Table of Contents

Read Data

```
clear;
camera = double(imread('lab2/images_lab2/cameraman.png'));
wagon = double(imread('lab2/images_lab2/wagon.png'));
wagonnoise = double(imread('lab2/images_lab2/wagon_shot_noise.png'));
lines = double(imread('lab2/images_lab2/lines.png'));
circle = double(imread('lab2/images_lab2/circle.png'));
rectangle = double(imread('lab2/images_lab2/rectangle.png'));
```

Q1 - Different Kernels

We will apply three different kernels in the spatial domain with one sharpening, one smoothing and apply them in different sizes.

Firstly we begin with showing the original image that we will use.

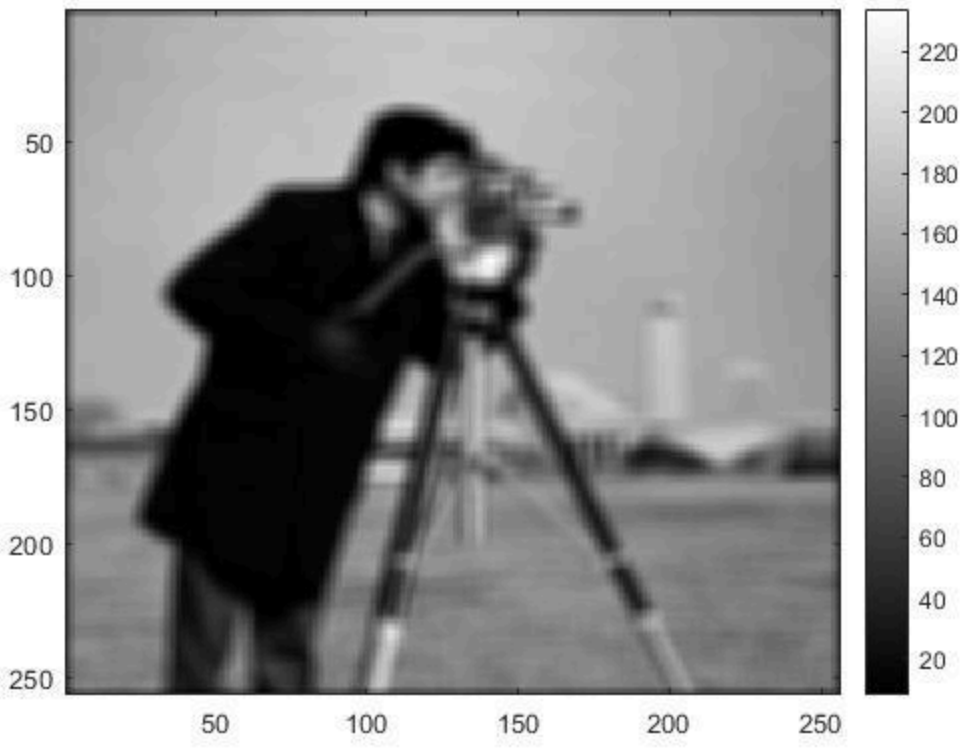
```
figure;
imagesc(camera);
colormap(gray);
colorbar;
```

Now, let's introduce a mean filter of size 3×3 and apply the convolution using **imfilter**. The mean filtered image is shown below.

```
h1 = fspecial('average', 3);
meancamera3 = imfilter(camera, h1);

figure;
imagesc(meancamera3);
colormap(gray);
colorbar;
```

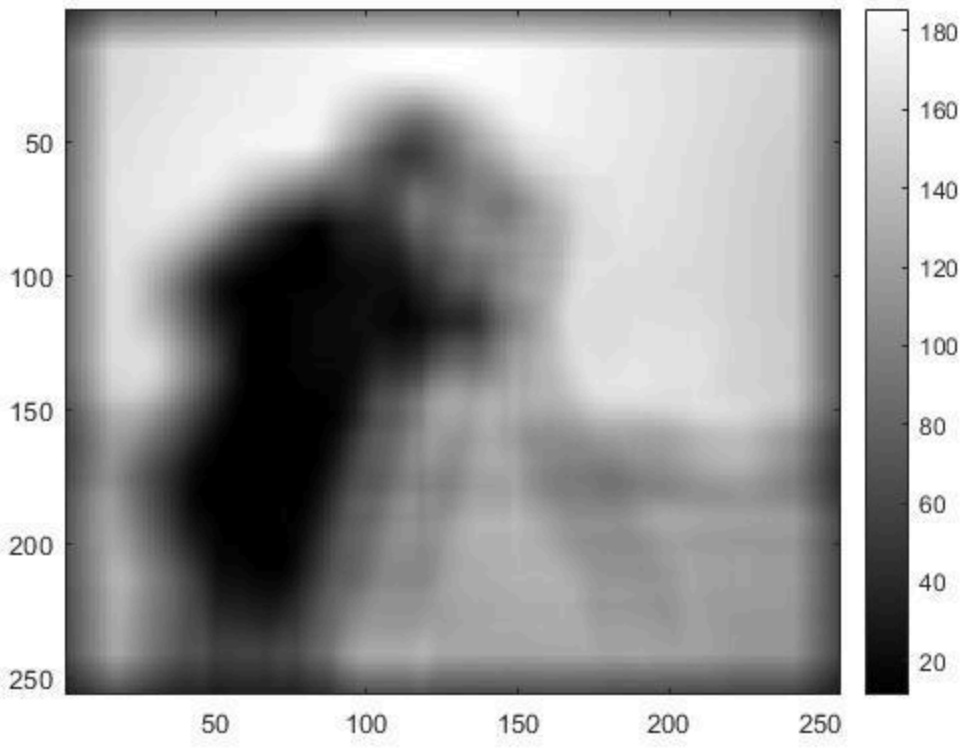
And using mean filter of sie 7×7 we get

```
h2 = fspecial('average', 7);
meancamera7 = imfilter(camera, h2);

figure;
imagesc(meancamera7);
colormap(gray);
colorbar;
```



And lastly, a 31×31 mean filter.

```
h3 = fspecial('average', 31);
meancamera31 = imfilter(camera, h3);

figure;
imagesc(meancamera31);
colormap(gray);
colorbar;
```

