EECS 740 DIGITAL IMAGE PROCESSING

Assignment #3

Dr. Guanghui (Richard) Wang

Lohith Reddy Nanuvala

2790468

11/29/2015

Abstract:

In this assignment we are going to discusses basic principles for filtering in the frequency domain and image restoration. The usual convolution operation in spatial domain transforms to array multiplication in frequency domain, so any transformation can be performed in frequency domain by just multiplying with suitable filter matrix. Techniques like Gaussian low pass filter, Butterworth band filter, Median filter, Adaptive median filter, Full inverse filtering and Wiener filtering are discussed and implemented using two ideal images. We will add noise to the images, apply the filtering techniques and observe the results. We will check this results with theoretical results and draw conclusions.

Technical discussion:

Filtering in Frequency Domain:

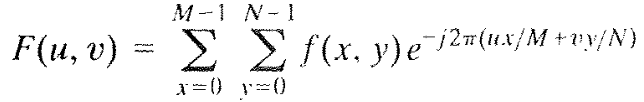
Filtering in frequency domain is much easier because of convolution.

*f(x,y)\*h(x,y)* ⬄ *F(u,v)H(u,v)*

*f(x,y)h(x,y)* ⬄ *F(u,v)\*H(u,v)*

If the two functions *f(x),* *h(x)* are of different lengths, then when we do the convolution there could be wraparound error. To avoid this, we append them with zeros.

The Discrete Fourier Transform (DFT) of an image *f(x,y)* is given by the expression



Where M, N are the dimensions of the image.

Edges and other sharp transitions (such as noise) in the gray levels of an image contribute significantly to the high-frequency content of its Fourier transform. Hence blurring (smoothing) is achieved in the frequency domain by attenuating a specified range of high frequency components in the transform of a given image.

The general procedure for performing filtering in frequency domain can be listed as follows,

1. Given an input image *f(x,y)* of size M*x*N, obtain the padding parameter P and Q. (typically we select P=2\*M and Q=2\*N).
2. Form a padded image, *fp(x,y)* of size P*x*Q by appending the necessary # of zeros to *f(x,y).*
3. Multiply *fp(x,y)* by *(-1)x+y* to center its transform.
4. Compute the DFT *F(u,v)* of the image obtained above.
5. Generate a real symmetric filter function *H(u,v)* of size P*x*Q with center at coordinates (P/2,Q/2). Form the product *G(u,v)* = *H(u,v)F(u,v)* using array multiplication; that is *Gp(i,k)*=*H(i,k)F(i,k).*
6. Obtain the processed image



Where the real part is selected in order to ignore parasitic complex components resulting from computational inaccuracies, *p* indicates we are working with the padded image.

1. Obtain the processed result, *g(x,y)* by extracting the MxN region from the top, left quadrant of *gp*.

Gaussian Low pass Filter:

* Gaussian low pass filter is used to smooth the image.
* Gaussian low pass filter in 2 dimensions is given by



We can replace sigma by D0

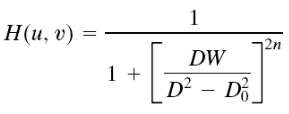


* D0 is a positive constant and D(u,v) is the distance between a point (u,v) in the frequency domain and the center of the frequency rectangle.



Butterworth Band Filter:

* Band filters work on specific band instead of the entire spectrum.
* The Butterworth filter is the best compromise between attenuation and phase response. It has no ripple in the pass band or the stop band.
* The transformation function for Butterworth band pass filter is



Where W is the width of the band, *D* is the distance *D(u,v)* from the center of the filter.

Median Filter:

A median filter works by replacing value at each pixel by the median value of pixels present in a small window around the chosen pixel.

F(x,y) = median(s,t)ϵSxy{g(s,t)}

Adaptive Median Filter:

The Adaptive Median Filter works in 2 stages,

Stage A:

A1 = Zmed – Zmin; A2 = Zmed – Zmax

If A1>0 and A2<0, go to stage B

Else increase the window size

If window size ≤ Sxy, repeat Stage A; Else output Zmed

Stage B:

B1 = Zxy – Zmin; B2 = zxy – Zmax

If B1>0 and B2<0, output zxy;

Else output Zmed

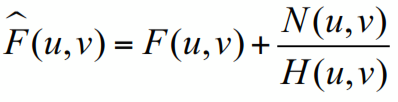
Atmospheric turbulence is a model of degradation by environmental conditions.



*k* is a constant dependant on nature of turbulence.

Full Inverse Filter:

An Inverse filter tries to estimate the transform of the original image. Once the model is established we do operations in reverse order of the assumed model. Thus we restore the image. In the figure shown below, Fhat is the degraded image, F is the original image, N is the noise and H is the transformation function.

