Image Processing
Lab 2

Kevin Gevers (s25595987)
Jeroen Overschie (s2995697)

January 20, 2021

Note, that all used source code can be found attached next to this report's pdf file. Its structure should be self-explanatory; all requested functions are named accordingly and any extra functions are explained in the report. Note that (almost) every function has a corresponding test script, which is named just like its function, but with a suffix '`_test`'. Where possible, we followed the terminology from the book [1] for variable naming.

Exercise 1

(a)

For the first exercise of this lab, we had to compute the *centered Fourier spectrum* of the input image *characters.tif*. This involves three steps, first, we need to apply the Fourier transformation, then we need to center it, and lastly, we need to take the magnitude of it.

The first step is to calculate the 2D Discrete Fourier Transform (DFT) of the image, according to Formula 1 (Eq.4-67 in the book). Since we are allowed to use Matlab's `fft2` function we did not implement this ourselves.

$$F(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(ux/M+vy/N)} \tag{1}$$

For the second step, we need to center the Fourier transform. For displaying and filtering purposes we want to move the four quarter periods (shown with the dotted lines on the left side of Figure 1) so that we have one full period instead. To do this we need to apply Formula 2 (Eq.4-76 in the book). Seeing as we are allowed to use Matlab's *fftshift* function we did not have to implement this ourselves.

$$f(x,y)(-1)^{x+y} \Leftrightarrow F(u-M/2, v-N/2) \tag{2}$$

The third and final step is taking the magnitude of the centered Fourier transform, which gives us the spectrum. The magnitude of each pixel is the absolute value of each pixel. Since the centered Fourier transform consists of complex numbers we need to apply Formula 3 (Eq.4-87 in the book). This gives us the polar representation of the pixel and can be used to visualize the spectrum. Matlab's standard function `abs` handles complex numbers using this formula, so we again did not implement this.

$$|F(u,v)| = \sqrt{R^2(u,v) + I^2(u,v)} \tag{3}$$

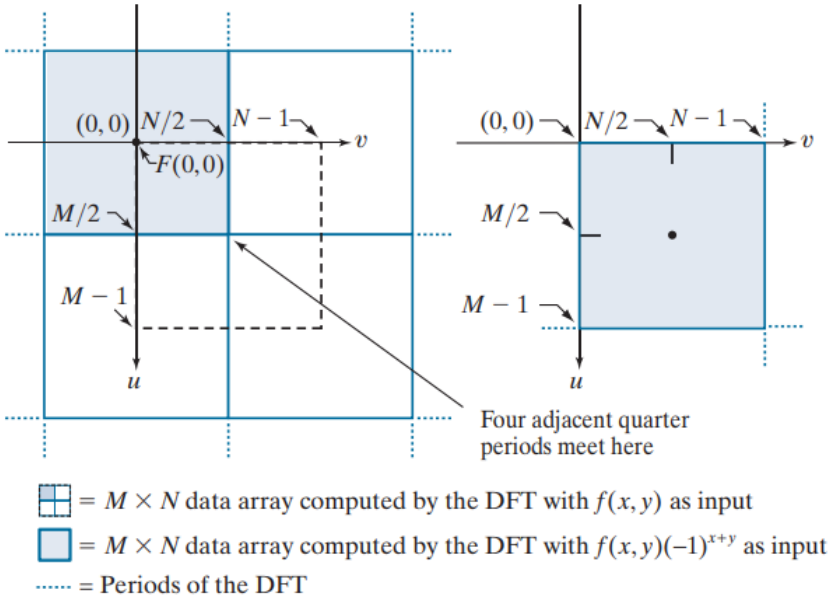

Figure 1: Image showing what is happening to the image when the Fourier transform is being centered [1].

(b)

Now that we have the centered Fourier spectrum of the input image we can visualize it. The centered Fourier spectrum is visualized in Figure 2b, but is near impossible to see (please use the Matlab code to get an interactive image if you do want to see it properly). To make sure the spectrum is easily visible we use a log transformation. In Figure 2d the log transformation of the spectrum can be seen and is much easier to interpret. Figure 2a and 2c show the non-centered Fourier spectrum without and with the log transformation respectively.

