Mathematical Modeling

Homework 10

DeYi Zeng

2019051712

Information Computer Science, JBJI

941659672@qq.com

June 18, 2021

Contents

1 Load a 256 × 256 greyscale image: 'Cameraman.png'

1.1 Codes

```
Images=imread('Cameraman.png');
imshow(Images,[])
```

1.2 Results

Figure 1: Original Image

2 Add Gaussian noise for the loaded image

It is known that the internal function imnoise will give random noise to our image.Therefore, it is very important for us to save the noisy image becasue we need to make sure that in the following procedures and comparisons we are using the same noisy images.

2.1 Codes

```
NoiseLevel=20;
sigma=(NoiseLevel*NoiseLevel)/(255*255);
Im_noisy=imnoise(Images,'gaussian',0,sigma);
%Save the noisy image
imwrite(uint8(Im_noisy),'Noisy Image20.png')
imshow(im2double(Im_noisy),[]);
```

2.2 Results

Figure 2: Noisy Image with Gaussian noise at level $\sigma = 20$

3 Formulating four functions: operatorB, operatorBT, prox, FPPAdraw

3.1 function operatorB

B is a difference operator of a vector. However, **B** is too large that we cannot store it as a dense matrix. Then we try to write a function named after operatorB which plays the same role as **B**. According to the definition,we write the following function:

```
function Bx=operatorB(x,m,n)
%Input 1: x is a vectorized m*n noisy image with Gaussian
%noise, where d=m*n.
%Input 2: m is the number of rows of pre-vectorized nosiy
%image
%Input 3: n is the number of columns of pre-vectorized
%nosiy image
%Output 1:Bx is the result of B multiplies x
%Reformulate the vector x to a matrix X, since we need the
%columns of X, then it is easier for us to process with
%a matrix X
X=reshape(x,m,n);
%To construct the upper matrix
IDx=[];
for i=1:n
    IDx=[IDx;[0;diff(X(:,i))]];
end
%To construct the lower matrix
DIx(m,1)=0;
for i=2:n
    DIx=[DIx;X(:,i)-X(:,i-1)];
end
%Joint the upper and lower matrix to be the full natrix
Bx=[IDx;DIx];
```

3.2 function operatorBT

$\mathbf{B}^T$ is also a difference operator of a vector. However, $\mathbf{B}^T$ is too large that we cannot store it as a dense matrix. Then we try to write a function named after operatorBT which plays the same role as $\mathbf{B}^T$. According to the definition,we write the following function:

```
function BTz=operatorBT(z,m,n)
%Input 1: z is a column vector,with dimension tp be 2d=2*m*
    n
%Input 2: m is the number of rows of pre-vectorized nosiy
```

```
%image
%Input 3: n is the number of columns of pre-vectorized
%nosiy image
%Output 1:BTz is the result of BT multiplies z
p=z(1:m*n);
q=z(m*n+1:2*m*n);
%Reformulate the vector p and q to a matrix P and Q, since
%we need the columns of P and Q, then it is easier for us
%to process with matrices P and Q
P=reshape(p,m,n);
Q=reshape(q,m,n);
%To construct the left-hand-side matrix
IDTp=[];
for i=1:n
%The following is a procedure that can construct new
%columns and append it to the original column vector
    temp=-diff(P(:,i));
    temp(1)=-P(2,i);
    IDTp=[IDTp;[temp;P(m,i)]];
end
%To construct the right-hand-side matrix
DITq(1:m,1)=-Q(:,2);
for i=2:n-1
    DITq=[DITq;Q(:,i)-Q(:,i+1)];
end
DITq=[DITq;Q(:,n)];
%Joint the left and right-hand-side matrix to be the
%full matrix
BTz=IDTp+DITq;
```

3.3 function prox

According to properties:

Let $\rho \in (0, \infty]$

For $x \in \mathbb{R}$, $\quad \text{prox}_{\rho|\cdot|}(x) = \max(|x| - \rho, 0) \cdot \text{sgn}(x)$

Then if we substitute ρ with $\frac{\mu}{\lambda}$, then it becomes the following:

For $x \in \mathbb{R}$, $\quad \text{prox}_{\frac{\mu}{\lambda}|\cdot|}(x) = \max(|x| - \frac{\mu}{\lambda}, 0) \cdot \text{sgn}(x)$

For $\boldsymbol{x} \in \mathbb{R}^n$, $\quad \text{prox}_{\rho\|\cdot\|}(\mathbf{x}) = [\text{prox}_{\rho|\cdot|}(x_1), \text{prox}_{\rho|\cdot|}(x_2), ..., \text{prox}_{\rho|\cdot|}(x_n)]^T$

Then if we substitute ρ with $\frac{\mu}{\lambda}$, then it becomes the following:

For $\boldsymbol{x} \in \mathbb{R}^n$, $\quad \text{prox}_{\frac{\mu}{\lambda}\|\cdot\|}(\mathbf{x}) = [\text{prox}_{\frac{\mu}{\lambda}|\cdot|}(x_1), \text{prox}_{\frac{\mu}{\lambda}|\cdot|}(x_2), ..., \text{prox}_{\frac{\mu}{\lambda}|\cdot|}(x_n)]^T$

Thus, we formulate the prox function with MATLAB internal function max, abs and sign:

```
function out=prox(x,lambda,mu)
%Input 1: x is the vector to be operated on
%Input 2: lambda
%Input 3: mu
%Output 1: out is the return value of prox(x)
out=max(abs(x)-mu/lambda,0).*sign(x);
```

3.4 function FPPAdraw

Function FPPAdraw is designed for performing fixed point proximity algorithm when lambda and mu are specified. In this function, user can select whether to draw the graphs.

- Algorithm terminates if iteration times exceeds 2000 times
- Algorithm terminates if normalized relative error(NRE) is less than 0.9×10^{-3}
- NRE(k) := $\frac{\left\|\boldsymbol{u}^k - \boldsymbol{u}^{k-1}\right\|_2}{\left\|\boldsymbol{u}^k\right\|_2}$
- The normalized square root mean square error (NRMSE) is the level (ratio) of difference between origial image and denoised image
- NRMSE(k) := $\frac{\left\|\boldsymbol{u}^k - \boldsymbol{u}_{\text{true}}\right\|_2}{\left\|\boldsymbol{u}_{\text{true}}\right\|_2}$
- PSNR is an index that judge the difference level between Original Image and denoised Image,higher PSNR tells that higher approximation to real image
- PSNR := $20log_{10}(\frac{255}{\left\|\boldsymbol{u}_{\text{true}} - \boldsymbol{u}_{\text{end}}\right\|_2})$

```
function [psnr,k,Denoised_Image]=FPPAdraw(Images,Im_noisy,
    draw,lambda,mu)
function [psnr,k,Denoised_Image]=FPPAdraw(Images,Im_noisy,
    draw,lambda,mu)
%Input 1: If we do not have original images,then
```

```
%input []. Else input the original Images
%Input 2: Input the noisy images
%Input 3: If you want to draw the pictures, then
%input 1. Else input 0.
%Input 4: lambda
%Input 5: mu
%Output 1: PSNR
%Output 2: Iteration times
%Output 3: Denoised Image
format long g
%Vectorize the Im_noisy matrix and convert it to double
v=double(Im_noisy(:));
m=size(Im_noisy,1);
n=size(Im_noisy,2);
%Initialize y0 and u0
y=operatorB(v,m,n);
ulast=v-lambda*operatorBT(operatorB(v,m,n),m,n);
%This is the first terminal condition, maximum iteration
%times limitation, if k>2000, then the algorithm terminates
for k=1:2000
z=operatorB(v,m,n)+y-lambda*operatorB...
    (operatorBT(y,m,n),m,n);
y=z-prox(z,lambda,mu);
u=double(v-lambda*operatorBT(y,m,n));
if (~isempty(Images))
    NRMSE(k)=norm(u-double(Images(:)),2)/...
    norm(double(Images(:)),2);
end
NRE(k)=norm(u-ulast,2)/norm(u,2);
f(k)=0.5*(norm(u-v,2))^2+mu*norm...
    (operatorB(u,m,n),1);
ulast=u;
%NRE is the second terminal condition, if NRE is less than
% 0.9*10^(-3), then break out the loop
if NRE(k)<0.9*10^(-3)
```

```
39      break
40  end
41  end
42  Denoised_Image=uint8(reshape(u,m,n));
43  if (draw==1)&&(~isempty(Images))
44      subplot(1,3,1); imshow((Images),[]);
45      title('Original image')
46      A=(double(Images(:))-u)/255;
47      psnr=20/log(10)*(log(255)-log(norm(A,2)));
48  elseif (draw==0)&&(~isempty(Images))
49      A=(double(Images(:))-u)/255;
50      psnr=20/log(10)*(log(255)-log(norm(A,2)));
51  else
52      psnr=('PSNR does not exists without Original Image');
53  end
54  if (draw==1)
55      Im_noisy=double(Im_noisy);
56      subplot(1,3,2); imshow(im2double(Im_noisy),[]);
57      title('Nosiy image')
58      subplot(1,3,3); imshow(reshape(u,m,n),[])
59      title('Denoised imaged')
60  end
61  if (draw==1)&&(~isempty(Images))
62      figure(3)
63      plot(NRMSE(1:k),'Color',[48 151 164]/255)
64      title('NRMSE')
65      xlabel('Iteration times')
66  end
67  if (draw==1)
68      figure(2)
69      plot(f(1:k),'Color',[0,70,222]/255)
70      title('Objective value [0,255]')
71      xlabel('Iteration times')
72      figure(4)
73      plot(f(1:k)/(255^2),'Color',[0,70,222]/255)
```

```
74      title('Objective value [0,1]')
75      xlabel('Iteration times')
76      figure(5)
77      plot(NRE(1:k),'r')
78      title('NRE')
79      xlabel('Iteration times')
80  end
```

