)dx = 1$ I first consider that the box function is 0 until $-\frac{a}{2}$ and also 0 after $\frac{a}{2}$ so the integral can be easily simplified by writing

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

Between $-\frac{a}{2}$ and $\frac{a}{2}$ the function is $\frac{1}{a}$ and that is why I replaced b_a with $\frac{1}{a}$. The constant goes in front of the integral and integral of *1* is *x*. After replacing *x* with the limits of the function and doing simple calculations the result is 1.

Another easy way to demonstrate this is to calculate the area in the graphic of the function. The function is a box, so a rectangle with height of $\frac{1}{a}$ and width of $\frac{2a}{2}$ and calculating the area of this rectangle we get the same result: *1*.

Exercise 1.d.ii

First of all, I will demonstrate that $B(k) = \frac{1}{ak\pi}\sin\frac{ak}{2}$. The Fourier transform of the function is:

$$B(k) = \frac{1}{2\pi}\int_{-\infty}^{\infty} b_a(x)e^{-ikx}dx = \frac{1}{2\pi}\int_{-\frac{a}{2}}^{\frac{a}{2}}\frac{1}{a}e^{-ikx}dx = \frac{1}{2\pi a}\int_{-\frac{a}{2}}^{\frac{a}{2}} 1\,e^{-ikx}dx = \frac{-1}{2\pi aik}e^{-ikx}$$

The reasons why I did like that is simple: as in the first part of the exercise, there is no reason to consider the function outside the box. After getting the constant $\frac{1}{a}$in front of the integral, I calculated the integral of the result. In next stage, I will replace *x* from the power of *e* and then I will write the polar form of the complex number:

$$B(k) = \frac{-1}{2\pi aik}\left(e^{-ik\frac{a}{2}} - e^{-ik\frac{-a}{2}}\right)$$
$$= \frac{-1}{2\pi aik}\left(\cos\left(-\frac{ka}{2}\right) + isin\left(-\frac{ka}{2}\right) - \cos\left(+\frac{ka}{2}\right) - isin\left(+\frac{ka}{2}\right)\right)$$
$$= \frac{-1}{2\pi aik}\left(-2isin\left(+\frac{ka}{2}\right)\right)$$
$$= \frac{1}{ak\pi}\sin\frac{ak}{2}$$

By using the properties of cos and sin that I used at exercise 1.c and simplifications of the result, I can demonstrate that $B(k) = \frac{1}{ak\pi}\sin\frac{ak}{2}$

In the next stage, I will turn the Fourier transform of the box function into *sinc*:

$$B(k) = \frac{1}{ak\pi}\sin\frac{ak}{2} = \frac{\sin\frac{ak}{2}}{ak\pi} = \frac{\sin\frac{ak}{2}}{\frac{ak}{2}2\pi}$$

$$= \frac{1}{2\pi}\frac{\sin\frac{ak}{2}}{\frac{ak}{2}}$$

$$= \frac{1}{2\pi}\ sinc(\frac{ak}{2})$$

In my solution, I considered $\frac{ak}{2}$ to be the x from the $sinc(x) = \frac{sinx}{x}$. In order to do this, I ensured that I do not change the shape of the function so I got the parameter $\frac{1}{2\pi}$ in front of the *sinc*.

Exercise 1.d.iii

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

$$= \lim_{a\to 0}\frac{1}{2\pi}\frac{\sin\frac{ak}{2}}{\frac{ak}{2}} = \frac{1}{2\pi}$$

In order to demonstrate that $\lim_{a\to 0} B(k) = \frac{1}{2\pi}$ I used the *sinc* form obtained at (*ii*) for the Fourier transform of the box function and the property that $\lim_{x\to 0}\frac{sinx}{x} = 1$

The results I obtained so far for the Fourier transform of the $b_a$function prove the entry for Rectangle function in the 5.2 table. The result I obtained at (*ii*) looks much like the Fourier transform of the Rectangle function.