(c)

The last part of the exercise simply wants us to use the result from (a) to compute the average value of the image. Formula 4 (Eq.4-92 in the book) shows that the zero-frequency term of the DFT is proportional to the mean of the original image. The following steps are clearly outlined in ❖ Listing 1. The average of the original image is 0.7997 and using Formula 4 we determined that $MN\bar{f} = 3.7855e+05$. By selecting the middle pixel of the centered Fourier spectrum (the result of (a)) we get the zero-frequency term of our result, which is also $3.7855e+05$. We divide that by the dimensions of the image (M and N) and get the average value of the original image again: 0.7997.

$$\begin{aligned} F(0,0) &= MN\frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y) \\ &= MN\bar{f} \end{aligned} \tag{4}$$

The resulting code for the entire exercise can be seen in ❖ Listing 1.

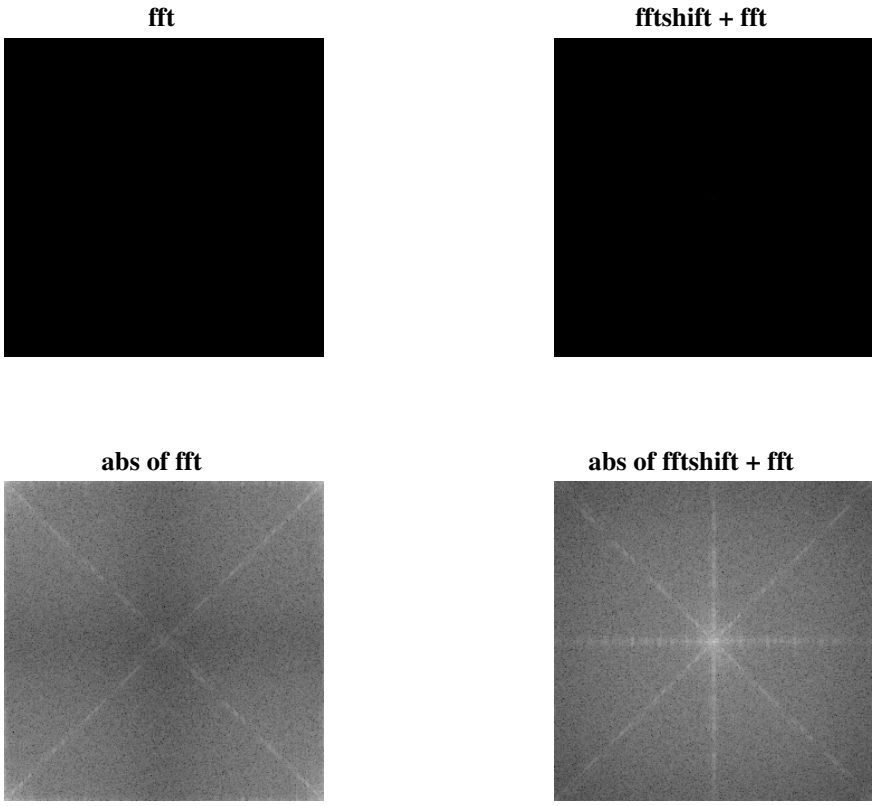

Figure 2: (a) non-centered Fourier spectrum. (b) centered Fourier spectrum. (c) log transformation of the non-centered Fourier spectrum. (d) log transformation of the centered Fourier spectrum.

Exercise 2

(a)

In this exercise we are asked to implement a *Butterworth Highpass Filter* (BHPF) transfer function. To do this, we first built a distance function D as according to Eq (4-112):

$$D(u,v) = [(u - P/2)^2 + (v - Q/2)^2]^{1/2}$$

This function computes the distance for some given coordinate (u,v) to the center of the $P \times Q$ frequency rectangle (note the sizes of P and Q in Eqs. (4-100) and (4-101): $2M$ and $2N$, respectively), which can be intuitively understood as the Euclidean distance in two-dimensional u/v space. The distance function is implemented in ❖ Listing 2. Once we have our distance function, we can define our Butterworth highpass filter transfer function, Eq (4-121):

$$H(u,v) = \frac{1}{1 + [D_0/D(u,v)]^{1/2}}$$

Where D_0 is the cutoff frequency and the order is given by n. For the implementation of this transfer function, see ❖ Listing 3. We are now able to create and visualize the filter we just built. If we take some appropriately-sized matrix, e.g. of 500x500 size, we can visualize what the transfer function looks like. We iterate the function over (1) a 3D u/v

space with the z-coordinate set as the function value $H(u,v)$, (2) a 2D *u/v* with the function value $H(u,v)$ set as the grayscale color intensity and finally (3) a cross-section of the transfer function, made over the constant value $v = 500$ and letting u range from 500 to 1000, i.e. $u = [500, 501, ..., 1000]$. We set the x axis as the distance function value, $D(u,v)$, indicating how far off the given point is from the center. See Figure 3.

