AMATH582 HW3 IMAGE PROCESSING AND ANALYSIS TO SAVING DEREK ZOOLANDER

SHUANG WU

Feb. 4th

Abstract

Images or photos are ubiquitous, but how can we recognize a bad quality image that is hard to identify? The way I deal with this problem is by using named image processing and analysis. In this project, I will use Fourier analysis and diffusion to try to make the bad quality image looks clearer.

1 INTRODUCTION

There are a variety kinds of images. Ultrasound, CAT scans, Radar imaging are some typical kinds of images in medical or physical area. Digital photos are the most familiar images, which we can easily make by devices such as our cellphones. Although images are easy to access, when at the first glance, especially in the medical and physical area, the images may have noises or blurs that make the information we look for hard to find. De-noise and de-blurring will be applied to make the image better and some mathematical method will be introduced to show the theoretical way of improving the image quality.

2 THEORETICAL BACKGROUND

2.1 *Fourier analysis and linear filter*

Like the projects I have already done before, I have some data sets at first, and I would like to transfer the data into Fourier domain. In the Fourier domain, I can use the old fashioned method, Gaussian filter. This is the method of how to denoise the data. In the following analysis, denoise is also the majority part or the most common thing I need do.

Thus, as far as I have my Fourier domain data set, I use the Gaussian filter:

$$F(k_x, k_y) = e^{(-\sigma_x(k_x-a)^2-\sigma_y(k_y-b)^2)}$$

where σ_x and σ_y are the filter widths in the x and y directions, and a and b are the center frequencies of the image. For example, if an image is 600*800 pixel, the center frequencies will be 301 and 401. Also, in convenience, I assume that σ_x and σ_y are equal to each other.

Having the filter ready, I multiple the filter with the data set to find the data after filtering. The key factor to make the image quality better is to find the best σ (the width of the filter), and the type of the filter. In this project, I will only apply the Gaussian filter, which is the most simple one.

2.2 *Diffusion and image processing*

Filtering is not the only method to clean the image. Diffusion can also be useful for cleanness.

It is hard to demonstrate these process by equations or methodology. I will introduce the basic equations here and describe the methods more specific in the next section.

Hear equation:

$$u_t = D\nabla^2 u$$

for

$$\nabla^2 = \partial_x^2 + \partial_y^2$$

Fourier transform:

$$\hat{u}_t = -D(k_x^2 + k_y^2)\hat{u} \rightarrow \hat{u} = \hat{u}_0 e^{-D(k_x^2+k_y^2)t}$$

Modified hear equation:

$$u_t = \nabla \cdot (D(x,y)\nabla u)$$

where $D(x,y)$ is a spatial diffusion coefficient.

Second-order scheme:

$$\frac{\partial^2 u}{\partial x^2} = \frac{1}{\Delta x^2}[u(x+\Delta x, y) - 2u(x,y) + u(x-\Delta x, y)]$$

$$\frac{\partial^2 u}{\partial y^2} = \frac{1}{\Delta y^2}[u(x, y+\Delta y) - 2u(x,y) + u(x, y-\Delta y)]$$

Vector $\vec{u}$:

$$\vec{u} = \begin{bmatrix} u_1 \\ u_2 \\ \cdots \\ u_n \end{bmatrix}$$

Sparse matrix A:

$$A = \begin{bmatrix} -2 & 1 & 0 & \cdots & 0 & 1 \\ 1 & -2 & 1 & 0 & \cdots & 0 \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & \cdots & 0 & 1 & -2 & 1 \\ 1 & 0 & \cdots & 0 & 1 & -2 \end{bmatrix}$$

2-D Diffusion:

$$\frac{d\vec{u}}{dt} = \frac{k}{\delta^2}A\vec{u}$$

3 ALGORITHM IMPLEMENTATION

3.1 *Black and White image with filter*

1. Load image and transfer the data.

   The first step is using the **imread** in `Matlab` to load my image. After loading the image, I examine the workspace to find the structure that the image was named, which is `Uint8`. It is a special format for saving the image. In order to do the Fourier and the following analysis, I transfer the `Unit8` format to double precision with the `Matlab` command **double**. Also, because I am dealing with a black and white image, the matrix is 2-Dimensional. For a color image, it will be in 3-D, and I will talk more about this case later.