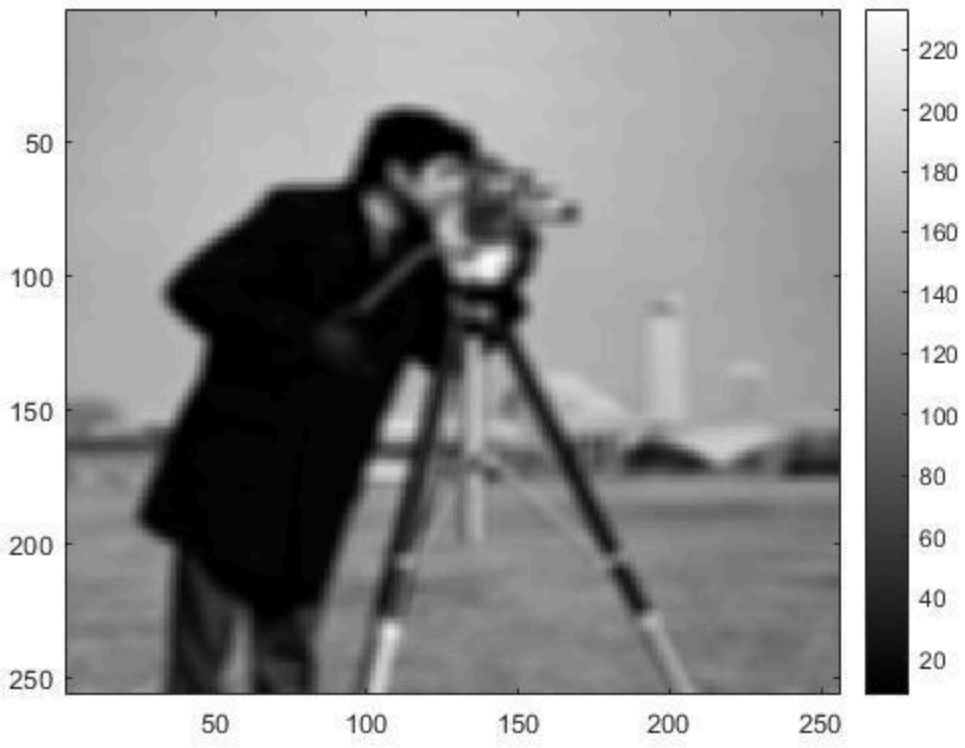


Lets introduce gaussian filters and perform the same calculations as above. Since the gaussian filter is based on the gaussian distribution We also have an additional parameter in addition to the size of the kernel, namely σ. We have assumed this to be $\sigma = 3$ for this exercise. We use the function imgaussfilt due to the documentation recommending to use that one instead of imfilter.

```
gausscamera3 = imgaussfilt(camera, 3, FilterSize=3);

figure;
imagesc(gausscamera3);
colormap(gray);
colorbar;
```

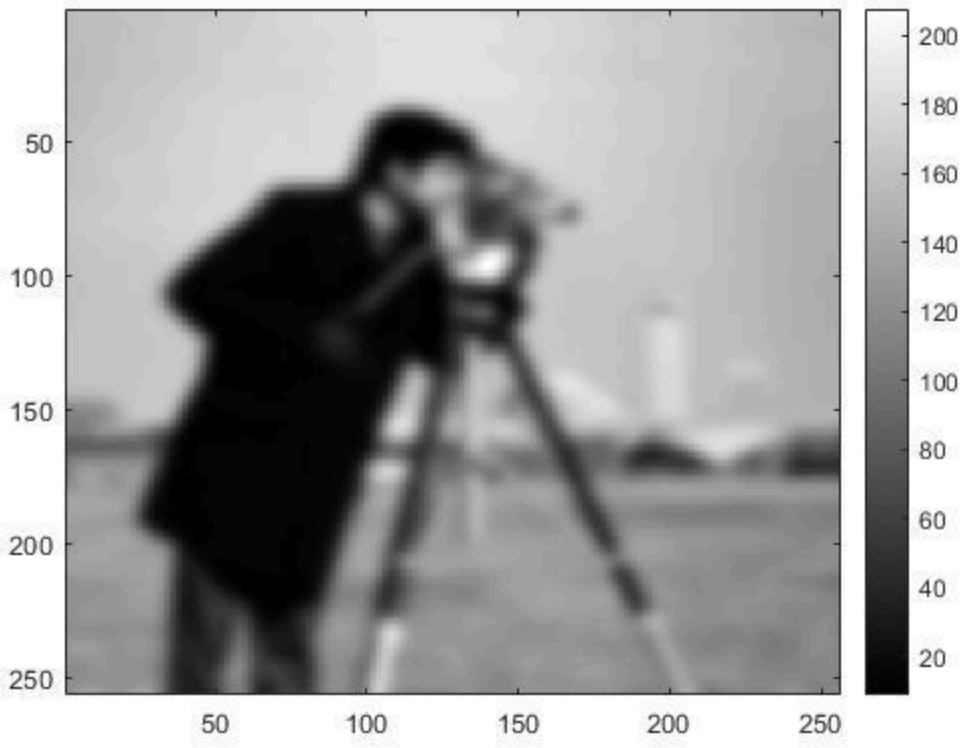
And using a gaussian filter size of 7×7 we get

```
gausscamera7 = imgaussfilt(camera, 3, FilterSize=7);

figure;
imagesc(gausscamera7);
colormap(gray);
colorbar;
```



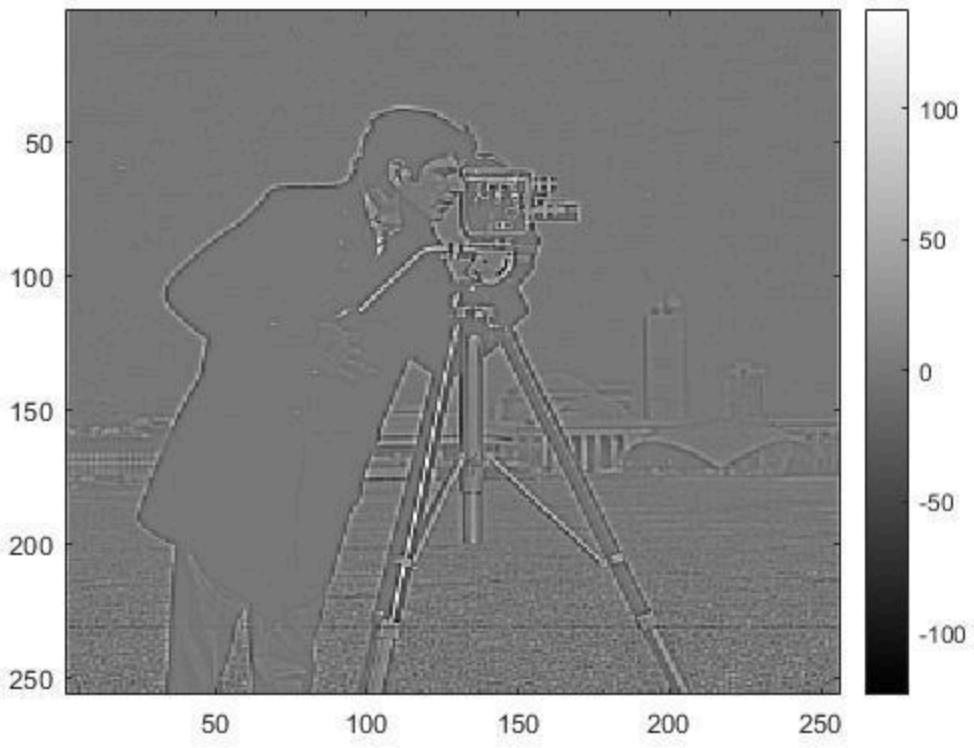
And lastly using a gaussian filter size of 31×31 we get

```
gausscamera31 = imgaussfilt(camera, 3, FilterSize=31);

figure;
imagesc(gausscamera31);
colormap(gray);
colorbar;
```