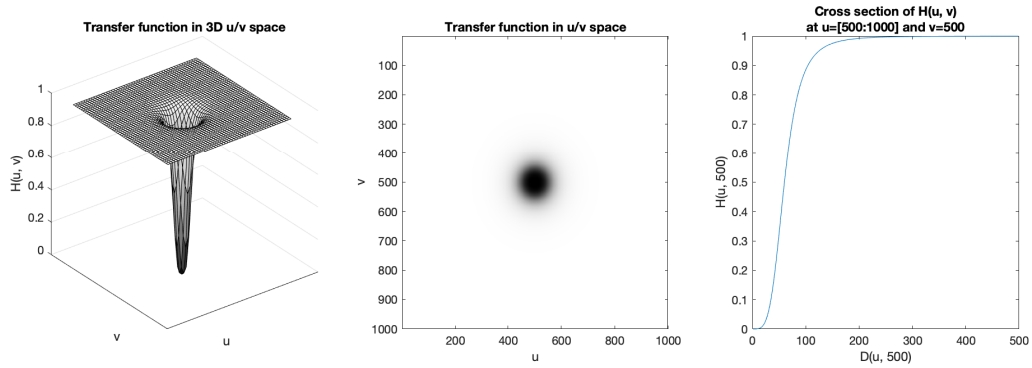

Figure 3: Visualization of the BHPF transfer function $H(u,v)$. (1) H in 3D space, (2) H as a grayscale intensity value in a 2D space and (3) a cross-section of H versus the distance function D.

Due to a high amount of curiosity, we were wondering whether we could also recreate an *Ideal Highpass Filter* (IHPF). This only required a new transfer function; due to our modular code structure, we were able to re-use the distance function. See Figure 4 for the IHPF transfer function, similar to Figure 4.51 in the book.

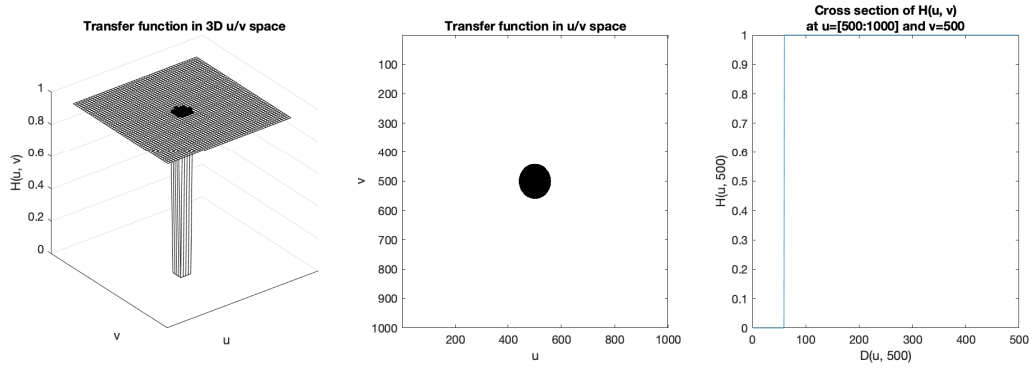

Figure 4: Visualization of a IHPF 'Ideal' highpass filter - built and included out of curiosity. Visualized in 3 ways: a 3D- and 2D plot, as well as a cross-section of $H(u,v)$ over the distance function D.

(b)

In this next assignment, we perform convolution filtering of some input image, x, given some transfer function H that operates in the frequency domain. Our implementation in part follows Eq (4-104) from the book:

$$g(x,y) = Real\{\Im^{-1}[H(u,v)F(u,v)]\}$$

Where F is the *Discrete Fourier Transform* from the input image (DFT) and $\Im^{-1}$ is the inverse Discrete Fourier Transform. See Eqs. (6-67) and (6-68), respectively. Also, $H(u,v)$ is some transfer function, e.g. the BHPF we just built-in (a). Furthermore, for the entire operation, the summary in section 4.7 of the book was followed. Disregarding irony, the summary can be summarized as follows:

1. Obtain padding sizes as $P = 2M$ and $Q = 2N$.
2. Form padded image $P \times Q$.

3. Multiply padded image by $(-1)^{x+y}$ to center Fourier Transform. In our implementation we use `fftshift` to perform this operation at once.
4. Compute the DFT, $F(u,v)$. We use `fft2` for this.
5. Construct filter transfer function, $H(u,v)$.
6. Compute element-wise product $G = F * H$.
7. Obtain filtered image by using inverse DFT. Uses `ifft2` and `ifftshift`.
8. Obtain the final result by extracting part of the padded image.