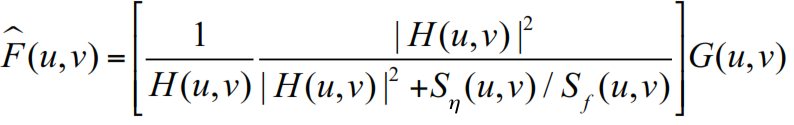


**Limitations:**

* We cannot recover the original image exactly because we don’t know the noise N(u,v).
* If the degradation function has zero or very small values, then the ratio N(*u*, *v*) / *H*(*u*, *v*) could easily dominate the estimate.

Wiener Filter:

* Wiener filter is based on the minimum mean square error filtering.
* The general expression for estimate is given by

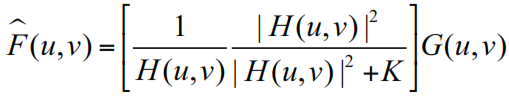


Where Fhat is the estimate, G is degraded image, H is degradation function,

H\* is complex conjugate of H, *Sn*(u,v) = |N(u,v)|^2 is power spectrum of noise,

*Sf*(u,v)=|F(u,v)|^2 is power spectrum of undegraded image.

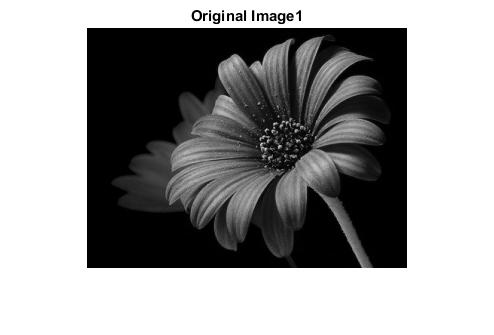
* The expression for wiener filter is given by

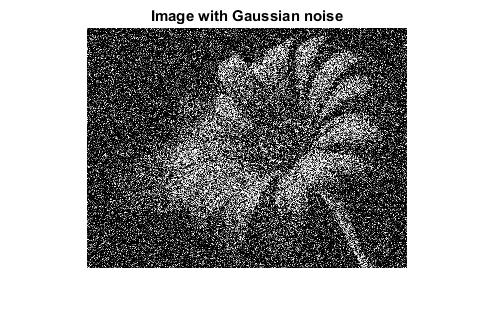


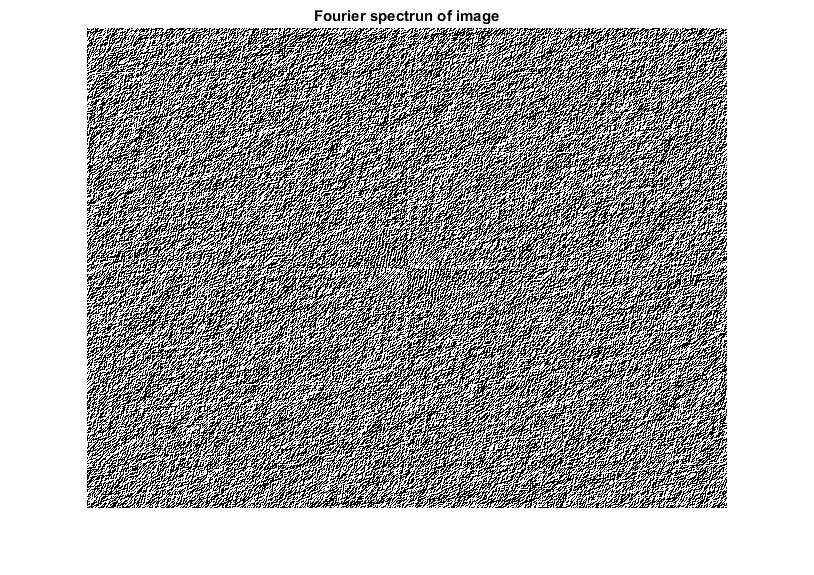
K is a specified constant that is added to all terms of H(u,v)^2. K is chosen interactively to get better visual results.