Exercise 1.d.iv

I have written the following Matlab code:

```
%%
%sinc
w = 10; h = 5;
x =[-w:0.1:w];
x = x/h;
plot(x,(1/(2*pi))*sinc(x));title('sinc');xlabel('x');ylabel('y'
```

Fig. 1. Matlab code for plotting the Fourier Transform of the box function with different widths and heigths

After doing several tests, I concluded that for a constant height of the box but changing the width, I get wider *sinc* form for smaller values but very compact filter for larger values. But for a constant width and increasing the height of the box function, I get a larger filter, which becomes very close to being a constant function for very large values. But for smaller heights I get more compact filters.

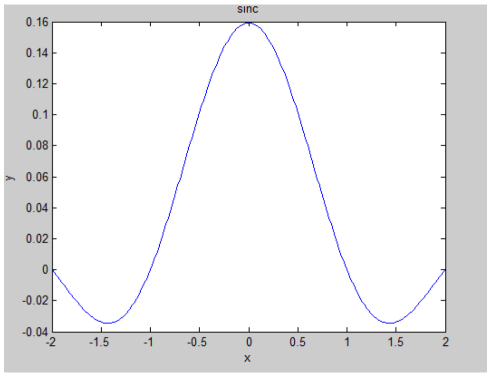

Fig. 2. Fourier transform of box function for *w=10* and *h=5*

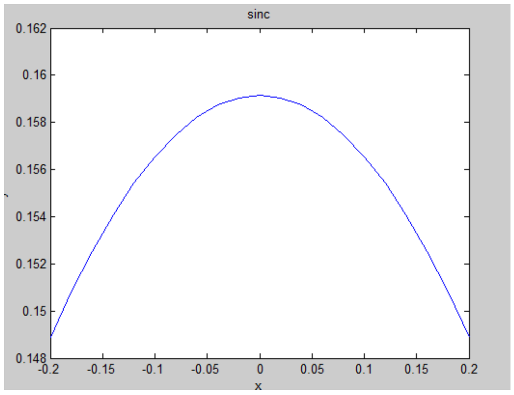

Fig. 3 Fourier transform of box function for *w=1* and *h=5*

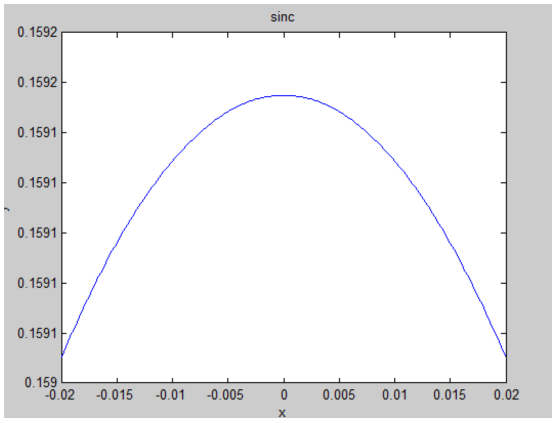

Fig. 4 Fourier transform of box function for *w=10* and *h=500*

Exercise 2.a

```
%%
%2.1 Power spectrum
B = imread('D:\master\image processing\week3\lena.tiff');
subplot(1,2,1),imshow(B);

NP = abs(fftn(B));
NPshift = fftshift(NP);
subplot(1,2,2),imagesc(log(1+NPshift));
```

Fig. 5. Code for calculating and showing the power spectrum of the *lena.tiff* image

About the code, I have at first loaded the *lena.tiff* image found on the Internet and calculated the Fourier Transform using Matlab implementation. I also applied the absolute function to calculate the power spectrum.

