CSE585/EE555: Digital Image Processing II
Computer Project # 3
Non-Linear Filtering and Anisotropic Diffusion

Saptarashmi Bandyopadhyay ; Bharadwaj Ravichandran

03/29/2019

---

1 Objectives

The main objective of this project is to implement 4 different non-linear filters: median, alpha-trimmed mean, sigma, symmetric nearest neighbor mean, and the Anisotropic Diffusion algorithm and finally analyze the filtered images qualitatively and quantitatively.

2 Code Organization

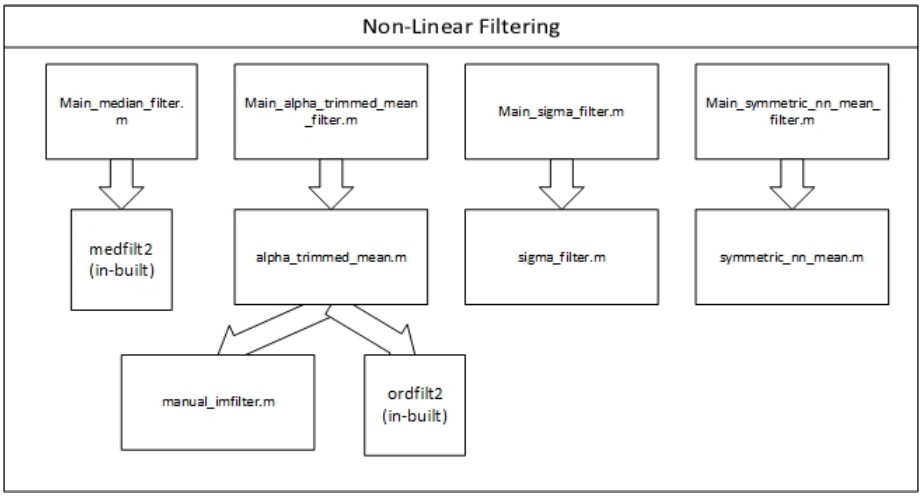


Figure 1: Block diagram and code flow for Non-Linear Filtering

Figure 1 shows the overall code flow of the Non-Linear Filtering. Each filter implementation has its own Demo Main script for setting up the parameters and a function script that implements the algorithm. The manual_imfilter.m is just a manual implementation of MATLAB's imfilter function. We made this script in order to avoid using inbuilt functions that are not specified in the project description.

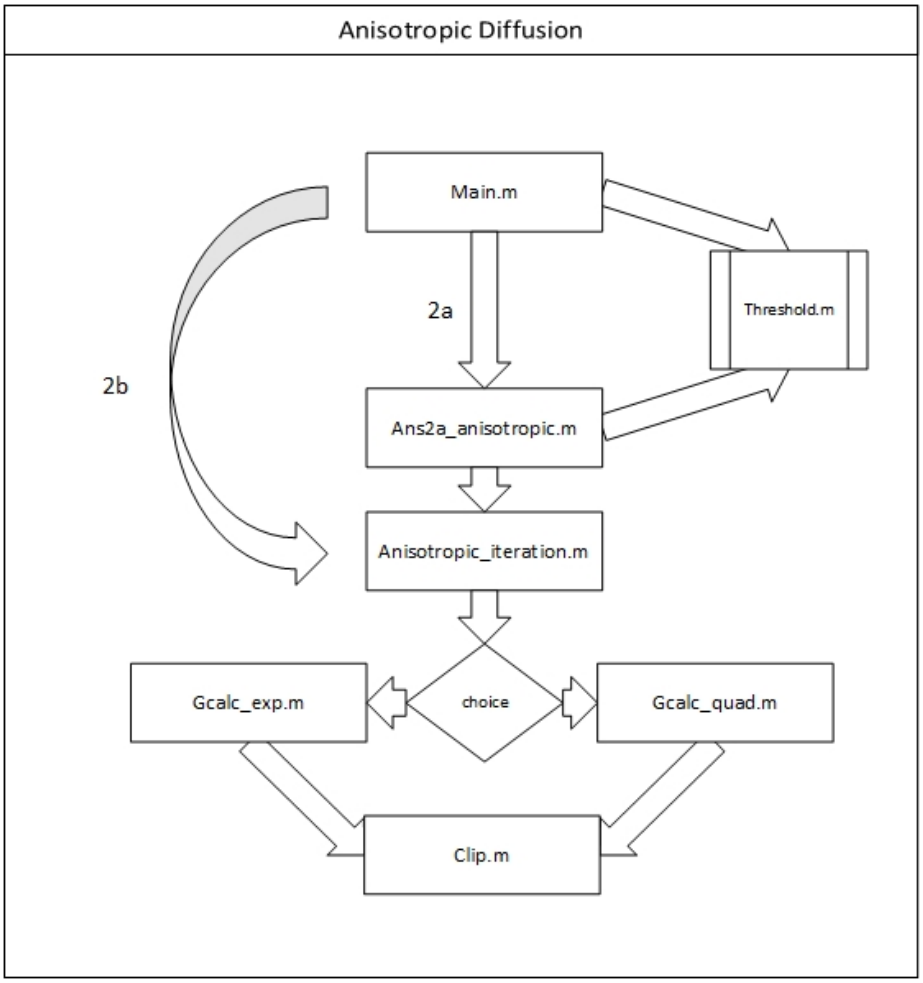


Figure 2: Block diagram and code flow for Anisotropic Diffusion

Figure 2 shows the overall code flow of the Anisotropic Diffusion. The main control flow is in the MATLAB function ain.m. The code anisotropic_iteration.m executes the non linear filtering method of Anisotropic Diffusion. The exponential g(.) and the inverse quadratic g(.) have been implemented in gcalc_exp.m and gcalc_invquad.m respectively. The other MATLAB codes like clip.m and threshold.m have been described in details in the Meth-

ods Section. Ans2a_anisotropic.m is implemented separately as the histograms, 2D plots and segmented images are not required to be generated in Question 2b.
The code for Non-Linear Filtering and Anisotropic Diffusion are stored in separate subfolders and each subfolder has its own README.md file on how to run the MATLAB code.

3 Methods

3.1 Median Filter

The median filter considers the middle value in a sorted sequence of pixel values inside a given window of length $m \times n$. In our case, the filter is a square of size $n \times n$. So, the median filter sorts the pixel values and chooses the $(\lfloor \frac{n^2}{2} \rfloor + 1)$th highest value as the output of the filtered window region of the image. Like this, the filter slides over the entire image, giving as a filtered image. Also, an important thing to note is that the alpha-trimmed mean filter acts as a median filter if $\alpha = 0.5$. In this project, the median filter is implemented using the **medfilt2** matlab function. So, a median filter of size 5×5 can be implemented as follows:

$$medfilt2(im, [55])$$
$$or$$
$$ordfilt2(im, 13, ones(5, 5))$$

The padding method is set to zero-padding by default for both medfilt2 and ordfilt2.

3.2 Alpha-Trimmed Mean Filter

The alpha-trimmed mean filter is a type of L-filter which calculates the mean of the pixel values over the range specified by the α value. The highest and lowest values are trimmed. In this project, the α value is set to 0.25. If $\alpha = 0$, then the filter operates as a mean filter. If $\alpha = 0.5$, the filter operates as a median filter . The alpha -trimmed mean filter is less influenced by salt/pepper noise than the mean filter. The mathematical description of the Alpha-Trimmed Mean Filter is as follows:

$$(x_{(k-N)}, x_{(k-N+1)}, ..., x_{(k)}, ..., x_{(k+N)})$$

$$\downarrow$$

$$(x_{(1)}, x_{(2)}, ..., x_{(N+1)}, ..., x_{(2N+1)})$$

$$y_k = \Big(\frac{1}{n - 2\lfloor \alpha n \rfloor}\Big) \sum_{i=\lfloor \alpha n \rfloor + 1}^{n-\lfloor \alpha n \rfloor} x_{(i)}$$

$$n = 2N + 1 \rightarrow \text{window length}$$

$$0 \leq \alpha \leq 0.5$$

The pseudocode of alpha_trimmed_mean.m is as follows:

Function: alpha_trimmed_mean (im, alpha,n)

Objective: To perform the alpha trimmed mean filtering operation on the image im using a filter of size n x n and an alpha value of 0.25

Input Parameters:
im: Grayscale Image

alpha: The parameter that decides the values to be considered for the averaging
n: Dimension of filter

Returned Result:
f: alpha-trimmed mean filtered image

Processing Flow:

```
f ← sliding window multiplication of im with ones[n x n]
for i = 1
    f ← f − ordfilt2(im,i,filter)
until ⌊αn²⌋

for j = n² − ⌊αn²⌋ + 1
    f ← f − ordfilt2(im,j,filter)
until n²

f ← f / (n²−2⌊αn²⌋)
```

Restrictions/Notes:

This function requires a grayscale image as input.

The following functions are called:
manual_imfilter.m (This function is a manual implementation of the imfilter inbuilt function)
ordfilt2(A,order,domain) replaces each element in A by the orderth element in the sorted set of neighbors specified by the nonzero elements in domain. domain here is ones(5,5)

The following functions are calling:

Main_alpha_trimmed_mean_filter.m

3.3 Sigma Filter

The Sigma filter is related to the Alpha-Trimmed Mean filter. Unlike the Alpha-trimmed mean filter, the sigma filter does not require sorting. Given a value of standard deviation (σ), the sigma filter chooses the pixels in the neighborhood within 2σ of the pixel of interest and averages based on the number of pixels chosen. The

mathematical description of the Sigma Filter is as follows:

$$(x_{(k-N)}, x_{(k-N+1)}, ..., x_{(k)}, ..., x_{(k+N)})$$
$$\downarrow$$
$$\hat{y_k} = \frac{1}{N_C} \sum_{i=-N}^{N} \delta_i x_{k-i}$$
$$\delta_i = \begin{cases} 1 & , \ |x_{k-i} - x_k \leq 2\sigma| \\ 0 & , \text{otherwise} \end{cases}$$
$$N_C \triangleq \# \text{ of points } x_{k-i} \text{ having } \delta_i = 1$$

If $x_k \epsilon$ Gaussian Distribution $N(x_k, \sigma)$, then the condition equation for δ_i selects points (which belong to same distribution as x_k) within $\pm 2\sigma$ of x_k

The pseudocode of sigma_filter.m is as follows:

Function: sigma_filter (im, n,sigma)

Objective: To perform the sigma filtering operation on the image im using a filter of size n x n and a sigma value of 20.

Input Parameters:
im: Grayscale Image
n: Dimension of filter
sigma: specified standard deviation value.