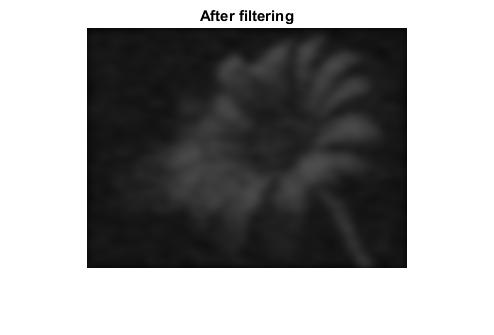
**Results:**

**Problem 1**

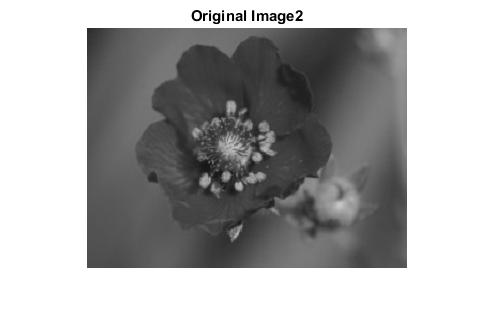
****

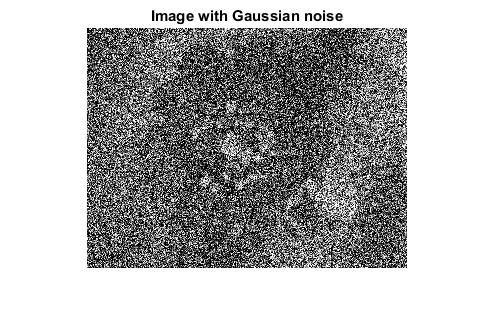
****

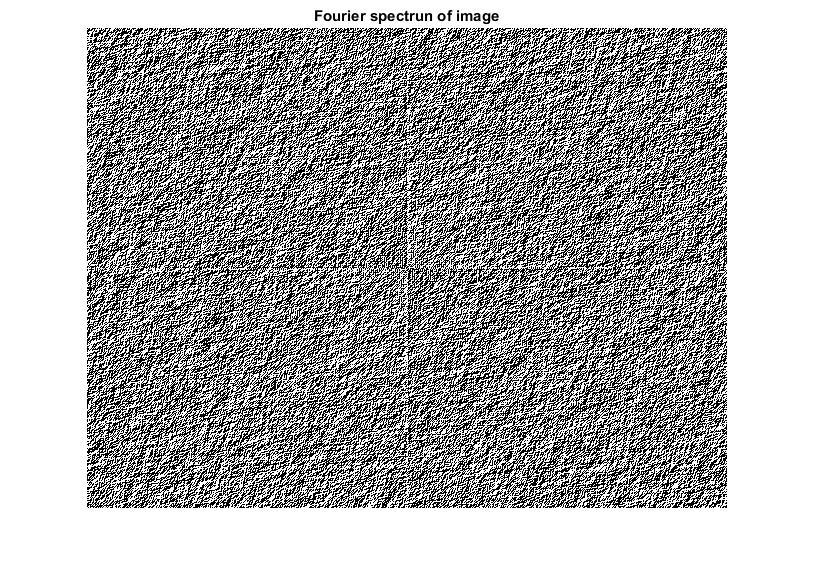
****

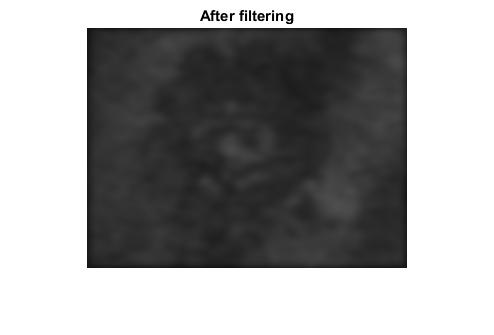
****

****

****

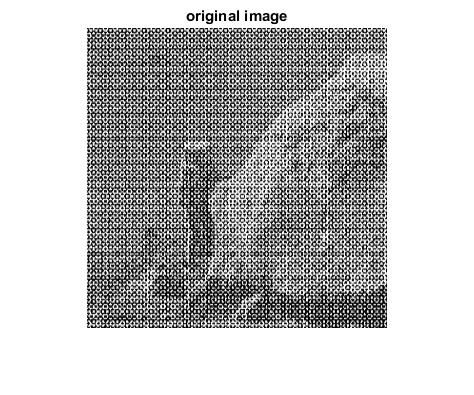
****

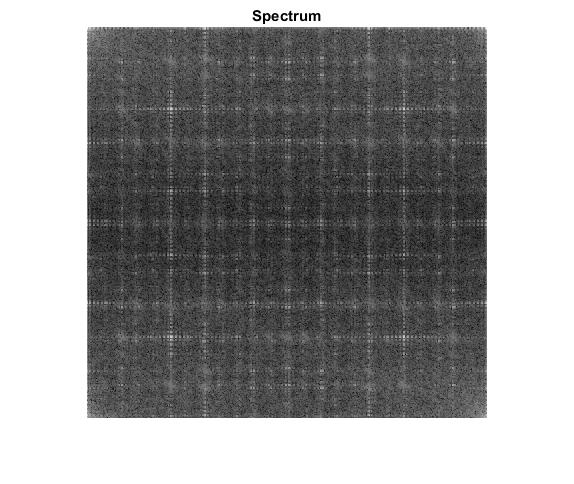
****

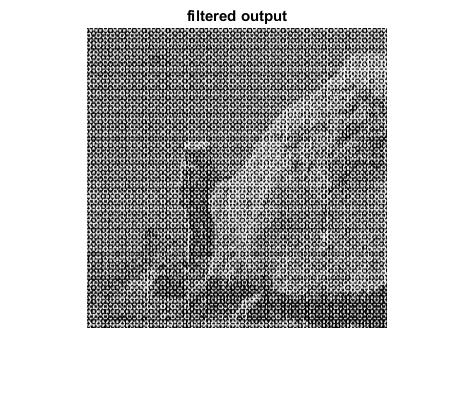
