Digital Image Processing
Lab 2

Group 50:
Henry Maathuis (S2589540)
Sebastian Wehkamp (S2589907)

December 10, 2017

Contents

1 Exercise 1 — Fourier spectrum **3**

1.1 a 3

1.2 b 3

1.3 c 4

2 Exercise 2 — Highboost filtering **4**

2.1 a 4

2.2 b 4

2.3 c 5

2.4 d 5

1 Exercise 1 — Fourier spectrum

1.1 a

In this section we created the function listed in Listing 1 which first applies the fourier transform and shifts it. After shifting we take the absolute value, and apply logarithmic scaling. We apply logarithmic scaling to visualise the spectrum since the centre value contains a very high value in contrast to the rest of the spectrum. Not applying scaling results in a blank image. The *+1* is needed since *log(0)* is not defined.

```
function [average] = IPfourierspectrum (img)
    %Perform fourier transform
    F = fft2(img);
    %Size of img
    [m, n] = size(img);
    %Compute average
    average = F(1,1)/(m*n);
    %Apply shift, scaling, and show image
    imagesc(log(abs(fftshift(F)))+1);
    %Set colormap for the image to gray
    colormap(gray);
    title('Fourier Spectrum magnitude');
end
```

Listing 1: Code to create fourier spectrum

1.2 b

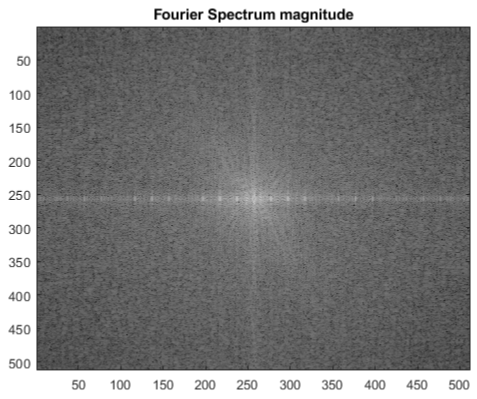


Figure 1: Picture of fourier spectrum from characters

Using Listing 1 we created Figure 1 which looks like we expected the spectrum to look like.

1.3 c

The average value is located on location *(1,1)*. This location contains the DC term or the 0 Hz term which is equivalent to the average of all the samples in the window. After dividing this value by the total number of pixels we get an average value of 207. This is in agreement with the average we got when we computed the mean of the original image using *mean(mean(image, 2))* which also returned 207.

2 Exercise 2 — Highboost filtering

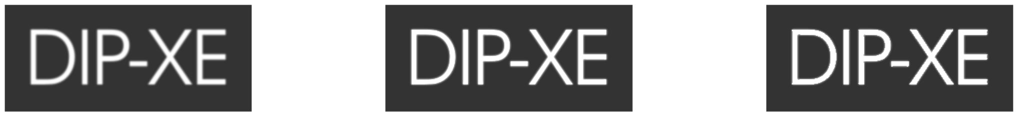
2.1 a

In Listing 2 you can see the implementation of highboost filtering in the spatial domain. First a filter is set and applied using the previously created IPfilter. Then a mask is calculated and applied to the image.

```
function [g] = IPhighboost (img, k)
    %Set filter h_smooth
    filter = [0 1/5 0; 1/5 1/5 1/5; 0 1/5 0];
    %Apply smoothing function of lab1
    f_smooth = IPfilter(img, filter);
    %Calculate g_mask
    g_mask = img - f_smooth;
    %Calculate g using equation 1 from the assignment
    g = img + k * g_mask;
    imshow(g)
end
```

Listing 2: Implementation of highboost filtering in the spatial domain

In Figure 2 you can see the highboost filter algorithm applied to dipxetext with varying k's. The image in the book is most similar to K = 10 but a more precise comparison is hard since we do not have a digital version of their image.



(a) K = 1 (b) K = 10 (c) K = 100

Figure 2: Images obtained by applying Listing 2 to dipxetext.tif with varying k's

2.2 b

In Fourier space we multiple each pixel in the smoothing filter H with the corresponding pixel in the Fourier transformed image F. Since the frequency domain transfer function of the smoothing filter is centred around $(0, 0)$, we know that $H(0, 0)$ is used to adapt to the highest frequency in the image. $H(0,0) = 1$ means that the DC value remains the same when using H. This DC value corresponds to the average of the image. Because of this relationship between the average and the DC value, and because the DC value remains the same, so does the average of the image. The average remains the same but we emphasise the high frequency component while trying to keep the low frequency components as well. Due to these properties we end up with a sharpening operator. The sharpening aspect can be seen in Exercise 2d.