Returned Result:
result: sigma filtered image

Processing Flow:

```
[x,y] ← size of image after zero padding
for i = ⌊n/2⌋ + 1
    for j = ⌊n/2⌋ + 1
        a ← 1
```

```
        for k = ⌊n/2⌋
            b ← 1
            for l = ⌊n/2⌋
                if |im(i − k, j − l) − im(i, j)| ≤ 2σ
                    δ(a, b) ← 1
                else
                    δ(a, b) ← 0
                b ← b + 1
                l ← l − 1
            until − ⌊n/2⌋
            a ← a + 1
            k ← k − 1
        until − ⌊n/2⌋
        filter ← im(i − ⌊n/2⌋ : i + ⌊n/2⌋, j − ⌊n/2⌋ : j + ⌊n/2⌋) .∗ δ
        result(i − ⌊n/2⌋, j − ⌊n/2⌋) ← (sum over all pixels of filter) / (number of 1s in δ)
        j ← j + 1
    until y − ⌊n/2⌋
    i ← i + 1
until x − ⌊n/2⌋
```

Restrictions/Notes:

This function requires a grayscale image as input.

The following functions are calling:

Main_sigma_filter.m

3.4 Symmetric Nearest-Neighbor Mean Filter

1) For each pair of points symmetrically opposite each other

$$\{x_{(k-i\ ,\ l-j)}, x_{(k+i\ ,\ l+j)}\},$$

select point most similar to $x_{k,l}$, i.e., select point x that minimizes $|x - x_{k,l}|$.

2) Average together selected points and assign to $x_{k,l}$

The pseudocode of symmetric_nn_mean.m is as follows:

Function: symmetric_nn_mean (im, n)

Objective: To perform the Symmetric Nearest Neighbor Mean filtering operation on the image im using a filter of size n x n.

Input Parameters:
im: Grayscale Image
n: Dimension of filter

Returned Result:
result: symmetric nearest neighbor mean filtered image

Processing Flow:

```
[x,y] ← size of image
ykl ← empty vector of size [1x ⌈n²/2⌉ ]
for i = ⌊n/2⌋+1
    for j = ⌊n/2⌋+1
        ykl_count ← 1

        for k = 1
            for l = - ⌊n/2⌋
                if |im(i+k,j+l) - im(i,j)| < |im(i-k,j-l) - im(i,j)|
                    ykl(ykl_count) ← im(i+k,j+l)
                else
                    ykl(ykl_count) ← im(i-k,j-l)
                ykl_count ← ykl_count + 1
                l ← l + 1
            until ⌊n/2⌋
            k ← k + 1
        until ⌊n/2⌋

        for k = ⌊n/2⌋
            if |im(i,j-k) - im(i,j)| < |im(i,j+k) - im(i,j)|
                ykl(ykl_count) ← im(i,j-k)
            else
                ykl(ykl_count) ← im(i,j+k)
            ykl_count ← ykl_count + 1
            k ← k - 1
        until 1
        ykl(ykl_count) ← im(i,j)
        ykl_mean ← sum(ykl)/length(ykl)
        result(i,j) ← ykl_mean
        j ← j + 1
    until y - ⌊n/2⌋
```

```
    i ← i + 1
until x − ⌊n/2⌋
```

Restrictions/Notes:

This function requires a grayscale image as input.

The following functions are calling:

Main_symmetric_nn_mean_filter.m

3.5 Anisotropic Diffusion

Anisotropic Diffusion is an iterative non-linear filtering method which can be used to smooth regions while preserving the edges. In the project, the code for Anisotropic Diffusion has been executed in the MATLAB function anisotropic_iteration.m. The mathematical description of Anisotropic Diffusion is as follows:

$$I_t = c(x, y, t)\nabla^2 I + \nabla c(x, y, t) * \nabla I \tag{1}$$

where c(x,y,t) is a function to emphasize points (x,y) differently and thereby ensures piece-wise smoothing, t is the number of iterations and I is the input grayscale image. The image behavior in c(x,y,t) has been implemented by the g(.) function, which can be either exponential or an inverse quadratic function. The g(.) function has been described in Section 3.6. The equation representing the discrete implementation of Anisotropic Diffusion is as follows:

$$I_{i,j}^{t+1} = I_{i,j}^{t} + \lambda * [c_N^t \nabla_N I + c_S^t \nabla_S I + c_E^t \nabla_E I + c_W^t \nabla_W I]_{i,j}^{t} \tag{2}$$

where $I_{i,j}^0$ is the graylevel value of the original image at the (i,j) position, t changes from 0 to the Number of iterations - 1, $I_{i,j}^{t+1}$ is the image with enhanced graylevel value at the (i,j) position after t+1 iterations and λ is a parameter which is provided as 0.25 in the question. If $I_{i,j}^{t+1}$ becomes greater than 255 or less than 0, after computation in equation (2), the pixel value is clipped to 255 or 0 respectively. The clip mechanism has been described in Section 3.7. The ∇ and the c_N operators are expressed as follows:

$$\nabla_N I_{i,j}^t = I_{i-1,j}^t - I_{i,j}^t \tag{3}$$

$$\nabla_S I_{i,j}^t = I_{i+1,j}^t - I_{i,j}^t \tag{4}$$

$$\nabla_E I_{i,j}^t = I_{i,j+1}^t - I_{i,j}^t \tag{5}$$

$$\nabla_W I_{i,j}^t = I_{i,j-1}^t - I_{i,j}^t \tag{6}$$

$$c_N^t(i,j) = g(|\nabla_N I_{i,j}^t|) \tag{7}$$

$$c_S^t(i,j) = g(|\nabla_S I_{i,j}^t|) \tag{8}$$

$$c_E^t(i,j) = g(|\nabla_E I_{i,j}^t|) \tag{9}$$

$$c_W^t(i,j) = g(|\nabla_W I_{i,j}^t|) \tag{10}$$

The exponential and the inverse quadratic versions of the g(.) function have been used in the project which has been explained clearly in Section 3.6.

The pseudocode of anisotropic_iteration.m is as follows:

Function: anisotropic_iteration (I, iter,lambda,choice)

Objective: To execute Anisotropic Diffusion on a grayscale input image for a number of iterations with exponential g(.) or inverse-quadratic g(.)

Input Parameters:

I: Grayscale Image of typle double
iter: Number of iterations during Anisotropic Diffusion
lambda: 0.25, a parameter
choice: choice whether g(.) is exponential or inverse-quadratic

Returned Result:

I_t: enhanced image after t iterations

Processing Flow:

```
[I_t] ← initialized to input image I
[M,N] ← size of image I

for t=1
    for i=1
        for j=1
            if i==1 [1st row] or i==M [last row] or j==1 [1st column]
               or j==N [last column] ← edge cases
                if i==1 [1st row]
                    ∇_N I^t_{i,j}=0; ← No pixel position is above the 1st row
                    ∇_S I^t_{i,j}=I_t(i+1,j)−I_t(i,j);
                elseif i==M [last row]
                    ∇_N I^t_{i,j}=I_t(i−1,j)−I_t(i,j);
                    ∇_S I^t_{i,j}=0; ← No pixel position is below the last row
                else
                    ∇_N I^t_{i,j}=I_t(i−1,j)−I_t(i,j);
                    ∇_S I^t_{i,j}=I_t(i+1,j)−I_t(i,j);
                end
                if j==1 [1st column]
                    ∇_W I^t_{i,j}=0;←No pixel position is to the left of the 1st column
                    ∇_E I^t_{i,j}=I_t(i,j+1)−I_t(i,j);
                elseif j==N [last column]
                    ∇_E I^t_{i,j}=0; ← No pixel position is to the right of the last
                                       column
                    ∇_W I^t_{i,j}=I_t(i,j−1)−I_t(i,j);
                else
                    ∇_E I^t_{i,j}=I_t(i,j+1)−I_t(i,j);
                    ∇_W I^t_{i,j}=I_t(i,j−1)−I_t(i,j);
                end
            else
                ∇_N I^t_{i,j}=I_t(i−1,j)−I_t(i,j);
                ∇_S I^t_{i,j}=I_t(i+1,j)−I_t(i,j);
                ∇_E I^t_{i,j}=I_t(i,j+1)−I_t(i,j);
```

```
                    ∇_W I^t_{i,j}=I_t(i,j-1)-I_t(i,j);
                end
                if choice == 1 ← Exponential g(.)
                    c^t_N(i,j), c^t_N(i,j), c^t_N(i,j) and c^t_N(i,j) are computed by calling
                    gcalc_exp(respective image gradient) function
                elseif choice ==2 ← Inverse quadratic g(.)
                     c^t_N(i,j), c^t_N(i,j), c^t_N(i,j) and c^t_N(i,j) are computed by calling
                    gcalc_invquad(respective image gradient) function
                end
                I_tnext(i,j) = I_t(i,j) + λ*[(c^t_N(i,j)*∇_N I^t_{i,j}) + (c^t_S(i,j)*∇_S I^t_{i,j}) +
                                (c^t_E(i,j)*∇_E I^t_{i,j}) + (c^t_W(i,j)*∇_W I^t_{i,j}) ];
                I_tnext = clip(I_tnext);
            until N
        until M
        I_t = I_tnext;
    until iter
```

Restrictions/Notes:

This function requires a grayscale image as input of type uint8. However, while processing each pixel in the 5-pixel neighborhood are considered double type.

The following functions are calling:
main.m and ans2a_anisotropic.m

The following functions are called:

gcalc_exp.m, gcalc_invquad.m and clip.m

3.6 Image behavior in g(.)

The image behavior has been represented in the equation (1) as c(x,y,t). It can be computed by the g(.) function which can be either be exponential or inverse quadratic. The argument to the g(.) function is the norm of the gradient of the image. The norm of the gradient at each arc can be approximated with the absolute value of its projection along the direction of each arc which is $|\nabla_{direction} I^t_{i,j}|$ It is expected that g(s) is monotonically decreasing for $s >= 0$ and $g(0) = 1$. The mathematical

equation of the exponential g(.) is as follows:

$$g(|\nabla_{direction} I^t_{i,j}|) = e^{-(\frac{|\nabla_{direction} I^t_{i,j}|}{k})^2} \tag{11}$$

It preserves high-contrast edges. It has been executed by the function gcalc_exp.m. It has been called by the function anisotropic_iteration.m. The value of k is 25 for questions 2(a) and 2(b).

The mathematical equation of the inverse quadratic g(.) is as follows:

$$g(|\nabla_{direction} I^t_{i,j}|) = \frac{1}{1 + (\frac{|\nabla_{direction} I^t_{i,j}|}{k})^2} \tag{12}$$

It favours wide regions in an image. It has been executed by the function gcalc_invquad.m. It has been called by the function anisotropic_iteration.m. The value of k is 25 for questions 2(a) and 2(b).

Both the functions gcalc_exp.m and gcalc_invquad.m take $|\nabla_{direction} I^t_{i,j}|$ as input and return $c^t_{direction}(i,j)$ as output.

