Contrast Stretching

1. Read in the file

```
>>Pc=imread('mrt-train.jpg');
```

2.Show Pc's properties

```
>>whos Pc

Name        Size              Bytes   Class    Attributes
 Pc       320x443x3            425280  uint8
```

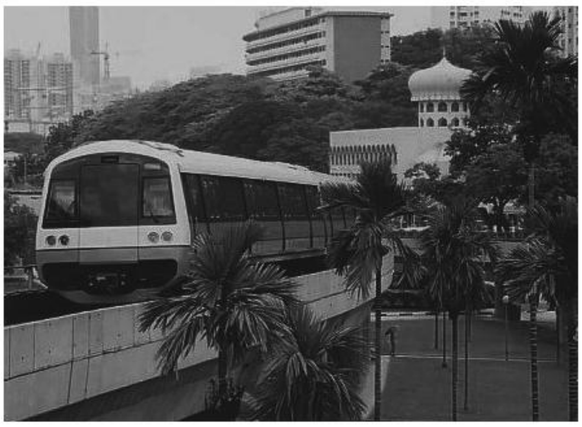
The size is three-dimensional, it means this picture is an RGB image, therefore we need to convert it to gray level.

3.Convert to grayscale image

```
>>P=rgb2gray(Pc);
```

4.Show the image

```
>>imshow(P);
```



5.Check the minimum and maximum intensities in the image:

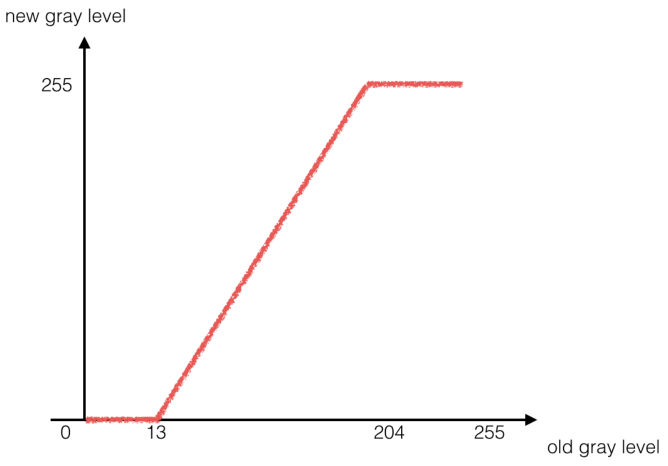
```
>>min(P(:))
   14
>>max(P(:))
   204
```

6. Write *two lines* of MATLAB code to do contrast stretching

```
    >>Pd=double(P);
    >>Pdn=(Pd-13)*((255-0)/(204-13))+0;
```

newvalue=(oldvalue-oldmin)((newmax-newmin)/(oldmax-oldmin))+newmin*

This formula maps 13 to 0 and 204 to 255 and number between 13 and 204 will be stretched linearly.



7.Covert P to uint8 and show the image

```
    >>Pn=uint8(Pdn);
    >>imshow(Pn);
```

9.Check the minimum and maximum again

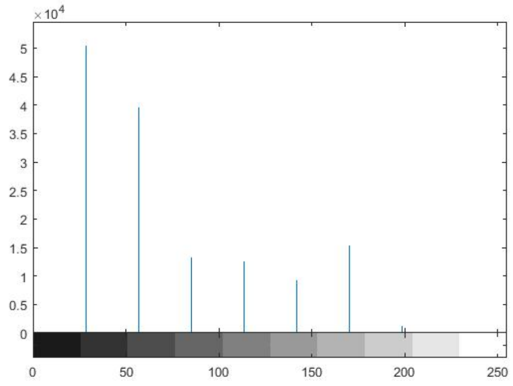
```
>>min(Pn(:))
  0
>>max(Pn(:))
  255
```

Histogram Equalization

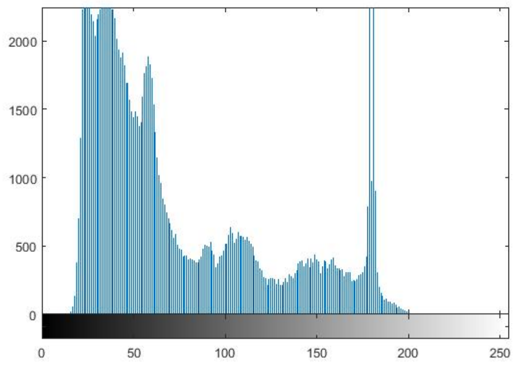
1. Display the image intensity histogram of P using 10 bins and 256 bins

```
>>imhist(P,10);
>>imhist(P,256);
```

histogram with 10 bins



histogram with 256 bins



What are the differences?

We can see more details about the distribution of different gray level in 256 bins histogram than in 10 bins histogram. Actually in 256 bins histogram we can obtain the intensity of every gray level, since there are 256 gray levels in a grayscale picture and

256 bins in the histogram. Between 150 and 200 there are some gray levels with large intensity which can only be observed in 256 bins histogram.

