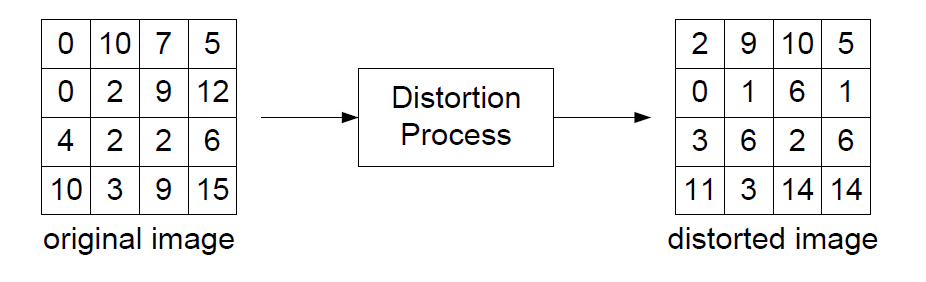
ECE 613, Winter 2022, Homework 1

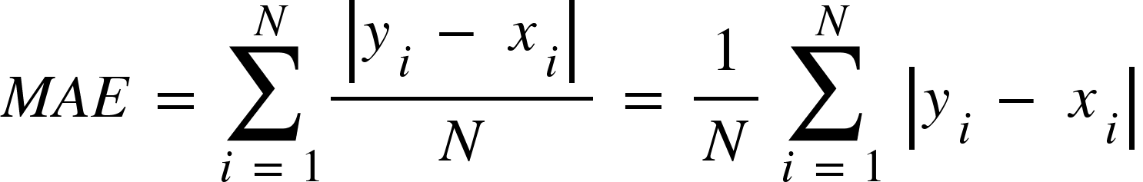
Parth Nitinbhai Patel, WatID: 20942013

**Q1 Solution:**



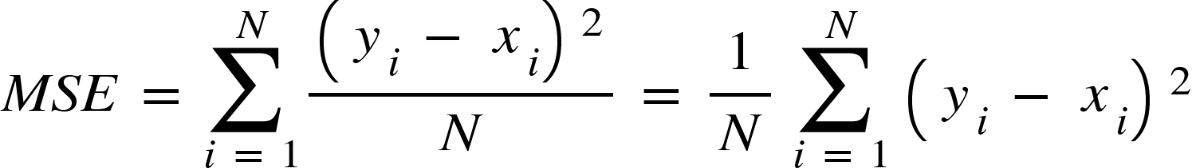
Let suppose, the original image is denoted by *x* and the distorted image is denoted by *y. N* represents the total number of pixels. *xi* denotes the *ith* pixel in the original image and similarly, *yi* denotes the *ith* pixel in the distorted image. So, the value of *i* ranges from 1 – 16, because as we can see that each image represents 16 pixels in total. Hence, *N* = 16.

**Part (a) Mean Absolute Error (MAE):**



equals space fraction numerator 2 space plus space 1 space plus space 3 thin space plus space 0 thin space plus space 0 space plus space 1 space plus space 3 space plus 11 space plus space 1 space plus space 4 space plus space 0 space plus space 0 space space plus space 1 space plus space 0 space plus space 5 space plus space 1 space over denominator 16 end fraction
equals space 33 over 16
bold equals bold space bold 2 bold. bold space bold 0625

**Part (b) Mean Squared Error (MSE):**



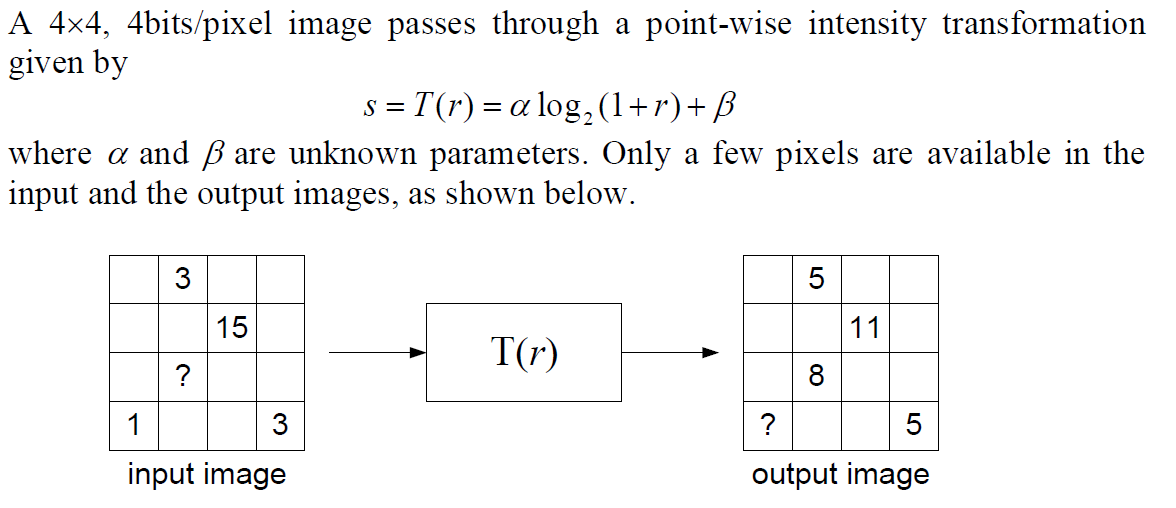
equals space fraction numerator 4 space plus space 1 space plus space 9 space plus space 0 space plus space 0 space plus space 1 space plus space 9 space plus space 121 space plus space 1 space plus space 16 space plus space 0 space plus space 0 space plus 1 space plus space 0 space plus 25 space plus space 1 over denominator 16 end fraction
equals space 189 over 16
equals space 11.8125

**Part (c) Peak Signal-to-Noise Ratio (PSNR:**

Here, we are taking 4 bits/pixel. So, the MAX*I* is the maximum possible intensity which is given as 2B – 1, where B is the number of bits. So for our given example MAX*I* = (24 – 1) = 15.

P S N R thin space equals space 10 log open parentheses fraction numerator M A X subscript I superscript 2 over denominator M S E end fraction close parentheses
P S N R space equals space 10 log open parentheses fraction numerator 225 over denominator 11.8125 end fraction close parentheses
P S N R space equals space 12.7984 space d B

**Q2 Solution:**



From the above given values in the input and output images we can form 2 equations to solve the value of α and β. These two equations are shown below:

5 = α log(1 + 3) + β

**5 = 2α + β**

11 = α log(1 + 15) + β

**11 = 4α + β**

**Part (a):**

Solving the above two equations we get the values of α and β as:

**α = 3, β = -1.**

**Part (b):**

Value of pixel in third row:

Let the value be r. Then,

8 = α log2(1+r) + β

8 = 3 log2(1+r) -1 (Substituting the values of α and β)

3 = log2(1+r)

**r = 7**

**Part (c):**

Value of pixel in fourth row:

Let the value of pixel be s. Then,

s = α log2(1+1) + β

s = 3 log2(1+1) -1 (Substituting the values of α and β)

**s = 2**

**Q3 Solution:**

We know that the intensity value of black pixel corresponds to 0 and the intensity value of white pixel corresponds to 1.

A] s space equals space r squared

This above **Gamma transformation** will make the output image darker since the intensity values range between 0 - 1 and square of such a number will be less than itself. Therefore, the resulting image from this transformation should be image **E**.

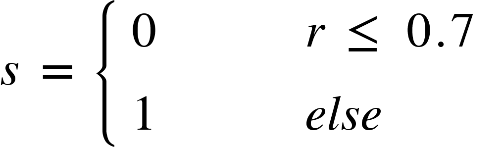
B] s space equals space 1 space minus space r

This above transformation is similar if we invert an image. It follows **inverting pixel intensities** and hence, it can be seen in the output image **C**.

C] s space equals space 0.5 r space plus space 0.25

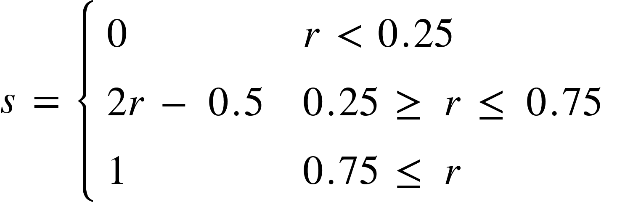
From the above **Reducing Contrast** transformation equation, it can be seen that it halving the original pixel intensity and adds a small marginal value back, which result in the new intensity that is lower than the original when original value is greater than 0.5 and greater than the original when the original value is less than 0.5. And hence, the output image shows lightness characteristic in darker regions and visa versa. The same can be seen in image **D**.

D]



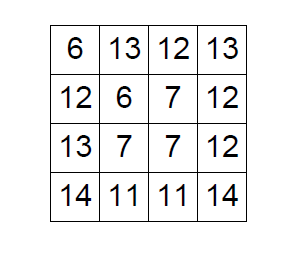
The above **Thresholding** transformation will convert the input image into a binary image as it is assigning either 0 or 1 depending upon the threshold values. Hence, this can be seen in option **B**.

E]



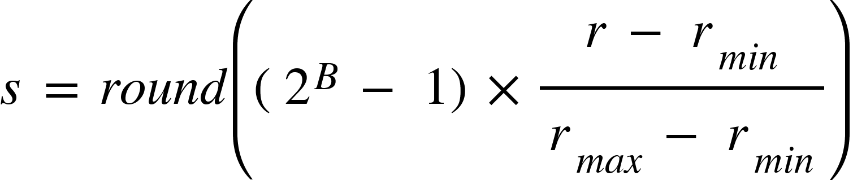
Here, with the help of the above transformation the lower pixel values are made darker (intense) and the higher pixel values are made lighter. The middle range is spreading out using a linear transformation as can be seen from the equation, which makes the image even brighter. This can be seen in **A**.

**Q4 Solution:**



**Part (a):**

The Full-scale contrast stretch is done with the help of following equation:



Here, rmax = maximum value from the above input image = 14, and

rmin = minimum value from the above input image = 6. It is also given that the input image is 4 bits/pixel, so (2B - 1) = 15.

