Image Processing | Lab 7 & 8

Different types of spatial filters & 1-D Discrete Fourier Transform

Raj Panchal
17CEUBG104

Important Function for this lab :

Homomorphic filter:

```
function s = my_homomorphic_filter(r,d0,N)
  log_img = log(0.001+im2double(r));
  high_pass_img  = my_butter_worth_filter(log_img,d0,N);
  s = exp(high_pass_img);
endfunction
```

Butterworth filter:

```
function s = my_butter_worth_filter(r,d0,N)
  [m,n]=size(r);
   p=2*m;
   q=2*n;
   pad=zeros(p,q);
   pad(1:m,1:n)=im2double(r);

  for i=1:p
    for j=1:q
      pad(i,j)=pad(i,j)*(-1)^(i-1+j-1);
    endfor
  endfor

  fta = fft2(pad);
  filt = zeros(p,q);

  for i=1:p
    for j=1:q
      d=sqrt((i-p/2)^2+(j-q/2)^2);
      denominator = 1+((d0/d)^(2*N));
      filt(i,j) = 1/denominator;
    endfor
  endfor

  g=fta.*filt;

  sg = real(ifft2(g));
  for i=1:p
    for j=1:q
      t(i,j)=sg(i,j)*(-1)^(i-1+j-1);
    endfor
  endfor

  s=t(1:m,1:n);
endfunction
```

Arithmetic mean Filter:

```
function s=my_arithmetic_mean_filter(r,m,n)
[M,N]=size(r);
a=(m-1)/2;
b=(n-1)/2;
new_imsize = zeros(M+2*a,N+2*b);
new_imsize(1+a:M+a,1+b:N+b)= r;
s = zeros(size(r));
for i=1+a:M+a,
  for j=1+b:N+b,
    k=new_imsize(i-a:i+a,j-a:j+a);
    s(i-a,j-b)=sum(sum(k))/(m*n);
  endfor
endfor
s=uint8(s);
endfunction
```

Geometric mean Filter:

```
function s=my_geometric_mean_filter(r,m,n)
[M,N]=size(r);
a=(m-1)/2;
b=(n-1)/2;
new_imsize = zeros(M+2*a,N+2*b);
new_imsize(1+a:M+a,1+b:N+b)= r;
s = zeros(size(r));
for i=1+a:M+a,
  for j=1+b:N+b,
    k=new_imsize(i-a:i+a,j-a:j+a);
    s(i-a,j-b)=prod(prod(k))^(1/(m*n));
  endfor
endfor
s=uint8(s);
endfunction
```

Contra harmonic mean Filter:

```
function s = my_contra_harmonic_mean_filter(r,m,n,Q)
[M,N]=size(r);
a=(m-1)/2;
b=(n-1)/2;
new_imsize = zeros(M+2*a,N+2*b);
new_imsize(1+a:M+a,1+b:N+b)= r;
s = zeros(size(r));
for i=1+a:M+a,
  for j=1+b:N+b,
            k = new_imsize(i-a:i+a,j-b:j+b);
            numerator = (sum(sum(k))).^(Q+1);
            denominator = (sum(sum(k.^(Q))));
  if(k==0)
    s(i-a,j-b)=0;
  elseif(Q<0)
              s(i-a,j-b)=(m*n)*(numerator/denominator);
  elseif(Q>0)
    s(i-a,j-b)=(numerator/denominator)/(m*n);
  elseif(Q==0)
    s(i-a,j-b)=(numerator/denominator);
  endif
  endfor
endfor
s = uint8(s);
endfunction
```

Median Filter:

```
function s=my_median_filter(r,m,n)
[M,N]=size(r);
a=(m-1)/2;
b=(n-1)/2;
new_imsize = zeros(M+2*a,N+2*b);
new_imsize(1+a:M+a,1+b:N+b)= r;
s = zeros(size(r));
for i=1+a:M+a,
  for j=1+b:N+b,
    k=new_imsize(i-a:i+a,j-a:j+a);
    s(i-a,j-b)=median(median(k));
  endfor
endfor
s=uint8(s);
endfunction
```

Adaptive mean filter:

```
function s=my_adaptive_median_filter(r,m,n)
[M,N]=size(r);
a=(m-1)/2;
b=(n-1)/2;
new_imsize = zeros(M+2*a,N+2*b);
new_imsize(1+a:M+a,1+b:N+b)= r;
s = zeros(size(r));
for i=1+a:M+a,
  for j=1+b:N+b,
    ws=3;
    smax=9;
    k=new_imsize(i-a:i+a,j-b:j+b);
    z_med = median(median(k));
    z_min = min(min(k));
    z_max = max(max(k));
    A1 = z_med-z_min;
    A2 = z_med-z_max;
    flag=0;
    while(not(A1>0 && A2<0))
      ws=ws+2;
      new_a=(ws-1)/2;
      new_b=(ws-1)/2;
      if(ws>smax)
        flag=1;
        break;
      endif
      k=new_imsize(i-new_a:i+new_a,j-new_b:j+new_b);
      z_med = median(median(k));
      z_min = min(min(k));
      z_max = max(max(k));
      A1 = z_med-z_min;
      A2 = z_med-z_max;
    endwhile
    if(flag==1)
      s(i-a,j-b)=new_imsize(i,j);
    else
      B1 = new_imsize(i,j)-z_min;
      B2 = new_imsize(i,j)-z_max;
      if(B1>0 && B2<0),
        s(i-a,j-b)=new_imsize(i,j);
      else
        s(i-a,j-b)=z_med;
      endif
    endif
  endfor
```

```
endfor
s=uint8(s);
endfunction
```

Midpoint Filter:

```
function s=my_mid_point_filter(r,m,n)
[M,N]=size(r);
a=(m-1)/2;
b=(n-1)/2;
new_imsize = zeros(M+2*a,N+2*b);
new_imsize(1+a:M+a,1+b:N+b)= r;
s = zeros(size(r));
for i=1+a:M+a,
  for j=1+b:N+b,
    k=new_imsize(i-a:i+a,j-a:j+a);
    s(i-a,j-b)=(1/2)*(max(max(k))+min(min(k)));
  endfor
endfor
s=uint8(s);
endfunction
```

a).Implement homomorphic filter and apply it to your photo with pepper noise, salt noise and salt and pepper noise. Conclude the results.

Code:

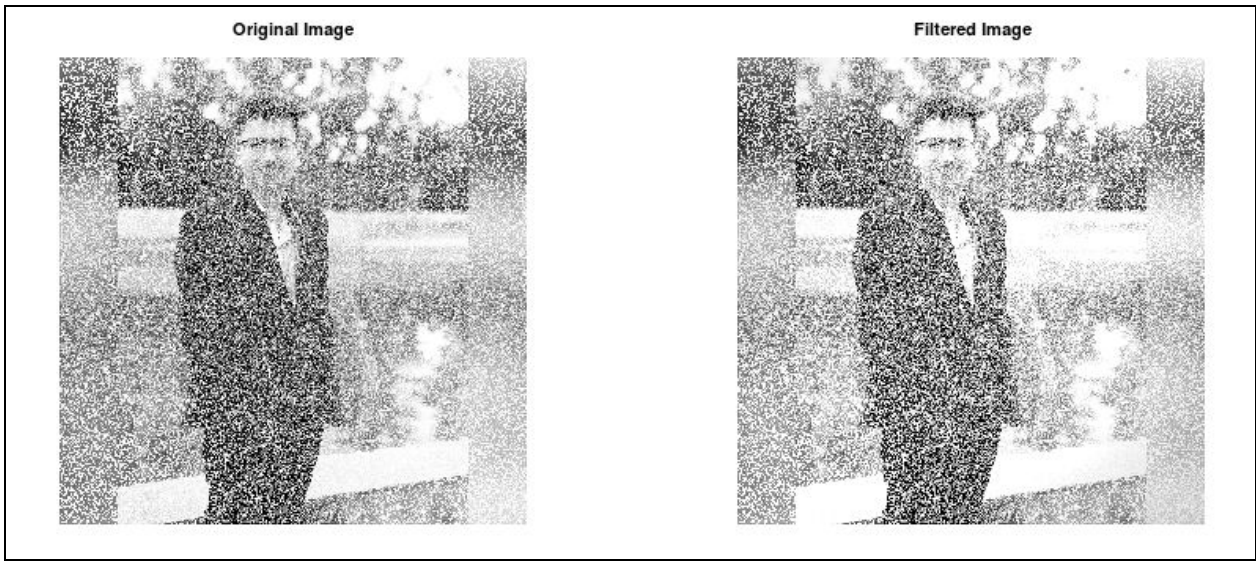
```
pkg load image;

r1 = imread("pepper_with_0.3p.jpg");
r1 = imresize(r1,[512,512]);
subplot(1,2,1);
imshow(r1);
title("Original Image");

s1 = my_homomorphic_filter(r1,1,5);
subplot(1,2,2);
imshow(s1);
title("Filtered Image");

figure;

r2 = imread("salt_with_0.5p.jpg");
```

```
r2 = imresize(r2,[512,512]);
subplot(1,2,1);
imshow(r2);
title("Originals Image");

s2 = my_homomorphic_filter(r2,1,5);
subplot(1,2,2);
imshow(s2);
title("Filtered Image");

figure;

r3 = imread("salt_pepper_with_0.2.jpg");
r3 = imresize(r3,[512,512]);
subplot(1,2,1);
imshow(r3);
title("Original Image");

s3 = my_homomorphic_filter(r3,1,5);
subplot(1,2,2);
imshow(s3);
title("Filtered Image");
```