After having applied these steps, a filtered image, g, will be obtained from the input image f, given some transfer function H. See ❖ Listing 4 for the implementation in Matlab.

(c)

Finally, we apply the filter using our built transfer function on some image file, `characters.tif`. See Figure 5.

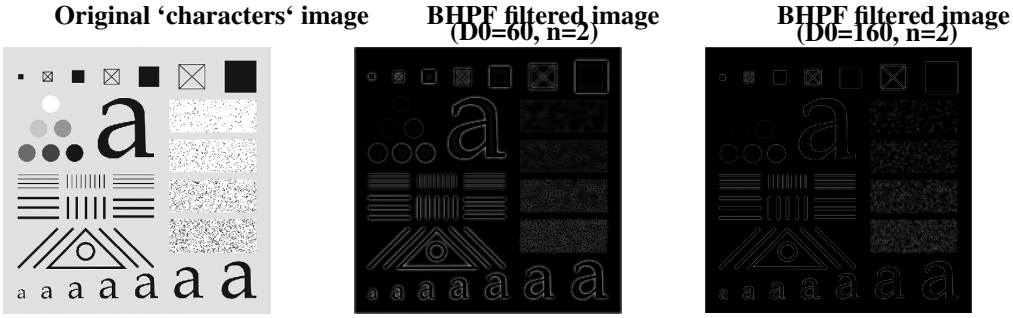

Figure 5: BHPF filter applied to `characters.tif` image. Left shows original, middle shows BHPF filter applied using $D_0 = 60$ and $n = 2$ and right-most shows filter applied using $D_0 = 160$ and $n = 2$.

We can observe in the figure that the images look exactly like the examples in the book. The difference between having varied the D_0 parameter also match up. We can see that edges and other 'abrupt changes' in intensity levels have been highlighted by the highpass filter, which is like expected; high-frequency components are highlighted.

Exercise 3

(a)

For this exercise we are tasked to create a function, `IPmedian`, that performs median filtering in a $(2k+1) \times (2k+1)$ window, with a variable k. A median filter uses Formula 5 (Eq. 5-27 in the book). This comes down to taking the median of the pixel values that fall into the region of the mask. So for a 3×3 mask, the center of the mask will be placed over the pixel and the values of all the pixels in the mask will be placed into a set of which the median is taken. The resulting value is the output value. This is repeated for all pixels of the image. For our implementation see ❖ Listing 5.

$$\widehat{f}(x,y) = \underset{(r,c)\in S_{xy}}{\text{median}}\{g(r,c)\} \tag{5}$$

(b)

For the next part, we had to load the image `circuitboard.tif` and add salt-and-pepper noise to it, with $P_s = P_p = 0.2$. As suggested by the assignment we used the Matlab function `imnoise` for this. The resulting image can be seen

in Figure 6b.