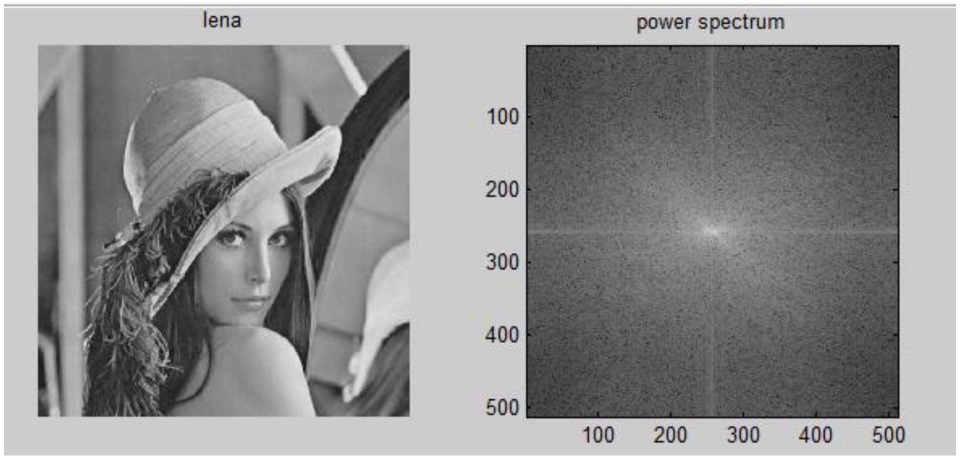

Fig. 6 Greyscale lena image and the results for calculating the power spectrum

After that I have shifted the image using *fftshift* function of Matlab and then shown the result after applying algorithm for all the pixels. The reason for doing this is that in the middle of the resulting image is a small number of pixels that are white and they cannot be seen otherwise.

The result shows that in the middle, close to origins the Euler number with 0 exponent of the Fourier transform becomes 1 so the Fourier transform is basically the integral of the image, which is a very large number (a sum of the resulting values). The pixels are becoming darker and target to the edges because the exponent has lower values for high values of *x*.

Exercise 2.b

I have chosen to create a function that takes an image and a kernel and then applies the chosen filter (kernel) over the entire image.

```
function B2 = filter( B, kernel )
[Rimage,Cimage]=size(B);
[Rkernel,Ckernel]=size(kernel);

B2 = zeros(Rimage,Cimage);

for R = 1: Rimage-Rkernel
    for C = 1: Cimage-Ckernel
        middle = 0;
        for r = 1: Rkernel
            for c = 1: Ckernel
                middle = middle + double(B((R+r),(C+c)))*double(kernel(r,c));
            end
        end
        B2(R,C)= middle;
    end
end

end
```

Fig. 7 Function that applies the kernel over an image

About the implementation, I read the sizes of both matrices (image and kernel) and then created a new empty matrix with the size of the original image.

After that I have used 4 *for* loops to apply the kernel. The first 2 loops are for translating the kernel over the entire image. I used the subtractions (*Rimage-Rkernel* and *Cimage-Ckernel*) to avoid getting out of the bounds of array and I have considered this to be most optimal way to avoid it. The other 2 loops are for looking inside the filter and multiply the values from the image with values from the filter and then added all these values in parameter middle and changed the value of the pixel in the original image.

After implementing this function, I used it to apply a correlation filter on *lena.tiff* image and then I got both the image and kernel to frequency domain.

```
%2.2
B = imread('D:\master\image processing\week3\lena.tiff');
subplot(2,2,1),imshow(B),title('lena');

kernel = [-1 -1 -1; -1 8 -1; -1 -1 -1];

B2 = filter(B,kernel);

subplot(2,2,2),imagesc(B2),title('B*kernel');

Bfft = fft2(B);
subplot(2,2,3),imshow(Bfft),title('B frequency domain');

kernelfft = fft2(kernel);
kernelfft2 = zeros(Rimage,Cimage);
xpos = 1; ypos = 1;
kernelfft2(xpos:xpos+r-1,ypos:ypos+c-1) = kernel;
kernelfft2 = fft2(kernelfft2);


subplot(2,2,4),imshow(kernelfft2);
K = Bfft.*kernelfft2;
subplot(2,2,4),imagesc(real(ifft2(K))),colormap(gray),title('inverse fft for
B(x)kernel(x)');
```