3.7 Clipping of the pixel values

The pixel value $I^{t+1}_{i,j}$ that has been computed in equation (2) can become greater than 255 or less than 0 during the computation. So,the pixel value has been clipped to 255 or 0 respectively in such cases. It has been executed by the code clip.m in MATLAB. The pseudocode of clip.m is as follows:

Function: clip (pixel)

Objective: To clip pixel values greater than 255 to 255 and less than 0 to 0.

Input Parameters:
pixel: pixel value

Returned Result:
I: clipped pixel

Processing Flow:

```
if pixel > 255
     I = 255
else if pixel <0
     I = 0
else
     I = pixel
end
```

Restrictions/Notes:

This function requires a scalar pixel value as input.

The following functions are calling:

anisotropic_iteration.m

3.8 Histograms and 2D plots

The histogram of the input image and the images after anisotropic diffusion with 5, 20 and 100 iterations are generated by the MATLAB function imhist. The histogram is generated in main.m and ans2a_anisotropic.m. The histogram has been analyzed to determine the threshold value to segment out the gray spokes of the wheel in the cwheelnoise image.

The 2D plot have been generated for the input image and the mages after anisotropic diffusion with 5, 20 and 100 iterations in the MATLAB codes main.m ans2a_anisotropic.m. For each image, at first the line y=128 is superimposed on it. This divides the image into the upper and the lower halves. Then, for each pixel

value along the line y=128, where the pixel positions of x change, the pixel values have been plotted using the MATLAB function plot. This gives an idea of the image behavior for the pixel positions along the middle of the image.

3.9 Segmentation of the image by manual thresholding

The original image and the images obtained after 5,20 and 100 iterations of Anisotropic Diffusion with exponential g(.) and inverse quadratic g(.) has been segmented to obtain the gray spokes in the cwheelnoise by manual thresholding. The threshold value is selected by observation of the histogram of the images along with trial-and-error method. The thresholding has been implemented by the MATLAB code threshold.m. The pseudocode of threshold.m is as follows:

Function: threshold (X,val)

Objective: To manually threshold the image X

Input Parameters:
X: grayscale input image
val: threshold value

Returned Result:
bX: image after thresholding

Processing Flow:

```
[M,N] ← size of image I
bX ← is initialized to a matrix of ones with size MxN
for i=1
    for j=1
        if X(i,j) > val
           bX(i,j) = 255
        else if X(i,j) < val
```

```
            bX(i,j) = 0
        end
    until N
until M
```

Restrictions/Notes:

This function requires a grayscale input image of type uint8 and a threshold value.

The following functions are calling:

main.m and ans2a_anisotropic.m

4 Results

4.1 "disk.gif"

In this subsection, we are going to look at the input image for the non-linear filtering processes, the pixel distribution and also the mean and standard deviation of the Region of Interest in the original image.

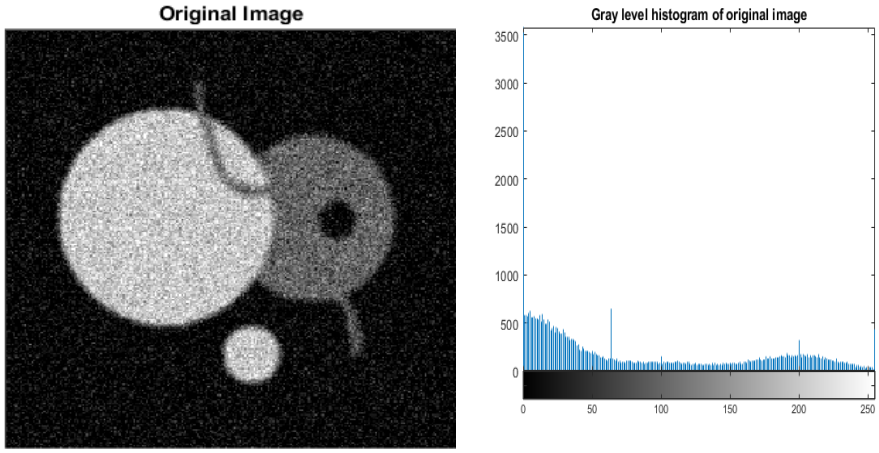


Figure 3: (a) Original Image (b) Pixel Distribution of Original Image

Figure 3 (a) is the Original Image that is taken as input to each of the non-linear filtering techniques. Figure

3 (b) shows the pixel distribution of the input image. It looks like it is evenly distributed with a couple peak values for approx. grayscale values of 60 and 200. The pixel intensity is measured on a scale between 0-255. (uint8).

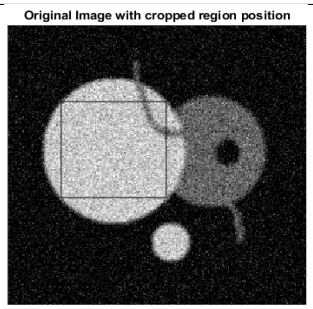
ROI	Mean	Standard Deviation
	197.3497	31.4776

Table 1: Table containing mean and standard deviation for an ROI in the original image

Table 1 shows the original image overlayed with the Region of Interest (ROI) for which the mean and standard deviation are calculated. The ROI is the region covered by the rectangle inside the larger disk region.

4.2 Results for Median Filter

In this subsection, we will be showing the results of the Median filter on the "disk.gif" image.

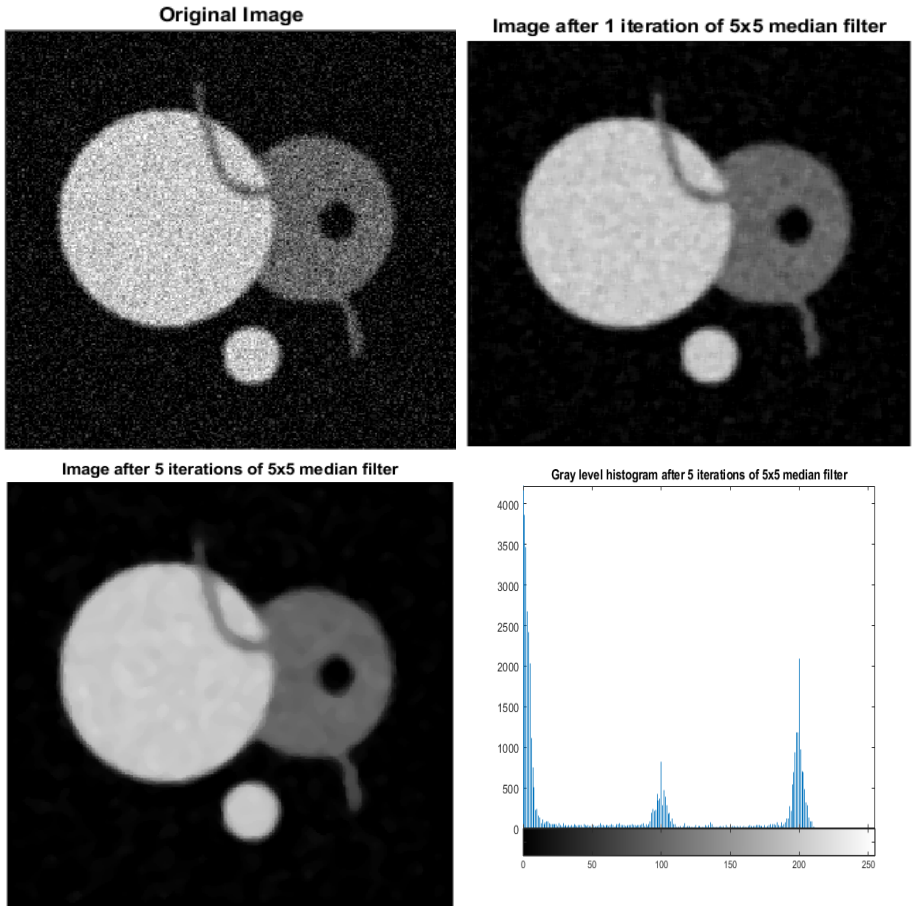


Figure 4: (a) Original Image (Top-Left) (b) Median Filter applied 1 iteration (Top-Right) (c) Median Filter applied 5 iterations (Bottom-Left) (d) Pixel Distribution of (c)

Figure 4 (c) shows that the salt and pepper noise is removed after five iterations of the median filter. We can also observe that the edges are preserved, but it looks like the edge region of the large, bright disk that passes through the dark disk seems to be a little smudged. Also, a disadvantage is the computational cost due to the sorting process. The histogram shows the number of pixels in the y-axis and the grayscale pixel value in the x-axis.

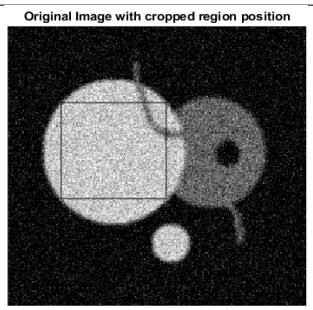
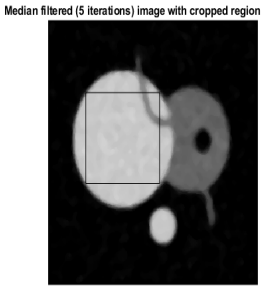
ROI	Mean	Standard Deviation
	197.3497	31.4776
	197.5819	11.7433

Table 2: Table containing the mean and standard deviation for an ROI in the image filtered for 5 iterations using Median Filter

Table 2 shows us that the mean pixel value within the ROI (the region bounded by the rectangle within the large bright disk) slightly increases from the original image to the median filtered image and the standard deviation is drastically reduced. This shows that, after filtering, the pixels in the region do not vary a lot from the mean. This is because of the noise reduction.

4.3 Results for Alpha-Trimmed Mean Filter

In this subsection, we will be showing the results of the Alpha-Trimmed Mean filter on the "disk.gif" image.

