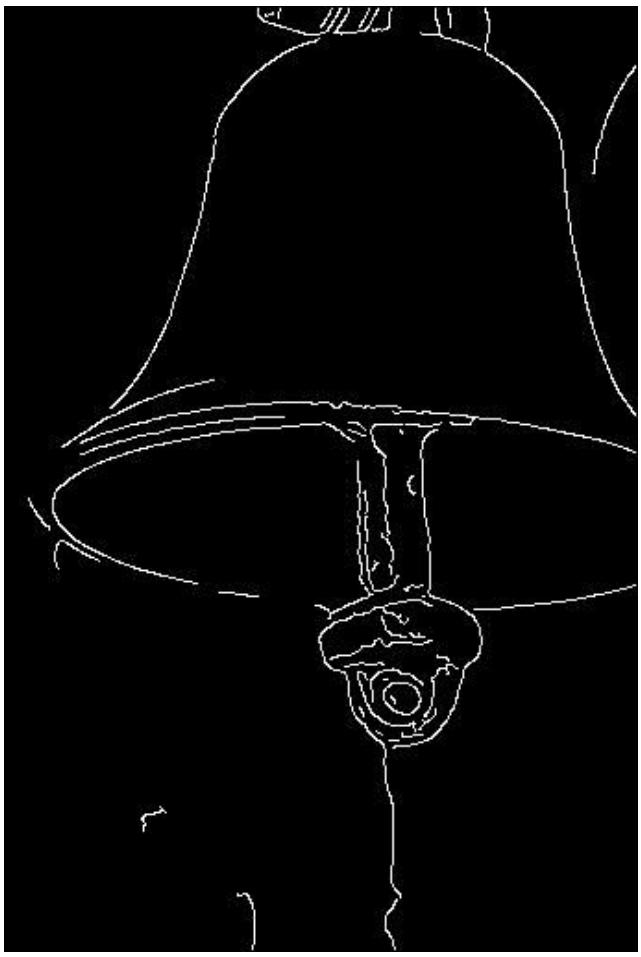
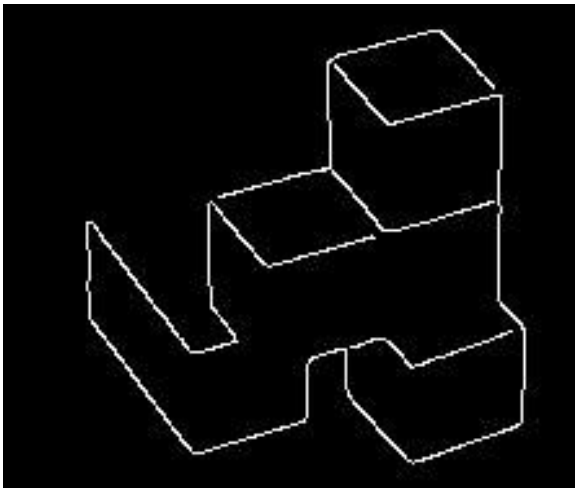
Digital Image Processing
Assignment 2
Meher Shashwat Nigam (20171062)

Assignment 2

QUESTION 1 (q1.m)

Part 1





The following output was obtained by using the canny edge detection function on bell.jpg and cubes.png; using matlab's edge function. Tried various values till I arrived at this result.\

```
% Canny edge detection

canny_bell = edge(rgb2gray(bell),'Canny',[0.1 0.2]);

imwrite(canny_bell,'../output_data/canny_bell.jpg');

canny_cubes = edge(rgb2gray(cubes),'Canny',[0.1 0.25]);

imwrite(canny_cubes,'../output_data/canny_cubes.jpg');
```

The following minVal and maxVal values gave better results.

Part 2

Roberts filter:

```
% Roberts Filter

Rx = [0 1;-1 0];

Ry = [1 0; 0 -1];
```

Prewitt's filter:

```
%  Prewitt Filter

Px = [-1 0 1;-1 0 1;-1 0 1];

Py = [ 1 1 1; 0 0 0;-1 -1 -1];
```

Sobel Filter:

```
%  Sobel Filter

Sx = [-1 0 1;-2 0 2;-1 0 1];

Sy = [ 1 2 1; 0 0 0;-1 -2 -1];
```