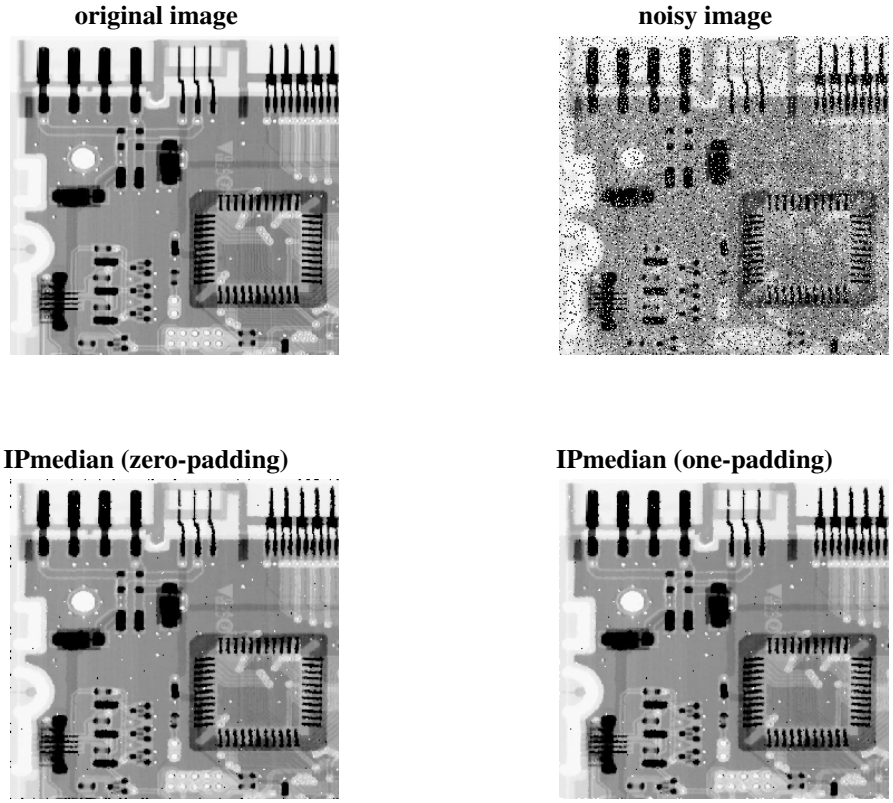

Figure 6: (a) original input image. (b) salt-and-pepper noisy image. (c) `IPmedian` function applied to b with a 3×3 mask and (incorrect) zero-padding. (d) `Ipmedian` function applied to b with a 3×3 mask and (correct) one-padding.

(c)

For the last part we had to apply a 3×3 filter to the image we created in (b). The result of our function can be seen in Figure 6c. The main difference between our image and Fig.5.10(b) of the book is the number of dark pixels at the edges of the image*. This turned out to be caused by the zero-padding we used and changing that into one-padding (because we use the image in double format in Matlab) this was solved and can be seen in Figure 6d. With this improved image the only real difference with Fig.5.10(b) of the book is the amount of noise that is left. Seeing as we used $P_s = P_p = 0.2$ and the book used $P_s = P_p = 0.1$ we started out with about twice as much noise. This can also be seen when comparing Figure 6b and Fig.5.10(a) of the book. Because of this, there is slightly more noise left in our image after a single pass. All the shapes of the original image are still intact, so making two more passes results in almost exactly the same image as Fig.5.10(d) of the book, as can be seen in Figure 7.

*Note that some of the images are hard to make out in PDF format, so please run `IPmedian_test.m` to get the high-resolution images.

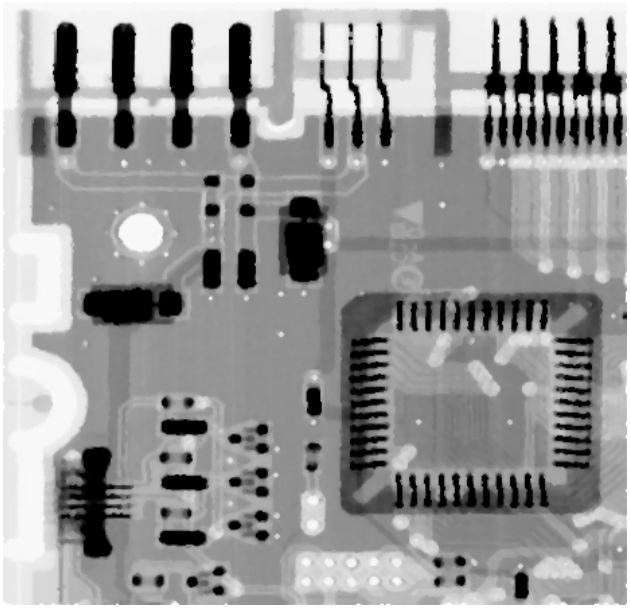

Figure 7: The result of making 3 passes with a 3×3 mask with `IPmedian`.

References

[1] Rafael C. Gonzalez and Richard E. Woods. *Digital image processing*. Prentice Hall, Upper Saddle River, N.J., 2008.