2. Set up the parameters.

   Because I have the image in 2-D, the problem is solving in 2-D instead of the 1-D problems in previous projects. The **Meshgrid** command can be used to generate the 2-D wavenumbers in both x and y directions. Thus, I first set the sequence for k_x and k_y, and then used `meshgrid` for both of them to make them 2D. One thing need to notice is that the k_x is the number that shows in y direction for the pixel, and vice versa. For example, if I have a image that is 600*800 pixel: $k_x = 1 : 800$ and $k_y = 1 : 600$. After applying `meshgrid`, I will have two new vectors K_x and K_y, which are almost the same as k_x and k_y, but in 2D. I will use these new vectors in the filter later.

3. Fourier transform.

   After setting up, I do the Fourier transform for the matrix. One thing need to be noticed here is that I do the 2D Fourier transform, **FFT2**.

4. Build filter.
   Constructing the Gaussian filter use the previous equation. The canter frequency is 181 for x direction and 127 for y direction, which are the values of k_x or k_y plus one divided by 2. Also, I use the K_x and K_y when build the filter to make sure I have filter in 2D.

5. Apply the filter and transfer back.

   Use the filter times the data after transformation and shift. The most important thing I need to do **fftshift** for the data before I multiply by the filter is to make sure they are in the same position. After this step, use inverse Fourier to transform the data back with **ifft2**.

6. Show the image and frequency plot.

   After I did the first time Fourier transform, I can plot the frequency plot that will show the center frequency directly by visualization. Doing the filter and inverse transformation, I want to know if my filter works and want to show the final plot. Using **imshow** command will show the plot, but before showing, the plot I need transfer the format back to `Uint8`, which is in double format now. I use **Uint8** command to transfer the format back and with the command, **imshow** I can easily see what happens after filtering the image.

3.2 *Color image with filter*

The only different between color and black-white image is the size of matrix. For example, a 600*800 pixel image has 600*800 data points for black-white image. However, the color image will have 600*800*3 data points. The number 3 in the color image means red, green and blue (RGB). Therefore, for the color image, it has the same structure for each different color domain, and combining the 3 color domain will give us the color image.

Thus, when I want to apply the filter method into color image, I just need to create a **forloop**, which will analyze each color domain at one time and combine all three at the end of the loop.

3.3 *Black and White image with Diffusion*

1. Load the image and find the right area to do the diffusion.

   Loading the image like what I have done in the previous part. Looking at the image, I find that I only need to do a small part instead of doing the whole part. So I plot the original image and zoom in to find the right number for x and the direction to figure out the only part that I need to solve, which is 135:165 for the x direction and 155:185 for the y direction.

2. Set up the parameter.

   Set the size to the value corresponding to the area I will deal with. Find the right time rate and diffusion rate as $tspan$ and D. It is hard to get the exact right values for the first time, but by repeating the step and changing the number for multiple times, can to find the suitable choices.

3. Set up the matrix A and reshape the vector u.

   Using **spdiags** to set up the matrix A shown in the previous section. And then use **kron** to set the big matrix L. Also, as what I have showen in the previous section, my vector u can be viewed as an n by 1 matrix. Here, the data from image is the same as the initial condition value, which is just the vector u. Thus, in order to use this as my initial value, I need put the values **reshape** into and n by 1 matrix.

4. Solve the ordinary differential equation.

   Use **ode113** or **ode45** to solve this equation. Before solving it, I need first create a function file that has the right hand side of my function $D * L * u$. With the function and all the parameters ready, solve the function directly with `matlab` command.

5. Show the solution.

   After getting the solution, which is after diffusion, select the best area, because there is a black frame outside the plot. Then, transfer the format back to `Uint8` and make plot like I have done before to compare the differences.

3.4 *Color image with Diffusion*

Almost like what I have done for the color with filter before, create a big `for` loop to deal with each color domain separately. At the end of the `for` loop, combine all the color domains together.

4 COMPUTATION RESULTS

4.1 *Black and White image with filter*

First, I would like to compare the original image with the image after filter and then talk about the improvements in detail.