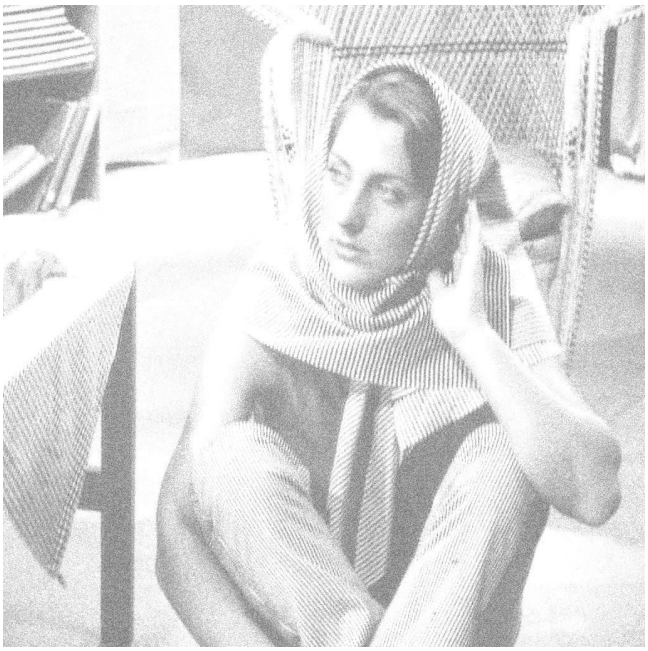
Laplacian Filter :

```
% Laplacian filter

L1 = [0 1 0; 1 -4 1; 0 1 0];

L2 = [1 1 1; 1 -8 1; 1 1 1];
```

Canny’s edge detection:

Part 3

Added gaussian noise to barbara.png.



Roberts:

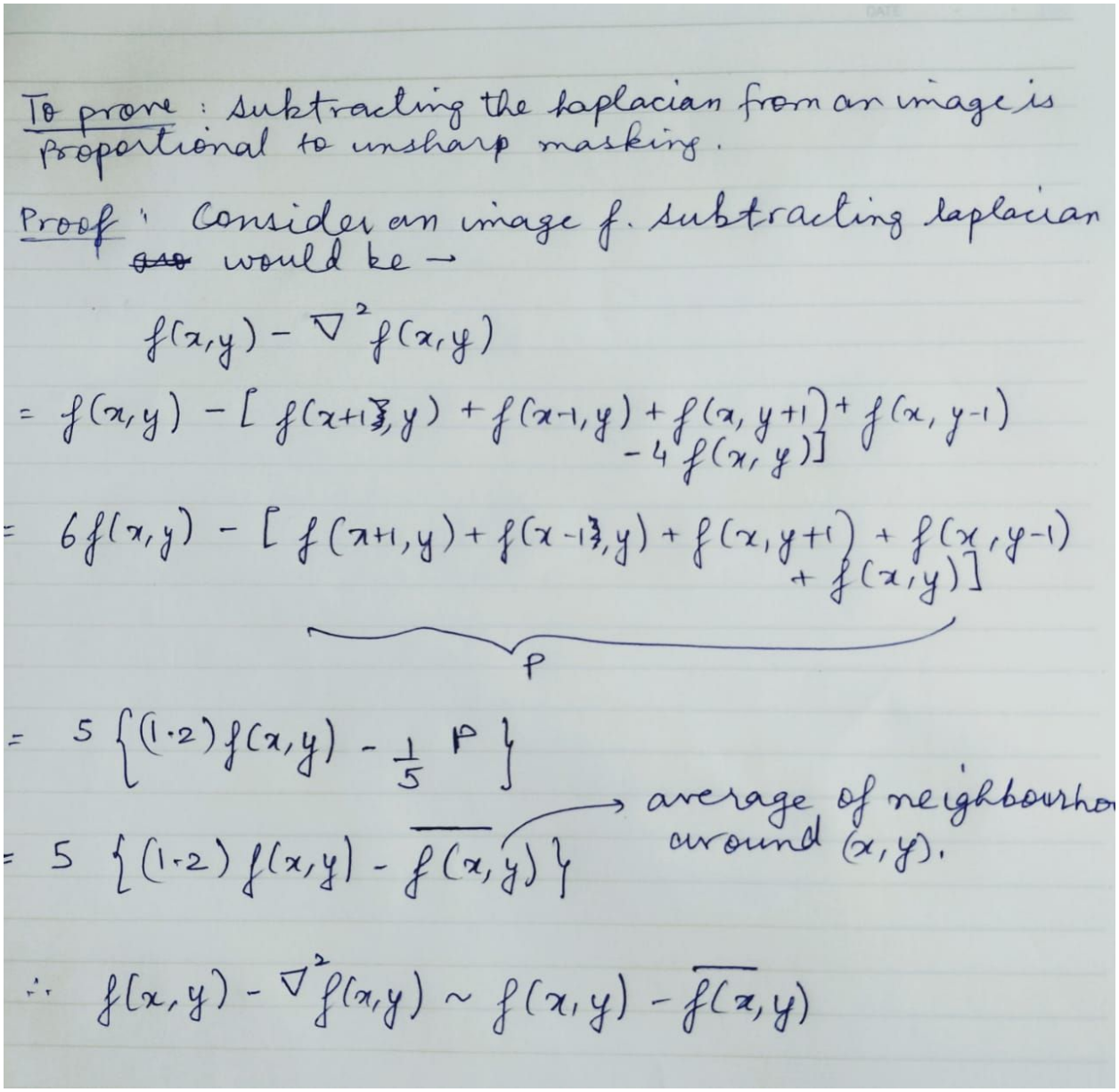
Prewitt:

Sobels:

Laplacian:

Canny:

QUESTION 2



To prove : subtracting the laplacian from an image is proportional to unsharp masking.

Proof : Consider an image f. subtracting laplacian ~~also~~ would be →

$$f(x,y) - \nabla^2 f(x,y)$$

$$= f(x,y) - [f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)]$$

$$= 6f(x,y) - [f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) + f(x,y)]$$

$$= 5\{(1.2)f(x,y) - \frac{1}{5}P\}$$

$$= 5\{(1.2)f(x,y) - \overline{f(x,y)}\}$$

→ average of neighbourho around (x,y).

$$\therefore f(x,y) - \nabla^2 f(x,y) \sim f(x,y) - \overline{f(x,y)}$$

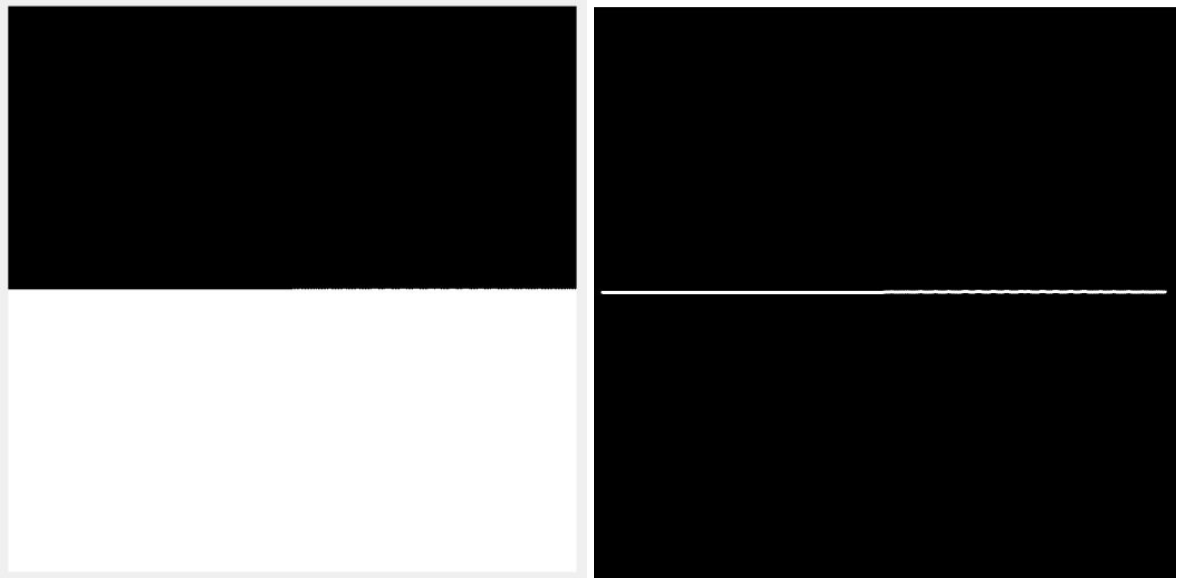
QUESTION 3

Part 1

A prewitt filter in the Y direction works for detecting an edge (transition from black and white).

```
Py = [ 1 1 1; 0 0 0;-1 -1 -1];
```