4 Call your denoising function to denoise the image with Gaussian noise

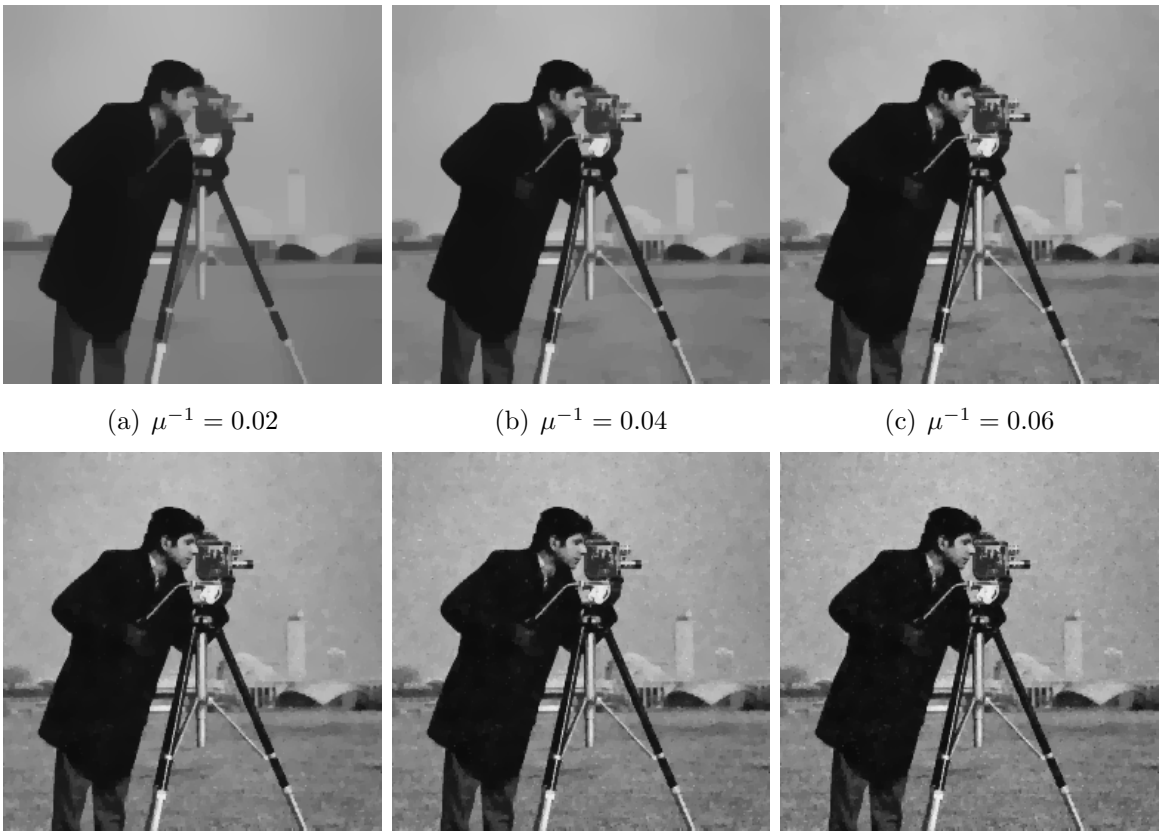
4.1 Codes

- Reload the original images and noisy image which has been processed in the section 2.1 by MATLAB internal function imnoise.
- In this test, we choose $\lambda = 0.15, 0.2499$ and $\mu = 0.02, 0.04, 0.06, 0.08, 0.09, 0.1$
- Save and display the denoised images and the chart of PSNR and iteration times.

```
1  Images=imread('Cameraman.png');
2  Im_noisy=imread('Noisy Image20.png');
3  draw=0;
4  for lambda=[0.15,0.2499]
5      for ii=[2,4,6,8,9,10]
6          mu=100/ii;
7          [PSNR, Ite, Denoised_Image]=FPPAdraw...
8          (Images, Im_noisy, draw, lambda, mu);
9          PSNR, Ite
10         %Save and display the image
11         if lambda==0.15
12             imwrite(Denoised_Image,[...
13             'Denoised Image0.15',...
14             num2str(ii),'.png'])
15             figure(ii)
16             imshow(Denoised_Image,[])
17         else
```

```
                imwrite(Denoised_Image,['Denoised Image',...
                num2str(ii),'.png'])
                figure(ii+10)
                imshow(Denoised_Image,[])
            end
        end
    end
```

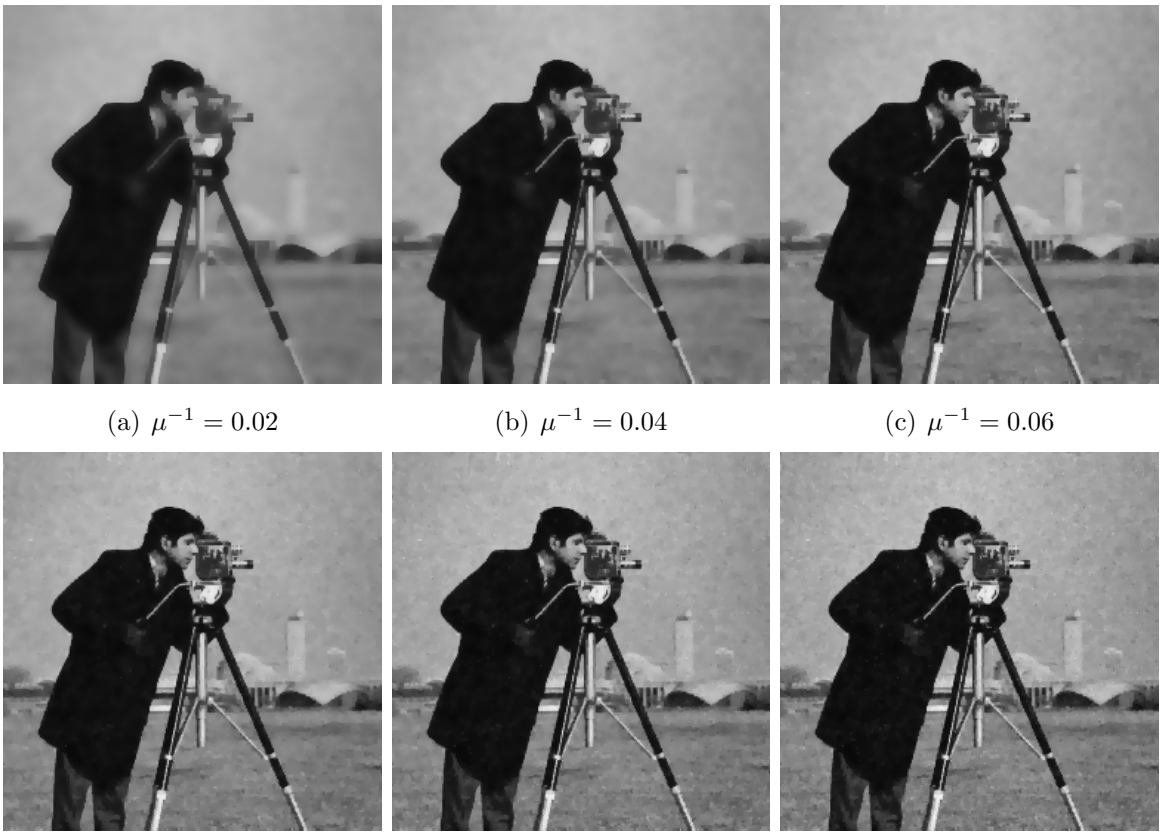
4.2 Results



(a) $\mu^{-1} = 0.02$ (b) $\mu^{-1} = 0.04$ (c) $\mu^{-1} = 0.06$

(d) $\mu^{-1} = 0.08$ (e) $\mu^{-1} = 0.09$ (f) $\mu^{-1} = 0.10$

Figure 3: Denoising the noisy image of 'Cameraman' with Gaussian noise at level $\sigma = 20$ and $\lambda = 0.2499$ by FPPA. (a)$\mu^{-1} = 0.02$, PSNR $= 23.93$, Ite $= 496$;(b)$\mu^{-1} = 0.04$, PSNR $= 26.52$, Ite $= 207$;(c)$\mu^{-1} = 0.06$, PSNR $= 27.94$, Ite $= 92$;(d)$\mu^{-1} = 0.08$, PSNR $= 28.65$, Ite $= 45$;(e)$\mu^{-1} = 0.09$, PSNR $= 28.77$, Ite $= 34$;(f)$\mu^{-1} = 0.10$, PSNR $= 28.76$, Ite $= 26$;



(a) $\mu^{-1} = 0.02$ (b) $\mu^{-1} = 0.04$ (c) $\mu^{-1} = 0.06$

(d) $\mu^{-1} = 0.08$ (e) $\mu^{-1} = 0.09$ (f) $\mu^{-1} = 0.10$

Figure 4: Denoising the noisy image of 'Cameraman' with Gaussian noise at level $\sigma = 20$ and $\lambda = 0.15$ by FPPA. (a)$\mu^{-1} = 0.02, \text{PSNR} = 23.95, \text{Ite} = 32$;(b)$\mu^{-1} = 0.04, \text{PSNR} = 26.52, \text{Ite} = 21$;(c)$\mu^{-1} = 0.06, \text{PSNR} = 27.89, \text{Ite} = 17$;(d)$\mu^{-1} = 0.08, \text{PSNR} = 28.58, \text{Ite} = 15$;(e)$\mu^{-1} = 0.09, \text{PSNR} = 28.70, \text{Ite} = 14$;(f)$\mu^{-1} = 0.10, \text{PSNR} = 28.70, \text{Ite} = 13$;

We can draw some basic conclusions: Peak-signal-to-noise ratio (PSNR) fluctuates little, Number of Iterations increases, remained noise decreases and sharpness decreases when λ increases. Peak-signal-to-noise ratio (PSNR) decreases and Number of Iterations increases, remained noise decreases and sharpness decreases when μ increases(μ^{-1} decreases). From here we notice that the remained noise level increases when sharpness increases.

5 Show the comparison of the original image, the noisy image and the denoised image

In this section we make comparsion between original image, the noisy image with Noise-Level $\sigma = 20$ and two types of denoised image.

Since we have obtained and 12 denoised pictured and their performances under FPPA in section 3, we choose two of the them to show their images.

In addition, the denoised images have been saved so we can use it directly by reading it from the MATLAB internal file folder.

5.1 Codes

```
Images=imread('Cameraman.png');
Im_noisy=imread('Noisy Image20.png');
subplot(2,2,1)
imshow((Images),[]);
title('Original image')
subplot(2,2,2)
imshow(Im_noisy)
title('Noisy Image')
subplot(2,2,3)
imshow('Denoised Image6.png')
title('Denoised Image,{\lambda}=0.2499,\mu^{-1}=0.06')
subplot(2,2,4)
imshow('Denoised Image0.156.png')
title('Denoised Image,{\lambda}=0.15,\mu^{-1}=0.06')
```

5.2 Results

(a) Original Image

(b) Noisy Image

(c) Denoised Image,$\lambda = 0.2499, \mu^{-1} = 0.06$

(d) Denoised Image,$\lambda = 0.15, \mu^{-1} = 0.06$

Figure 5: Comparison

6 Plot the figures of objective function value, normalized root mean square error (NRMSE) and normalized relative error (NRE) versus the number of iteration

The FPPA is used to finding the minimizer as well as minimum of the objective value. Objective value(k):= $\frac{1}{2}\left\|\boldsymbol{u}^k-\boldsymbol{v}\right\|_2^2+\mu\left\|\boldsymbol{B}\boldsymbol{u}^{\boldsymbol{k}}\right\|_1$. The normalized square root mean square error (NRMSE) is a type of level of difference between original images and the denoised images. NRMSE(k) := $\frac{\left\|\boldsymbol{u}^k-\boldsymbol{u}_{\text{true}}\right\|_2}{\left\|\boldsymbol{u}_{\text{true}}\right\|_2}$. NRE measures the variation before and after each iteration. NRE(k) := $\frac{\left\|\boldsymbol{u}^k-\boldsymbol{u}^{k-1}\right\|_2}{\left\|\boldsymbol{u}^k\right\|_2}$.

Since in function FPPAdraw, we have designed a switch named after draw ploting objective value, NRE and NRMSE versus iteration times. Thus, we select two representative cases and switch on plotting.