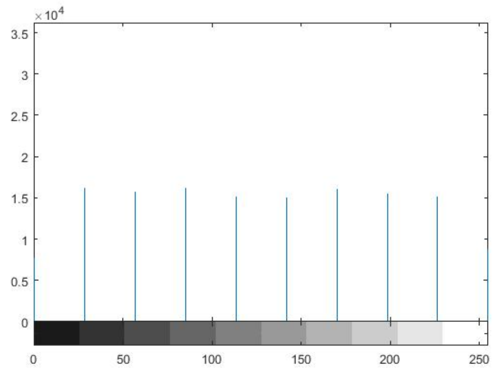
2. Carry out histogram equalization

```
    >>P3 = histeq(P,255);
```

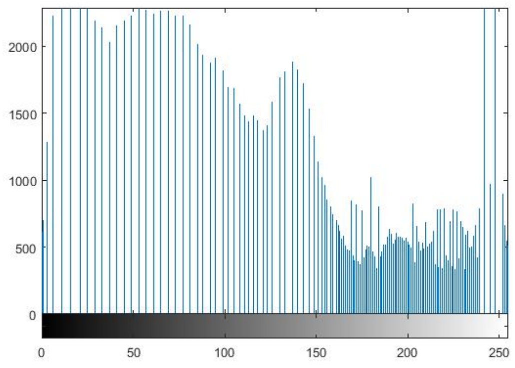
This transforms the intensity image P, returning in P3 an intensity image with 255 discrete gray levels. A roughly equal number of pixels is mapped to each of the 255 levels in P3, so that the histogram of P3 is approximately flat.

3. Redisplay the histograms for P3 with 10 and 256 bins

```
    >>imhist(P3,10);
```



```
    >>imhist(P3,255);
```



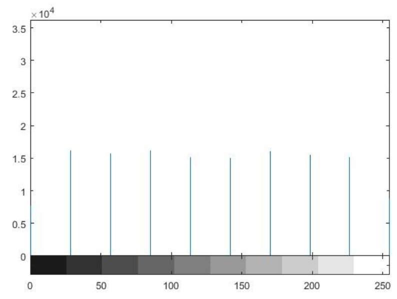
Are the histograms equalized? What are the similarities and differences between the latter two histograms?

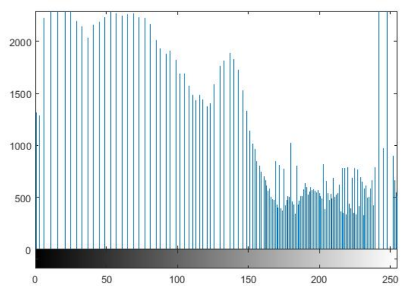
Both 10 bins and 256 bins histogram becomes flatter, 256 bins histogram becomes sparser, since some bins are moved and combined with other bins.

10 bins histogram looks quite flat, but 255 bins histogram still looks rugged. When we carry out histogram equalization, it first calculates the average pixel quantity in each bin. And then it tries to move the pixels in one bin to another bin in order to make all the bins has pixel quantity that close to average pixel quantity. When moving the pixels in one bin to another bin, it will move the pixels in the entire bin to another, it can not only move a part of the pixels in the bin, since it doesn't know how to decide which part of pixels in the same grey-level should be brighter or darker. That is the main reason why 255 bins histogram still looks so rugged, it is difficult to make every bin perfectly equalized by moving entire bin to another bin without dividing them. But if we examine the histogram in a larger scale, display it in 10 bins histogram, it looks fairly equalized.

4. Rerun the histogram equalization on P3

```
>>P3 = histeq(P3,255);
>>imhist(P3,10);
>>imhist(P3,256);
```





Does the histogram become more uniform? Give suggestions as to why this occurs.

It doesn't become more uniform. The result is only slightly different from the first result, since most of the bins has been moved to the right place after first histogram equalization. So it doesn't change a lot to rerun histogram equalization.

Linear Spatial Filtering

1. Generate the filter

   1.1 create function h1(sigma=1)and h2 (sigma=2)

```
>>syms x y

>>sigma=1;
>>h1(x,y)=(1/(2*pi*sigma*sigma))*exp(((x^2+y^2)/(2*sigm
a*sigma)));

>>sigma=2;
>>h2(x,y)=(1/(2*pi*sigma*sigma))*exp(((x^2+y^2)/(2*sigm
a*sigma)));
```

   1.2 Make h1 and h2 into two matrices G1 and G2 and normalize them

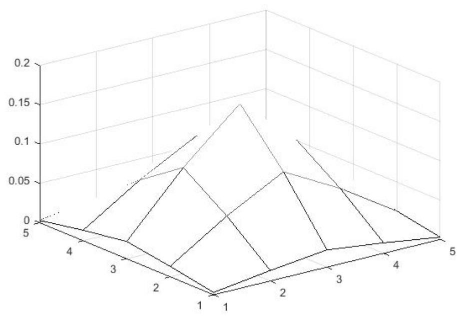
```
>>[X,Y] = meshgrid(-2:1:2, -2:1:2);
>>G1=h1(X,Y);
>>G1=h1(X,Y)/sum(G1(:));% normalization
>>s1=double(sum(G1(:))); %to check whether sum==1
>>G2=h2(X,Y);
>>G2=h2(X,Y)/sum(G2(:));
>>s2=double(sum(G2(:)));
>>G1d=double(G1);
>>G2d=double(G2)
G1:
0.0030    0.0133    0.0219    0.0133    0.0030
0.0133    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

G2:
0.0232    0.0338    0.0383    0.0338    0.0232
0.0338    0.0492    0.0558    0.0492    0.0338
0.0383    0.0558    0.0632    0.0558    0.0383
```