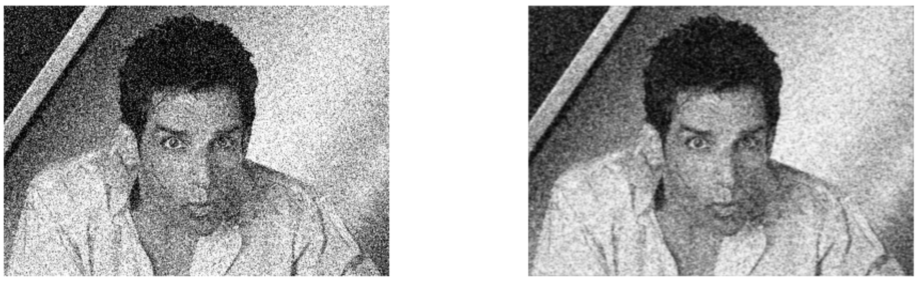
Figure 1: Left: image before filter. Right: image after filter.

In Figure1, maybe the image is still not clear enough even after filtering (the right image), it is clearer than before (the left image). Although the clearness of the image becomes better, I have another issue: blurring for the new image at the right. This problem always happens. So, when I denoise a image with many noises, I can make it clearer, but the image becomes blur. Thus, when I want to denoise an image, I need first decide what detail(s) I want to know from the image, and just denoise the specific area for the desired detail(s).

When doing the filter for the image, I also made the frequency domain plot:

The Figure2 shows a very well plot, which has a very bright point in the middle. In fact, this point is my central frequency point. If I zoom in a lot, I can find that the central frequency is at the point (181,127) for x and y direction.

After getting the central frequency, I build the Gaussian filter to filter the plot. After filtering, I use inverse Fourier to transform the image back and show, which is the right plot in the Figure 1.

4.2 *Color image with filter*

As I mentioned in the previous algorithm implementation section, dealing with a color image is almost the same like what I have done for the black and white image. The only difference is, for the color image, I need to use a `for` loop to filter the image in each color domain: red, green and blue. So I can regard dealing with color image is same as dealing with black and white image for 3 times.

Look at the Figure3, the result is just like Figure1 that is shown in the page before. I have clearer image but with blurring.

