DIP Homework Assignment #1

姓名：鍾毓安

學號：B01902040

系級：資訊四

Email: [iamyuanchung@gmail.com](mailto:iamyuanchung@gmail.com)

**Source Code**

The following script and functions were implemented and their introductions were described as follows. As for more detailed descriptions, please refer to each corresponding function file.

1. README.m: This is the main script that works like the main function. All the required tasks, including Warm-Up, Problem 1 & 2 will be completed one by one when README.m is executed.
2. flipVertical.m: The function flips the given 2D image matrix up-side-down.
3. flipHorizontal.m: The function reverses the given 2D image matrix left-to-right.
4. plotHistogram.m: For a given 2D gray-scale image matrix, plotHistogram returns a 256-length 1D array where the i-th entry stores the number of pixels whose value equals to (i – 1).
5. histEqual.m: The function performs the histogram equalization on the given 2D image matrix. Histogram equalization enhances the given image G by first converting the histogram of G into cumulative distribution function (CDF), then mapping the CDF to a uniformly distributed one so as to make the histogram of the enhanced G more uniformly distributed.
6. localHistEqual.m: The function performs the local histogram equalization on the given 2D image matrix. Similar to histogram equalization, local histogram equalization also tries to enhance the given image using the histogram. However, local histogram equalization introduces an extra window that will go through the image from top to bottom and from left to right. Then, histogram equalization is applied to the region inside the window, and this is how the word “local” comes from. The original histogram equalization is sometimes referred to as the global histogram equalization to make it distinguishable from the local version.
7. logTransform.m: The function performs log transform on the given 2D image matrix. Log transform enhances the low intensity pixels due to its property of concave downward. Especially, for each entry of an given image matrix , where is already scaled to range by dividing 255, the log transform does the following transformation:

,

where is the scaling constant that ensures the resulting image has a maximum magnitude of 255.  
(Reference: http://homepages.inf.ed.ac.uk/rbf/HIPR2/pixlog.htm)

1. invLogTransform.m: The function performs inverse log transform on the given 2D image matrix. Different from log transform, the concave upward property makes the inverse log transform more useful in a situation when more details of high intensity pixels are desired. Especially, for each entry of an given image matrix , where is already scaled to range by dividing 255, the inverse log transform does the following transformation:

,

where is the scaling constant that ensures the resulting image has a maximum magnitude of 255.

1. powerLawTransform.m: The function performs the power-law transform on the given 2D image matrix. For each entry of an given image matrix , where is already scaled to range by dividing 255, the power-law transform does the following transformation:

,

where is the scaling constant that ensures the resulting image has a maximum magnitude of 255, and is a parameter that can be flexibly controlled. When , ’s property of concave downward makes it a suitable choice for enhancing image with low intensity pixels; when , becomes concave upward, which is a good option for enhancing image with high intensity pixels; and when , simply performs the linear mapping.  
(Reference: http://funnotes.net/tofpages/TopicOfFortnight.php?tofTpcFl=topicoffortnight22)

1. addGaussianNoise.m: The function adds the Gaussian (Normal) distributed noise to the given 2D image matrix. The formula is as follows:

,

where is the amplitude of the generated Gaussian noise, and are the mean and variance for Gaussian distribution, respectively.

1. addSaltPepperNoise.m: The function adds the Salt and Pepper noise to the given 2D image matrix, where salt means a totally white (255) pixel and pepper means a totally black (0) pixel. For each entry in the given 2D image matrix , the salt and pepper noise is generated and added as follows:

,

where is a parameter of probability. When is small, the outcome image tends to be similar with the clean image; as grows larger, the outcome becomes noisier.

1. calcPSNR.m: The function calculates the Peak signal-to-noise ratio (PSNR) between two gray-scale 2D image matrix and of same shape. The formula for calculating PSNR is as follows:

.

The larger is, the more similar and are.

1. lowPassFilter.m: There are usually two types of noise, the first one is uniform noise, including additive uniform noise and Gaussian noise, and the second one is impulse noise, including salt and pepper noise. Low-pass filter is designed to remove the uniform noise. Especially, low-pass filter introduces a mask , which has the general form as follows (3x3 example):

,

The noise removal procedure is done by convolving with the noisy image. During my implementation, I focus on of size 3x3 and adjust to achieve the best outcome.

1. outlierDetection.m: For impulse noise, outlier detection and median filter are designed to remove the noise. I implement the function by simply following the course slides. The threshold is tuned to locate the best one.
2. myMedianFilter.m: Since there’s a built-in function named medianFilter.m in Matlab, I use myMedianFilter.m for my implementation. The square median filter is implemented and the method doesn’t need any other controlling parameters.