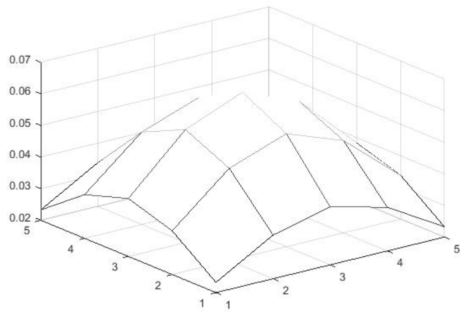
```
0.0338    0.0492    0.0558    0.0492    0.0338
0.0232    0.0338    0.0383    0.0338    0.0232
```

2.View G1 and G2 as 3D graphs

```
>>mesh(G1d);
>>mesh(G2d);
```

G1 G2

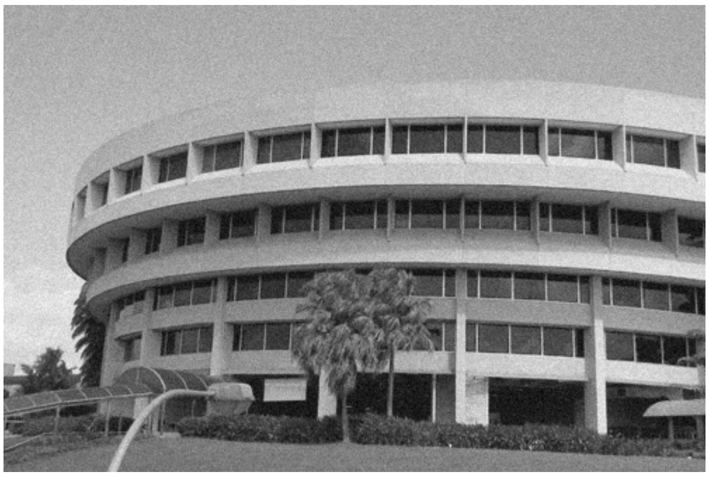




Sigma determines the deviation of the Gaussian averaging filter. Smaller sigma makes the center pixel in the filter more important.

3.Read in image with Gaussian noise

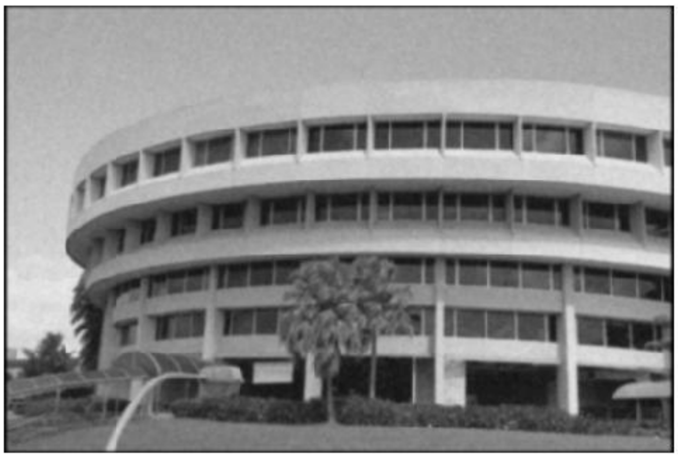
```
>>P=imread('ntu-gn.jpg');
>>imshow(P);
```



4.Apply Gaussian averaging filters on the image

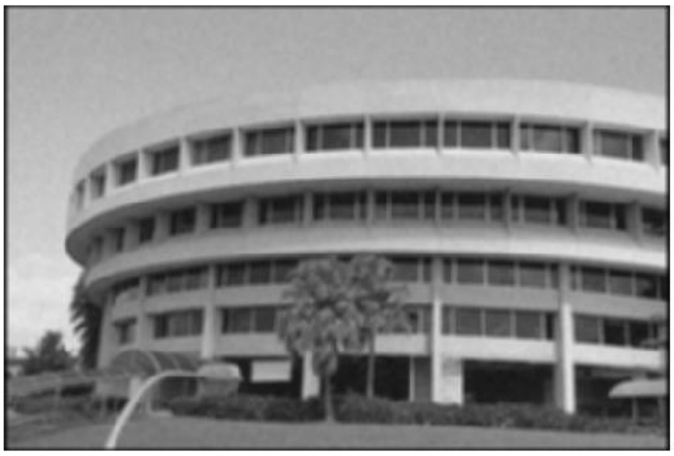
4.1 Apply G1(sigma=1)

```
>>Pd=double(P);
>>Pnd=conv2(G1d,Pd);
>>Pn=uint8(Pnd);
>>imshow(Pn);
```



4.2 apply G2 (sigma=2)

```
>>Pnd=conv2(G2d,Pd);
>>Pn=uint8(Pnd);
>>imshow(Pn);
```



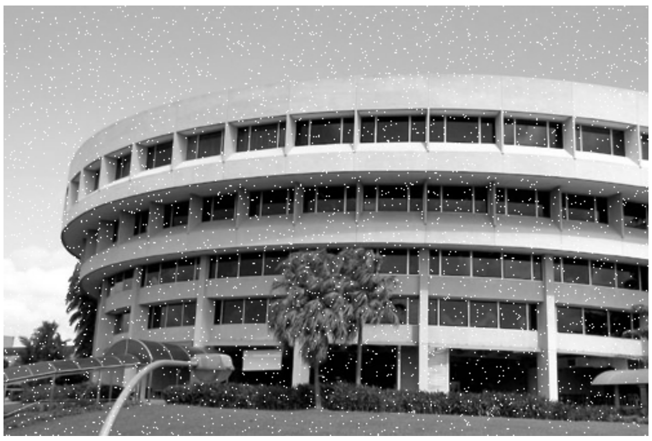
How effective are the filters in removing noise? What are the trade-offs between using either of the two filters, or not filtering the image at all?