****

****

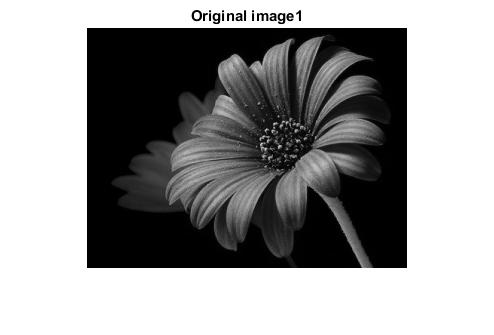
**Problem 2:**

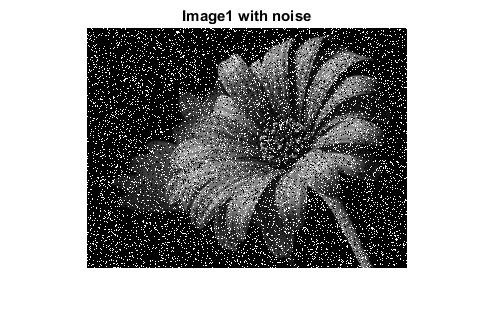
****

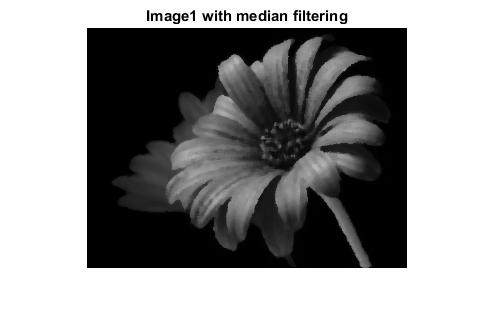
****

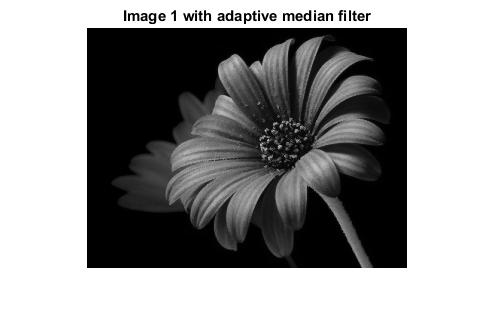
****

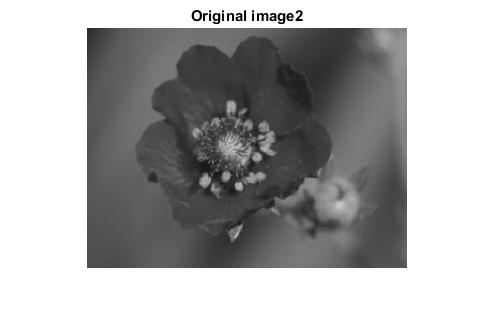
**Problem 3:**

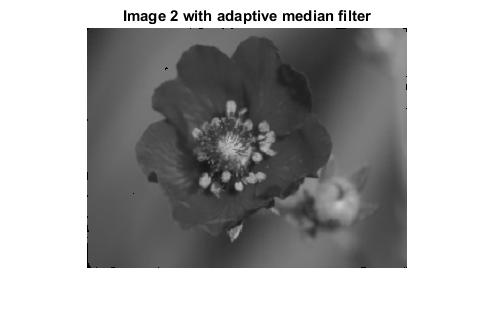
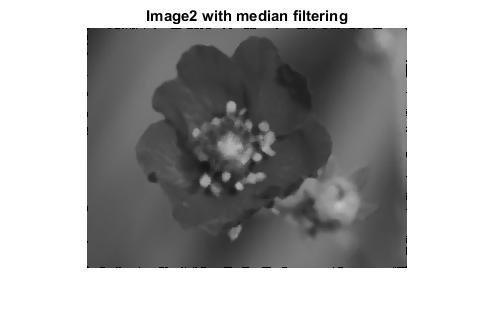
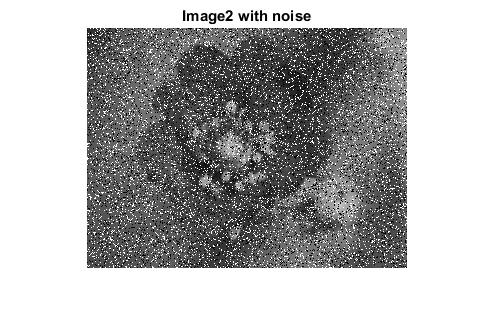
****