**Warm-Up: Simple Manipulation**

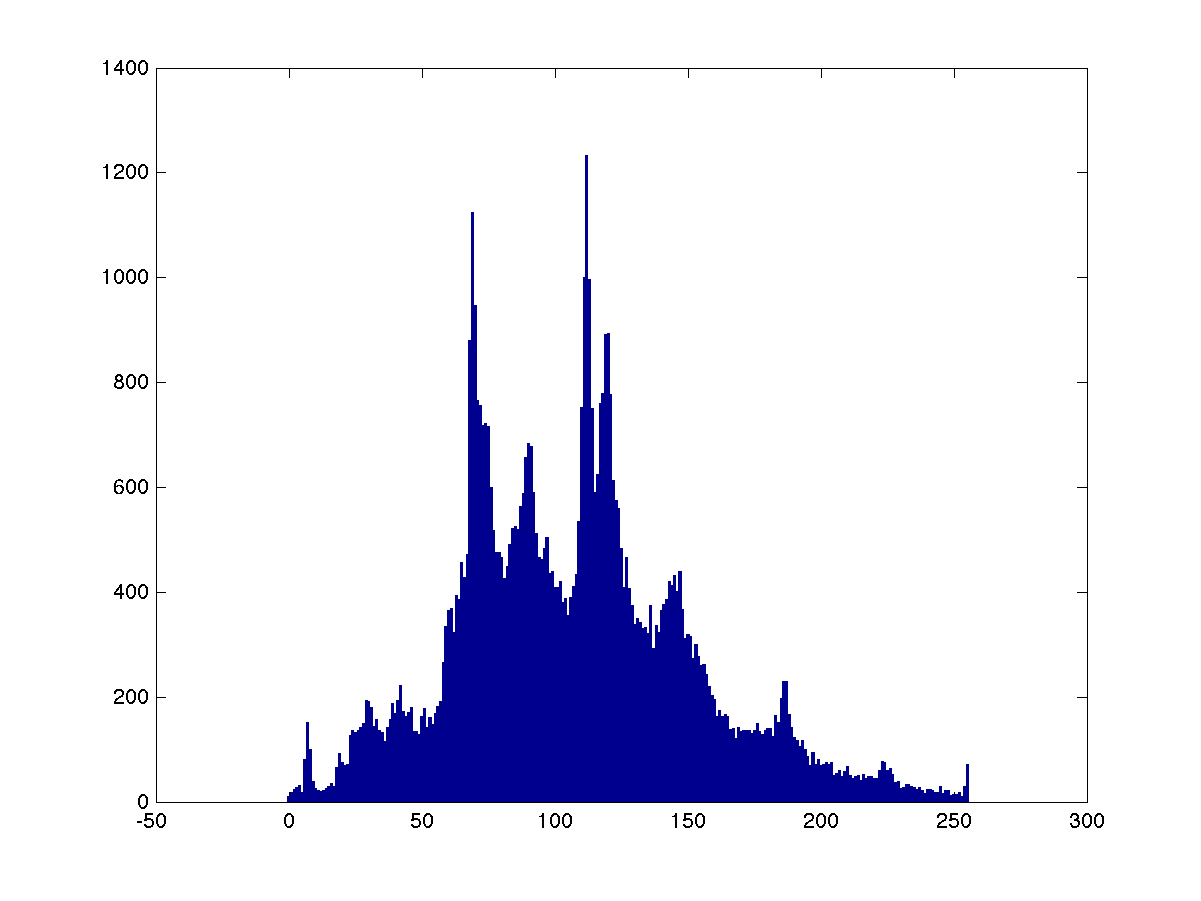
Flipped I vertically and horizontally. The resultant images were displayed as follows.