After filtered by Gaussian averaging filter Gaussian noise becomes less obvious, but the images become blurred. Images filtered by G2 are more blurred than images filtered by G1.

For those images don't have too many edges, Gaussian averaging filter may be a good choice because we don't need to worry about the image would looks blurred after being filtered. But for this picture, I think the original picture looks better than the images filtered by Gaussian averaging filters.

5.Read in image with speckle noise

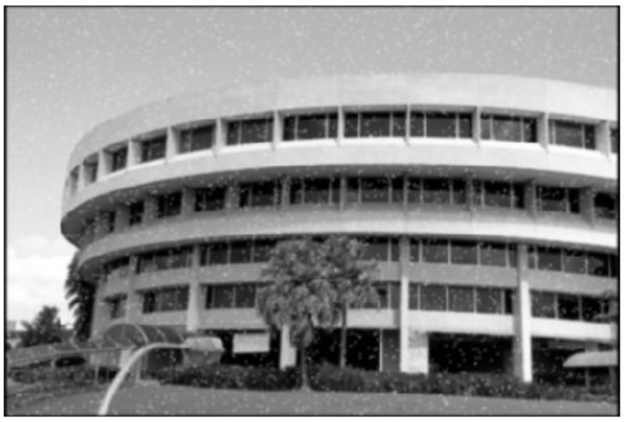
```
>>P=imread('ntu-sp.jpg');
>>imshow(P);
```



6. Apply Gaussian averaging filters on the image

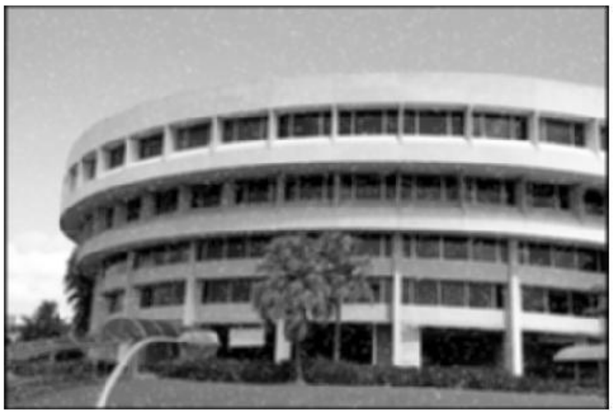
6.1 Apply G1 on it

```
>>Pd=double(P);
>>Pnd=conv2(G1d,Pd);
>>Pn=uint8(Pnd);
>>imshow(Pn);
```



6.2 Apply G2 on it

```
>>Pnd=conv2(G2d,Pd);
>>Pn=uint8(Pnd);
>>imshow(Pn);
```



Are the filters better at handling Gaussian noise or speckle noise?

Both of the images become blurred, but it seems that Gaussian averaging filter is not very effective in removing speckle noise. Although the images become blurred, speckles on it still can be seen. Gaussian averaging filters are better at handling Gaussian noise.

Median Filtering

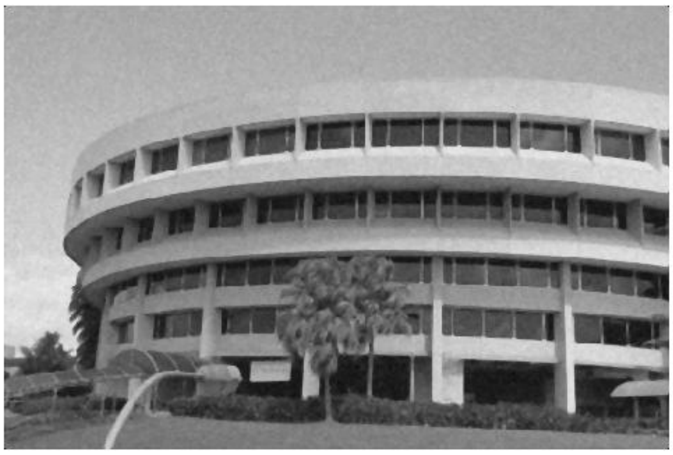
1. Apply median filters on the image with Gaussian noise

```
>>P=imread('ntu-gn.jpg');
>>imshow(P)
```

1.1 Apply 3x3 median filter

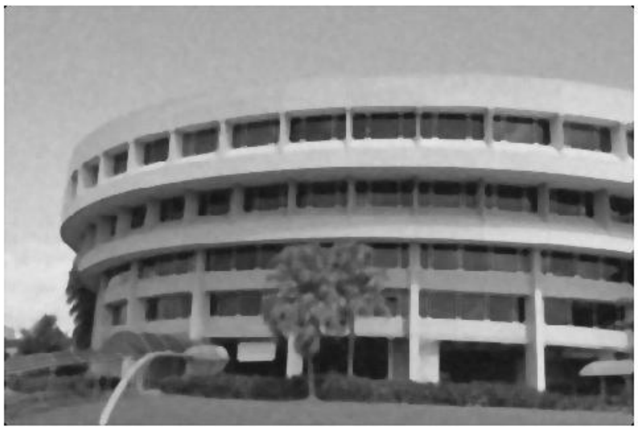
```
>>Pd=double(P);
>>Pnd=medfilt2(Pd, [3 3]);
>>Pn=uint8(Pnd);
```

```
>>imshow(Pn)
```