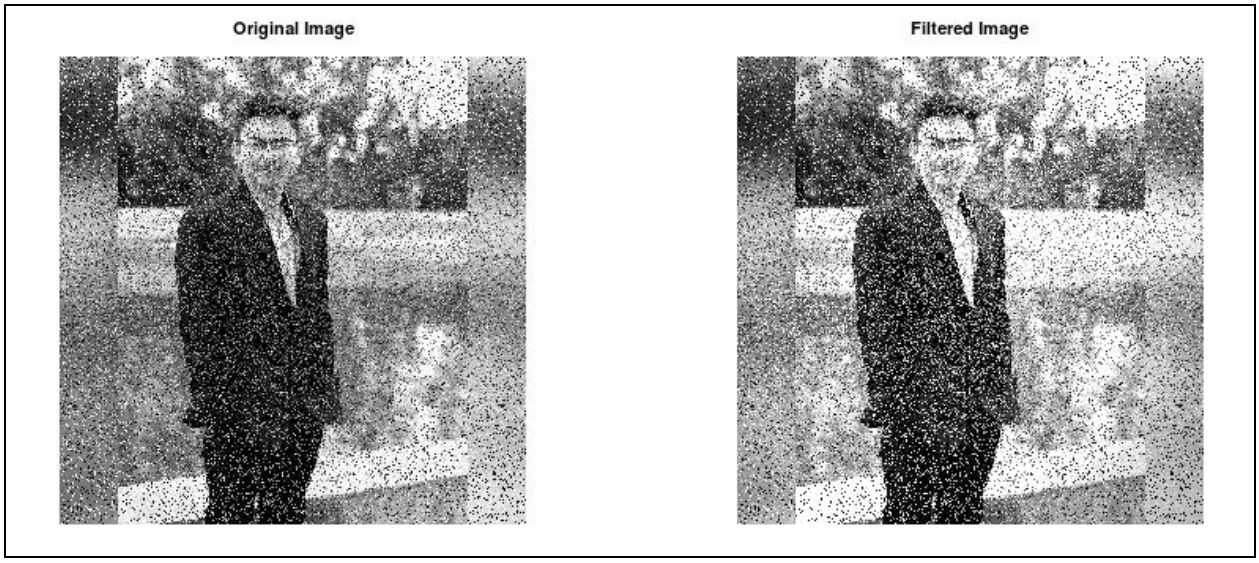
Output:

For pepper noise:

For Salt noise:



For Salt & Pepper noise:



Comment :

- After applying homomorphic filters, images get more brighter compared to original images.

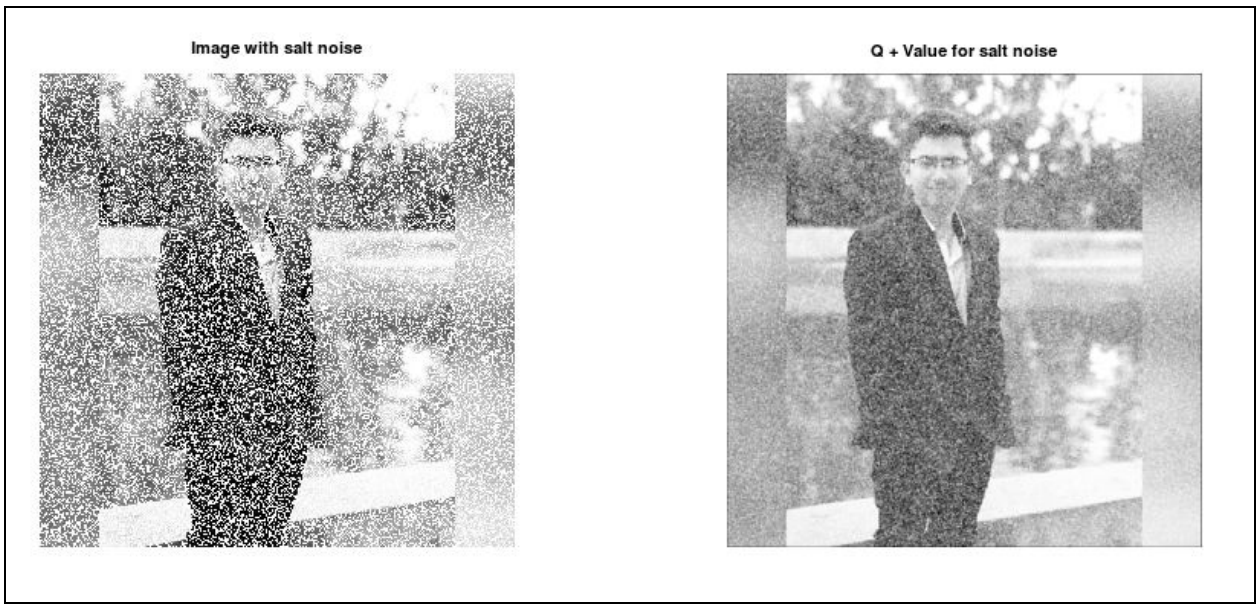
b).Use contra harmonic filter with Q +ve value for salt noise, pepper noise and salt and pepper noise. Comment on your outcome.

Code:

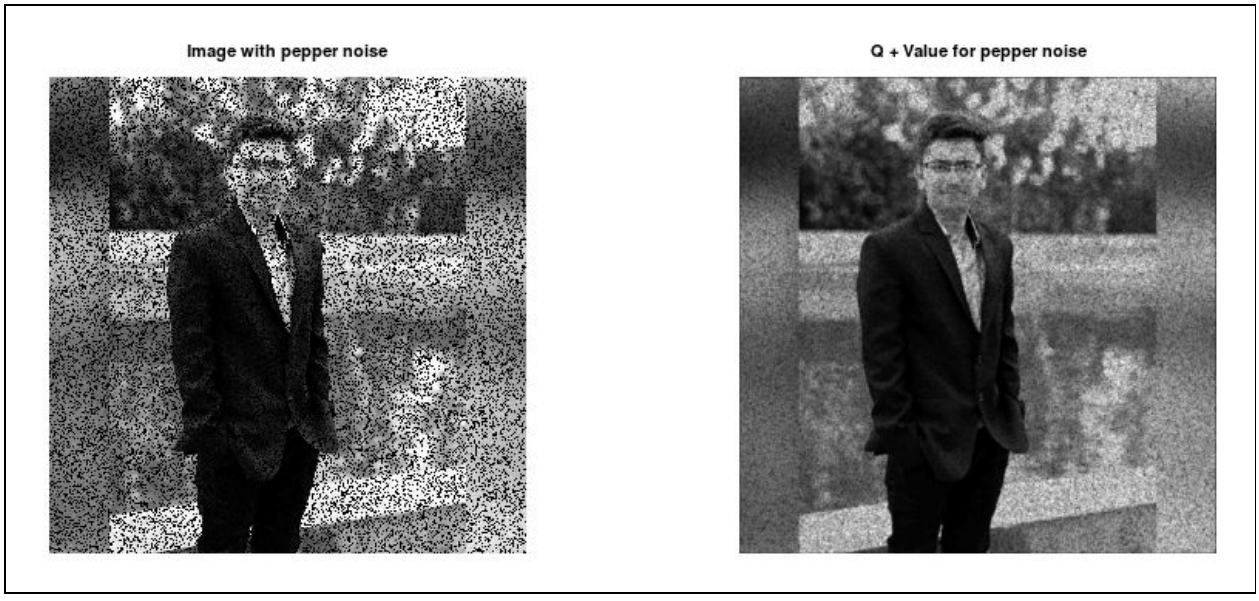
```
pkg load image;
r1 = imread("salt_with_0.5p.jpg");
subplot(1,2,1);
imshow(r1);
title("Image with salt noise");

s1 = my_contra_harmonic_mean_filter(r1,5,5,1);
subplot(1,2,2);
imshow(s1);
title("Q + Value for salt noise");

figure;

r2 = imread("pepper_with_0.3p.jpg");
subplot(1,2,1);
imshow(r2);
title("Image with pepper noise");

s2 = my_contra_harmonic_mean_filter(r2,5,5,1);
subplot(1,2,2);
imshow(s2);
title("Q + Value for pepper noise");

figure;

r3 = imread("salt_pepper_with_0.2.jpg");
subplot(1,2,1);
imshow(r3);
title("Image with salt & pepper noise");

s3 = my_contra_harmonic_mean_filter(r3,5,5,1);
subplot(1,2,2);
imshow(s3);
title("Q + Value for salt & pepper noise");
```

Output:

For Salt noise:



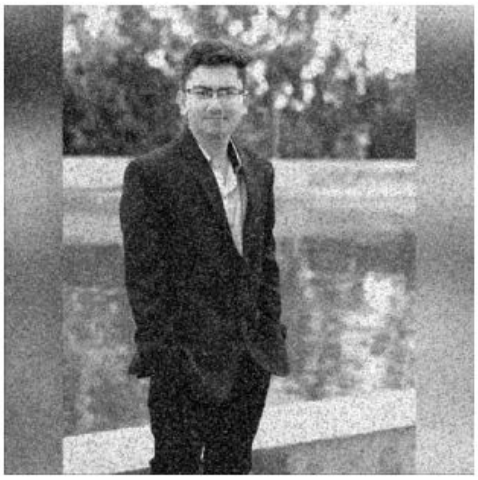
For Pepper noise:



For Salt & Pepper noise:

Image with salt & pepper noise

Q + Value for salt & pepper noise



Comment: Contra harmonic mean filter with positive values of Q, it eliminates pepper noise as we can see from above but not eliminates salt noise effectively. Image with salt-and-pepper noise it also works fine to eliminate noise.

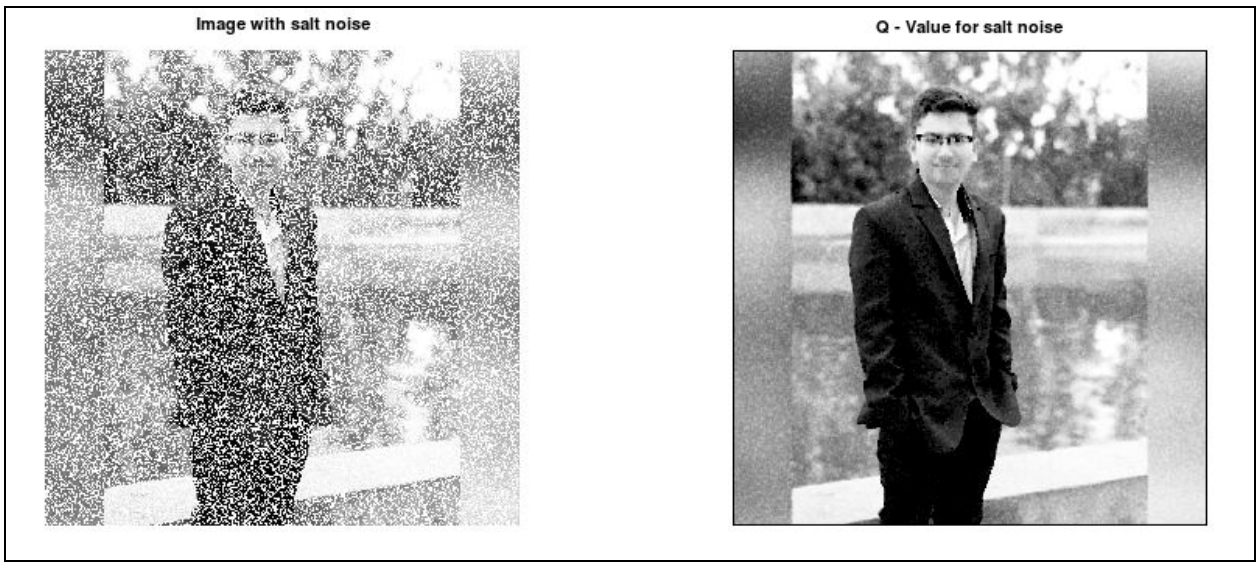
c).Use contra harmonic filter with Q values –ve for value for salt noise, pepper noise and salt and pepper noise. Comment on your outcome.

Code:

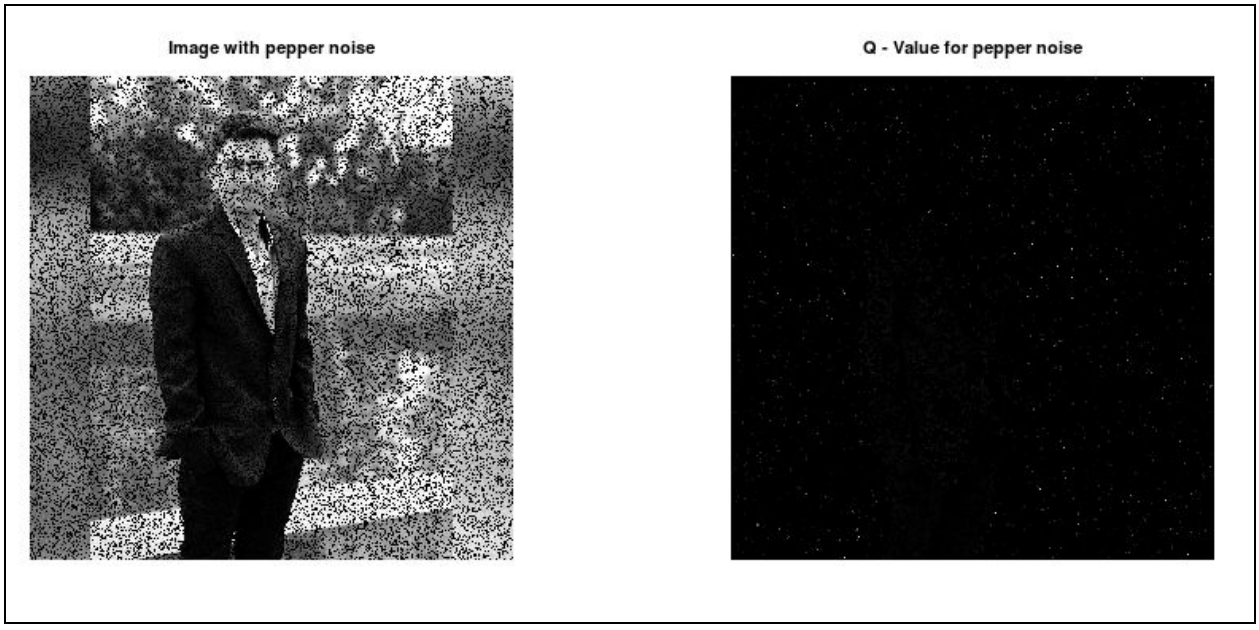
```
pkg load image;
r1 = imread("pepper_with_0.3p.jpg");
subplot(1,2,1);
imshow(r1);
title("Image with pepper noise");

s1 = my_contra_harmonic_mean_filter(r1,5,5,-1);
subplot(1,2,2);
imshow(s1);
title("Q - Value for pepper noise");

figure;

r2 = imread("salt_with_0.5p.jpg");
subplot(1,2,1);
imshow(r2);
title("Image with salt noise");

s2 = my_contra_harmonic_mean_filter(r2,5,5,-1);
subplot(1,2,2);
imshow(s2);
title("Q - Value for salt noise");

figure;

r3 = imread("salt_pepper_with_0.2.jpg");
subplot(1,2,1);
imshow(r3);
title("Image with salt & pepper noise");

s3 = my_contra_harmonic_mean_filter(r3,5,5,-1);
subplot(1,2,2);
imshow(s3);
title("Q - Value for salt & pepper noise");
```

