CSE585/EE555: Digital Image Processing II
Computer Project # 3

Nonlinear Filtering and Anisotropic Diffusion

Mudit Garg, Mayank Murali, Niranjan Thirusangu

Date: 03/27/2020

A. Objectives

The purpose of this project is to understand non-linear filtering and anisotropic diffusion for image filtering. Following have to be implemented -

1. Implement the various non-linear filters and observe their effects on a given image, thus, understanding the pros & cons of these filters.
2. Implement the anisotropic diffusion algorithm and perform the parameter tuning to obtain the optimal results.

B. Methods

The algorithms/ methods used in this project for both parts are described below in this section:

Part A (Non-linear Filtering)

Mean filter

1. In 1-D, mean filter is given by the following equation:

$$y_k = \frac{1}{2N+1}\sum_{i=1}^{2N+1} x_{k-N+i-1}$$

   where, mask length $n = 2N + 1$ (L12-5).
   Basically, the value of k^{th} pixel is replaced by the average of $(2N + 1)$ pixels surrounding it.
2. The real images are two dimensional. The approach stated in point 1 can be converted to 2D.

3. The mask of $(2N+1) \times (2N+1)$ is used. This mask is used as a sliding window over the image.
4. The mean of all the pixels in sliding window is taken and the pixel at $(N+1, N+1)$ location is replaced with this average value.
5. The step 4 is repeated until all the pixels are iterated.
6. In order to deal with boundary pixels, the given image is padded with zeros of *N* rows and *N* columns surrounding it.
7. Our mean filter can be invoked by calling $mean\_filter(\cdot,\cdot)$. The first argument is the image on which mean filter needs to be applied. The second argument is mask size. If the mask size is not given, it is defaulted to 5×5.

Median filter

1. As per L9, suppose there is a 1D image with $2N+1$ pixels:$\{x_{k-N}, \dots, x_k, \dots, x_{k+N}\}$. When a median filter is applied at pixel k, all the pixels are sorted in ascending order, and the middle element is chosen as the value for pixel k. Thus, following equation defines the median filter:
$$y_k = x_{(N+1)}$$
where, $x_{(1)}, x_{(2)}, \dots, x_{(N+1)}, \dots, x_{(2N+1)}$ are the image pixels in ascending order.
2. Our use case was to create a median filter for 2-dimensional image. Thus, we used a mask of size $(2N+1 \times 2N+1)$. Similar to the mean filter, this mask was used a sliding window.
3. We calculated the median of all the pixels present within the sliding window. The pixel at location $(N+1, N+1)$ was replaced with this median value.
4. Step 3 is repeated unless all the pixels are iterated.
5. In order to handle the pixels at boundary of image, image was padded with zeros of *N* rows and *N* columns.
6. Our function for median filter can be invoked using $median\_filter(\cdot,\cdot)$. The first argument is the image on which filter needs to be applied and the second argument is mask size. If the mask size is not given, it is defaulted to [5,5].

Alpha-trimmed mean filter

1. As per L12-4 , alpha-trimmed mean filter is given by :
$$y_k = \left(\frac{1}{n - 2\lfloor \alpha n \rfloor}\right) \sum_{i=\lfloor \alpha n \rfloor + 1}^{n-\lfloor \alpha n \rfloor} x_{(i)}$$
where,
$x_{(i)} = \text{ordered set of input}$, window length is $n = 2N+1$ and 0$\leq \alpha \leq 0.5$

2. The above equation is for 1-D images. Our use-case is of 2D-images.
3. We used a mask of size $(2N + 1 \times\ 2N + 1)$. Similar to the mean filter, this mask was used a sliding window.
4. Equation in step 1 was applied to pixels in the sliding window. Basically, we are calculating the pixel at location $(N + 1, N + 1)$ of sliding widow.
5. In order to deal with boundary pixels, we padded the image with zeros of *N* rows and *N* columns.
6. Our function for alpha-trimmed mean filter can be invoked by using $alphatrimmed(\cdot,\cdot,\cdot)$, where the first argument is image, second argument is alpha value and third argument is mask size. If mask size and alpha are not given, they are defaulted to [5,5] and 0.25 respectively,

Sigma filter

1. Let there are $2N + 1$ inputs: $\{x_{k-N}, \dots, x_k, \dots, x_{k+N}\}$, as per L12-7, the pixel x_k after applying sigma filter to it is given as:
$$\widehat{y_k} = \frac{1}{N_C} \sum_{i=-N}^{N} \delta_i x_{k-i}$$
where,
$$\delta_i = \begin{cases} 1, |x_{k-i} - x_k| \leq 2\sigma \\ 0, \quad\quad otherwise \end{cases}$$
$$N_c \triangleq \text{Number of points } x_{k-i} \text{ having } \delta_i = 1$$
2. The equation in point 1 is for 1D images. The equation for 2D images is analogous to it.
3. The mask for 2D images is of size $(2N + 1) \times (2N + 1)$. This mask is used as a sliding window over the image.
4. The equation mentioned in point 1 is used over the sliding window and the pixel at location $(N + 1, N + 1)$ is replaced by the value obtained using this equation.
5. In order to apply the sigma filter over the boundary pixels, image is padded similar to the way it is done for other filters.
6. Our sigma filter is invoked by calling $sigma\_filter(\cdot,\cdot,\cdot)$. The first argument is an input image, the 2nd one is sigma and the third contains the mask size. If the sigma and mask are not given, they are defaulted to 20 and [5,5] respectively.

Symmetric Nearest Neighbor Mean filter

1. In this filter, we formed the pairs of all the points symmetrically opposite to each other in the given mask.
2. Suppose the pixel on which symmetric nearest neighbor mean filter needs to be applied is $x_{k,l}$.
3. We pick those pixels among the pairs which are most similar to $x_{k,l}$ i.e. select a pixel x that minimizes $|x - x_{k,l}|$.
4. Take the average of all the selected pixels and assign to $x_{k,l}$.
5. Our implementation of this filter can be invoked by calling $snn\_mean\_filter(\cdot,\cdot)$. The first argument is image and the second argument is mask size. If mask size is not given, it is defaulted to [5,5].
6. Again, for the handling of pixels at the boundary, image was padded with zeros of *N* rows and *N* columns, where $n(mask - length) = 2N + 1$.

Calculation of Mean and Standard Deviation of a disk

We are required to calculate the mean and standard deviation of the interior of the largest disk in given image – 'disk.*gif*'. We used the $imtool$ functionality of MATLAB to calculate the pixel locations which form the largest disk. Using those pixel locations, we found the pixel values and calculated the mean and standard deviation of the data.

Part B (Anisotropic Diffusion for Image Filtering)

Anisotropic diffusion is an iterative non-linear filtering method. It was introduced by Perona and Malik. This method is useful in reducing the noise from images while preserving the edges.

Anisotropic Diffusion Algorithm

1. We represent N, S, E, W, as North, South, East, and West neighbors of image I at position (i,j) respectively.

2. The equation 7 in Perona Malik for anisotropic diffusion can be represented as:

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

where $0 \leq \lambda \leq {}^{1}\!/_{4}$ and t represents the iteration.

3. Conduction Coefficients $\{c_N^t, c_S^t, c_E^t, c_W^t\}$ can be defined as:

$$c_N^t(i,j) = g\left(|\nabla_N I_{i,j}^t|\right)$$
$$c_S^t(i,j) = g\left(|\nabla_S I_{i,j}^t|\right)$$
$$c_E^t(i,j) = g\left(|\nabla_E I_{i,j}^t|\right)$$
$$c_W^t(i,j) = g\left(|\nabla_W I_{i,j}^t|\right)$$

4. There are two choices for $g(\bullet)$:
    a. $g(\nabla I) = \exp\{-\left(\left\|\nabla_N I_{i,j}^t\right\|/K\right)^2\}$ for high-contrast edges and so forth for E, W and S.
    b. $g(\nabla I) = \frac{1}{1+\left(\|\nabla_N I_{i,j}^t\|/K\right)^2}$ for wide regions and so forth for E, W, and S.
5. Our function for anisotropic diffusion can be invoked by calling $anisotropic\_diffusion(\cdot,\cdot,\cdot,\cdot)$. Here, the first argument is the image, 2nd one is the value of k, 3rd one is the number of iterations that are needed to be done and the last one is the coefficient type.
6. If the coefficient type is 1, it refers to '$exponential$' type, else if it is 2, the coefficient is '$quadratic$'. λ is hardcoded to 0.25

Flowchart Diagram (Illustration of MATLAB code)

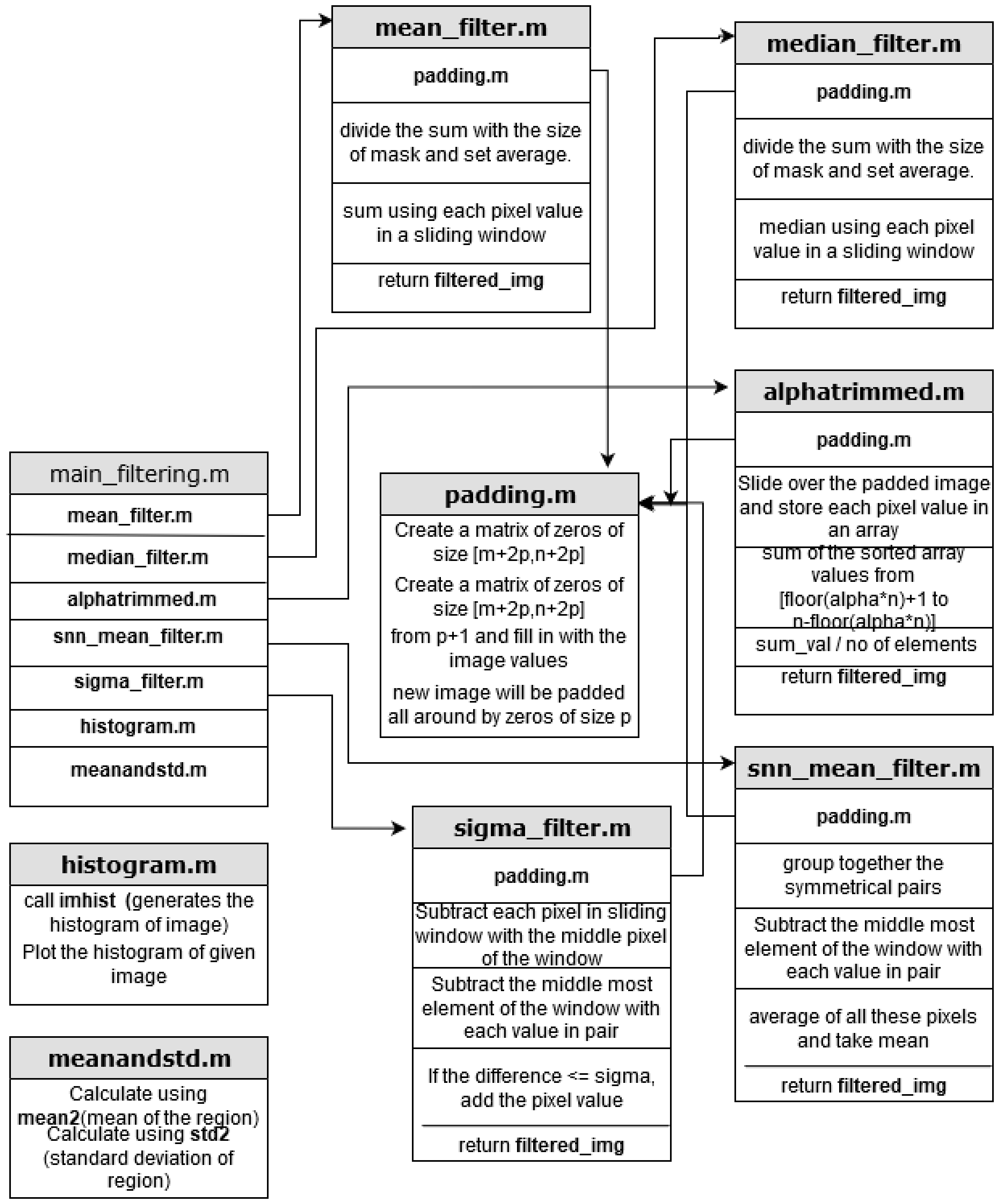
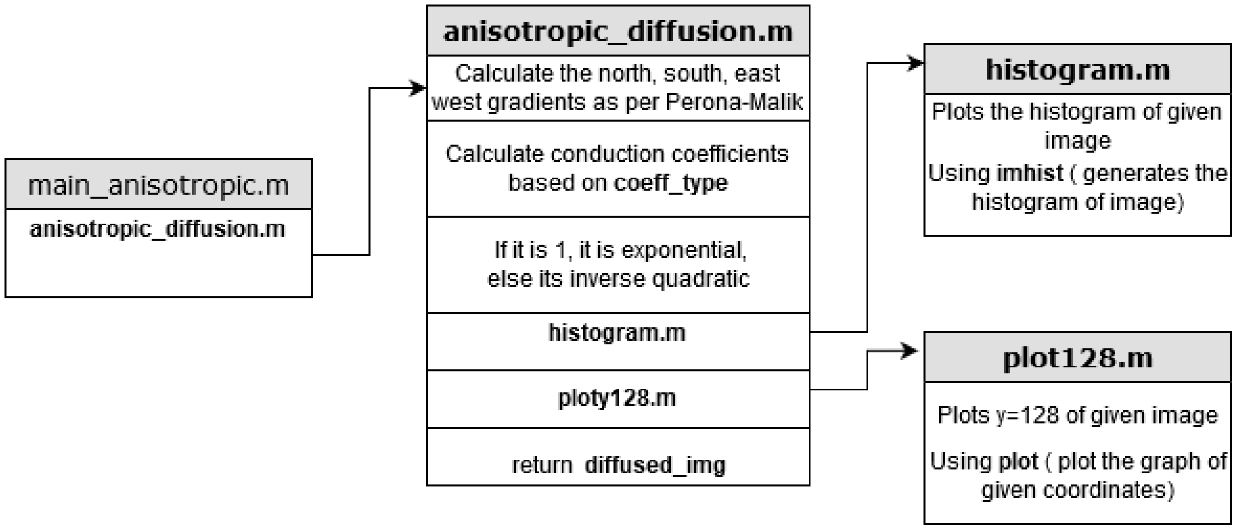

mean_filter.m
padding.m
divide the sum with the size of mask and set average.
sum using each pixel value in a sliding window
return filtered_img
median_filter.m
padding.m
divide the sum with the size of mask and set average.
median using each pixel value in a sliding window
return filtered_img
alphatrimmed.m
padding.m
Slide over the padded image and store each pixel value in an array
sum of the sorted array values from [floor(alpha*n)+1 to n-floor(alpha*n)]
sum_val / no of elements
return filtered_img
main_filtering.m
mean_filter.m
median_filter.m
alphatrimmed.m
snn_mean_filter.m
sigma_filter.m
histogram.m
meanandstd.m
padding.m
Create a matrix of zeros of size [m+2p,n+2p]
Create a matrix of zeros of size [m+2p,n+2p]
from p+1 and fill in with the image values
new image will be padded all around by zeros of size p
snn_mean_filter.m
padding.m
group together the symmetrical pairs
Subtract the middle most element of the window with each value in pair
average of all these pixels and take mean
return filtered_img
histogram.m
call imhist (generates the histogram of image)
Plot the histogram of given image
sigma_filter.m
padding.m
Subtract each pixel in sliding window with the middle pixel of the window
Subtract the middle most element of the window with each value in pair
If the difference <= sigma, add the pixel value
return filtered_img
meanandstd.m
Calculate using mean2(mean of the region)
Calculate using std2 (standard deviation of region)

Figure 0.1: Flow-chart showing the code execution of main_filtering.m. It's the Problem 1 of the project

Figure 0.2: Flow-chart showing code execution of main_anisotropic.m. It's the Problem 2 in this project.


main_anisotropic.m
anisotropic_diffusion.m
anisotropic_diffusion.m
Calculate the north, south, east west gradients as per Perona-Malik
Calculate conduction coefficients based on coeff_type
If it is 1, it is exponential, else its inverse quadratic
histogram.m
ploty128.m
return diffused_img
histogram.m
Plots the histogram of given image
Using imhist (generates the histogram of image)
plot128.m
Plots y=128 of given image
Using plot (plot the graph of given coordinates)

Running MATLAB code:

In order to obtain the results which are shown in the next section, just run $main\_filtering$ to get the images and histograms after applying the non-linear filters mentioned in Methods section.

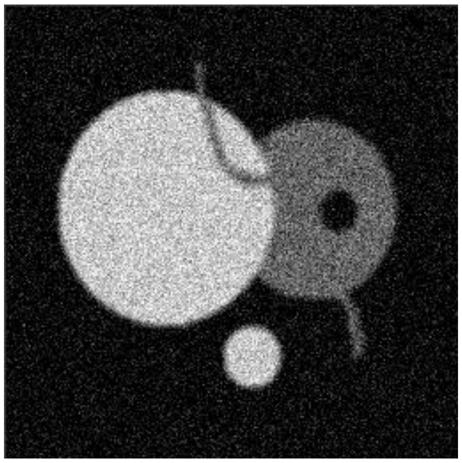
For anisotropic diffusion, run file $main\_anisotropic$. It will save all the images obtained after required iteration of anisotropic algorithm. Also, we are required to obtain histograms and y=128 plot of each of the images obtained.

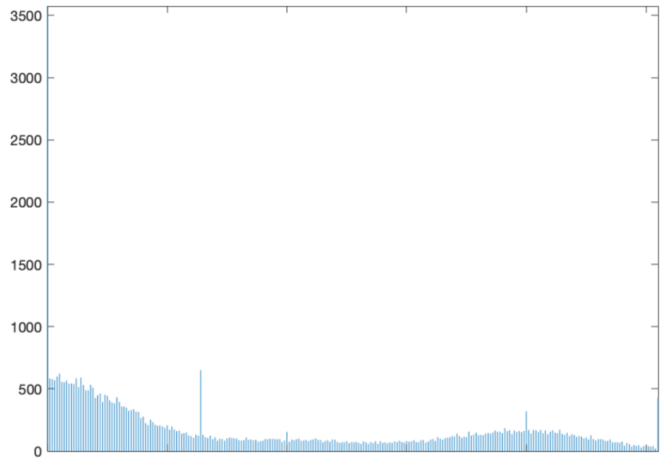
For the image segmentation, we manually threshold each of the image using histograms. Thresholding is done by calling file $manual\_threshold.m$.

C. Results

Part A:

All the five filters discussed in Methods section were applied on *'disk.gif'*. These filters were applied recursively on the given image for 5 iterations. Results below give the image obtained after 1st and 5th iteration of each filter. Also, the image histogram was obtained for the filtered image after 5th iteration. The mask size which was used in each of the filter was 5×5.


(a) Original Image (disk.gif)



(b) Histogram of original Image

Figure 1: Original Image and its histogram.