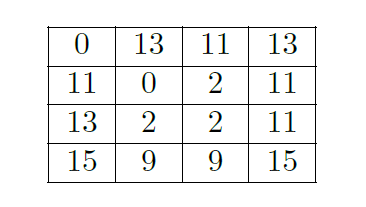
Hence,

s space equals space r o u n d open parentheses 1.875 space cross times space open parentheses r space minus space 6 close parentheses close parentheses

Substituting all the pixel values from the above given input image we have:

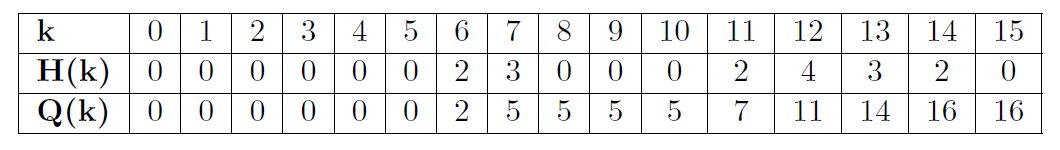
s open parentheses 6 close parentheses space equals space r o u n d open parentheses 1.875 space cross times space open parentheses 6 space minus space 6 close parentheses close parentheses space equals space 0
s open parentheses 7 close parentheses space equals space r o u n d open parentheses 1.875 space cross times space open parentheses 7 space minus space 6 close parentheses close parentheses space equals space 2
s open parentheses 11 close parentheses space equals space r o u n d open parentheses 1.875 space cross times space open parentheses 11 space minus space 6 close parentheses close parentheses space equals space 9
s open parentheses 12 close parentheses space equals space r o u n d open parentheses 1.875 space cross times space open parentheses 12 space minus space 6 close parentheses close parentheses space equals space 11
s open parentheses 13 close parentheses space equals space r o u n d open parentheses 1.875 space cross times space open parentheses 13 space minus space 6 close parentheses close parentheses space equals space 13
s open parentheses 14 close parentheses space equals space r o u n d open parentheses 1.875 space cross times space open parentheses 14 space minus space 6 close parentheses close parentheses space equals space 15

Using the above calculated values and putting those values we get the following image:

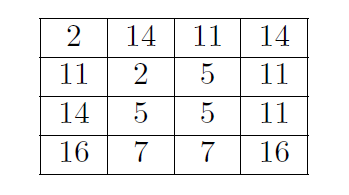


**Part (b):**

In order to perform histogram equalization, we first need to compute cumulative histogram Q(k) from the histogram H(k) and finally apply the full-scale contrast stretch to obtain the desired result.



Hence, the intermediate image will be like this:



Now, we will apply the full-scale contrast stretch to this image. So, from the above image we get rmax = 16 and rmin = 2. It is also given that the input image is 4 bits/pixel, so (2B - 1) = 15.

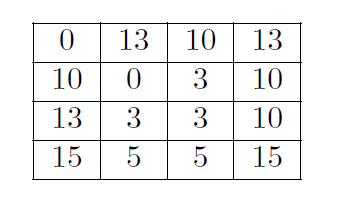
Therefore,

s space equals space r o u n d open parentheses 1.071 space cross times space open parentheses r space minus space 2 close parentheses close parentheses

Substituting all the pixel values from the above given input image we have:

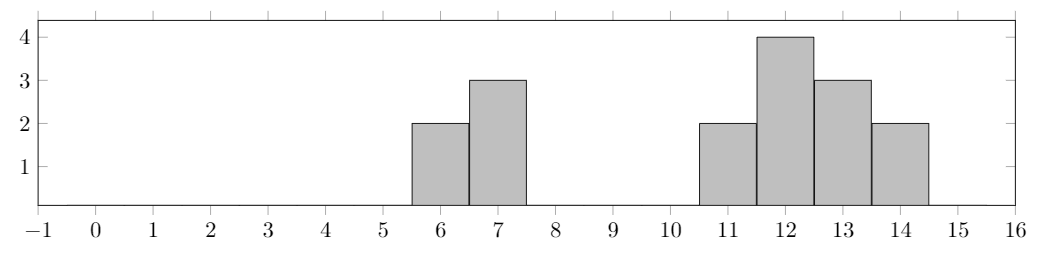
s open parentheses 2 close parentheses space equals space r o u n d open parentheses 1.071 space cross times space open parentheses 2 space minus space 2 close parentheses close parentheses space equals space 0
s open parentheses 5 close parentheses space equals space r o u n d open parentheses 1.071 space cross times space open parentheses 5 space minus space 2 close parentheses close parentheses space equals space 3
s open parentheses 7 close parentheses space equals space r o u n d open parentheses 1.071 space cross times space open parentheses 7 space minus space 2 close parentheses close parentheses space equals space 5
s open parentheses 11 close parentheses space equals space r o u n d open parentheses 1.071 space cross times space open parentheses 11 space minus space 2 close parentheses close parentheses space equals space 10
s open parentheses 14 close parentheses space equals space r o u n d open parentheses 1.071 space cross times space open parentheses 14 space minus space 2 close parentheses close parentheses space equals space 13
s open parentheses 16 close parentheses space equals space r o u n d open parentheses 1.071 space cross times space open parentheses 16 space minus space 2 close parentheses close parentheses space equals space 15

Using the above calculated values and putting those values we get the following image:

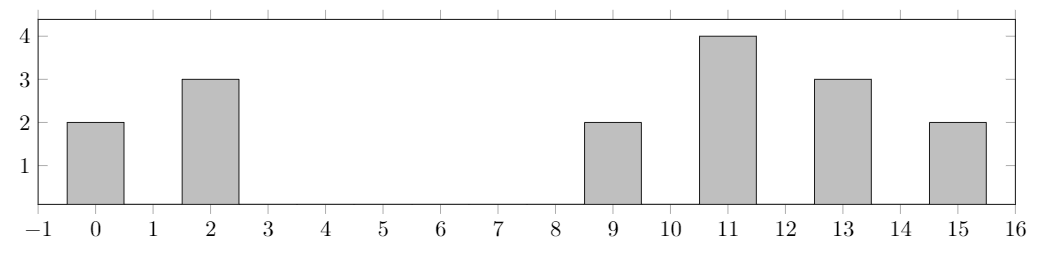


**Part (c):**

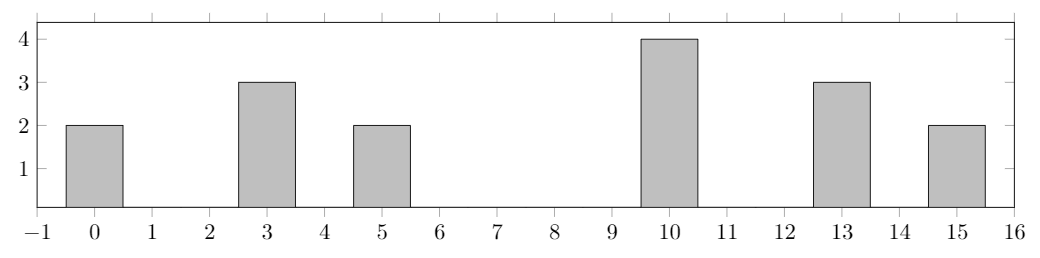
**Histogram of the original image:**



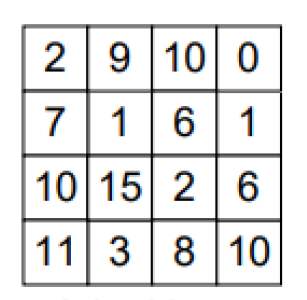
**Histogram of Fill-Scale Contrast Stretched image:**



**Histogram of histogram equalized image:**

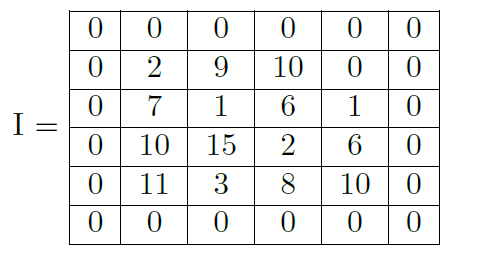


**Q5 Solution:**



Original Image

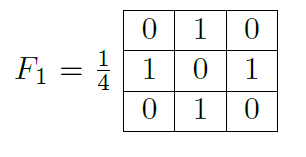
Zero padding is done in order to get the same dimensions of the output image as that of the input image. Hence, the zero padded image is given below:



**Part (a):**

Shift-Invariant Linear Filter is given as:

Filter 1:



To calculate the output, we will slide the filter F1 on the zero padded image I with a stride of 1. Here, we will also take the inner product. Thus, the pixel values of the output image will be the sum of all the elements of the inner product.

O(*m,n*) =

Hence, the filtered output image 1 is,

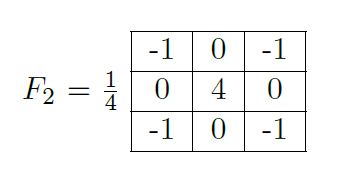
|  |  |  |  |
| --- | --- | --- | --- |
| 4 | 3.25 | 3.75 | 2.75 |
| 3.25 | 9.25 | 3.5 | 3 |
| 8.25 | 4 | 8.75 | 3.25 |
| 3.25 | 8.5 | 3.75 | 3.5 |

O1 =

**Part (b):**

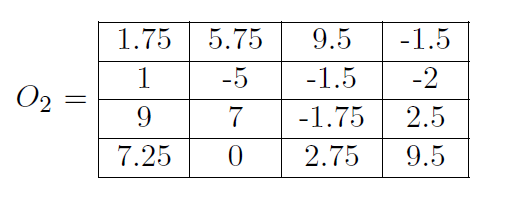
Shift-Invariant Linear Filter is given as:

Filter 2:



To calculate the output, we will slide the filter F1 on the zero padded image I with a stride of 1. Here, we will also take the inner product. Thus, the pixel values of the output image will be the sum of all the elements of the inner product.

Hence, the filtered output image 2 is,

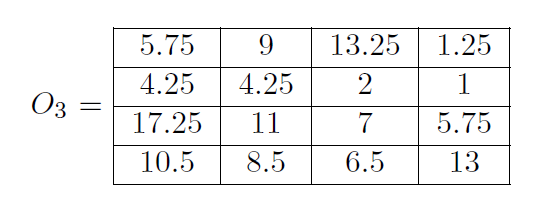


**Part (c):**

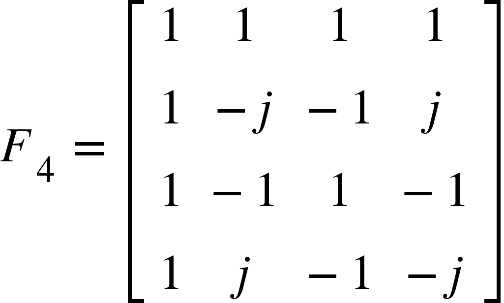
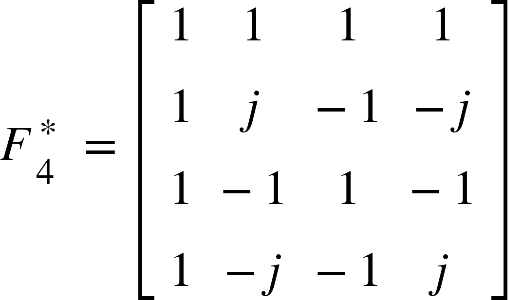
Here, the filter 3 is the combination of filter 1 and filter 2. So, F3 = F1 + F2. So, we can get the penultimate step image by addition of images we get from Part (a) and Part (b) without the division operations. We can directly add those images and perform division operation on the resulting image to get the final image as the result.