6.1 Codes and Results

```
Images=imread('Cameraman.png');
Im_noisy=imread('Noisy Image20.png');
draw=1;
lambda=0.2499;
mu=100/2;
[PSNR, Ite , Denoised_Image]=FPPAdraw...
(Images , Im_noisy , draw , lambda , mu);
```

```
Images=imread('Cameraman.png');
Im_noisy=imread('Noisy Image20.png');
draw=1;
lambda=0.15;
mu=100/8;
[PSNR, Ite , Denoised_Image]=FPPAdraw...
(Images , Im_noisy , draw , lambda , mu);
```

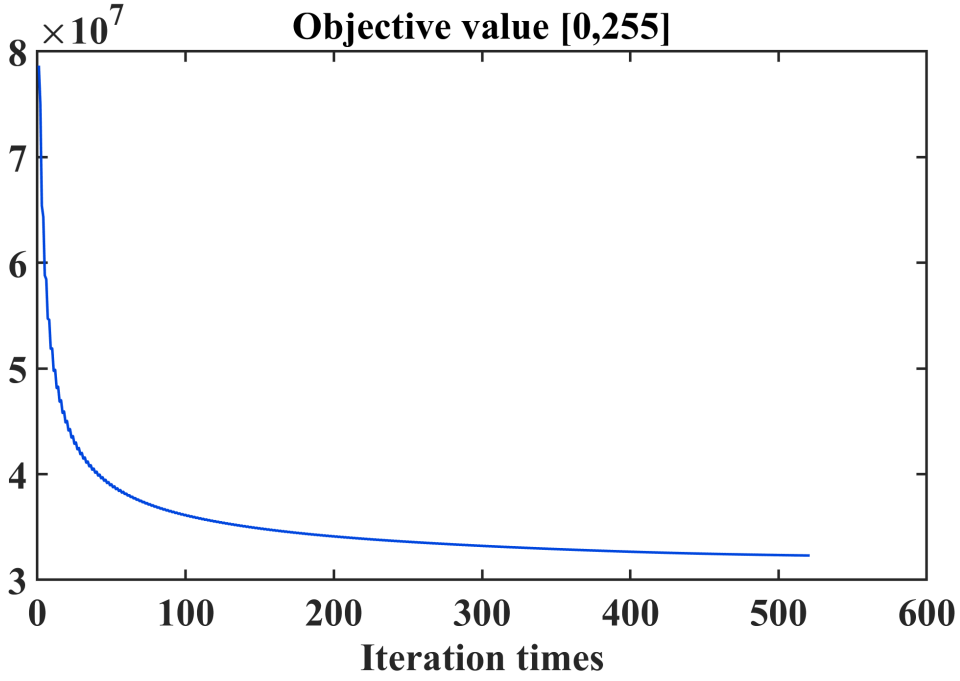


Figure 6: $\lambda = 0.2499, \mu^{-1} = 0.02$

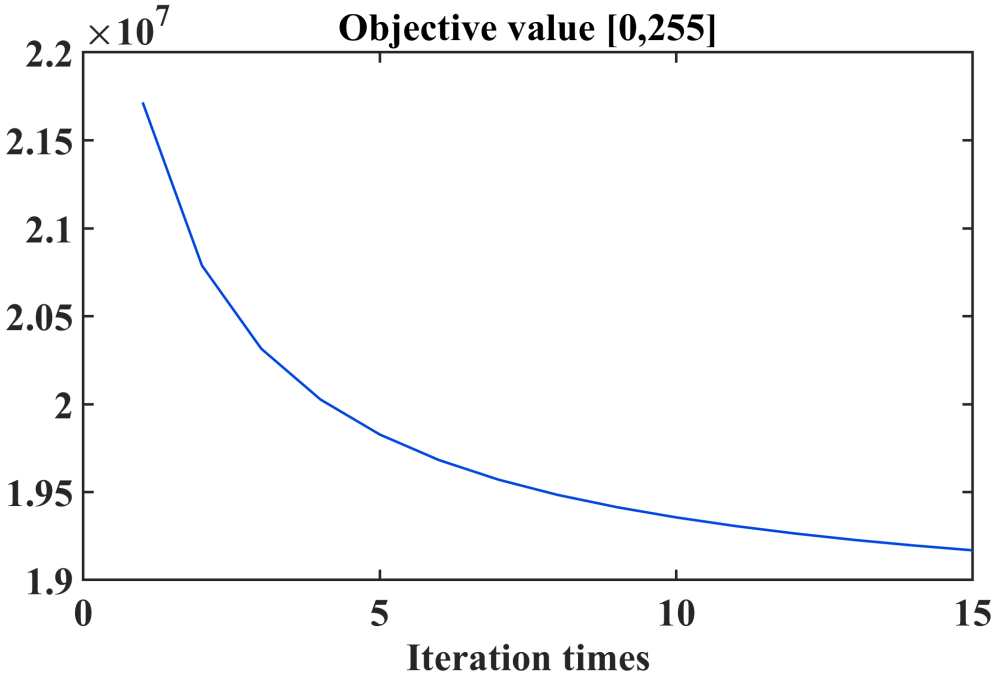


Figure 7: $\lambda = 0.15, \mu^{-1} = 0.08$

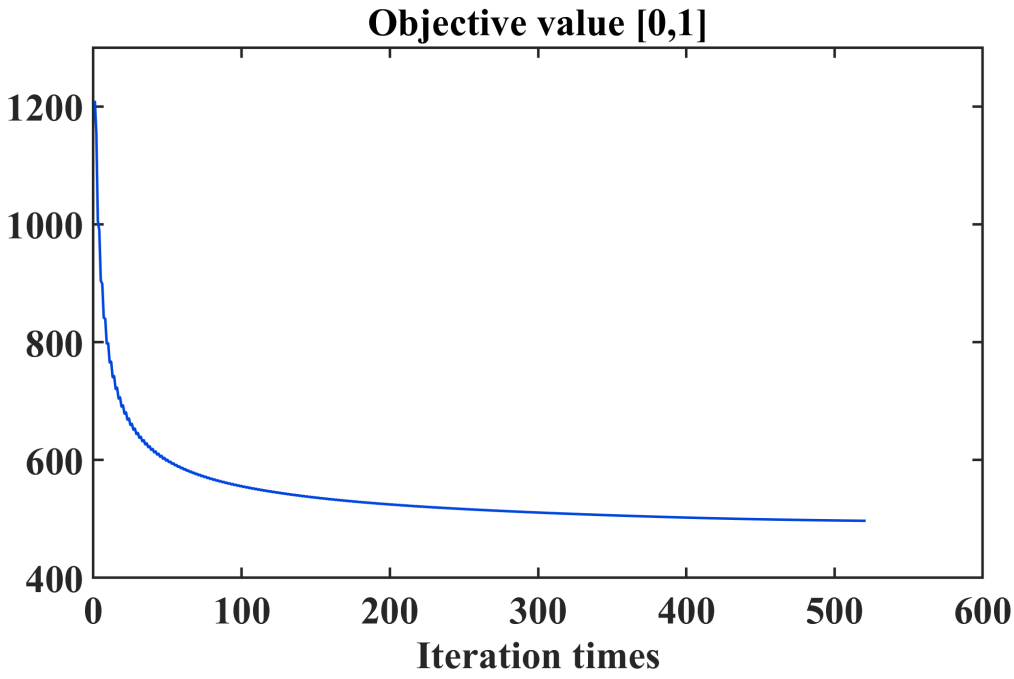


Figure 8: $\lambda = 0.2499, \mu^{-1} = 0.02$

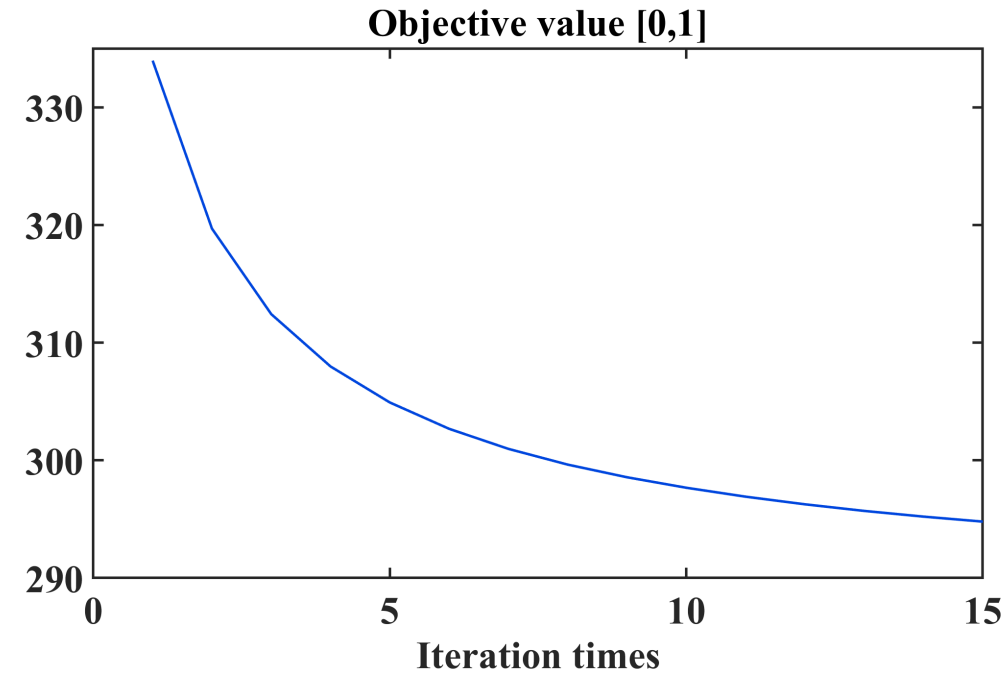


Figure 9: $\lambda = 0.15, \mu^{-1} = 0.08$

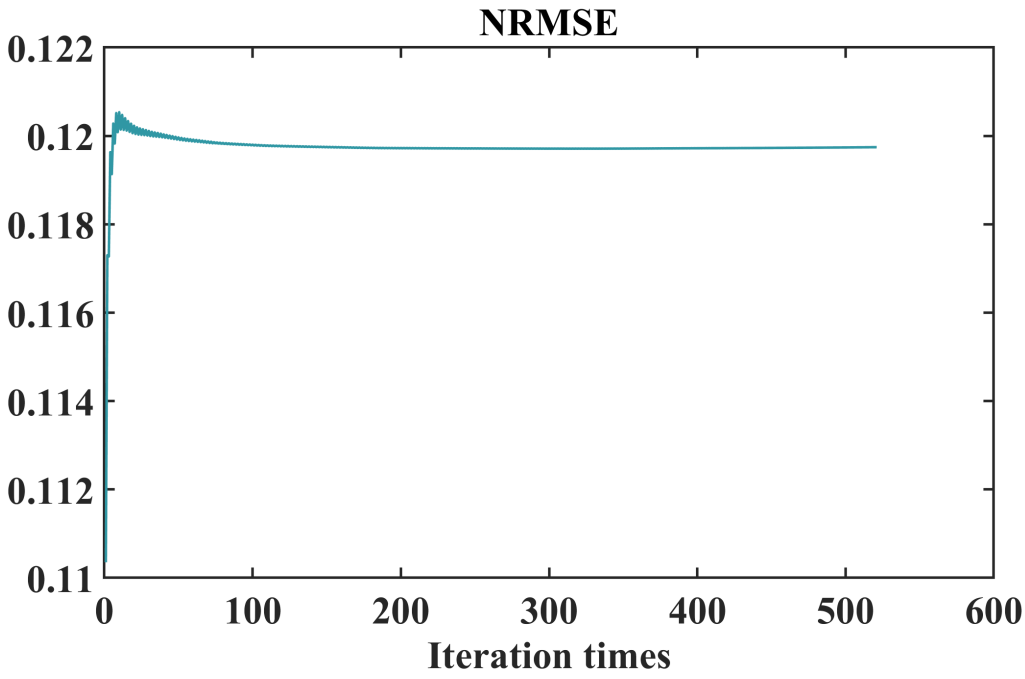


Figure 10: $\lambda = 0.2499, \mu^{-1} = 0.02$

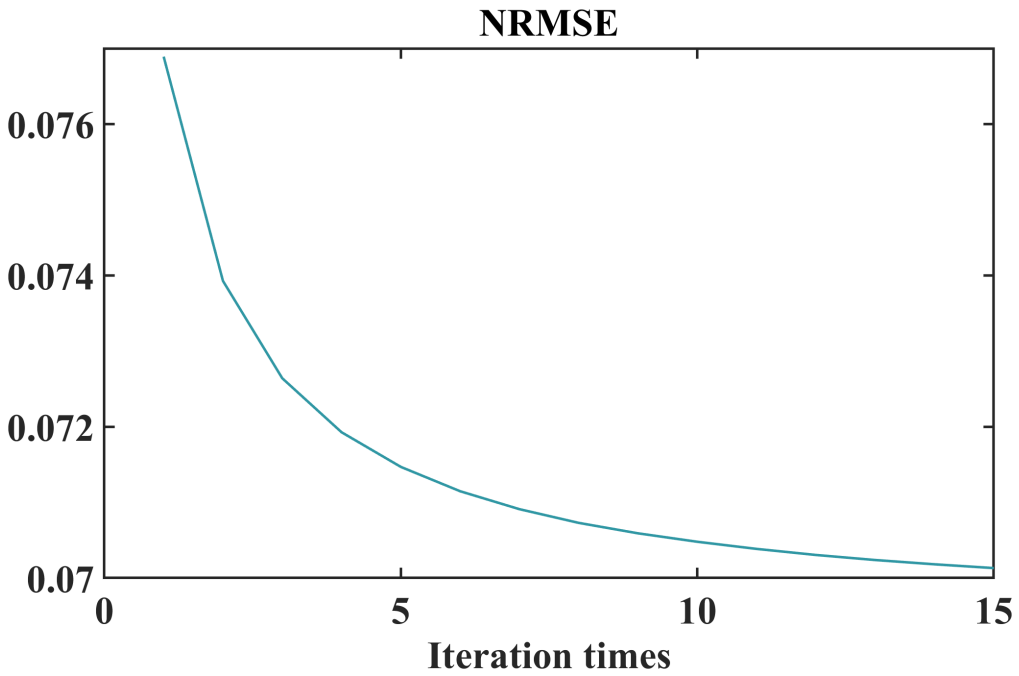


Figure 11: $\lambda = 0.15, \mu^{-1} = 0.08$

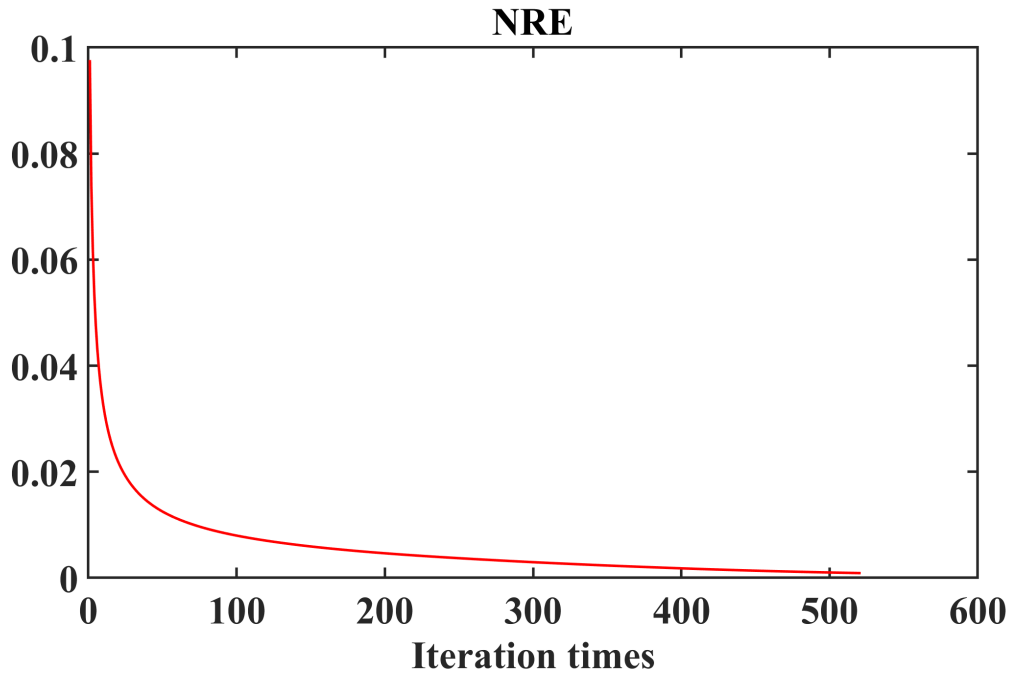
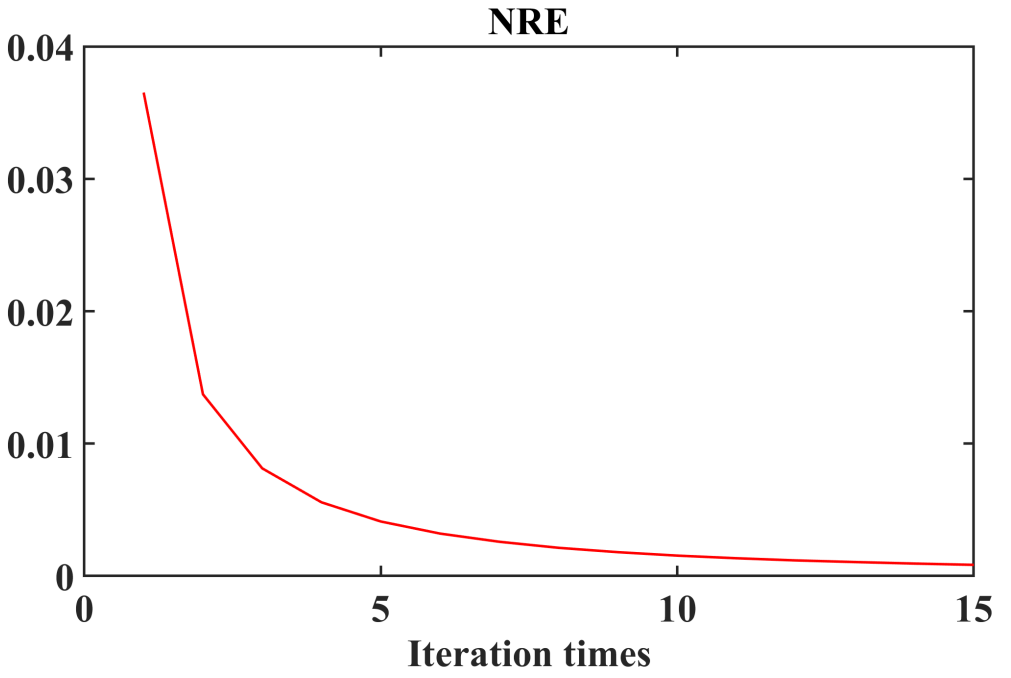


Figure 12: $\lambda = 0.2499, \mu^{-1} = 0.02$



;

Figure 13: $\lambda = 0.15, \mu^{-1} = 0.08$

6.2 Analysis

We will do some analysis of these eight graphs.

- For Figure 6,7 and Figure8,9, they are precisely the same things under different magnitudes, [0,1] and [0,255] respectively. If your iamges matrices are in the [0,1], the objective value is 65025 times less than objective value[0,255].
- When focusing on the objective function, we notice that they will eventually descendingly converge stably to a certain value. There may be some differences when you calculating ovjective value in different magnitude. But as we have stated in the last item, they are essentially the same thing as their trends are completely the same.
- In reality, sometimes(when $\lambda \geq 0.25$) they are not strictly descending as it has been shown in the given figures, they will fluctuate at the beginning because of the properties of fixed-point iterations. However, whatever they vary at the beginning they will eventually converge to a certain value. And the minimizer μ that makes the function value lying around this value is exactly what we want.
- NRMSE is one of two indices that indicates the difference level between the original image and the denoised image(The other one is PSNR).However, we need to be very careful here that higher NRMSE and PSNR do not imply the better denoising. We can focus on figure 3 and 4. The higher difference level they are, the less denoising as well.