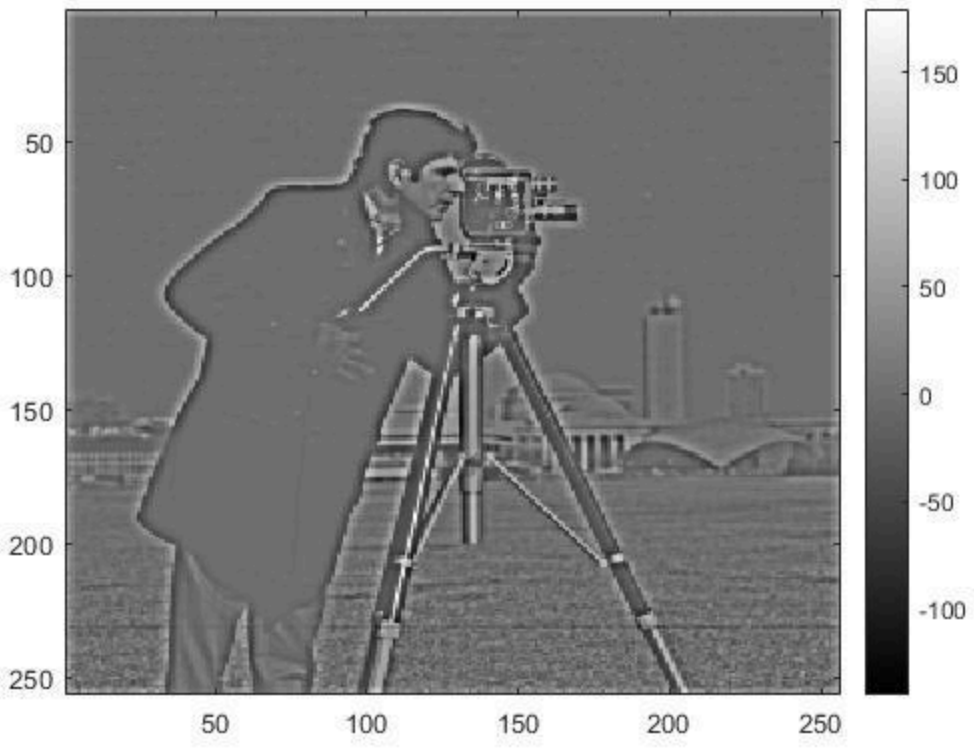
For a sharpening (high pass) filter, we will use a unsharp masking of the mean filter for different sizes. Below we do this for a 3×3 unsharping mean filter.

```
siz = 3;
h4 = fspecial('average', siz);
h4 = h4 * -1;
h4(siz - floor(siz/2), siz - floor(siz/2)) = h4(siz - floor(siz/2),
 siz - floor(siz/2)) + 1;

msharpcamera3 = imfilter(camera, h4);

figure;
imagesc(msharpcamera3);
colormap(gray);
colorbar;
```



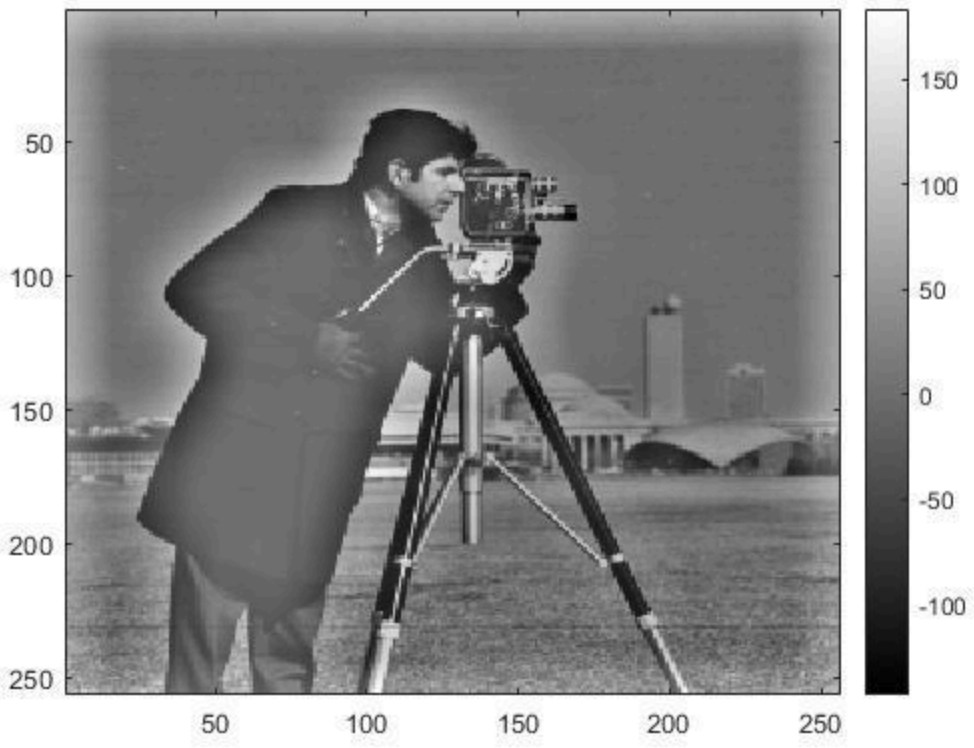
And once again for the mask of size 7×7

```
siz = 7;
h5 = fspecial('average', siz);
h5 = h5 * -1;
h5(siz - floor(siz/2), siz - floor(siz/2)) = h5(siz - floor(siz/2),
 siz - floor(siz/2)) + 1;

msharpcamera7 = imfilter(camera, h5);

figure;
imagesc(msharpcamera7);
colormap(gray);
colorbar;
```