Hence,

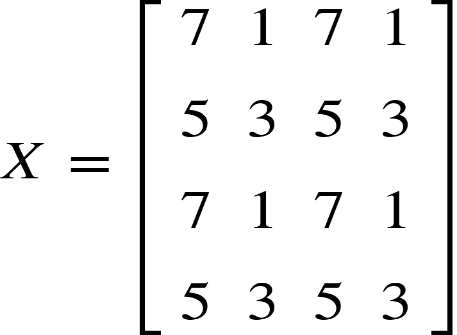
O subscript 3 space end subscript equals space open parentheses F subscript 1 space plus space F subscript 2 close parentheses space circled times space I
O subscript 3 space equals space open parentheses F subscript 1 space circled times space I close parentheses space plus thin space open parentheses F subscript 2 space end subscript space circled times space I close parentheses
O subscript 3 space end subscript space equals space O subscript 1 space plus thin space O subscript 2



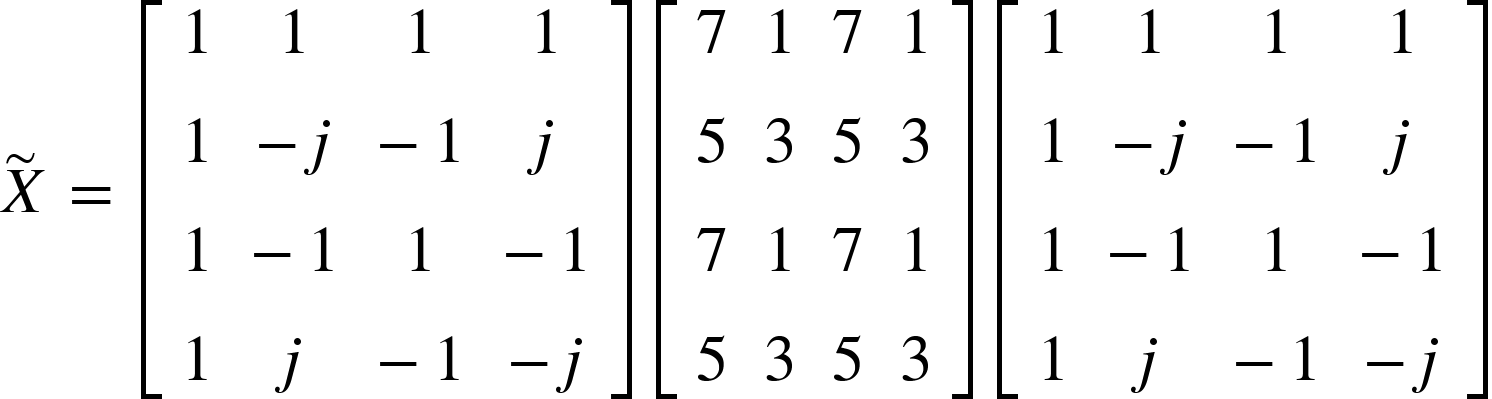
**Q6 Solution:**

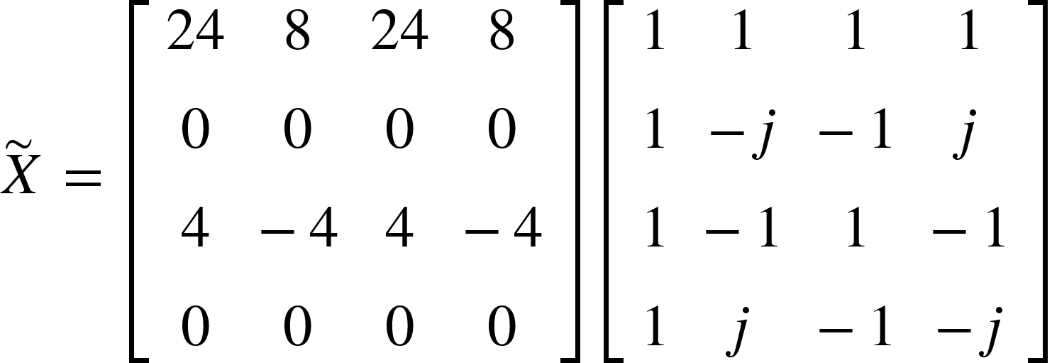
 

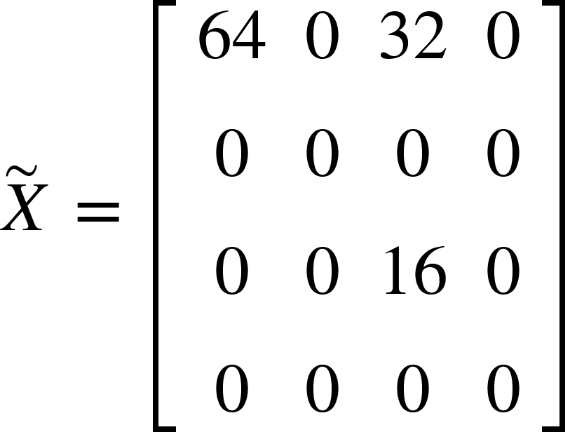
**Part (a):** Compute the 2D-DFT of the 4 x 4 gray-scale image given below:

 We have given our original input image X.

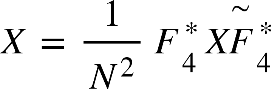
Computing 2D-DFT of the original image X with tilde on top space equals space F subscript 4 X F subscript 4 for a 4 x 4 image.







**Part (b):** The inverse 2D-DFT of the given three 4 x 4 images is computed by

. Here, N = 4.

For Pattern 1,

X subscript 1 space equals space 1 over 16 space open square brackets table row 1 1 1 1 row 1 j cell negative 1 end cell cell negative j end cell row 1 cell negative 1 end cell 1 cell negative 1 end cell row 1 cell negative j end cell cell negative 1 end cell j end table close square brackets space open square brackets table row 64 0 0 0 row 0 0 0 0 row 0 0 0 0 row 0 0 0 0 end table close square brackets space open square brackets table row 1 1 1 1 row 1 j cell negative 1 end cell cell negative j end cell row 1 cell negative 1 end cell 1 cell negative 1 end cell row 1 cell negative j end cell cell negative 1 end cell j end table close square brackets

X subscript 1 space equals space 1 over 16 space open square brackets table row 64 0 0 0 row 64 0 0 0 row 64 0 0 0 row 64 0 0 0 end table close square brackets space open square brackets table row 1 1 1 1 row 1 j cell negative 1 end cell cell negative j end cell row 1 cell negative 1 end cell 1 cell negative 1 end cell row 1 cell negative j end cell cell negative 1 end cell j end table close square brackets

X subscript 1 space equals space open square brackets table row 4 4 4 4 row 4 4 4 4 row 4 4 4 4 row 4 4 4 4 end table close square brackets

For Pattern 2,

X subscript 2 space equals space 1 over 16 space open square brackets table row 1 1 1 1 row 1 j cell negative 1 end cell cell negative j end cell row 1 cell negative 1 end cell 1 cell negative 1 end cell row 1 cell negative j end cell cell negative 1 end cell j end table close square brackets space open square brackets table row 0 0 32 0 row 0 0 0 0 row 0 0 0 0 row 0 0 0 0 end table close square brackets space open square brackets table row 1 1 1 1 row 1 j cell negative 1 end cell cell negative j end cell row 1 cell negative 1 end cell 1 cell negative 1 end cell row 1 cell negative j end cell cell negative 1 end cell j end table close square brackets

X subscript 2 space equals space 1 over 16 space open square brackets table row 0 0 32 0 row 0 0 32 0 row 0 0 32 0 row 0 0 32 0 end table close square brackets space open square brackets table row 1 1 1 1 row 1 j cell negative 1 end cell cell negative j end cell row 1 cell negative 1 end cell 1 cell negative 1 end cell row 1 cell negative j end cell cell negative 1 end cell j end table close square brackets

X subscript 2 space equals space open square brackets table row 2 cell negative 2 end cell 2 cell negative 2 end cell row 2 cell negative 2 end cell 2 cell negative 2 end cell row 2 cell negative 2 end cell 2 cell negative 2 end cell row 2 cell negative 2 end cell 2 cell negative 2 end cell end table close square brackets

For Pattern 3,

X subscript 3 space equals space 1 over 16 space open square brackets table row 1 1 1 1 row 1 j cell negative 1 end cell cell negative j end cell row 1 cell negative 1 end cell 1 cell negative 1 end cell row 1 cell negative j end cell cell negative 1 end cell j end table close square brackets space open square brackets table row 0 0 0 0 row 0 0 0 0 row 0 0 16 0 row 0 0 0 0 end table close square brackets space open square brackets table row 1 1 1 1 row 1 j cell negative 1 end cell cell negative j end cell row 1 cell negative 1 end cell 1 cell negative 1 end cell row 1 cell negative j end cell cell negative 1 end cell j end table close square brackets

X subscript 3 space equals space 1 over 16 space open square brackets table row 0 0 cell negative 16 end cell 0 row 0 0 cell negative 16 end cell 0 row 0 0 cell negative 16 end cell 0 row 0 0 cell negative 16 end cell 0 end table close square brackets space open square brackets table row 1 1 1 1 row 1 j cell negative 1 end cell cell negative j end cell row 1 cell negative 1 end cell 1 cell negative 1 end cell row 1 cell negative j end cell cell negative 1 end cell j end table close square brackets

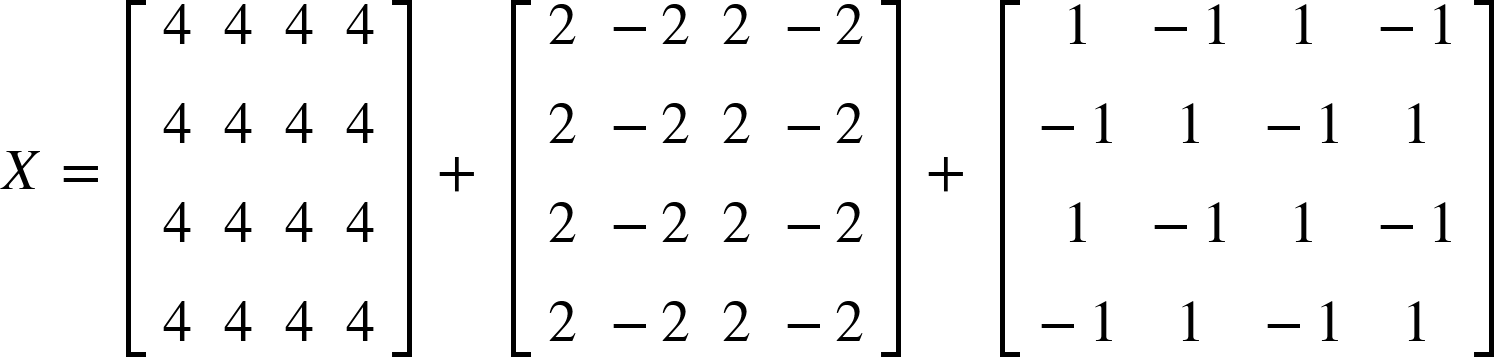
X subscript 3 space equals space open square brackets table row 1 cell negative 1 end cell 1 cell negative 1 end cell row cell negative 1 end cell 1 cell negative 1 end cell 1 row 1 cell negative 1 end cell 1 cell negative 1 end cell row cell negative 1 end cell 1 cell negative 1 end cell 1 end table close square brackets

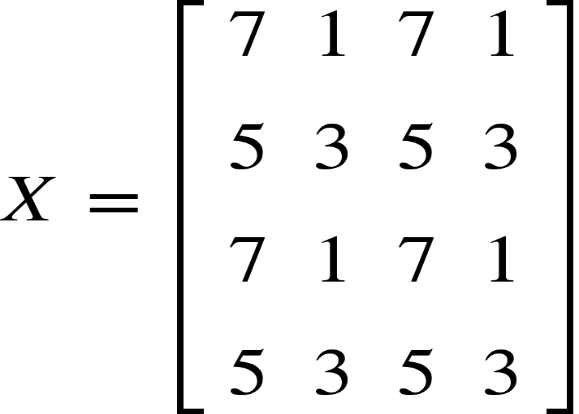
**Part (c):**

Now, we will add the results which we obtained in part (b).