2.3 c

We derive the formula $H(u,v) = k + 1 - kH_{smooth}(u,v)$ below.

$$g(x,y) = f(x,y) + k * g_{mask}(x,y) \tag{1}$$
$$g_{mask}(x,y) = f(x,y) - f_{smooth}(x,y) \tag{2}$$
$$g(x,y) = f(x,y) + kf(x,y) - kf_{smooth}(x,y) \tag{3}$$
$$G(u,v) = F(u,v) + kF(u,v) - kF_{smooth}(u,v) \tag{4}$$
$$F_{smooth}(u,v) = H_{smooth}(u,v)F(u,v) \tag{5}$$
$$G(u,v) = F(u,v) + kF(u,v) - kH_{smooth}(u,v)F(u,v) \tag{6}$$
$$G(u,v) = H(u,v)F(u,v) \tag{7}$$
$$H(u,v) = 1 + k - kH_{smooth}(u,v) \tag{8}$$
$$Q.E.D \tag{9}$$

We start with the original equation (1) and defined mask (2). We obtain (3) by substitution of (2) in (1). Then we obtain (4) by transforming from spatial to frequency domain using Linearity. A smoothed version of the image is given by multiplication of the smoothing kernel and image in Fourier space (5). By substituting (5) in (4) we obtain (6). The output image is given by multiplication of a kernel and an output image in Fourier space (7). One can rewrite to (8) by factoring out F(u, v) in (6) using (7).

The property that we use is the fact that convolution of an image with a kernel in the spatial domain is the same as a multiplication of an image and kernel in the frequency domain (Convolution theorem). Additionally we use the linearity property which states that: $af_1(x,y) + bf_2(x,y) \Leftrightarrow aF_1(u,v) + bF_2(u,v)$.

2.4 d

We created two implementations of highboost filtering in the frequency domain. First of we created an implementation which used $H(u,v) = k + 1 - kH_{smooth}(u,v)$ as the transfer function, the implementation can be found in Listing 3. The code is mainly based on the implementation of a LaPlacian filter provided in the slides. First it prepossesses both the filter and image by adding padding and creating the correct mask. Then both the mask and the image are transferred to frequency space, $H(u,v) = k + 1 - kH_{smooth}(u,v)$ is applied, and the image is transferred back to the spatial domain. In Figure 3 you can see the results of applying Listing 3 to dipxetext.tif using varying k's. Note that K = 1 is equal to the original image.

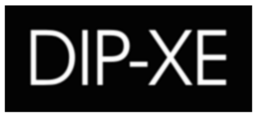
```
function [g] = IPhighboost_freq2 (f, k)
    % Kernel h=[0 1/5 0; 1/5 1/5 1/5; 0 1/5 0]
    % Zero padding is applied to avoid boundary overlap effects
    M=size(f,1); N=size(f,2); % nr of rows/columns of image f
    C=3; D=3; % nr of rows/columns of kernel h
    P=M+C-1; Q=N+D-1; % nr of rows/columns after padding
    fp=zeros(P,Q); % zero padding: start with zeroes
    fp(1:M,1:N)=f; % insert f into image fp
    hp=zeros(P,Q); % Construct filter matrix hp, same size as fp.
    hp(1,1)=1/5; hp(2,1)=1/5; hp(1,2)=1/5; % Center is at (1,1)
    hp(P,1)=1/5; hp(1,Q)=1/5; % Indices modulo P or Q
    Fp=fft2(double(fp), P, Q); % FFT of image fp
    Hp=fft2(double(hp), P, Q); % FFT of kernel hp

    % Apply transfer function
    Hp = k + 1 - k * Hp;
    Gp = Hp .* Fp;

    gp=ifft2(Gp); % Inverse FFT
    gp=real(gp); % Take real part
    g=gp(1:M, 1:N); % Undo zero padding

    % Plot the image
    % Force the image to use the lowest value as black and highest as white
    imshow(g, [min(g(:)) max(g(:))]);
end
```

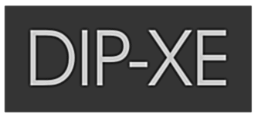
Listing 3: Implementation of highboost filtering in the frequency domain using $H(u,v) = k + 1 - kH_{smooth}(u,v)$