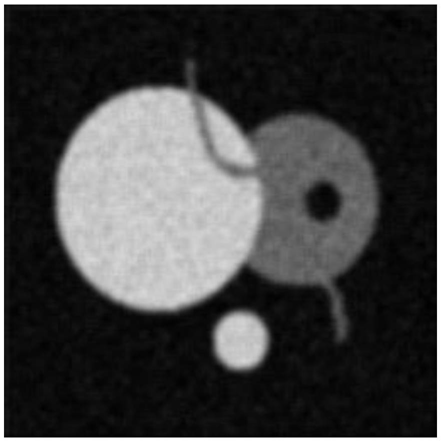
The mean of the largest disk is 160.3561 and the standard deviation of the same disk is 75.0696. The largest disk location was found using $imtools$ functionality of MATLAB. It was found to be $[50:180, 32:152]$.

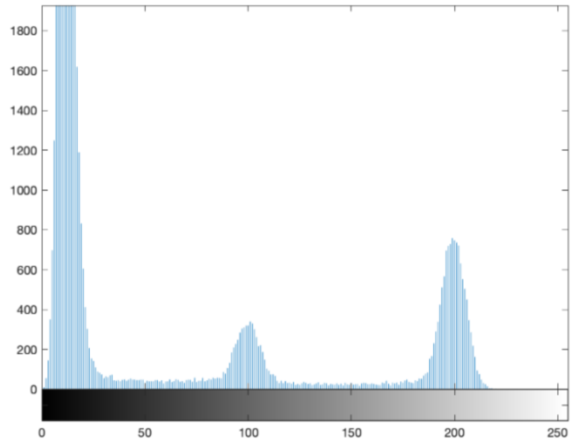
Mean Filter

A 5×5 mean filter was applied on the image shown in figure 1(a). This filter was applied recursively for 5 iterations where each pixel value was replaced with the mean of the pixels present within the mask which was used as a sliding window over image. Fig. 2: (a) and (b) shows the image and histogram when mean filter was applied for the first time and Fig. 2: (c) and (d) shows the image and histogram in the 5th iteration of mean filter respectively.

(a) Image after 1st iteration

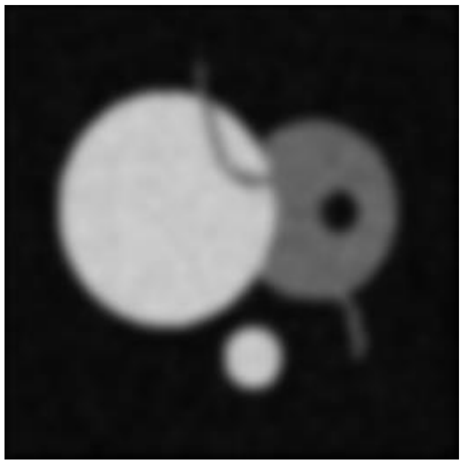
(b) Histogram of image on LHS





(c) Image after 5th iteration

(d) Histogram of image on LHS



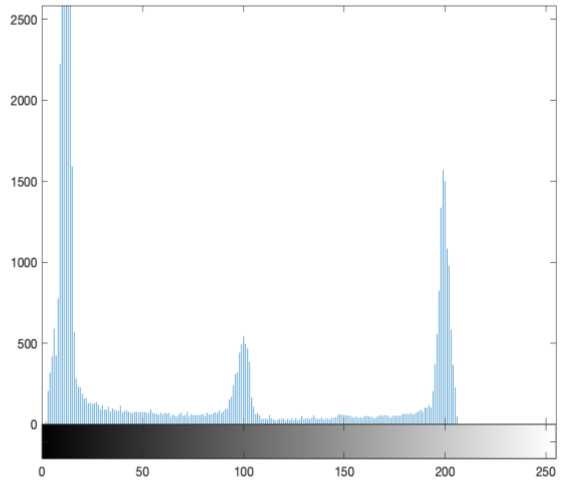


Figure 2: Image results and histogram after applying mean filter

Below is a tabular representation of the iteration number, mean value and standard deviation value observed for the interior region (square window) of the large disk:

Iteration #	Mean	Standard Deviation
1	160.0532	68.5324
5	159.1641	64.9527

What we observe:

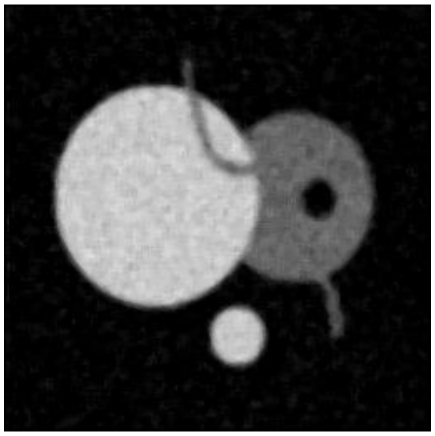
It is pretty much evident from the images in Fig. 2 that noise is getting reduced from the image. But on the same side, we can see that image is also getting blurred. The histograms in Fig. 2 (d) have higher peaks although the valleys are same compared to Fig. 2 (b).

The edges in the image are getting lost after applying the mean filter. The mean and standard deviation values reduce as the number of iterations increase. Also, to note, the edges get blurred, which generally is not ideal.

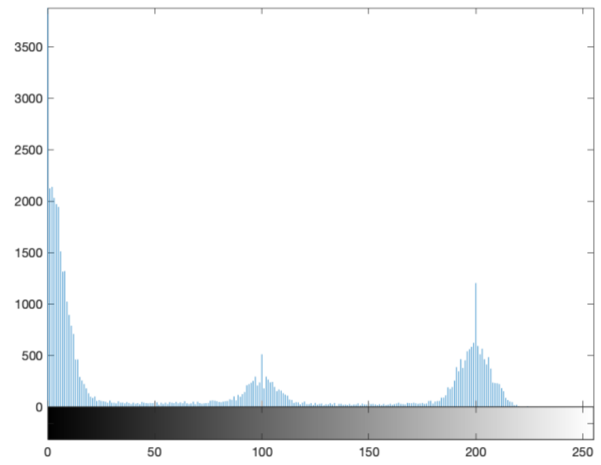
Median Filter

A 5×5 median filter was applied on the image shown in figure 1. This filter was applied recursively for 5 iterations where each pixel value was replaced with the median of the pixels present within the mask which was used as a sliding window over image. Fig. 3: (a) and (b) shows the image and histogram when median filter was applied for the first time and Fig. 3: (c) and (d) shows the image and histogram in the 5th iteration of median filter respectively.

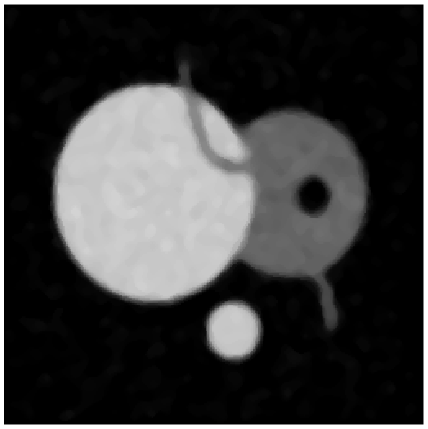
(a) Image after 1st iteration



(b) Histogram of image on LHS



(c) Image after 5th iteration



(d) Histogram of image on LHS

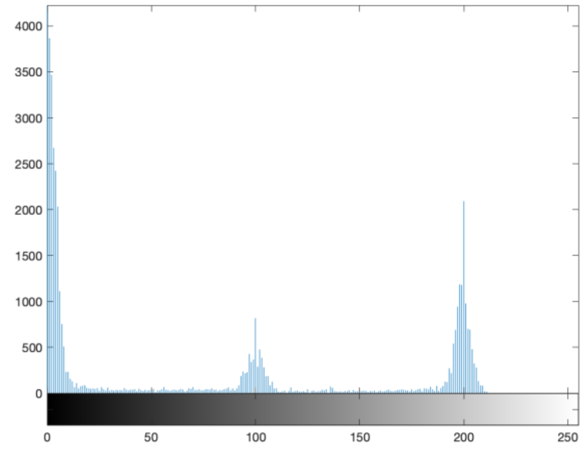


Figure 3: Image results and histogram after applying median filter

Below is a tabular representation of the iteration number, mean value and standard deviation value observed for the interior region (square window) of the large disk:

Iteration #	Mean	Standard Deviation
1	158.9099	72.1864
5	158.5453	71.7934

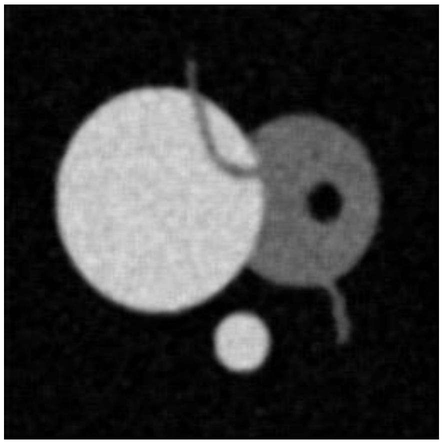
What we observe:

Comparing Fig. 3 (a) and (c), we can see that there's noise reduction, hence more smoothness (not so grainy). But unlike the mean filter, edges are preserved here. The mean and standard deviation values almost remain same as the number of iterations increase. The histograms in Fig. 3 (d) have higher peaks although the valleys are same compared to Fig. 3 (b).

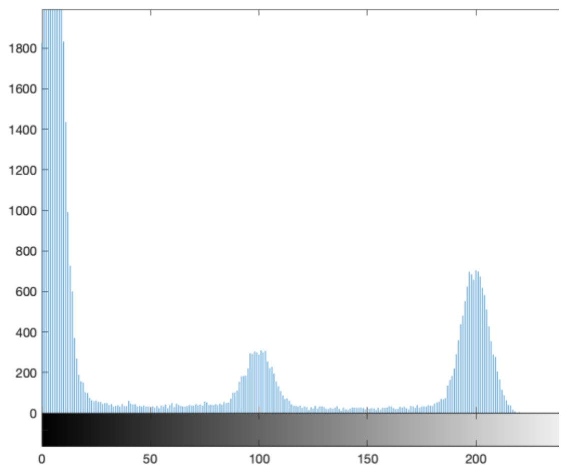
Alpha-Trimmed Mean Filter

A 5×5 alpha-trimmed mean filter was applied on the image shown in figure 1. We used $\alpha = 0.25$ in this alpha-trimmed mean filter. This filter was applied recursively for 5 iterations where each pixel value was replaced with the value obtained using the equation mentioned in Methods section. Fig. 4: (a) and (b) shows the image and histogram when alpha-trimmed mean filter was applied for the first time and Fig. 4: (c) and (d) shows the image and histogram in the 5th iteration of this filter respectively.

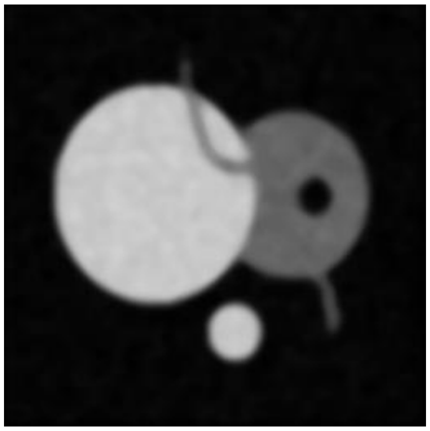
(a) Image after 1st iteration



(b) Histogram of image on LHS



(c) Image after 5th iteration



(d) Histogram of image on LHS

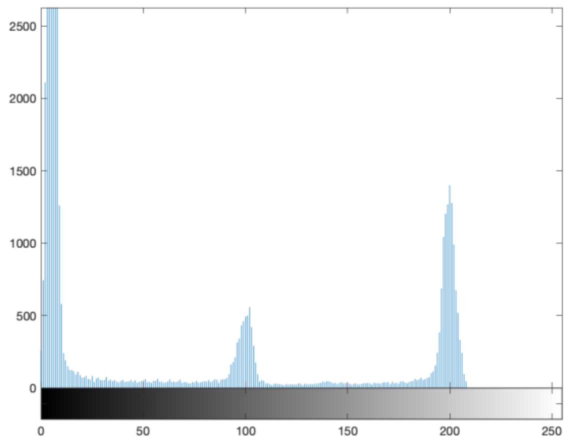


Figure 4: Image results and histogram after applying alpha trimmed mean filter

Below is a tabular representation of the iteration number, mean value and standard deviation value observed for the interior region (square window) of the large disk:

Iteration #	**Mean**	**Standard Deviation**
1	159.285	71.1476
5	158.6186	69.7933

What we observe:

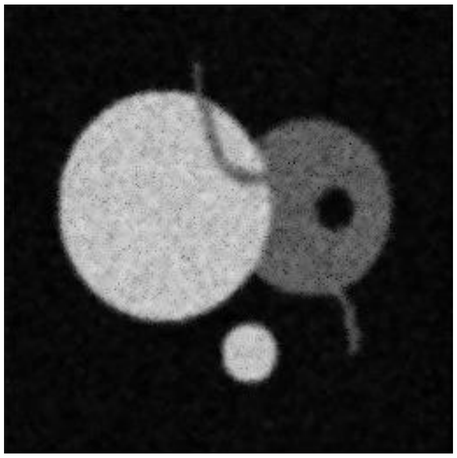
Alpha-trimmed mean filter is less influenced by salt or pepper noise compared to mean filter based on the standard deviation values. Also, mean filter is a special case of alpha-trimmed mean filter when the value of $\alpha = 0$. The edges are preserved, and slight smoothening can be observed as we perform more iterations. The histograms in Fig. 4 (d) have higher peaks although the valleys are same compared to Fig. 4 (b).

This filter method is computationally intense since each iteration requires sorting operation to be performed.

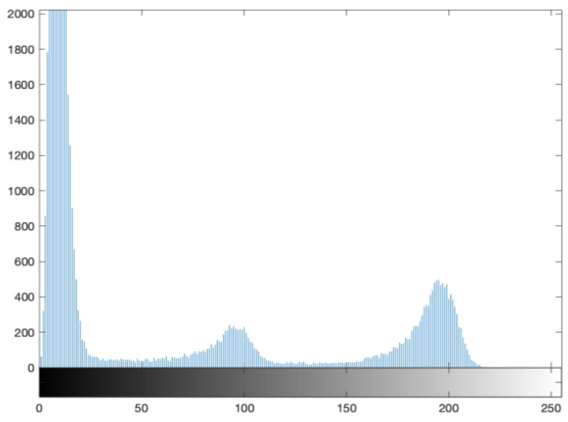
Sigma Filter

A 5×5 sigma filter was applied on the image shown in figure 1. We used $\sigma = 20$ in this filter. This filter was applied recursively for 5 iterations where each pixel value was replaced with the value obtained using the equation mentioned in Methods section. Fig. 5: (a) and (b) shows the image and histogram when this filter was applied for the first time and Fig. 5: (c) and (d) shows the image and histogram in the 5th iteration of this filter respectively.

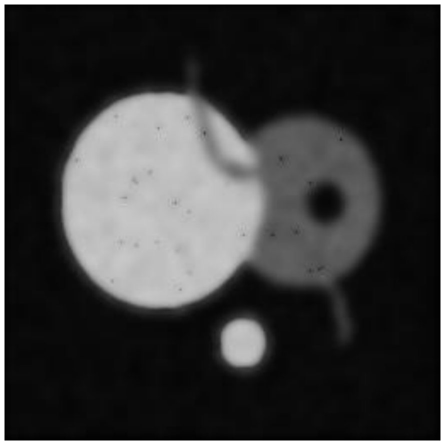
(a) Image after 1st iteration



(b) Histogram of image on LHS



(c) Image after 5th iteration



(d) Histogram of image on LHS

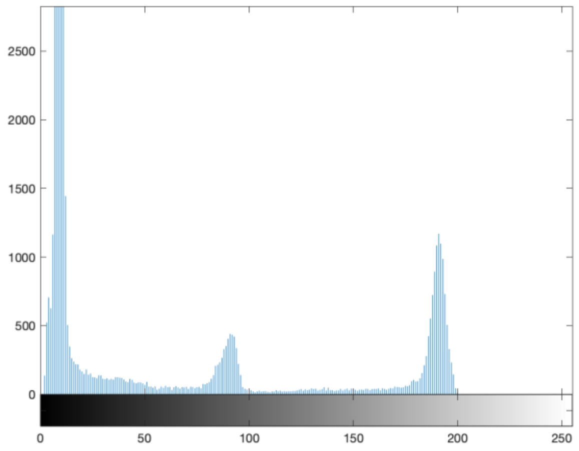


Figure 5: Image results and histogram after applying alpha trimmed mean filter

Below is a tabular representation of the iteration number, mean value and standard deviation value observed for the interior region (square window) of the large disk:

Iteration #	Mean	Standard Deviation
1	150.9537	69.5644
5	144.2935	69.2619

What we observe:

Clearly, there’s smoothening happening, but evidently on the foreground of the image, the pepper noises still remain, if we compare Fig. 5 (a) and (c). The histograms in Fig. 5 (d) are almost the same compared to Fig. 5 (b).

Symmetric Nearest Neighbor Mean Filter (SNNMF Filter)

A 5×5 symmetric nearest neighbor filter was applied on the image shown in figure 1. This filter was applied recursively for 5 iterations. In each iteration, symmetrically opposite pairs of pixels are formed. All those points are selected which are most similar to the pixel we want to replace. Then, we find the average of all those selected pixels along with the pixel to be replaced. This average value is chosen as the value of the center most pixel (pixel whose value needs to be replaced) in the mask. Fig. 6: (a) and (b) shows the image and histogram when this filter was applied for the first time and Fig. 6: (c) and (d) shows the image in the 5th iteration of this filter respectively.