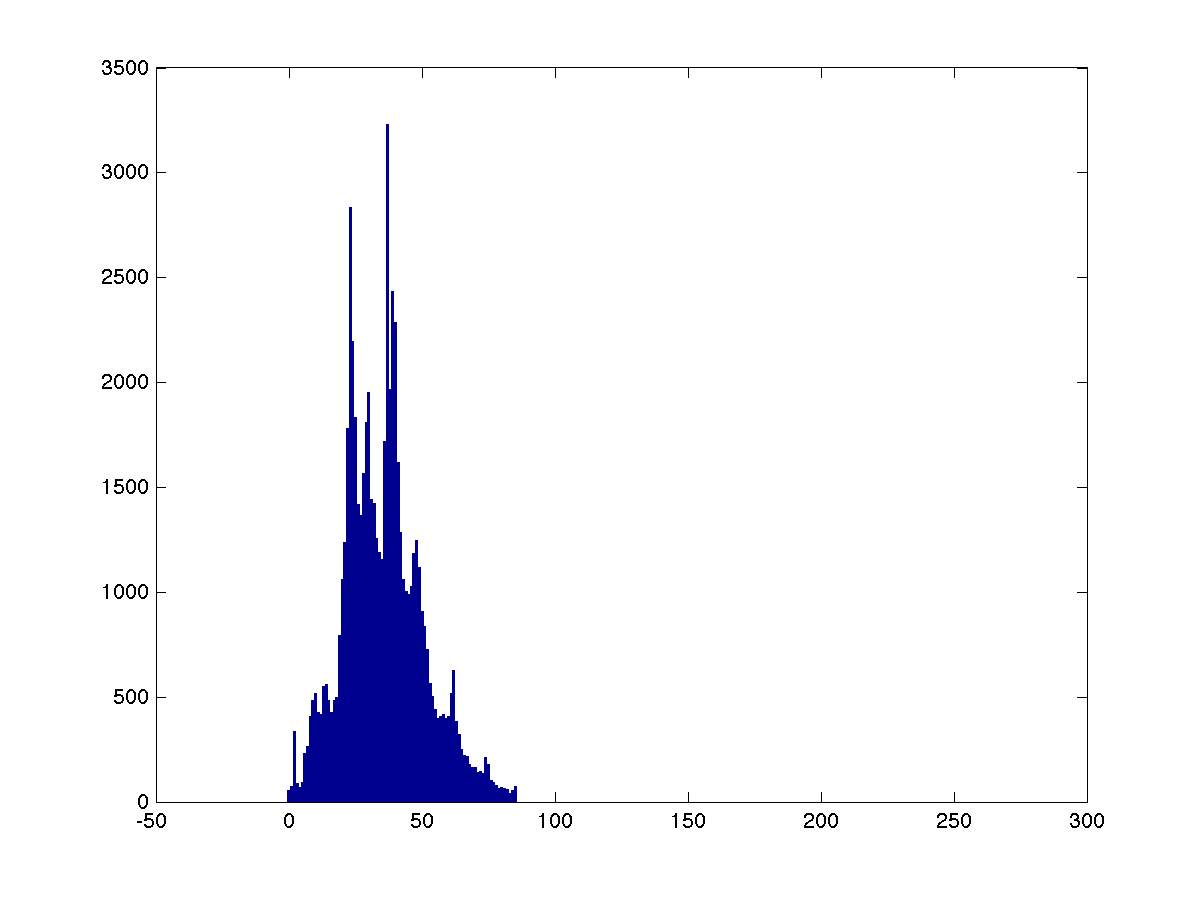
sample1.vertical.png: Flipped I vertically sample1.horizontal.png: Flipped I horizontally

**Problem 1: Image Enhancement**

1. Plot the histograms of I and D. What can you observe from these two histograms? What can you do to make D look like I?