****

****

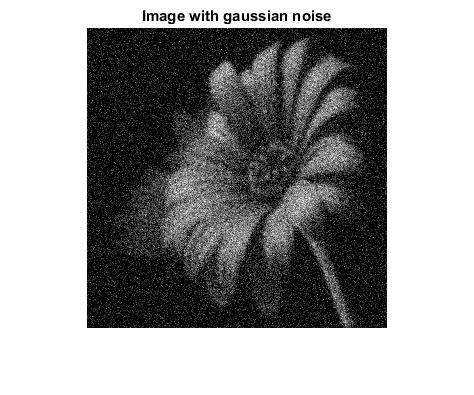
****

****

****

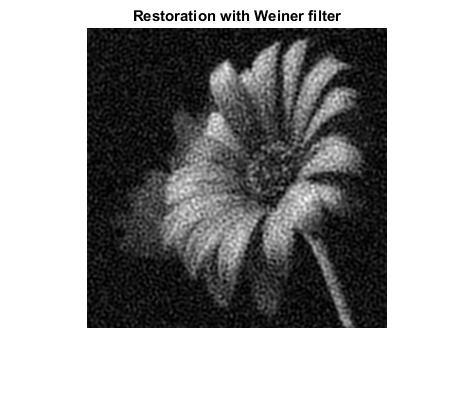
**Problem 4:**

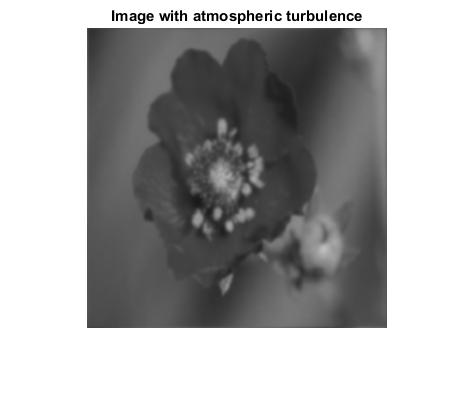
****

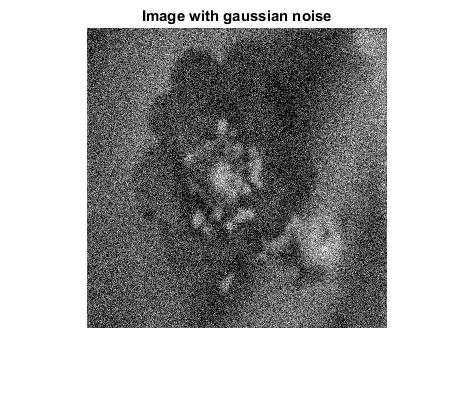
****

****

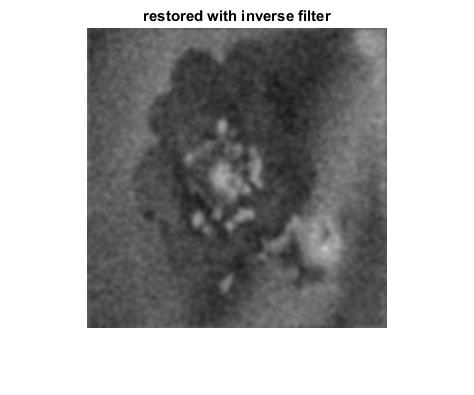
****

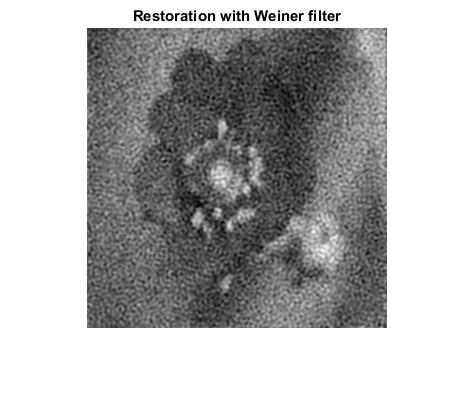
****

****

****

****

****

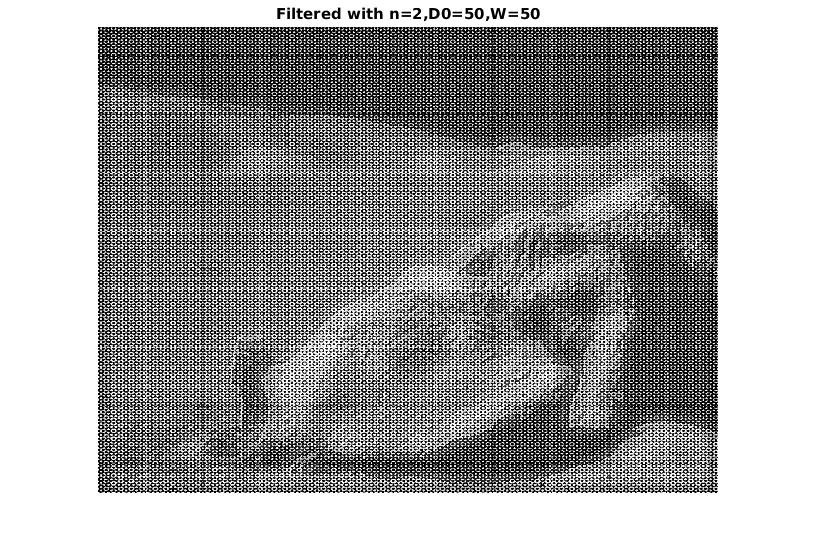
****

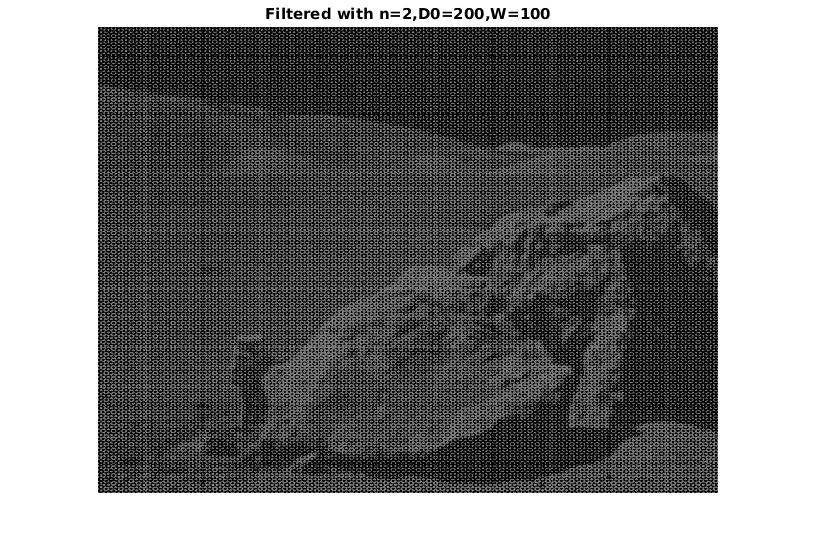