And lastly, one last time for the mask of size $31 \times 31£

```
siz = 31;
h6 = fspecial('average', siz);
h6 = h6 * -1;
h6(siz - floor(siz/2), siz - floor(siz/2)) = h6(siz - floor(siz/2),
 siz - floor(siz/2)) + 1;

msharpcamera31 = imfilter(camera, h6);

figure;
imagesc(msharpcamera31);
colormap(gray);
colorbar;
```



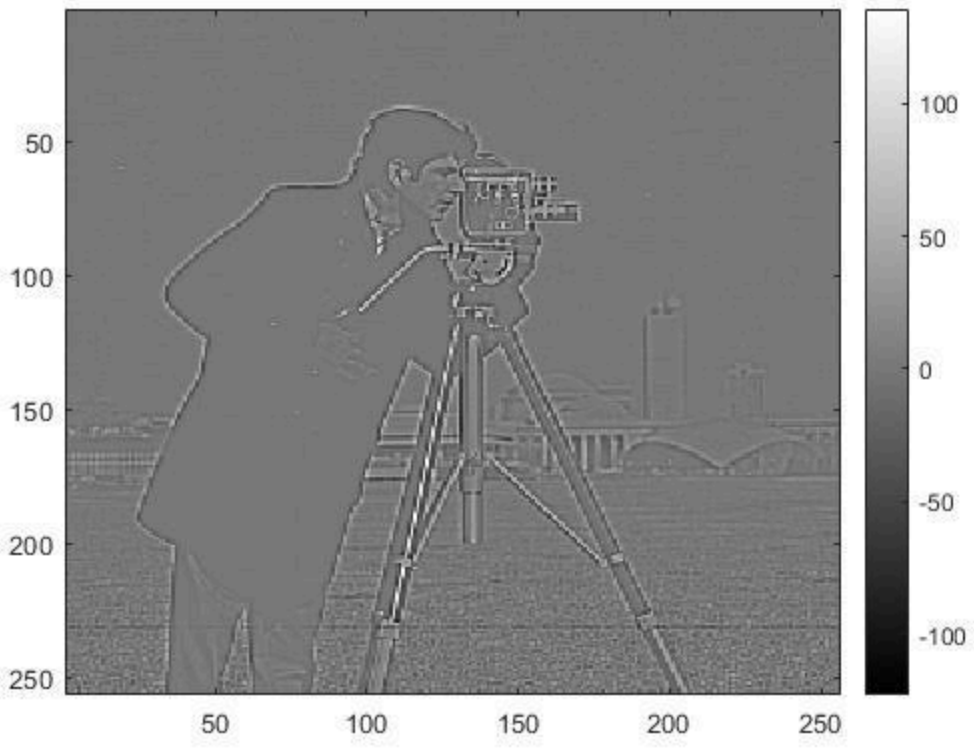
Q2 - Low Pass Filters

In fspecial we have the filters **average**, **disk** and **gaussian**. All of which are instances of low-pass filters. They calculate new pixel values by incorporating neighbouring pixel values and "average" the rapid changes. The **average** filter does this by calculating the sample mean of the defined neighbourhood, disk uses thhe same notion but uses a circular notion of distance from the center and the gaussian uses a gaussian distribution where pixels far away affect the new pixel value less.

Q3 - Construct new filters from simple arithmetic.

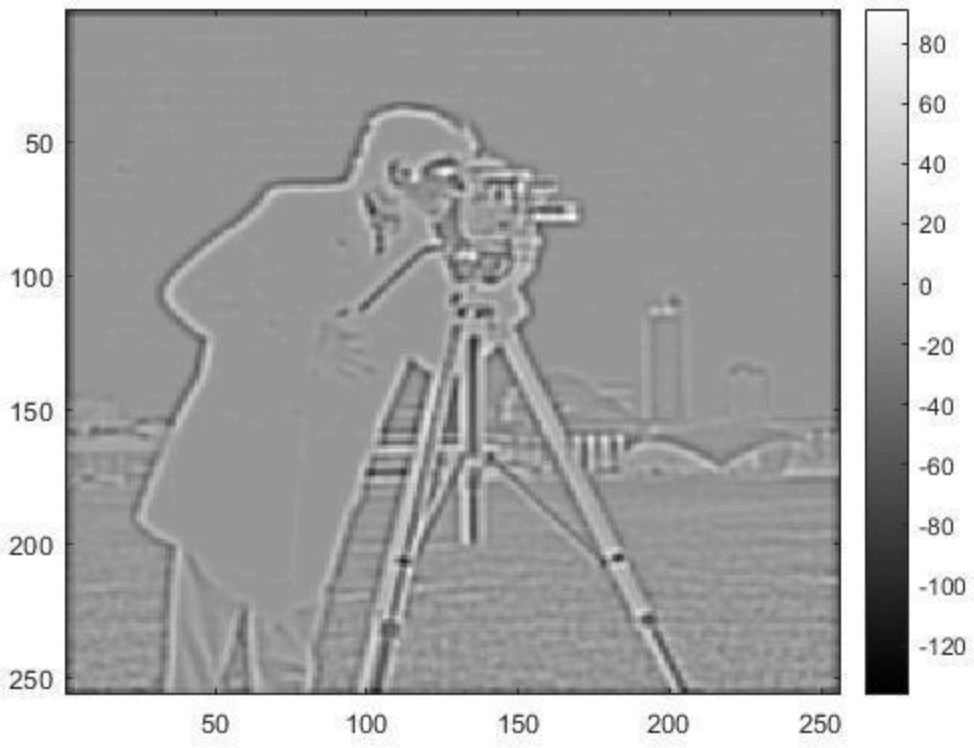
A low pass filter can be denoted $lp(x, y)$. We know from the textbook that a high-pass image $hp(x, y) = \delta(x, y) - lp(x, y)$ i.e subtracting a low pass from the original image.

```
imagesc(camera - gausscamera3);
colormap(gray);
colorbar;
```



A bandreject filter is a original image minus a low pass and a high pass filter. But since we have defined the highpass filter in terms of a low pass, we can also define bandreject from only low pass filters.

```
imagesc(camera - (gausscamera3 + (camera - meancamera7)));
colormap(gray);
colorbar;
```