sample1.hist.png: The histogram of image I



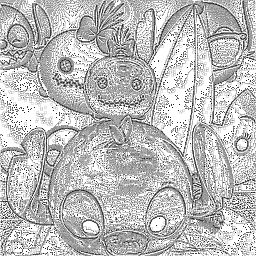
sample2.hist.png: The histogram of image D

1. Perform histogram equalization on D and out the result as H.



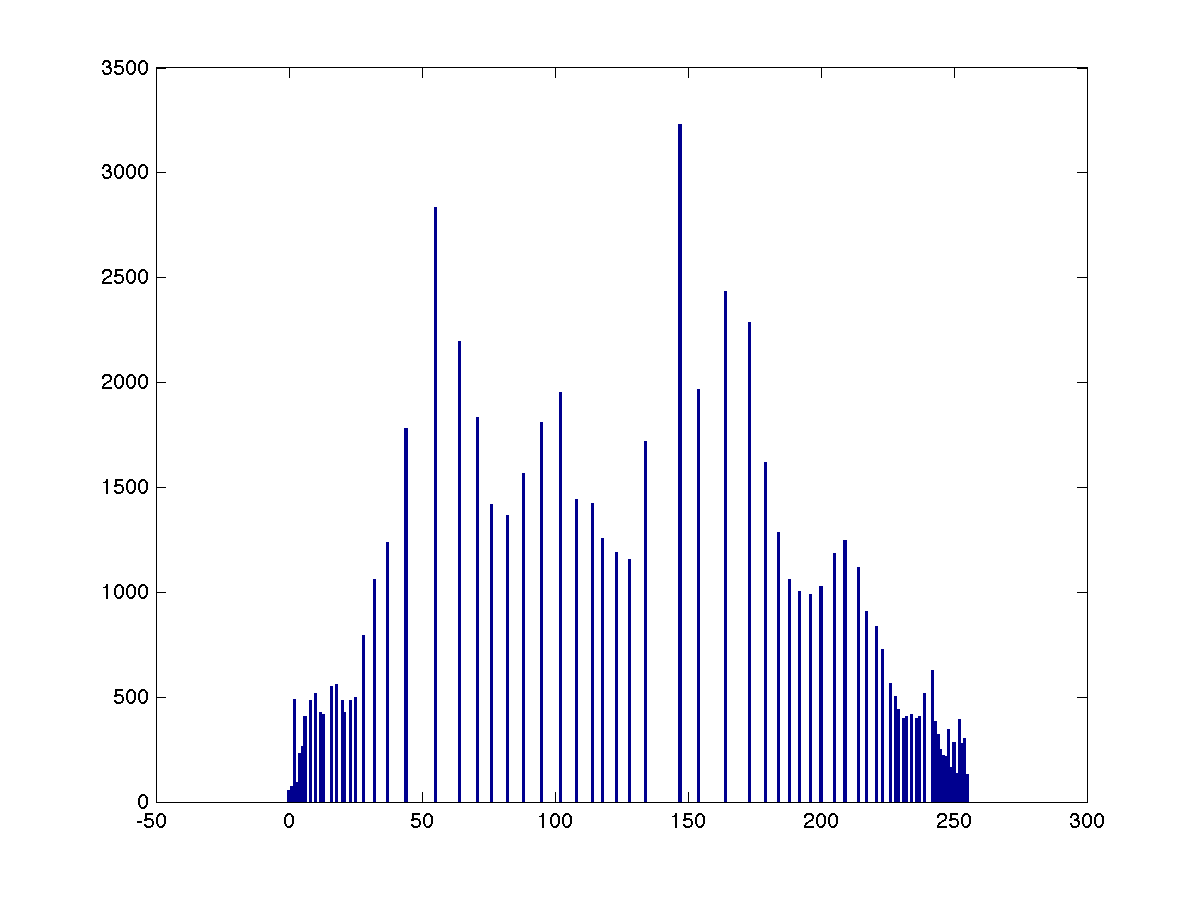
sample2.hist.equal.png: The histogram equalized image D