- We observe that they are two types of evolutional trends in our graphs provided. One experiences a fluctutaion at the beginning, followed by a decreasingly convergece to a certain value. Another experience a continuous decrease to a certain value.They are all possible cases as the denoising procedures may lead the imgae away from the real image.
- NRE indicates the variation level between the last and the subsequent iteration, and it is the terminal condition of our algorithm. With the iteration times increases, the NRE will become less. And our terminal condition is that the NRE less than a tolerance which is set to be 0.9×10^{-3}.

7 Testing several regularization parameter λ and algorithmic parameter μ

In this section, we decide to combine the test of regularization parameter λ and algorithmic parameter μ into a joint test,by choosing the following value:

- $\mu^{-1} = 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1$
- $\lambda = 0.01 : 0.01 : 0.25$

In this section, we will only give an example of making part of heatmap of different λ and μ under the condition that $\sigma = 20$. For the full version of heatmap and other noise level, please refer to Appendix. These heatmaps are of great significance since we will extract information from them to list a table and perform some further analysis in section 9.

7.1 Codes of PSNR and Number of Iterations under noise level $\sigma = 20$

```
Images=imread('Cameraman.png');
Im_noisy=imread('Noisy Image20.png');
draw=0;
PSNRtable=[];
Itetable=[];
for ii=1:10
mu=100/ii;
for lambda=0.01:0.03:0.25
[PSNR, Ite, Denoised_Image]=FPPAdraw...
(Images, Im_noisy, draw, lambda, mu);
PSNRtable=[PSNRtable,PSNR];
Itetable=[Itetable, Ite];
end
end
PSNRtable=reshape(PSNRtable,9,10);
Itetable=reshape(Itetable,9,10);
figure(1)
h1=heatmap(PSNRtable);
h1.Colormap=bone;
h1.CellLabelFormat='%0.4g';
```

```
h1.XLabel='{\mu}^{-1}';
h1.YLabel='{\lambda}';  h1.Title='PSNR';
h1.XDisplayLabels={'0.01','0.02','0.03','0.04','0.05','0.06
    ','0.07','0.08','0.09','0.1'};
h1.YDisplayLabels={'0.01','0.04','0.07','0.1','0.13','0.16'
    ,'0.19','0.22','0.25'};
figure(2)
h2=heatmap(Itetable);
h2.XLabel='{\mu}^{-1}';
h2.YLabel='{\lambda}';   h2.Title='Ite';
h1.XDisplayLabels={'0.01','0.02','0.03','0.04','0.05','0.06
    ','0.07','0.08','0.09','0.1'};
h2.YDisplayLabels={'0.01','0.04','0.07','0.1','0.13','0.16'
    ,'0.19','0.22','0.25'};
```