1.2 Apply 5x5 median filter

```
>>Pnd=medfilt2(Pd, [5 5]);
>>Pn=uint8(Pnd);
>>imshow(Pn)
```



2. Apply median filter on the image with speckle noise

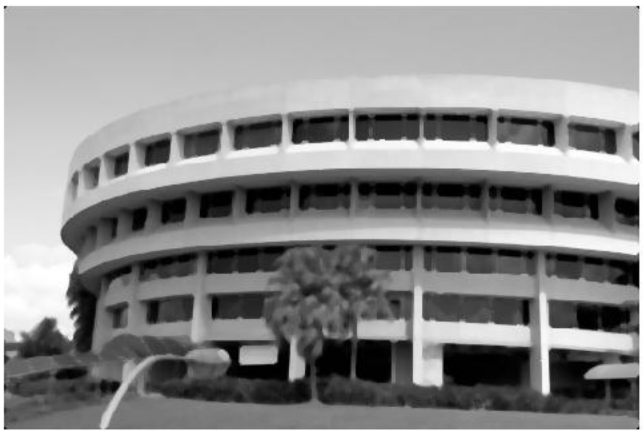
```
>>P=imread('ntu-sp.jpg');
>>imshow(P)
```

2.1 Apply 3x3 median filter

```
>>Pd=double(P);
>>Pnd=medfilt2(Pd, [3 3]);
>>Pn=uint8(Pnd);
>>imshow(Pn)
```

2.2 Apply 5x5 median filter

```
>>Pnd=medfilt2(Pd, [5 5]);
>>Pn=uint8(Pnd);
>>imshow(Pn)
```



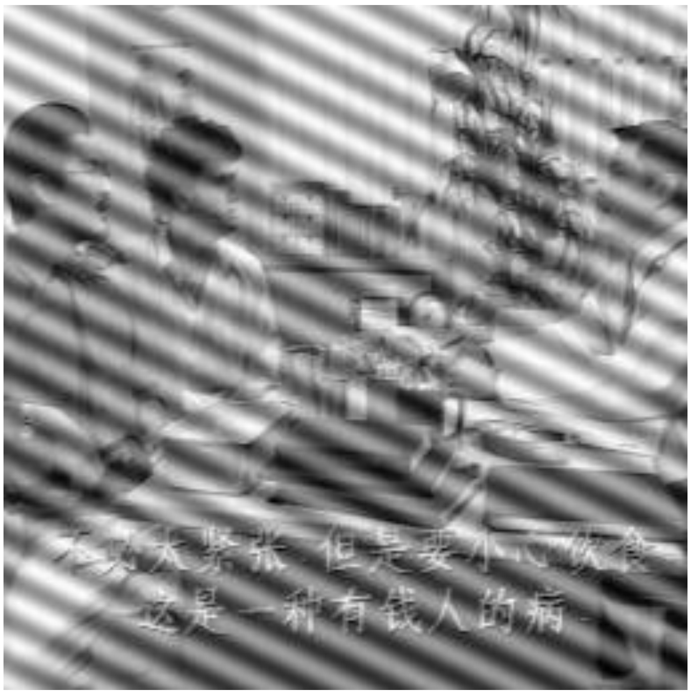
How does Gaussian filtering compare with median filtering in handling the different types of noise? What are the tradeoffs?

After filtered by median filter, Gaussian noise indeed becomes less obvious, but also the image becomes blurred. However, median filters are very effective in removing speckle noise. So it is better to use Gaussian averaging filter to filter image with Gaussian noise, and use median filter to filter image with speckle noise.

Suppressing Noise Interference Patterns

1.Read in image

```
>>P=imread('pck-int.jpg');
>>imshow(P);
```



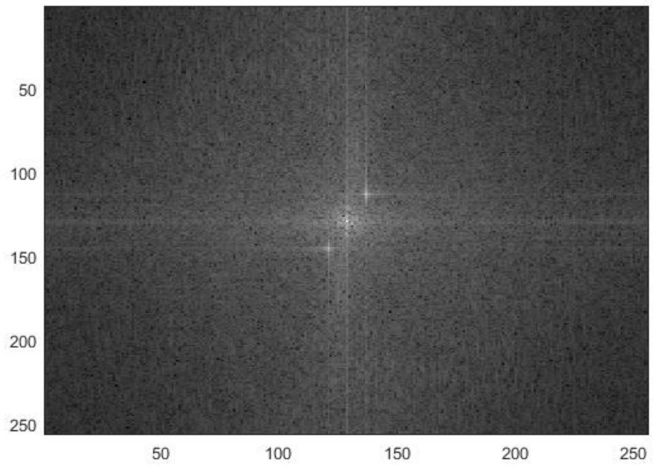
2.Obtain the Fourier transform and power spectrum

```
>>F=fft2(P);
>>S=abs(F);
```

Aboslute value of the spectrum represents the magnitude of each frequency.

