Problem 1&2 All the result images are stored in the folder named "Filters_Results"

Codes are "my_conv.m" and "Fliters_test" "Verification.m"

1. Smoothing Filters:

(1) Box filter: soften the sharpness of the original image

(2) Weighed Box Filter: sharper than box filter, reserve more information of the central pixel

(3) Horizontal Filter: reserve information of central pixel and two beside it, and it produces some horizontal traces.

(4) Vertical Filter: reserve information of central pixel and two pixels above and below it, and it produces some vertical traces.

(5) Circular-Shape Filter: reduce white dots on the original image

(6) 5/5 Box Filter: softer than Box filter, over blurry

(7) 5/5 Circular Shape Filter: similar to 5/5 box filter and darker as a whole

As to "Lenna", the best filter is Weighed Box Filter

As to "Traffic", the best filter is Circular-Shape Filter

The difference between Box and Weighed Box filter could be the weighed one take more information from the central pixel to produce the filtered pixel, thus the sharpness is higher.

2. Enhancement Filters:

(1) 4 × 4 Laplacian: It enhances the outlier of the image content, and make the lines white and background black, it extract the pixels that were at the joint of dark and light pixels

(2) 8 × 8 Laplacian: Similar effects while the outliers are enhanced thicker and lighter while more dots are shown

The 4 × 4 Laplacian is better

3. Edge Filters:

(1) Prewitt X: extract the edge of the image and more concentration is given to left side of content and vertical lines

(2) Prewitt Y: more concentration is given to the content above and horizontal lines

(3) Sobel X : more concentration is given to right side of content and vertical lines

(4) Sobel Y: more concentration is given to the content below and horizontal lines

Problem 3. Gaussian Smoothing Code are “GaussianSmoothing.m” and “GaussianFilter.m”

Compare with fspecial() Function : (part of results)

```
H =
    0.0049    0.0092    0.0134    0.0152    0.0134    0.0092    0.0049
    0.0092    0.0172    0.0250    0.0283    0.0250    0.0172    0.0092
    0.0134    0.0250    0.0364    0.0412    0.0364    0.0250    0.0134
    0.0152    0.0283    0.0412    0.0467    0.0412    0.0283    0.0152
    0.0134    0.0250    0.0364    0.0412    0.0364    0.0250    0.0134
    0.0092    0.0172    0.0250    0.0283    0.0250    0.0172    0.0092
    0.0049    0.0092    0.0134    0.0152    0.0134    0.0092    0.0049

H_compare =
    0.0049    0.0092    0.0134    0.0152    0.0134    0.0092    0.0049
    0.0092    0.0172    0.0250    0.0283    0.0250    0.0172    0.0092
    0.0134    0.0250    0.0364    0.0412    0.0364    0.0250    0.0134
    0.0152    0.0283    0.0412    0.0467    0.0412    0.0283    0.0152
    0.0134    0.0250    0.0364    0.0412    0.0364    0.0250    0.0134
    0.0092    0.0172    0.0250    0.0283    0.0250    0.0172    0.0092
    0.0049    0.0092    0.0134    0.0152    0.0134    0.0092    0.0049
```

```
H =
    0.0030    0.0133    0.0219    0.0133    0.0030
    0.0133    0.0596    0.0983    0.0596    0.0133
    0.0219    0.0983    0.1621    0.0983    0.0219
    0.0133    0.0596    0.0983    0.0596    0.0133
    0.0030    0.0133    0.0219    0.0133    0.0030

H_compare =
    0.0030    0.0133    0.0219    0.0133    0.0030
    0.0133    0.0596    0.0983    0.0596    0.0133
    0.0219    0.0983    0.1621    0.0983    0.0219
    0.0133    0.0596    0.0983    0.0596    0.0133
    0.0030    0.0133    0.0219    0.0133    0.0030
```

```
H =
    0.0113    0.0838    0.0113
    0.0838    0.6193    0.0838
    0.0113    0.0838    0.0113

H_compare =
    0.0113    0.0838    0.0113
    0.0838    0.6193    0.0838
    0.0113    0.0838    0.0113
```

The filtered image of Figure1 is stored in the ‘**Gaussian_Filters**’ folder.

