PROJECT 2

UIN: 01183457

Part (a) Filtering with LPF and HPF:

Obtaining the spatial domain coefficients of 2-D averaging filter, it’s transformation in the frequency domain and its amplitude spectrum.

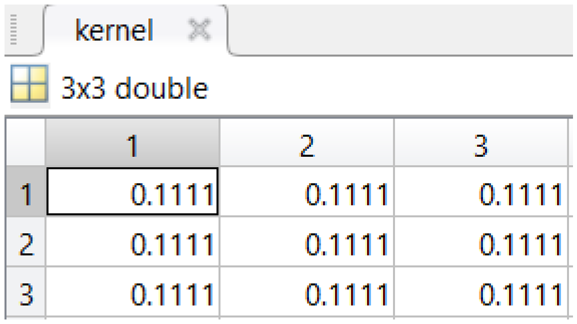
Code:

```
windowSize = 3;
kernel = ones(windowSize, windowSize) / windowSize ^ 2;  % 
Averaging filter kernel
s = load('clown.mat')
rgbImage = ind2rgb(s.X, s.map);
rgbImage=imresize(rgbImage,[128,128]);
I= rgb2gray(rgbImage);
%%% Resizing the original Image to 256X256
figure;
subplot(2,2,1),imshow(I);
title('Original Image')

blurredImage = imfilter(I, kernel, 'symmetric');
subplot(2,2,2);
imshow(blurredImage);
title('Blurred Image (Averaging filter)')

[m,n]=size(kernel);
wm= zeros(m,n);
wn= zeros(m,n);
for u=0:(m-1)
    for x=0:(m-1)
        wm(u+1,x+1)=exp(-2*pi*1i/m*x*u); % fourier 
transform in x axis
    end
end

for v=0:(n-1)
    for y=0:(n-1)
        wn(y+1,v+1)=exp(-2*pi*1i/n*y*v); %fourier transform 
in y axis
    end
```

```
end
f=wm*im2double(kernel)*wn     % coefficients of 3*3 kernel
in frequency domain

f1=log(1+fftshift(f));
amplitude=(real(f1))
figure;
imshow(amplitude,[]);
title('Amplitude Spectrum of  Averaging filter')
```

Output:

Spatial domain coefficients:



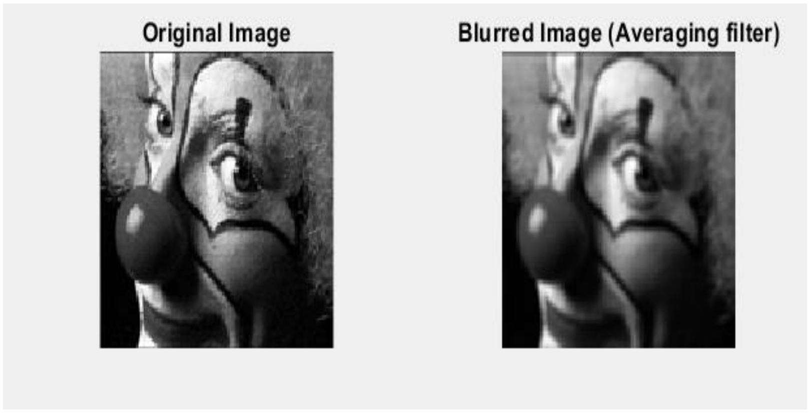
kernel

3x3 double

	1	2	3
1	0.1111	0.1111	0.1111
2	0.1111	0.1111	0.1111
3	0.1111	0.1111	0.1111

Frequency domain coefficients:

```
f =

   1.0000 + 0.0000i  -0.0000 - 0.0000i   0.0000 - 0.0000i
  -0.0000 - 0.0000i   0.0000 + 0.0000i  -0.0000 + 0.0000i
   0.0000 - 0.0000i  -0.0000 + 0.0000i  -0.0000 - 0.0000i
```



Obtain the coefficients of the 2-D Gaussian filter kernel and the filter coefficients in both the spatial and frequency domains.