That is, X = X1 + X2 + X3.

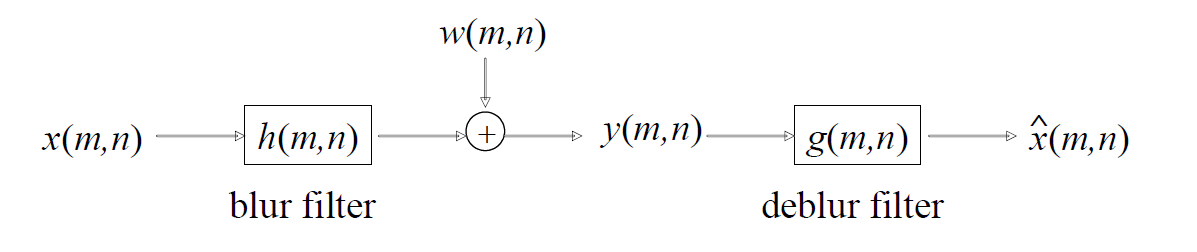
So,



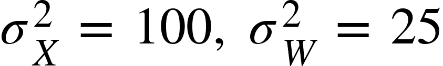


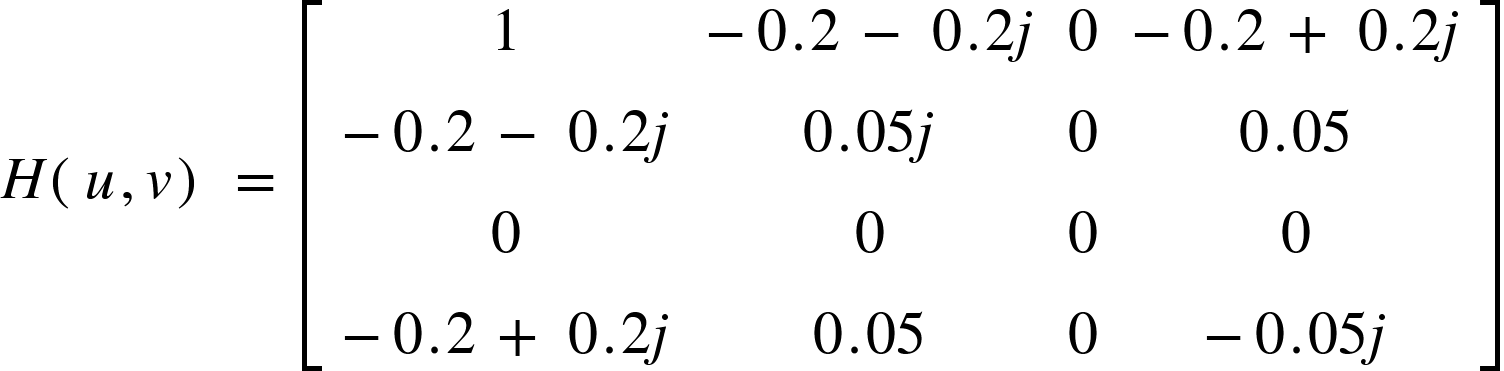
**Observation:** Here, we can clearly see that adding all the three images gives the original input image. That is the distribution law of inverse 2D-DFT which implies that computing them independently and adding the consecutive results produces the initial input image. 2D-DFT is inversible and also has the preserving property.

**Q7 Solution:**

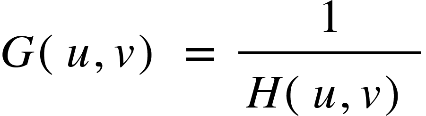


We have given that:





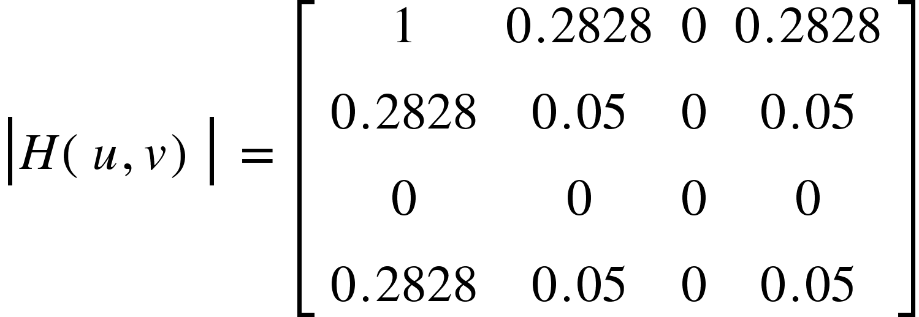
**Part (a):** Design deblur filter *G(u,v)* using inverse filtering approach.

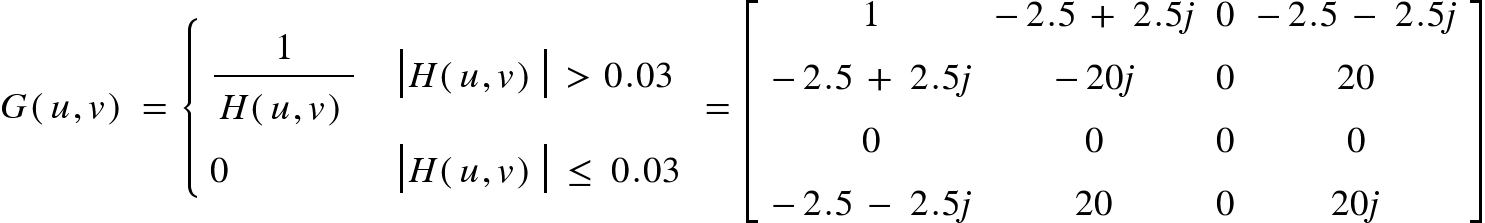


G open parentheses u comma v close parentheses space equals space fraction numerator 1 over denominator open square brackets table row 1 cell negative 0.2 space minus space 0.2 j end cell 0 cell negative 0.2 space plus space 0.2 j end cell row cell negative 0.2 space minus space 0.2 j end cell cell 0.05 j end cell 0 cell 0.05 end cell row 0 0 0 0 row cell negative 0.2 space plus space 0.2 j end cell cell 0.05 end cell 0 cell negative 0.05 j end cell end table close square brackets end fraction

G open parentheses u comma v close parentheses space equals space open square brackets table row 1 cell negative 2.5 space plus space 2.5 j end cell cell i n f end cell cell negative 2.5 space minus space 2.5 j end cell row cell negative 2.5 space plus space 2.5 j end cell cell negative 20 j end cell cell i n f end cell 20 row cell i n f end cell cell i n f end cell cell i n f end cell cell i n f end cell row cell negative 2.5 space minus space 2.5 j end cell 20 cell i n f end cell cell 20 j end cell end table close square brackets

**Part (b):** Design deblur filter *G(u,v)* using pseudo inverse filtering approach with delta space equals space 0.03.



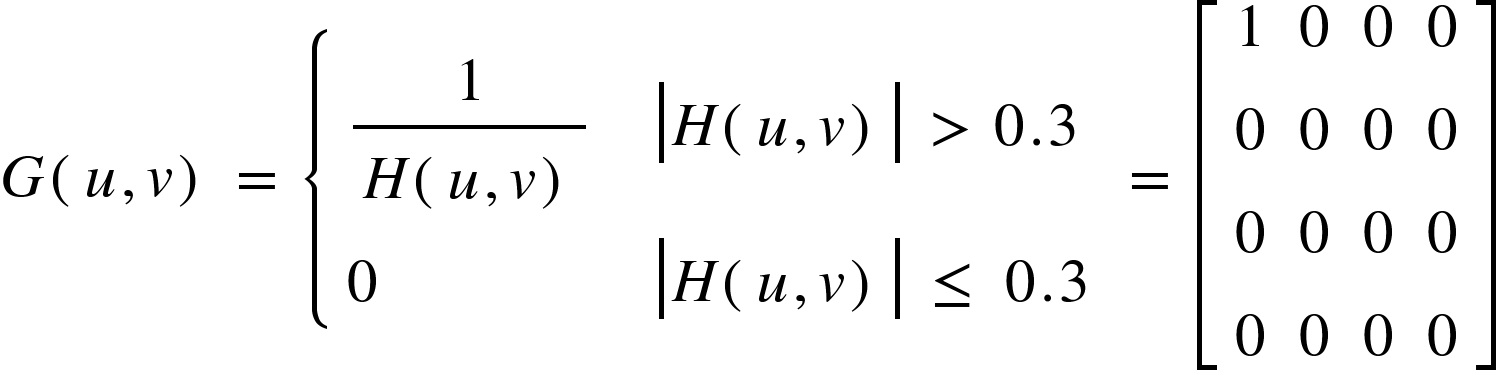


**Part (c):** Design deblur filter *G(u,v)* using pseudo inverse filtering approach with delta space equals space 0.1. Using |*H(u,v)*| from part (b).

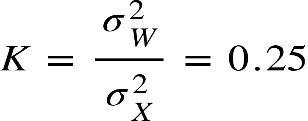
G open parentheses u comma v close parentheses space equals space open curly brackets table attributes columnalign left columnspacing 1.4ex end attributes row cell fraction numerator 1 over denominator H open parentheses u comma v close parentheses end fraction end cell cell open vertical bar H open parentheses u comma v close parentheses close vertical bar space greater than space 0.1 end cell row 0 cell open vertical bar H open parentheses u comma v close parentheses close vertical bar space less or equal than space 0.1 end cell end table close

G open parentheses u comma v close parentheses space equals space open square brackets table row 1 cell negative 2.5 space plus space 2.5 j end cell 0 cell negative 2.5 space minus space 2.5 j end cell row cell negative 2.5 space plus space 2.5 j end cell 0 0 0 row 0 0 0 0 row cell negative 2.5 space minus space 2.5 j end cell 0 0 0 end table close square brackets

**Part (d):** Design deblur filter *G(u,v)* using pseudo inverse filtering approach withdelta space equals space 0.3. Using |*H(u,v*| from part (b).