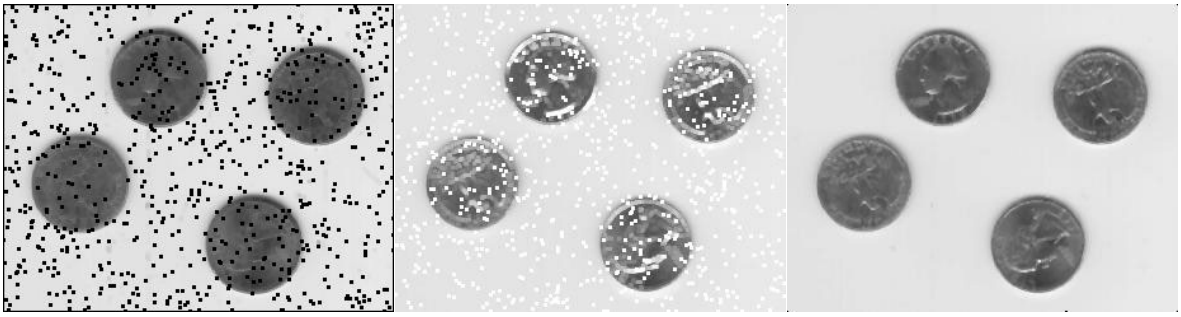
The effect of increasing window size: bigger the window size, less detail will be shown. It blurs the image

The effect of increasing sigma: the sigma decide how smoothing will the image be filtered to, the bigger sigma, more smoothing it will be.

Problem 4. Non-Linear Filters Code are “NonLinear.m” “Max_Filter.m” “Med_Filter.m” “Min_Filter.m”

The results are stored in the folder named “Non_linear”

Best setting for each type of filter



The best setting for each type of filter is 3 × 3, and the Median Filter performs the best.

Problem 5. Salt and Pepper vs White Noise

The results are store in the folder named "Noises"

Code are "Noises.m"

Gaussian Filter is best for white noise. Because the white noise is normally accorded with Gaussian distribution, and it is a low pass filter used for reducing noise and blurring regions of an image. The filter helps the average value of the surrounding pixel or neighboring pixels replaces the noisy pixel present in the image which is based on Gaussian distribution.

Median Filter is the best for salt and pepper noise. Because the "salt" and "pepper" which are presented by pixels which more than 250 or less than 5, thus once they are divided into a window and sorted, the pixel with median intensity replaces the noise.

Problem 6. Separable Convolution 2D

Code are "SeparableConvolution.m" and "separate_conv.m"

I'm not sure about the "problem 2" meaning, if it indicates 7 filters in Problem2, then No.1 2 3 4 6 are separatable, and No.5 7 aren't. Because when the filter is separatable, its rank has to be 1. Using function Rank(filter), we can tell whether its rank is 1 or not. Also isfilterseparable(filter) can be used.

If it indicates the filters listed in second question section, then No.1 3 4 5 are separable, and No.2 isn't, using the same method.

The time consumed by two convolution methods is listed as form:

	1	2*	3	4	5
Separable	0.002322s	/	0.002082s	0.002003s	0.002077s
Brute force	0.001104s	0.001078s	0.001528s	0.003995s	0.007529s

The time consumed above is based upon the conv2() Function, because I wanted it to be fair.

Using my own functions, the time ratio between two approaches are similar.

The results with two approaches are the same.

We can tell bigger the kernel size is, separable method is more likely to have advantage over brute force methods. Though the calculation of Brute force approach is XYMN, while to Separable approach, it's XY(M+N). Nevertheless, the separable 2D convolution requires additional storage (buffer) to keep the intermediate computations. So whether it saves time depend on the filters we choose.

Problem 7. Deblurring - Lucy Richardson

The images related are stored in the folder named "Lucy_Richardson"

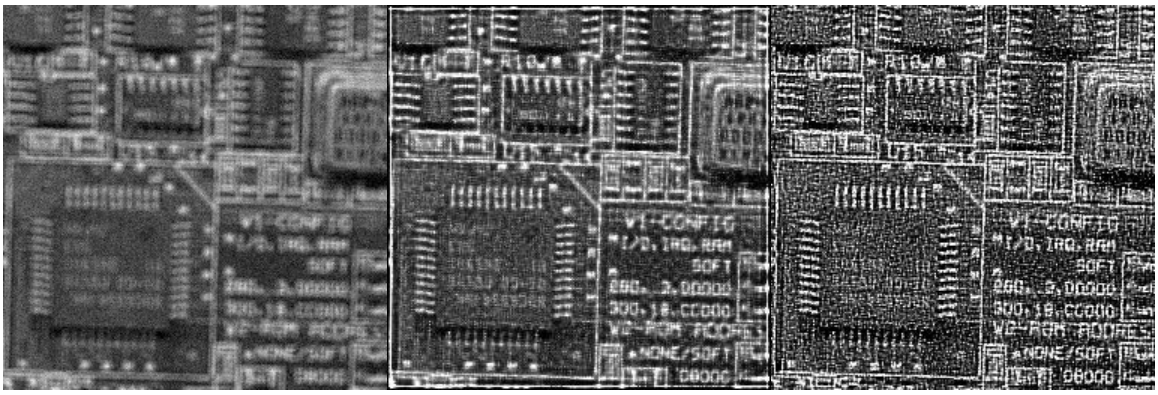
Codes are "Deblurring.m" and "SharpeningFilters"