Fig. 8. Code for applying the correlation kernel over the image and then calculate the Fourier Transfrom of the image and kernel, then multiply them and calculate inverse fft

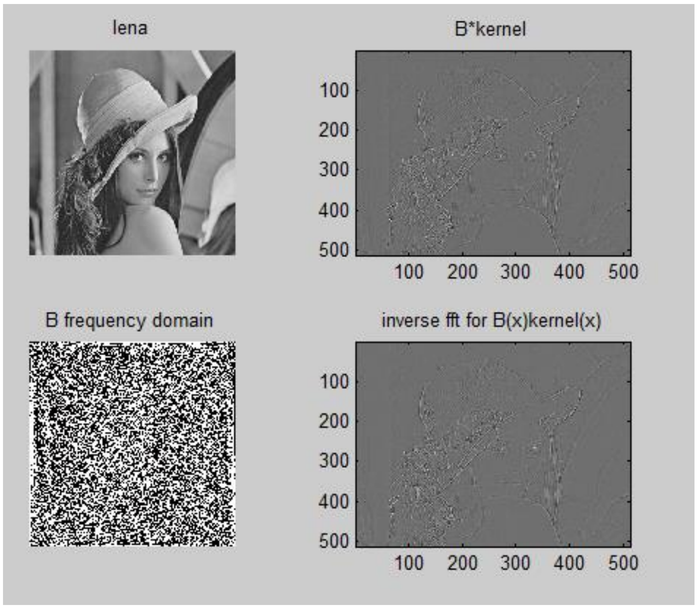

Fig. 9 The results for applying the correlation kernel over the image and then calculate the Fourier Transfrom of the image and kernel, then multiply them and calculate inverse fft

All this experiment has shown that the convolution theorem is correct because the image obtained by convolving the image with a kernel is the same as multiplying the image with the kernel in frequency domain and then calculating the inverse fft.

Another interesting result is obtained by changing the position of the kernel in the frequency domain. That shifts the image because the phase of the kernel in frequency domain is changed.

It is basically the effect that was happening on old TVs that had bad quality signal

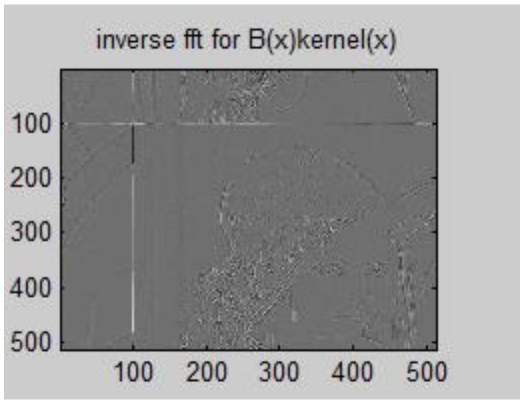

Fig. 10 Changin filter position of the kernel in the frequency domain

Because of the large number of for loops, for increasing the size of kernel the computation time is higher when using large sizes for the filter.

Exercise 2.c

```
%2.3
B = imread('D:\master\image processing\week3\lena.tiff');
subplot(2,2,1),imshow(B),title('lena');


F = zeros(size(B));

a = 255;
v = 1;
w = 5;

for x = 1: size(F,1)
    for y = 1: size(F,2)
        F(x,y) = a*cos(v*x + w*y);
        B(x,y) = B(x,y) + F(x,y);
    end
end

subplot(2,2,2),imshow(B),title('lena+function');

Bfft = fftn(B);

NP = abs(Bfft);
NPshift = fftshift(NP);
subplot(2,2,3),imagesc(log(1+NPshift)),title('power spectrum');

Filter = zeros(size(B));

Ffft = fftn(F);

NP = abs(Ffft);
NPshift = fftshift(NP);
subplot(2,2,4),imagesc(log(1+NPshift)),title('power spectrum');
```

Fig. 11. Code that creates an image with function *a0cos(v0x + w0y)* that is added to lena image and then displayed the power spectrum of the new image and of the function

I have created the image that implements the function *a0cos(v0x + w0y)* so that it will be easy to change and try different values for frequencies. After adding the function to the *lena.tiff* image I have displayed the power spectrum of both images.