Output:

For Salt noise:



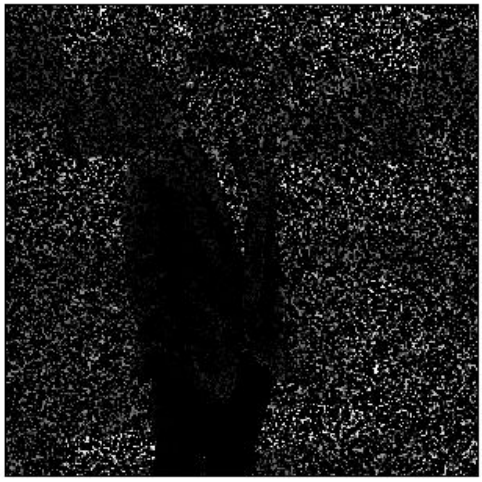
For Pepper noise:



For Salt & Pepper noise:

Image with salt & pepper noise

Q - Value for salt & pepper noise



Comment: Contra harmonic mean filter with negative values of Q, it eliminates salt noise as we can see from above, but is not able to eliminate pepper noise & salt-and-pepper noise. For Q=-1 it acts as a harmonic mean filter.

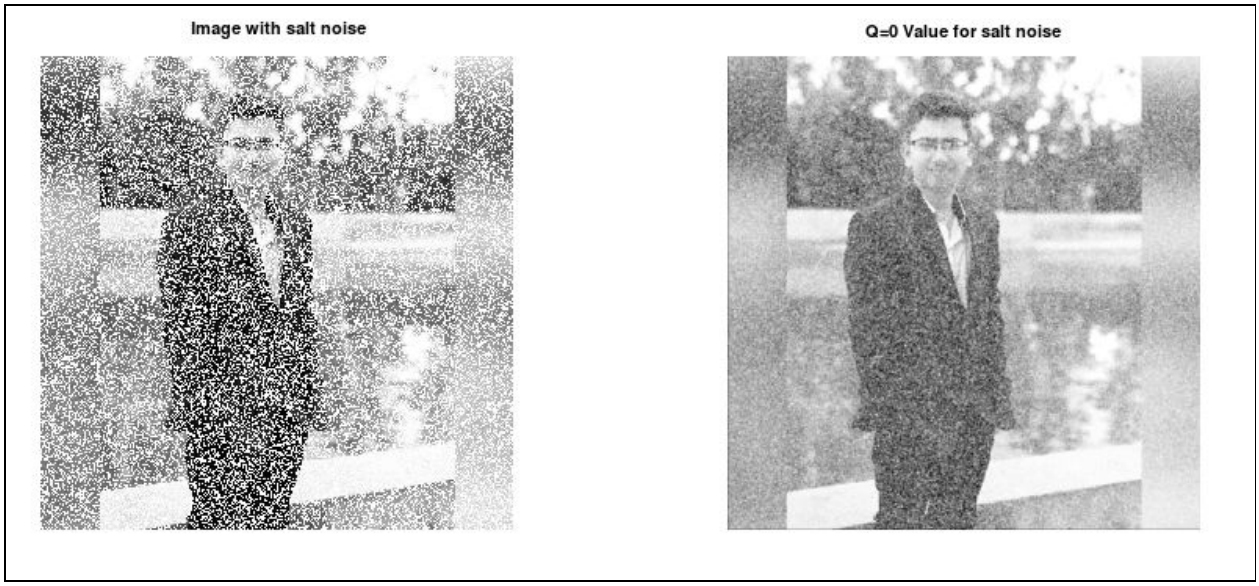
d).If Q value is taken zero in a contra harmonic filter it will be suitable for which kind of noise? Why?Justify your answer by implementation and results.

Code:

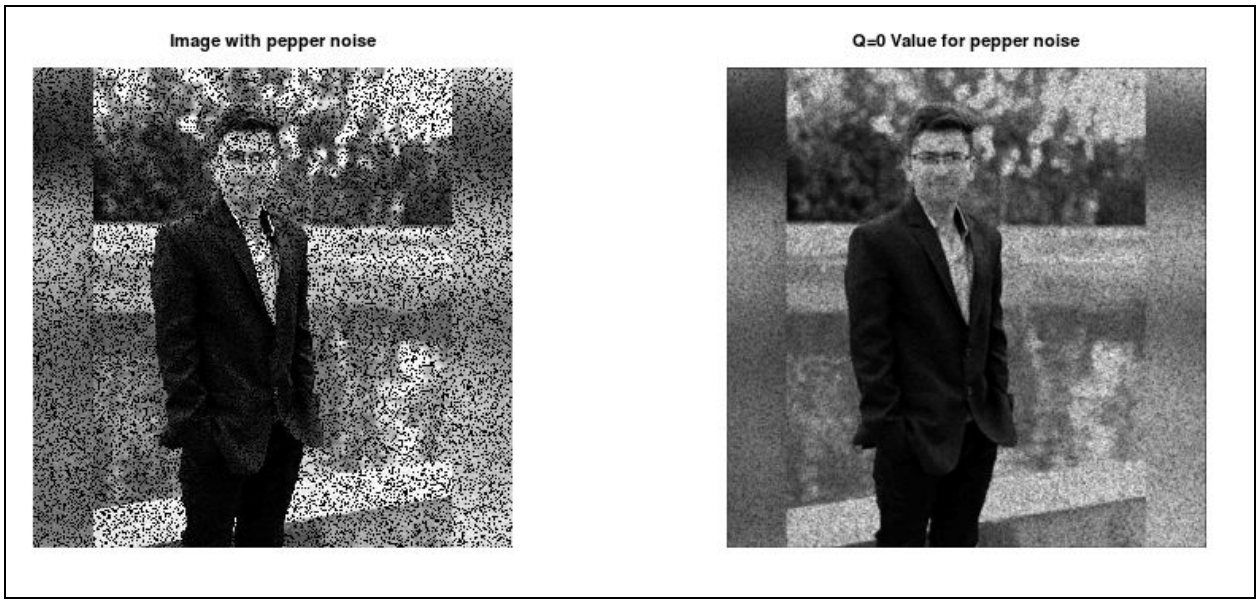
```
pkg load image;

r1 = imread("salt_with_0.5p.jpg");
subplot(1,2,1);
imshow(r1);
title("Image with salt noise");

s1 = my_contra_harmonic_mean_filter(r1,5,5,0);
subplot(1,2,2);
imshow(s1);
title("Q=0 Value for salt noise");

figure;

r2 = imread("pepper_with_0.3p.jpg");
subplot(1,2,1);
imshow(r2);
title("Image with pepper noise");

s2 = my_contra_harmonic_mean_filter(r2,5,5,0);
subplot(1,2,2);
imshow(s2);
title("Q=0 Value for pepper noise");

figure;

r3 = imread("salt_pepper_with_0.2.jpg");
subplot(1,2,1);
imshow(r3);
title("Image with salt & pepper noise");

s3 = my_contra_harmonic_mean_filter(r3,5,5,0);
subplot(1,2,2);
imshow(s3);
title("Q=0 Value for salt & pepper noise");
```