A Code

A.1 Exercise 1

Listing 1: `exercise_1.m`: Code for exercise 1.

```
clc;
clear;
close all;

imname = 'characters';
inputfile = ['input_images/', imname,'.tif'];
f = imread(inputfile);
f = im2double(f);

im1 = abs(fft2(f));
im2 = abs(fftshift(fft2(f)));
im3 = log(abs(fft2(f)));
im4 = log(abs(fftshift(fft2(f))));

% Display the spectrum.
figure;
subplot(221);
imshow(im1,[])
title('fft')
subplot(222);
imshow(im2,[])
```

```
title('fftshift + fft')
subplot(223);
imshow(im3,[])
title('abs of fft')
subplot(224);
imshow(im4,[])
title('abs of fftshift + fft')

% Write current figure to file
set(gcf, 'PaperUnits', 'normalized')
saveas(gcf, 'output_plots/exercise1.svg');

% Use your result in (a) to compute the average value of the image.

% average of the image
average_image = mean(f, 'all');

% average of the image proportional to the zero-frequency term (Eq.4-92)
[M, N] = size(f);
average_image_proportional = M*N*mean(f, 'all');

% zero-frequency term in the result of (a)
spectrum = abs(fftshift(fft2(f)));
zero_frequency_term_of_a = spectrum(M/2+1,N/2+1);

% average of the original image using result of (a)
average_image_using_a = spectrum(M/2+1,N/2+1)/(M*N);
```

A.2 Exercise 2

A.2.1 Exercise 2 (a)

Listing 2: IPfreqrectdists.m: Compute distances to center of $P \times Q$ frequency rectangle.

```
function D = IPfreqrectdists(M, N)
% IPfreqrectdists Computes distances between points (u, v) in the frequency
% domain and the center of the P x Q frequency rectangle, i.e. Eq (4-112).
%   Input arguments:
%       M x N = size
%   Output arguments:
%       D = distance function

P = 2*M; % Eq (4-102)
Q = 2*N; % Eq (4-103)
u = 0:(P - 1);
v = 0:(Q - 1);

% Make coordinate grid
[U, V] = meshgrid(u, v);

% Distance between a point (u, v) in the frequency domain and the center
% of the P x Q frequency rectangle
D = sqrt((U - P/2).^2 + (V - Q/2).^2); % Eq (4-112)
end
```

Listing 3: IPbhpf.m: Butterworth High-Pass Filter (BHPF).

```
function H = IPbhpf(D0, n, M, N)
% IPbhpf Butterworth highpass filter (BHPF) implementation.
%   Input arguments:
%       D0 = cutoff frequency
%       n = order
%       M x N = size
%   Output arguments:
%       H = transfer function

D = IPfreqrectdists(M, N); % Obtain distances to PxQ center

% Transfer function of Butterworth highpass filter (BHPF)
H = 1 ./ (1 + (D0 ./ D) .^ (2 * n)); % Eq (4-121)
end
```

A.2.2 Exercise 2 (b)

Listing 4: IPftfilter.m: Convolute a filter in the frequency domain.

```
function g = IPftfilter(x, H)
% IPftfilter Perform convolution filtering. Partially following Eq (4-104)
% and steps as defined in Section 4.7 'Summary of steps for filtering in
% the frequency domain'.
%   Arguments:
%       x = input image
%       H = transfer function
%   Output:
%       g = filtered image
[M, N] = size(x); % height, width

% Determine padding sizes
P = 2*M; % Eq (4-102)
Q = 2*N; % Eq (4-103)

% Form a padded image using zero-padding
f_p = zeros(P, Q);
f_p(1:M, 1:N) = im2double(x); % copy over original image

% Compute DFT
F = fftshift(fft2(f_p)); % Eq (4-67)

% Form the product G(u, v)
G = F .* H; % Eq (4-104)

% Obtain filtered image, compute inverse DFT of G(u, v)
g_p = real(ifft2(ifftshift(G))); % among others, IDFT from Eq (4-68)
g = g_p(1:M, 1:N);
end
```

A.3 Exercise 3

Listing 5: IPmedian.m: Median filter.

```
function result = IPmedian(f, k)
    [M, N] = size(f); %M = height, N = width
```

```
    windowSize = 2*k+1;
    result = zeros(M,N);
    % loop over all the pixels in the image
    for x=1:M
        for y=1:N
            set = zeros(windowSize);
            % loop over the window size
            for r=-k:k
                for c=-k:k
                    if (x+r < 1 || y+c < 1 || x+r > M || y+c > N)
                        % fixed one padding
                        set(k+1+r, k+1+c) = double(1);
                        continue;
                    end
                    set(k+1+r, k+1+c) = f(x+r, y+c);
                end
            end
            % take the median of the pixels inside the window
            result(x,y) = median(set, [1,2]);
        end
    end
end
```