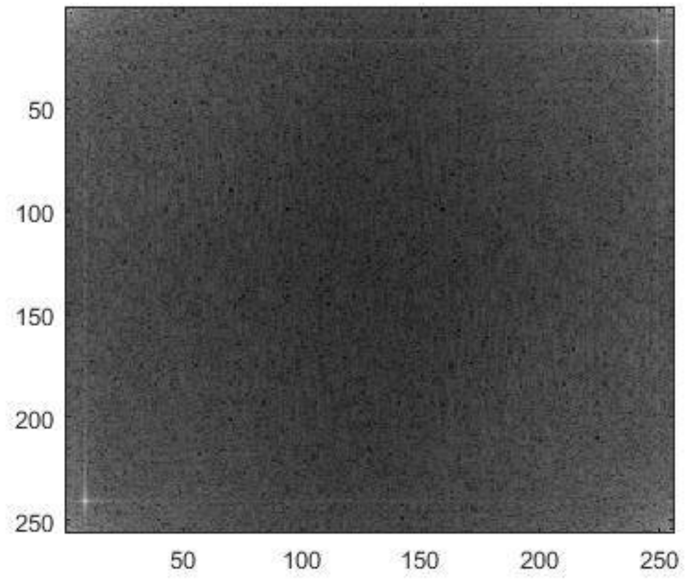
3.Display power spectrum

```
>>imagesc(fftshift(S.^0.1));
>>colormap('default'); >>F=fft2(P);
>>S=abs(F);
```



3. Redisplay the power spectrum without fftshift and Measure the actual locations of the peaks

```
>>imagesc(S.^0.1);
```



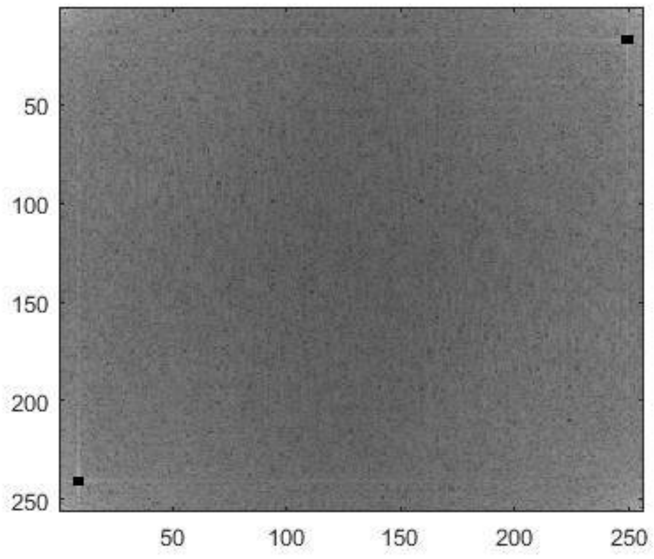
```
>>%[x,y]=ginput  x=9.3553,249.2547 y=240.9834,16.7840
>>x1=9;
>>y1=241;
>>x2=249;
>>y2=17;
```

5. Set to zero the 5x5 neighborhood elements at locations corresponding to the above peaks *in the Fourier transform F*

```
>>x1=9;
>>y1=241;
>>x2=249;
>>y2=17;
>>m=2;

>>F(y2-m:y2+m,x2-m:x2+m) = zeros(2*m + 1);
>>F(y1-m:y1+m, x1-m:x1+m) = zeros(2*m + 1);
```

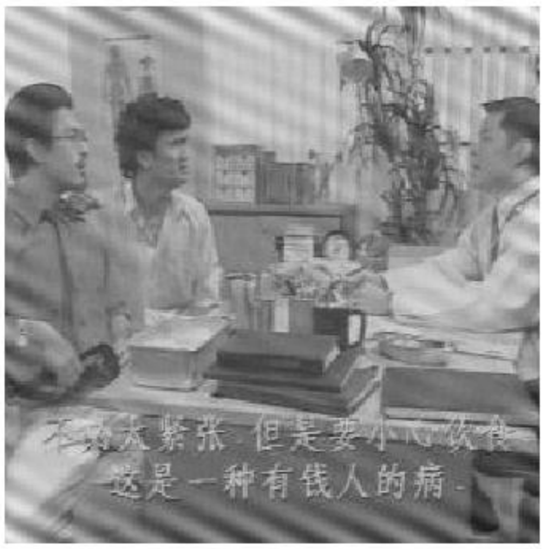
6. Recompute the power spectrum and display it

```
>>S=abs(F);
>>imagesc(S.^0.1);
>>colormap('default');
```



7. Compute the inverse Fourier transform using ifft2 and display the resultant image

```
>>Pn=uint8(ifft2(F));
>>imshow(Pn);
```



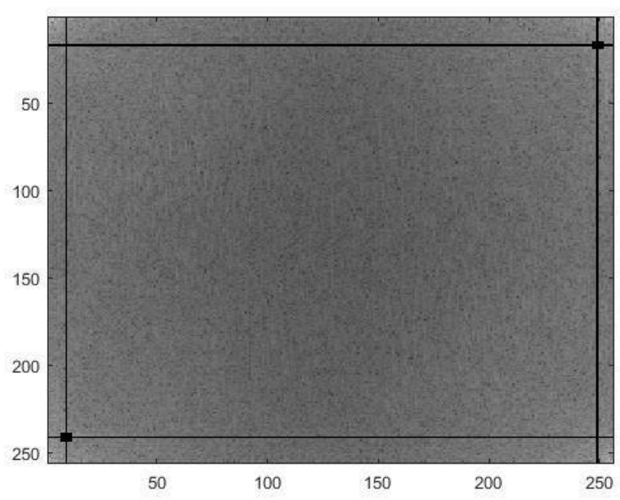
Comment on the result and how this relates to step (c)

The interference patterns are less obvious. We can see two white dots in the power spectrum of the original image, which means there are some frequencies in the image that is extremely high, and that is very likely to be the frequencies that cause interference patterns. So we set zero the 5x5 neighborhood elements at locations corresponding to the white dots and redisplay the image, some interference patterns are thus removed but some are still there.