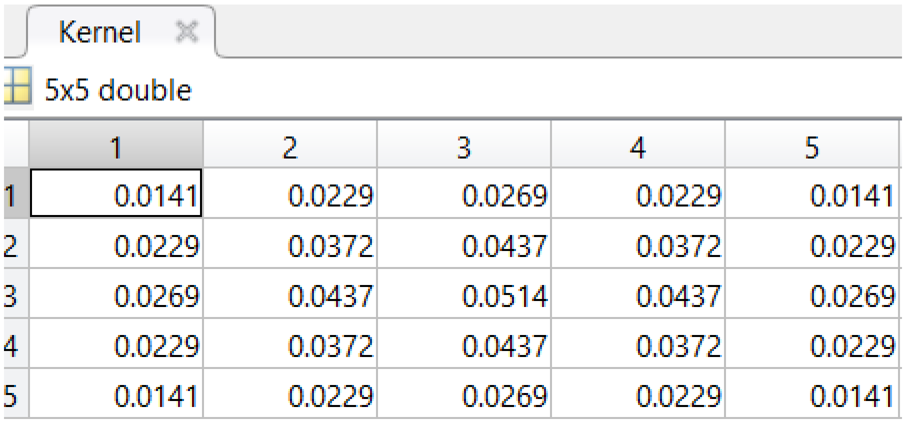
Code:

```
%Design the Gaussian Kernel
%Standard Deviation
sigma = 1.76;
%Window size
sz = 2;
[x,y]=meshgrid(-sz:sz,-sz:sz);  %% Formula:
1/2*pi*(sigma)^2 * exp((x^2+y^2)/2*(sigma)^2)

M = size(x,1)-1;
N = size(y,1)-1;
Exp_comp = -(x.^2+y.^2)/(2*sigma*sigma);
Kernel= exp(Exp_comp)/(2*pi*sigma*sigma); % Gaussian Kernel

%%%%Frequency Domain Transformation of the Gaussian
Kernel%%%%%
[m,n]=size(Kernel);
wm= zeros(m,n);
wn= zeros(m,n);
for u=0:(m-1)
    for x=0:(m-1)
        wm(u+1,x+1)=exp(-2*pi*1i/m*x*u); % fourier
transform in x axis
    end
end

for v=0:(n-1)
    for y=0:(n-1)
```

```
        wn(y+1,v+1)=exp(-2*pi*1i/n*y*v); %fourier transform
in y axis
    end
end
f=wm*im2double(Kernel)*wn     % coefficients of 3*3 kernel
in frequency domain

f1=log(1+fftshift(f));
amplitude=(real(f1))
figure;
imshow(amplitude,[]);
title('Amplitude Spectrum of  Gaussian filter')
```

Output:

Coefficients in spatial domain:



Kernel

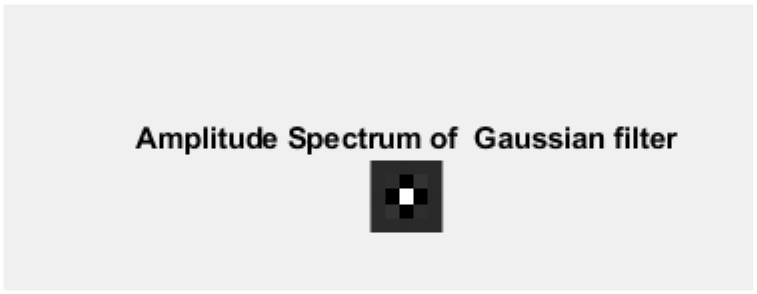
5x5 double

	1	2	3	4	5
1	0.0141	0.0229	0.0269	0.0229	0.0141
2	0.0229	0.0372	0.0437	0.0372	0.0229
3	0.0269	0.0437	0.0514	0.0437	0.0269
4	0.0229	0.0372	0.0437	0.0372	0.0229
5	0.0141	0.0229	0.0269	0.0229	0.0141