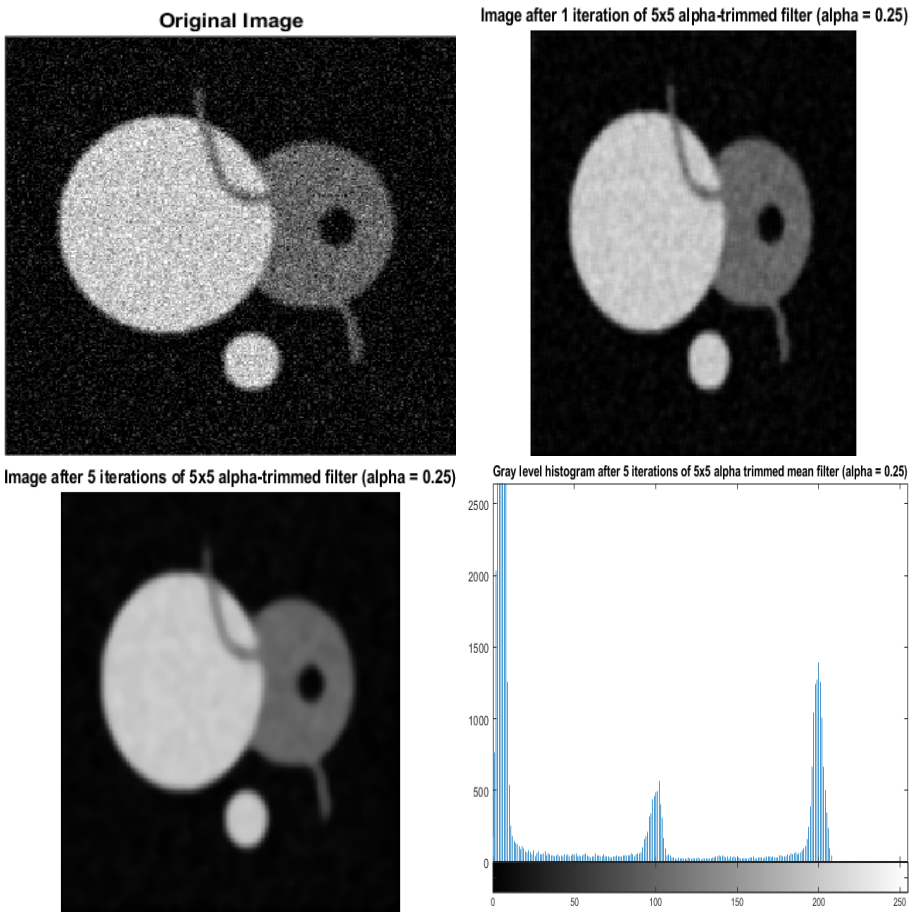


Figure 5: (a) Original Image (Top-Left) (b) Alpha-Trimmed Mean Filter applied 1 iteration (Top-Right) (c) Alpha-Trimmed Mean Filter applied 5 iterations (Bottom-Left) (d) Pixel Distribution of (c)

Figure 5 (c) shows that the salt and pepper noise is removed after five iterations of applying the alpha-trimmed mean filter. We can also observe that the edges are preserved and edge regions are regular (not smudged like in median filter). Similar to the median filter, a disadvantage is the computational cost due to the sorting process. Also, alpha-trimmed mean becomes a median filter when $\alpha = 0.5$ and a mean filter when $\alpha = 0$. The

histogram shows the number of pixels in the y-axis and the grayscale pixel value in the x-axis.

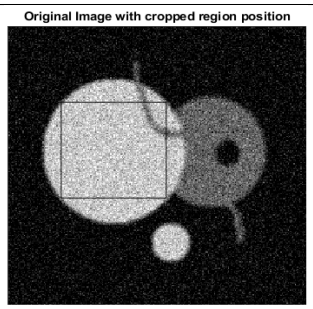
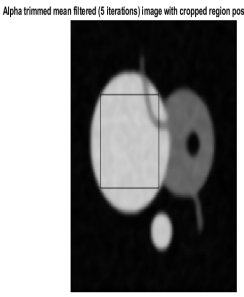
ROI	Mean	Standard Deviation
	197.3497	31.4776
	197.4843	11.2833

Table 3: Table containing the mean and standard deviation for an ROI in the image filtered for 5 iterations using Alpha Trimmed Mean Filter

Table 3 shows us that the mean pixel value within the ROI (the region bounded by the rectangle within the large bright disk) slightly increases from the original image to the filtered image and the standard deviation is drastically reduced. This shows that, after filtering, the pixels in the region do not vary a lot from the mean. This is because of the noise reduction.

4.4 Results for Sigma Filter

In this subsection, we will be showing the results of the Sigma filter on the "disk.gif" image.

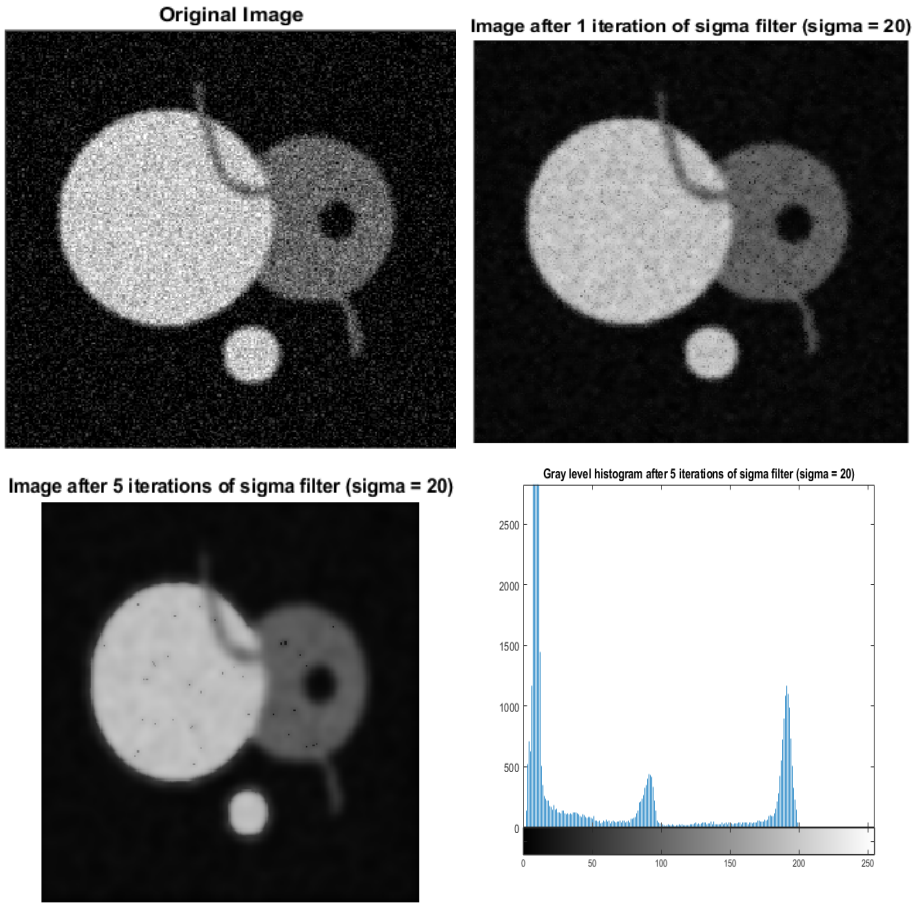


Figure 6: (a) Original Image (Top-Left) (b) Sigma Filter applied 1 iteration (Top-Right) (c) Sigma Filter applied 5 iterations (Bottom-Left) (d) Pixel Distribution of (c)

Figure 6 (C) shows that the filtering process is affected by the presence of salt and pepper noise. Even after filtering, pepper noise remains on the disk regions. Other than that, the sigma filter preserves edges, edge junctions

and thin lines. Also, the edges are blurred and a bit smudged at the edge junctions of the disks. Also, both the tails of the dark disk seemed to be blurred and faded. The histogram shows the number of pixels in the y-axis and the grayscale pixel value in the x-axis.

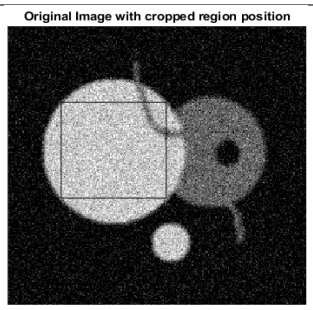
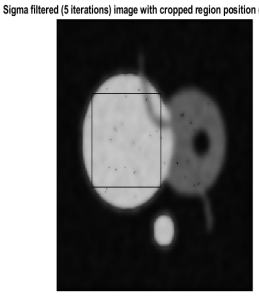
ROI	Mean	Standard Deviation
	197.3497	31.4776
	187.2514	14.3703

Table 4: Table containing the mean and standard deviation for an ROI in the image filtered for 5 iterations using Sigma Filter

Table 4 shows that the mean pixel value within the ROI (the region bounded by the rectangle within the large bright disk) decreases by 10 from the original image to the filtered image and the standard deviation is drastically reduced. This shows that, after filtering, the pixles in the ROI are mostly consistent except for the few pepper noise pixels which brings the mean down.

4.5 Results for Symmetric Nearest Neighbor Mean Filter

In this subsection, we will be showing the results of the Symmetric Nearest Neighbor Mean filter on the "disk.gif" image.

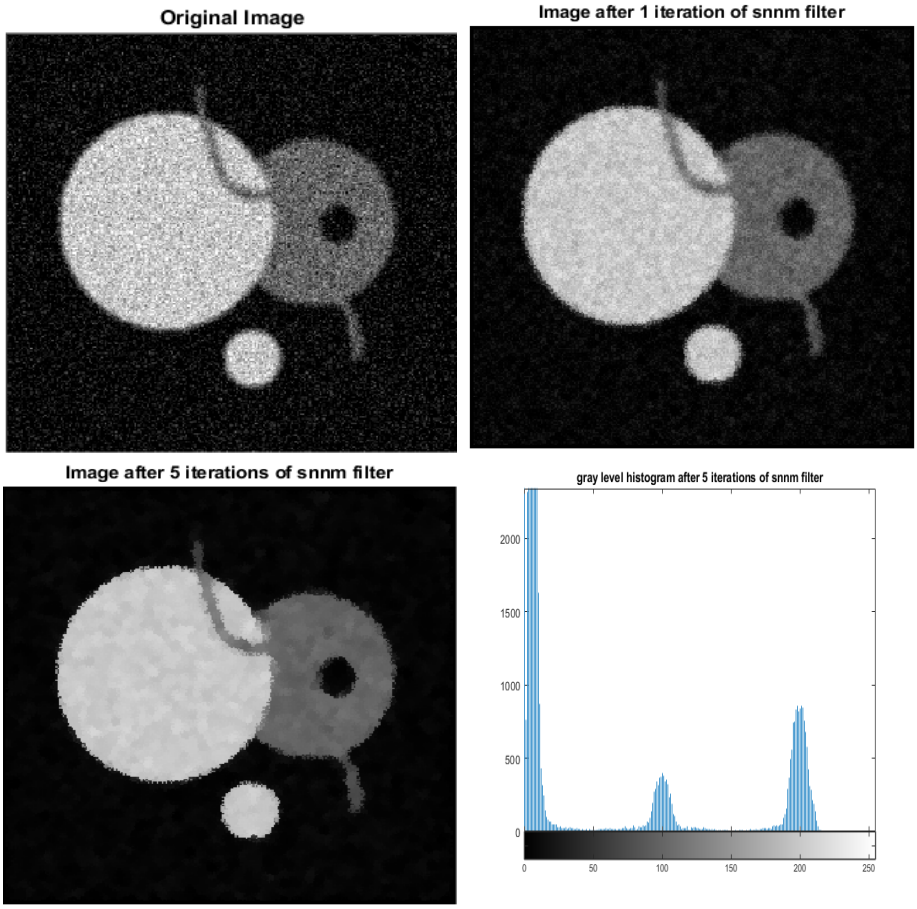


Figure 7: (a) Original Image (Top-Left) (b) Symmetric Nearest Neighbor Mean Filter applied 1 iteration (Top-Right) (c) Symmetric Nearest Neighbor Mean Filter applied 5 iterations (Bottom-Left) (d) Pixel Distribution of (c)

Figure 7 (c) shows that the noise is reduced effectively and the edges of the disks are sharpened instead of being

blurred like in the other 3 filters. The edges appear to be serrated. The histogram shows the number of pixels in the y-axis and the grayscale pixel value in the x-axis.