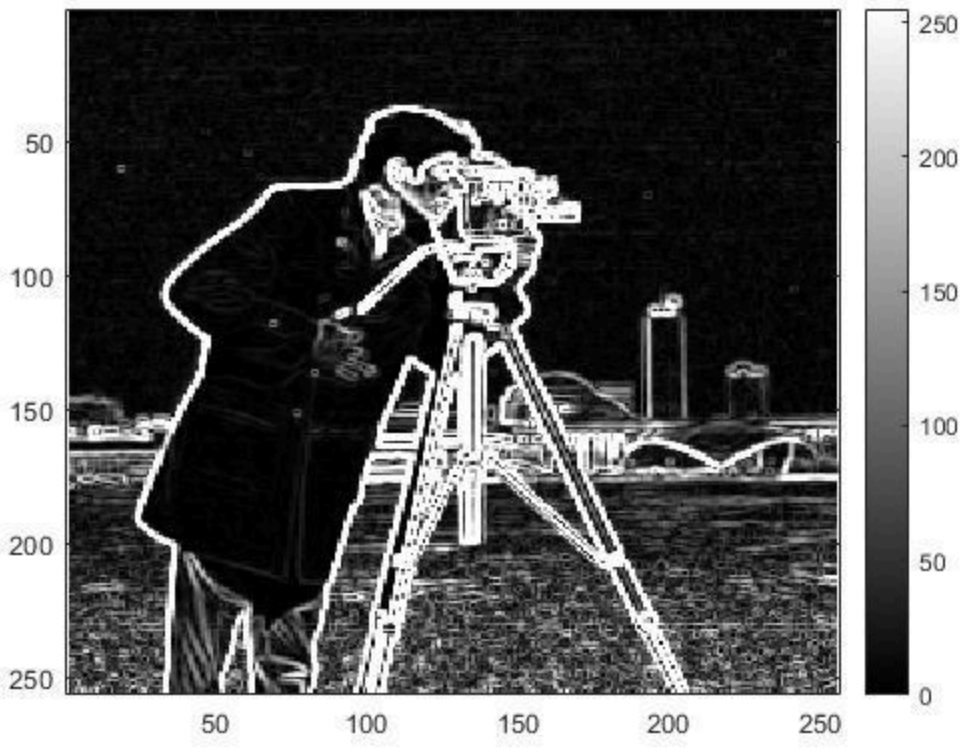
A bandpass filter is a original image minus a bandreject.

```
imagesc(camera - (camera - (gausscamera3 + (camera - meancamera7))));
colormap(gray);
colorbar;
```

Q4 - The Sobel Filter

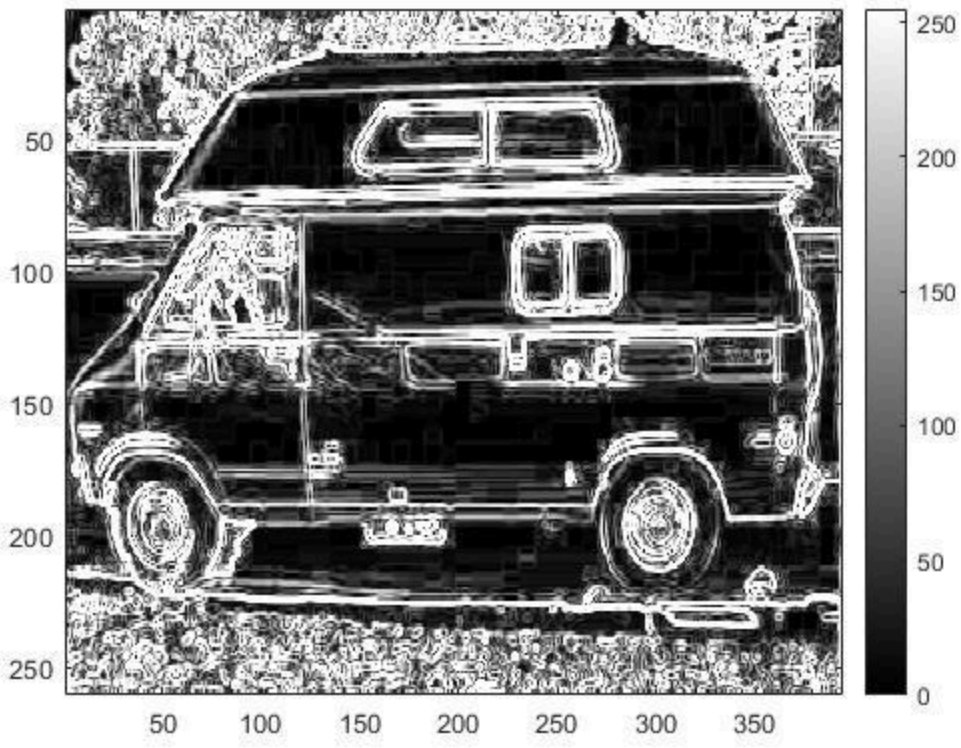
Lets apply the filter on the cameraman image. Since this is a directional kernel, some arithmetic must be done on the two images.

```
h7 = fspecial('sobel');
camerasobelx = imfilter(double(camera), h7);
camerasobely = imfilter(double(camera), h7');

figure;
imagesc(uint8(sqrt(camerasobelx.^2 + camerasobely.^2)));
colormap(gray);
colorbar;
```



We also do the same of the wagon image

```
wagonsobelx = imfilter(double(wagon), h7);
wagonsobely = imfilter(double(wagon), h7');

figure;
imagesc(uint8(sqrt(wagonsobelx.^2 + wagonsobely.^2)));
colormap(gray);
colorbar;
```

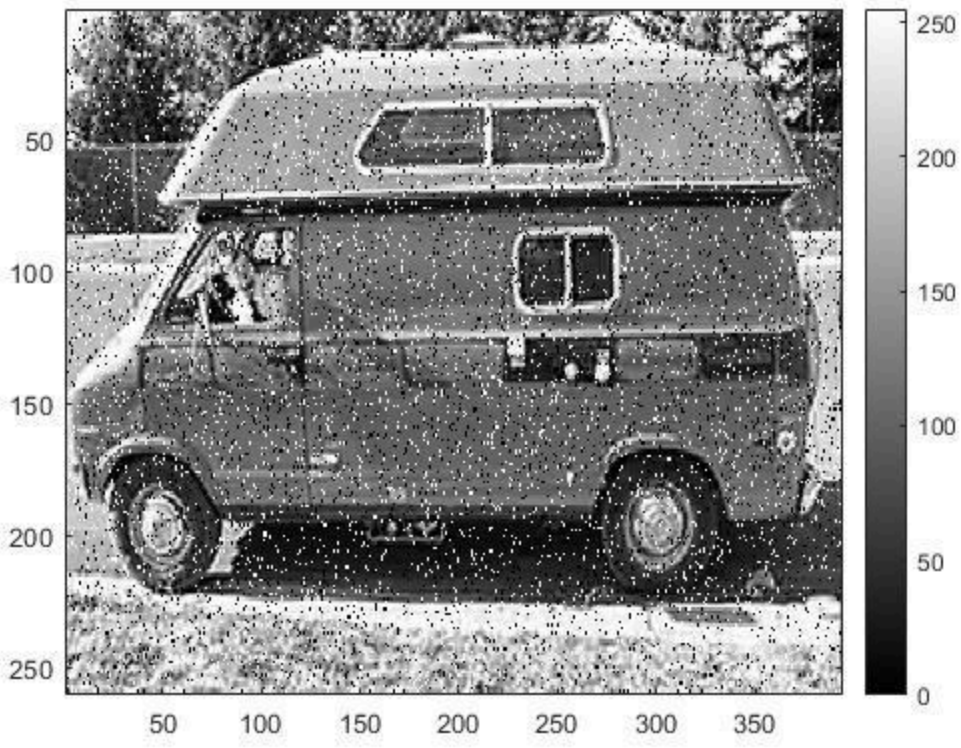


Q5 - Median filter.