If I view the power spectrum of the function I can see 4 white lines corresponding to the harmonics of the cosine function. For the image where I added the cosine function to the image I notice that the lines in the simple cosine function are added to the power spectrum of the image.

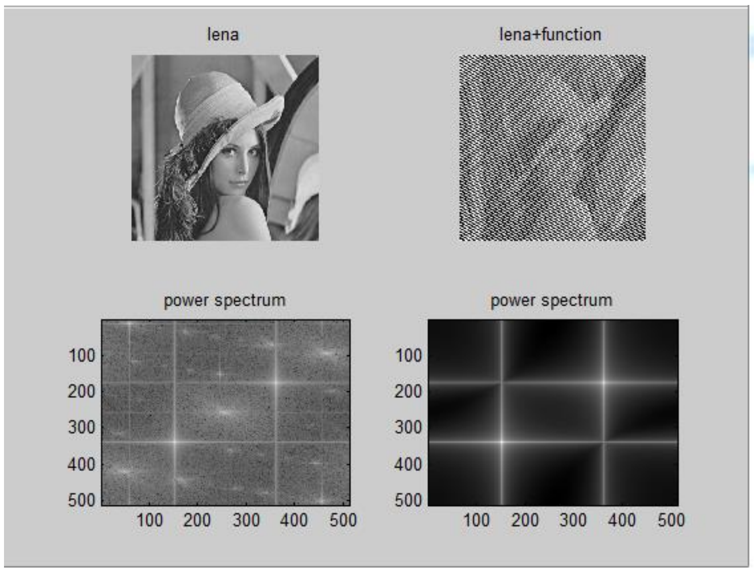

Fig. 12 Results obtained for adding a cosine function to lena image

A filter to remove this noise is very easy to implement using the property called "*addition theorem*" written in table 5.3 from the course book.

```
Ffft = fft2(F);
Bfft = Bfft-Ffft;

NP = abs(Ffft);
NPshift = fftshift(NP);
figure;
imagesc(real(ifft2(Bfft))),title('restored image'),colormap(gray);
```

Fig. 13 Code for removing the cosine function that was added to original image

I obtained the Fourier transform for the function that was added to image in space domain, and using simple subtraction between the transforms of both matrices and applying the inverse Fourier transform I obtained the original image.

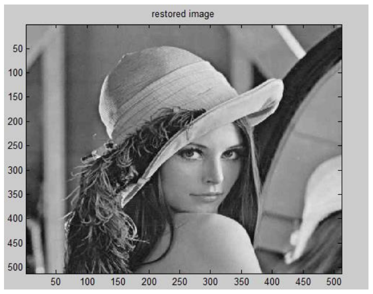

Fig. 14 Image that was restored using the code in Fig. 13

Exercise 2.d

```
function B2 = scale( B,G, m, n )

[x, y]=meshgrid(round(-G/2):round(G/2), round(-G/2):round(G/2));
f=exp(-x.^2/(2*m^2)-y.^2/(2*n^2));
f=f./sum(f(:));

B2 = imfilter(B,f,'same','conv');

end
```

Fig. 15 Function for convolving an image with a Gaussian filter having the standard deviation parameterized

I am not sure if I understood correctly but I think that for this exercise I have to create a Gaussian filter having standard deviation on both direction, *x* and *y* parameterized. I have found on the Internet how to create a Gaussian filter with same deviation on both directions and changed the values. After that I have convolved the filter over the entire image.