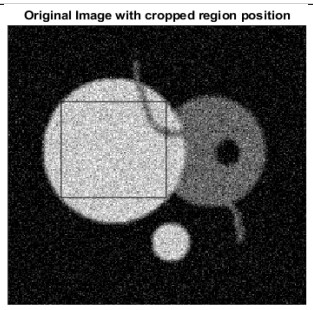
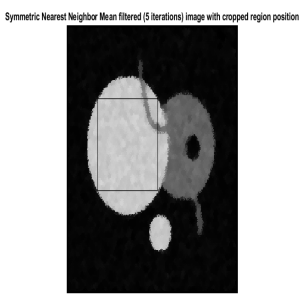
ROI	Mean	Standard Deviation
	197.3497	31.4776
	197.8245	13.6322

Table 5: Table containing the mean and standard deviation for an ROI in the image filtered for 5 iterations using Symmetric Nearest Neighbor Mean Filter

Table 5 shows that the mean of the ROI (the region bounded by the rectangle within the large bright disk) is increased by a small margin and standard deviation is decreased drastically. This means that the pixels are more consistent than the original image due to noise reduction.

4.6 Overall filtered image comparisons after 5 iterations

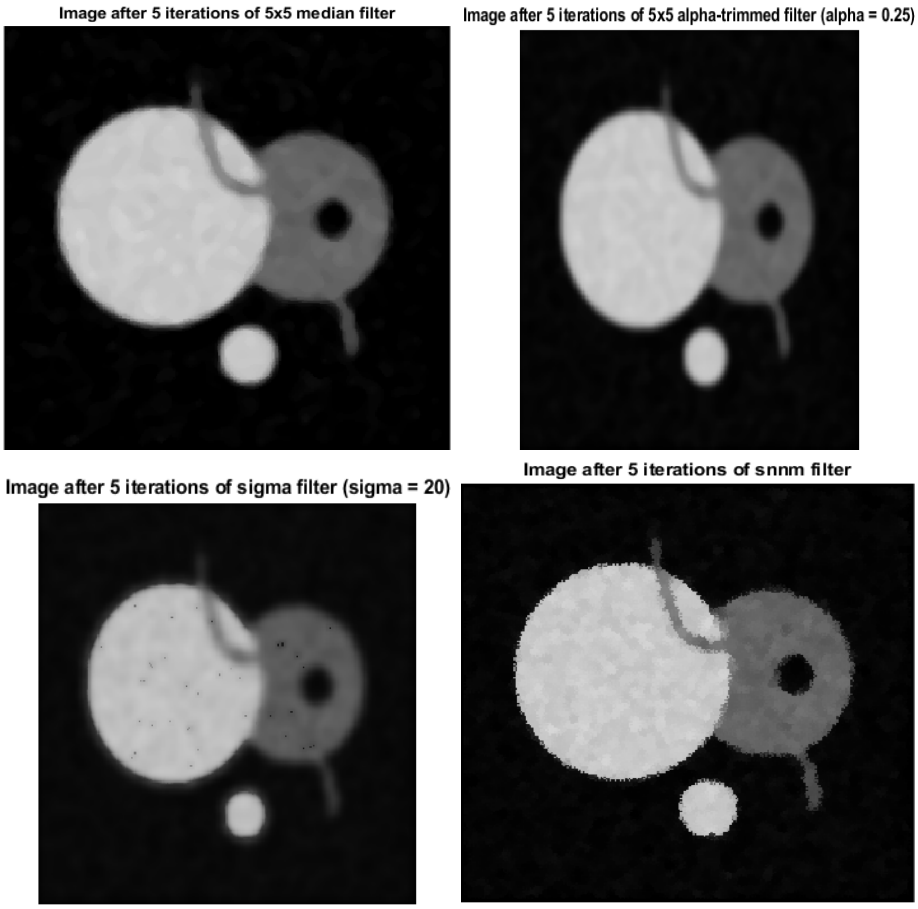


Figure 8: Filtered results after 5 iterations using (a) Median Filter (b) Alpha-Trimmed Mean Filter (c) Sigma Filter (d) Symmetric Nearest Neighbor Mean

Based on Figure 8, we can see that the median, alpha-trimmed mean and sigma filters blur/smooth the edges of the disks whereas the symmetric mean filter sharpens the edges of the disks. Among the filters producing smooth edges, the alpha trimmed mean with $\alpha = 0.25$ produces edge regions which look regular. But, the sigma filter

and the median filter seem to have a cloud-like finish near the edge regions of the disks.

4.7 Overall Mean and Standard Deviation comparison

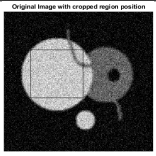
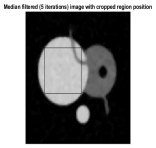
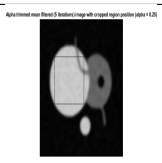
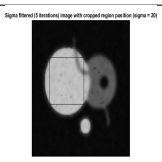
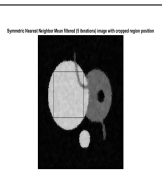
ROI	Name	Mean	Standard Deviation
	Original Image	197.3497	31.4776
	Median filtered(5 iterations)	197.5819	11.7433
	Alpha Trimmed Mean filtered(5 iterations)	197.4843	11.2833
	Sigma filtered(5 iterations)	187.2514	14.3703
	Symmetric Nearest Neighbor Mean filtered(5 iterations)	197.8245	13.6322

Table 6: Comparison of mean and standard deviation for an ROI in the image filtered for 5 iterations using the 4 non-linear filters

Based on table 6, overall, it looks like the median filter and the alpha-trimmed mean filter do a good job of removing noise since they have low standard deviation values. In terms of feature enhancement (like edges),

the symmetric nearest neighbor mean filter does a good job since it has the highest mean.

4.8 Results of Anisotropic Diffusion in Questions 2a and 2b

The iterative non-linear filtering method of Anisotropic Diffusion has been executed on the cwheelnoise and the lake image for exponential g(.) and inverse quadratic g(.). The results of Anisotropic Diffusion have been obtained for 5, 20 and 100 iterations for $\lambda = 0.25$. The histograms, 2D plots of pixel values along X axis for y=128 and the segmented gray spokes of the wheel have obtained for the cwheelnoise. The value of k has been selected as 25, for the experiments in Question 2a and Question 2b. It has helped to verify the correctness of the experiments by comparing the images with images of cwheelnoise after anisotropic diffusion in L14-19 for $k = 25$.

4.8.1 Input grayscale image cwheelnoise and its segmented version

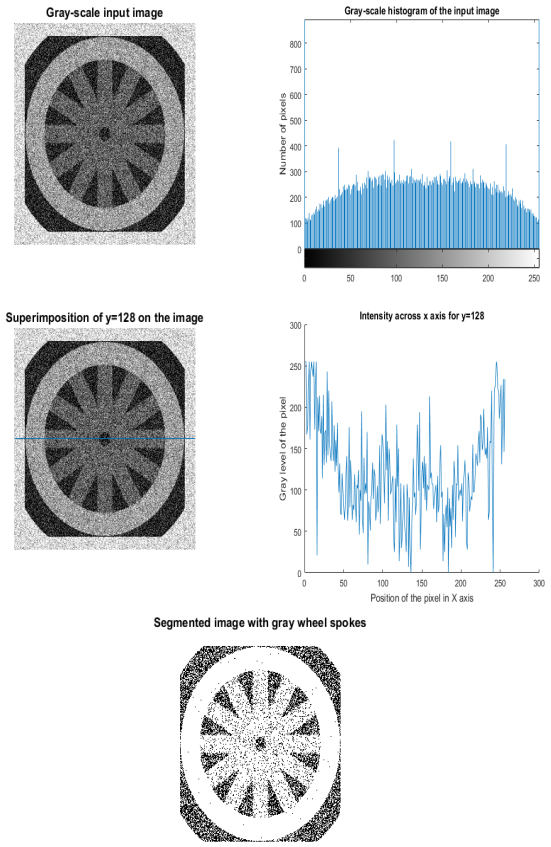


Figure 9: (a) Grayscale Input Image cwheelnoise (Top-Left) (b) Histogram of the grayscale input image (Top-Right) (c) Superimposition of y=128 plot on the grayscale input image (Centre-Left) (b) Plot of the Intensity of pixels in the image along y=128 (Centre-Right)(e) Segmented version of the grayscale input image with spokes of the wheel(Bottom)

The histogram in Figure 9 (b) of the input grayscale image in Figure 9(a) is extremely noisy and the frequency distribution of the pixels are similar. So, the background and foreground of the image cannot be identified clearly.

It is manifested in the segmented image in Figure 9(e) obtained by manual thresholding. The spokes can be obtained but the image is very noise. A threshold value of 50 has been selected for thresholding based on the histogram where there is a sharp peak soon after the pixel value of 50, which could be due to the difference of pixel values outside the spokes of the wheel. y=128 has been superimposed on the input image in Figure 10(c) in order to generate the 2D plot of pixel values along the X axis for y=128. The 2D plot is also extremely noisy, although it follows a decreasing nature.