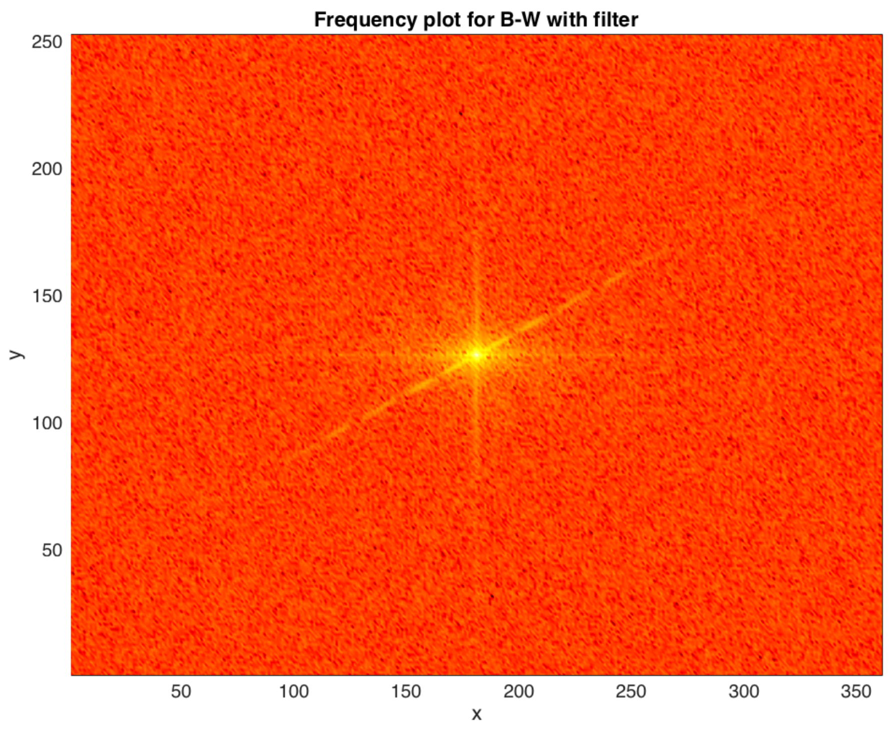

Figure 2: Plot for the image in frequency domain

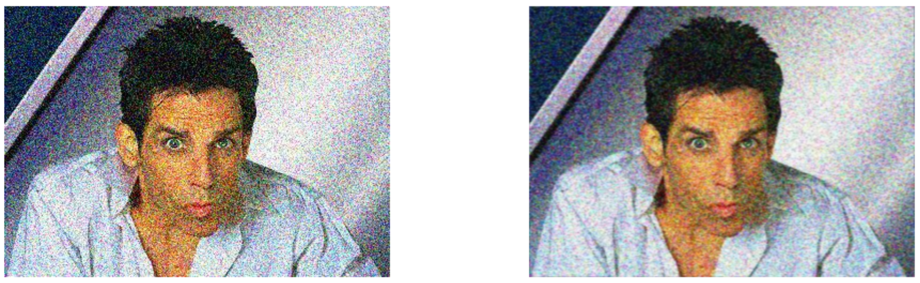

Figure 3: Left: image before filter. Right: image after filter.

Also, while doing the filter for each color domain, I plot the frequency domain plot to make sure if the central frequency will always be the same for each domain. From the Figure4, it's clearly that the answer is yes.

One more thing need to be mentioned for both black-white and color filter processing, the parameter σ, which is the width of the filter window will effectively affect the clearness of the image. When I have a very small σ, I will have a wide window. Because in my equation, I have $e^{-\sigma}$. The wider the window I have, the unclear the image comes out after filter. But if I make the window narrower, I will have a more blur image.

This is very similar like I have done in the previous project that, when using the Gabor method, I had a trade off between the information of time and the information of the frequency. In this problem, I have to sacrifice the clearer image for the less blur, or reduce the noises at the cost of more blur.

The best way will always involve finding the best parameter.

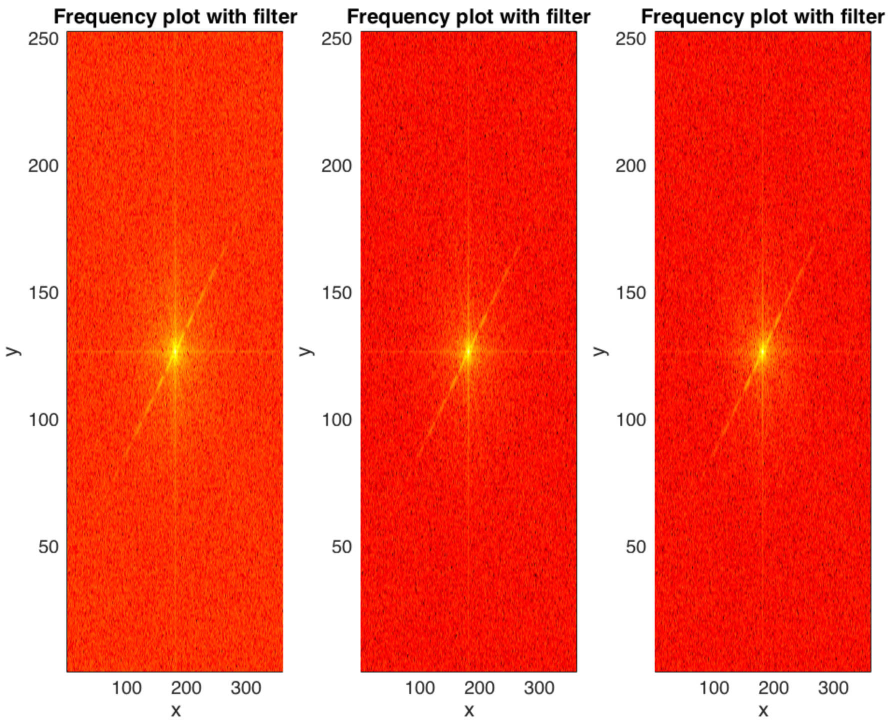

Figure 4: Frequency plot for color image with filter

4.3 *Black and White image with diffusion*

This part is more difficult than the previous part. Look at Figure 5:

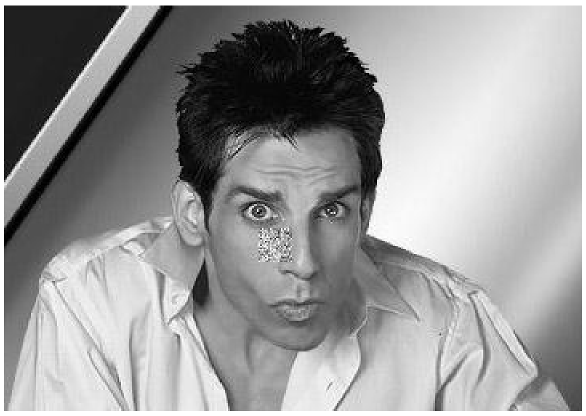

Figure 5: Frequency plot for color image with filter

It is easy to notice that there is a rash near his nose, otherwise, the image looks good. So, I need to select out the specific area to do the diffusion instead of doing the diffusion for the whole image.

I use the old fashioned way of plotting and zooming in until I get a small square that includes all the rash with some areas outside the rash. After repeating several times of zooming and plotting, I find that the specific area is (135:165, 155:185) for x and y directions. And the Figure 6 shows the image for how that specific area looks like.

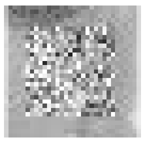

Figure 6: Selected area for the rush image.

After finding the area I apply the diffusion method into this area, and plot the result for the specific area for different time rate, tspan, which is in Figure 7.

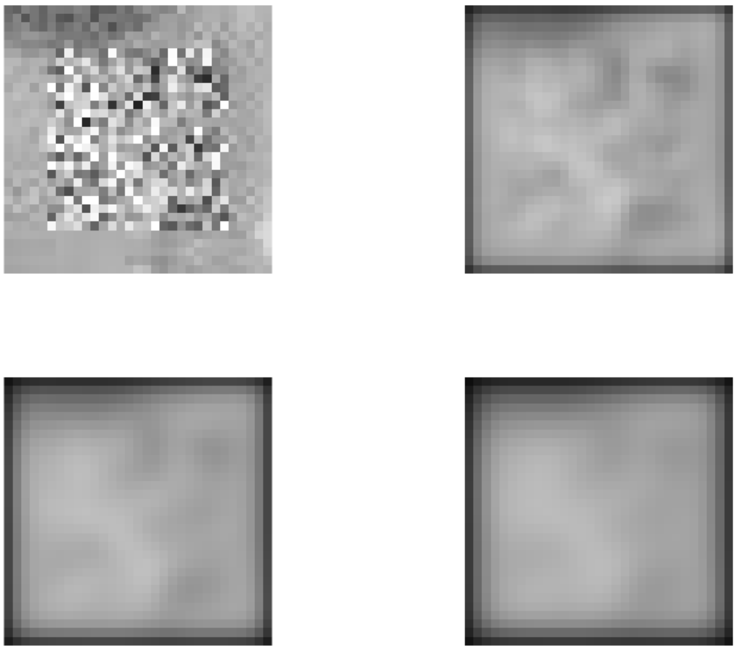

Figure 7: Selected area after diffusion for different time rates with D, diffusion coefficient = 0.5. The top left is tspan = 0. The top right is tspan = 0.002. The bottom left is tspan = 0.004. The bottom right is tspan = 0.006

I would like to choose the last one, because it's almost like the person's skin. Also, I notice that there is a black window outside the image, which means I only need to replace the middle area with the original plot instead of replacing the whole image. This is also the reason when I select the specific area, I select a larger area than the actual rash area
.

After replacing the new image and plotting. Look at the Figure 8, I think I can use the word 'perfect'. If you do not look that the image really carefully, or zoom in the noise to see that part only, it is hard to find that there is a small square in that rash area. Compare this plot with the original plot, I use the diffusion to make the image looks good.