7.2 Results

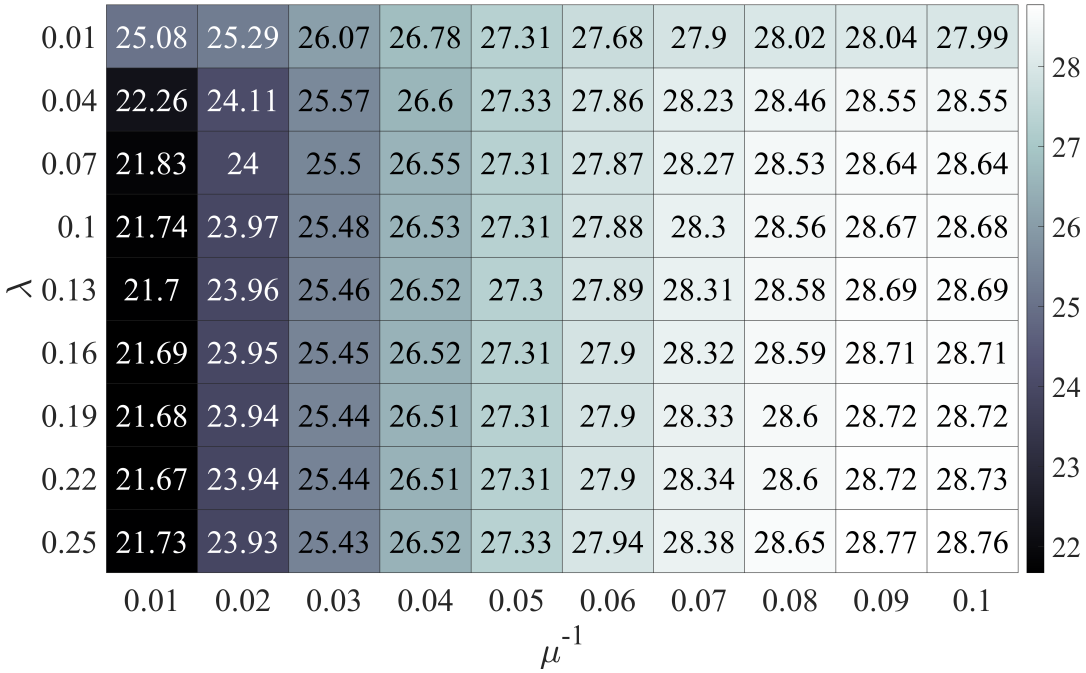


Figure 14: Heatmap of PSNR $\sigma = 20$

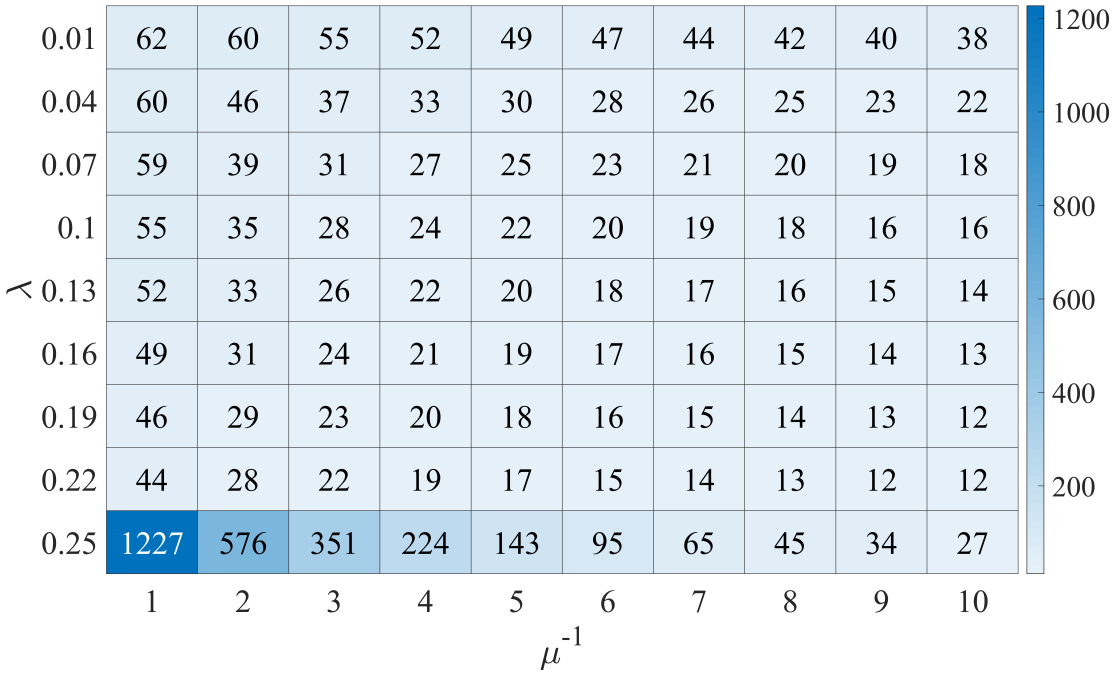


Figure 15: Heatmap of Iteration Times $\sigma = 20$

7.3 Codes of Objective Value and NRMSE

```
format long g
for lambda=0.05:0.05:0.25
for ii=2:2:10
    mu=100/ii;
    v=double(Im_noisy(:));
    m=size(Im_noisy,1);
    n=size(Im_noisy,2);
    y=operatorB(v,m,n);
    ulast=v-lambda*operatorBT(operatorB(v,m,n),m,n);
    for k=1:2000
        z=operatorB(v,m,n)+y-lambda*operatorB...
        (operatorBT(y,m,n),m,n);
        y=z-prox(z,lambda,mu);
        u=double(v-lambda*operatorBT(y,m,n));
        NRMSE(k)=norm(u-double(Images(:)),2)/...
```

```
16            norm(double(Images(:)),2);
17            NRE(k)=norm(u-ulast,2)/norm(u,2);
18            f(k)=0.5*(norm(u-v,2))^2+mu*norm...
19            (operatorB(u,m,n),1);
20            ulast=u;
21            if NRE(k)<0.9*10^(-3)
22            break
23            end
24        end
25        [lambda,ii/100,f(k),f(k)/255^2,NRMSE(k)]
26    end
27    end
```

7.4 Table of variable μ

According to the heatmaps given in the Section 7.2, we form the following two tables.

Table 1:

Gaussian noise at $\sigma = 20$,$\lambda = 0.15$					
μ^{-1}	0.02	0.04	0.06	0.08	0.1
PSNR(dB)	23.95	26.52	27.89	28.58	28.70
Number of Iterations(Ite)	32	21	17	15	13
Objective Value[0,255](10^7)	3.43	2.55	2.15	1.90	1.72
Objective Value[0,1]	528.14	392.07	331.02	292.54	264.09
NRMSE	0.12	0.09	0.08	0.07	0.07

7.5 Table of variable λ

Table 2:

Gaussian noise at $\sigma = 20$,$\mu^{-1} = 0.06$					
λ	0.05	0.1	0.15	0.2	0.25
PSNR(dB)	27.87	27.88	27.89	27.90	27.94
Number of Iterations(Ite)	26	20	17	16	95
Objective Value[0,255](10^7)	2.22	2.17	2.15	2.14	1.72
Objective Value[0,1]	341.36	334.06	331.02	328.84	264.09
NRMSE	0.08	0.08	0.08	0.08	0.08

7.6 Joint Table for λ and μ

We have obtained several heatmaps concerning Iteration times and peak-signal-to-noise ratio under different noise level in Section 7.1.For the full version of codes and heatmaps, please refer to Appendices. Now we select some representative datas to form a table.For a clear understanding of the algorithm results, we decide to list part of datas in a table similar to Table2 in Reference[1].But here we only focus on PSNR and Number of Iteration times.(PSNR,Ite)

Table 3: Numerical results of the FPPA for the images of 'Cameraman':(PSNR,Ite)

μ^{-1}	$\lambda = 0.05$	$\lambda = 0.1$	$\lambda = 0.15$	$\lambda = 0.2$	$\lambda = 0.25$
Gaussian noise at level $\sigma = 15$					
0.02	(24.22,42)	(24.12,35)	(24.10,31)	(24.09,29)	(24.07,550)
0.03	(25.76,34)	(25.66,27)	(25.62,24)	(25.60,22)	(25.57,398)
0.04	(26.86,29)	(26.77,24)	(26.74,21)	(26.72,19)	(26.68,302)
0.05	(27.70,26)	(27.62,21)	(27.59,19)	(27.58,17)	(27.55,213)
0.06	(28.36,25)	(28.30,19)	(28.28,17)	(28.28,15)	(28.26,145)
0.07	(28.89,23)	(28.86,18)	(28.85,16)	(28.85,14)	(28.85,97)
0.08	(29.32,22)	(29.31,17)	(29.31,15)	(29.32,13)	(29.33,71)
0.09	(29.65,21)	(29.67,16)	(29.68,14)	(29.68,12)	(29.71,54)
0.1	(29.90,20)	(29.94,15)	(29.96,13)	(29.97,12)	(30.00,42)

μ^{-1}	$\lambda = 0.05$	$\lambda = 0.1$	$\lambda = 0.15$	$\lambda = 0.2$	$\lambda = 0.25$
Gaussian noise at level $\sigma = 25$					
0.02	(23.90,44)	(23.83,36)	(23.81,32)	(23.81,29)	(23.81,535)
0.03	(25.30,36)	(25.26,29)	(25.25,25)	(25.25,23)	(25.25,422)
0.04	(26.23,32)	(26.23,25)	(26.24,22)	(26.24,20)	(26.27,230)
0.05	(26.86,29)	(26.89,23)	(26.91,20)	(26.92,18)	(26.98,119)
0.06	(27.22,27)	(27.29,21)	(27.32,18)	(27.34,16)	(27.40,72)
0.07	(27.35,25)	(27.43,19)	(27.47,16)	(27.49,15)	(27.55,46)
0.08	(27.27,23)	(27.37,18)	(27.40,15)	(27.42,14)	(27.47,32)
0.09	(27.07,22)	(27.16,17)	(27.19,14)	(27.21,13)	(27.24,25)
0.1	(26.79,21)	(26.87,16)	(26,89,13)	(26.90,12)	(26.92,20)
Gaussian noise at level $\sigma = 35$					
0.02	(23.47,46)	(23.44,37)	(23.44,32)	(23.44,29)	(23.44,812)
0.03	(24.66,38)	(24.70,30)	(24.71,26)	(24.72,24)	(24.76,383)
0.04	(25.31,34)	(25.39,26)	(25.43,23)	(25.44,20)	(25.52,119)
0.05	(25.42,30)	(25.54,24)	(25.58,20)	(25.60,18)	(25.66,56)
0.06	(25.15,28)	(25.25,21)	(25.28,18)	(25.30,16)	(25.34,35)
0.07	(24.66,26)	(24.73,19)	(24.76,16)	(24.77,14)	(24.80,26)
0.08	(24.10,23)	(24.16,18)	(24.18,15)	(24.19,13)	(24.20,20)
0.09	(23.57,22)	(23.61,16)	(23.62,14)	(23.63,12)	(23.64,16)
0.1	(23.08,20)	(23.11,15)	(23.12,12)	(23.13,11)	(23.13,14)

8 Show your denoising results for all provided sample images

According to the photos displayed in section 4.2, we decide to use the visually optimal parameters with $\lambda = 0.2499, \mu^{-1} = 0.06$ under noise level $\sigma = 20$

8.1 Codes

```
%Initialisation of parameters
NoiseLevel=20;
sigma=(NoiseLevel*NoiseLevel)/(255*255);
draw=0;
mu=100/6;
lambda=0.2499;
%Cameraman
Images=imread('Cameraman.png');
Im_noisy=imnoise(Images,'gaussian',0,sigma);
[PSNR, Ite, Denoised_Image]=FPPAdraw...
(Images, Im_noisy, draw, lambda, mu);
figure(1)
imwrite(Im_noisy,'Noisy_Cameraman.png')
imwrite(Denoised_Image,'Denoised_Cameraman.png')
subplot(1,3,1)
imshow(Images)
title('Original Image')
subplot(1,3,2)
imshow(Im_noisy)
title('Noisy Image')
subplot(1,3,3)
imshow(Denoised_Image)
title('Denoised Image')
%Lena
Images=imread('Lena.png');
Im_noisy=imnoise(Images,'gaussian',0,sigma);
[PSNR, Ite, Denoised_Image]=FPPAdraw...
```

```
(Images,Im_noisy,draw,lambda,mu);
figure(2)
imwrite(Im_noisy,'Noisy_Lena.png')
imwrite(Denoised_Image,'Denoised_Lena.png')
subplot(1,3,1)
imshow(Images)
title('Original Image')
subplot(1,3,2)
imshow(Im_noisy)
title('Noisy Image')
subplot(1,3,3)
imshow(Denoised_Image)
title('Denoised Image')
%Pepper
Images=imread('Pepper.png');
Im_noisy=imnoise(Images,'gaussian',0,sigma);
[PSNR,Ite,Denoised_Image]=FPPAdraw...
(Images,Im_noisy,draw,lambda,mu);
figure(3)
imwrite(Im_noisy,'Noisy_Pepper.png')
imwrite(Denoised_Image,'Denoised_Pepper.png')
subplot(1,3,1)
imshow(Images)
title('Original Image')
subplot(1,3,2)
imshow(Im_noisy)
title('Noisy Image')
subplot(1,3,3)
imshow(Denoised_Image)
title('Denoised Image')
```