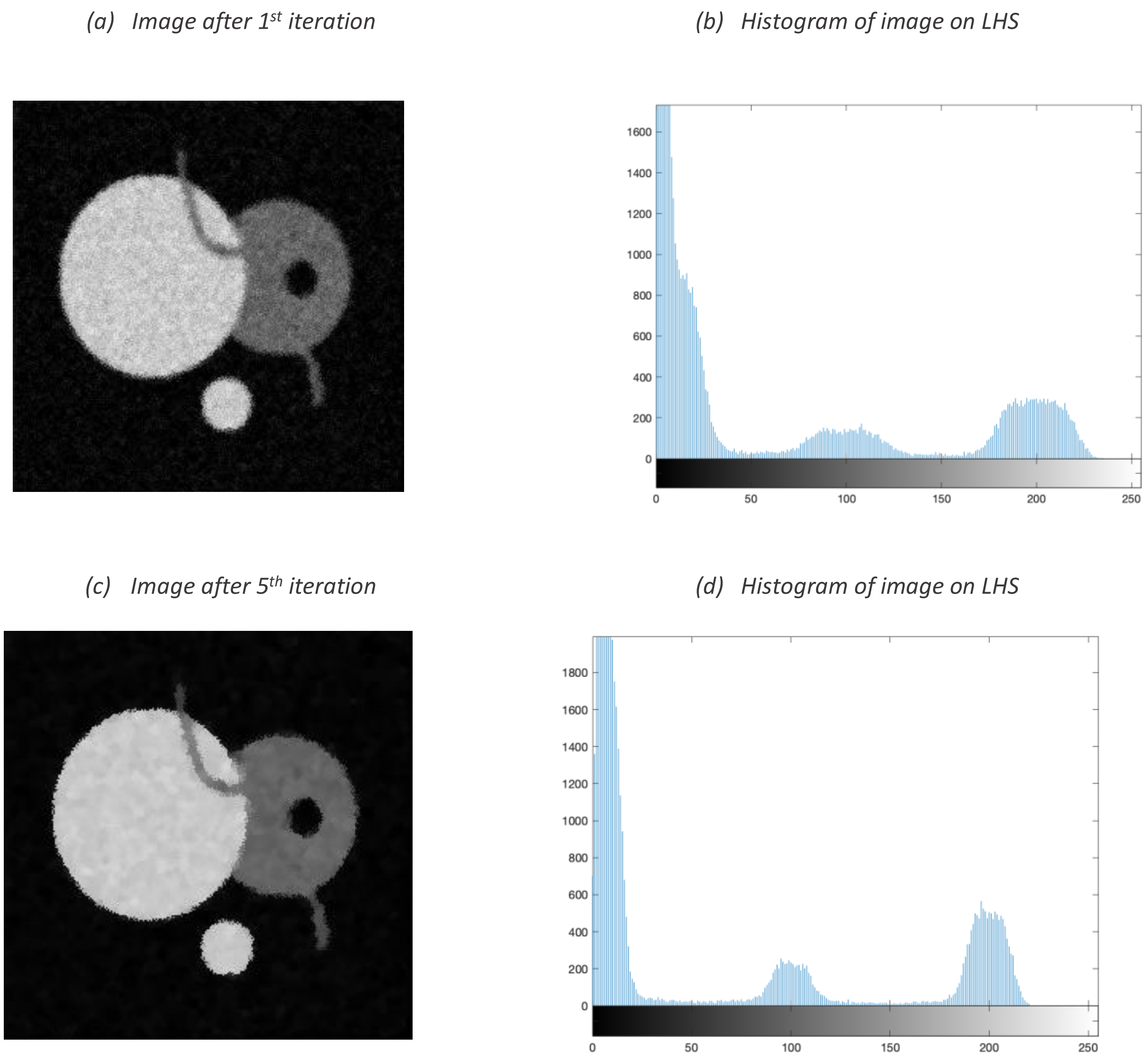


Figure 6: Images and histograms after applying SNNM Filter

Below is a tabular representation of the iteration number, mean value and standard deviation value observed for the interior region (square window) of the large disk:

Iteration #	Mean	Standard Deviation
1	159.851	72.5448
5	159.6422	73.8704

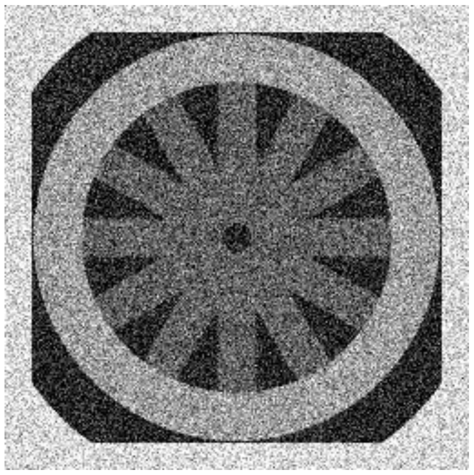
What we observe:

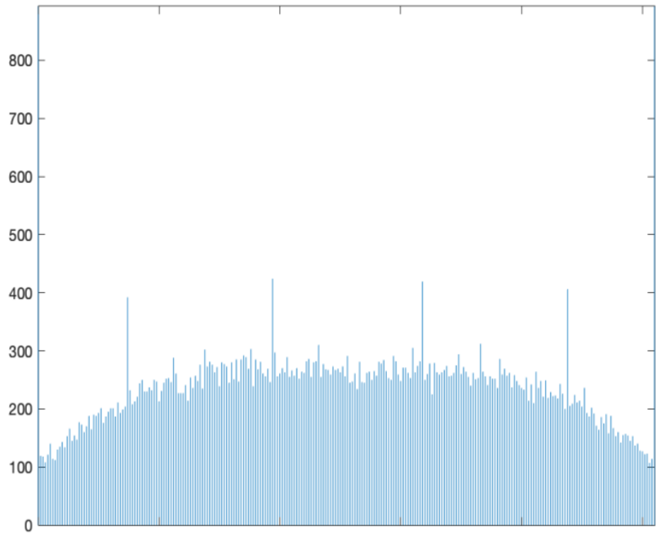
Comparing images in Fig 6. (a) and (c), we observe that there's quite a lot of salt or pepper noise removal and distinct edge identification (sharper) without blurring, which is ideal. The histograms in Fig. 6 (d) have higher peaks and slightly lower valleys as compared to Fig. 6 (b).

Part 2

In this problem, we have been asked to implement anisotropic diffusion algorithm. We used the equation 7 of Perona & Malik which is explained in Methods section.

In the first part of this second problem, we have been asked to run anisotropic diffusion on *'cwheelnoise.gif'*. This image is shown in Figure below. The histogram of this image is also shown besides it.


(a) Original image(cwheelnoise.gif)



(b) Grayscale histogram of cwheelnoise image

Figure 7: Image cwheelnoise.gif and its histogram

From the images shown above, it is pretty clear that the image is very noisy. The histogram has lot of spikes and it shows there is lot of disturbance in the image. Using anisotropic diffusion, we intend to reduce this noise and smoothen the image. There are two types of $g(\cdot)$ –

(a) Exponential
(b) Inverse Quadratic.

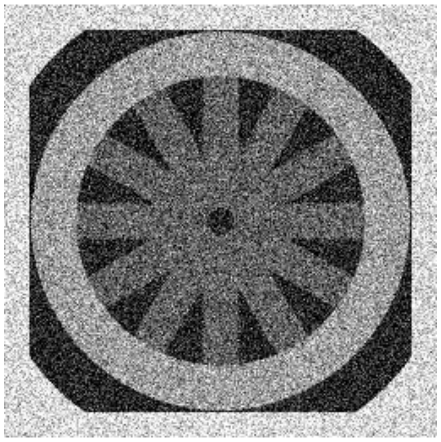
We ran our algorithm on both the conduction coefficients with different values of K. Also, we have been asked to plot $y = 128$ for every image obtained.

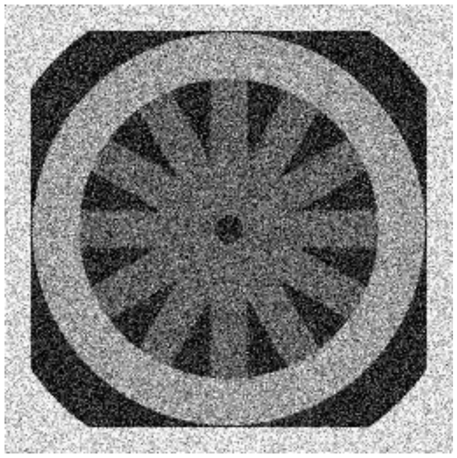
Let us first show the results with Inverse Quadratic Conduction coefficients:

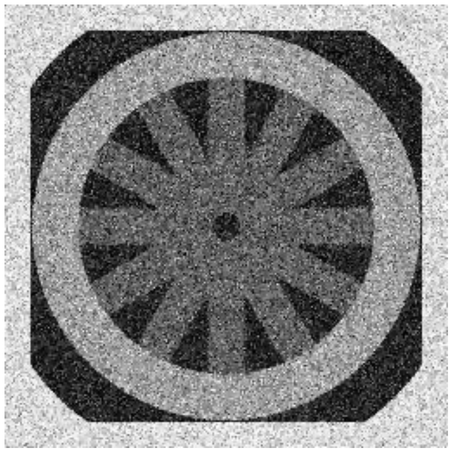
Quadratic Conduction Coefficients

We ran our algorithm using quadratic coefficients for $K = \{6,7,8,9\}$. Below are the results:

K=6


(a) Iteration 0 (original image)


(b) Iteration 5


(c) Iteration 20

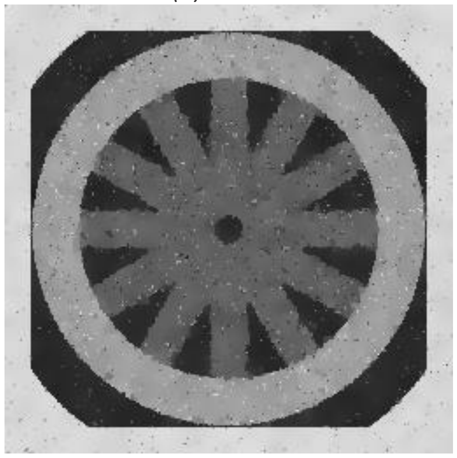
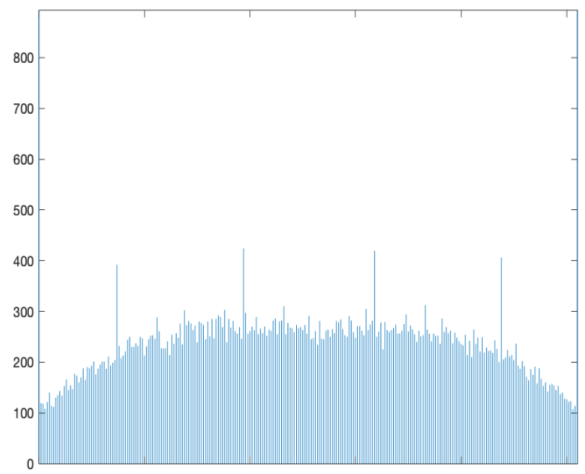
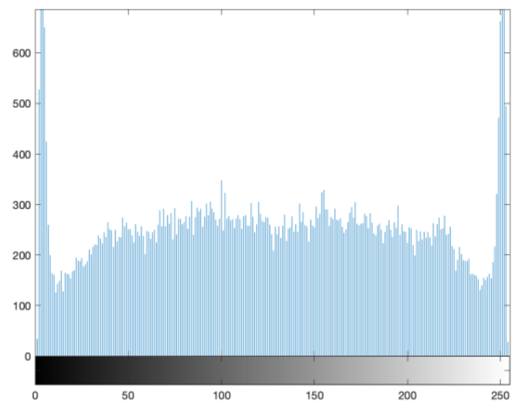

(d) Iteration 100

Figure 8: Images after running anisotropic diffusion with k=6 and g(•)= quadratic

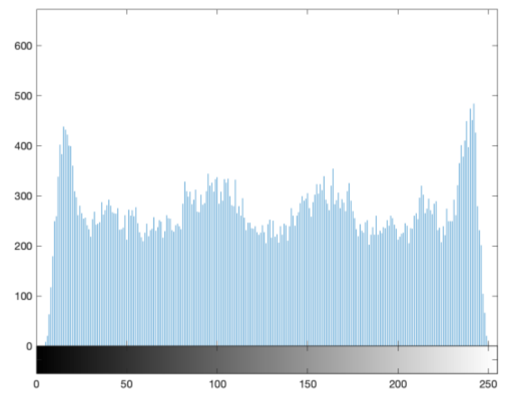
Images in Fig. 8 are obtained after running anisotropic diffusion algorithm with $k = 6$ and $g(\cdot) = 'quadratic'$. We also implemented the histograms for these iterations and y=128 plot for each image. Further we did manual threshold of the image segmenting the gray spokes of the wheel.



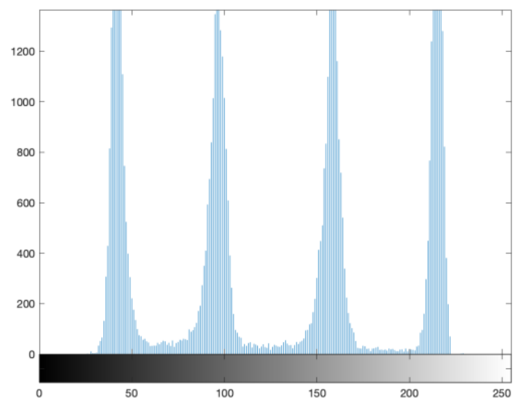
(a) Iteration 0



(b) Iteration 5



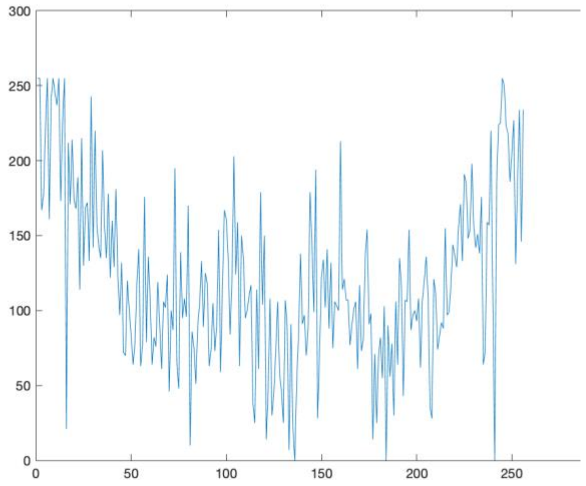
(c) Iteration 20



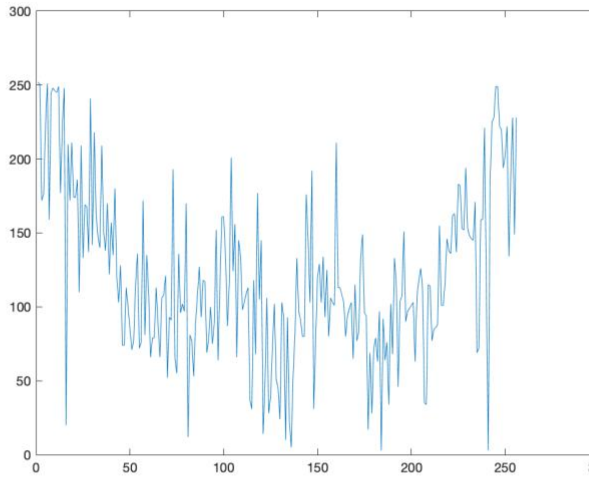
(d) Iteration 100

Figure 9: Corresponding histograms of images in Fig 8

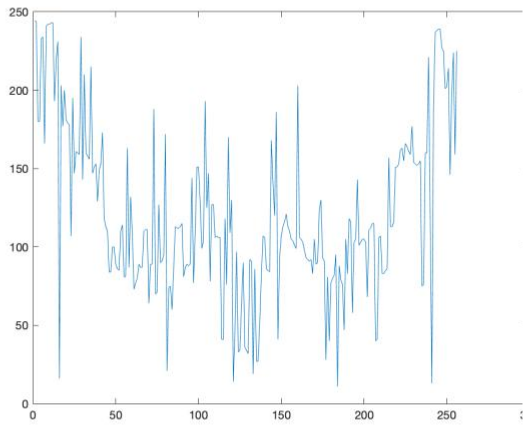
Fig. 9 shows that the histogram is getting concentrated on four values of the spectrum indicating the four shades of grey originally used in the image, and the noise is getting reduced with each successive iteration. Below is the y=128 plots of the images obtained.

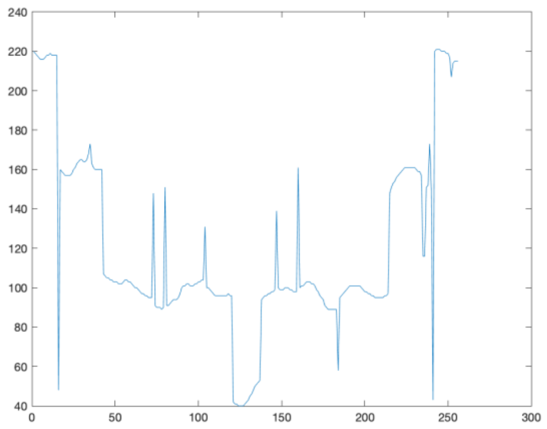


(a) Iteration 0



(b) Iteration 5

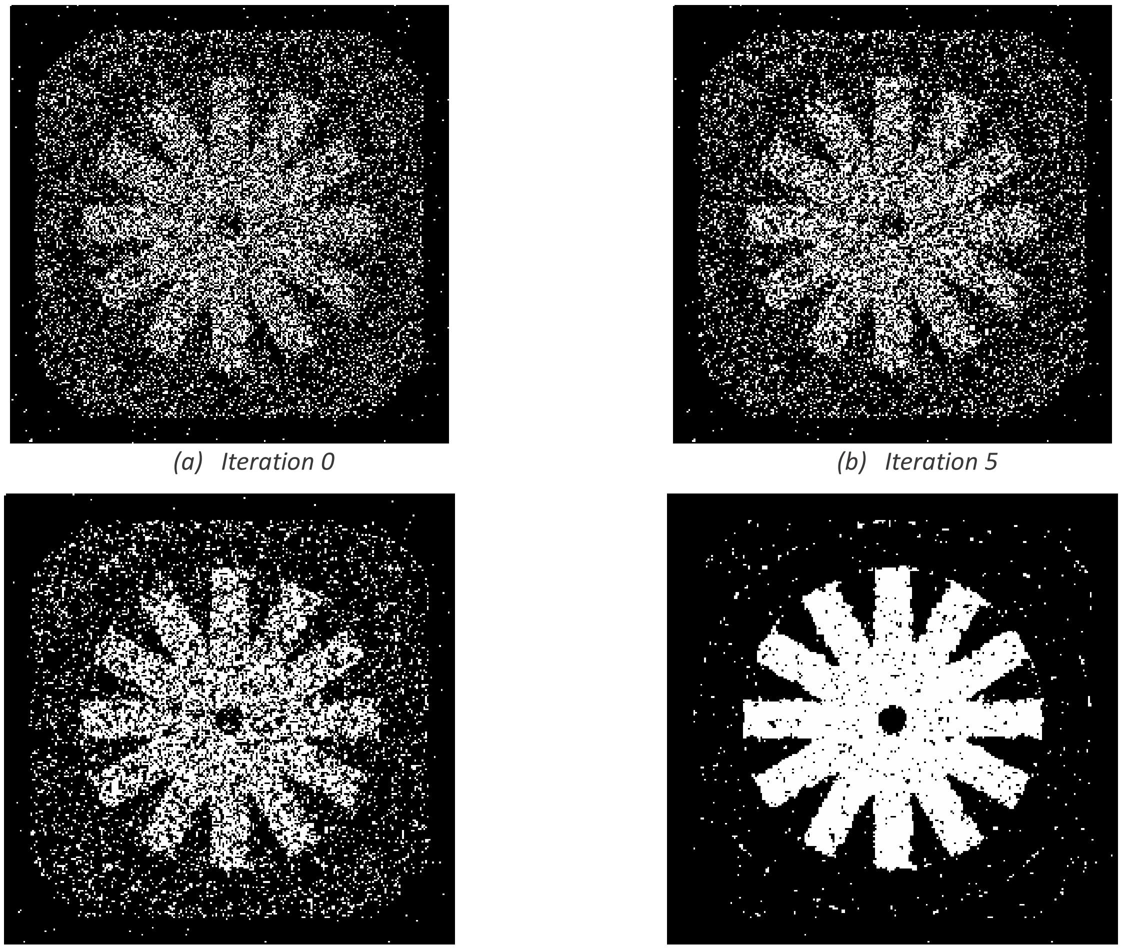




(c) Iteration 20 *(d) Iteration 100*

Figure 10: Corresponding y=128 plots of images in Fig 8

From Fig 10, it is seen that with the increase in iteration, plot is getting more symmetrical, as the original image. It also proves that noise is getting reduced, by reduced spikes in the graph. All the images obtained were manually threshold using the histogram of the images obtained. Image was segmented to obtain the gray spokes in wheel. We checked the histogram of the images and found that the gray spokes in the wheel lies in the 2nd peak. So, we picked the two lowest ends of the 2nd peak and came up with a value range [72,120] to select the spokes grey area. Figure 11 shows the segmented images of the one obtained in Fig 8.