1. Perform local histogram equalization on image D and output the result as L.

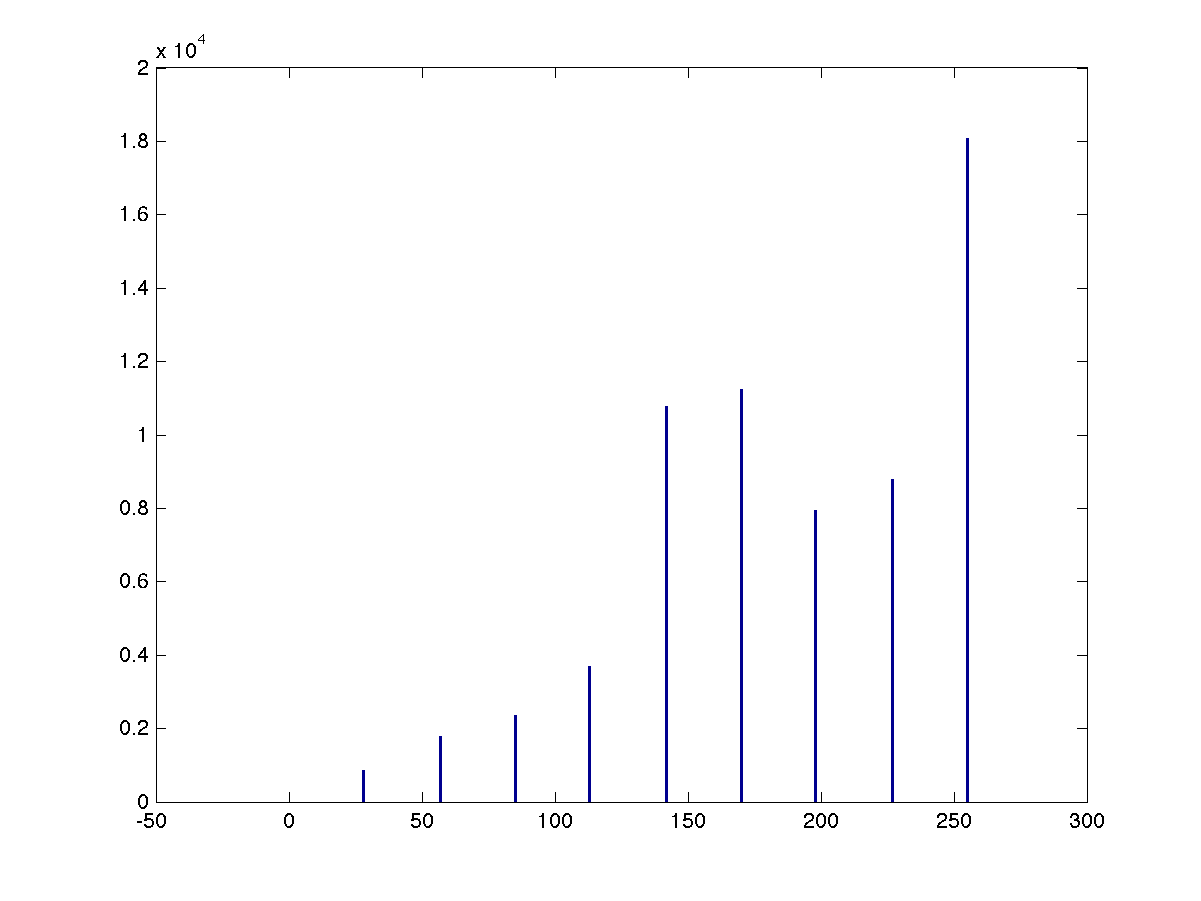


sample2.local.hist.equal.png: The local histogram equalized image D

1. Plot the histograms of H and L. What’s the main difference between local and global histogram equalization?



sample2.hist.equal.hist.png: The histogram of the histogram equalized image D



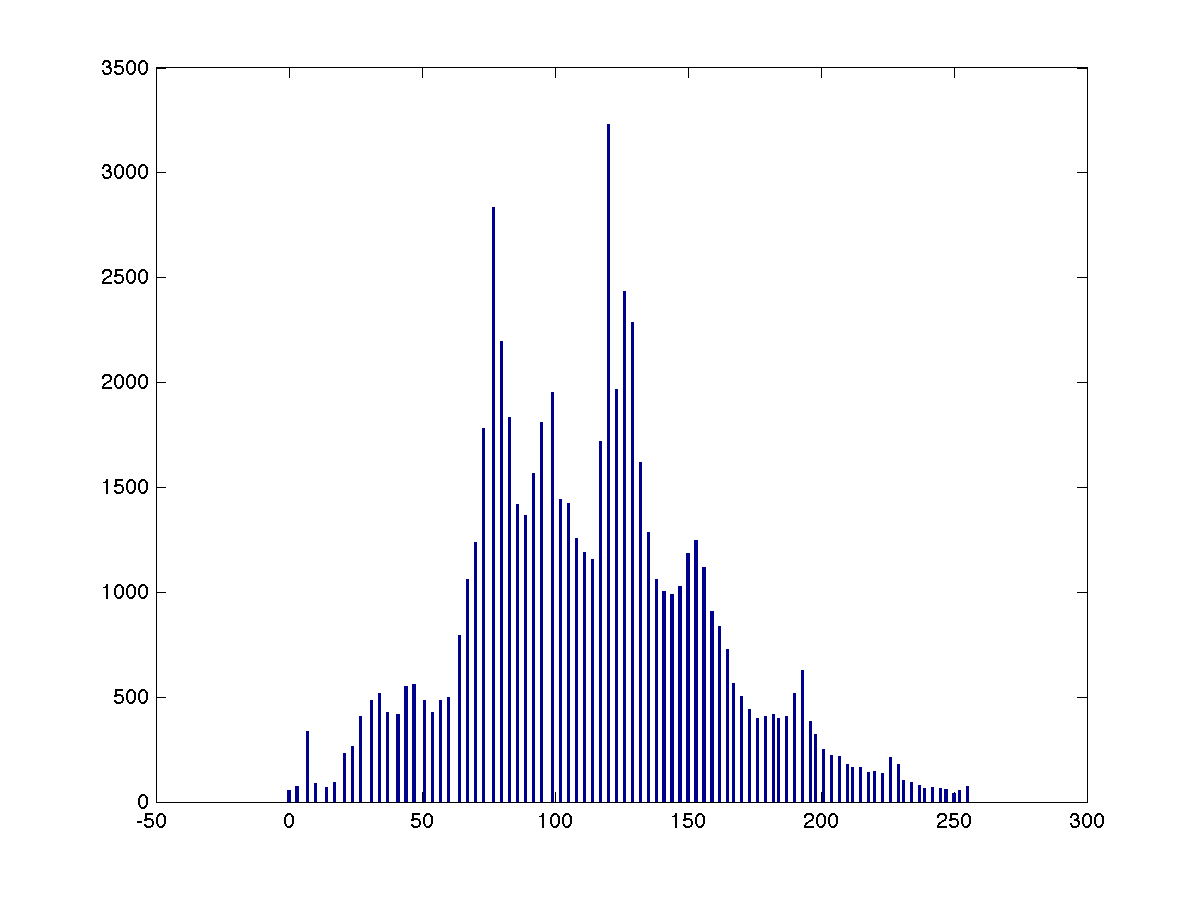
sample2.local.hist.equal.hist.png: The histogram of the local histogram equalized image D

1. Perform the log transform, inverse log transform and power-law transform to enhance image D. Please adjust the parameters as best as you can. Show the parameters, output images and corresponding histograms. Provide some discussions on the results as well.