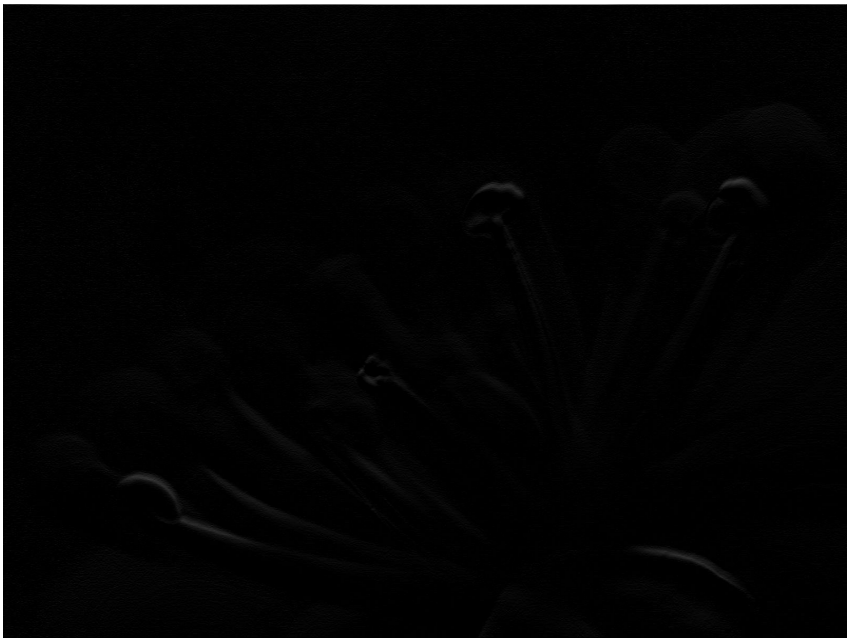
Output on convolving with box.png:



Part 2

```
%  Prewitt Filter

Px = [-1 0 1;-1 0 1;-1 0 1];

Py = [ 1 1 1; 0 0 0;-1 -1 -1];
```

Convolving with these two on blur.jpg :



Observation:

Px is able to detect horizontal edges in the image, while Py is able to detect vertical edges, to get a combination of both we can take a euclidean sum of both the images.

QUESTION 4 (q4.m)

-> The input image (bell.jpg). Outputs:

Following are the outputs for the following kernels used for convolution:

For the first figure (Window size =5)

-1	-1	-1	-1	-1
-1	-1	-1	-1	-1
-1	-1	99	-1	-1
-1	-1	-1	-1	-1
-1	-1	-1	-1	-1

For the second figure (Window size =3)

-1	-1	-1
-1	17	-1
-1	-1	-1

Part 2

Input and output image obtained using the following filter:

$$\begin{matrix} -1 & -1 & -1 \\ -1 & 17 & -1 \\ -1 & -1 & -1 \end{matrix}$$

Part 3

How is high boost filtering different from bilateral filtering ?

- High boost filtering is a second derivative sharpening filter which is used to enhance the high frequency components while keeping the low frequency components.
- A bilateral filter is a non-linear, edge-preserving, and noise-reducing smoothing filter for images. It replaces the intensity of each pixel with a weighted average of intensity values from nearby pixels.

Difference between the two filters:

The difference between a bilateral filter and high boost filter is that high boost filter preserves the edges, but doesn't look consider the intensities, whereas bilateral takes into consideration both the intensities and the edges, hence preserving the edges and also maintaining the intensity.