(a) Iteration 0

(b) Iteration 5

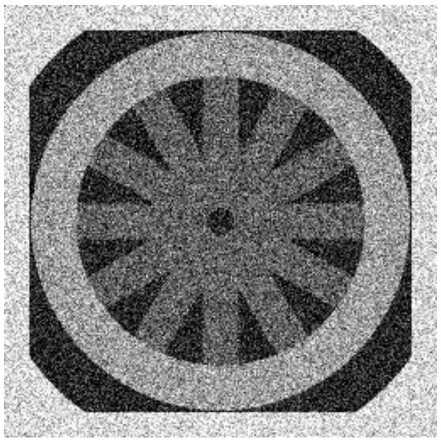
(c) Iteration 20

(d) Iteration 100

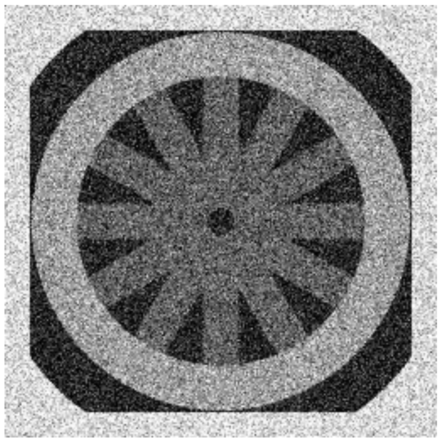
Figure 11: Corresponding segmented images of the ones in Fig 8.

The segmented images obtained in fig 11 also shows that noise is getting reduced at each iteration. Gray colored spokes are coming cleared with each iteration. But, still Fig. 11(d) is a noisy image. Results further will show if this image can be improved or not.

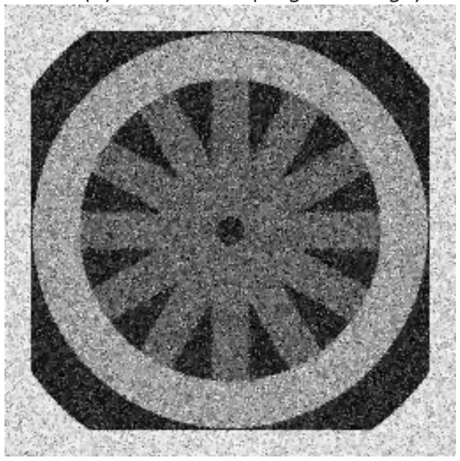
K=7



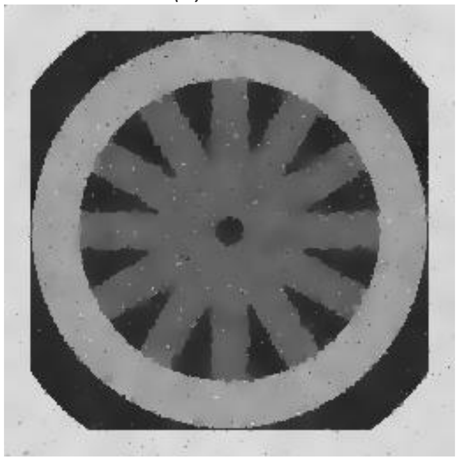
(a) Iteration 0 (original image)



(b) Iteration 5



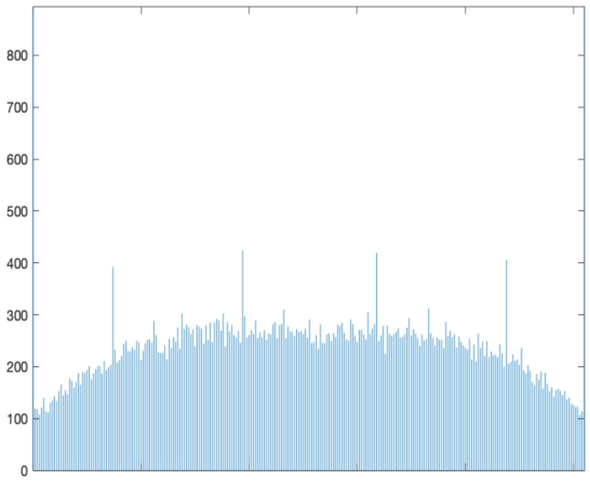
(c) Iteration 20



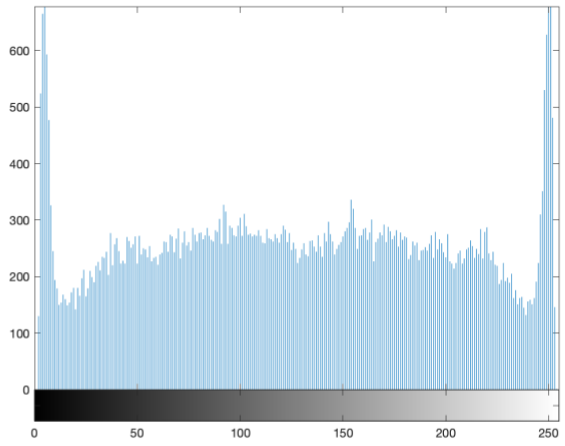
(d) Iteration 100

Figure 12: Images after running anisotropic diffusion with k=7 and g()= quadratic

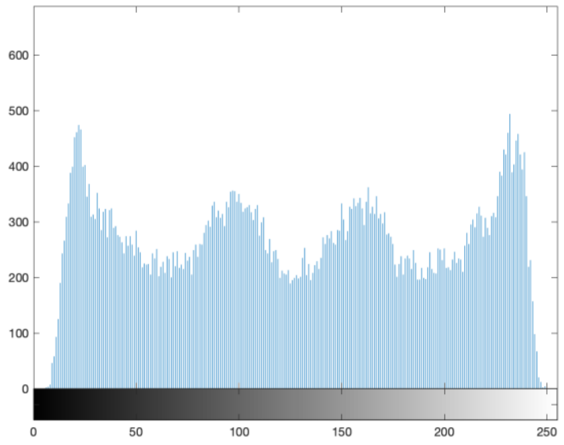
Fig. 12 shows the images obtained with K=7. Fig. 12(d) seems to be better than Fig. 8(d) i.e. there is less noise.



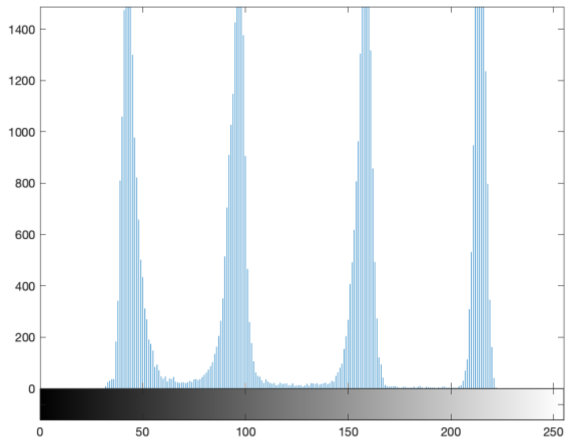
(a) Iteration 0



(b) Iteration 5



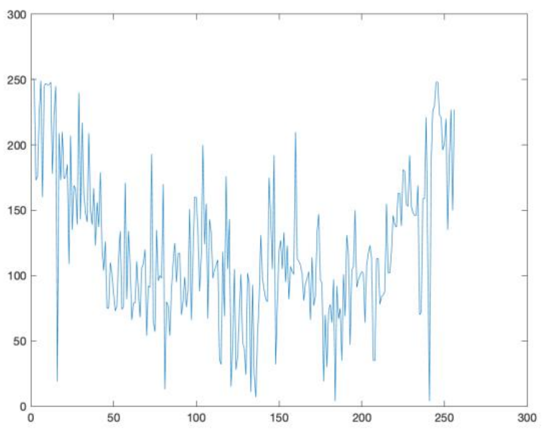
(c) Iteration 20



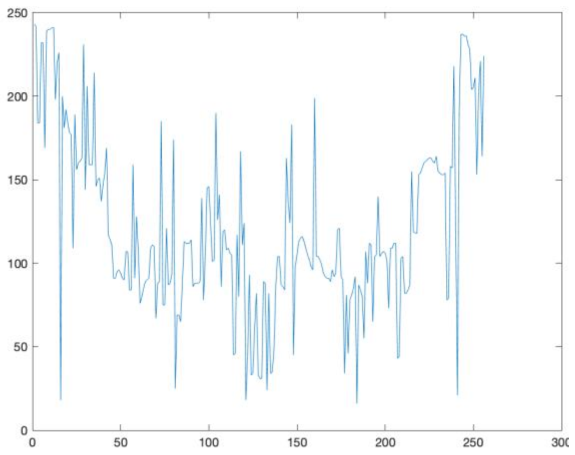
(d) Iteration 100

Figure 13: Corresponding histograms of images in Fig 12

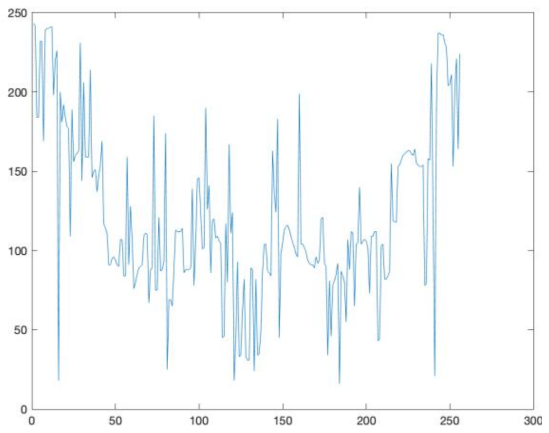
Histograms also show less fluctuations. We have more finer peaks in Fig 13(d).



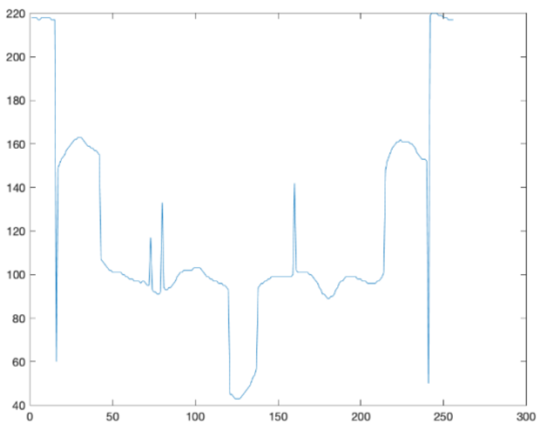
(a) Iteration 0 (original image)



(b) Iteration 5



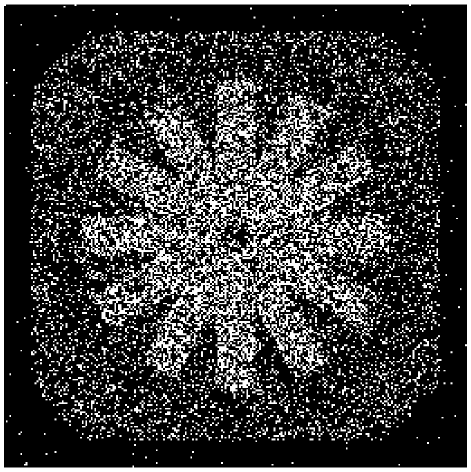
(c) Iteration 20



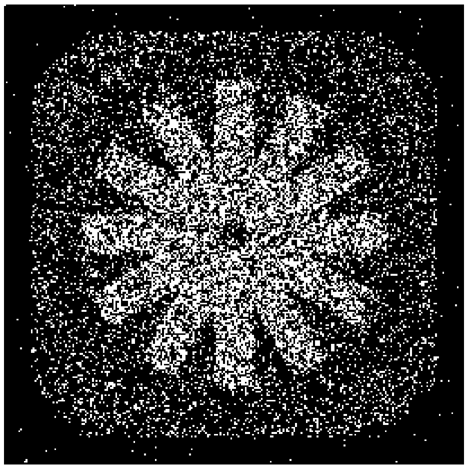
(d) Iteration 100

Figure 14: Corresponding y=128 plots of images in Fig 12

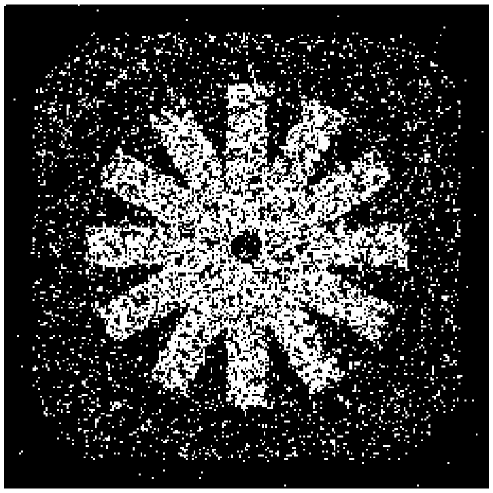
Again, the plot of y=128 is getting more symmetrical. There is much less noise than the ones obtained earlier.



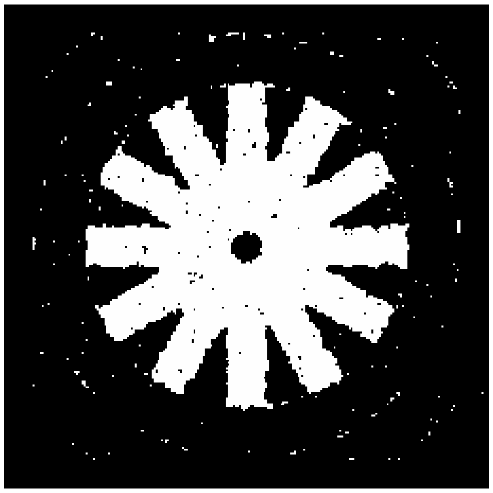
(a) Iteration 0



(b) Iteration 5



(c) Iteration 20

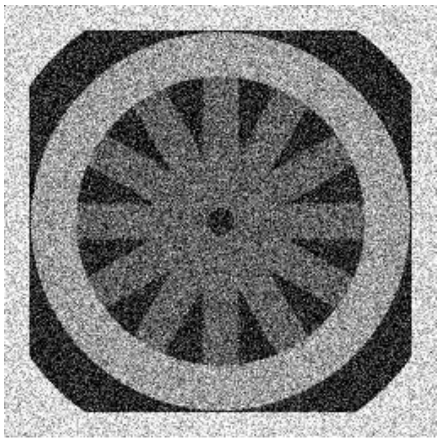


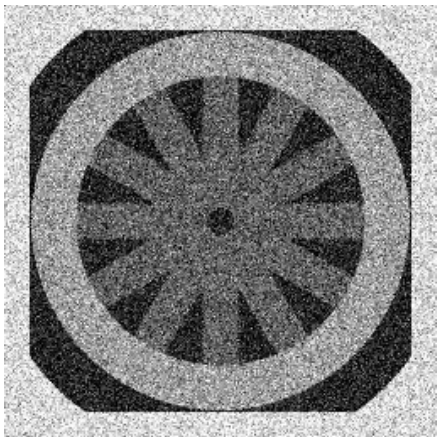
(d) Iteration 100

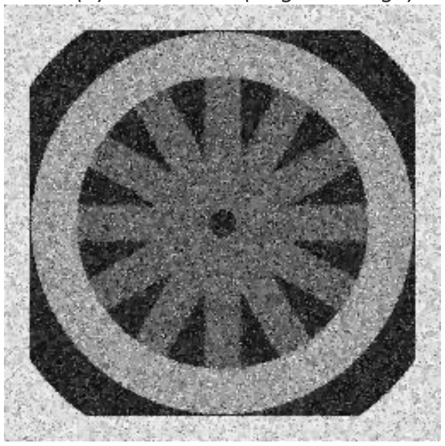
Figure 15: Corresponding segmented images of the ones shown in Fig 12.

The segmented image at 12(d) is much clearer and less noisy than in Fig. 11(d). Edges are getting preserved more.

K=8


(a) Iteration 0 (original image)


(b) Iteration 5


(c) Iteration 20

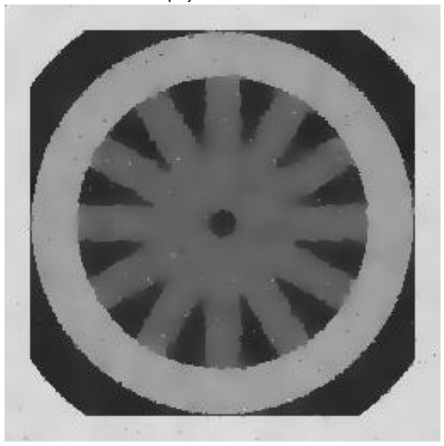
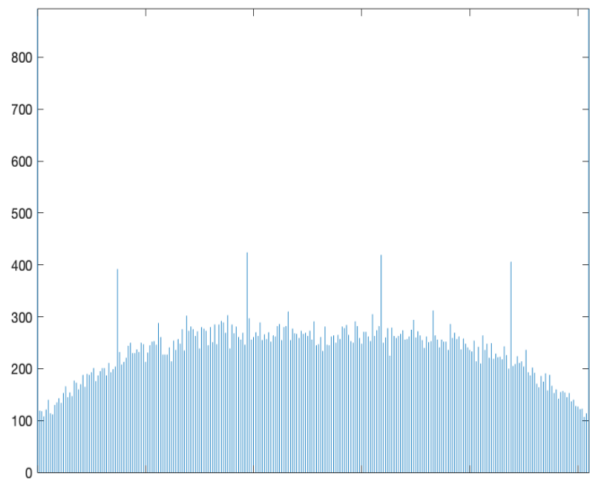
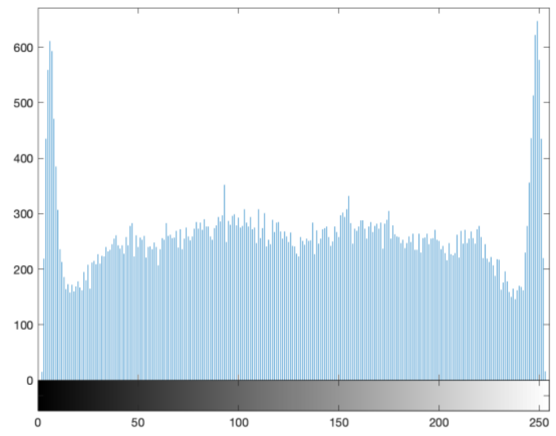

(d) Iteration 100

Figure 16: Images after running anisotropic diffusion with k=8 and g () = quadratic

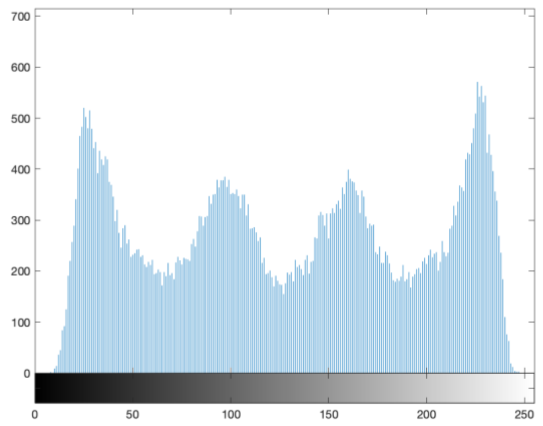
Fig. 16 shows the images obtained with K=8. Fig. 16(d) seems to be better than Fig. 12(d) in terms of noise. However, we also feel that there is a little bit loss of edges in this image. Thus, increasing the value of K might deteriorate the image more.