Coefficients in frequency domain:

```
f =

   0.7227 + 0.0000i  -0.1056 - 0.0767i  -0.0031 - 0.0097i  -0.0031 + 0.0097i  -0.1056 + 0.0767i
  -0.1056 - 0.0767i   0.0073 + 0.0224i  -0.0006 + 0.0017i   0.0015 - 0.0011i   0.0236 + 0.0000i
  -0.0031 - 0.0097i  -0.0006 + 0.0017i  -0.0001 + 0.0001i   0.0001 - 0.0000i   0.0015 + 0.0011i
  -0.0031 + 0.0097i   0.0015 - 0.0011i   0.0001 + 0.0000i  -0.0001 - 0.0001i  -0.0006 - 0.0017i
  -0.1056 + 0.0767i   0.0236 + 0.0000i   0.0015 + 0.0011i  -0.0006 - 0.0017i   0.0073 - 0.0224i
```

Amplitude Spectrum:



It is observed that Fourier transform of a gaussian kernel is also a gaussian kernel which means it will act as a low pass filter but also high frequency components commensurate with how quickly it's tail decreases from center (mean) point. On the other hand, low pass filter completely removes high frequency components but in come cases edges contain high frequency components and that might be required in some cases to have edge information as well as removal of high frequency noise, therefore, gaussian filter is a appropriate choice in that case.

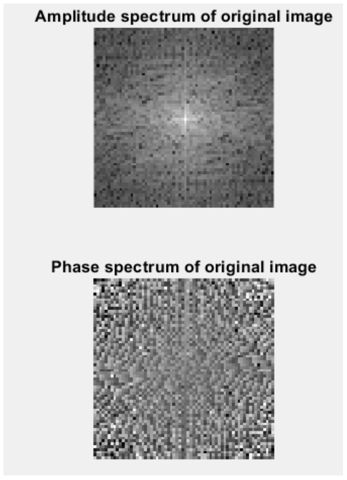
Apply a LoG to an image in the frequency domain. The program should allow for a variable length LoG and output both the filtered images and the frequency domain amplitude (in dB) and phase spectrum of the filtered images. Compare the amplitude and phase of the images before and after filtering.