**Analysis:**

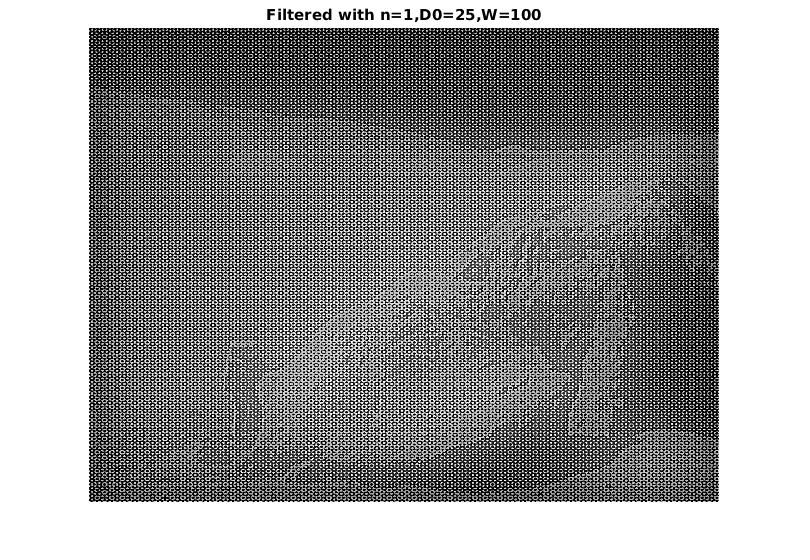
Problem 1: The Gaussian filter was successfully applied in frequency domain. It is clearly visible that the noise which we have added is very high and thus the filtering technique could not remove all the noise at once. However, we could restore the original image with some post processing after the filter was applied.

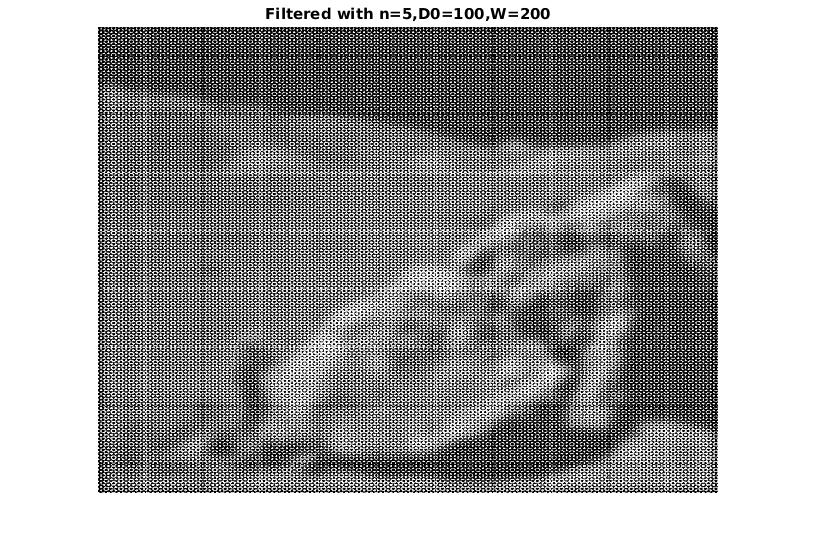
Problem 2:

I was able to produce the spectrum as expected by using the log of the transformation. However designing the butterworth filter posed a great challenge and I was not able to produce an image which was completely removed of noise. Some of the results when I tried using different combinations of n( order of the filter), D0, W(width of the band) are,









Problem 3:

The salt and pepper noise was added successfully and the resulting noisy images were restored successfully using the median and adaptive median filters. Points to be noted here are,

* The image produced by median filter is a little blurred when compared to the one with adaptive median filter. Adaptive median filter produced sharper output.
* There is a clear distinction between the foreground and background images in first image, where as in the second image they were similar in some parts. Due to this reason, while filtering the window size had to be increased for the second image. The local window size for adaptive median filtering for first image is 7x7 whereas for 2nd it had to be 15x15 to produce better results.

Problem 4:

The process of adding atmospheric turbulence, Gaussian noise and then applying inverse and wiener filtering were carried out successfully. We could see a slightly blurred image when atmospheric turbulence is applied. Adding Gaussian noise to the blurred image makes it even worse which was clearly visible. I then applied Gaussian filter to remove Gaussian noise, the result is a blurred image as expected. Then I applied the inverse filter by dividing the noisy image with the turbulence function modelled at the beginning. The result was better. The result of applying wiener filter is very better than inverse filtering. Similar results were observed in the 2nd image as well.

**Appendix:**

**Problem 1:**

Problem.m

% Program for Problem 1

clear;

close all;

I = imread('flower1.jpg');

I = im2double(I);

I = rgb2gray(I);

figure, imshow(I); title('Original Image1');

Process(I);

J = imread('flower2.jpg');

J = im2double(J);

J = rgb2gray(J);

figure, imshow(J); title('Original Image2');

Process(J);

% end

Process.m

function [ Iout ] = Process( In )

% This does all the processing for the input image

Inoise = imnoise(In,'gaussian',0,0.2);

figure, imshow(Inoise); title('Image with Gaussian noise');

% Filtering in frequency domain begins here Lecture #22

[M, N] = size(Inoise);

% padding parameters

P = 2\*M;

Q = 2\*N;

% padded image

Fp = zeros(P,Q);

Fp(1:M,1:N) = Inoise;

% shifting image to center

for u = 1:P

for v = 1:Q

b = power(-1, (u+v));

Fp(u,v) = b\*Fp(u,v);

end

end

Itr = real(fft2(Fp));

figure, imshow(real(Itr)); title('Fourier spectrun of image');

% filtering using gaussian lowpass filter

I2 = gauss\_lowpass(Itr);

I3 = real(ifft2(I2));

for u = 1:P

for v = 1:Q

I3(u,v) = power(-1,(u+v))\*I3(u,v);

end

end

% applying the inverse fourier transform

Iout = I3(1:M,1:N);

figure, imshow(Iout); title('After filtering');

Ifinal = Inoise - Iout;

Imask = Ifinal - In;

Ifinal2 = Inoise - Imask;

figure, imshow(Ifinal2); title('Restored image with post processing');

end

gauss\_lowpass.m

function [out] = gauss\_lowpass(In)

% function to implement the gaussian low pass filter.

[P , Q] = size(In);

% D0 = sum(sum(In))/(P\*Q);

D0 = 20;

% out = In;

H = zeros(P,Q);

a = P/2;

b = Q/2;

K = power(D0,2);

K = 2\*K;

D = 0;

for i = 1:P

for j = 1:Q

D = power((i-a),2) + power((j-b),2);

D = sqrt(D);

k = power(D,2);

H(i,j) = exp((-1\*k)/ K);

k = 0;

end

end

out = H.\*In;

end

**Problem 2:**

Problem2\_Final.m

clear;

close all;

I = imread('i7w0S.1.png');

I = rgb2gray(I);

figure, imshow(I); title('original image');

I = im2double(I);

[M, N] = size(I);

% padding parameters

P = 2\*M;

Q = 2\*N;

Fpadded = zeros(P,Q);

Fpadded(1:M, 1:N) = I;

% centering the transform.

for u = 1:P

for v = 1:Q

Fpadded(u,v) = power(-1,(u+v))\*Fpadded(u,v);

end

end

% fourier transform

F = fft2(Fpadded);

F = fftshift(F);

F1 = log(1+abs(F));