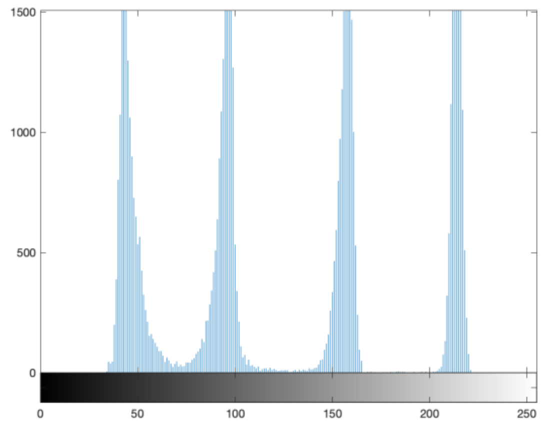
(a) Iteration 0



(b) Iteration 5



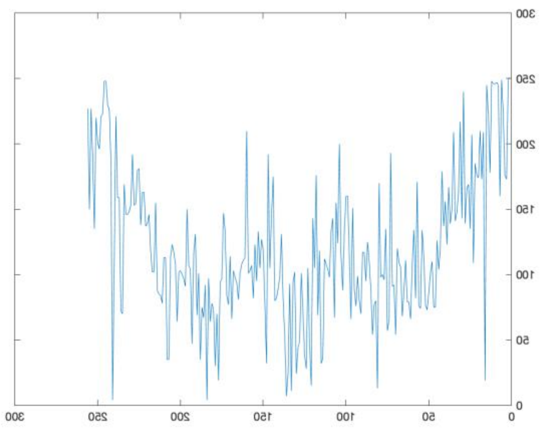
(c) Iteration 20



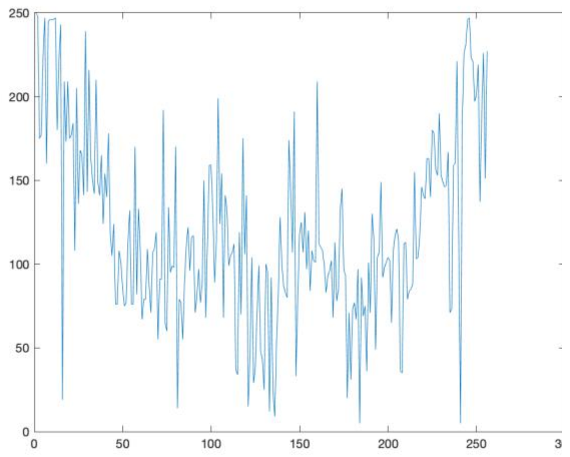
(d) Iteration 100

Figure 17: Corresponding histograms of images in Fig 16

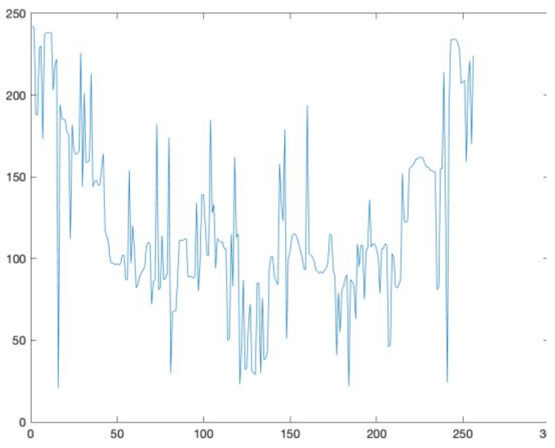
We have more finer peaks hear. Thus, we have reduced a lot of noise here.



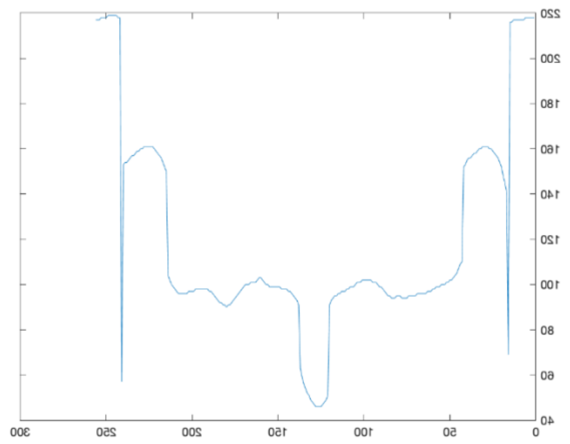
(a) Iteration 0



(b) Iteration 5



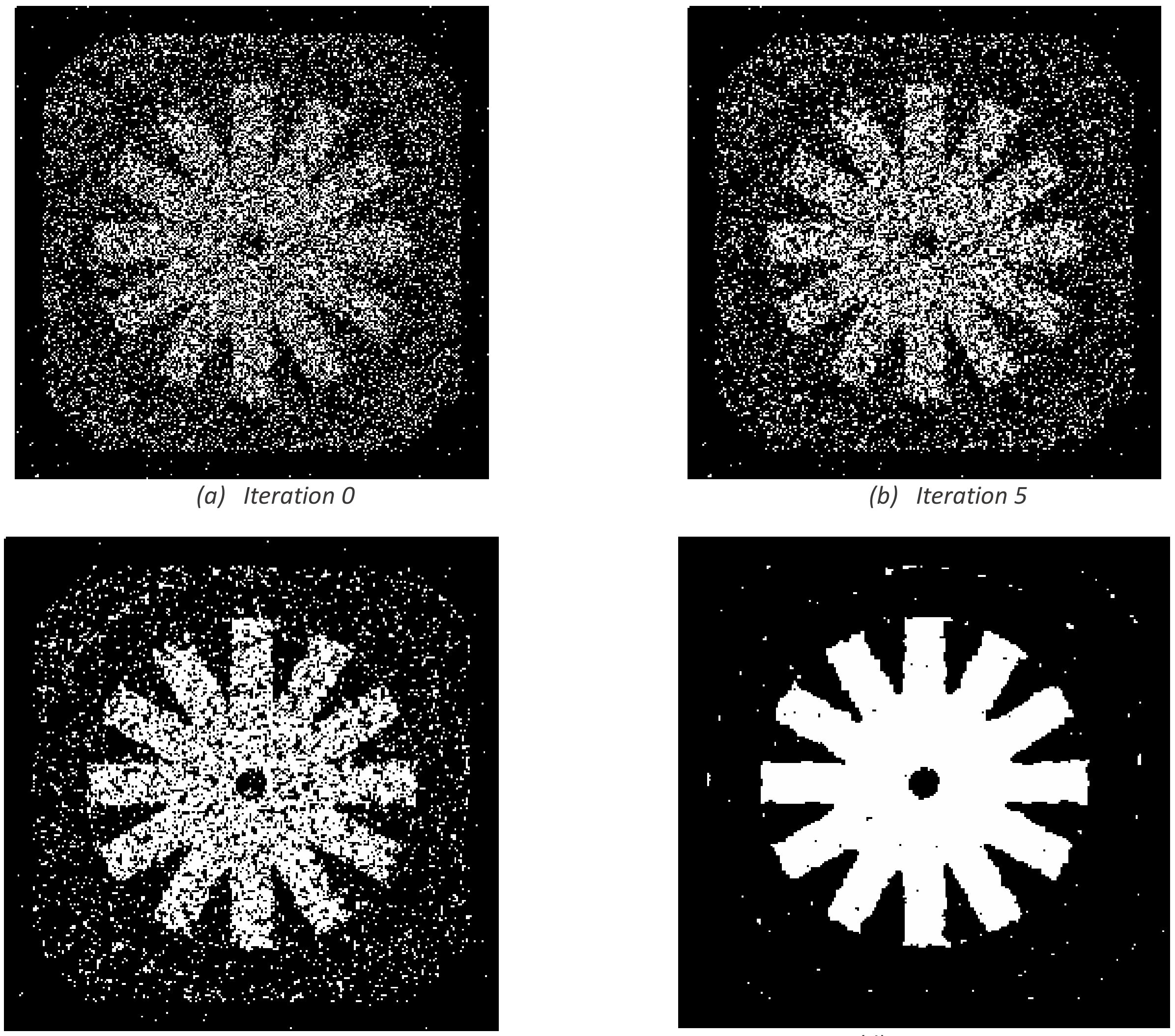
(c) Iteration 20



(d) Iteration 100

Figure18: Corresponding y=128 plots of images in Fig 16

Fig 18 shows the y=128 plots of the images obtained. It is pretty much evident that the plot in Fig 18(d) is much symmetrical than the ones obtained earlier.

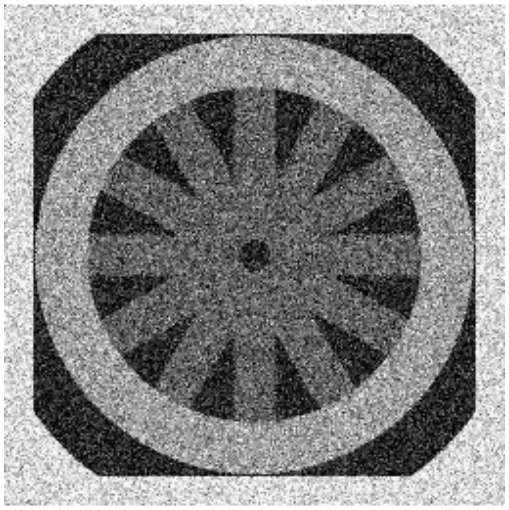

(a) Iteration 0 (b) Iteration 5

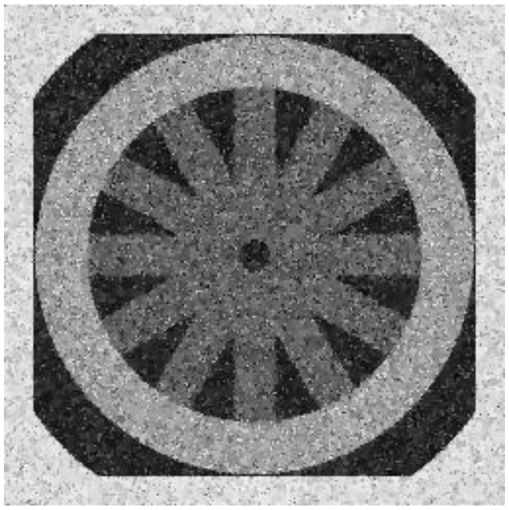
(c) Iteration 20 (d) Iteration 100

Figure 19: Corresponding segmented images of the ones in Fig 16

This is the best segmented image we have obtained till now. Edges are preserved and the noise is least in Fig. 19(d) than the ones obtained with other value of K. Also, it is pretty much evident that the image is getting diffused more with the increase in iterations.

K=9


(a) Iteration 5


(b) Iteration 20

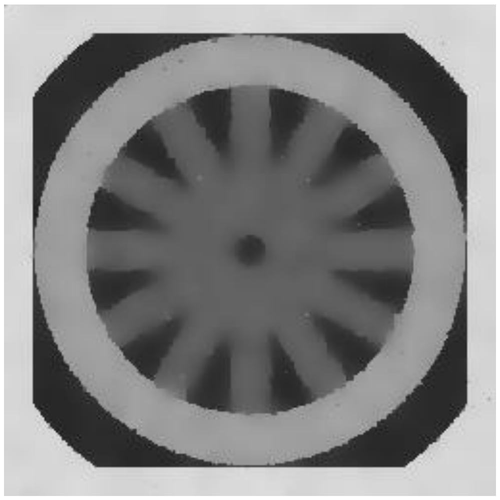
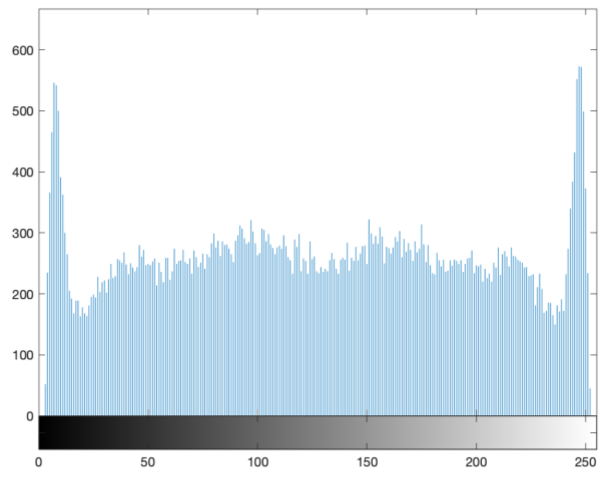
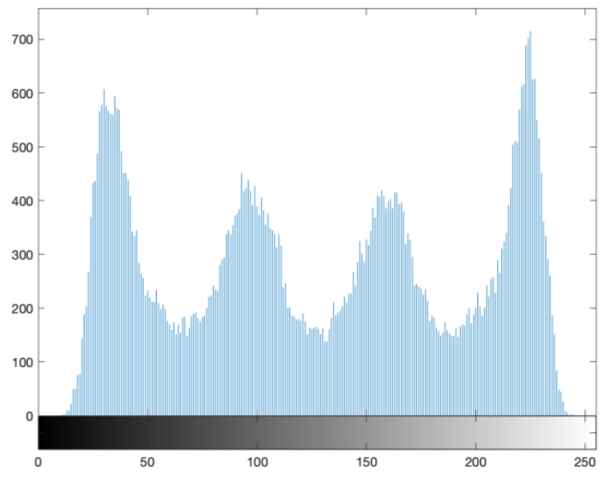

(c) Iteration 100

Figure 20: Images after running anisotropic diffusion with k=9 and g () = quadratic

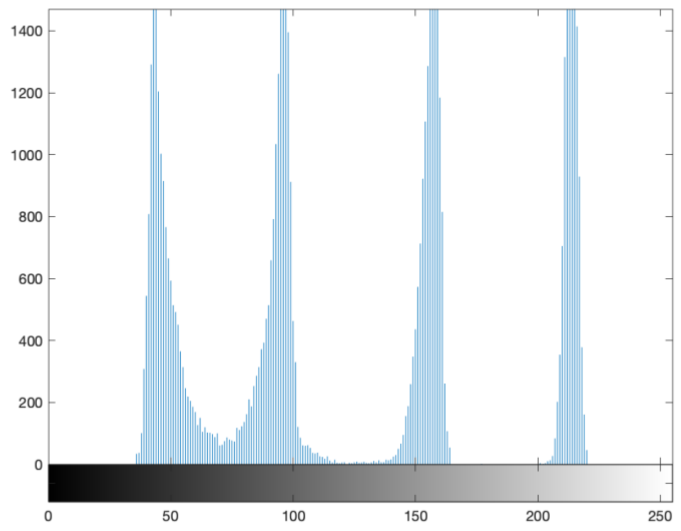
The images obtained here have much less noise. But the edges are getting lost. The image is getting blurred as per Fig 20(d).



(a) Iteration 5



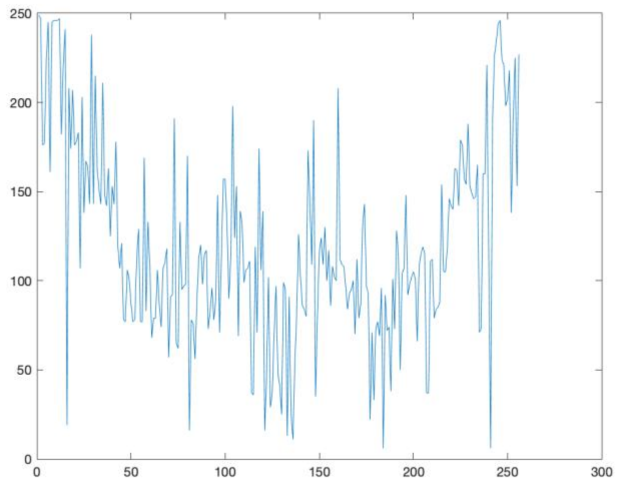
(b) Iteration 20



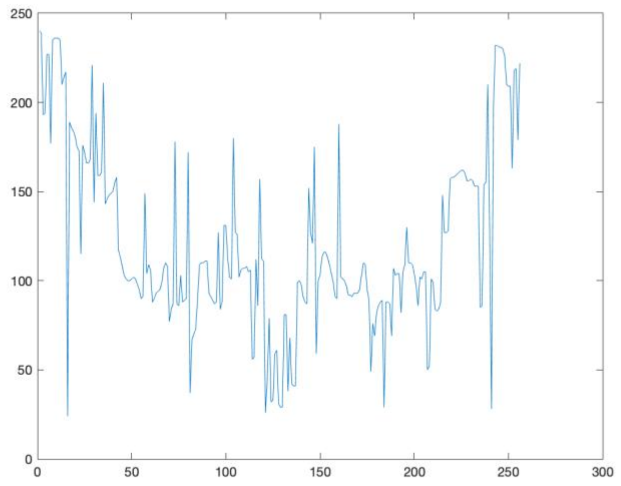
(c) Iteration 100

Figure 21: Corresponding histogram of images shown in Fig 20

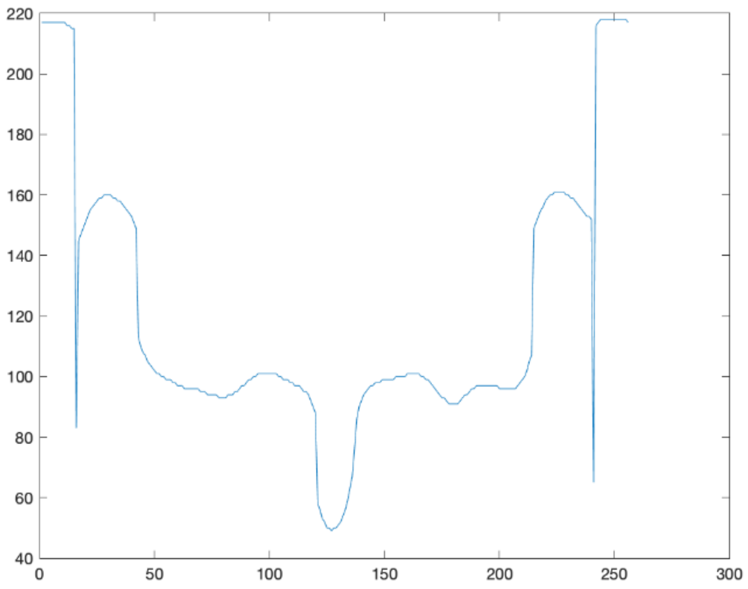
The histograms show finer peaks, but they also reflect the loss of edges as we have obtained a valley between 1st peak and 2nd peak.