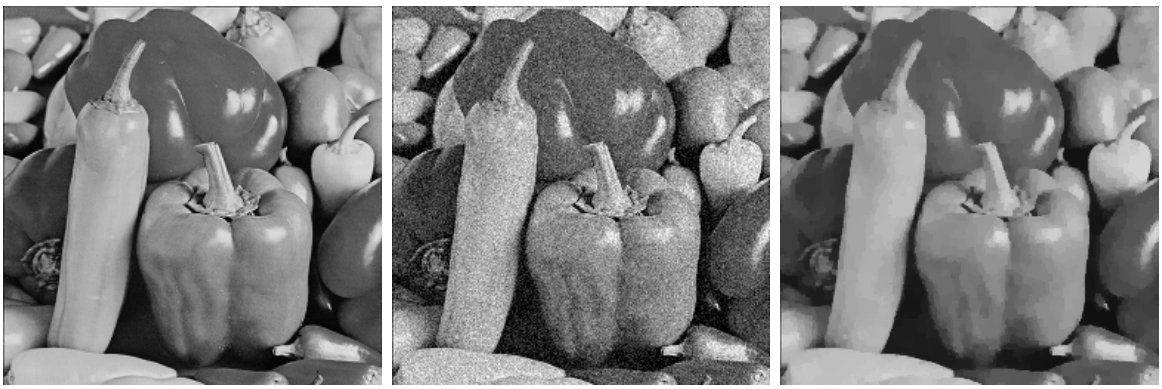
8.2 Results

(a) Original Image (b) Noisy Image (c) Denoised Image

Figure 16: Cameraman

(a) Original Image (b) Noisy Image (c) Denoised Image

Figure 17: Lena



(a) Original Image (b) Noisy Image (c) Denoised Image

Figure 18: Pepper

9 Further Analysis of Algorithm and Parameters

We have listed the tables in Section 7 and heatmaps in Appendices. We are going to do some further analysis according to these graphs.

- Number of Iterations witnesses decreasing trend when the $\lambda < 0.24$ and μ fixed. However, if the λ is approximating 0.25, then the Number of Iterations soar to a much higher level.
- Number of Iterations increases as μ increases(μ^{-1} decreases) and λ fixed.
- Before talking about PSNR, we need to restate that the higher PSNR does not imply the denoising effectiveness.
- The optimal parameters that maximize the PSNR of denoised images always have the $0.24 \leq \lambda \leq 0.25$ and μ increases(μ^{-1} decreases) as the noise level σ becomes higher.
- The effectiveness is mainly decided by the model parameter(regularization parameter) λ and secondly decided by algorithmic parameter μ, so it is of great significance for us to optimize the model parameter λ.

References

[1] Charles A Micchelli, Lixin Shen, and Yuesheng Xu. Proximity algorithms for image models: denoising. *Inverse Problems*, 27(4):045009, 2011.

[2] Yizun Lin. Lecture 6. Fixed-Point Proximity Algorithm for Image Denoising. *Mathematical Modeling*, 2021.

10 Appendices

10.1 Codes and results of noise level $\sigma = 15$

```
Images=imread('Cameraman.png');
NoiseLevel=15;
sigma=(NoiseLevel*NoiseLevel)/(255*255);
Im_noisy=imnoise(Images,'gaussian',0,sigma);
draw=0;
PSNRtable=[];
Itetable=[];
for ii=1:10
mu=100/ii;
for lambda=0.01:0.01:0.25
[PSNR, Ite, Denoised_Image]=FPPAdraw...
(Images, Im_noisy, draw, lambda, mu);
PSNRtable=[PSNRtable,PSNR];
Itetable=[Itetable, Ite];
end
end
PSNRtable=reshape(PSNRtable,25,10);
Itetable=reshape(Itetable,25,10);
figure(1)
h1=heatmap(PSNRtable);
h1.Colormap=bone;
h1.CellLabelFormat='%0.4g';
h1.XLabel='{\mu}^{-1}(x10^{-2})';
h1.YLabel='{\lambda(x10^{-2}})';
figure(2)
h2=heatmap(Itetable);
h2.XLabel='{\mu}^{-1}(x10^{-2})';
h2.YLabel='{\lambda(x10^{-2}})';
```

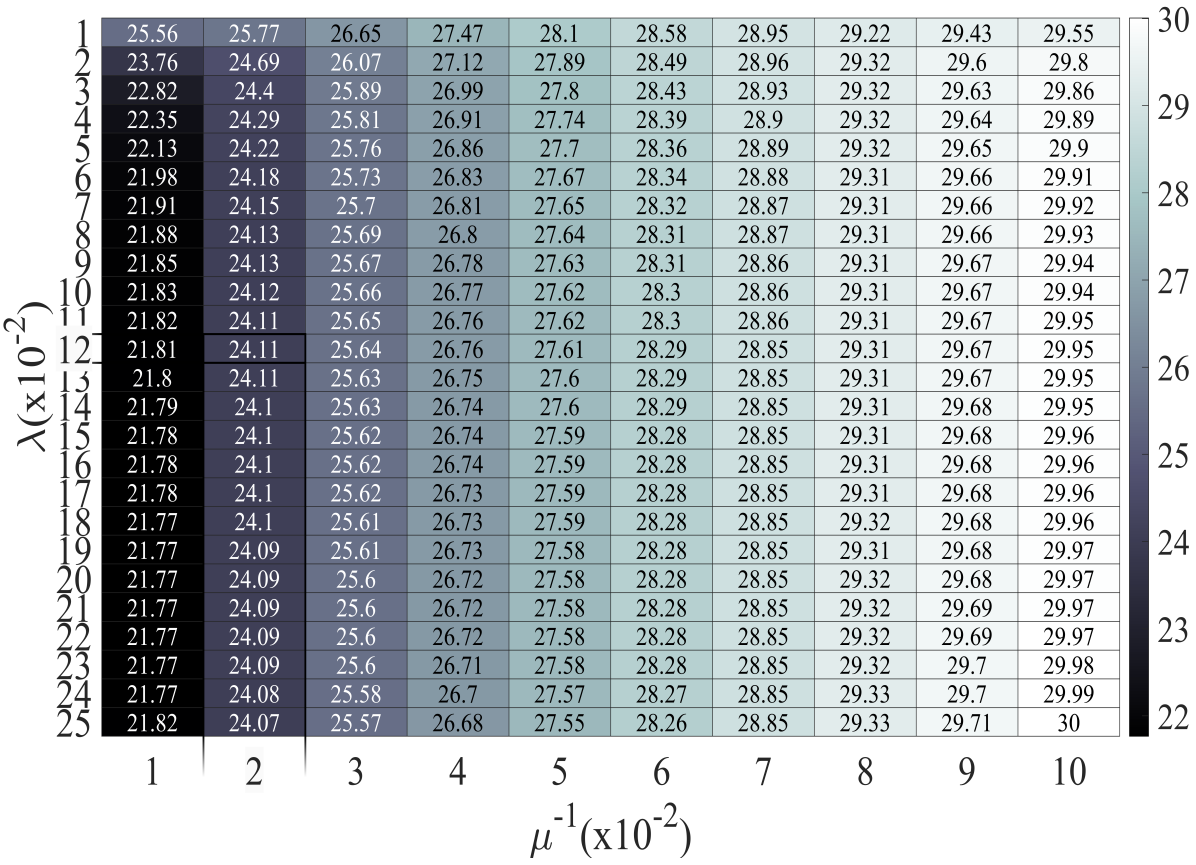


Figure 19: Heatmap of PSNR $\sigma = 15$

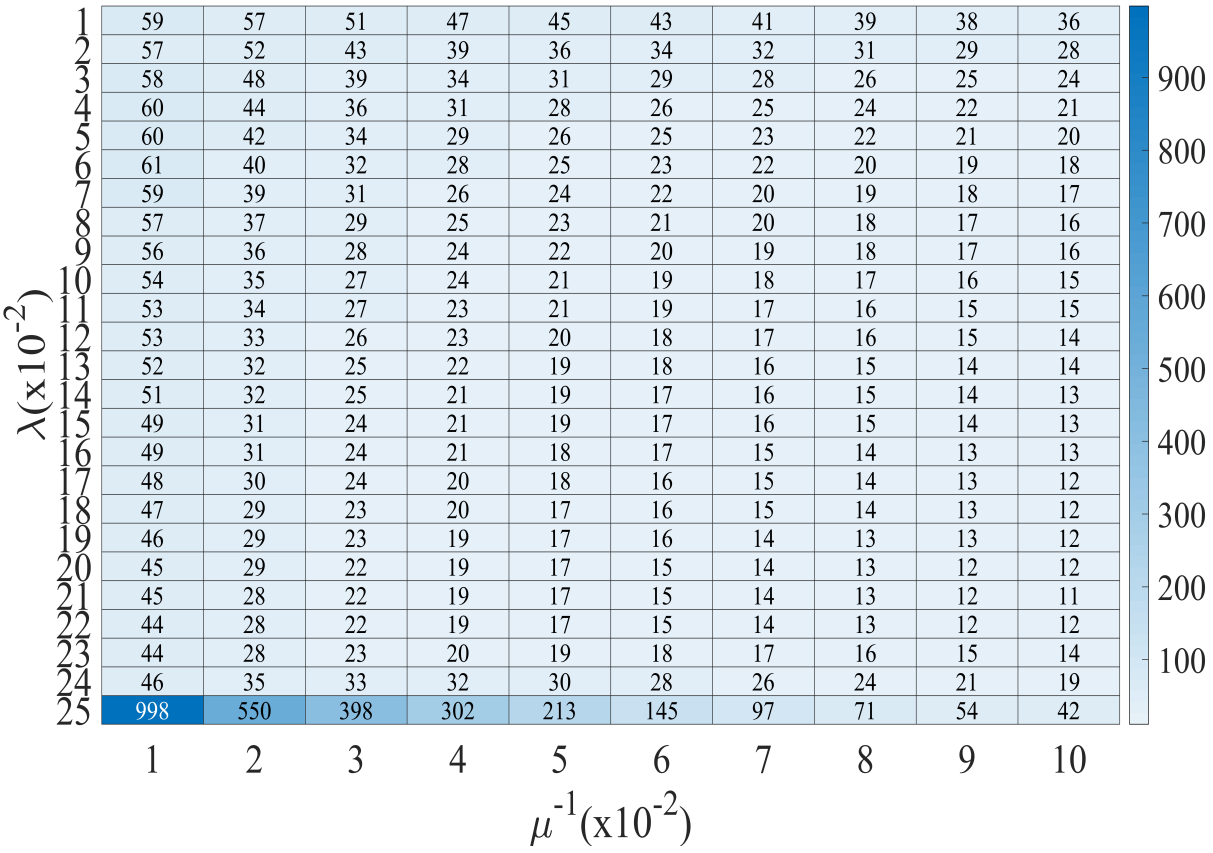


Figure 20: Heatmap of Iteration Times $\sigma = 15$

10.2 Codes and results of noise level $\sigma = 20$

```
Images=imread('Cameraman.png');
Im_noisy=imread('Noisy Image20.png');
draw=0;
PSNRtable=[];
Itetable=[];
for ii=1:10
    mu=100/ii;
    for lambda=0.01:0.01:0.25
        [PSNR, Ite, Denoised_Image]=FPPAdraw...
        (Images, Im_noisy, draw, lambda, mu);
        PSNRtable=[PSNRtable,PSNR];
        Itetable=[Itetable, Ite];
    end
end
PSNRtable=reshape(PSNRtable,25,10);
Itetable=reshape(Itetable,25,10);
figure(1)
h1=heatmap(PSNRtable);
h1.Colormap=bone;
h1.CellLabelFormat='%0.4g';
h1.XLabel='{\mu}^{-1}(x10^{-2})';
h1.YLabel='{\lambda(x10^{-2}})';
figure(2)
h2=heatmap(Itetable);
h2.XLabel='{\mu}^{-1}(x10^{-2})';
h2.YLabel='{\lambda(x10^{-2}})';
```