We will perform median filtering on the image. Let's plot both the original image and the filtered one.

Below is the original image with pepper and salt noise.

```
figure;
imagesc(wagonnoise);
colormap(gray);
colorbar;
```



And the median filtered image.

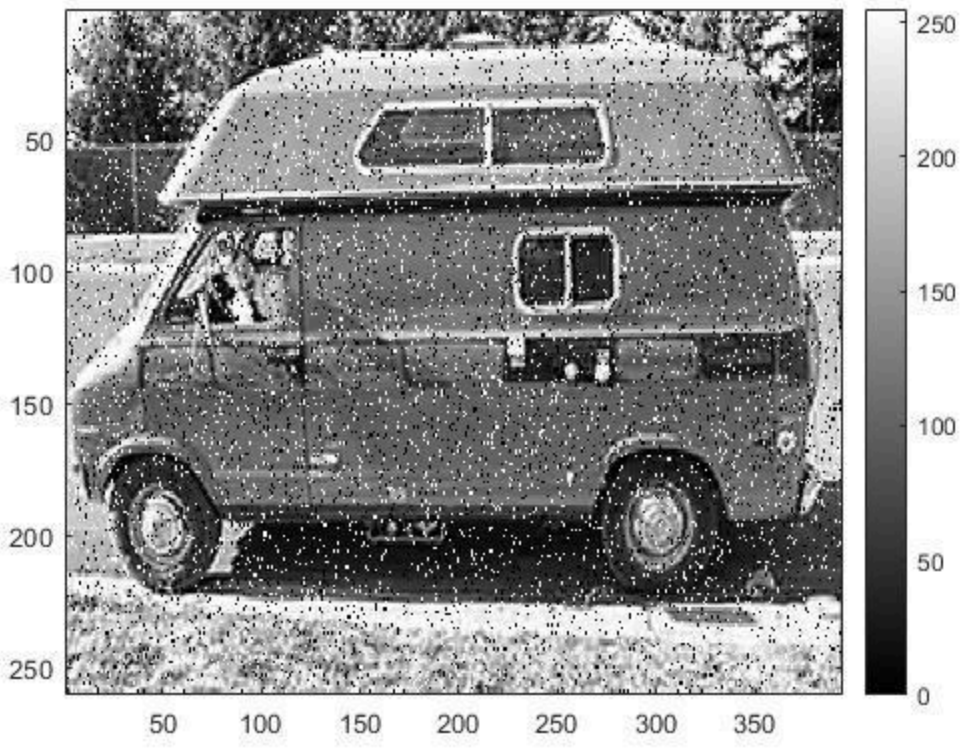
```
figure;
imagesc(medfilt2(wagonnoise));
colormap(gray);
colorbar;
```

Q6 - Comparing filters

Let us compare the gaussian filtered image with the median filtered image and mean filtered image.

Below is the original image.

```
figure;
imagesc(wagonnoise);
colormap(gray);
colorbar;
```



And now we use the median filter

```
figure;
imagesc(medfilt2(wagonnoise));
colormap(gray);
colorbar;
```

The mean filter of size 3×3

```
figure;
h8 = fspecial('average', 3);
imagesc(imfilter(wagonnoise, h8));
colormap(gray);
colorbar;
```

And lastly, the gaussian filter with $\sigma = 2$

```
figure;
imagesc(imgaussfilt(wagonnoise, 1));
colormap(gray);
colorbar;
```

What we see is that for this particular noise, the median filter works really well, this is due to the median being a "robust" estimator of centrality and is not easily skewed by outliers. So when there appears some extreme values in the neighbourhood of the pixel, the median still estimates a more appropriate value as it is not skewed by the extremes.

Q7 - Median Filter complexity

The median filter is a nonlinear filter and specifically an order-statistic filter. Which means that before calculating the new pixel value, the pixel values in the neighbourhood must be sorted.

Q8 - Median Filter Implementation

Below is my implementation of a median filter.

```
m = mymedianfilt(wagonnoise);

figure;
imagesc(m);
colormap(gray);
colorbar();
```