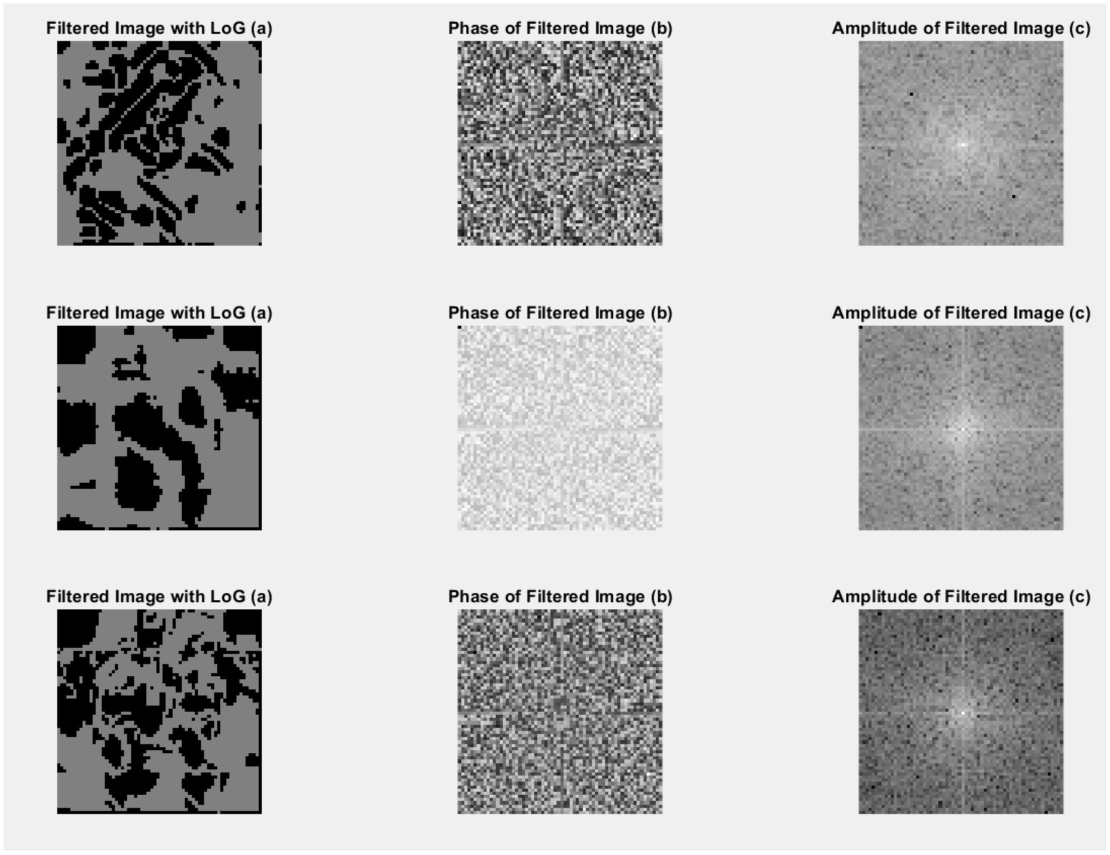
CODE:

```
s = load('clown.mat')
rgbImage = ind2rgb(s.X, s.map);
rgbImage=imresize(rgbImage,[64,64]);
I= rgb2gray(rgbImage);
I=double(I);                                        %%% Resizing
the original Image to 256X256
% figure;
% subplot(2,2,1),imshow(I);
% title('Original Image')

%%%%Frequency domain Transformation of original image%%%%%
[m,n]=size(I);
wm= zeros(m,n);
wn= zeros(m,n);
for u=0:(m-1)
```

```
    for x=0:(m-1)
        wm(u+1,x+1)=exp(-2*pi*1i/m*x*u); % fourier
transform in x axis
    end
end

for v=0:(n-1)
    for y=0:(n-1)
        wn(y+1,v+1)=exp(-2*pi*1i/n*y*v); %fourier transform
in y axis
    end
end
f_image=wm*im2double(I)*wn
f1=log(1+fftshift(f_image));
am=real(f1)
phase1=angle(f1);
figure;
subplot(2,1,1);
imshow(am,[]);
title('Amplitude spectrum of original image')
subplot(2,1,2);
imshow(phase1,[]);
title('Phase spectrum of original image')

sigmas = [2 8 16];
figure;

for i=1:length(sigmas)

    % Step 1
    % Filter the image by Gaussian lowpass filter
    N = 25;
    [X, Y] = meshgrid(-N/2:N/2-1, -N/2:N/2-1);
    G = 1/(2*pi*sigmas(i)^2)*exp(-(X.^2 +
Y.^2)/(2*sigmas(i)^2));
    G = G/sum(G(:));

    bluredImage = (conv2(I, G, 'same'));
    %subplot(3,4,4*(i-1)+1);
    %imshow(uint8(bluredImage))
    %title('Blurred Picture')


    % Filter image with Laplacian filter
    H = [-1 1; 1 -1];
    laplacian = conv2(bluredImage, H, 'same');
    logImage = laplacian;
    logImage(abs(laplacian) < .04*max(laplacian(:))) = 128;

    subplot(3,4,4*(i-1)+1);
    imshow(uint8(logImage))
    title('Filtered Image with LoG (a)')

    %%%%Frequency domain Transformation of filtered
image%%%%%
    [m,n]=size(logImage);
    wm= zeros(m,n);
    wn= zeros(m,n);
    for u=0:(m-1)
      for x=0:(m-1)
        wm(u+1,x+1)=exp(-2*pi*1i/m*x*u); % fourier
transform in x axis
      end
    end

    for v=0:(n-1)
      for y=0:(n-1)
        wn(y+1,v+1)=exp(-2*pi*1i/n*y*v); %fourier transform
in y axis
      end
    end
    log_im=wm*im2double(logImage)*wn
    f1=log(1+fftshift(log_im));
    amplitude=(real(f1))
    phase1=angle(f1);
    subplot(3,4,4*(i-1)+2);
    imshow(phase1,[]);
    title('Phase of Filtered Image (b)')
    subplot(3,4,4*(i-1)+3);
    imshow(amplitude,[]);
    title('Amplitude of Filtered Image (c)')


end
```