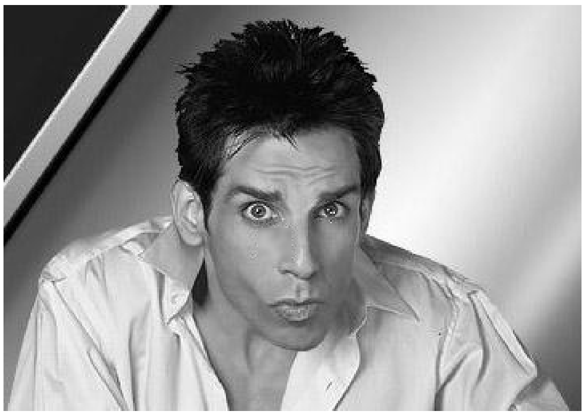

Figure 8: Black and white image after diffusion

4.4 *Color image with diffusion*

There is no surprise that the color image requires the same method as the same as we have done for the filter one. Repeat the diffusion for 3 times for each of the color domain and combine them together to get the plot.
Here, I will only show the image before diffusion and after diffusion, Figure 9, to show how this method works perfectly. Because the plot or steps are identity as before.

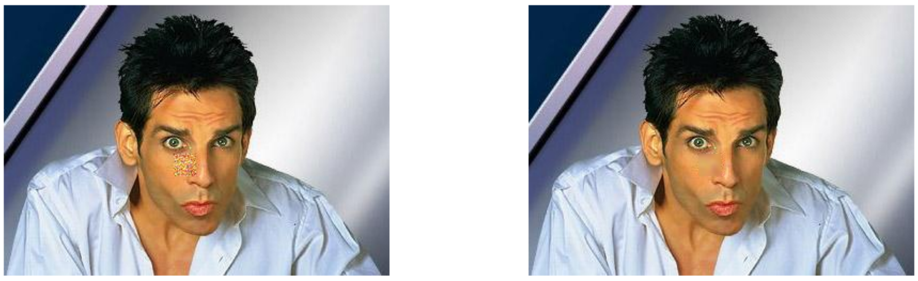

Figure 9: Right: Color image before diffusion. Left: Color image after diffusion.

4.5 *Additional important thing found - VERY IMPORTANT*

After finishing the report, I go through the discussion board and find that there is a way to make the black window disappear. Because the report has already been finished when I see the post, and I do not want to rewrite the report. So I will just give some ideas and demonstrate how it work.

The black window appears because of the boundary conditions. Let's look at the Figure 10 first.

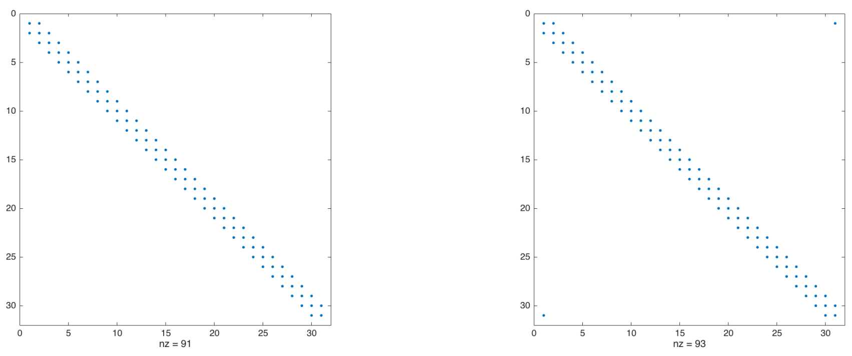

Figure 10: Left: spdiag matrix. Right: spdiag matrix with period B.C.

Compare these two plots carefully, we will observe that the right plot has two more dots than the left plot. The two dots, in fact, affect the black window. Also, the two plots are the structure plots for the sparse matrix. When I first create this matrix, I only have 2 values for the first and last row, which means if I look for the n-1 value for the first row, for n is the value at (1,1), is missed. But if I make the matrix period condition, which means the last value of the first column is the same as the value on the right of the value at (1,1), I directly make the boundary period instead of missing value. The same theory is applied for the last row. After making the boundary period conditions, there is no more black window, because there is no missing value at all. All the values then can be evaluated when doing the **ode113** or **ode45**.

Figure 11 shows the new after-diffuse plot for the rush area for different tspan.