The second implementation we created is based on the transfer function $H_{smooth}(u,v) = \frac{1}{5}(1 + 2cos(\frac{2\pi u}{M}) + 2cos(\frac{2\pi v}{N})$ which can be seen in Listing 4. The implementation is very similar to the implementation of Listing 3 with the main difference being the transfer function. After a discussion with the TA, we were told that both implementations result in the correct image. The TA stated that the Listing 4 implemenation results in in a bit more precise result although the loss of precision is roughly 0.2 at max on a scale of 0-255. In Figure 4 you can see the results of applying Listing 4 to dipxetext.tif using varying k's. Note that K = 1 is equal to the original image.

```
function [g] = IPhighboost_freq3 (f, k)
    % Kernel h=[0 1/5 0; 1/5 1/5 1/5; 0 1/5 0]
    % Zero padding is applied to avoid boundary overlap effects
    M=size(f,1); N=size(f,2); % nr of rows/columns of image f
    C=3; D=3; % nr of rows/columns of kernel h
    P=M+C-1; Q=N+D-1; % nr of rows/columns after padding

    rfloor = floor(P/2); rceil = ceil(P/2); % midpoint along rows
    cfloor = floor(Q/2); cceil = ceil(Q/2); % midpoint along columns
    u = -rfloor:rceil-1; % centered vector along rows
    v = -cfloor:cceil-1; % centered vector along columns
    [V,U] = meshgrid(v,u); % create mesh of size P x Q

    % The frequency domain transfer function of the smoothing ?lter (
        centered around (0,0))
    Hc = 1/5 + 2/5*cos(2*pi*U/P) + 2/5*cos(2*pi*V/Q);
    H = ifftshift(Hc); % centered at corner (1,1)

    fp=zeros(P,Q); % zero padding: start with zeroes
    fp(1:M,1:N)=f; % insert f into image fp
    Fp=fft2(double(fp), P, Q); % FFT of image fp

    Hp = k + 1 - k * H;
    Gp = Hp .* Fp; % Product of FFTs

    gp=ifft2(Gp); % Inverse FFT
    gp=real(gp); % Take real part
    g=gp(1:M, 1:N); % Undo zero padding

    % Plot the image
    % Force the image to use the lowest value as black and highest as white
    imshow(g, [min(g(:)) max(g(:))]);
end
```

Listing 4: Implementation of highboost filtering in the frequency domain using $H_{smooth}(u,v) = \frac{1}{5}(1 + 2cos(\frac{2\pi u}{M}) + 2cos(\frac{2\pi v}{N}))$


(a) K = 1


(b) K = 10

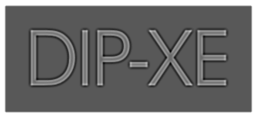
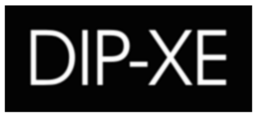
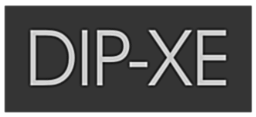

(c) K = 100

Figure 3: Images obtained by applying Listing 3 to dipxetext.tif with varying k's


(a) K = 1


(b) K = 10

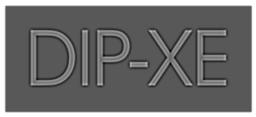
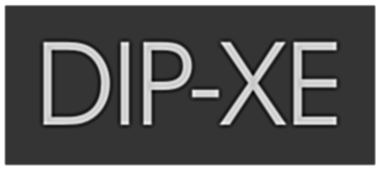
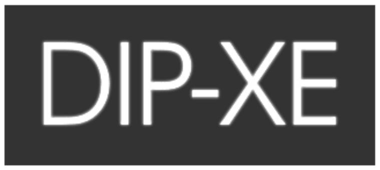

(c) K = 100

Figure 4: Images obtained by applying Listing 4 to dipxetext.tif with varying k's

You can see in Figure 3 and Figure 4 that the images are clearly similar. When we looked at the differences between the two they were in the order of e^{-12} so the differences are not significant. To compare the results of both the frequency and the spatial domain version we used the images for k = 10 which can be seen in Figure 5. Although both versions show improvements the results are clearly not identical. The Frequency domain version made the image more gray than the original. This is not the case for the spatial domain version.



(a) Frequency domain



(b) Spatial domain

Figure 5: Image comparison between the frequency domain and the spatial domain version of dipxetext.tif