OUTPUT:


Amplitude spectrum of original image
Phase spectrum of original image


Filtered Image with LoG (a)
Phase of Filtered Image (b)
Amplitude of Filtered Image (c)
Filtered Image with LoG (a)
Phase of Filtered Image (b)
Amplitude of Filtered Image (c)
Filtered Image with LoG (a)
Phase of Filtered Image (b)
Amplitude of Filtered Image (c)

In the figure, each row of figures corresponds to sigma values 2, 8, 16 respectively. For example, first row corresponds to sigma value 2, (a) filtered image, (b) phase of filtered image and (c) amplitude of filtered image.

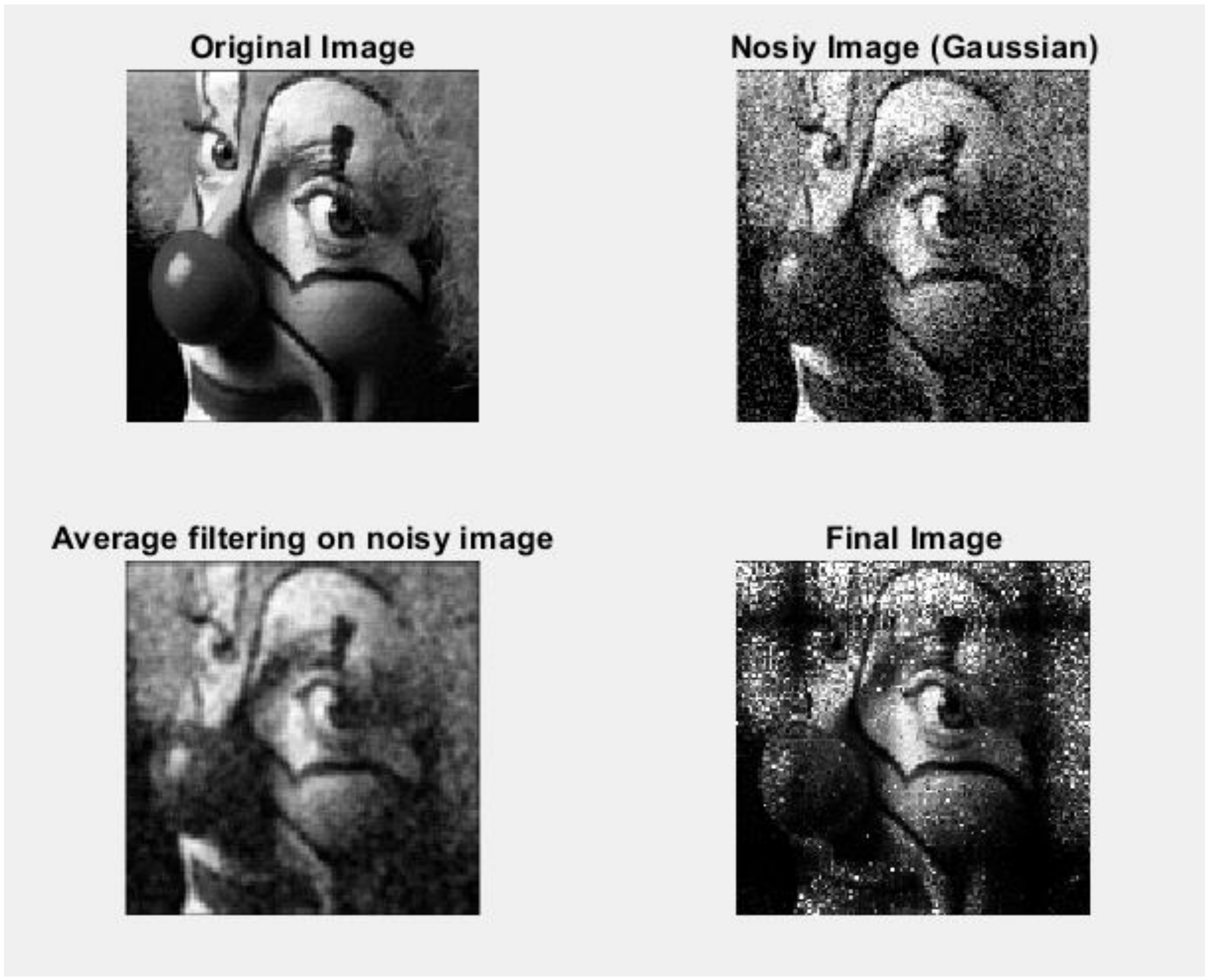
It is observed, with increment in sigma values, the amplitude spectrum of the filtered image become more prominent in the center point whereas the phase spectrum with increment of sigma value seems a little blurred.