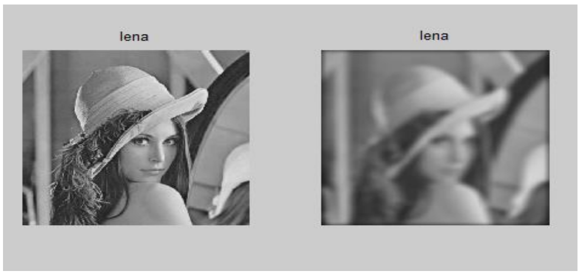

Fig. 16 I have applied scale (Im,[30 20], 10, 6]) on *lena*

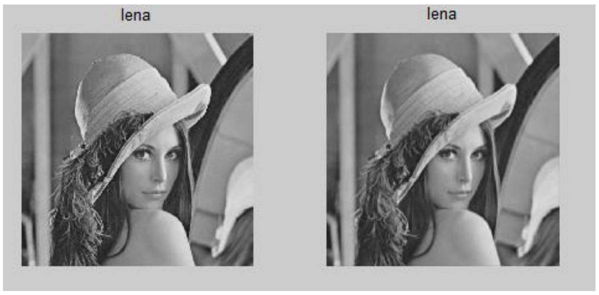

Fig. 17 I have applied scale (Im,[3 2], 1, 1]) on *lena*

The effect of applying the function that I have created is to smooth the image and makes it blurred for large values for kernel size and deviations.

From what I understood, this function calculated derivative of the image with order m,n which can have many applications.

Exercise 2.e

From what I understand, for this exercise I have to demonstrate that the Differentiation property from Table 5.3 in the course book is correct.

The theorem claims that the derivative of an image of order *m*,*n* can be calculated using the Fourier transform by doing simple multiplication.

```
%2.5
B = imread('D:\master\image processing\week3\lena.tiff');
subplot(1,2,1),imshow(B),title('lena');

%calculate derivative using the gaussian filter
m = 3;
n = 2;
devm = 1;
devn = 1;
Bf = scale(B,[m n],devm,devn);

%calculate fourier transform of the image
Bfft = fft2(B);

NP = abs(Bfft);
NPshift = fftshift(NP);
subplot(1,2,2),imagesc(log(1+NPshift)),title('power spectrum');

%calcualte gaussian kernel
G = [m n];

[m, n]=meshgrid(round(-G/2):round(G/2), round(-G/2):round(G/2));
kernel=exp(-m.^2/(2*devm^2)-n.^2/(2*devn^2));
kernel=kernel./sum(kernel(:));

%fourier transform of the kernel
kernelfft2 = zeros(size(B));
xpos = 1; ypos = 1;
kernelfft2(xpos:xpos+r-1,ypos:ypos+c-1) = kernel;
kernelfft2 = fft2(kernelfft2);

%multiply kernel with transform of the image
K = Bfft.*kernelfft2;
subplot(2,2,4),imagesc(real(ifft2(K))),colormap(gray),title('derivative using
fft');
```

In order to show that the property is correct, at first I calculated order m,n derivative of image using the Gaussian filter as I did at exercise 2.d.

At first I calculated the kernel and convolved it over the entire image. In next step I calculated fft of the original image and the Gaussian kernel and then multiplied them. In the end I calculated the inverse fft of the result to prove that the Differentiation is works.

From reasons I could not understand I did not manage to make the code work. It seems correct for me but I get unexplained error for line “*kernelfft2 = zeros(size(B))*” but it should work because I have already demonstrated at exercise 2.b. that the convolution of 2 matrixes is same as doing multiplication in frequency domain.

This property is very useful, because it can make faster to work with signal or image derivative in frequency domain. The convolution is using many for-loop while it is a lot faster to do matrix multiplication of the Fourier Transforms. In case we only need results for processing the fft of the image, it will be a lot faster to use multiplication with kernel than convolving.

Exercise 2.f

This can be done easily using the code in 2.e, by creating parameterized Gaussian filter, apply the Fourier Transform and multiplication of matrixes after transforming in frequency domain.

Because of the applications discovered at 2.e, this function must be very useful in applications.