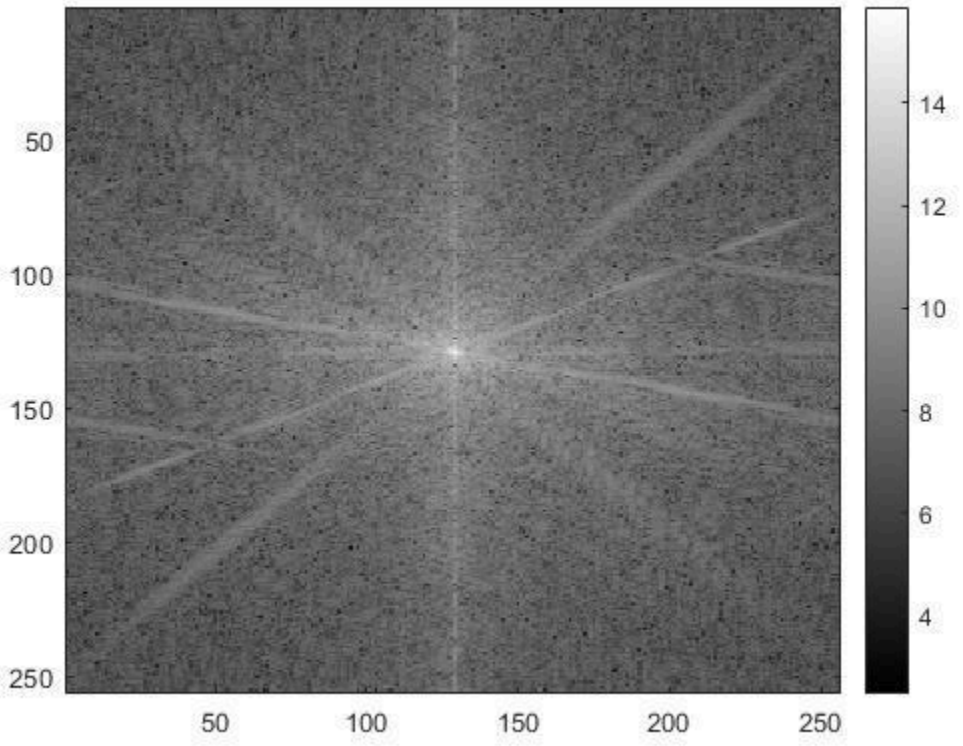
Q9

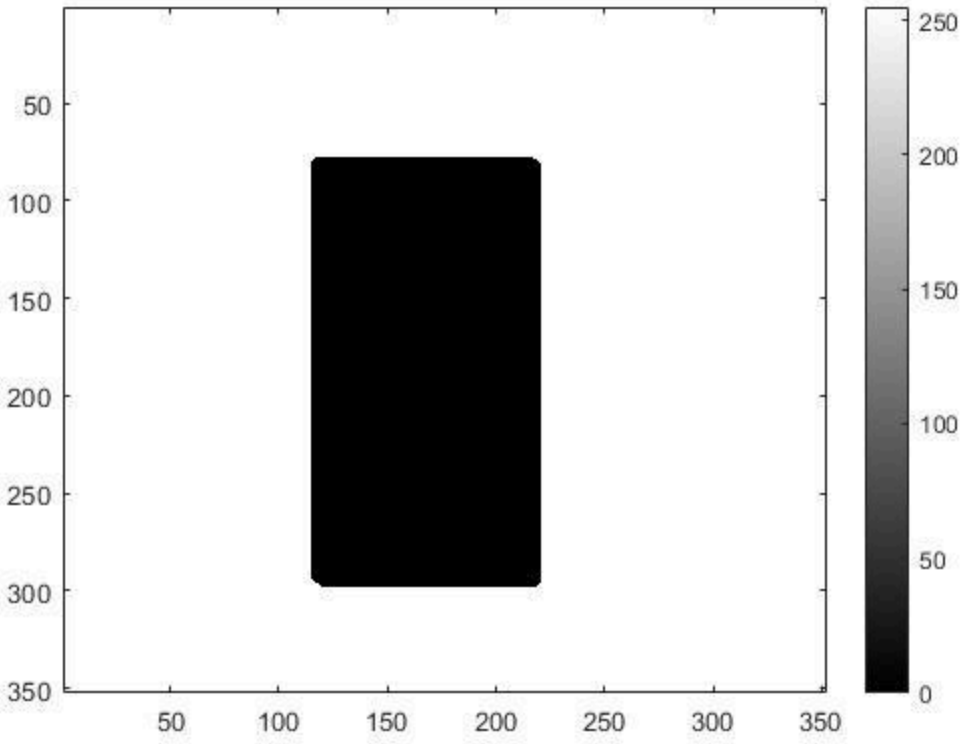
You get a black border due to MATLAB having to access pixel values outside the border of the image, and the default is to use zero padding.

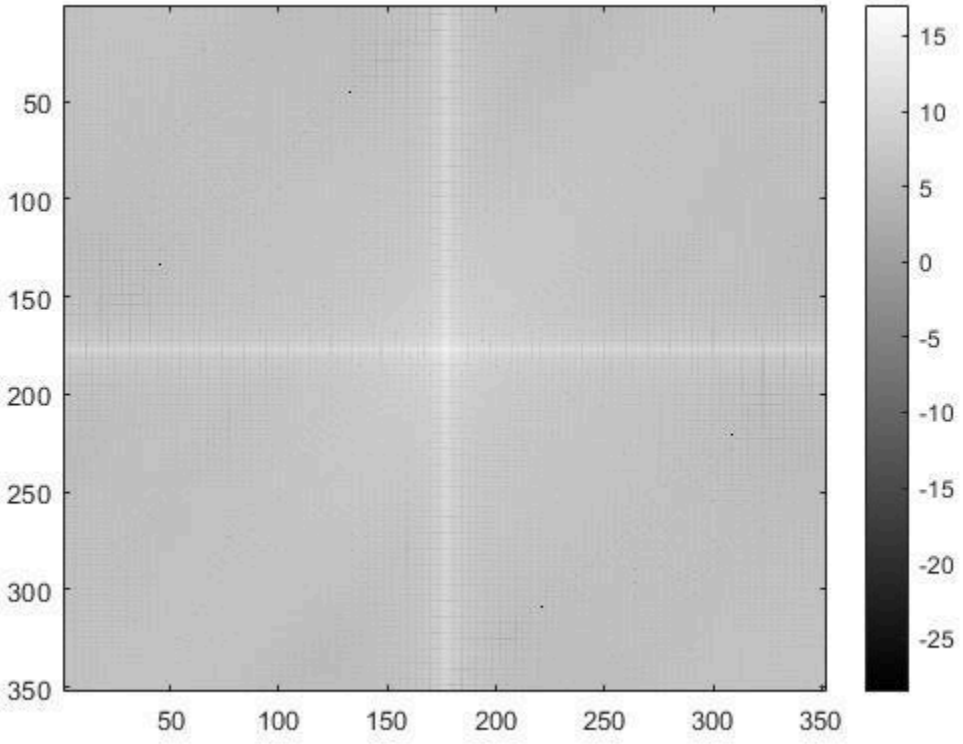
Q10

```
f = fftshift(fft2(camera));
figure; imagesc(camera); colormap(gray); colorbar();
figure; imagesc(log(abs(f))); colormap(gray); colorbar();

f = fftshift(fft2(rectangle));
figure; imagesc(rectangle); colormap(gray); colorbar();
figure; imagesc(log(abs(f))); colormap(gray); colorbar();
```


50
100
150
200
250
50
100
150
200
250
250
200
150
100
50


50
100
150
200
250
50
100
150
200
250
14
12
10
8
6
4


50
100
150
200
250
300
350
50
100
150
200
250
300
350
250
200
150
100
50
0


50
100
150
200
250
300
350
50
100
150
200
250
300
350
15
10
5
0
-5
-10
-15
-20
-25

Q11 - FFT and IFFT properties

We see that the transform below has symmetric values.

```
f = fftshift(fft2(rand(1,5)));
f
```

```
f =

  Columns 1 through 4

   0.6251 + 0.9903i   0.0235 + 0.2559i   2.8201 + 0.0000i   0.0235 -
 0.2559i

  Column 5

   0.6251 - 0.9903i
```

Now we will do sme filtering. We set $f(1,2) = 0$

```
f(1, 2) = 0;
f(1, 4) = 0;
im = ifft2(ifftshift(f));
im
```