**Part (e):** Design deblur filter *G(u,v)* using Wiener Filtering approach.

We already know that sigma subscript X superscript 2 space equals space 100 comma space sigma subscript W superscript 2 space equals space 100. Hence, 

G open parentheses u comma v close parentheses space equals space fraction numerator H asterisk times open parentheses u comma v close parentheses over denominator open vertical bar H open parentheses u comma v close parentheses close vertical bar squared space plus space K end fraction

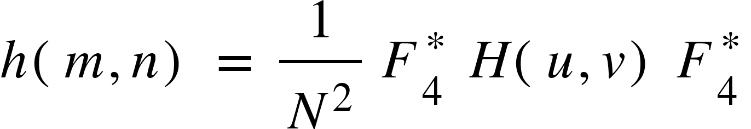
G open parentheses u comma v close parentheses space equals space fraction numerator open square brackets table row 1 cell negative 0.2 space plus space 0.2 j end cell 0 cell negative 0.2 space minus space 0.2 j end cell row cell negative 0.2 space plus space 0.2 j end cell cell negative 0.05 j end cell 0 cell 0.05 end cell row 0 0 0 0 row cell negative 0.2 space minus space 0.2 j end cell cell 0.05 end cell 0 cell 0.05 j end cell end table close square brackets over denominator open parentheses open square brackets table row 1 cell 0.2828 end cell 0 cell 0.2828 end cell row cell 0.2828 end cell cell 0.05 end cell 0 cell 0.05 end cell row 0 0 0 0 row cell 0.2828 end cell cell 0.05 end cell 0 cell 0.05 end cell end table close square brackets close parentheses squared space plus space 0.25 end fraction

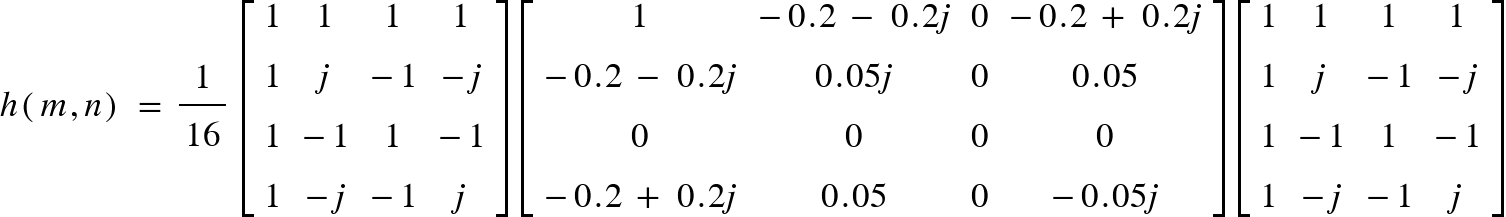


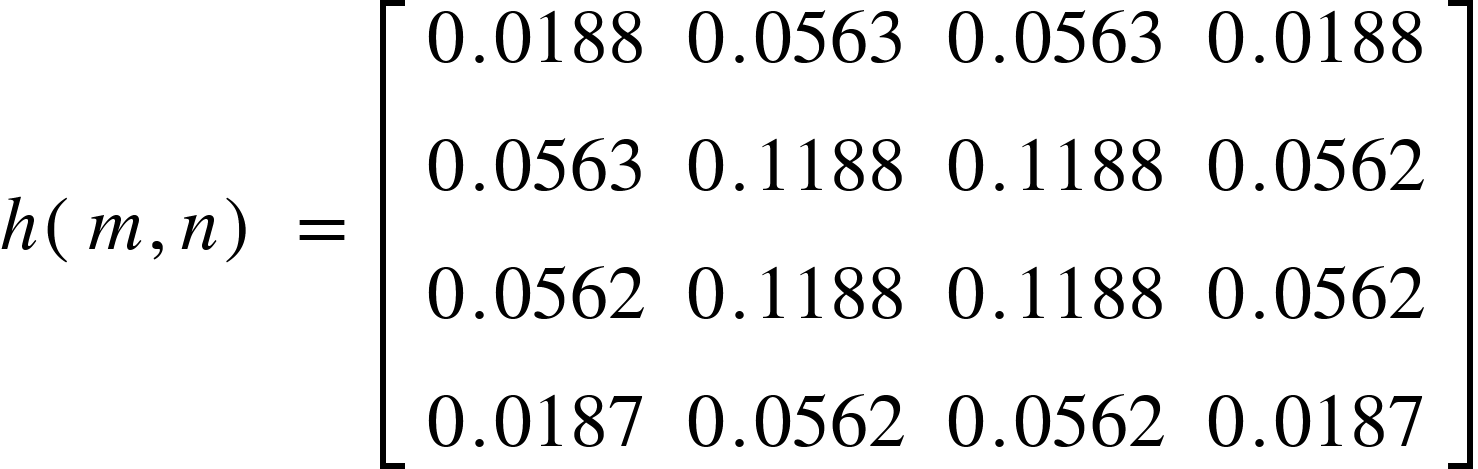
G open parentheses u comma v close parentheses space equals space fraction numerator open square brackets table row 1 cell negative 0.2 space plus space 0.2 j end cell 0 cell negative 0.2 space minus space 0.2 j end cell row cell negative 0.2 space plus space 0.2 j end cell cell negative 0.05 j end cell 0 cell 0.05 end cell row 0 0 0 0 row cell negative 0.2 space minus space 0.2 j end cell cell 0.05 end cell 0 cell 0.05 j end cell end table close square brackets over denominator open square brackets table row cell 1.25 end cell cell 0.33 end cell cell 0.25 end cell cell 0.33 end cell row cell 0.33 end cell cell 0.2525 end cell cell 0.25 end cell cell 0.2525 end cell row cell 0.25 end cell cell 0.25 end cell cell 0.25 end cell cell 0.25 end cell row cell 0.33 end cell cell 0.2525 end cell cell 0.25 end cell cell 0.2525 end cell end table close square brackets end fraction

G open parentheses u comma v close parentheses space equals space open square brackets table row cell 0.8 end cell cell negative 0.61 space plus space 0.61 j end cell 0 cell negative 0.61 space minus space 0.61 j end cell row cell negative 0.61 space plus space 0.61 j end cell cell negative 0.2 j end cell 0 cell 0.2 end cell row 0 0 0 0 row cell negative 0.61 space minus space 0.61 j end cell cell 0.2 end cell 0 cell 0.2 j end cell end table close square brackets

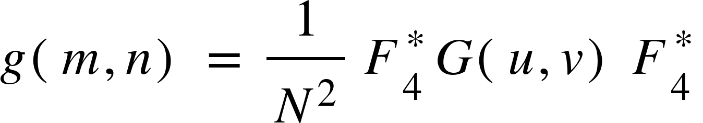
**Part (f):** To compute *h(m,n),* we find inverse 2D-DFT of *H(u,v)*.

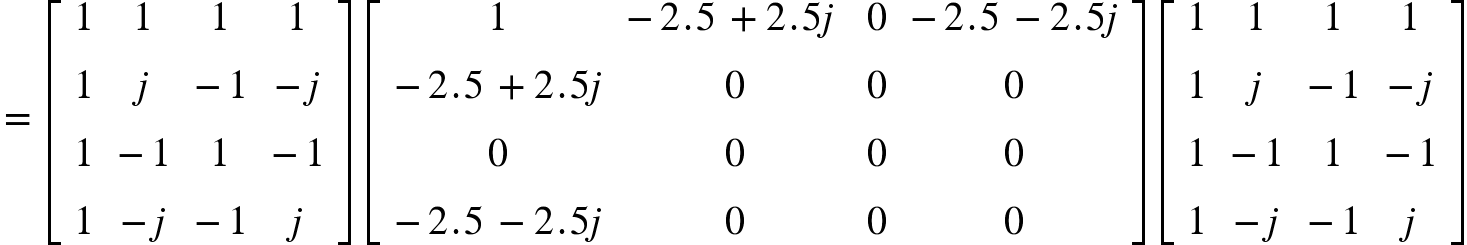


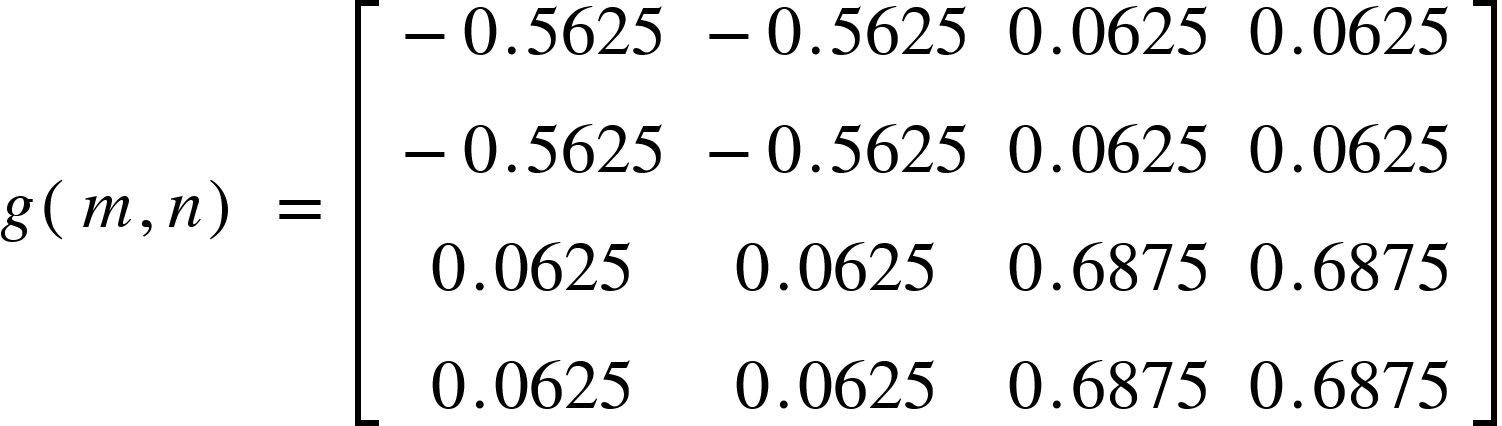




**Part (g):** To compute *g(m,n),* we find inverse 2D-DFT of *G(u,v)* from part (c).

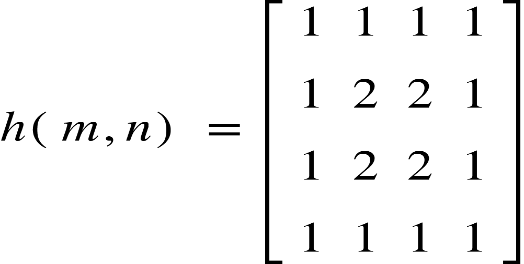






**Q8 Solution:**

We have given that, sigma subscript X superscript 2 space equals space 400 comma space sigma subscript W superscript 2 space equals space 100 space and



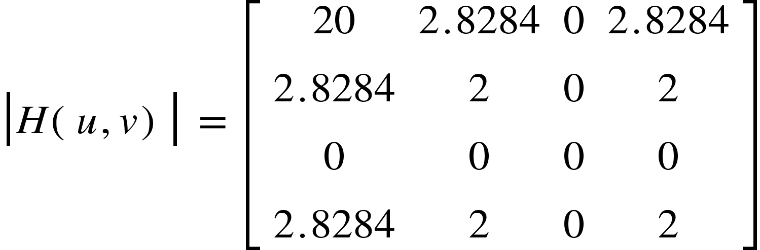
**Part (a):**

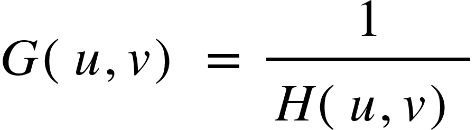
*H(u,v)* , the blur filter in the frequency (2D-DFT) domain.