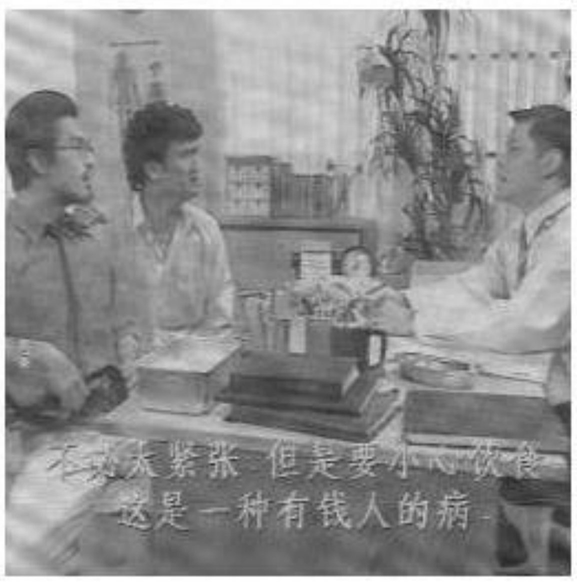
Can you suggest any way to improve this?

After setting zero the 5x5 neighborhood elements, there are actually still some white line on the spectrum. To improve the image, we can set those white part to zero

```
>> [h,w]=size(F);
>>F(:,x1) = zeros(h,1);
>>F(:,x2) = zeros(h,1);
>>F(y1,:) = zeros(1,w);
>>F(y2,:) = zeros(1,w);

>>S=abs(F);
>>imagesc(S.^0.1);
>>colormap('default');
```

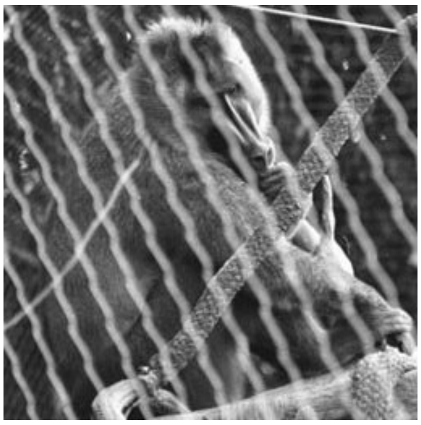
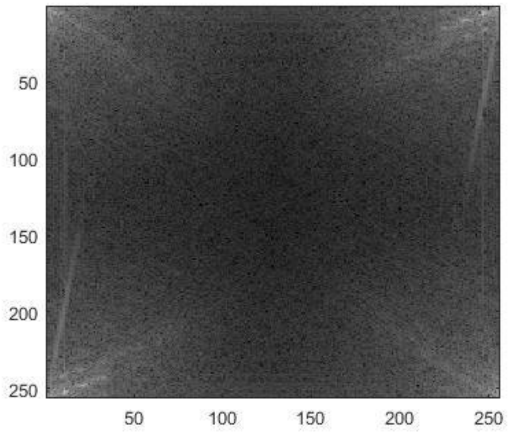


```
>>Pn=uint8(ifft2(F));
>>imshow(Pn);
```



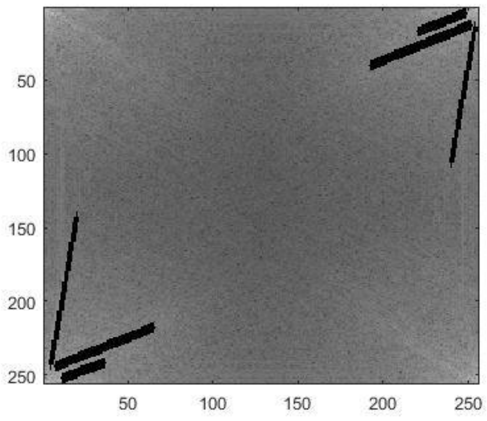
Now the image looks better.

8.Download the image `primate-caged.jpg' from edveNTUre which shows a primate behind a fence. Can you attempt to "free" the primate by filtering out the fence? You are not likely to achieve a clean result but see how well you can do

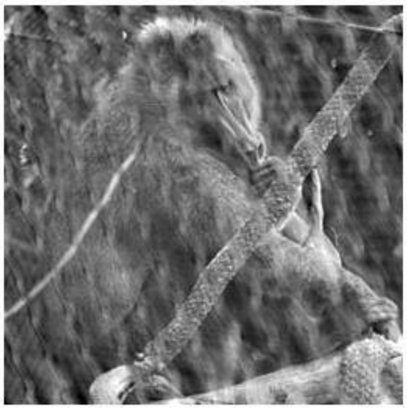



We can see several brighter lines on the spectrum. My method to remove the fence pattern is to set those brighter lines to zero.