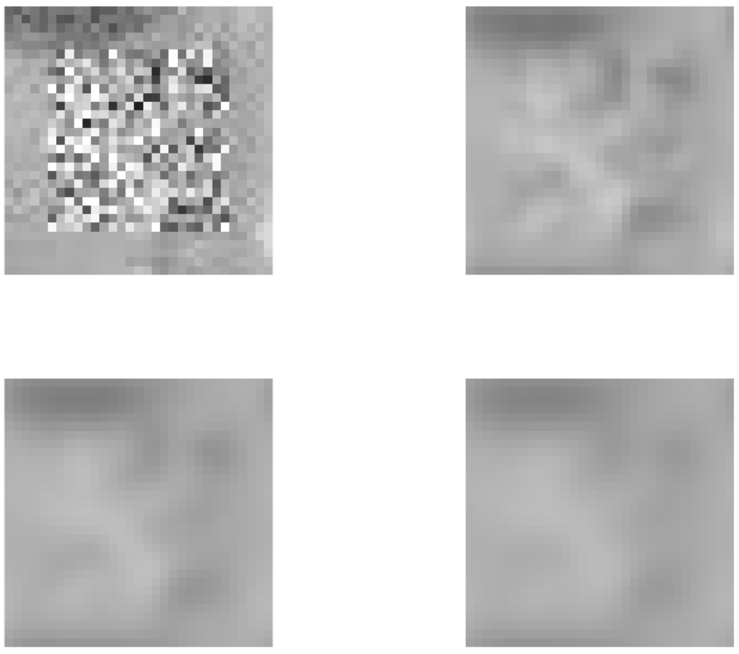

Figure 11: Selected area after diffusion for different time rate with D, diffusion coefficient =0.5. The top left is tspan = 0. The top right is tspan = 0.002. The bottom left is tspan = 0.004. The bottom right is tspan = 0.006, with period B.C.

There is no black window anymore, I can replace this into my original image directly. But I select the area which is actually larger than the exact rush area, so when I replace the new image to the old one, I also need to select the internal area. otherwise, I will have a blur square in the rash area which is also observable. If I select very small rush area, which is just the rash area itself, then I can diffuse that area and replace it direclty.

5 SUMMARY AND CONCLUSION

In this project, I know using the Fourier transform along with filter will denoise some of the noises in an image. But if I have a bunch of noises, it may be hard to denoise. Also, the width of the filter window and type of the filter will affect the result image a lot. There is no right or wrong filter or window width for the image, but only the best fit filter or window width. Thus, depending on the specific application, we choose the specific filter and window width. Also, just like the Gabor transform, there is a trade off between noise and blurring. Less noises will cause more blurring and less blurring will cause more noises.

For the diffusion part, it looks like more useful than the filter, but in fact, it is a different method to use in the specific application. Also, the diffusion method is harder because it requires to solve the ordinary differential equation even using `Matlab`. By the way, diffusion may not be applied to a whole image.

In conclusion, there are many other methods that been used to do the image analysis. There is no best method. The only thing I can do is applying the best fit method to a specific image problem.

6 APPENDIX I

Functions Used and Brief Implementation Explanations

1. `imread`: load the image into matlab.
2. `meshgrid`: create multiple dimention grid.
3. `fft2`: Fourier transform in 2D.
4. `fftshift`: Fourier transform postition shift.
5. `ifft2`: Inverse fourier transform in 2D.
6. `uint8`: transform into uint8 format, format of a special image.
7. `double`: Transform into double preciese format.
8. `pcolor`: Pseudocolor (checkerboard) plot.
9. `shading`: Color shading mode.
10. `size`: get the size of a vector or matrix.
11. `linspace`: create the linear sequence.
12. `ones`: Create a matrix fill with 1.
13. `spdiags`: Sparse matrix formed from diagonals.
14. `eye`: Identity matrix.
15. `kron`: The result is a large matrix formed by taking all possible products between the elements of X and those of Y.[1]
16. `ode113`: Solve non-stiff differential equations, variable order method.[2]
17. `ode45`: Solve non-stiff differential equations, medium order method.[3]

[1] Copy from Matlab
[2] Copy from Matlab
[3] Copy from Matlab

7 APPENDIX II

MATLAB codes

7.1 *Black and White image with filter*

```
        clear all; close all; clc
A=imread('derek2','jpeg'); %load image
A2=double(A);

kx=1:361; ky=1:253;
[Kx,Ky]=meshgrid(kx,ky);

At=fft2(A2);

sigma=0.0001;
F=exp(-sigma*(Kx-181).^2-sigma*(Ky-127).^2);
Ft=fftshift(F);
Atf=Ft.*At;
At2=ifft2(Atf);
Af=uint8(At2);
figure(2),imshow(A)
figure(1),pcolor(log(abs(fftshift(At))+1)), shading interp, colormap(hot)
figure(3),imshow(Af)
```