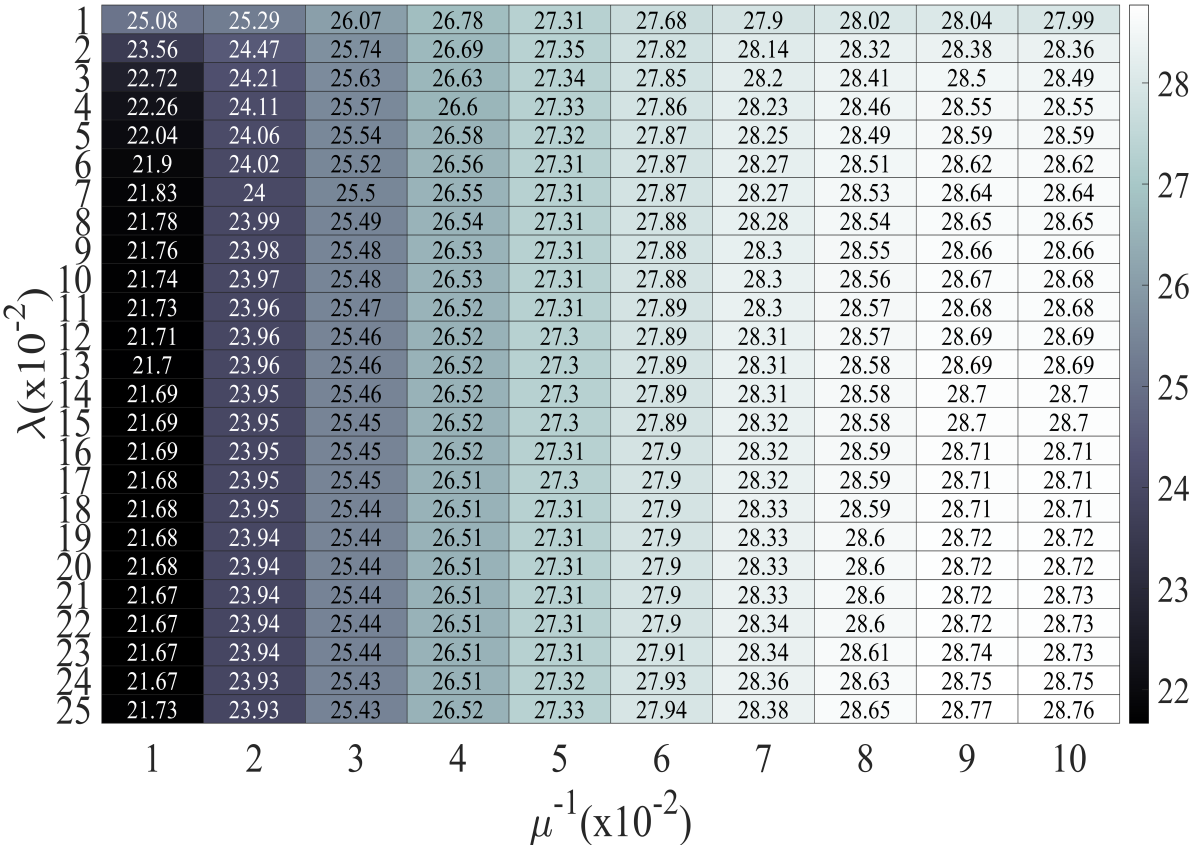


Figure 21: Heatmap of PSNR $\sigma = 20$

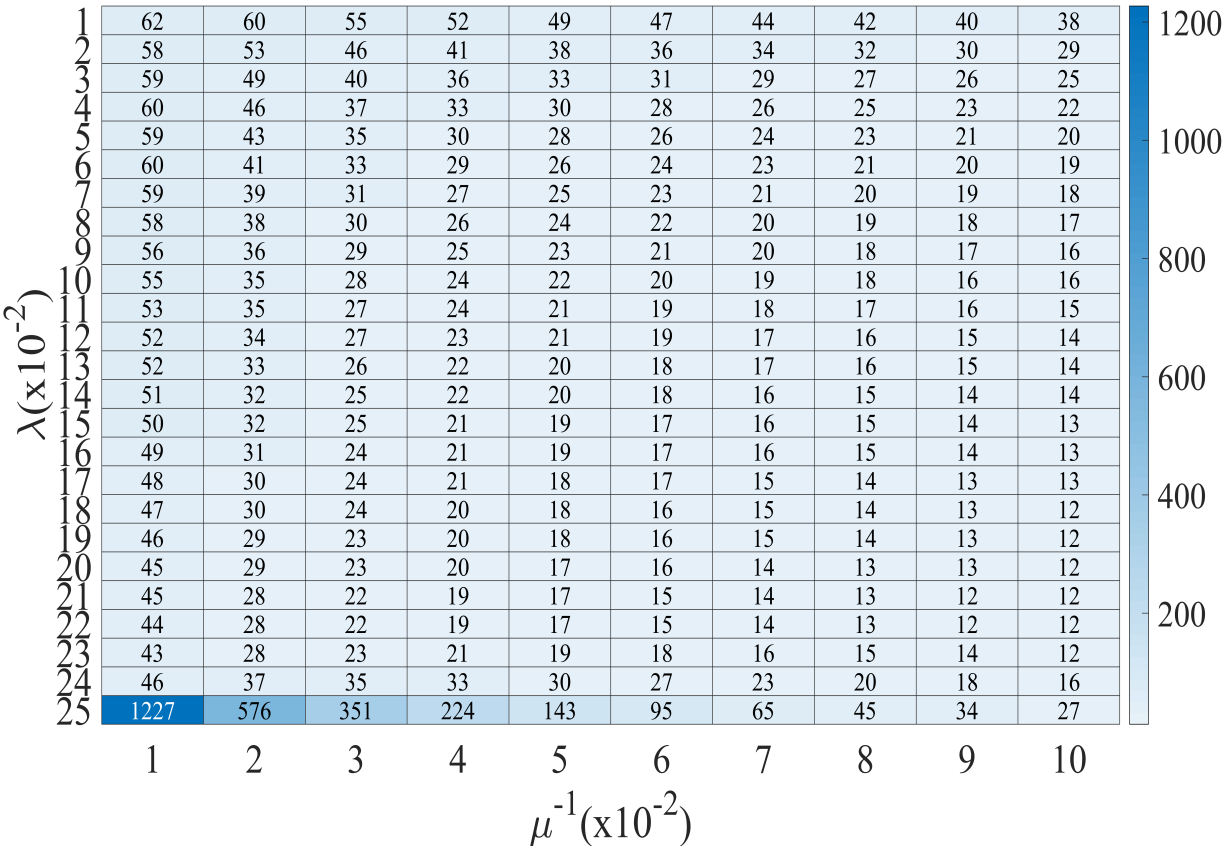


Figure 22: Heatmap of Iteration Times $\sigma = 20$

10.3 Codes and results of noise level $\sigma = 25$

```
Images=imread('Cameraman.png');
NoiseLevel=25;
sigma=(NoiseLevel*NoiseLevel)/(255*255);
Im_noisy=imnoise(Images,'gaussian',0,sigma);
draw=0;
PSNRtable=[];
Itetable=[];
for ii=1:10
mu=100/ii;
for lambda=0.01:0.01:0.25
[PSNR, Ite, Denoised_Image]=FPPAdraw...
(Images, Im_noisy, draw, lambda, mu);
PSNRtable=[PSNRtable,PSNR];
Itetable=[Itetable, Ite];
end
end
PSNRtable=reshape(PSNRtable,25,10);
Itetable=reshape(Itetable,25,10);
figure(1)
h1=heatmap(PSNRtable);
h1.Colormap=bone;
h1.CellLabelFormat='%0.4g';
h1.XLabel='{\mu}^{-1}(x10^{-2})';
h1.YLabel='{\lambda(x10^{-2}})';
figure(2)
h2=heatmap(Itetable);
h2.XLabel='{\mu}^{-1}(x10^{-2})';
h2.YLabel='{\lambda(x10^{-2}})';
```