Output:

For Salt noise:



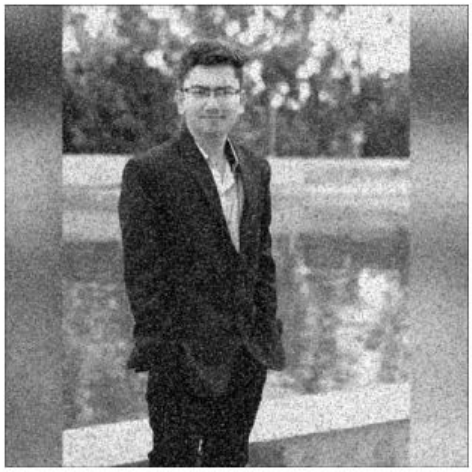
For Pepper noise:



For Salt & Pepper noise :

Image with salt & pepper noise

Q=0 Value for salt & pepper noise



Comment:

- Contra harmonic filter with 0 value of Q acts as an arithmetic mean filter.
- It smooth's local variations in an image and noise is reduced as a result of blurring.It is useful for eliminate pepper noise and salt and pepper noise because arithmetic mean will be calculated by ignoring the noisy pixel i.e., 0 and 255.If done this way, this new approach gives better results than the distorted image.

e).Apply arithmetic mean filter and geometric mean filter of various sizes to noisy image. Compare geometric mean filter and arithmetic mean filter in terms of blurring.

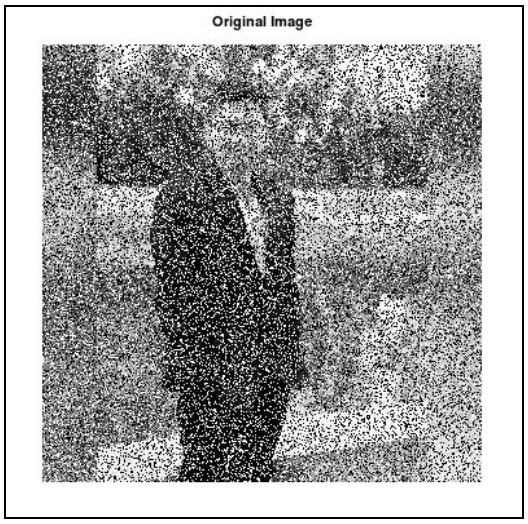
Code:

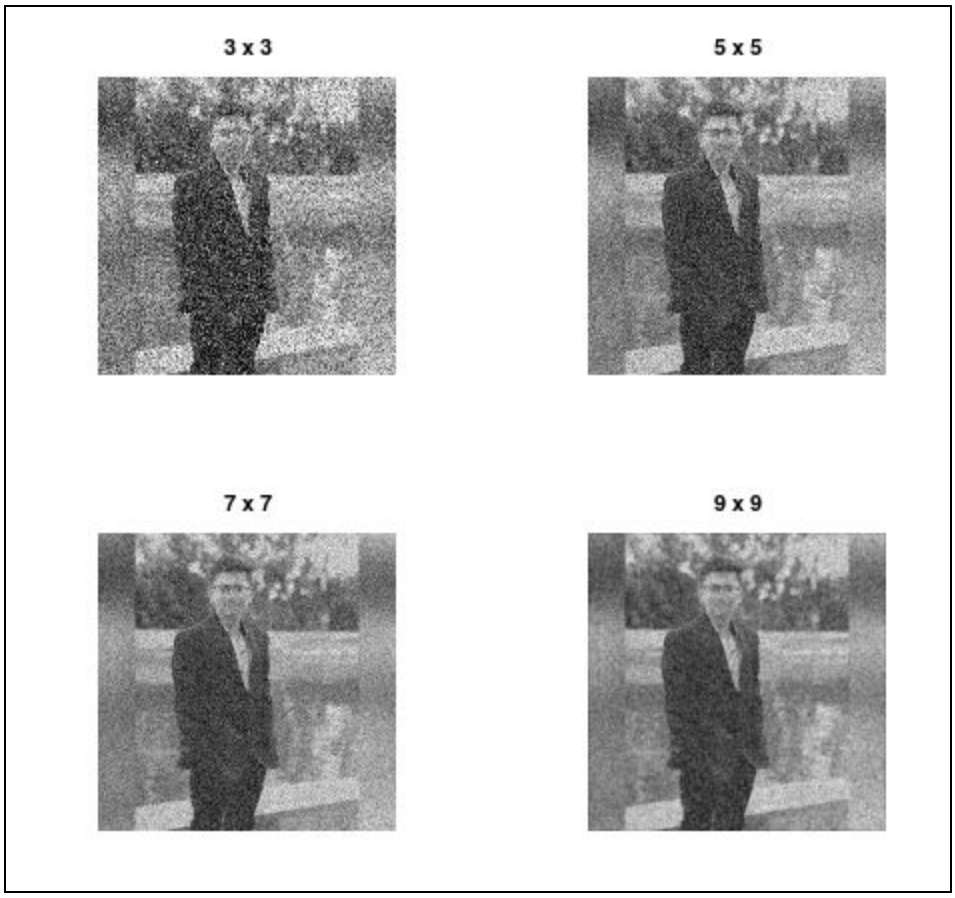
```
ra = imread("salt_pepper_random_prob.jpg");
imshow(r);
title("Original Image");

figure;

#Arithmetic filter
s1 = my_arithmetic_mean_filter(ra,3,3);
subplot(2,2,1);
imshow(s1);
title("3 x 3");

s2 = my_arithmetic_mean_filter(ra,5,5);
subplot(2,2,2);
imshow(s2);
title("5 x 5");

s3 = my_arithmetic_mean_filter(ra,7,7);
subplot(2,2,3);
imshow(s3);
title("7 x 7");

s4 = my_arithmetic_mean_filter(ra,9,9);
subplot(2,2,4);
imshow(s4);
title("9 x 9");

figure;

rb = imread("salt_with_0.5p.jpg");
imshow(r);
title("Original Image");

figure;

#Geometric filter
s1 = my_geometric_mean_filter(rb,3,3);
subplot(2,2,1);
imshow(s1);
title("3 x 3");

s2 = my_geometric_mean_filter(rb,5,5);
```

```
subplot(2,2,2);
imshow(s2);
title("5 x 5");

s3 = my_geometric_mean_filter(rb,7,7);
subplot(2,2,3);
imshow(s3);
title("7 x 7");

s4 = my_geometric_mean_filter(rb,9,9);
subplot(2,2,4);
imshow(s4);
title("9 x 9");
```