7.2 *Color image with filter*

```
        clear all; close all; clc
A=imread('derek1','jpeg'); %load image
kx=1:361; ky=1:253;
[Kx,Ky]=meshgrid(kx,ky);

Anew=[];
for j=1:3
Atemp=A(:,:,j);
At=fft2(Atemp);
sigma=0.0001;
F=exp(-sigma*(Kx-181).^2-sigma*(Ky-127).^2);
figure(j),pcolor(log(abs(fftshift(At))+1)), shading interp, colormap(hot)
Ft=fftshift(F);
Atf=Ft.*At;
At2=ifft2(Atf);
Anew(:,:,j)=At2;
end
Anew=uint8(Anew);
figure(4),imshow(A)
figure(5),imshow(Anew)
```

7.3 *Black and White image with diffusion*

```
        clear all; close all; clc;

A=imread('derek4','jpg');
Abw=double(A);

A2=Abw(135:165,155:185);
```

```
[nx,ny]=size(A2);
%  imshow(A)

x=linspace(0,1,nx);  dx=x(2)-x(1);
y=linspace(0,1,ny);  dy=y(2)-y(1);
onex=ones(nx,1); oney=ones(ny,1);
Dx=(spdiags([onex -2*onex onex],[-1 0 1],nx,nx))/dx^2;
Ix=eye(nx);
Dy=(spdiags([oney -2*oney oney],[-1 0 1],ny,ny))/dy^2;
Iy=eye(ny);
L=kron(Iy,Dx)+kron(Dy,Ix);

tspan=[0 0.002 0.004 0.006]; D=0.5;

u=reshape(A2,nx*ny,1);
[t,usol]=ode113('image_rhs',tspan,u,[],L,D);
figure(1)
for j=1:length(t)
    Abw_clean=uint8(reshape(usol(j,:),nx,ny));
    subplot(2,2,j), imshow(Abw_clean);
end

A(140:160,160:180)=uint8(Abw_clean(6:26,6:26));
figure(2)
imshow(uint8(A))
```

7.4 *Color image with diffusion*

```
    clear all; close all; clc;

A=imread('derek3','jpg');
imshow(A)

Abw=double(A);
A1=Abw(135:165,155:185,:);
[nx,ny,nz]=size(A1);

x=linspace(0,1,nx);  dx=x(2)-x(1);
y=linspace(0,1,ny);  dy=y(2)-y(1);
onex=ones(nx,1); oney=ones(ny,1);
Dx=(spdiags([onex -2*onex onex],[-1 0 1],nx,nx))/dx^2;
Ix=eye(nx);
Dy=(spdiags([oney -2*oney oney],[-1 0 1],ny,ny))/dy^2;
Iy=eye(ny);
L=kron(Iy,Dx)+kron(Dy,Ix);

tspan=[0 0.002 0.004 0.006]; D=0.5;

for j=1:3
A2=A1(:,:,j);
u=reshape(A2,nx*ny,1);
[t,usol]=ode113('image_rhs',tspan,u,[],L,D);
figure(j)
for k=1:length(t)
    Abw_clean=uint8(reshape(usol(k,:),nx,ny));
    subplot(2,2,k), imshow(Abw_clean);
end
```

```
A(140:160,160:180,j)=uint8(Abw_clean(6:26,6:26));

end

figure(4)
imshow(uint8(A))
```

7.5 *Right Hand Side(RHS) function*

```
    function rhs = image_rhs(t,A,dummy,L,D)
rhs=D*L*A;
```

7.6 *Code for B.C*

```
    Dx=(spdiags([onex -2*onex onex],[-1 0 1],nx,nx))/dx^2;
Dx(1,31)=1/dx^2;
Dx(31,1)=1/dx^2;
Ix=eye(nx);
Dy=(spdiags([oney -2*oney oney],[-1 0 1],ny,ny))/dy^2;
Dy(1,31)=1/dy^2;
Dy(31,1)=1/dy^2;
```