(a) Iteration 5



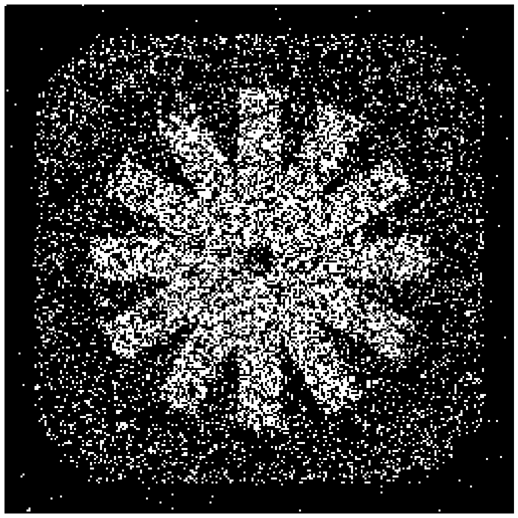
(b) Iteration 20



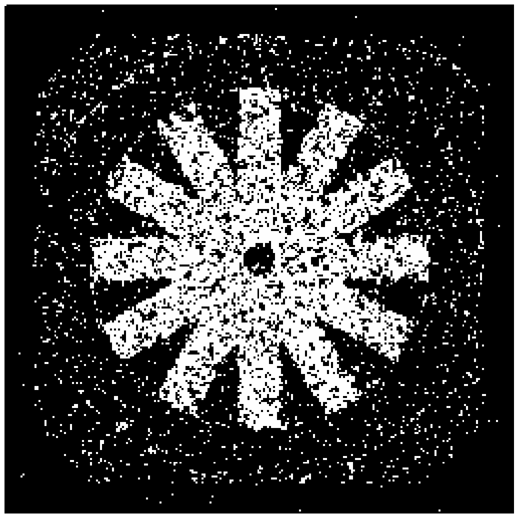
(c) Iteration 100

Figure 22: Corresponding y=128 plot of images shown in Fig 20

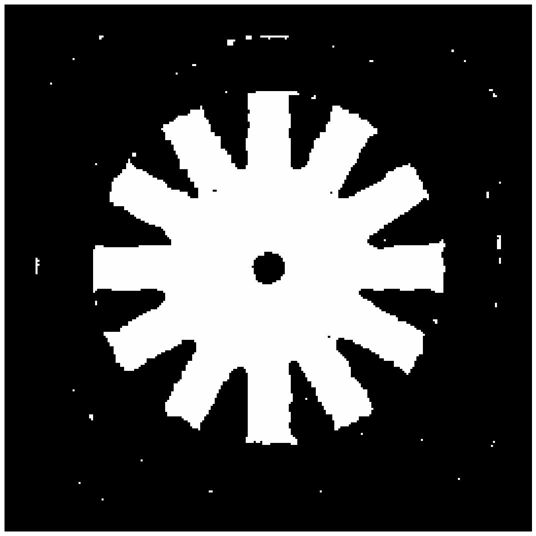
Fig. 22 shows the y=128 plot. The plot is very symmetrical.



(a) Iteration 5



(b) Iteration 20



(c) Iteration 100

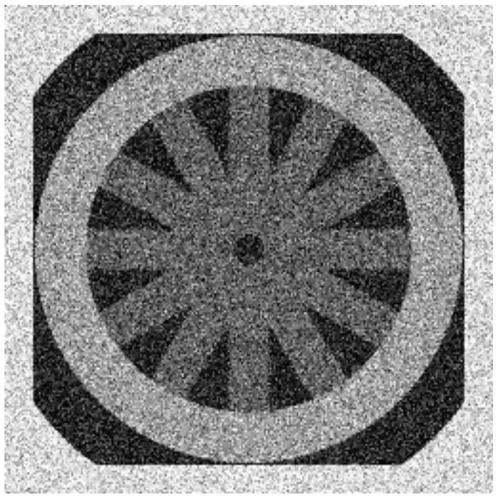
Figure 23: Corresponding segmented images of the ones shown in Fig 20

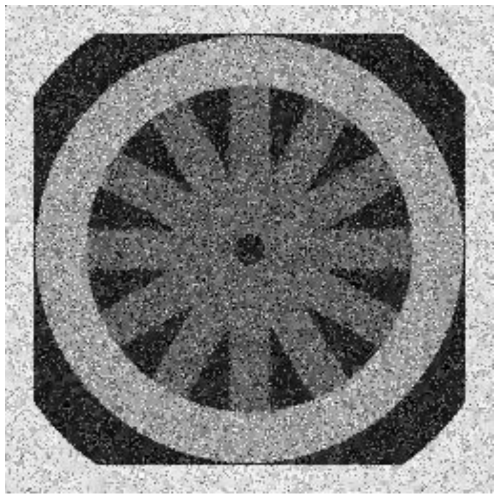
The segmented images have much less noise. But, if observed carefully, the edges have become rough i.e. edges are not preserved.

Exponential Conduction Coefficient

We ran our algorithm using quadratic coefficients for $k = \{20,30\}$. Below are the results shown for it:

K=20


(a) Iteration 5


(b) Iteration 20

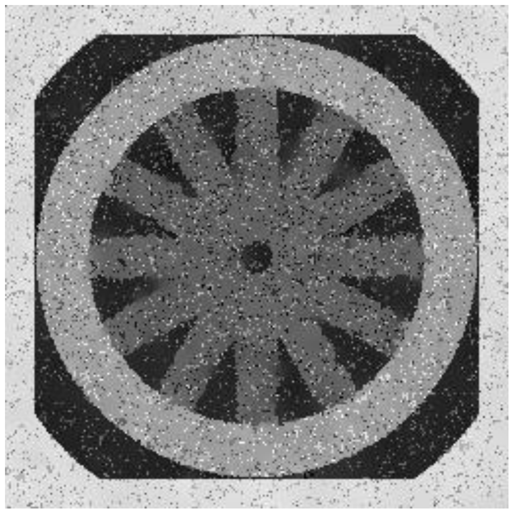
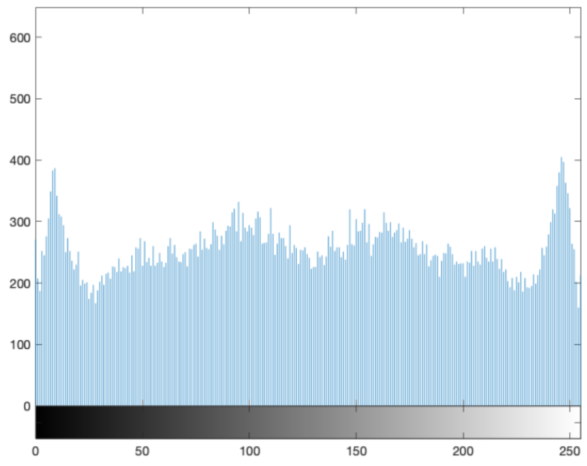
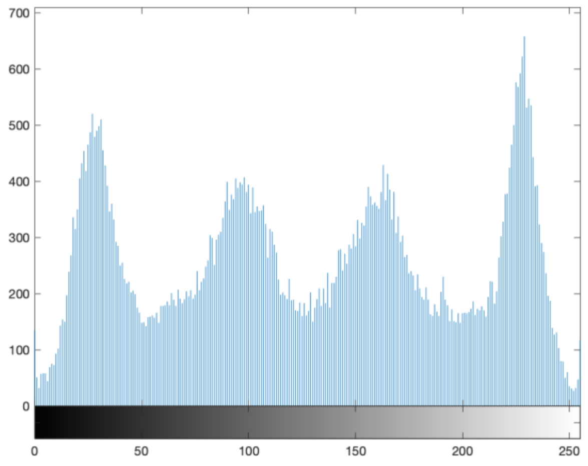

(c) Iteration 100

Figure 24: Images after running anisotropic diffusion with k=20 and g () = exp

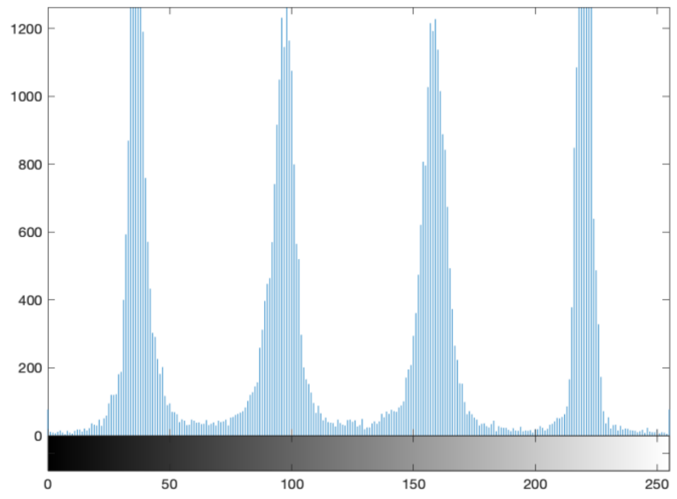
We have changed the conduction coefficient now. Fig 24 shows the images obtained with K=20. The image is getting diffused but not as efficient as the ones obtained with quadratic coefficients.



(a) Iteration 5



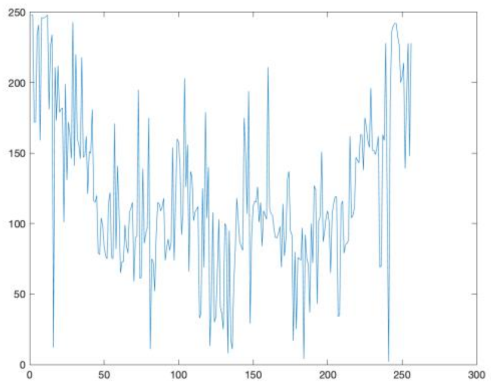
(b) Iteration 20



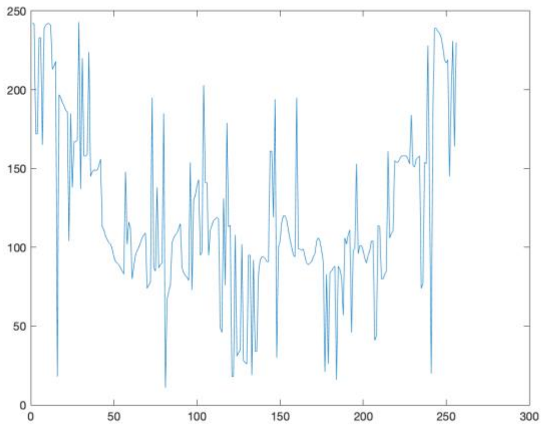
(c) Iteration 100

Figure 25: Corresponding histograms of images shown in Fig 24

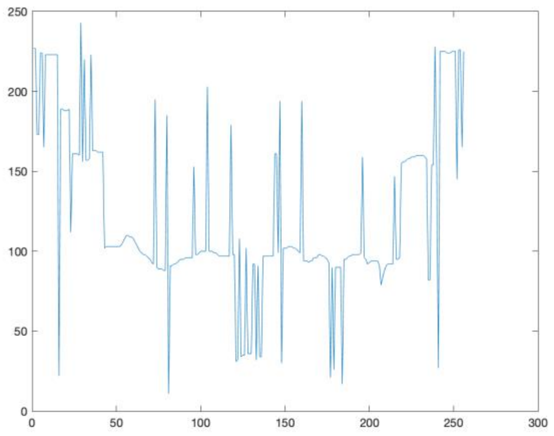
The histograms show separate peaks in the 100th iteration, but they much broader. Also, they form valleys, which shows edges are not preserved.



(a) Iteration 5



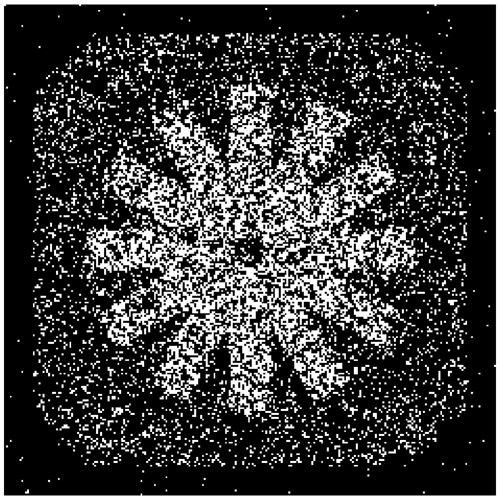
(b) Iteration 20

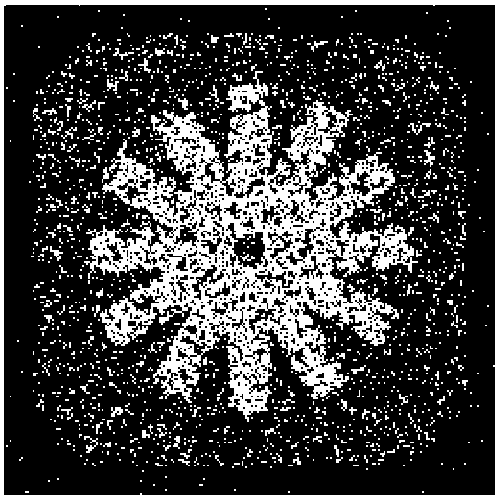


(c) Iteration 100

Figure 26: Corresponding y=128 of images shown in Fig 24

The plot is not at all symmetrical in this case. Thus, the image quality is definitely deteriorated.


(a) Iteration 5


(b) Iteration 20

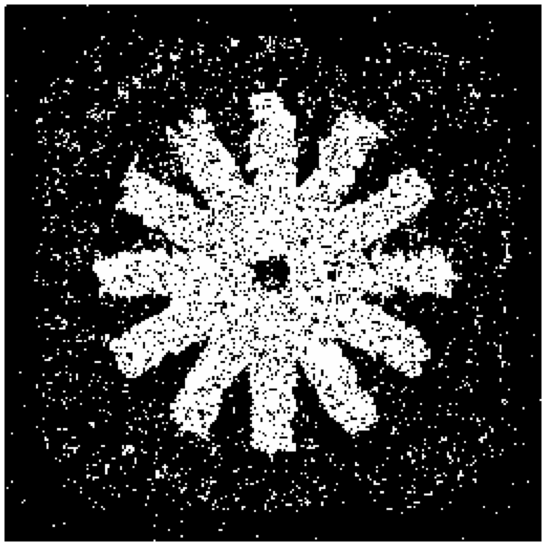
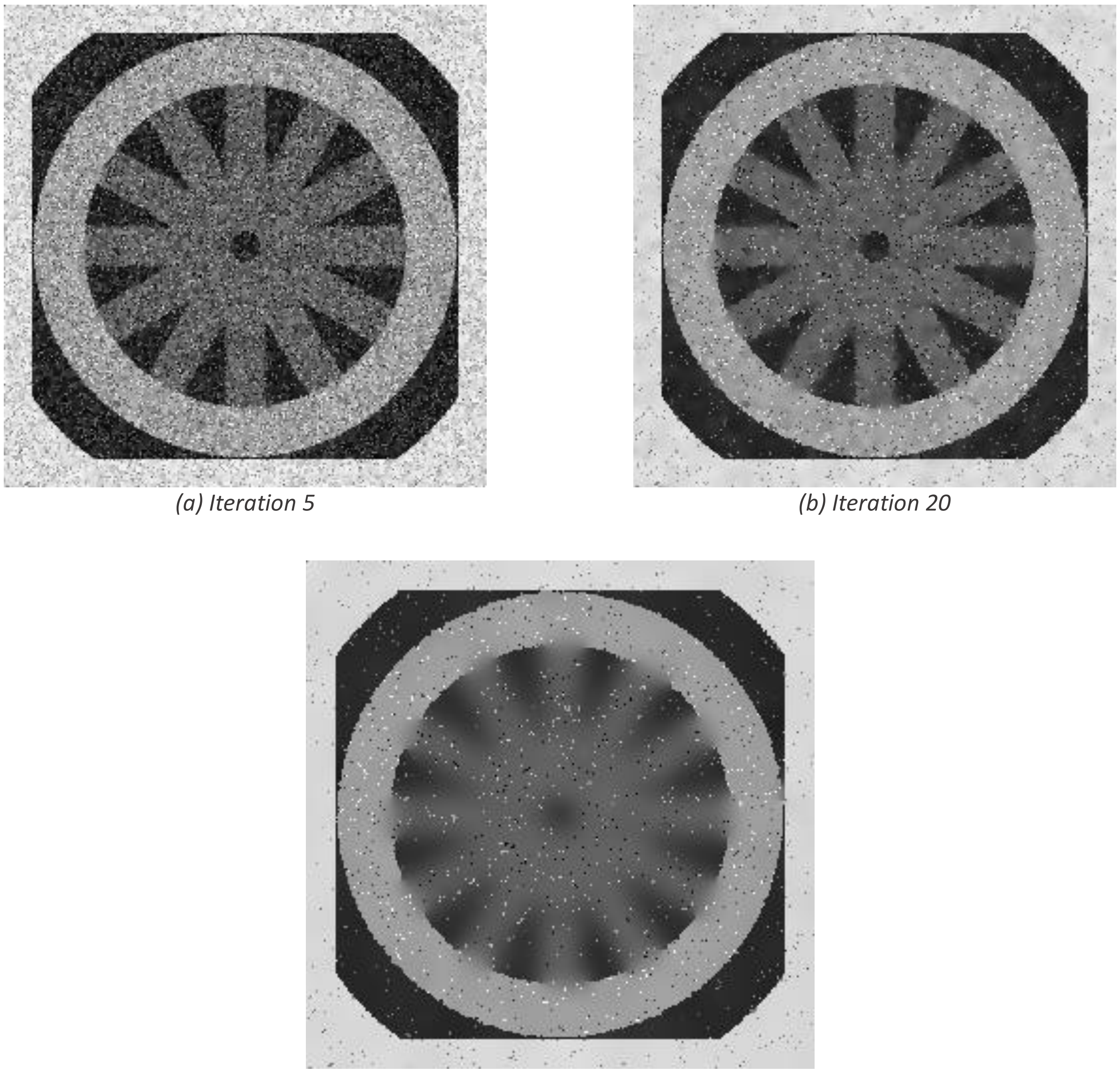

(c) Iteration 100

Figure 26: Segmented Images of the ones shown in Fig 24

The segmented images obtained here as very noisy and there are also no fine edges.

K=30

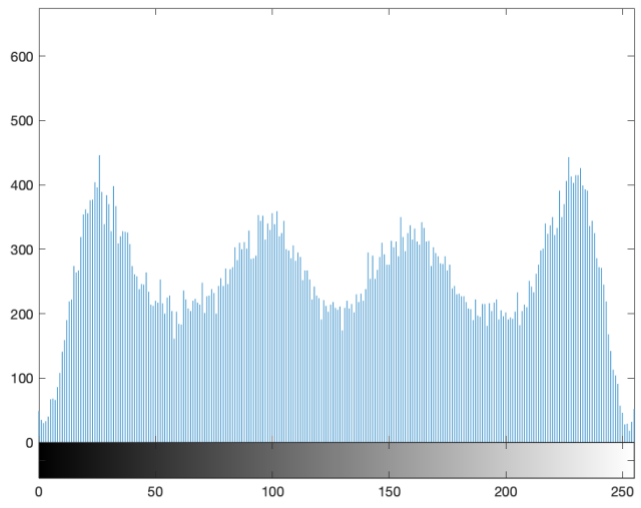

(a) Iteration 5

(b) Iteration 20

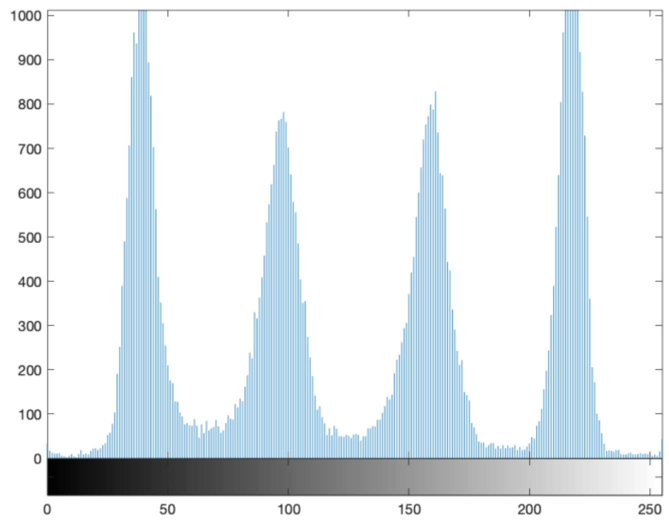
(c) Iteration 100

Figure 27: Images after running anisotropic diffusion with k=30 and g () = exp

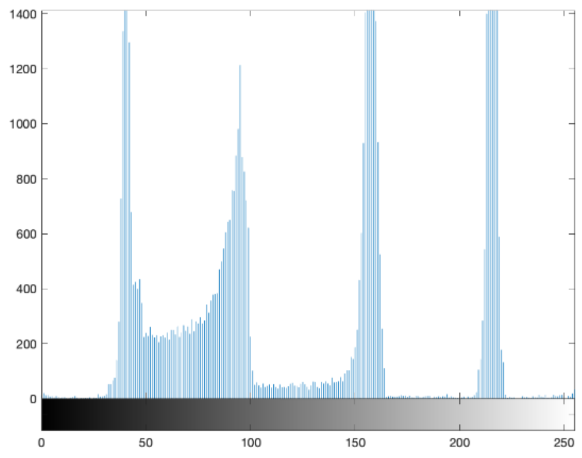
The images obtained with this configuration have deteriorated the results even further. The image has become blurry and there is still lot of noise.