**Log transform**



sample2.log.png: The log transformed image D

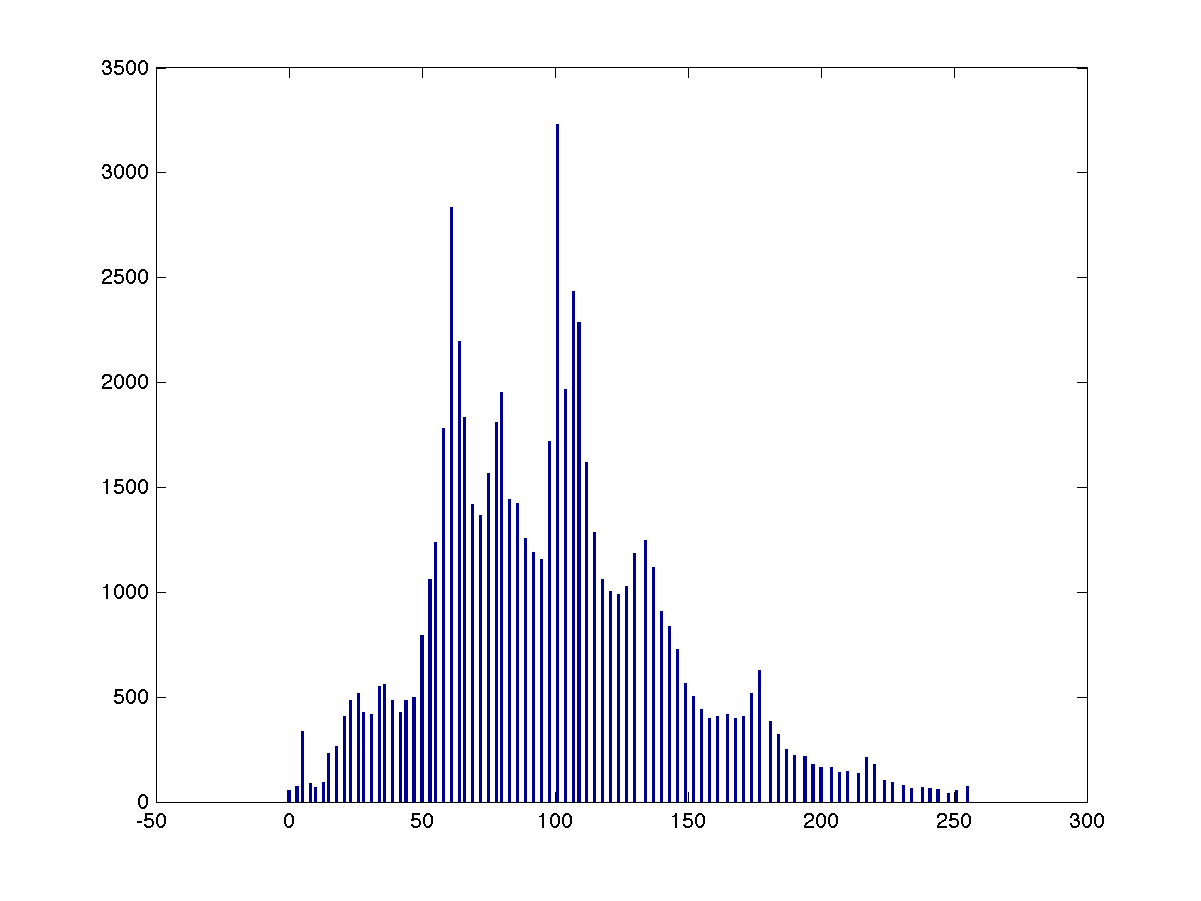


sample2.log.hist.png: The histogram of the log transformed image D

**Inverse log transform**



sample2.inv.log.png: The inverse log transformed image D



sample2.inv.log.hist.png: The histogram of the inverse log transformed image D

**Power-law transform**

**Problem 2: Noise Removal**

1. Add the same kind of noise as in sample3.raw to image I and denote the result as .

Since sample3.raw features sparsely occurring white and black [pixels](https://en.wikipedia.org/wiki/Pixel), I infer that sample3.raw is corrupted by Salt and Pepper noise. To add salt and pepper noise to image I (sample1.raw), we first need to specify , the larger the is, the larger extent of corruption of the resulting image will exhibit. Here I set , which I think is the closest to sample3.raw. The original image I and the corrupted version, denoted as , are displayed as follows, respectively.

sample1.png sample1.salt.pepper.png: Image I with salt and pepper noise () added

1. Add the same kind of noise as in sample4.raw to image I and the output is denoted as .

Since sample4.raw features a uniform corruption, I infer that sample4.raw is corrupted by Gaussian noise. I generated the corrupted image by specifying and the amplitude . The original image I and the corrupted version, denoted as are displayed as follows, respectively.