H open parentheses u comma v close parentheses space equals space F subscript 4 space h open parentheses m comma n close parentheses space F subscript 4
H open parentheses u comma v close parentheses space equals space open square brackets table row 1 1 1 1 row 1 cell negative j end cell cell negative 1 end cell j row 1 cell negative 1 end cell 1 cell negative 1 end cell row 1 j cell negative 1 end cell cell negative j end cell end table close square brackets space open square brackets table row 1 1 1 1 row 1 2 2 1 row 1 2 2 1 row 1 1 1 1 end table close square brackets space open square brackets table row 1 1 1 1 row 1 cell negative j end cell cell negative 1 end cell j row 1 cell negative 1 end cell 1 cell negative 1 end cell row 1 j cell negative 1 end cell cell negative j end cell end table close square brackets

H open parentheses u comma v close parentheses space equals space open square brackets table row 20 cell negative 2 space minus space 2 j end cell 0 cell negative 2 space plus space 2 j end cell row cell negative 2 space minus space 2 j end cell cell 2 j end cell 0 2 row 0 0 0 0 row cell negative 2 space plus space 2 j end cell 2 0 cell negative 2 j end cell end table close square brackets

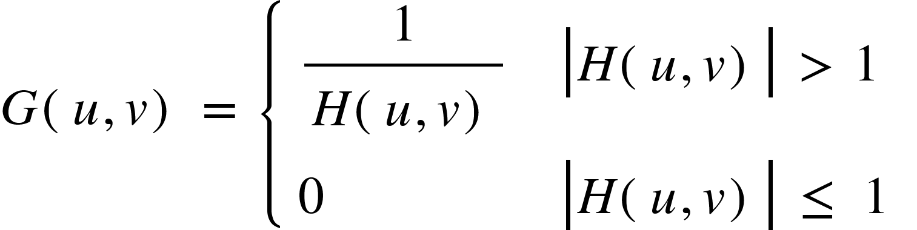

**Part (b):** Design a deblur filter *G(u,v)* using the pseudo-inverse filtering approach with delta space equals space 1.

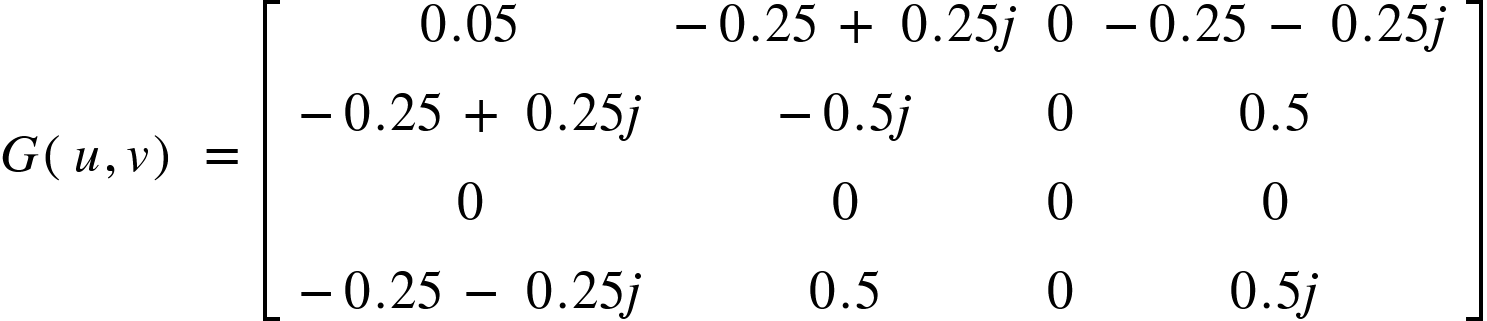




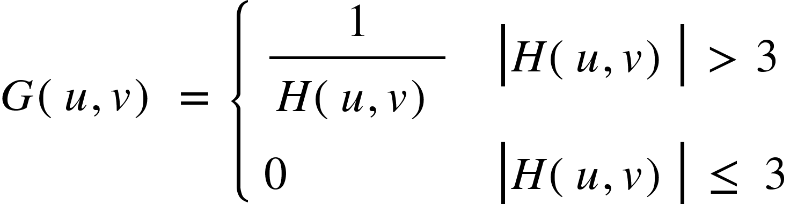
G open parentheses u comma v close parentheses space equals space fraction numerator 1 over denominator open square brackets table row 20 cell negative 2 space minus space 2 j end cell 0 cell negative 2 space plus space 2 j end cell row cell negative 2 space minus space 2 j end cell cell 2 j end cell 0 2 row 0 0 0 0 row cell negative 2 space plus space 2 j end cell 2 0 cell negative 2 j end cell end table close square brackets end fraction

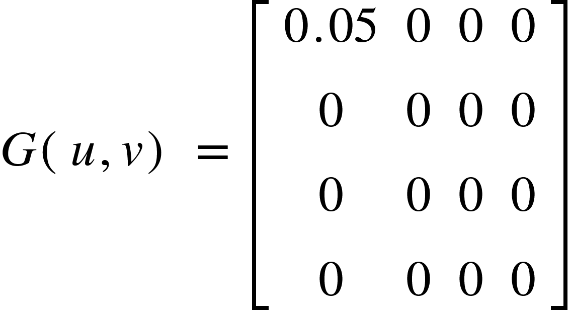
G open parentheses u comma v close parentheses space equals space open square brackets table row cell 0.05 end cell cell negative 0.25 space plus space 0.25 j end cell cell i n f end cell cell negative 0.25 space minus space 0.25 j end cell row cell negative 0.25 space plus space 0.25 j end cell cell negative 0.5 j end cell cell i n f end cell cell 0.5 end cell row cell i n f end cell cell i n f end cell cell i n f end cell cell i n f end cell row cell negative 0.25 space minus space 0.25 j end cell cell 0.5 end cell cell i n f end cell cell 0.5 j end cell end table close square brackets



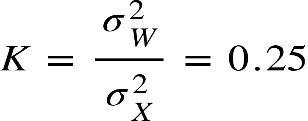


**Part (c):** Design a deblur filter *G(u,v)* using the pseudo-inverse filtering approach with delta space equals space 3.





**Part (d):** Design deblur filter *G(u,v)* using Wiener Filtering approach.

We already know that sigma subscript X superscript 2 space equals space 100 comma space sigma subscript W superscript 2 space equals space 100. Hence, 

G open parentheses u comma v close parentheses space equals space fraction numerator H asterisk times open parentheses u comma v close parentheses over denominator open vertical bar H open parentheses u comma v close parentheses close vertical bar squared space plus space K end fraction

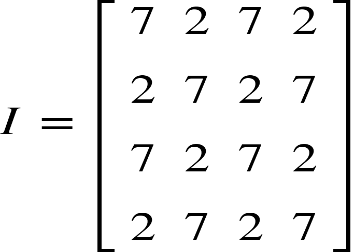
G open parentheses u comma v close parentheses space equals space fraction numerator open square brackets table row 20 cell negative 2 space plus space 2 j end cell 0 cell negative 2 space minus space 2 j end cell row cell negative 2 space plus space 2 j end cell cell negative 2 j end cell 0 2 row 0 0 0 0 row cell negative 2 space minus space 2 j end cell 2 0 cell 2 j end cell end table close square brackets over denominator open parentheses open square brackets table row 20 cell 2.8284 end cell 0 cell 2.8284 end cell row cell 2.8284 end cell 2 0 2 row 0 0 0 0 row cell 2.8284 end cell 2 0 2 end table close square brackets close parentheses squared space plus space 0.25 end fraction



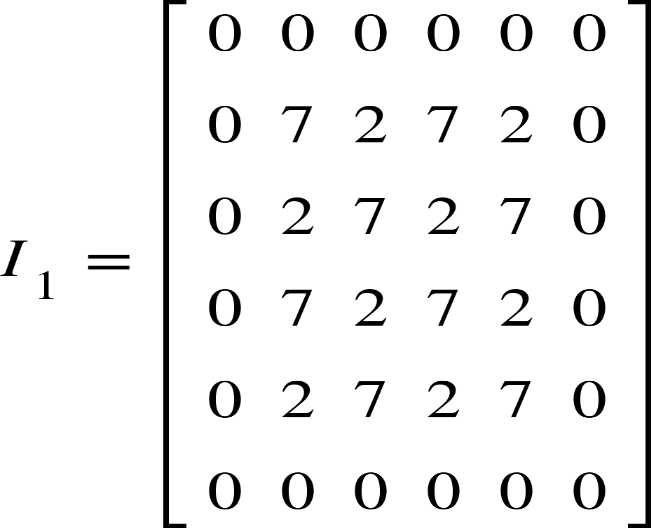
G open parentheses u comma v close parentheses space equals space fraction numerator open square brackets table row 20 cell negative 2 space plus space 2 j end cell 0 cell negative 2 space minus space 2 j end cell row cell negative 2 space plus space 2 j end cell cell negative 2 j end cell 0 2 row 0 0 0 0 row cell negative 2 space minus space 2 j end cell 2 0 cell 2 j end cell end table close square brackets over denominator open square brackets table row cell 400.25 end cell cell 8.25 end cell cell 0.25 end cell cell 8.25 end cell row cell 8.25 end cell cell 4.25 end cell cell 0.25 end cell cell 4.25 end cell row cell 0.25 end cell cell 0.25 end cell cell 0.25 end cell cell 0.25 end cell row cell 8.25 end cell cell 4.25 end cell cell 0.25 end cell cell 4.25 end cell end table close square brackets end fraction

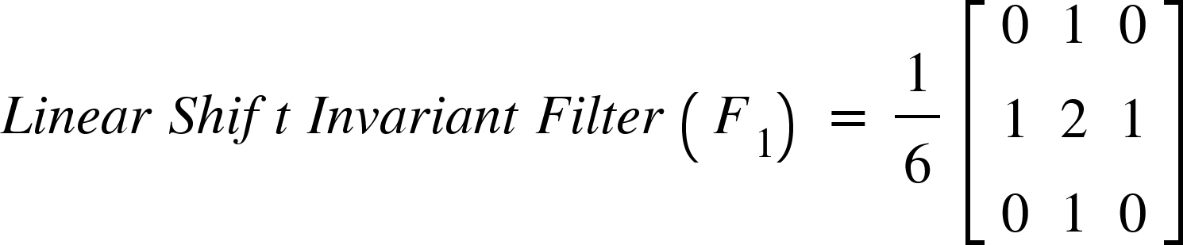
G open parentheses u comma v close parentheses space equals space open square brackets table row cell 0.05 end cell cell negative 0.24 space plus space 0.24 j end cell 0 cell negative 0.24 space minus space 0.24 j end cell row cell negative 0.24 space plus space 0.24 j end cell cell negative 0.47 j end cell 0 cell 0.47 end cell row 0 0 0 0 row cell negative 0.24 space minus space 0.24 j end cell cell 0.47 end cell 0 cell 0.47 j end cell end table close square brackets

**Q9 Solution:**

We have given that, 

**Part (a):** Zero padding basically means that we will add 0’s to the boundary of the image so that during sliding the filter on the image no pixel in the image remains unencountered. By using zero padding we can easily be able to retain the original size of the input image after the filter operation.