4.8.2 Grayscale image after 5 iterations of anisotropic diffusion on input image cwheelnoise with exponential g(.) and its segmented version

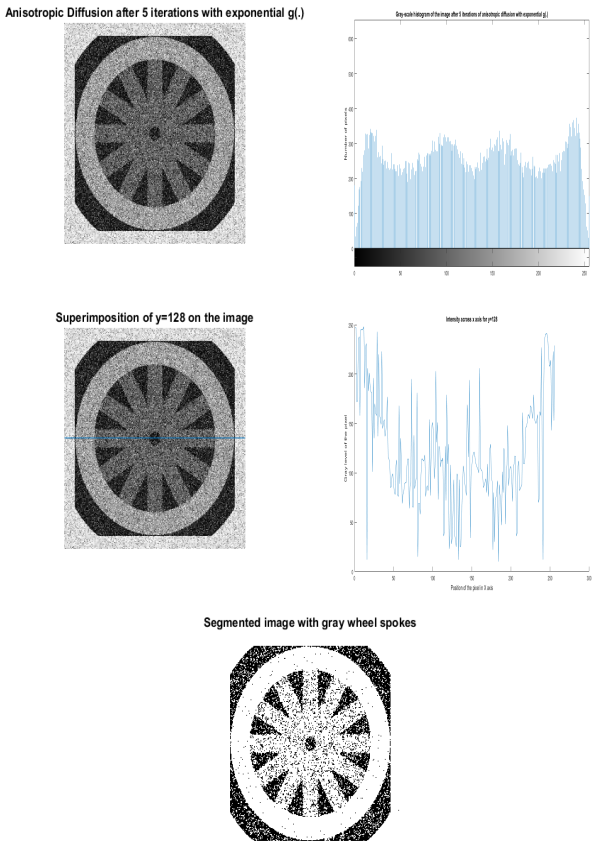


Figure 10: (a) Grayscale image after 5 iterations of anisotropic diffusion on input image cwheelnoise with exponential g(.)(Top-Left) (b) Histogram of the grayscale image in (a) (Top-Right) (c) Superimposition of y=128 plot on the grayscale image in (a) (Centre-Left) (b) Plot of the Intensity of pixels in the image in (a) along y=128 (Centre-Right)(e) Segmented version of the grayscale image in (a) with spokes of the wheel(Bottom)

5 iterations of anisotropic diffusion with exponential g(.) leads to 4 observable peaks in the histogram in Figure 10 (b) of the input grayscale image in Figure 10(a). However, the image is still noisy although it is smoother than the input image. The spokes of the wheel are segmented in Figure 10(e) by manual thresholding with a threshold value of 64. This threshold value has been selected as there is a local minima in the image at pixel value near 64, which could be due to the change of the background and the foreground that are separated by the spokes of the image. y=128 has been superimposed on the input image in Figure 10(c) in order to generate the 2D plot of pixel values along the X axis for y=128. The 2D plot in Figure 10 (d) is smoother than that in Figure 9(d) and is decreasing. But it is still noisy.

4.8.3 Grayscale image after 20 iterations of anisotropic diffusion on input image cwheelnoise with exponential g(.) and its segmented version

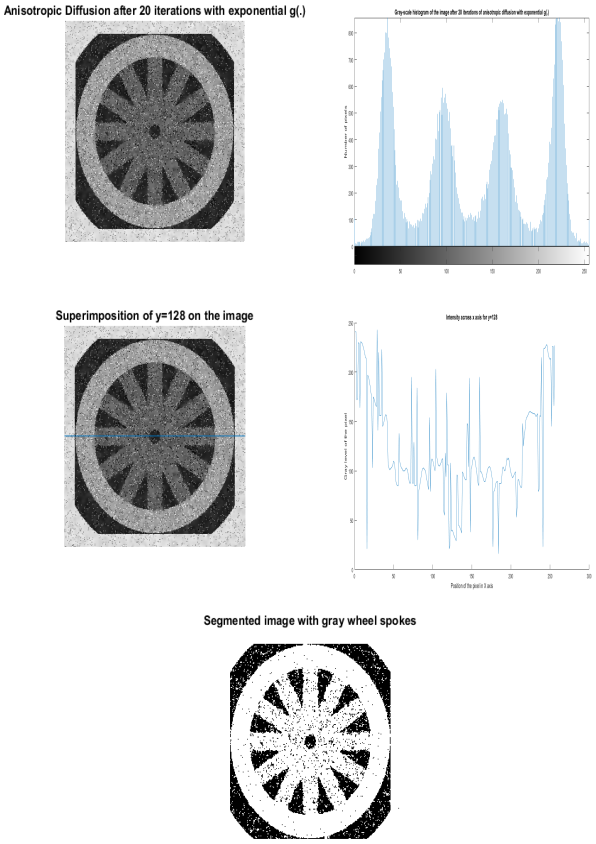


Figure 11: (a) Grayscale image after 20 iterations of anisotropic diffusion on input image cwheelnoise with exponential g(.)(Top-Left) (b) Histogram of the grayscale image in (a) (Top-Right) (c) Superimposition of y=128 plot on the grayscale image in (a) (Centre-Left) (b) Plot of the Intensity of pixels in the image in (a) along y=128 (Centre-Right)(e) Segmented version of the grayscale image in (a) with spokes of the wheel(Bottom)

20 iterations of anisotropic diffusion with exponential g(.) leads to 4 observable peaks in the histogram in Figure 11 (b) of the input grayscale image in Figure 11(a). The peaks are much sharper than that in Figure 10(b). The noise in the image has decreased to a large extent. The spokes of the wheel are segmented in Figure 11(e) by manual thresholding with a threshold value of 64. This threshold value has been selected as there is a local minima in the image at pixel value near 64, which could be due to the change of the background and the foreground that are separated by the spokes of the image. y=128 has been superimposed on the input image in Figure 11(c) in order to generate the 2D plot of pixel values along the X axis for y=128. The 2D plot in Figure 11 (d) is smoother than that in Figure 10(d) and is decreasing. But along y=128, there is a huge difference in pixel values as is observed in the plot.

4.8.4 Grayscale image after 100 iterations of anisotropic diffusion on input image cwheelnoise with exponential g(.) and its segmented version

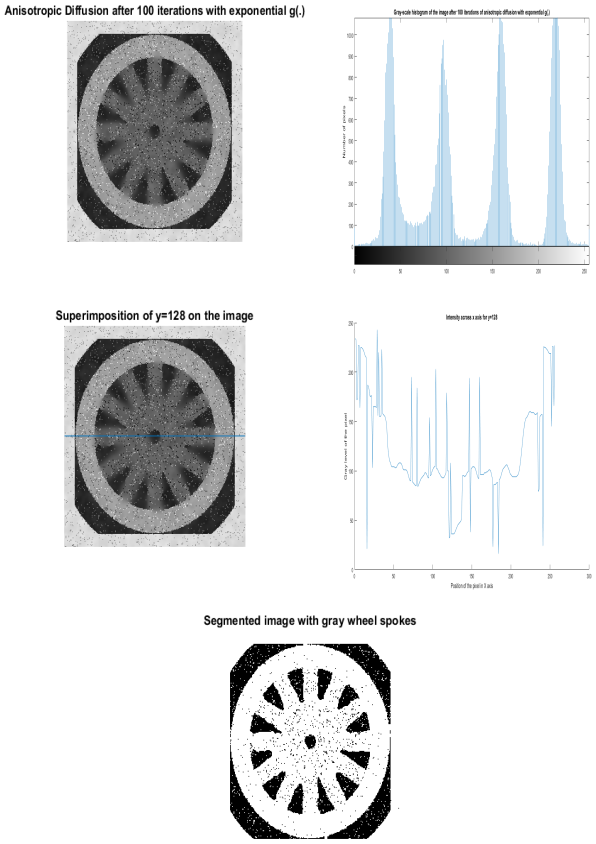


Figure 12: (a) Grayscale image after 100 iterations of anisotropic diffusion on input image cwheelnoise with exponential g(.)(Top-Left) (b) Histogram of the grayscale image in (a) (Top-Right) (c) Superimposition of y=128 plot on the grayscale image in (a) (Centre-Left) (b) Plot of the Intensity of pixels in the image in (a) along y=128 (Centre-Right)(e) Segmented version of the grayscale image in (a) with spokes of the wheel(Bottom)

100 iterations of anisotropic diffusion with exponential g(.) leads to 4 observable peaks in the histogram in Figure 12 (b) of the input grayscale image in Figure 12(a) which are the sharpest among the obtained results. The noise in the image has decreased even further than that for iterations 5 and 20. The spokes of the wheel are segmented in Figure 12(e) by manual thresholding with a threshold value of 64. This threshold value has been selected as there is a local minima in the image at pixel value near 64, which could be due to the change of the background and the foreground that are separated by the spokes of the image. However the loss in information content can be observed clearly as the spokes of the wheel appear to be corroded. y=128 has been superimposed on the input image in Figure 12(c) in order to generate the 2D plot of pixel values along the X axis for y=128. The 2D plot in Figure 12 (d) is smoother than that in Figure 11(d) and is decreasing. But along y=128, there are still some noisy pixels.

4.8.5 Grayscale image after 5 iterations of anisotropic diffusion on input image cwheelnoise with inverse quadratic g(.) and its segmented version

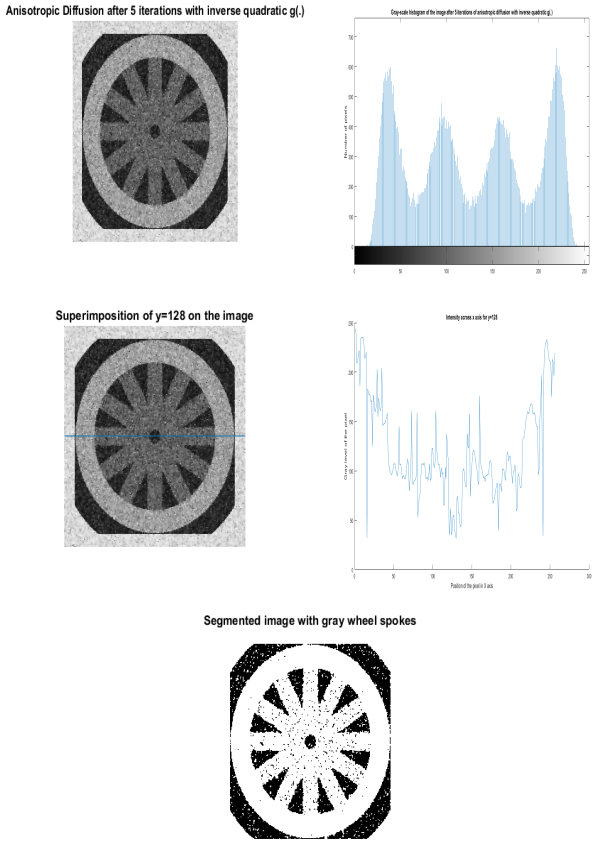


Figure 13: (a) Grayscale image after 5 iterations of anisotropic diffusion on input image cwheelnoise with inverse quadratic g(.)(Top-Left) (b) Histogram of the grayscale image in (a) (Top-Right) (c) Superimposition of y=128 plot on the grayscale image in (a) (Centre-Left) (b) Plot of the Intensity of pixels in the image in (a) along y=128 (Centre-Right)(e) Segmented version of the grayscale image in (a) with spokes of the wheel(Bottom)

5 iterations of anisotropic diffusion with inverse quadratic g(.) leads to 4 observable peaks in the histogram in Figure 13 (b) of the input grayscale image in Figure 13(a). The peaks are sharper than those for exponential g(.) with the same number of iterations. However, the image is noisy although it is smoother than the input image. The spokes of the wheel are segmented in Figure 13(e) by manual thresholding with a threshold value of 64. This threshold value has been selected as there is a local minima in the image at pixel value near 64, which could be due to the change of the background and the foreground that are separated by the spokes of the image. y=128 has been superimposed on the input image in Figure 13(c) in order to generate the 2D plot of pixel values along the X axis for y=128. The 2D plot in Figure 13 (d) is smoother than that for exponential g(.) and is decreasing. But it is still noisy. The wide regions in the spokes can be observed clearly in Figure 13(a).