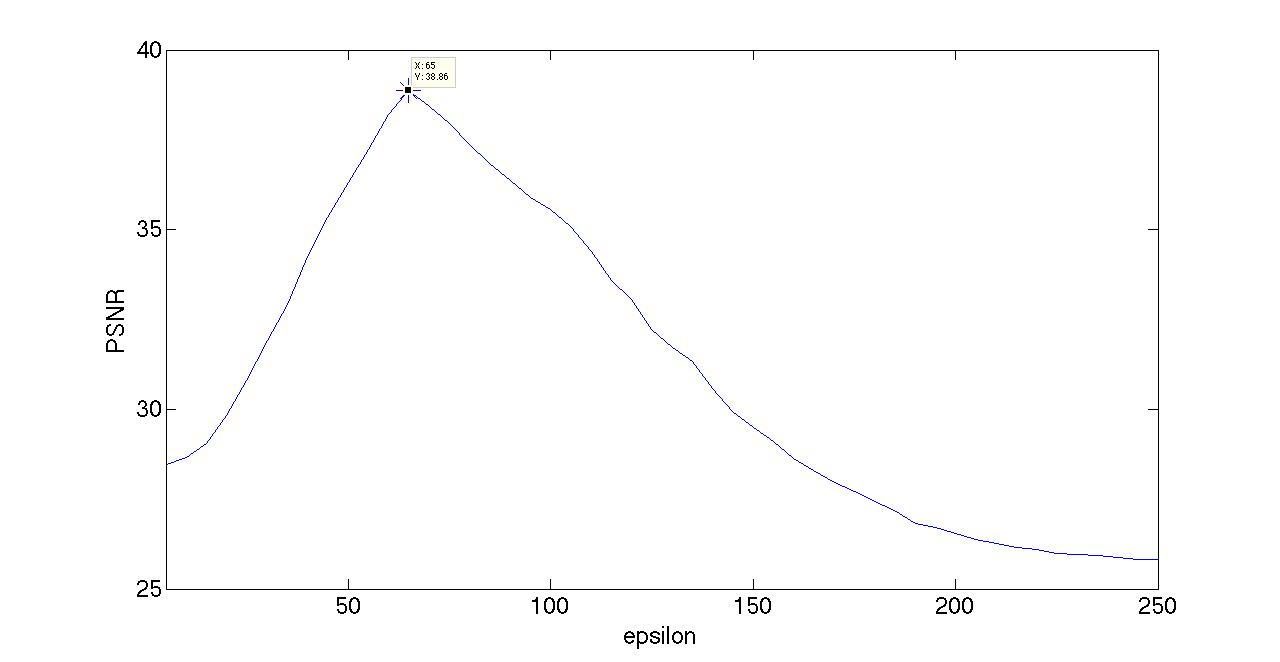
 

sample1.png sample1.gaussian.png: Image I with Gaussian noise () added

1. Choose proper filters and parameters to remove the noise in and , and denote the resultant images as and , respectively. Please specify the steps of your de-noise process and provide some discussions about the reason why those filters and parameters are chosen.

Since is contaminated by salt and pepper noise, which belongs to the impulse noise, median filter and outlier detection are two methods worthy trying. We will discuss them one by one.

For **outlier detection**, there’s a threshold that decides how large the gap between a certain pixel value with the average of its eight neighbors is that will be considered as an outlier. Unfortunately, such is usually hard to choose. Therefore, I decide to plot a curve, where the x-axis is and the y-axis is the corresponding PSNR value, and the that achieves the highest PSNR value will be selected as the best threshold . The curve is displayed as follows, and the range of is with sample rate 5.



outlier.psnr.png: The relationship between and the achieved PSNR value of outlier detection

According to the above figure, the maximum PSNR achieved is approximate 38.86, and the corresponding is 65. The original image I and the resulting image using outlier detection with are displayed as follows, respectively.

sample1.png sample1.salt.pepper.outlier.65.png: cleaned by outlier detection with

Comparing the above two figures, we can see that the outlier detection approach with , which achieves PSNR = 38.86, works quite well, as there were very few white and black points left. I also displayed other figures with different to get a feeling of low and high PSNR values.

sample1.salt.pepper.outlier.25.png: sample1.salt.pepper.outlier.50.png:

cleaned by outlier detection with cleaned by outlier detection with

sample1.salt.pepper.outlier.100.png: sample1.salt.pepper.outlier.150.png:

cleaned by outlier detection with cleaned by outlier detection with

sample1.salt.pepper.outlier.200.png: sample1.salt.pepper.outlier.250.png:

cleaned by outlier detection with cleaned by outlier detection with

Surprisingly, the result of outlier detection with also looks pretty good to me, although it achieves lower PSNR value (36.31). This may indicate that sometimes PSNR value does not really reveal the feeling of human perception. As for other , the resulting images were obviously worse than and , especially for and , the outlier detection approach seems not even working!

The second approach I tried is **median filter**. My implementation of median filter does not require any extra parameters, and the resulting image, which achieves PSNR = 33.08, is displayed along with the resulting image of outlier detection with , which achieves PSNR = 38.86, as follows, respectively.