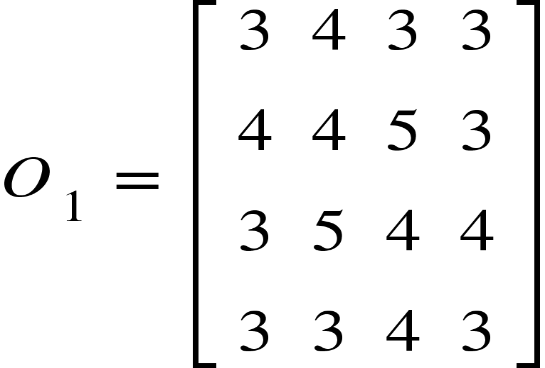




In order to calculate the filtered image we will slide the filter *F1* on the zero padded image with stride 1 and take the inner product. The output pixel values are the sum of all the elements of the inner product.

*O(m,n) =*

Hence, the filtered output image is,



Let the input image *I1* be represented by *p* and the output filtered image *O1* be represented by *q*.

**Mean Absolute Error (MAE):**

M A E space equals space sum from i equals 1 to N of fraction numerator open vertical bar q subscript i space minus space p subscript i close vertical bar over denominator N end fraction

M A E space equals space fraction numerator 4 space plus 2 space plus 4 space plus 1 plus 2 space plus 3 space plus 3 space plus space 4 space plus space 4 space plus 3 space plus 3 space plus 2 space plus 2 space plus 1 plus 4 space plus 2 space plus 4 space over denominator 16 end fraction

M A E space equals space 46 over 16

M A E space equals space 2.875

**Mean Squared Error (MSE):**

M S E equals space sum from i space equals space 1 to N of space open parentheses q subscript i space minus space p subscript i close parentheses squared over N

M S E space equals space fraction numerator 16 space plus space 4 space plus space 16 space plus space 1 space plus space 4 space plus space 9 space plus space 9 space plus space 16 space plus space 16 space space plus space 9 space space plus space 9 space plus space 4 space plus space 1 space plus 16 space plus space 4 space plus space 16 over denominator 16 end fraction

M S E space space equals space 150 over 16

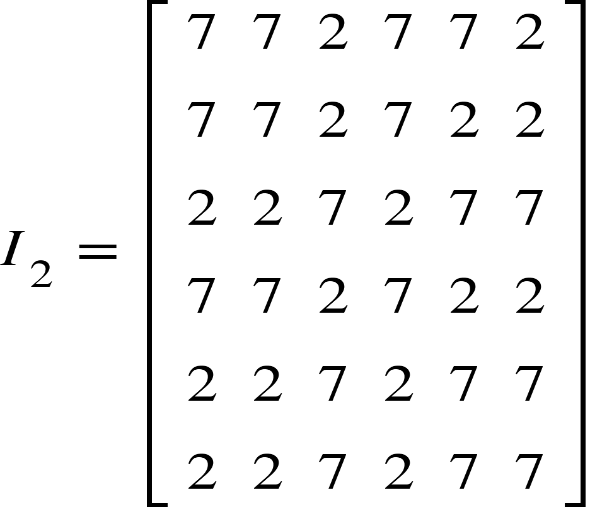
M S E space equals space 9.375

**Peak Signal-to-noise Ratio (PSNR):**

Here, we are taking 4 bits/pixel. So, the *MAXI* is the maximum possible intensity which is given as 2B – 1, where B is the number of bits. So for our given example *MAXI* = (24 – 1) = 15.

P S N R thin space equals space 10 log open parentheses fraction numerator M A X subscript I superscript 2 over denominator M S E end fraction close parentheses
P S N R space equals space 10 log open parentheses fraction numerator 225 over denominator 9.375 end fraction close parentheses
P S N R space equals space 13.8021 space d B

**Part (b):** In this case we will pass the original input image through a median filter with replicate padding,

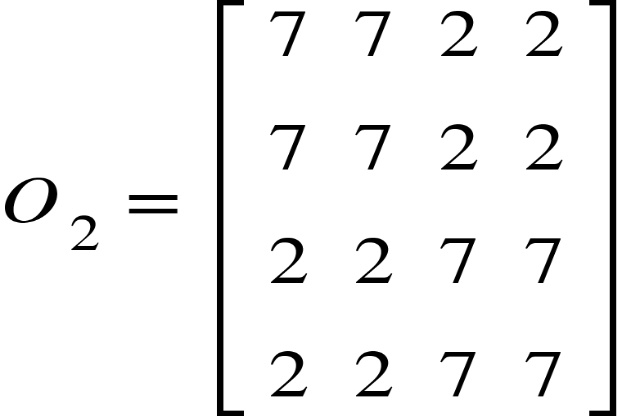


To calculate the filtered image using a 3 x 3 median filter with the replicate padding input image we take a 3 x 3section of the input image and pass the median value from all the intensities in the 3 x 3 section. For example,

O subscript 2 space end subscript open parentheses 0 comma 0 close parentheses space equals space m e d i a n open parentheses open square brackets table row 7 7 2 row 7 7 2 row 2 2 7 end table close square brackets close parentheses space

O subscript 2 space end subscript open parentheses 0 comma 0 close parentheses space equals space m e d i a n open parentheses 2 comma 2 comma 2 comma 2 comma 7 comma 7 comma 7 comma 7 comma 7 close parentheses space equals space 7

Hence, the filtered output image is given as:



Let the input image *I2* be represented by p and the output filtered image *O2* be represented by q.

**Mean Absolute Error (MAE):**

M A E space equals space sum from i equals 1 to N of fraction numerator open vertical bar q subscript i space minus space p subscript i close vertical bar over denominator N end fraction

M A E space equals space fraction numerator 5 space plus space 5 space plus space 5 space plus space 5 space space plus space 5 space plus space 5 space plus space 5 space plus space 5 over denominator 16 end fraction

M A E space equals space 40 over 16

M A E space equals space 2.5

**Mean Squared Error (MSE):**

M S E equals space sum from i space equals space 1 to N of space open parentheses q subscript i space minus space p subscript i close parentheses squared over N

M S E space equals space fraction numerator 25 space plus space 25 space plus space 25 space plus space 25 space plus 25 space plus 25 space plus 25 space plus 25 over denominator 16 end fraction

M S E space space equals space 200 over 16

M S E space equals space 12.5

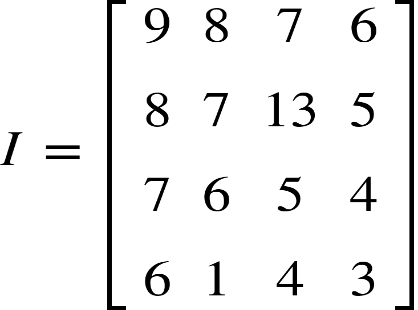
**Peak Signal-to-noise Ratio (PSNR):**

Here, we are taking 4 bits/pixel. So, the MAXI is the maximum possible intensity which is given as 2B – 1, where B is the number of bits. So for our given example MAXI = (24 – 1) = 15.

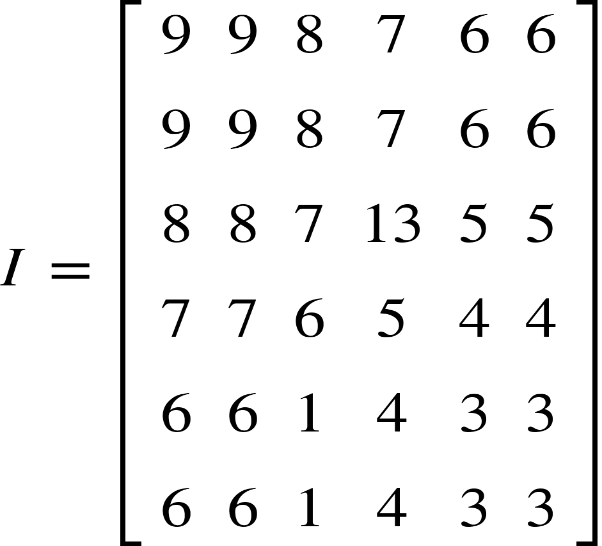
P S N R thin space equals space 10 log open parentheses fraction numerator M A X subscript I superscript 2 over denominator M S E end fraction close parentheses
P S N R space equals space 10 log open parentheses fraction numerator 225 over denominator 12.5 end fraction close parentheses
P S N R space equals space 12.5527 space d B

**Q10 Solution:**

We have given that,



**Part (a):** First, we will apply replicate padding on the input image given above.

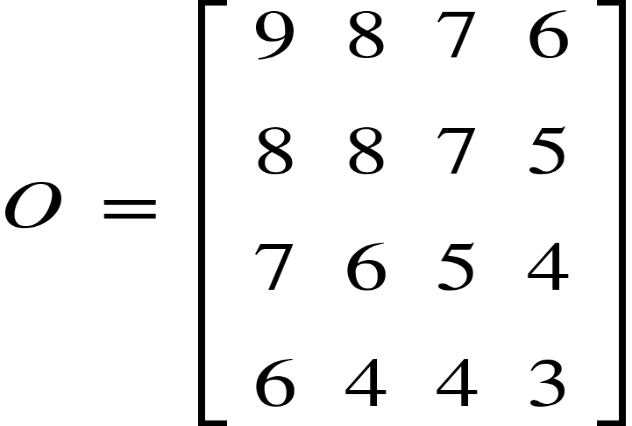


Now, we will apply median filter which is given in the problem on the above replicate padded image we will get the following image,