4.8.6 Grayscale image after 20 iterations of anisotropic diffusion on input image cwheelnoise with inverse quadratic g(.) and its segmented version

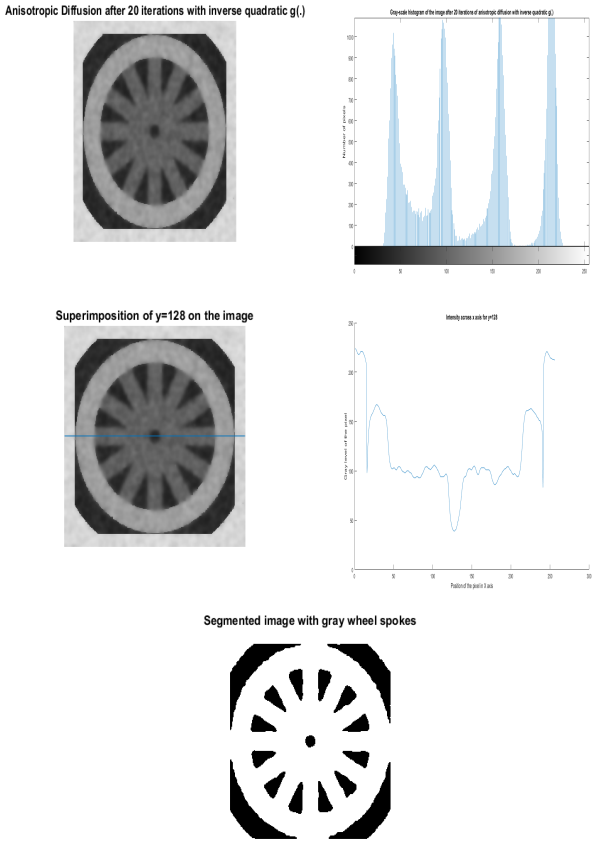


Figure 14: (a) Grayscale image after 20 iterations of anisotropic diffusion on input image cwheelnoise with inverse quadratic g(.)(Top-Left) (b) Histogram of the grayscale image in (a) (Top-Right) (c) Superimposition of y=128 plot on the grayscale image in (a) (Centre-Left) (b) Plot of the Intensity of pixels in the image in (a) along y=128 (Centre-Right)(e) Segmented version of the grayscale image in (a) with spokes of the wheel(Bottom)

20 iterations of anisotropic diffusion with inverse quadratic g(.) leads to 4 observable peaks in the histogram in Figure 14 (b) of the input grayscale image in Figure 14(a). The peaks are sharper than in the histogram in Figure 13(b). The noise has been almost removed in the image. The wide regions can be observed clearly. The spokes of the wheel are segmented in Figure 14(e) by manual thresholding with a threshold value of 64. This threshold value has been selected as there is a local minima in the image at pixel value near 64, which could be due to the change of the background and the foreground that are separated by the spokes of the image. y=128 has been superimposed on the input image in Figure 14(c) in order to generate the 2D plot of pixel values along the X axis for y=128. The 2D plot in Figure 14 (d) is smoother than that that in Figure 13(d) and is decreasing in nature.

4.8.7 Grayscale image after 100 iterations of anisotropic diffusion on input image cwheelnoise with inverse quadratic g(.) and its segmented version

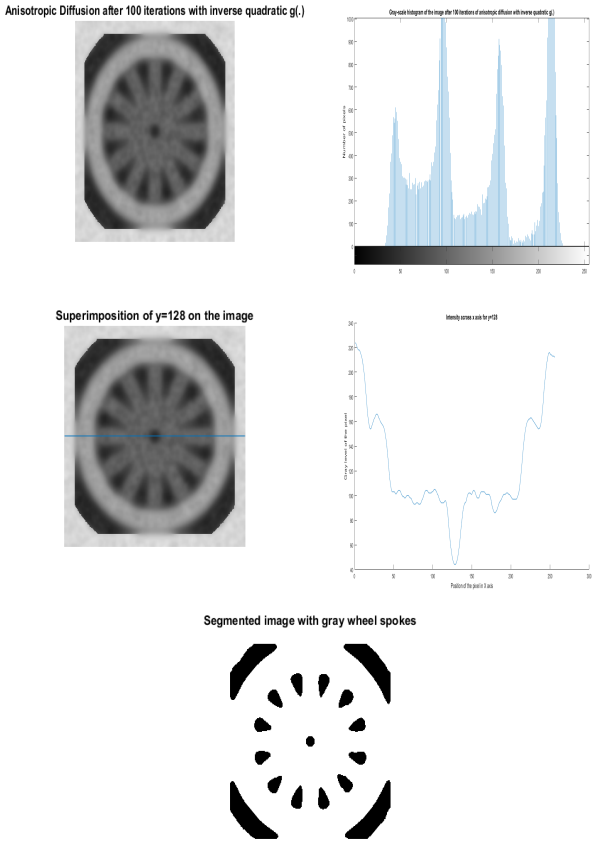


Figure 15: (a) Grayscale image after 100 iterations of anisotropic diffusion on input image cwheelnoise with inverse quadratic g(.)(Top-Left) (b) Histogram of the grayscale image in (a) (Top-Right) (c) Superimposition of y=128 plot on the grayscale image in (a) (Centre-Left) (b) Plot of the Intensity of pixels in the image in (a) along y=128 (Centre-Right)(e) Segmented version of the grayscale image in (a) with spokes of the wheel(Bottom)

100 iterations of anisotropic diffusion with inverse quadratic g(.) leads to 4 observable peaks in the histogram in Figure 15 (b) of the input grayscale image in Figure 15(a). The peaks are not as sharp as the peaks of the histogram in Figure 14(b) which could be due to the inverse quadratic nature of the image. The noise has been almost removed in the image. However, information content is getting lost from the image. The wide regions can be observed but not as clearly as for 20 iterations due to loss of information. The spokes of the wheel are segmented in Figure 15(e) by manual thresholding with a threshold value of 64. This threshold value has been selected as there is a local minima in the image at pixel value near 64, which could be due to the change of the background and the foreground that are separated by the spokes of the image. y=128 has been superimposed on the input image in Figure 15(c) in order to generate the 2D plot of pixel values along the X axis for y=128. The 2D plot in Figure 15 (d) is smoother than that that in Figure 14(d) and is decreasing in nature.

4.8.8 Input grayscale image lake

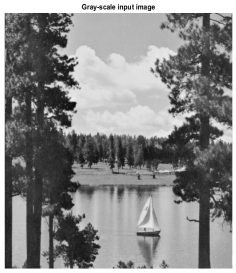


Figure 16: Grayscale input image lake

Figure 16 represents the grayscale input image of lake. This image is less noisy than the cwheelnoise image in Figure 9(a).

4.8.9 Grayscale image after 5 iterations of anisotropic diffusion on input image lake with exponential g(.) and inverse quadratic g(.)

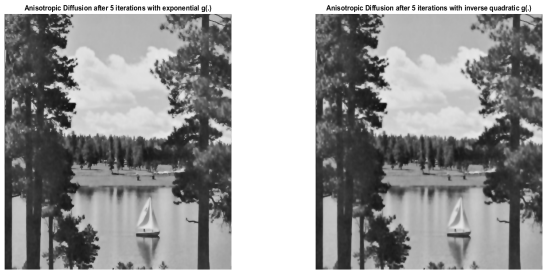


Figure 17: (a) Grayscale image after 5 iterations of anisotropic diffusion on input image lake with exponential g(.) (Left) (b) Grayscale image after 5 iterations of anisotropic diffusion on input image lake with inverse quadratic g(.) (Right)

The high contrast features are observed in Figure 17(a) with exponential g(.) after 5 iterations while the broad features can be observed in Figure 17(b) with inverse

quadratic g(.). However, these results can be improved with further iterations.

4.8.10 Grayscale image after 20 iterations of anisotropic diffusion on input image lake with exponential g(.) and inverse quadratic g(.)

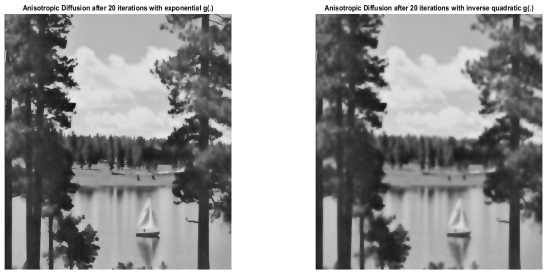


Figure 18: (a) Grayscale image after 20 iterations of anisotropic diffusion on input image lake with exponential g(.) (Left) (b) Grayscale image after 20 iterations of anisotropic diffusion on input image lake with inverse quadratic g(.) (Right)

The high contrast features are observed in Figure 18(a) with exponential g(.) after 20 iterations while the broad features can be observed in Figure 18(b) with inverse quadratic g(.). The quality of these images are better than those after 5 iterations in Figures 17(a) and 17(b).

4.8.11 Grayscale image after 100 iterations of anisotropic diffusion on input image lake with exponential g(.) and inverse quadratic g(.)

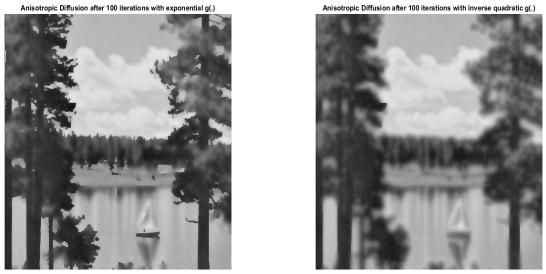


Figure 19: (a) Grayscale image after 100 iterations of anisotropic diffusion on input image lake with exponential g(.) (Left) (b) Grayscale image after 100 iterations of anisotropic diffusion on input image lake with inverse quadratic g(.) (Right)

The high contrast features are observed clearly in Figure 19(a) with exponential g(.) after 100 iterations while the broad features can be observed in Figure 19(b) with inverse quadratic g(.). The quality of these images are best among all the iterations. However, some information content of the pixels appear to have been lost from the image.

4.9 Analysis of Anisotropic Diffusion in Questions 2c

The result of changing the number of iterations and the k values for the exponential g(.) and the inverse quadratic g(.) have been discussed in Sections 4.9.1,4.9.2,4.9.3 and 4.9.4 respectively. The results of Anisotropic Diffusion on the lake image versus the cwheelnoise image has been compared in Section 4.9.5.