After setting them to zero, spectrum looks like this



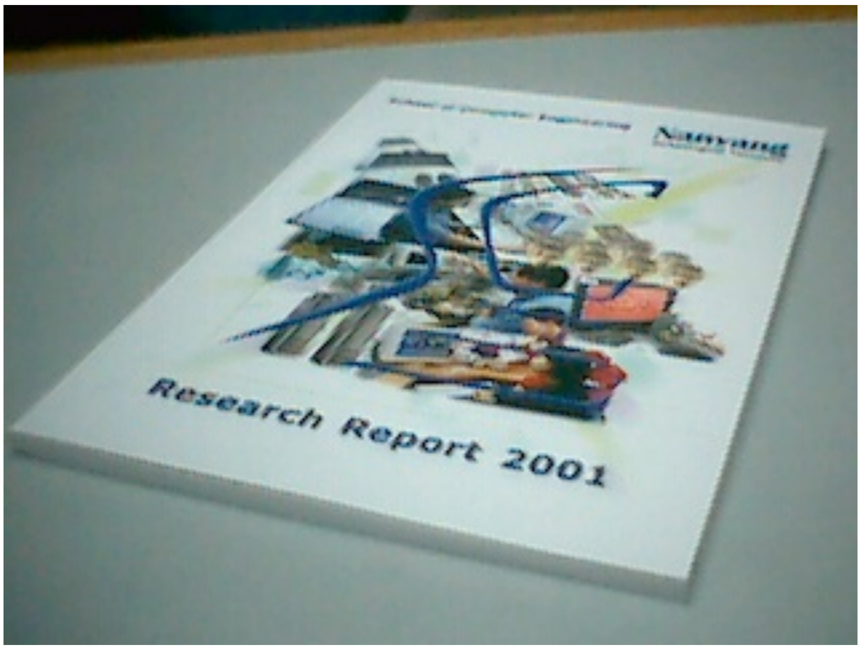
and now the fence is less obvious.



Undoing Perspective Distortion of Planar Surface

1.Read in the image

```
    >>P=imread('book.jpg');
    >>imshow(P)
```



2.Use ginput to get the coordinates of four corners and also set four points where corners should be in the new image

```
%[x,y]=ginput(4)
    >>x=[143;309;4;257];
    >>y=[28;47;158;215];
    >>xn=[0;210;0;210];
    >>yn=[0;0;297;297];
```

3. Set up the matrices

```
    >>A=[
        x(1),y(1),1,0,0,0,-xn(1)*x(1),-xn(1)*y(1);
        0,0,0,x(1),y(1),1,-yn(1)*x(1),-yn(1)*y(1);
        x(2),y(2),1,0,0,0,-xn(2)*x(2),-xn(2)*y(2);
        0,0,0,x(2),y(2),1,-yn(2)*x(2),-yn(2)*y(2);
        x(3),y(3),1,0,0,0,-xn(3)*x(3),-xn(3)*y(3);
        0,0,0,x(3),y(3),1,-yn(3)*x(3),-yn(3)*y(3);
        x(4),y(4),1,0,0,0,-xn(4)*x(4),-xn(4)*y(4);
        0,0,0,x(4),y(4),1,-yn(4)*x(4),-yn(4)*y(4);
    ]
>> v=[xn(1);yn(1);xn(2);yn(2);xn(3);yn(3);xn(4);yn(4)]
```

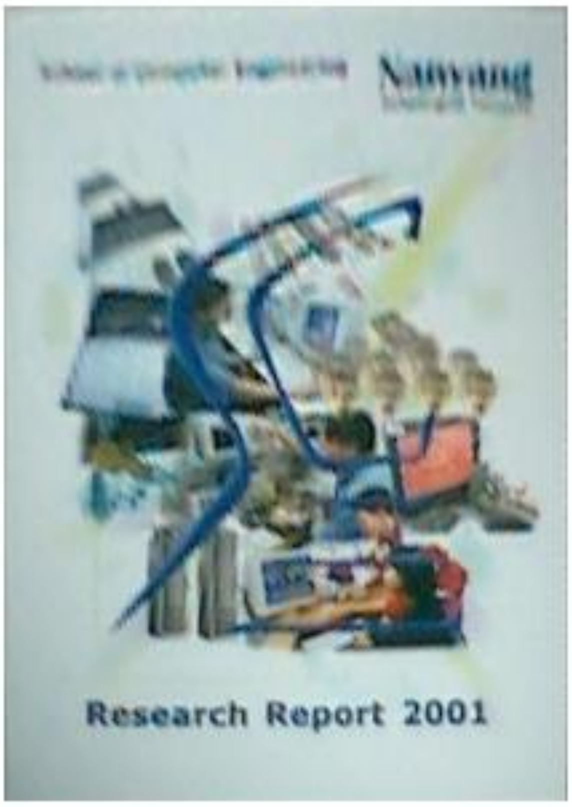
4.Compute the matrix u and convert the projective transformation parameters to the normal matrix

```
>>u = A \ v;
u =
    1.4802
    1.5827
-255.9813
  -0.4324
    3.7782
 -43.9495
    0.0002
    0.0054

>>U = reshape([u;1], 3, 3)';
U =
    1.4802    1.5827 -255.9813
    -0.4324    3.7782  -43.9495
    0.0002    0.0054    1.0000

>>w = U*[x'; y'; ones(1,4)];
w =
     0    275.7816   -0.0000    464.6991
     0    0           551.2695  657.2173
1.1788   1.3132      1.8561     2.2129


>>w = w ./ (ones(3,1) * w(3,:));
```

```
w =

      0  210.0000   -0.0000   210.0000
      0     0       297.0000  297.0000
   1.0000    1.0000    1.0000     1.0000
```

The transformation gives me back the 4 corners of the desired image, which are (0,0) (210,0) (0,297) (210,297).

```
>>T = maketform('projective', U');
>>P2 = imtransform(P, T, 'XData', [0 210], 'YData', [0 297]);
>>imshow(P2)
```



Is this what you expect? Comment on the quality of the transformation and suggest reasons.

Although the quality of the image is not good, this is exactly what I expected. Some part of the image become blurred. When we transform the original image to this new image we have to stretch some part of the image, and stretching may cause a pixel in the old image maps to several pixels in the new image, so several pixels in the new image are actually the same color, and that is the reason why it looks blurred.