Part (b) Filtering with linear phase filtering

Code:

```
s = load('clown.mat')
rgbImage = ind2rgb(s.X, s.map);
rgbImage=imresize(rgbImage,[128,128]);
I= rgb2gray(rgbImage);
%%% Resizing the original Image to 256X256
figure;
subplot(2,2,1),imshow(I);
title('Original Image')

% varI= std2(I)^2;
% SNR= 3;
% sigma_noise = sqrt(varI/10^(SNR/10));
IN2 = imnoise(I,'gaussian',0,0.025); % using imnoise
subplot(2,2,2),imshow(IN2);
title('Nosiy Image (Gaussian)')

%windowSize = 3;
%kernel = ones(windowSize, windowSize) / windowSize ^ 2;  %
Averaging filter kernel

I3=filter2(fspecial('average',3),IN2); % applying average
filter on noisy image
subplot(2,2,3),imshow(I3);
title('Average filtering on noisy image')

%%% DFT of noise filtered image%%%%
In=fft2(I3);
In=log(1+fftshift(In))
%In_real= real(In)
% figure;
% subplot(2,2,1),imshow(abs(In),[]);
% title('Amplitude spectrum of noise filtered image')
% % %%% DFT fourier transform of original image and getting
phase%%%%

I_or=fft2(I);
I_or=log(1+fftshift(I_or))
I_phase=I./abs(I_or);
% subplot(2,2,3),imshow(angle(I_or),[]);
% title('Phase spectrum of original image')
%


I_k=abs(In).*I_phase;
subplot(2,2,4);imshow(abs(I_k))
title('Final Image')
```

Output:



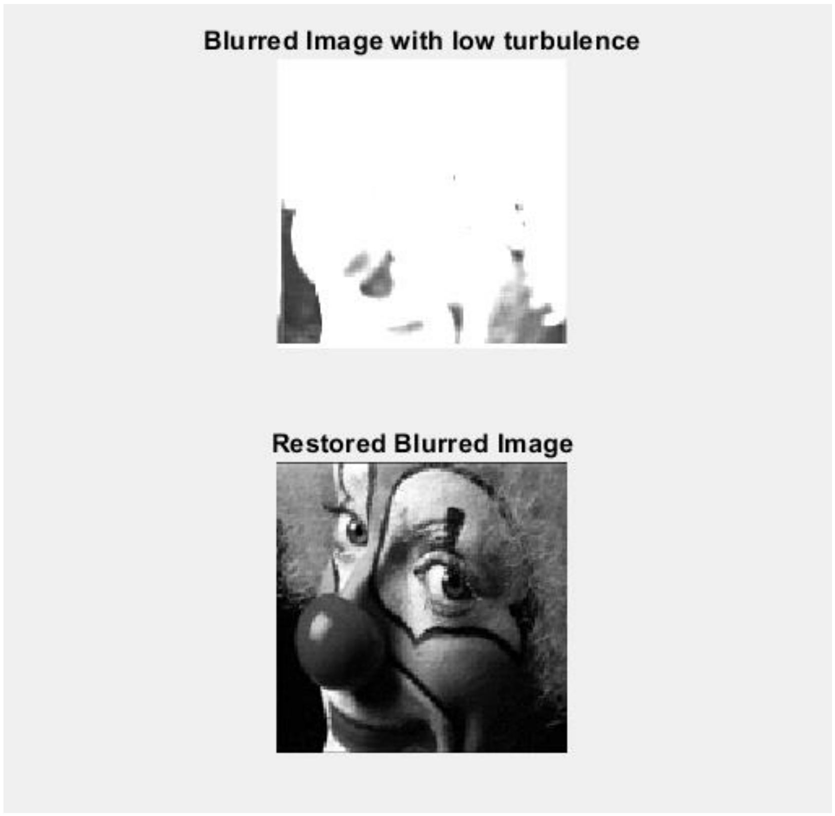
Part (c) Filtering with nonlinear filtering

CODE:

```
k=0.00025; %k=0.0025 (high), k=0.001 (mid), k=0.00025 (low)
sz=2;
[x,y]=meshgrid(-sz:sz,-sz:sz);
M = size(x,1)-1;
N = size(y,1)-1;
Exp_comp = -k*(x.^2+y.^2)*(5/6); % Creating low turbulence
with the formula (5.8)
Kernel= exp(Exp_comp)

s = load('clown.mat')
rgbImage = ind2rgb(s.X, s.map);
rgbImage=imresize(rgbImage,[128,128]);
I= rgb2gray(rgbImage);


Idouble = im2double(I);
blurred = imfilter(Idouble,Kernel,'conv','circular');
figure;
subplot(2,1,1);imshow(blurred)
title('Blurred Image with low turbulence')


wnr1 = deconvwnr(blurred,Kernel);
subplot(2,1,2); imshow(wnr1)
title('Restored Blurred Image')


[peaksnr, snr] = psnr(I, blurred);

fprintf('\n The Peak-SNR value for Blurred image is %0.4f\n
', peaksnr); %
https://www.mathworks.com/help/images/ref/psnr.html

fprintf('\n The SNR value for blurred image is %0.4f \n',
snr);

[peaksnr_r, snr_r] = psnr(I, wnr1);

fprintf('\n The Peak-SNR value for Restored image is
%0.4f\n ', peaksnr_r); %
https://www.mathworks.com/help/images/ref/psnr.html
```

```
fprintf('\n The SNR value for Restored image is %0.4f \n',
snr_r);
```

OUTPUT:



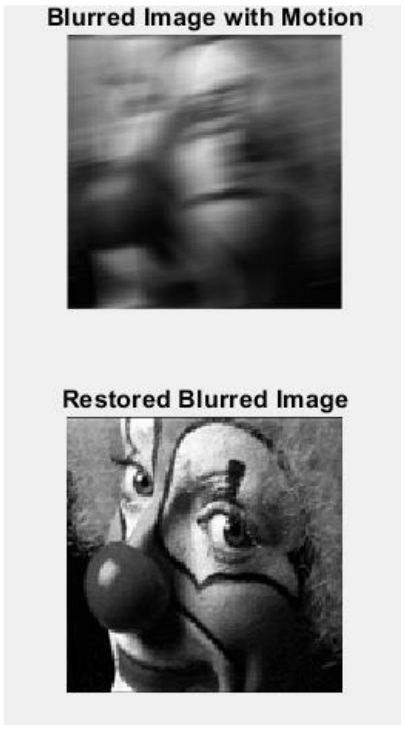
```
The Peak-SNR value for Blurred image is -19.6950

The SNR value for blurred image is 0.3610

The Peak-SNR value for Restored image is 266.2294

The SNR value for Restored image is 258.6354
```