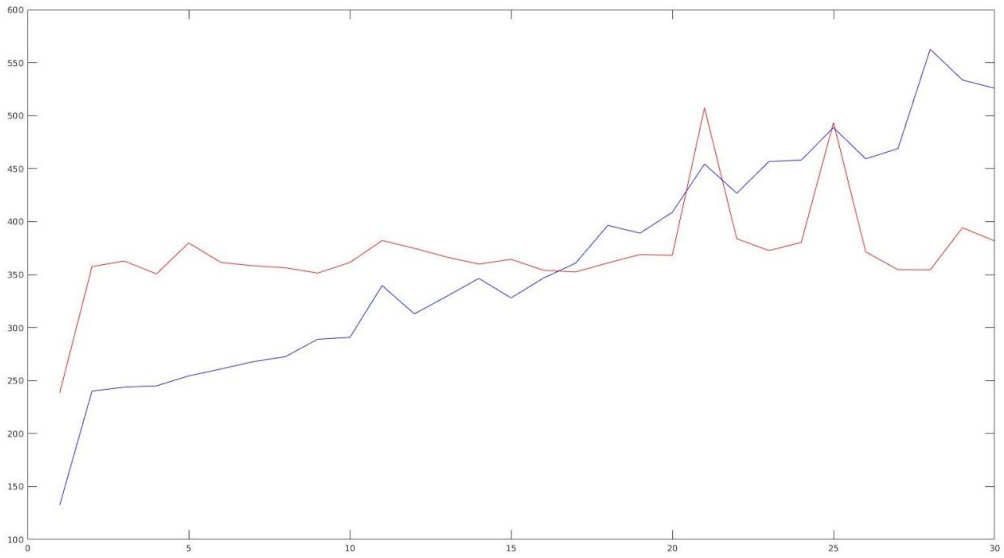
QUESTION 5

Part 1

Basically a convolution with a k*k filter (ones(k,k)/k*k).

Used the auxiliary file convolve_with.m to do the same.

Part 2(plot_average.py, EfficientAverageFilter.m)



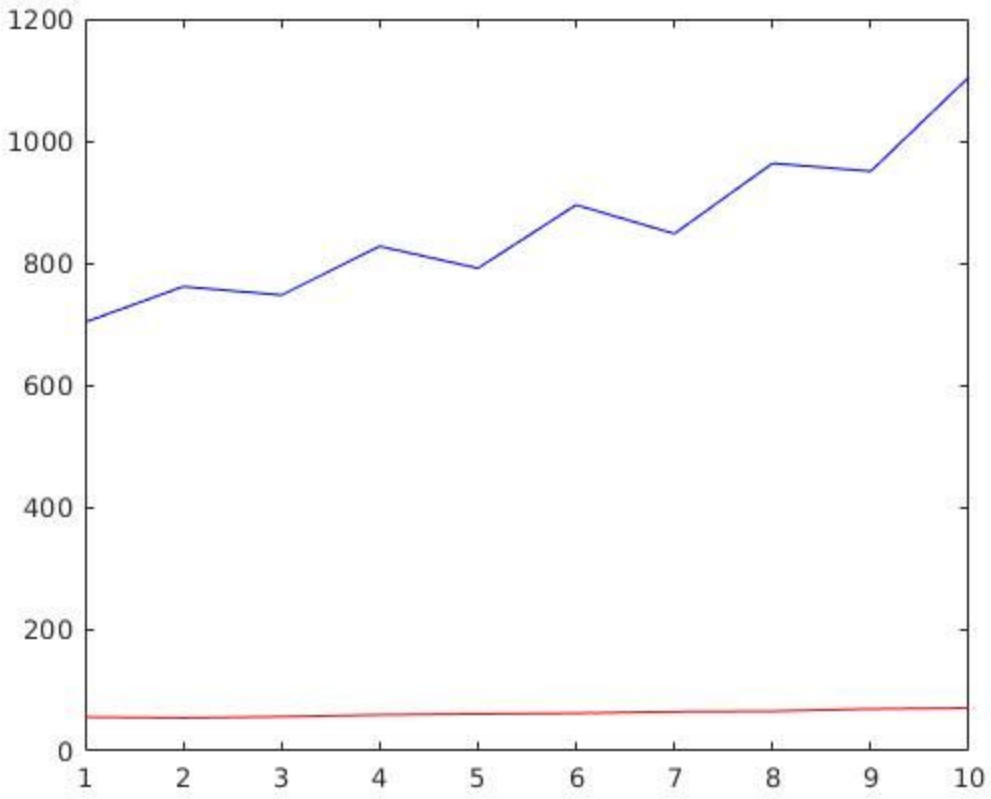
The Y axis is a scaled version of time taken for running the algorithm.

The X axis denotes the kernel size. I ran it for kernel size 1 to 30.

The purple line denotes the normal average filter implementation, the red line denotes the efficient implementation. As the filter is moved from one spatial location to the next one, the filter window shares many common pixels in adjacent neighborhoods. This observation is exploited to avoid wasteful computations.

The effect of the efficient computation is only evident at higher kernel sizes, at lower sizes, normal implementation still performs better.

Part 3 (median_efficient.py)



The Y axis is a scaled version of time taken for running the algorithm.