4.9.1 Impact of changing the Number of Iterations for exponential g(.) and inverse quadratic g(.) in Question 2c(i) and 2c(ii)

If the g(.) function is exponential, then the peaks in the histogram become sharper as the number of iterations in anisotropic diffusion increase from 5 in Figure 10 (b) to 20 in Figure 11 (b) and 100 in Figure 10 (b). Thus, it is understood that the high contrast edges are favored with increasing number of iterations.
The quality of the image is best for the image after 20 iterations in Figure 11 (a) in comparison to 5 iterations in Figure 10 (a) and 100 iterations in Figure 12 (a). This indicates that the optimal number of iterations to ensure the best quality of the image is somewhere between 20 and 100.
For 100 iterations, some information loss is observed as the segmented gray spokes of the wheel in Figure 12 (e) appears to be a bit eroded in comparison to the segmented spokes for 20 iterations in Figure 11 (e) and that for 5 iterations in Figure 10 (e).
The 2D plots of pixel values along the X axis for a fixed y=128 in Figure 10 (d), 11 (d) and 12 (d) indicate the image behavior and the g(.) function. It can be observed that the general trend of the plots is to decrease along the X axis. However, there are occasional peaks and crests which indicate that the image is noisy. The 2D plots are getting smoother with increasing number of iterations.

If the g(.) function is inverse quadratic, then the peaks in the histogram sharpens with increasing number of iterations of 5, 20 and 100 in Figure 13(b), 14(b) and 15

(b) respectively. However, the rate of increase of sharpness in the peaks of the histogram are not that much in comparison to corresponding histograms with exponential g(.), although the peaks of the histogram for 5 iterations in Figure 13 (b) is more sharp than that for 5 iterations with exponential g(.). Thus, it can be concluded, that wide regions in the image are favored more by inverse quadratic g(.).

This can be corroborated by observing the quality of images in Figure 13 (a), Figure 14 (a) and Figure 15 (a) as the number of iterations increase from 5,20 to 100.

For 5 iterations with inverse quadratic g(.), the segmented spokes of the wheel is a bit noisy in Figure 13(e) but the noise decreases in comparison to the same number of iterations for exponential g(.) in Figure 10 (e). As the number of iterations increase to 20 and 100, the noise is decreasing drastically in the images in Figures 14 (e) and 15 (e) in comparison to Figure 10 (e). However, some information is getting lost as the spokes appear to have corroded in Figures 14(e) and 15 (e). Thus the optimal number of iterations to obtain the segment of the gray spokes clearly should be somewhere between 5 and 20 iterations.

The 2D plots of pixel values along the X axis for a fixed y=128 get smoother from Figure 13 (d) to Figures 14 (d) and 15 (d) as the number of iterations increases. However, the 2D plots are for each iteration are more smooth than corresponding plots with exponential g(.). This could be because the rate of change of the image gradient is higher for exponential g(.) than inverse quadratic

g(.) although both functions have the same decreasing nature.

4.9.2 Impact of changing the k value for exponential g(.) and inverse quadratic g(.) in Question 2c(i) and 2c(ii)

The k value can be considered to be comparable to the norm of the image gradient while the image behavior is computed in g(.). For a fixed $\lambda = 0.25$, the image of cwheelnoise after anisotropic diffusion and the segmented gray spokes of the wheel have been observed for k=25,k=50 and k=12 for both exponential and inverse quadratic g(.). 20 iterations of anisotropic diffusion has been executed in all the cases.

The high contrast features are observed for k=25 and exponential g(.) in Figure 11 (a), while the wide regions in the image are observed with the same k value for exponential g(.). The results for k=50 and k=12 have been presented in Sections 4.9.3 and 4.9.4 respectively. They have been generated to analyze the impact of increasing and decreasing k for a fixed number of iterations and fixed parameter λ.

4.9.3 Grayscale image after 20 iterations of anisotropic diffusion on input image cwheelnoise with exponential g(.) and inverse quadratic g(.) along with its segmented version for k=5o

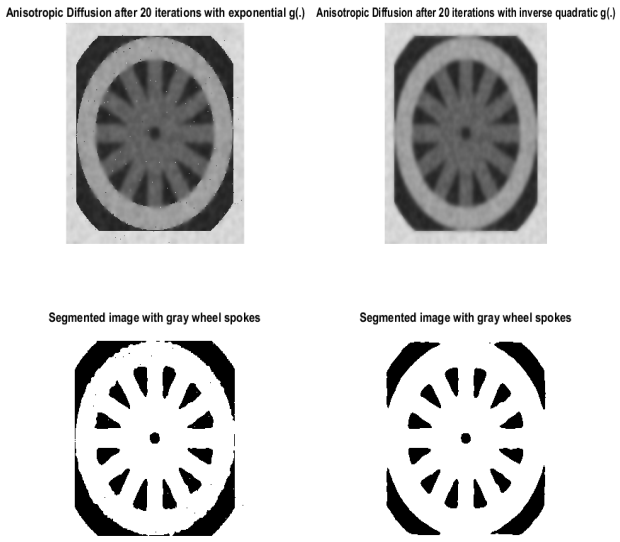


Figure 20: (a) Grayscale image after 20 iterations of anisotropic diffusion on input image cwheelnoise with exponential g(.) with k=50 (Top-Left) (b) Grayscale image after 20 iterations of anisotropic diffusion on input image cwheelnoise with inverse quadratic g(.) with k=50 (Top-Right) (c) Segmented version of the grayscale image in (a) with spokes of the wheel (Bottom-Left) (d) Segmented version of the grayscale image in (b) with spokes of the wheel (Bottom-Right)

When the k value is doubled to 50 for the same number of iterations, then in case of exponential g(.) in Figure 20(a), the noise decreases in comparison to Figure 11 (a) for k=25. The noise decreases in the segmented spokes of the wheel in Figure 20 (c) for k=50 than for k=25 in Figure 11 (e). However, the decrease in noise comes at a cost of information loss as the spokes are a bit corroded for k=50.

For inverse quadratic g(.), however, the image is more lighter for k=50 in Figure 20(b) than k=25 in Figure 14(b). The image quality decreases with increasing g(.) for inverse quadratic g(.). The segmented spokes of the wheel are smoother for k=50 in Figure 20 (d) in comparison to k=25 in Figure 14 (e). However the spokes are again getting corroded for increasing k.

4.9.4 Grayscale image after 20 iterations of anisotropic diffusion on input image cwheelnoise with exponential g(.) and inverse quadratic g(.) along with its segmented version for k=12

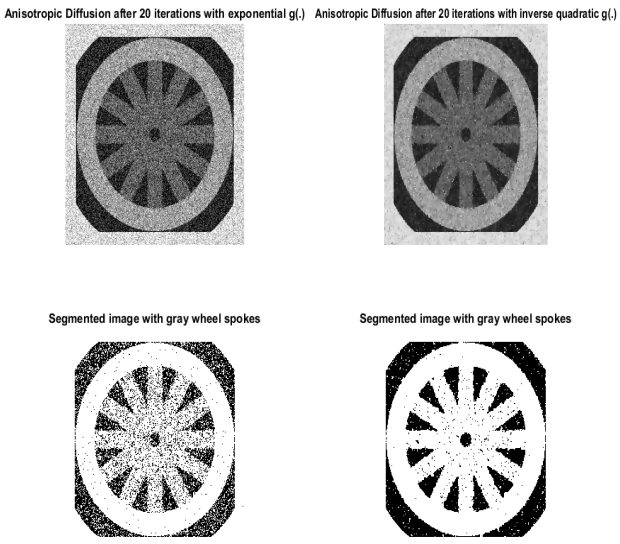


Figure 21: (a) Grayscale image after 20 iterations of anisotropic diffusion on input image cwheelnoise with exponential g(.) with k=12 (Top-Left) (b) Grayscale image after 20 iterations of anisotropic diffusion on input image cwheelnoise with inverse quadratic g(.) with k=12 (Top-Right) (c) Segmented version of the grayscale image in (a) with spokes of the wheel (Bottom-Left) (d) Segmented version of the grayscale image in (b) with spokes of the wheel (Bottom-Right)

When the k value is almost halved to 12 from 25 for the same number of iterations, then in case of exponential g(.) in Figure 21(a), the noise increases in comparison to Figure 11 (a) for k=25. The high contrast features cannot be observed clearly. The noise increases in the segmented spokes of the wheel in Figure 21 (c) for k=12 than for k=25 in Figure 11 (e).

For inverse quadratic g(.), however, the image quality is better for k=12 in Figure 21(b) than k=25 in Figure 14(b). The wide regions can be clearly observed in Figure 21 (b). The segmented spokes of the wheel can be clearly obtained for k=12 in Figure 21(d), although the segmented image with k=12 has more noise in comparison to k=25.

Thus, it can be concluded, that high values of k should be used for exponential g(.) and low values of k should be use for inverse g(.) to obtain better quality of the image. However, the improvement comes at the cost of loosing some information in the image.

4.9.5 Comparison of Anisotropic Diffusion on cwheelnoise image versus lake image in Question 2c(iii)

The high contrast features like the leaves of the trees or the edges of the boat in the lake image can be observed more clearly with exponential g(.) as the number of iterations increase from 5 in Figure 17(a) to 20 in Figure 18 (a) and 100 in Figure 19(a) for k=25 and $\lambda = 0.25$. However for the cwheelnoise image, the images after anisotropic diffusion with the same number of iterations for exponential g(.) focuses on high contrast fea-

tures, but there is a lot of noise in the images. This could be because, the input grayscale image of lake in Figure 16 is less noisy than the input cwheelnoise grayscale image in Figure 9(a). As the rate of change of the gradient in the exponential g(.) is very high, some noise in the input grayscale image can have a large impact on the final image.

The wide regions in the lake image like the sky or the water image can be observed more clearly with inverse quadratic g(.) as the number of iterations increase from 5 in Figure 17(b) to 20 in Figure 18 (b) and 100 in Figure 19(b) for k=25 and $\lambda = 0.25$. In comparison, the images after anisotropic diffusion of the cwheelnoise image with the same number of iterations for inverse quadratic g(.) favors wide regions but is noisy.However, the wide regions are not as distinct as the high contrast features obtained by using exponential g(.). This could be because, the rate of decrease of the gradient with inverse quadratic g(.) is less than that of exponential g(.).

5 Conclusions

The 4 Non-Linear Filters considered for this project do a fine job of removing most of the noise. But, the sigma filter has shown to be impacted by the presence of impulsive noise. Even after filtering, there are occurrences of pepper noise on the filtered image. Also, there is a difference in the behavior of filters to edge regions in the input image. All the 4 filters, except the Symmetric Nearest Neighbor Mean, blur the edge regions of the disk objects in the image, whereas the SNNM filter sharpens the edge regions.

It has been concluded from the experiments of the iterative non-linear filtering method of Anisotropic Diffusion that the high contrast features in the image can be observed if the g(.) function is exponential in nature. The broad features in the image can be observed if the g(.) function is inverse quadratic. The features can be observed clearly with increasing number of iterations at the risk of loosing some information content of the pixels for large number of iterations. The value of k should be large for exponential g(.) and small for inverse quadratic g(.). The different rate of decrease of the gradient of the image for exponential g(.) and inverse quadratic g(.) plays a significant role in noisy images. Further experimentation can be conducted on the parameter λ which is considered as 0.25 in all the experiments in the project.