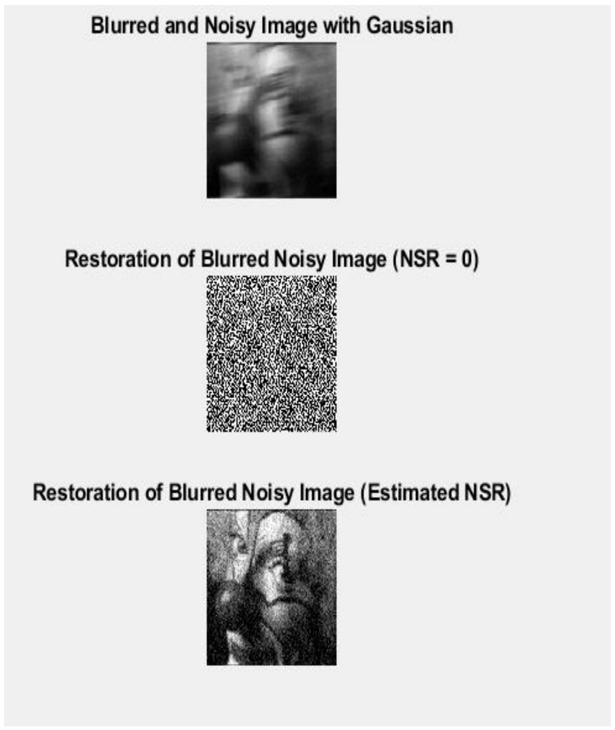
With Gaussian and motion noise:

CODE:

```
s = load('clown.mat')
rgbImage = ind2rgb(s.X, s.map);
rgbImage=imresize(rgbImage,[128,128]);
I= rgb2gray(rgbImage);

PSF = fspecial('motion',21,11);
Idouble = im2double(I);
blurred = imfilter(Idouble,PSF,'conv','circular');
figure;
subplot(2,1,1);imshow(blurred)
title('Blurred Image with Motion')

wnr1 = deconvwnr(blurred,PSF);
subplot(2,1,2); imshow(wnr1)
title('Restored Blurred Image')

noise_mean = 0;
noise_var = 0.0001;
blurred_noisy =
imnoise(blurred,'gaussian',noise_mean,noise_var);
figure;
subplot(3,1,1);imshow(blurred_noisy)
title('Blurred and Noisy Image with Gaussian')

wnr2 = deconvwnr(blurred_noisy,PSF);
subplot(3,1,2);imshow(wnr2)
title('Restoration of Blurred Noisy Image (NSR = 0)')

signal_var = var(Idouble(:));
NSR = noise_var / signal_var;
wnr3 = deconvwnr(blurred_noisy,PSF,NSR);
subplot(3,1,3);imshow(wnr3)
title('Restoration of Blurred Noisy Image (Estimated NSR)')
```

OUTPUT:




Blurred and Noisy Image with Gaussian
Restoration of Blurred Noisy Image (NSR = 0)
Restoration of Blurred Noisy Image (Estimated NSR)

Linear filtering is the filtering method in which the value of output pixel is linear combinations of the neighboring input pixels. it can be done with convolution. For examples, mean/average filters or Gaussian filtering. A non-linear filtering is one that cannot be done with convolution or Fourier multiplication. A sliding median filter is a simple example of a non-linear filter. In cases where the input data contains a large amount of noise, but the magnitude is low, a linear low-pass filter may suffice. Conversely, if an image contains a low amount of noise but with relatively high magnitude, then a median filter may be more appropriate.