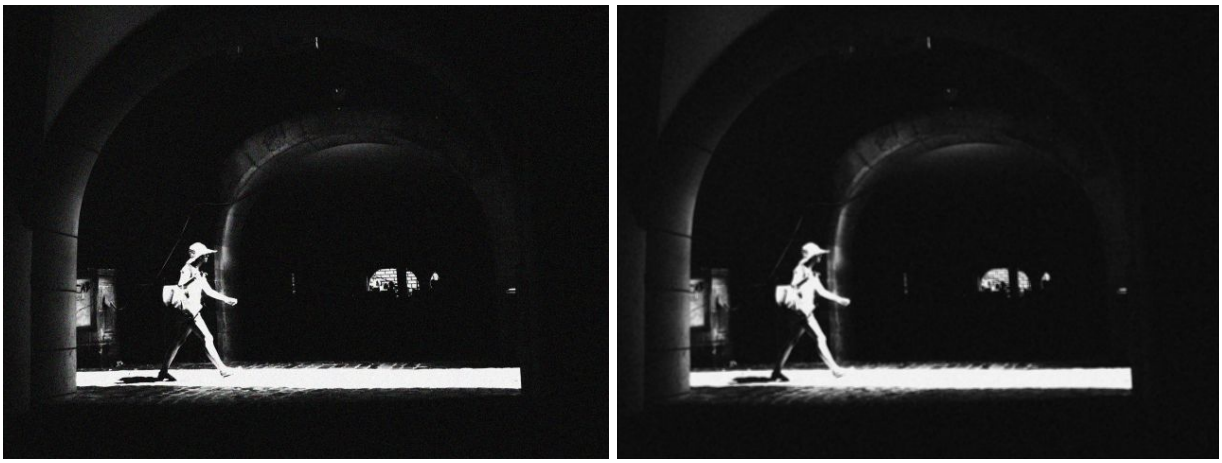
The X axis denotes the kernel size. I ran it for kernel size 1 to 10.

The purple line denotes the normal median filter implementation, and the red line denotes the efficient implementation of the median filter.

QUESTION 6

Part 1 (bilateral_filter.m)

Input and output images after applying bilateral filtering(sigd = 25,sigr= 60)



Input and output images after applying bilateral filtering(sigd = 60,sigr= 15)

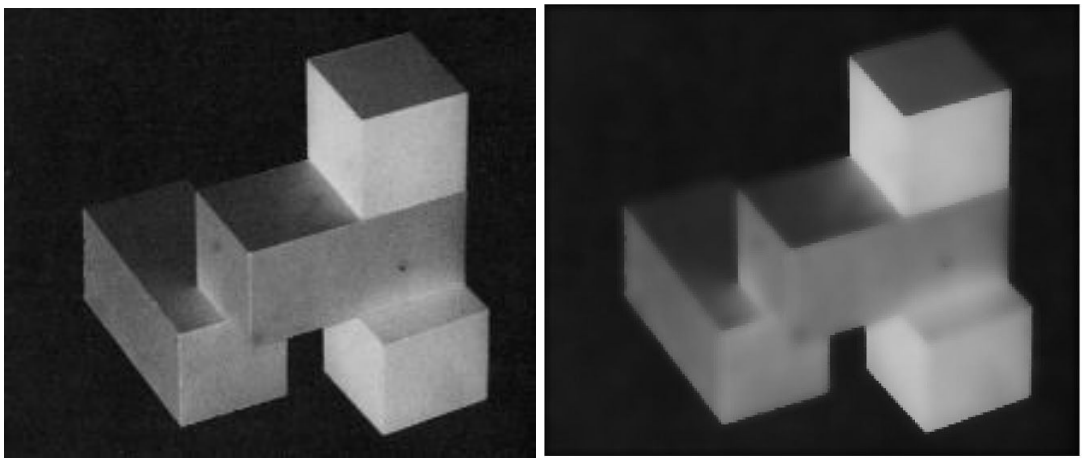
I found the lowest L2 distance from ground truth at the given sigma values.

QUESTION 7

Part 1

-> Output image from cross bilateral filter(kernel size = 7, sigd = 50,sigr =50).

Part 2



Input and output images after inverse bilateral filtering.(kernel size= 3,sid,sigr=30).

The applications of Inverse Bilateral Filtering are:

- They can be used in extracting salient features in images.
- They can also be used to differentiate between out-of-focus and in-focus images.

QUESTION 8

The given images are the original image and the degraded image respectively. We can clearly see it is a type of salt and pepper noise. A median filter would help remove it. Using median filter:

 -> The cleaned image I obtained (kernel size=3).