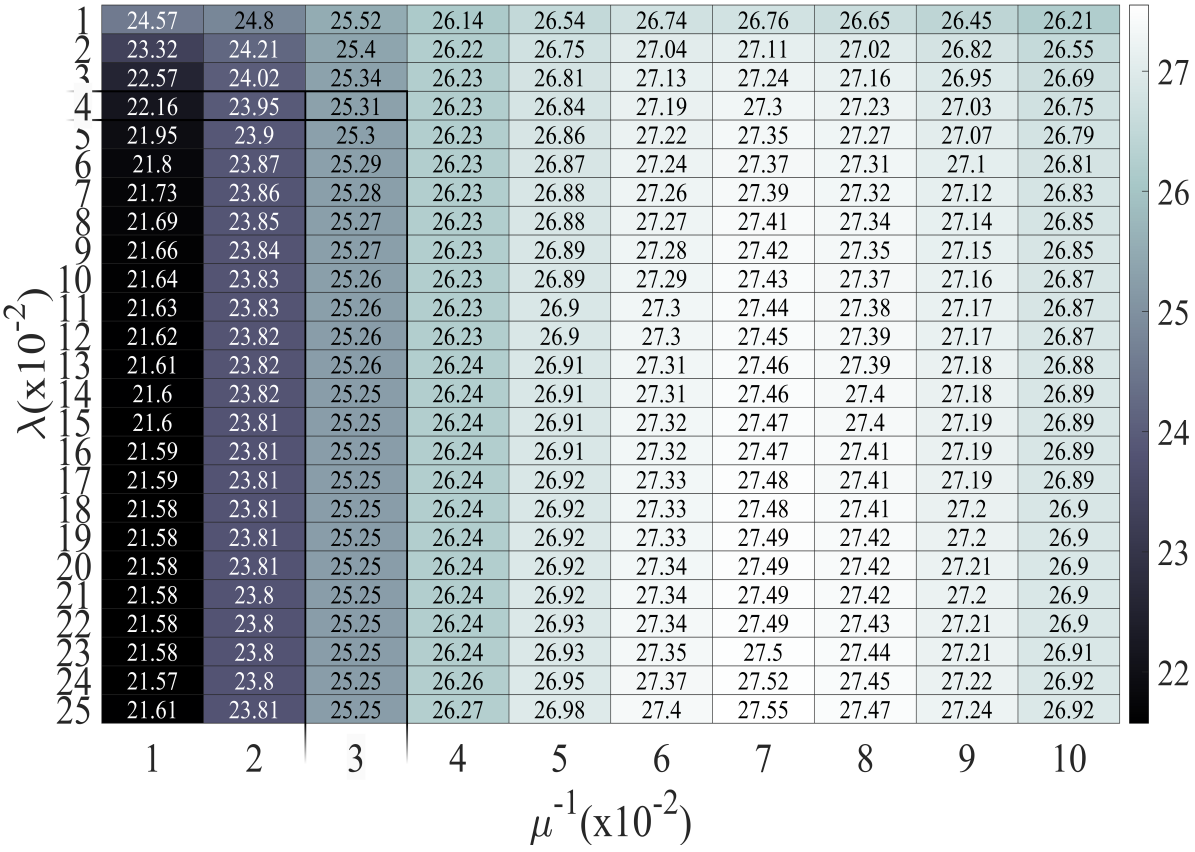


Figure 23: Heatmap of PSNR $\sigma = 25$

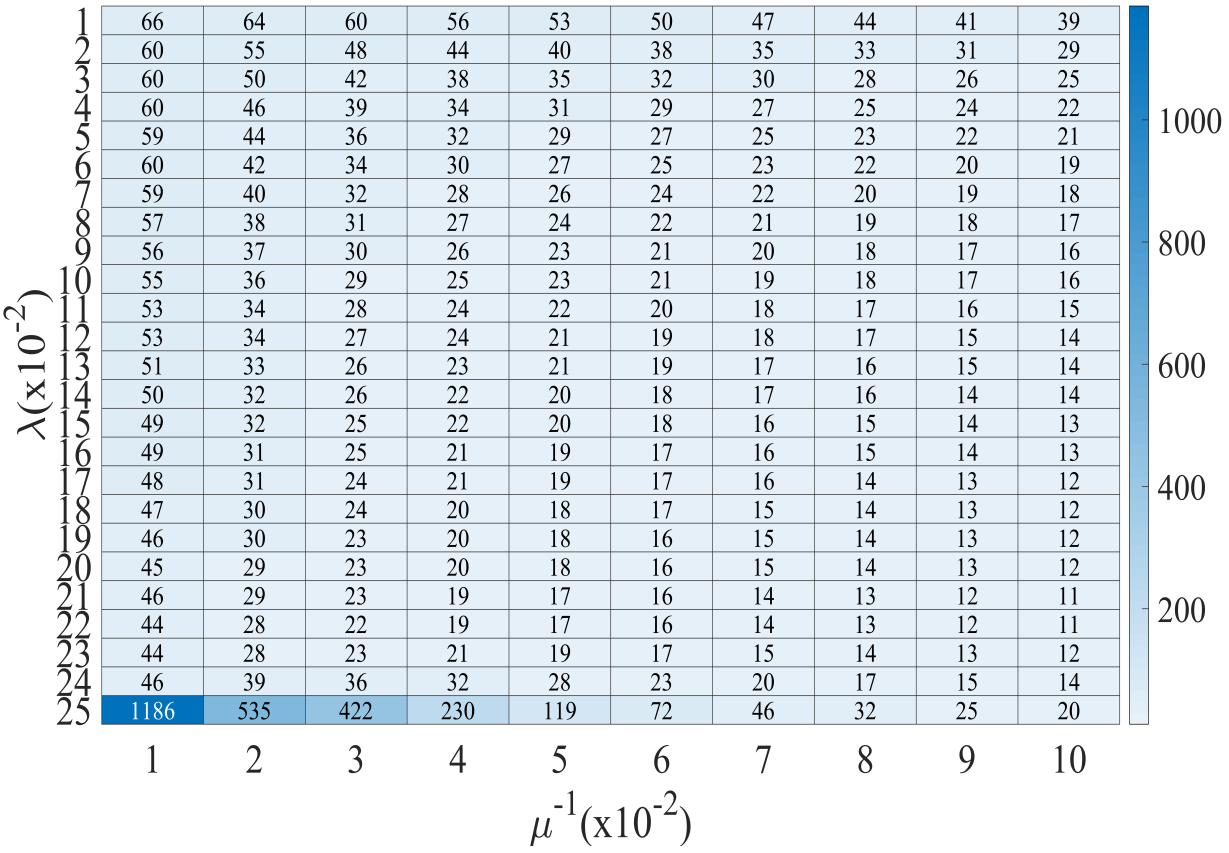


Figure 24: Heatmap of Iteration Times $\sigma = 25$

10.4 Codes and results of noise level $\sigma = 35$

```
Images=imread('Cameraman.png');
NoiseLevel=35;
sigma=(NoiseLevel*NoiseLevel)/(255*255);
Im_noisy=imnoise(Images,'gaussian',0,sigma);
draw=0;
PSNRtable=[];
Itetable=[];
for ii=1:10
mu=100/ii;
for lambda=0.01:0.01:0.25
[PSNR, Ite, Denoised_Image]=FPPAdraw...
(Images, Im_noisy, draw, lambda, mu);
PSNRtable=[PSNRtable,PSNR];
Itetable=[Itetable, Ite];
end
end
PSNRtable=reshape(PSNRtable,25,10);
Itetable=reshape(Itetable,25,10);
figure(1)
h1=heatmap(PSNRtable);
h1.Colormap=bone;
h1.CellLabelFormat='%0.4g';
h1.XLabel='{\mu}^{-1}(x10^{-2})';
h1.YLabel='{\lambda(x10^{-2}})';
figure(2)
h2=heatmap(Itetable);
h2.XLabel='{\mu}^{-1}(x10^{-2})';
h2.YLabel='{\lambda(x10^{-2}})';
```

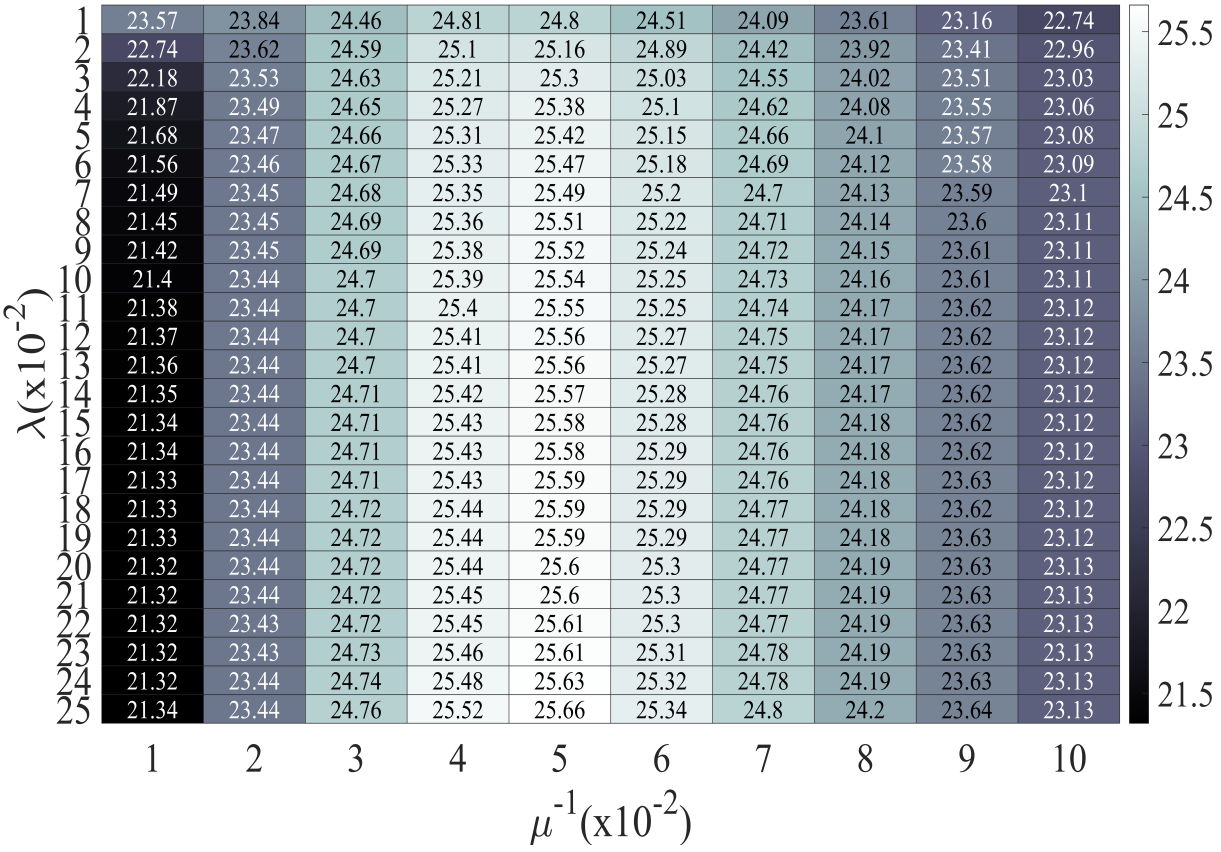


Figure 25: Heatmap of PSNR $\sigma = 35$

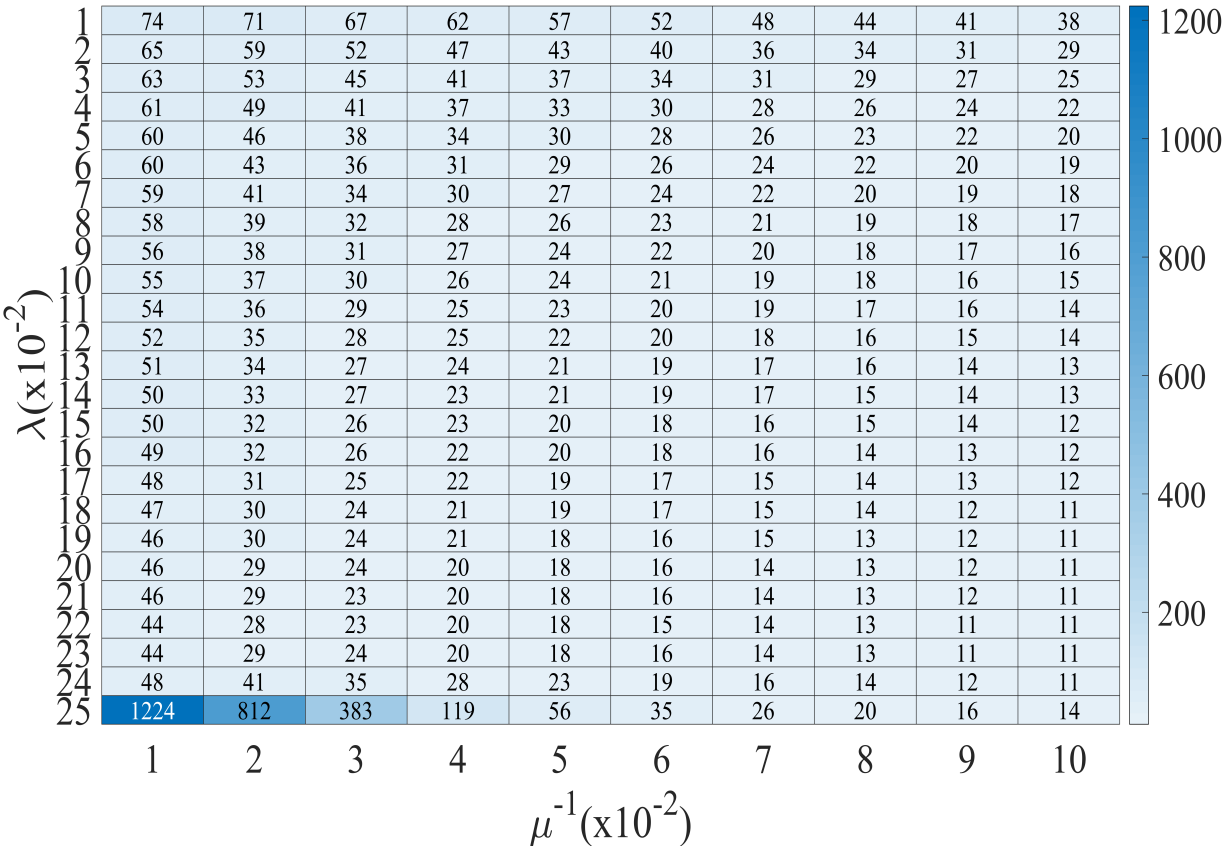


Figure 26: Heatmap of Iteration Times $\sigma = 35$