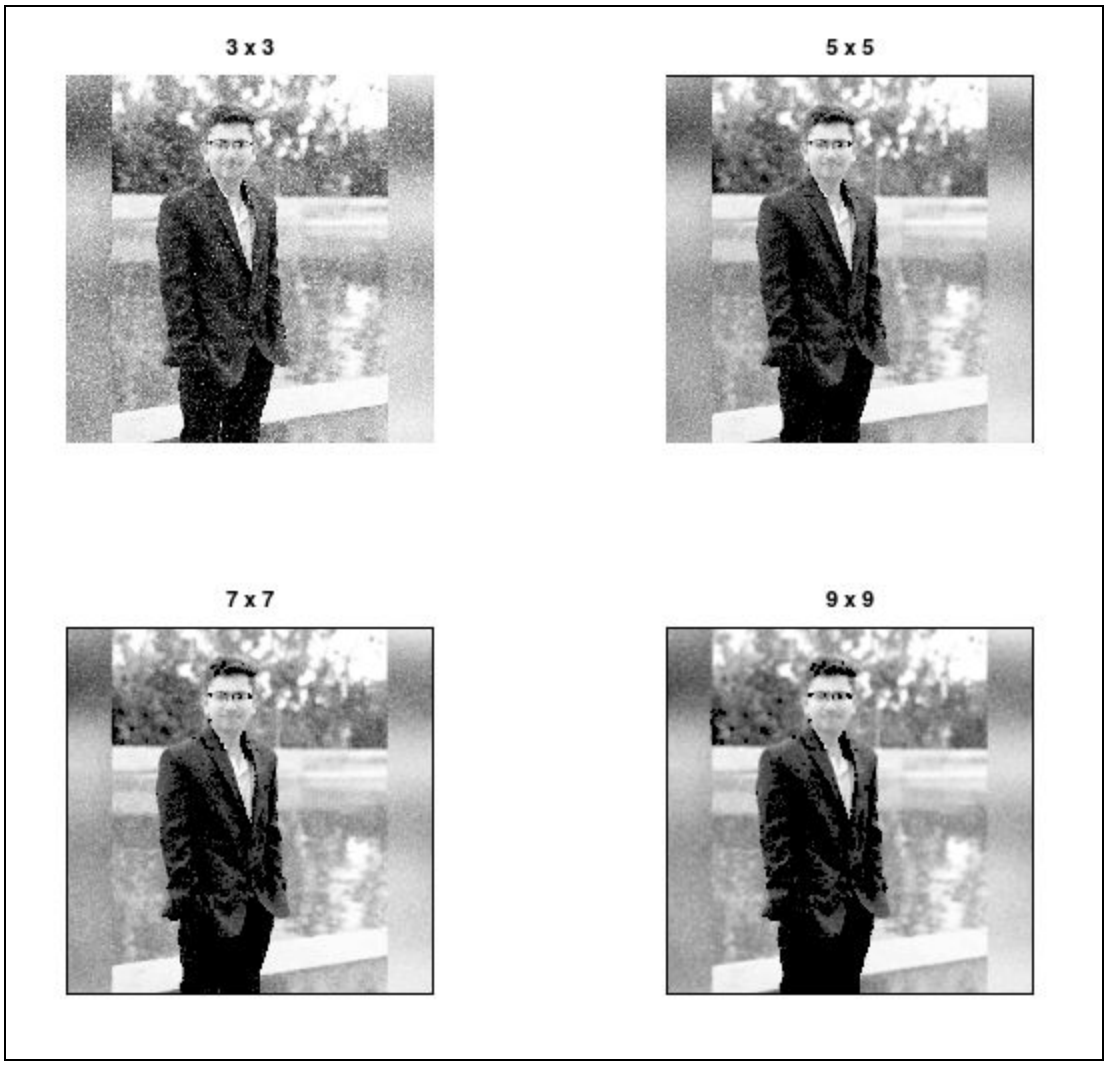
Output:

For Arithmetic mean filter:





For Geometric mean:



Comment :

- Arithmetic mean filter smooth's local variation in an image and noise is reduced as a result of blurring.
- But in Geometric mean filter each pixel is given by the product of the pixels in the sub image window, raising the power 1/mn. Hence, It achieves smoothing comparable to the arithmetic mean filter, but it tends to **less image detail** in the process.

f).Implement Adaptive median filter. compare the results with the median filter and comment on the outcome.

Code:

```
r = imread('sample_noise.jfif');
subplot(1,3,1);
imshow(r);
title("Original Image");

s1 = my_adaptive_median_filter(r,3,3);
subplot(1,3,2);
imshow(s1);
title("After Applying Adaptive Median Filter");

s2 = my_median_filter(r,3,3);
subplot(1,3,3);
imshow(s2);
title("After Applying Median Filter");
```

Output:

Comment:

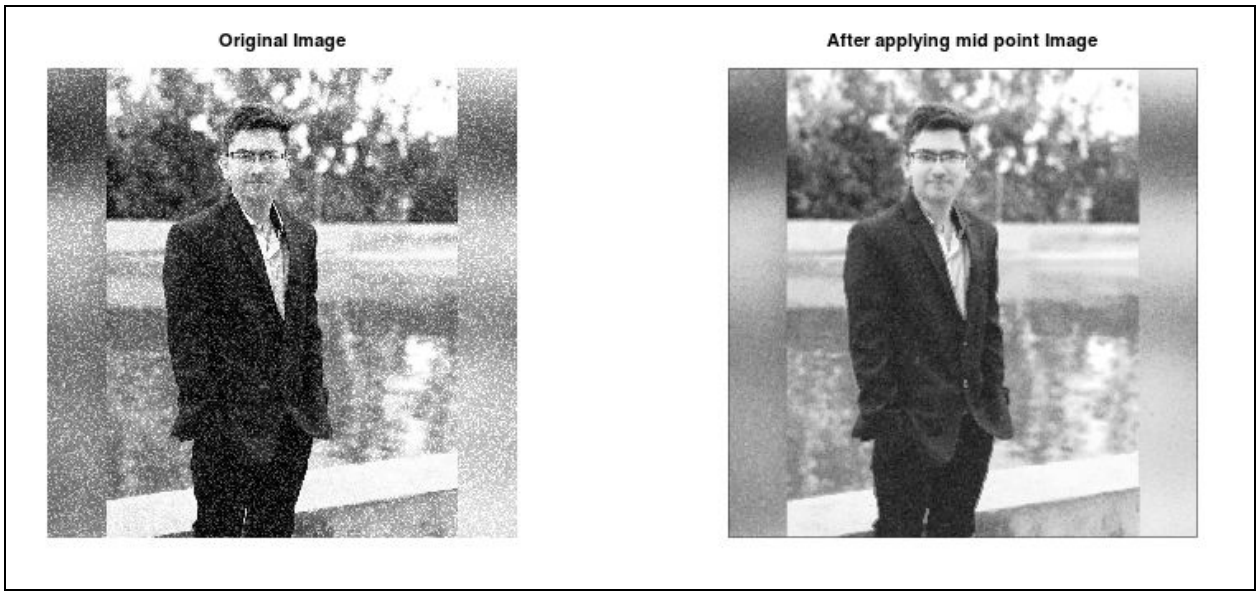
- By Using adaptive median filter, it removes noise from the image with less effect of blurring compared to median filter and also preserves details while smoothing non impulse noise.

g).Implement midpoint filter and apply it on your photo with Gaussian noise.

Code:

```
r = imread("Gaussian_with_0.4.jpg");
subplot(1,2,1);
imshow(r);
title("Original Image");

s = my_mid_point_filter(r,5,5);
subplot(1,2,2);
imshow(s);
title("After applying mid point Image");
```

Output:



h).Take your photo with salt and pepper noise (probability 0.4) and Apply multiple passes of median filter and conclude about the outcome and blurring. Also apply median filter of different sizes (3x3, 5x5,7x7) comment on the outcome.