(a) Iteration 5



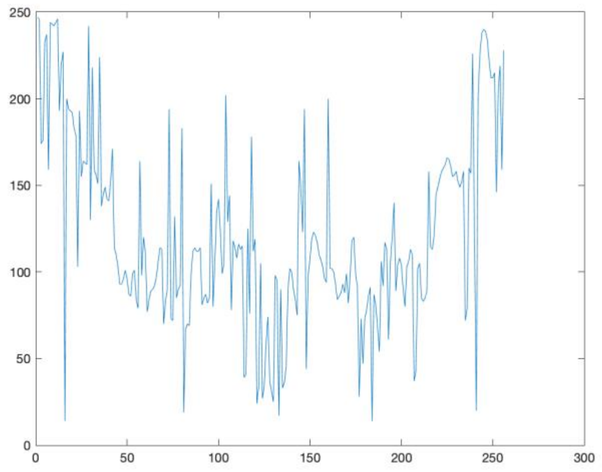
(b) Iteration 20



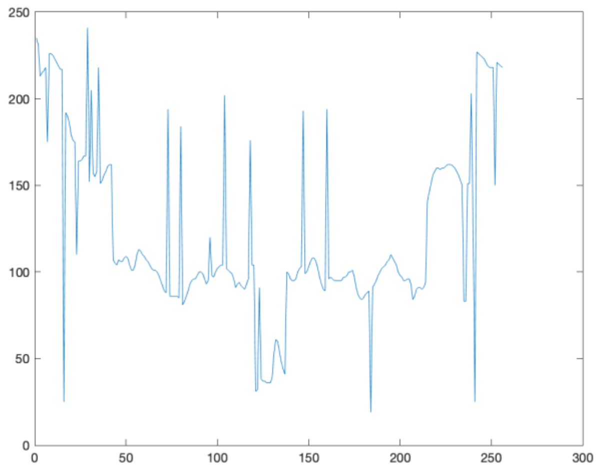
(c) Iteration 100

Figure 28: Corresponding histograms of images shown in Fig 27

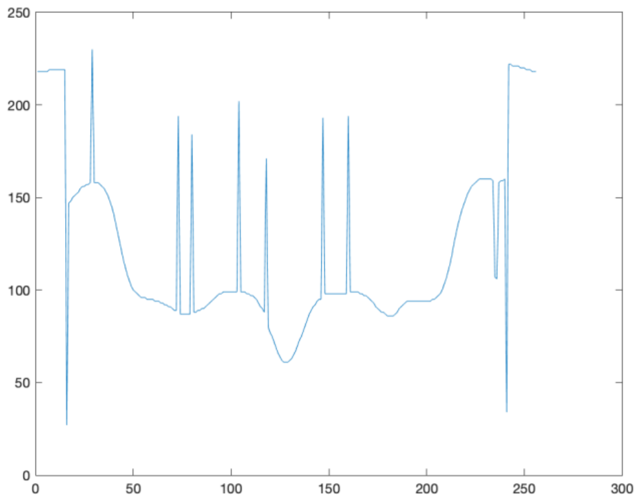
The same conclusion can be derived from the histograms as well. The image has become blurry in the 100th iteration. In Iteration 20th, there is finer peaks and valleys but in the 100th iteration, peaks and valleys have also become distorted. Thus, making the image bad.



(a) Iteration 5



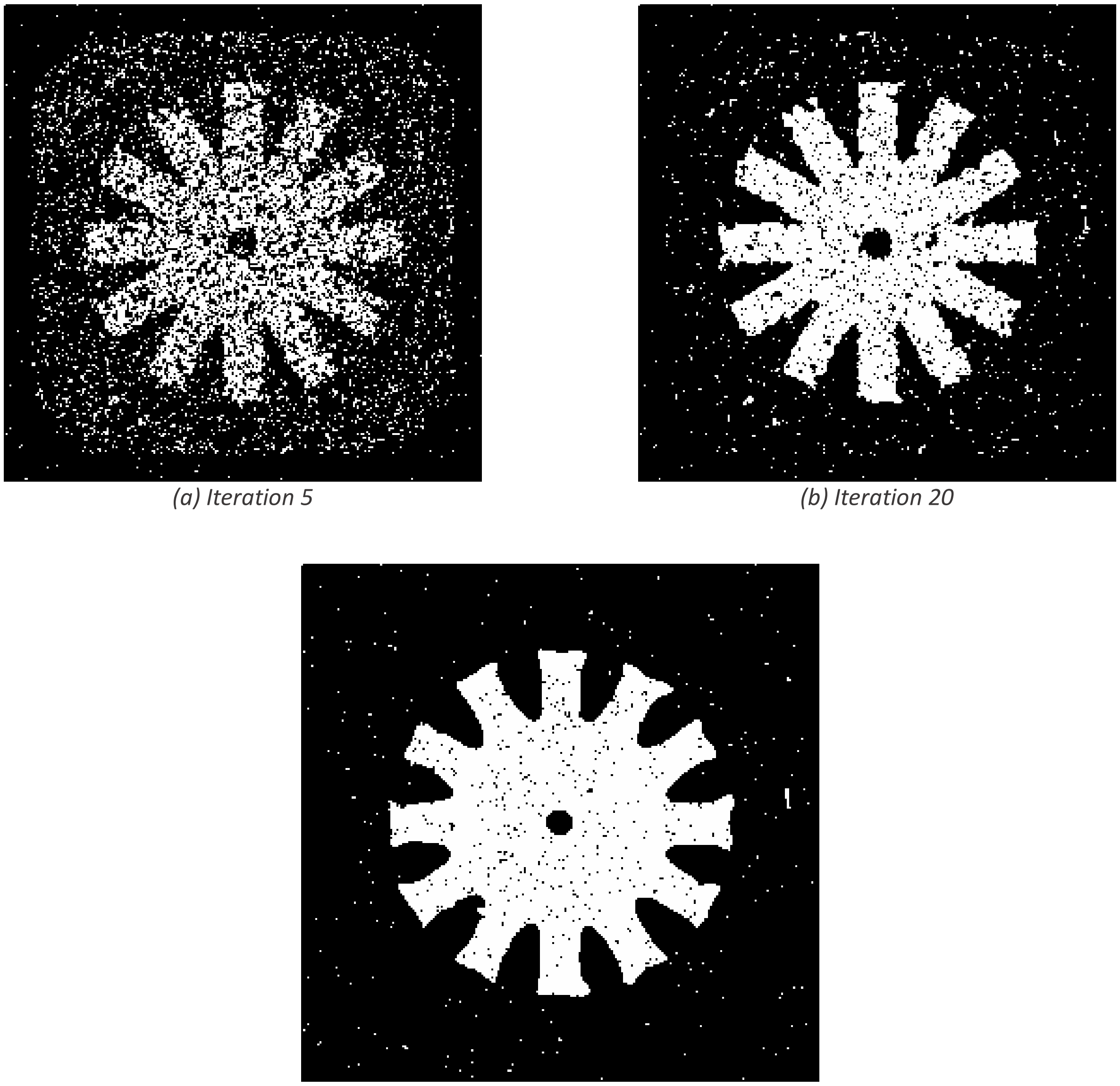
(b) Iteration 20



(c) Iteration 100

Figure 29: Corresponding y=128 of images shown in Fig 27

The plots obtained are also not at all symmetrical.



(a) Iteration 5

(b) Iteration 20

(c) Iteration 100

Figure 30: Segmented Images of the ones shown in Fig 27

It is evident from the segmented images that the edges are lost with this configuration and there is still noise. Thus, it seems like increasing the value of K will deteriorate image further. Also, the image obtained with K=20 is so noisy. So, if we reduce the value of K, the image will be noisier as there will be less diffusion. Thus, the **best results** obtained for this image is when we **used quadratic coefficient with K=8 (Fig. 16)**.

The second part of this 2nd problem is to run anisotropic diffusion on *'cameraman.tif'*. We ran the same algorithm, which was used in part 1, again, for both types of conduction coefficients and different values of k. The original image is shown in Fig 31. Below are shown the images obtained after running the algorithm which is implemented:

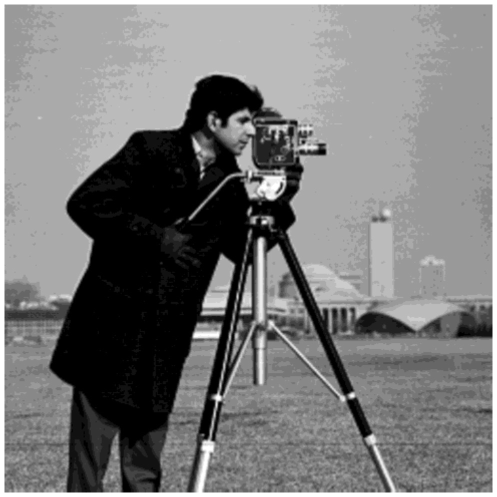
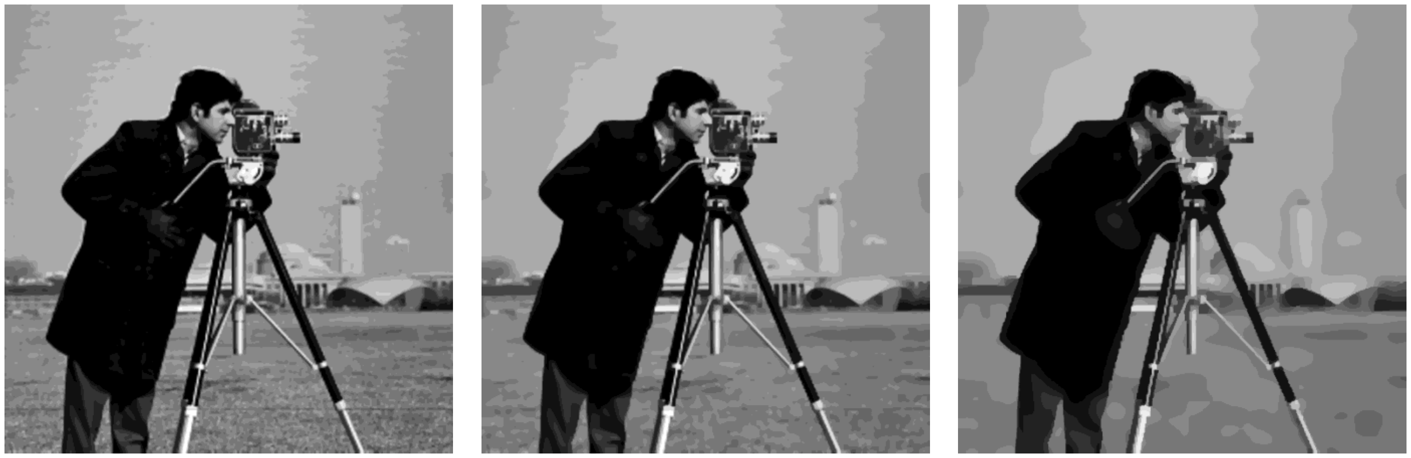


Figure 31: Original Image (cameraman.tif)

Quadratic Conduction Coefficient:

We ran our algorithm using quadratic coefficients for $k = \{7,8,9\}$. Below are the results shown for it:

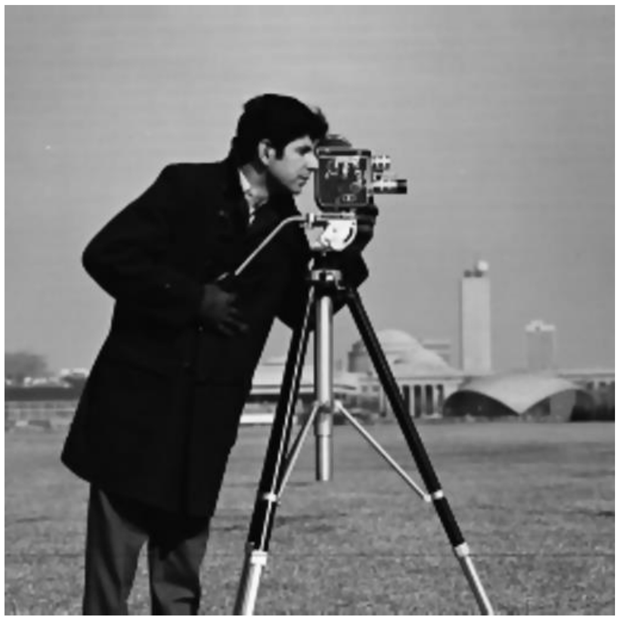
K=7

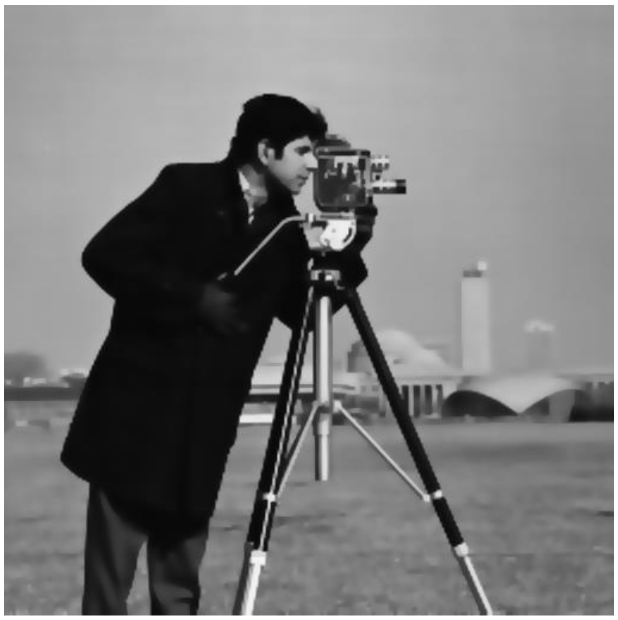


(a) Iteration 5 *(b) Iteration 20* *(c) Iteration 100*

Figure 32: Images after running anisotropic diffusion with k=7 and g () = quadratic

K=8


(a) Iteration 5


(b) Iteration 20

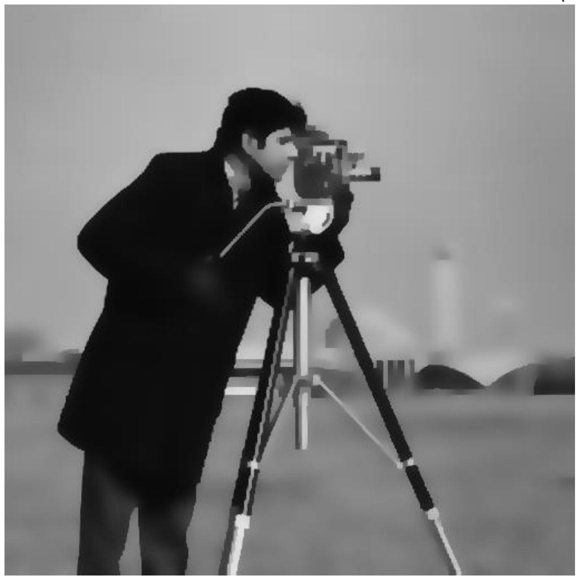
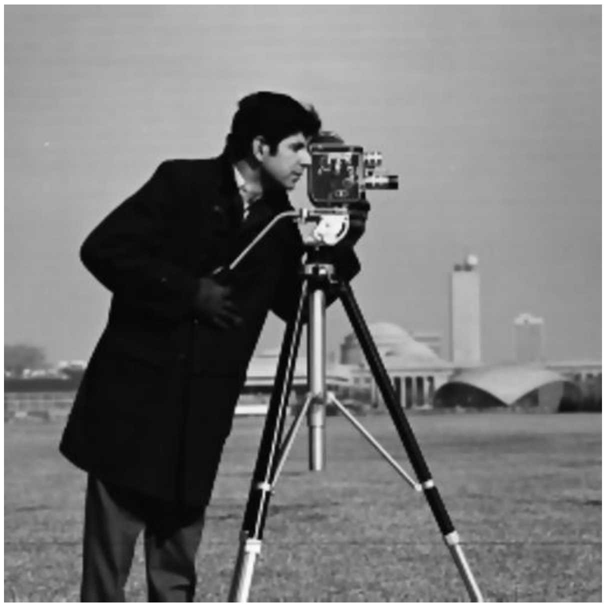
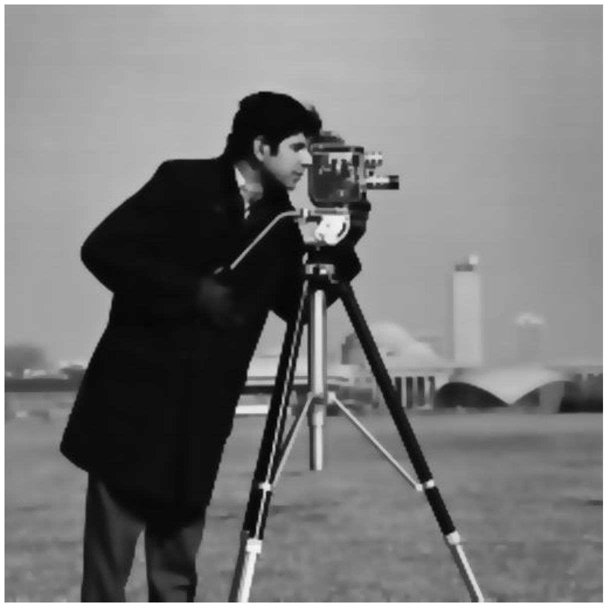

(c) Iteration 100

Figure 33: Images after running anisotropic diffusion with k=8 and g () = quadratic

The image seems to be getting more finer, but if observed carefully it is getting blurred. We are losing the edges.