Fm = max(max(F1));

figure, imshow(F1 ./ Fm); title('Spectrum');

n=10;

D0 =0.00001;

W = 1;

Hp = zeros(P,Q);

a = P/2;

b = Q/2;

for u = 1:P

for v = 1:Q

D = sqrt((u-a)^2 + (v-b)^2);

H = 1 / ((1+((D\*W)/(D^2-D0^2)))^(2\*n));

Hp(u,v) = H ;

end

end

G = Hp .\* F;

G = ifftshift(G);

J = real(ifft2(G));

% figure, imshow(J); title('before multiplying');

for i = 1:P

for j = 1:Q

J(i,j) = power(-1,(i+j))\*J(i,j);

end

end

figure, imshow(J(1:M,1:N)); title('filtered output');

**Problem 3:**

% this program is for impulse noise(salt & pepper) addition and removal

% using median and adaptive median filters

clear;

close all;

I = imread('flower1.jpg');

I = rgb2gray(I);

J = imread('flower2.jpg');

J = rgb2gray(J);

figure, imshow(I); title('Original image1');

figure, imshow(J); title('Original image2');

% adding noise

In = imnoise(I,'salt & pepper',0.2);

figure, imshow(In); title('Image1 with noise');

Jn = imnoise(J,'salt & pepper',0.2);

figure, imshow(Jn); title('Image2 with noise');

% median filter

Iout = medfilt2(In, [5 5]);

figure, imshow(Iout); title('Image1 with median filtering');

Jout = medfilt2(Jn, [5 5]);

figure, imshow(Jout); title('Image2 with median filtering');

% adaptive median filtering

Iad = ad\_medianfilter(I,7);

figure, imshow(Iad); title('Image 1 with adaptive median filter');

Jad = ad\_medianfilter(J,15);

figure, imshow(Jad); title('Image 2 with adaptive median filter');

% end

**Problem 4:**

Problem4.m

% This program reads the input files and calls the process function

clear;

close all;

I = imread('flower1.jpg');

Process4(I);

J = imread('flower2.jpg');

Process4(J);

% end

Process.m

function [ ] = Process4( In )

% Process4 does all the processing

In = rgb2gray(In);

In = im2double(In);

[M,N]=size(In);

figure, imshow(In); title('Original image');

Ft = fftshift(fft2(In));

% Atmospheric turbulence with k = 0.002

k = 0.002;

u0 = M/2;

v0 = N/2;

u = (1:M)-u0;

v = (1:N)-v0;

[U,V] = meshgrid(u,v);

D = (U.^2+V.^2);

H = exp(-k\*(D.^(5/6)));

Ftr = Ft .\* H;

Fblurred = real(ifft2(ifftshift(Ftr)));

figure, imshow(Fblurred); title('Image with atmospheric turbulence');

% lets add gaussian noise with mean = 0 and variance 0.02

Gn = imnoise(Fblurred,'gaussian',0,0.02);

figure, imshow(Gn); title('Image with gaussian noise');

% designing filter with width 70

Fblurrnoisy = fftshift(fft2(Gn));

Fblurr = gauss\_lowpass(Fblurrnoisy);

figure, imshow(real(ifft2(ifftshift(Fblurr)))); title('Gaussian noise removed');

radius = 70;

a = floor(M/2)+1;

b = floor(N/2)+1;

for i = 1:M

for j = 1:N

dist = ((i-a)^2 + (j-b)^2)^0.5;

if dist < radius

Fblurr(i,j) = Fblurr(i,j)/H(i,j);

else

Fblurr(i,j) = 0;

end

end

end

% restoring

Frreal = abs(ifft2(ifftshift(Fblurr)));

figure, imshow(Frreal); title('restored with inverse filter');

% Wiener filter

% W(i,j) = (1/H(i,j))\*((abs(H(i,j))^2)/(abs(H(i,j))^2+1000));

L = conj(H)./(H .\* (conj(H) + 1000));

% H1 = abs(H).^2;

% H2 = H1 + 1000;

% H3 = H1./H2;

% L = conj(H) \* H3;

FWiener = L .\* Fblurr;

figure, imshow(abs(ifft2(ifftshift(FWiener)))); title('Restored with Wiener filter');

end

gauss\_lowpass.m

function [out] = gauss\_lowpass(In)

% function to implement the gaussian low pass filter.

[P , Q] = size(In);

% D0 = sum(sum(In))/(P\*Q);

D0 = 20;

% out = In;

H = zeros(P,Q);

a = P/2;

b = Q/2;

K = power(D0,2);

K = 2\*K;

D = 0;

for i = 1:P

for j = 1:Q

D = power((i-a),2) + power((j-b),2);

D = sqrt(D);

k = power(D,2);

H(i,j) = exp((-1\*k)/ K);

k = 0;

end

end

out = H.\*In;

end