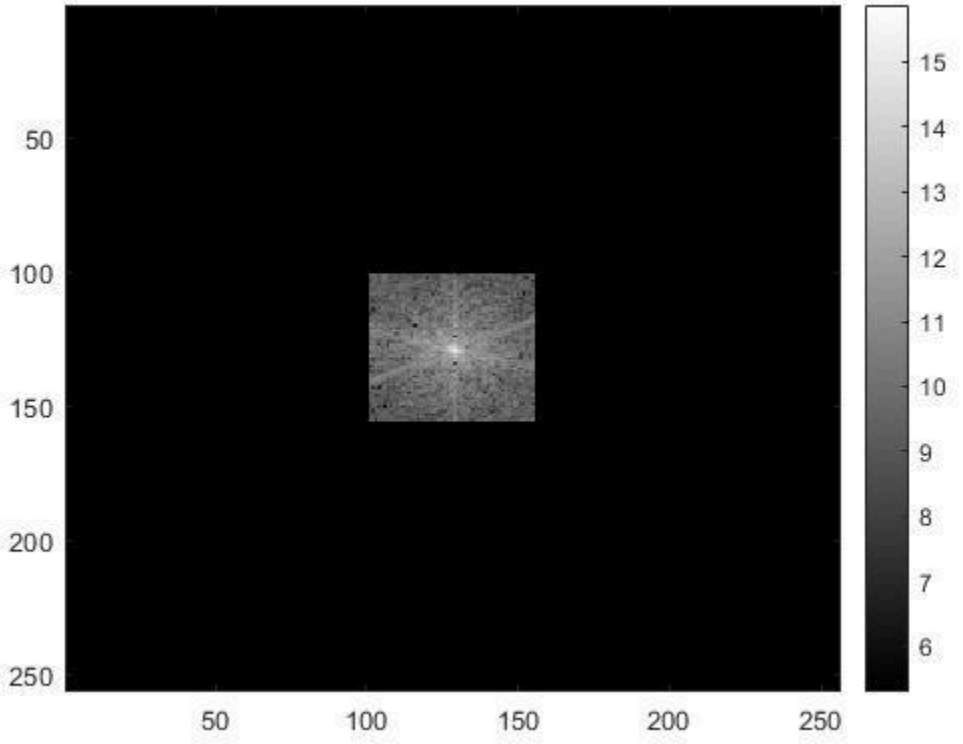
```
im =

    0.8140    0.5946    0.2645    1.0180    0.1289
```

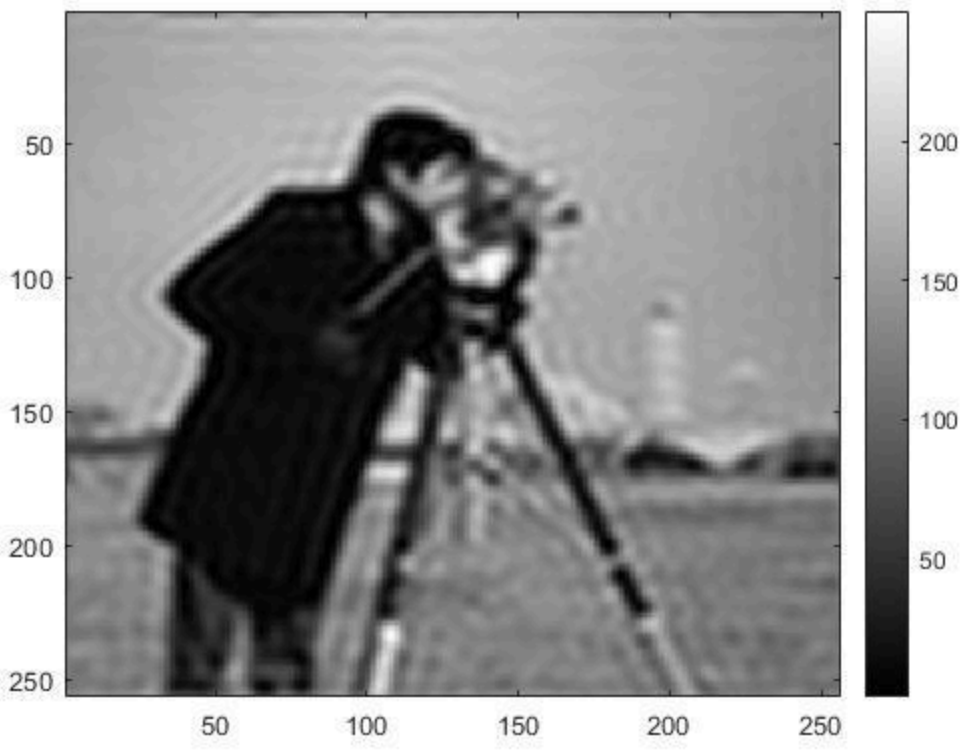
We see that if we dont filter symmetrically, we get a complex valued image.

Q12 - Filter Cameraman

```
figure; imagesc(camera); colormap(gray); colorbar();

f = fftshift(fft2(camera));
f(1:100, 1:end) = 0;
f(end-100:end, 1:end) = 0;
f(1:end, end-100:end) = 0;
f(1:end, 1:100) = 0;

figure; imagesc(log(abs(f))); colormap(gray); colorbar();

newim = ifft2(ifftshift(f));
figure; imagesc(abs(newim)); colormap(gray); colorbar();
```


50
100
150
200
250
50
100
150
200
250
250
200
150
100
50


50
100
150
200
250
50
100
150
200
250
15
14
13
12
11
10
9
8
7
6



Close all figures

```
close all;
```

Functions

Median Filter

```
function m = mymedianfilt(img)
    [a, b] = size(img);
    newI = zeros(a, b,'double');
    padded = padarray(img, [1, 1], median(img(:)));

    for i = 2:a
        for j = 2:b
            med = zeros(1, 9, 'double');
            it = 0;
            for k = 1:3
                for l = 1:3
                    it = it + 1;
                    med(it) = padded(i+k-1, j+l-1);
                end
            end
            newI(i,j) = median(med);
        end
    end
```

```
    m = newI;
end

% PaddedSize
function PQ = paddedsize(AB, CD, PARAM)
    if nargin == 1
        PQ = 2*AB;
    elseif nargin == 2 && -ischar(CD)
        PQ = AB + CD - 1;
        PQ = 2*ceil(PQ/2);
    elseif nargin == 2
        m = max(AB);  %  Maximum  dimension.

        %  Find  power-of-2 at least twice m.
        P = 2^nextpow2(2*m);
        PQ  =  [P,P];
    elseif (nargin == 3) && strcmpi(PARAM, 'pwr2')
        m  =  max([AB CD]);  %  Maximum  dimension.
        P  =  2^nextpow2(2*m);
        PQ  =  [P, P];
    else
        error('Wrong number of inputs.')
    end
end
```

Published with MATLAB® R2021a