K=9


(a) Iteration 5


(b) Iteration 20

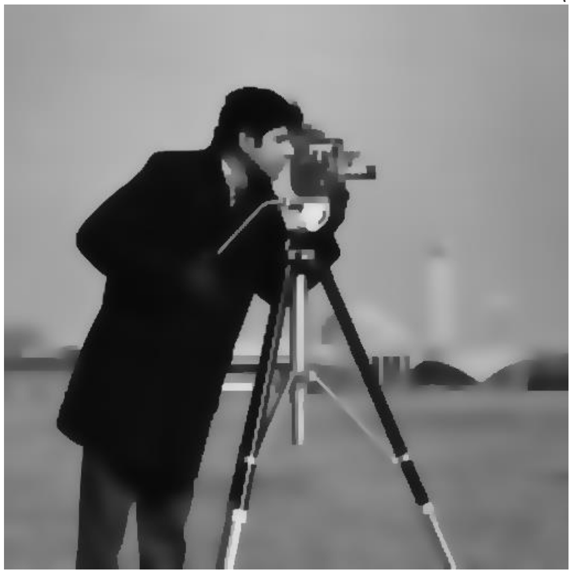

(c) Iteration 100

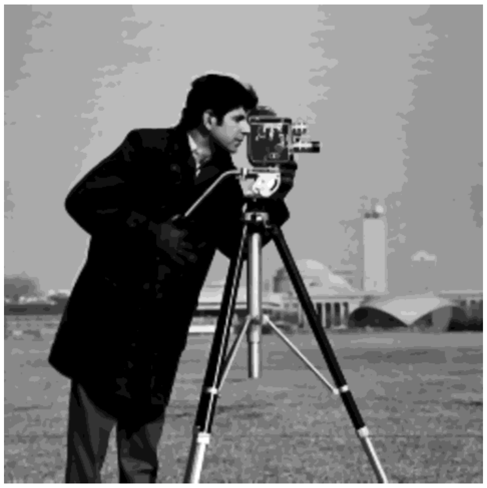
Figure34: Images after running anisotropic diffusion with k=9 and g () = quadratic

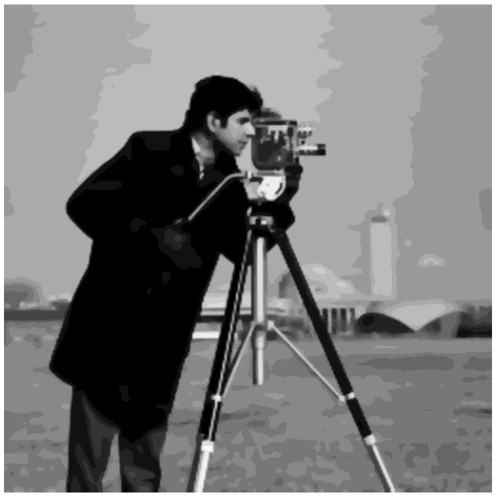
The images have already become more burred. Thus, increasing the value of K with quadratic coefficient will make the image worse. This is because of the choice of quadratic conduction coefficients we chose for the filter, as it’s evident that it favors image with wide region over small region.

Exponential Conduction Coefficients

We ran our algorithm using quadratic coefficients for $k = \{10{,}20{,}30{,}40\}$. Below are the results shown for it:

K=10


(a) Iteration 5


(b) Iteration 20

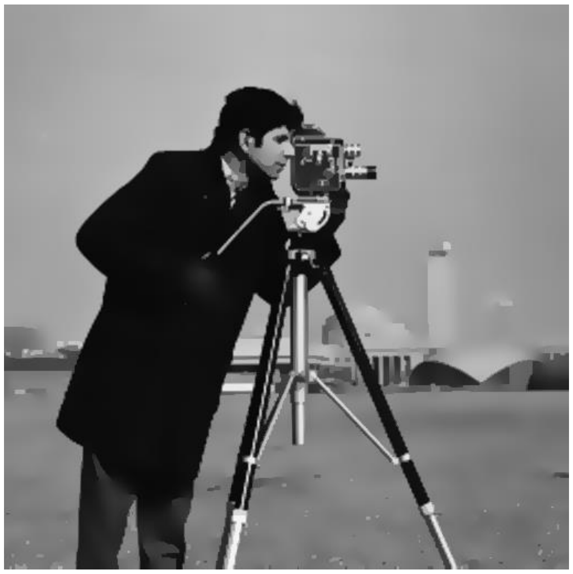
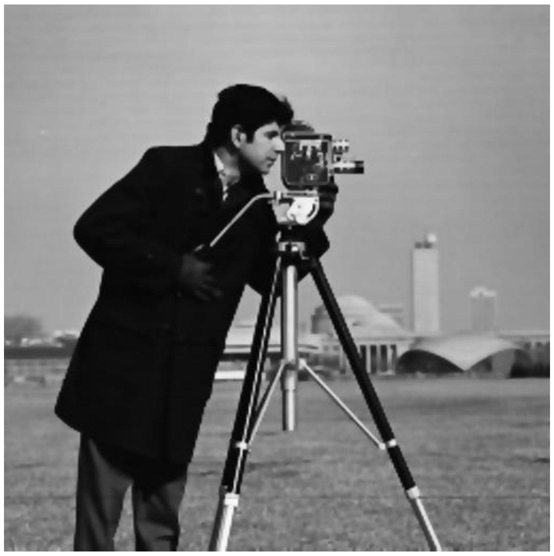
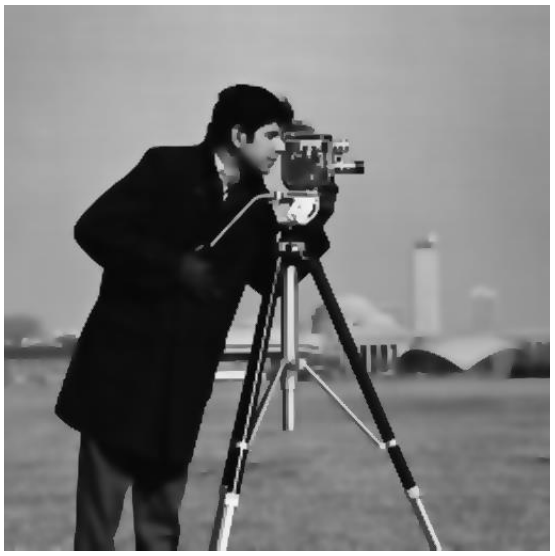

(c) Iteration 100

Figure 35: Images after running anisotropic diffusion with k=10 and g () = exp

Image is finer and smoother than the original one. However, in the 100th iteration, image is getting blurred as well.

K=20


(a) Iteration 5


(b) Iteration 20

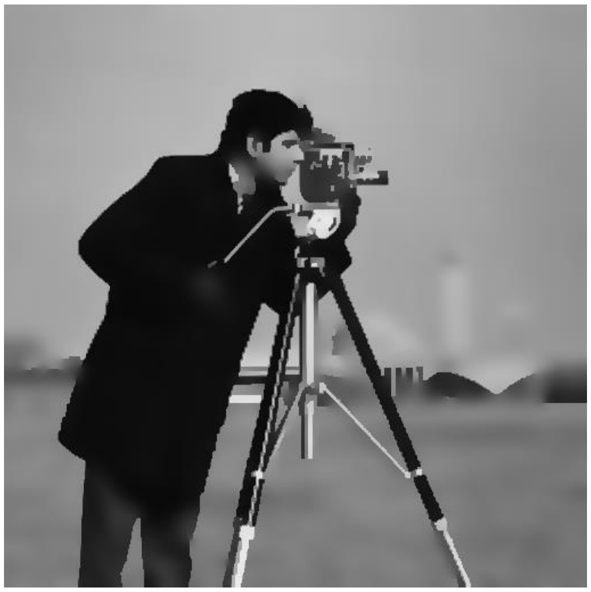
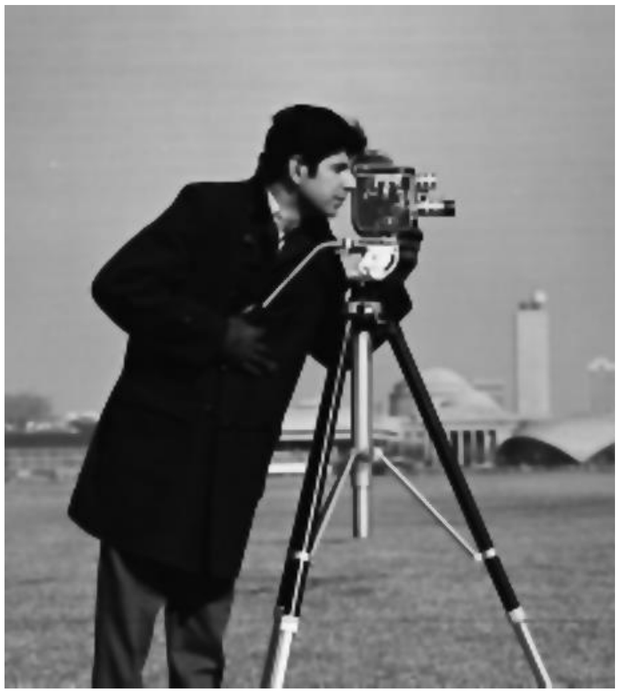
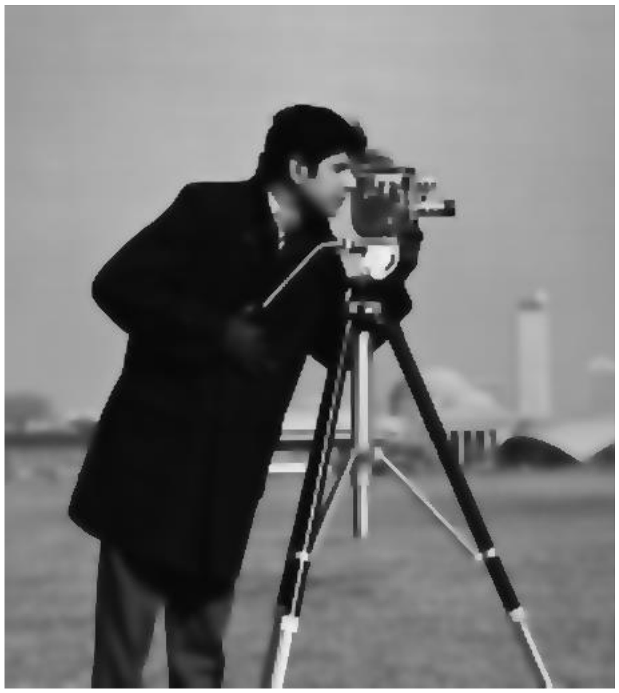

(c) Iteration 100

Figure36: Images after running anisotropic diffusion with k=20 and g()= exp

Again, image is finer and smoother. Edges also are finer. Since, this image was not noisy, it is not necessary to use 100 iterations for this image.

K=30


(a) Iteration 5


(b) Iteration 20

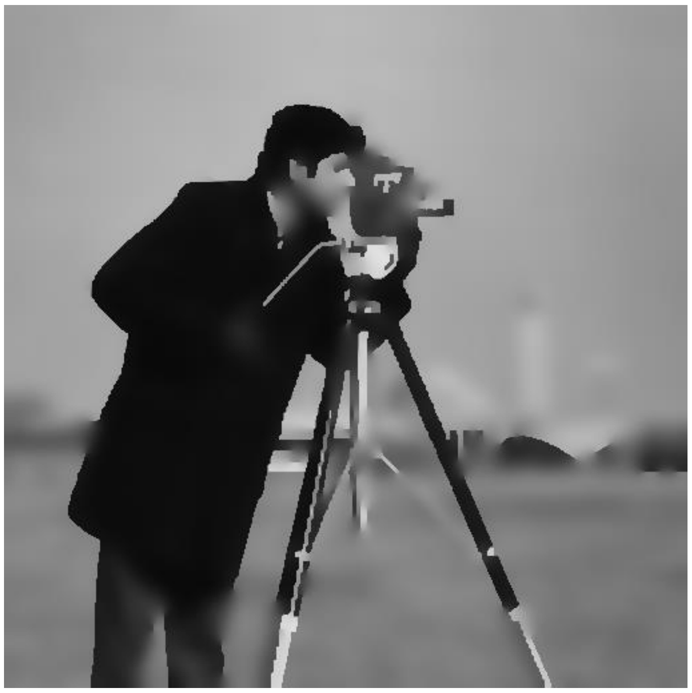
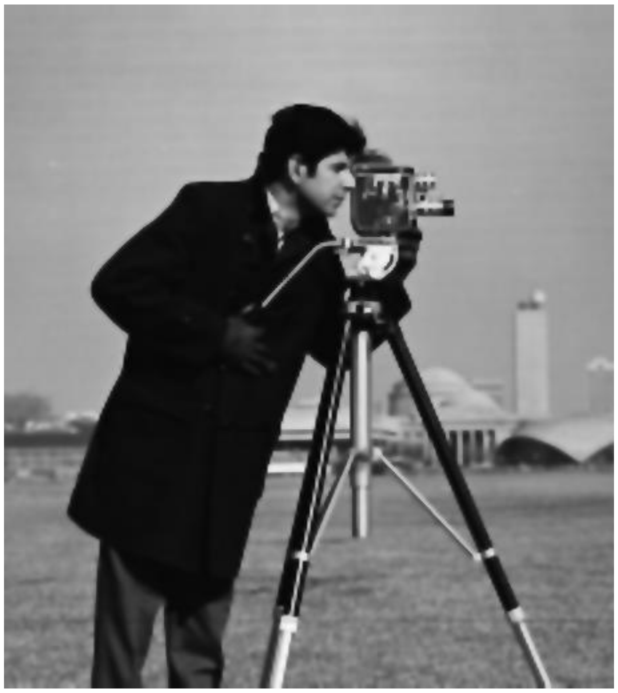
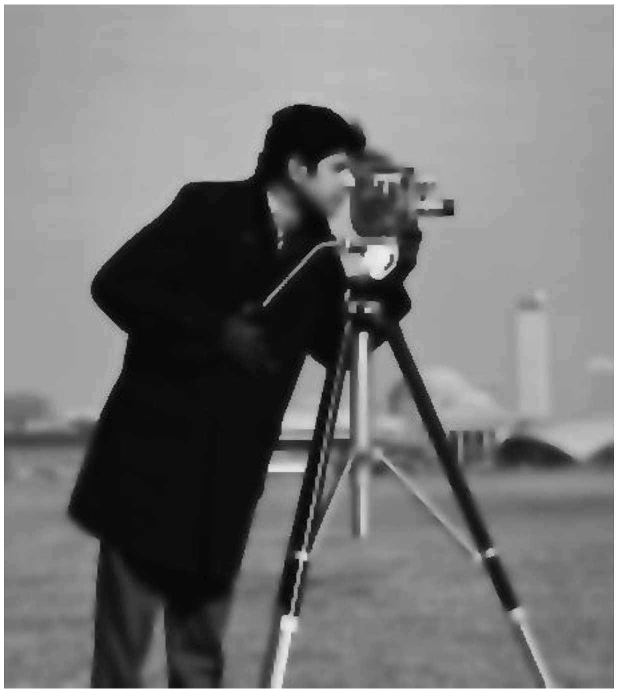

(c) Iteration 100

Figure37: Images after running anisotropic diffusion with k=30 and g () = exp

K=40


(a) Iteration 5


(b) Iteration 20

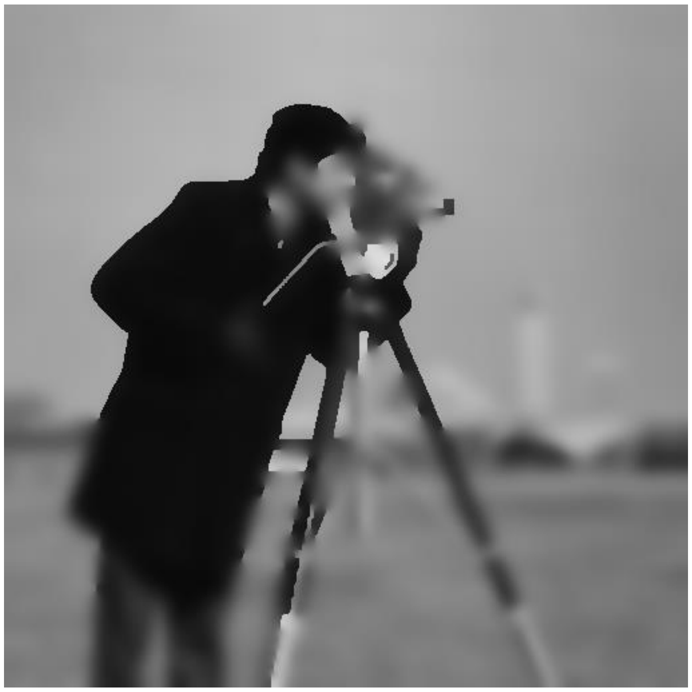

(c) Iteration 100

Figure38: Images after running anisotropic diffusion with k=40 and g () = exp

All the images reflect the same opinion in this part. It seems like the best image I obtained with **Exponential Conduction Coefficients, K=30 and in the 5th iteration Fig. 37(a)**.

From the results obtained in this section, we observe that as the iterations increase, the noise gets even more diffused making it blurry. We can identify that the noise diffusion is based on how we scale the value of K. The choice of iterations depends on the amount of noise in the image. Noisier the image, more the iterations. The higher number of iterations in less noisy image will make the image blurred and loose its edges.

We also observed that the $chwheelnoise\ image$ works better with quadratic coefficient and lower values of K. However, the $cameraman\ image$ works better with exponential coefficients and larger values of K. The only difference is in the structure of these images. Hence, we can conclude images with wider regions are favored with quadratic coefficients and the images with high contrast edges are favored with exponential coefficients. The choice of $g(\cdot)$ also reflects the changes in segmented images. We saw that spokes are better visible with each iteration, but when exponential coefficient was used, the segmented image was noisier than the ones with the use of quadratic coefficient.

Anisotropic diffusion works differently for both images, mainly because of the features within the image such as wide or narrow regions.

For '$cwheelnoise$' $image$, the diffusion works best lower value of K=8 using quadratic coefficient, as shown in Fig 16. Diffusion is taking place using exponential coefficient as well. But it is not as efficient as the quadratic ones. On the other hand, for 'cameraman' image, which has rather narrow regions, we observed that the diffusion worked better with the higher value of K= 30 using exponential conductive coefficient. When we applied quadratic coefficient, image got blurrier with increase in the value of K (See Fig 34 and Fig. 35).

D. Conclusion

From the results we have obtained in this project, we learned that non-linear filters can clean out salt and pepper noise as per the expectations, and also help in preserving edges. We also learned that anisotropic diffusion takes more time and iterations to achieve the same results compared to nonlinear filters. However, it can produce different results by changing the value of k and choosing different type of conduction coefficient.

The choice of conduction coefficient in anisotropic diffusion can be finalized by observing the image. The images with wider regions work better with quadratic coefficient and lower values of K. However, images with narrow regions work better with exponential coefficient and larger values of K (K>10).

Anisotropic diffusion also works different based on low or high iterations. For low iterations, it performs better with less noisy image whereas for higher iterations, the noise diffuses causing the image to blur. Considering the two types of conductive coefficient, the exponential function preserves the edges whereas the quadratic function preserves the large blobs in the image.

As we iterate over greater K values, both functions tend to smear the image and for higher iterations, the image gets blurred.