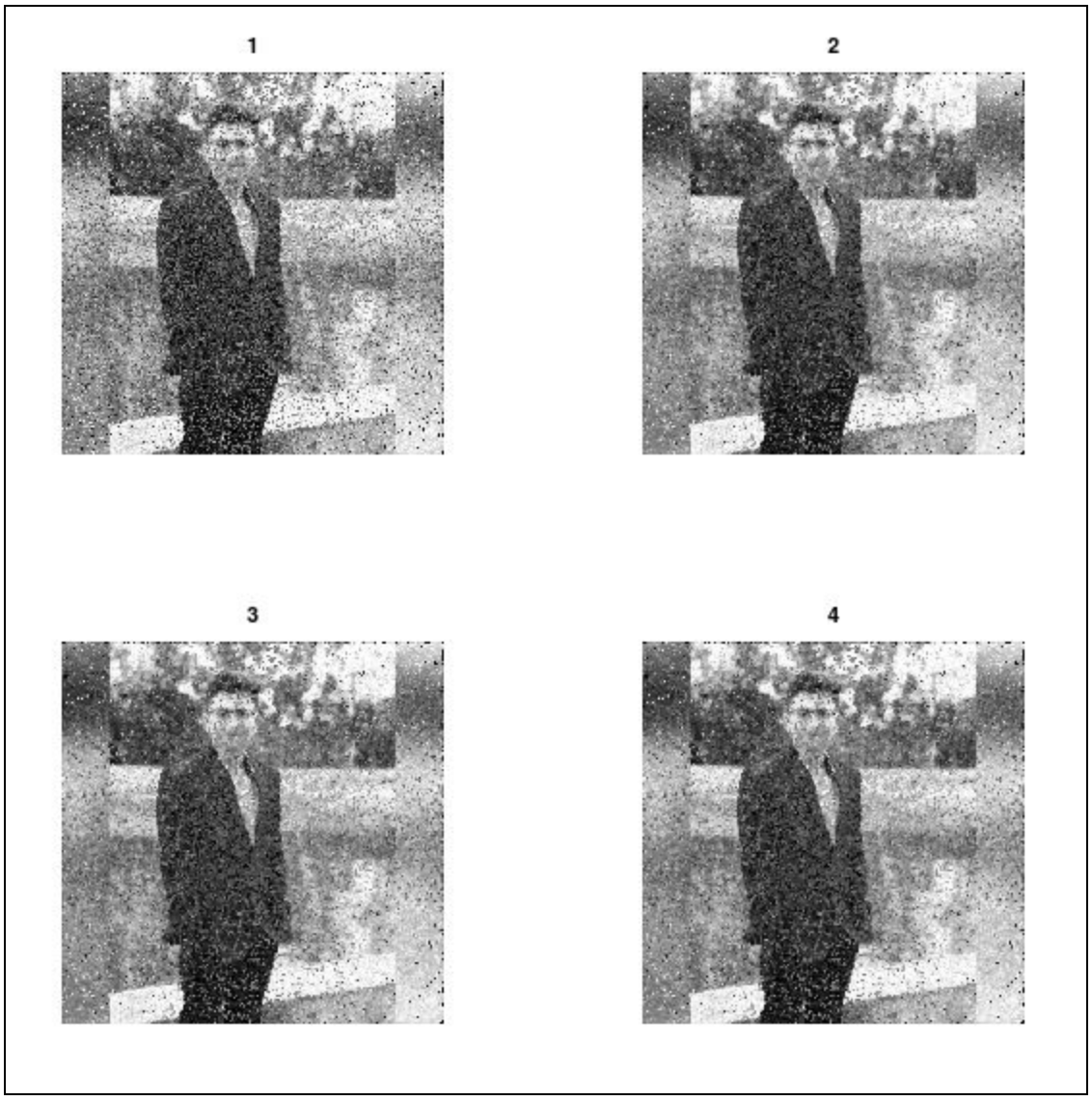
Code:

```
r=imread("salt&pepper(0.4).jpg");
r=imresize(r,[512,512]);
imshow(r);
title("Original Image");

figure;

s1 = my_median_filter(r,3,3);
subplot(2,2,1);
imshow(s);
title("1");

for i=2:4,
  s1 = my_median_filter(s1,3,3);
  subplot(2,2,i);
  imshow(s1);
  title(i);
endfor

figure;

s2 = my_median_filter(r,5,5);
subplot(2,2,1);
imshow(s);
title("1");

for i=2:4
  s2 = my_median_filter(s2,5,5);
  subplot(2,2,i);
  imshow(s2);
  title(i);
endfor

figure;

s3 = my_median_filter(r,7,7);
subplot(2,2,1);
imshow(s3);
title("1");
```

```
for i=2:4
  s3 = my_median_filter(s3,7,7);
  subplot(2,2,i);
  imshow(s3);
  title(i);
endfor

figure;

s4 = my_median_filter(r,9,9);
subplot(2,2,1);
imshow(s4);
title("1");

for i=2:4
  s4 = my_median_filter(s4,9,9);
  subplot(2,2,i);
  imshow(s4);
  title(i);
endfor
```

Output:

Filter of 3x3:



Filter of 5x5:

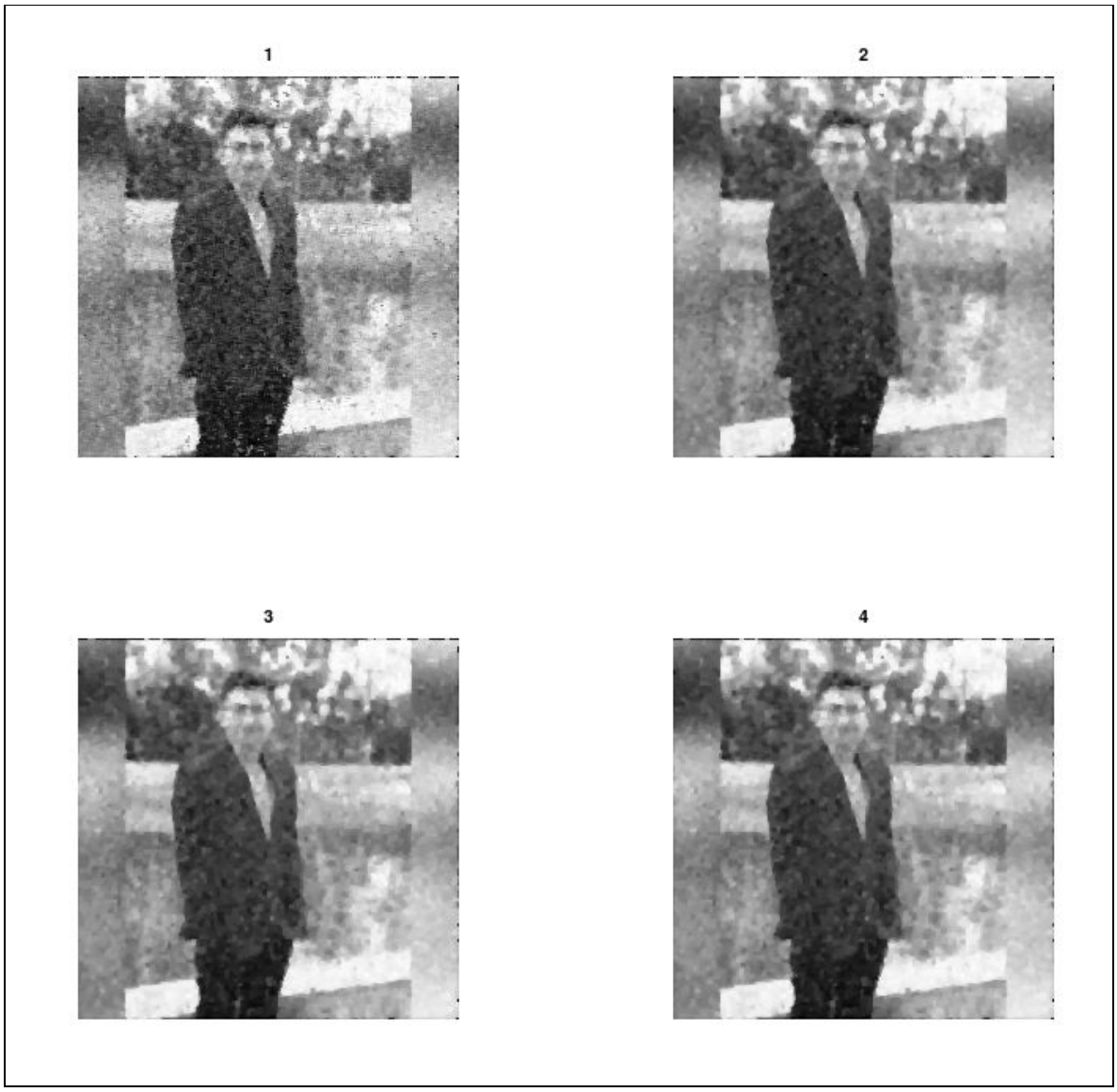
1

2

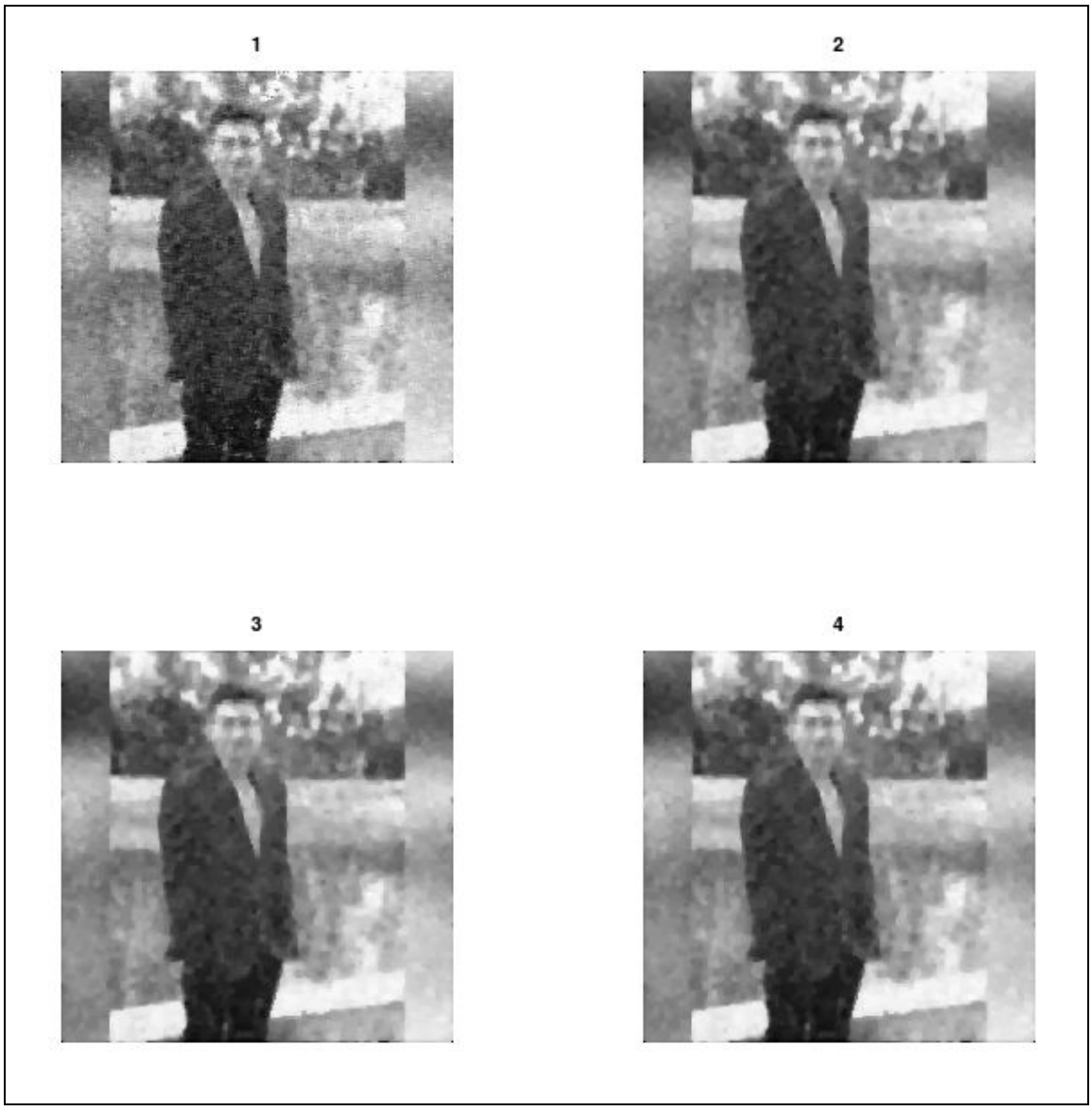
3

4

Filter of 7x7:



Filter of 9X9:



Comment :

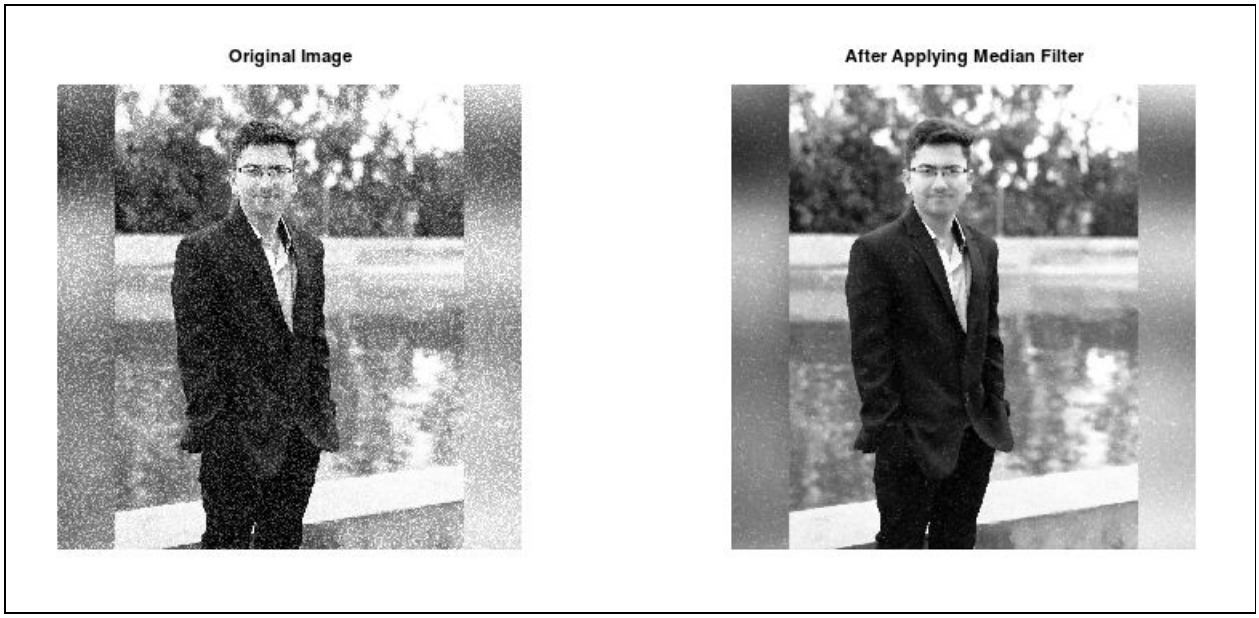
- As we increase the filter size i.e, 3x3 to 5x5 , 5x5 to 7x7 blurring effect increase effectively and removes salt and pepper noise as we can see above.
- For multiple passes of median filter it does not affect any kind of image so far, because after one pass the median will be always the same for sub image.

i).Take your photo with Gaussian noise and apply a median filter over it. Comment on the outcome.

Code:

```
r = imread("Gaussian_with_0.4.jpg");
subplot(1,2,1);
imshow(r);
title("Original Image");

s = my_median_filter(r,5,5);
subplot(1,2,2);
imshow(s);
title("After Applying Median Filter");
```

Output:



Comment :

- After applying a median filter it removes noise with less loss of detail effectively.

j). Implement 1-D Discrete Fourier Transform and Inverse Discrete Fourier Transform.

Code:

```
f = [1,0,0,1];

# 1-D Discrete Fourier Transform
[S,M] = size(f);
F = zeros(S,M);
for i=1:M,
   for k=1:M
     F(i) = F(i)+f(k)*e.^(-j*2*pi*((i-1)*(k-1))/M);
   endfor
endfor

# 1-D Inverse Discrete Fourier Transform
[IS,IM] = size(F);
inverse_f = zeros(IS,IM);
for i=1:IM,
  for k=1:IM,
    inverse_f(i) = inverse_f(i)+F(k)*e.^(j*2*pi*((k-1)*(i-1)/M));
  endfor
  inverse_f(i) = (1/IM)*(inverse_f(i));
endfor
```

Output:

```
>> F
F =

   2.00000 + 0.00000i   1.00000 + 1.00000i   0.00000 - 0.00000i   1.00000 - 1.00000i

>> inverse_f
inverse_f =

   1.00000 - 0.00000i  -0.00000 + 0.00000i   0.00000 + 0.00000i   1.00000 + 0.00000i
```