1. Comparison with Lucy-Richardson implementation:

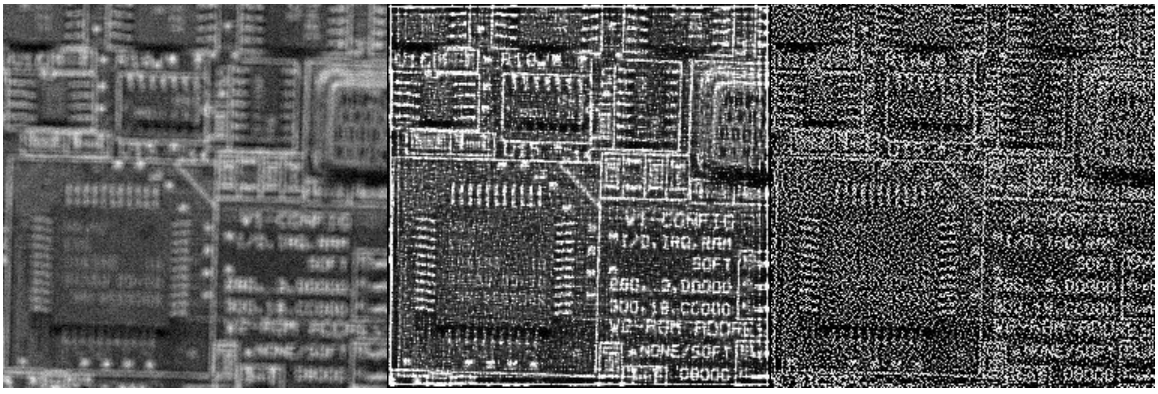
(1st: Original Image; 2nd : My Deblurring Scripts ; 3rd : `deconvlucy()`)

Circuit:

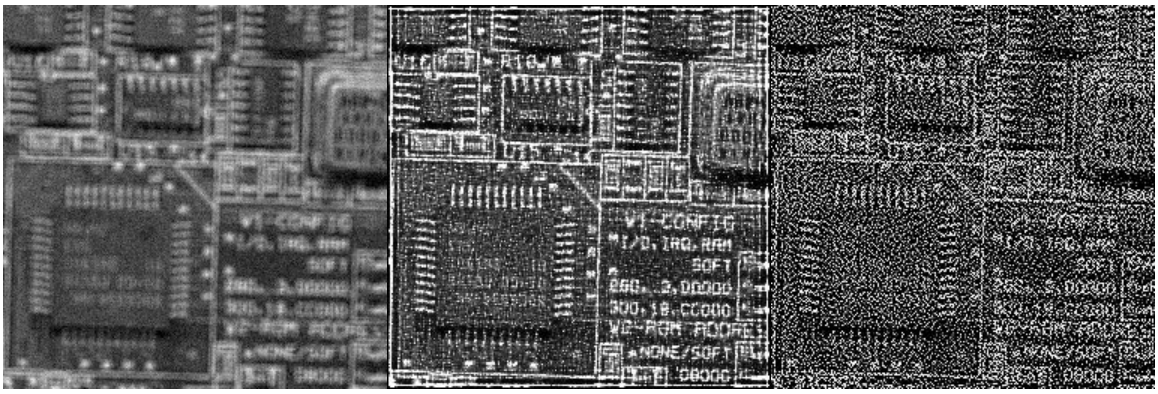
20 iterations:



200 iterations:



2000 iterations:

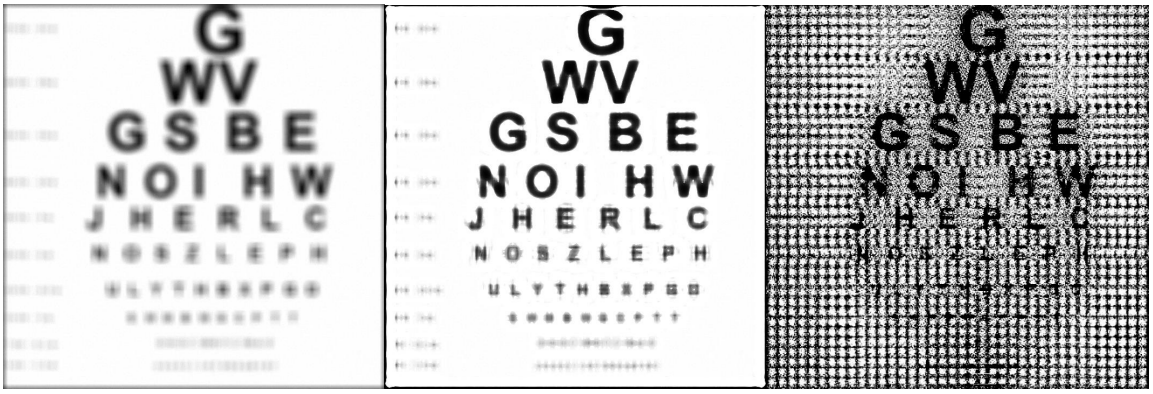


Eye chart:

20 iterations:

200 erations:

2000 iterations:

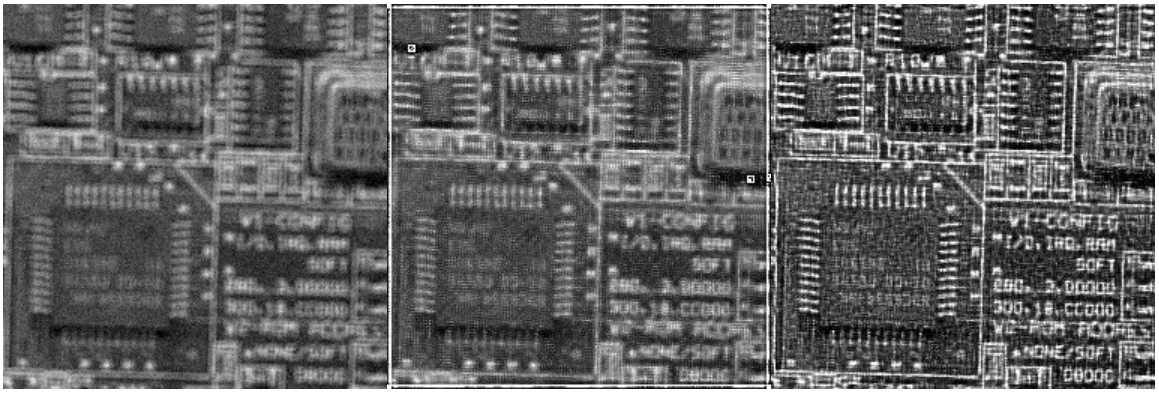


2. Using Med Filter Function to restore the blurred image:

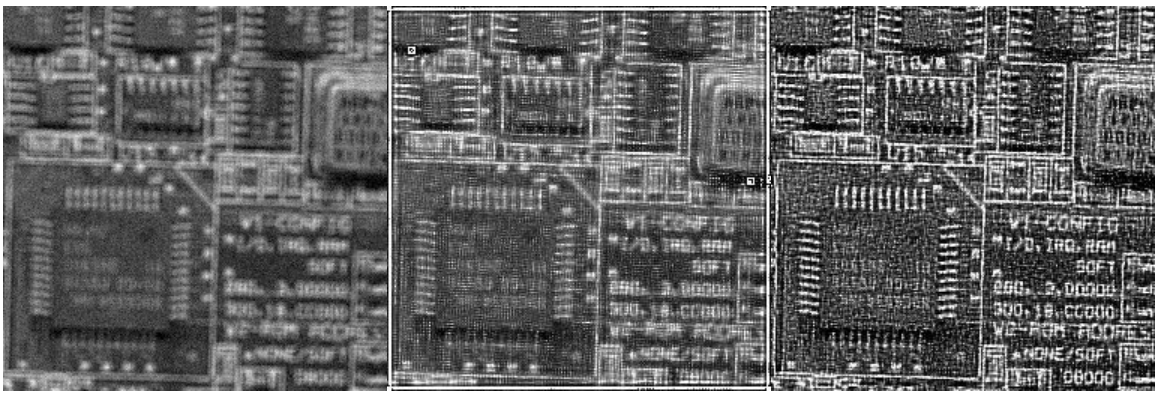
(1st: Original Image; 2nd : My Deblurring Scripts ; 3rd : `deconvlucy()`)

Circuits:

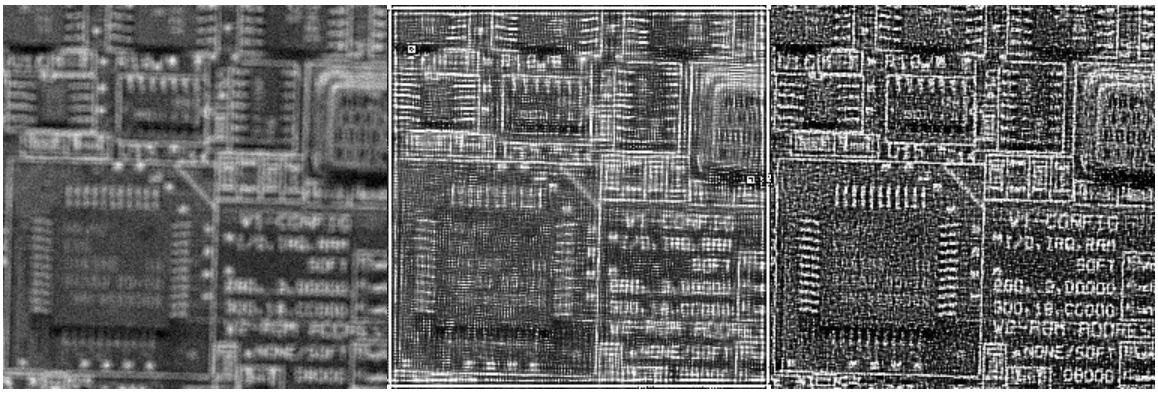
10 iterations:



15 iterations:



20 iterations:



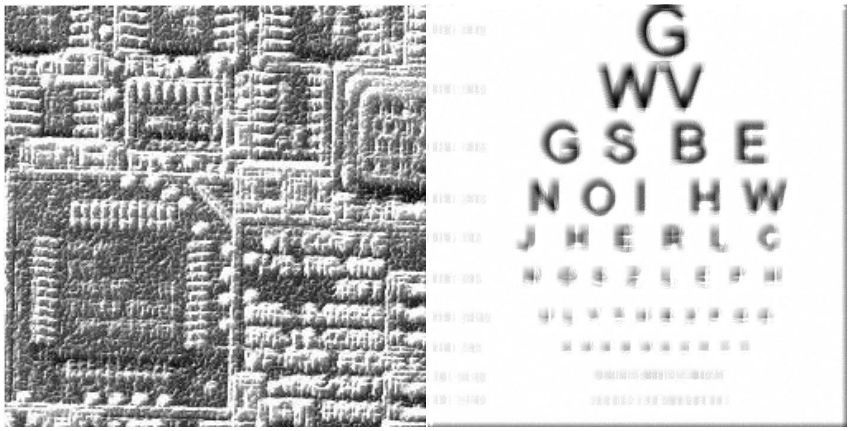
Eye chart: **the effect is poor, thus only 2 sets are shown here.**

10 iterations:

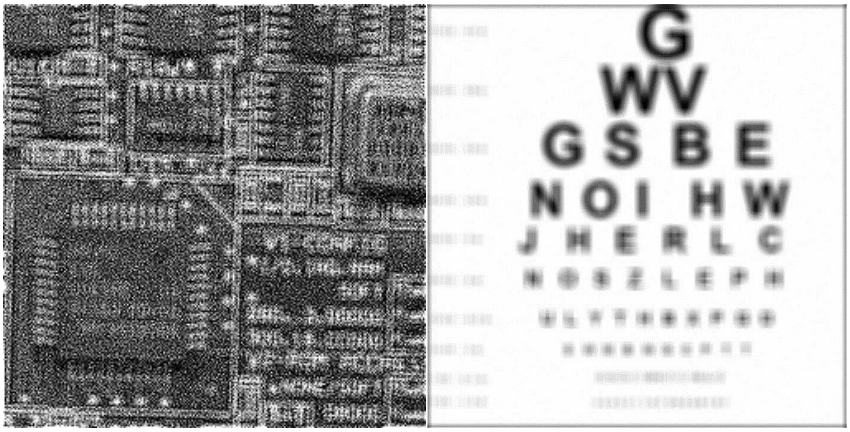
30 iterations:

Other sharpening filters:

Sobel Sharpening Image:



Laplacian Sharpening Image:



We can tell the Lucy-Richardson Approach is better than these two sharpening methods when processing these images.

Challenge:

The ringings are caused by noise amplification, it is a common problem of maximum likelihood methods that attempt to fit data as closely as possible. After many iterations, the restored image can have a speckled appearance, especially for a smooth object observed at low signal-to-noise ratios. Because the Lucy-Richardson Algorithm assumes the Poisson distribution, then after iterations, the difference between assumption and real noise could cause the ringings.

Solutions: (Internet)

Use the edgetaper function to preprocess your images before passing them to the deblurring functions. The edgetaper function removes the high-frequency drop-off at the edge of an image by blurring the entire image and then replacing the center pixels of the blurred image with the original image. In this way, the edges of the image taper off to a lower frequency.