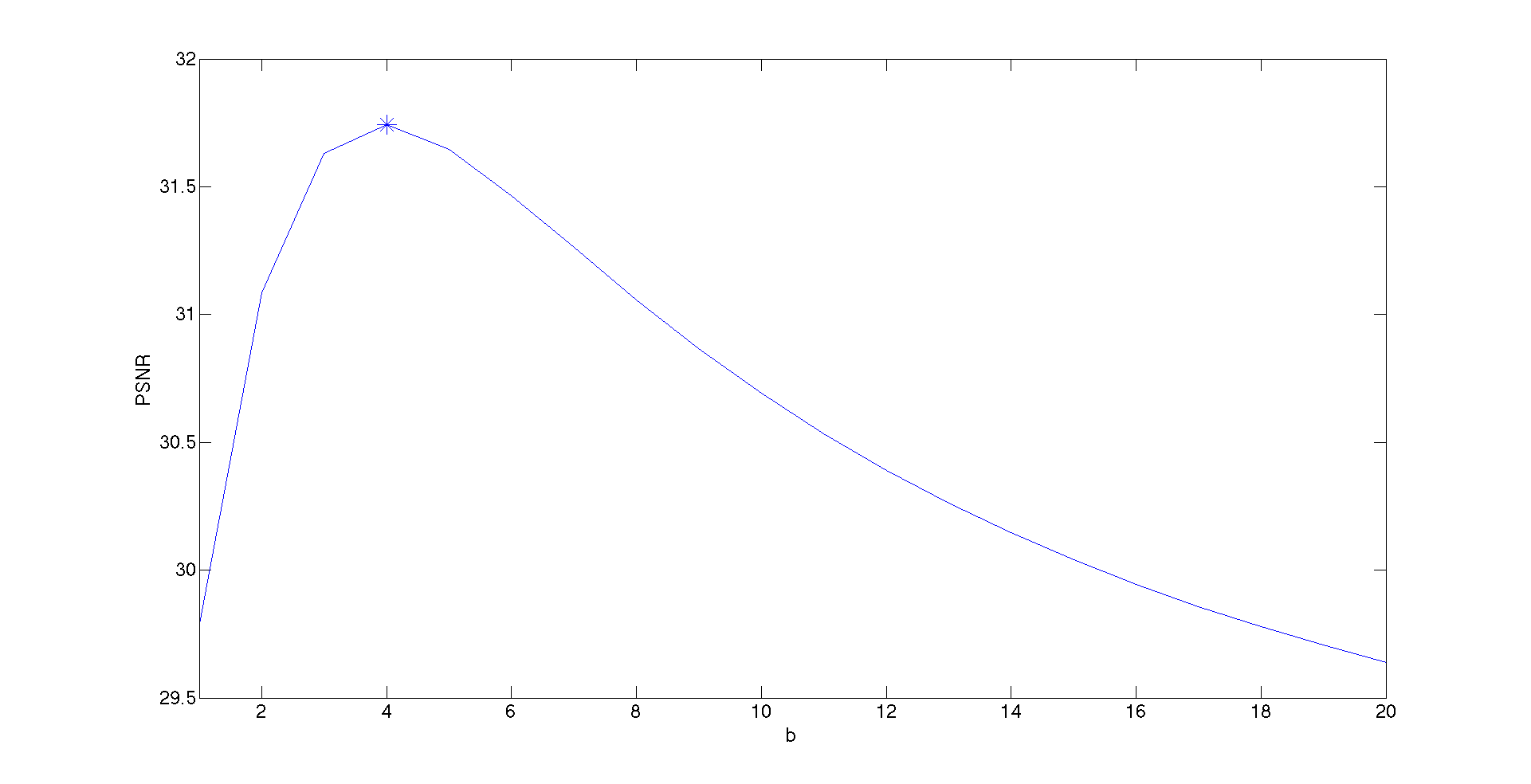
 

sample1.salt.pepper.median.png: sample1.salt.pepper.outlier.65.png:

cleaned by median filter cleaned by outlier detection with

Interestingly, no white and black points can be perceived in the figure on the left (cleaned by median filter), this may because that those white and black “outliers” were eliminated by the median operation. However, we can find that such median operation also wiped out the “continuous intensity” between the two neighbors, making the resulting image looks blurrier than the image cleaned by outlier detection approach. Therefore, cleaned by outlier detection with still looks better than that cleaned by the median filter, so let’s denote the former one as .

For uniform noise like Gaussian noise, low-pass filter is a wise choice. Similar with outlier detection, there’s one parameter that can be adjusted in low-pass filter (I fix the window size to 3x3), the , which determines how the filter looks like. Again, I plot the relationship between and the corresponding PSNR values to decide the best . The curve is displayed as follows.



low.pass.psnr.png: The relationship between and the achieved PSNR value of low-pass filter

According to the above curve, the maximum PSNR = 31.74 and is achieved by, that is, the filter looks like this:

.

We denote the image cleaned by low-pass filter with as , and the original image I and the cleaned image are displayed as follows, respectively.

sample1.png sample1.gaussian.low.pass.png: cleaned by low-pass filter with

The noisy image and the cleaned image are also displayed so that we can observe the improvement.

sample1.gaussian.png: Image I with Gaussian sample1.gaussian.low.pass.4.png: cleaned by

noise () added low-pass filter with

In my opinion, the improvement is passable and obviously imperfect. To further convince us that is the best choice, let’s take a look at the resulting images cleaned by (uniform) and :

sample1.gaussian.low.pass.1.png: cleaned sample1.gaussian.low.pass.10.png: cleaned by

by low-pass filter with low-pass filter with

The PSNR values achieved are 29.78 and 30.69, respectively, both are lower than (31.74). According to my observation, looked blurrier than , while still exhibited significant Gaussian noise, comparing with . somehow locates between them and can be reluctantly considered the best choice.

1. Compute the PSNR values of and and provide some discussions.

As mentioned in question (c), the outlier detection with is selected as , and low-pass filter with is selected as . The PSNR values achieved by and are 38.86 and 31.74, respectively. The discussion was already presented in question (c).