O open parentheses 0 comma 0 close parentheses space equals space m e d i a n open parentheses open square brackets table row blank 9 blank row 9 9 8 row blank 8 blank end table close square brackets close parentheses

O open parentheses 0 comma 0 close parentheses space equals space m e d i a n open parentheses 8 comma 8 comma 9 comma 9 comma 9 close parentheses space equals space 9

Apply the filter by shifting the filter one stride over the whole input image. So, the output median filtered image is given as:

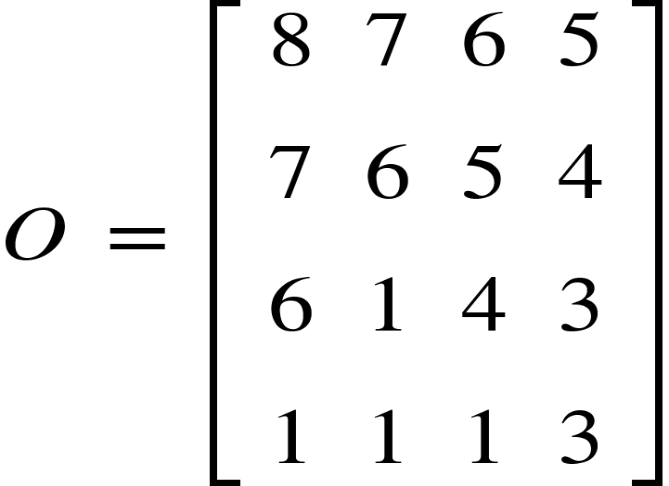


**Part (b):** Here, we will use the minimum filter of the same pattern in part (a) on the replicate padded image *I*. We will get the following resulting image:

O open parentheses 0 comma 0 close parentheses space equals space m i n i m u m open parentheses open square brackets table row blank 9 blank row 9 9 8 row blank 8 blank end table close square brackets close parentheses

O open parentheses 0 comma 0 close parentheses space equals space m i n i m u m open parentheses 8 comma 8 comma 9 comma 9 comma 9 close parentheses space equals space 8

Apply the filter by shifting the filter one stride over the whole input image. So, the output minimum filtered image is given as:

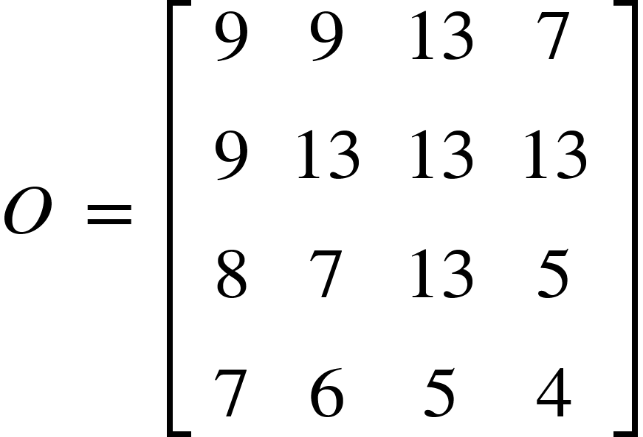


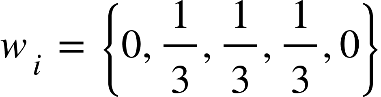
**Part (c):** Here, we will use the maximum filter of the same pattern in part (a) on the replicate padded image *I*. We will get the following resulting image:

O open parentheses 0 comma 0 close parentheses space equals space m a x i m u m open parentheses open square brackets table row blank 9 blank row 9 9 8 row blank 8 blank end table close square brackets close parentheses

O open parentheses 0 comma 0 close parentheses space equals space m a x i m u m open parentheses 8 comma 8 comma 9 comma 9 comma 9 close parentheses space equals space 9

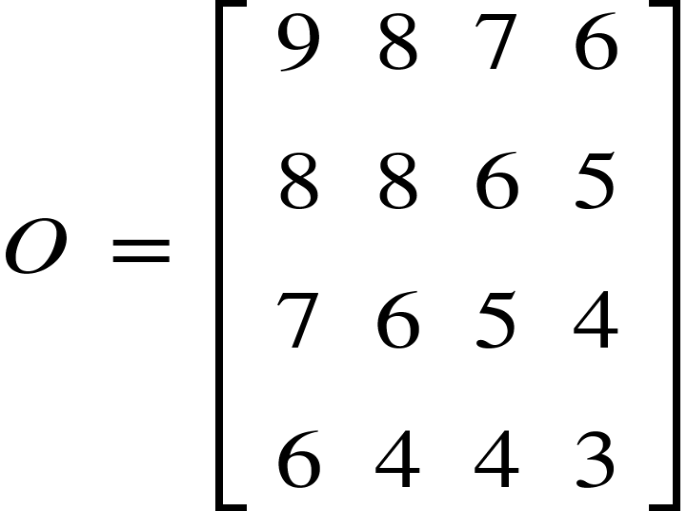
Apply the filter by shifting the filter one stride over the whole input image. So, the output maximum filtered image is given as:



**Part (d):** Here, there is a slight change. We will filter the replicate padded image *I* with the order statistics filter . After applying this filter, we will get the following result:

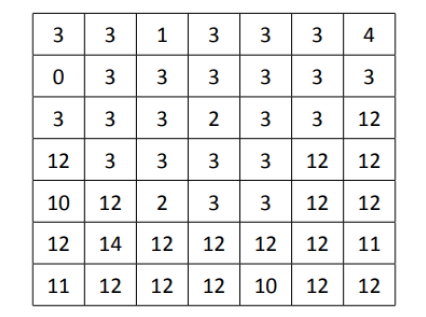
O open parentheses 0 comma 0 close parentheses space equals space r o u n d open parentheses s u m open parentheses open square brackets table row blank 9 blank row 9 9 8 row blank 8 blank end table close square brackets space asterisk times space open curly brackets 0 comma space 1 third comma space 1 third comma space 1 third comma space 0 close curly brackets space close parentheses close parentheses
O open parentheses 0 comma 0 close parentheses space equals space r o u n d open parentheses s u m open parentheses open square brackets 8 comma 8 comma 9 comma 9 comma 9 close square brackets space asterisk times space open square brackets 0 comma 1 third comma 1 third comma 1 third comma 0 close square brackets close parentheses close parentheses
O open parentheses 0 comma 0 close parentheses space equals space 9

Now, we will apply the filter by shifting the filter one stride over the whole input image and we will get the below result:



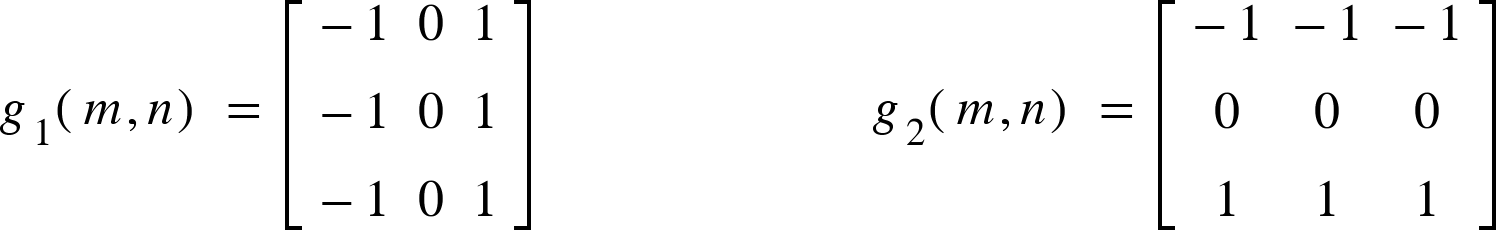
**Q11 Solution:**

Input Image(*I*) is Given as:

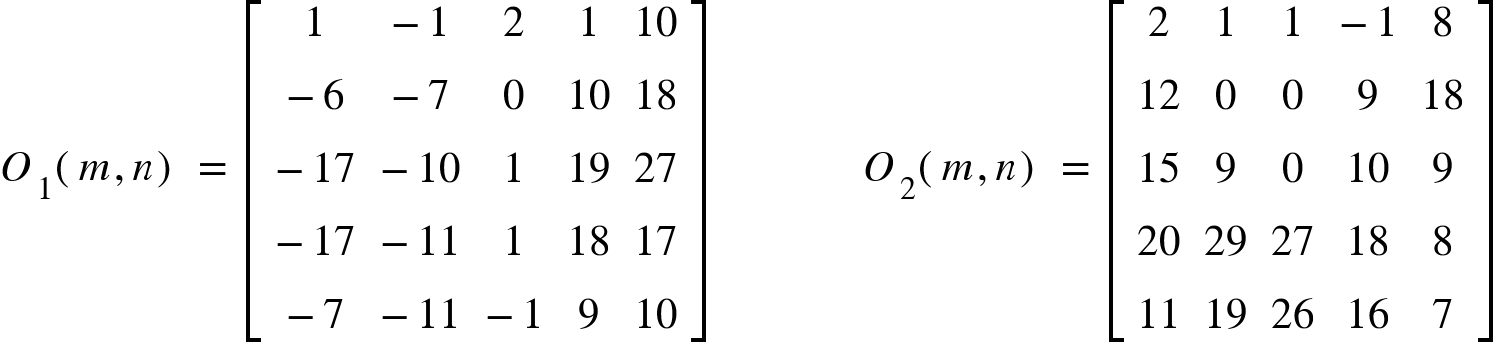


Prewitt gradient operator = |*g(m,n)*| = |*g1(m,n)*| + |*g2(m,n)*|, we have given the value of threshold, T = 22.

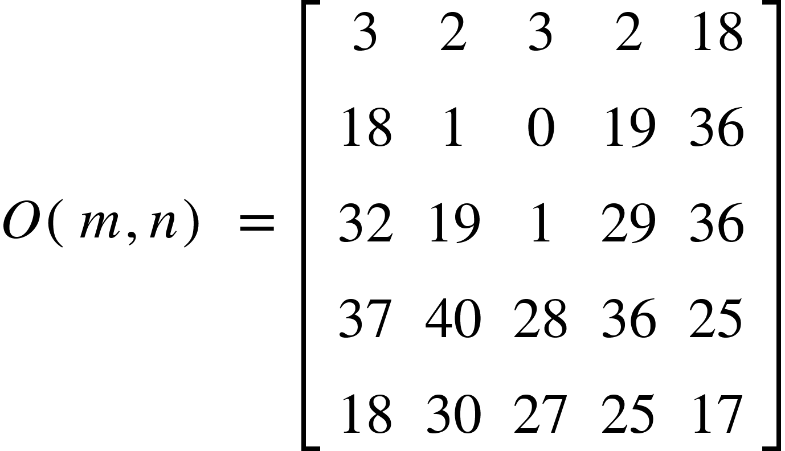
For Prewitt operator,

we

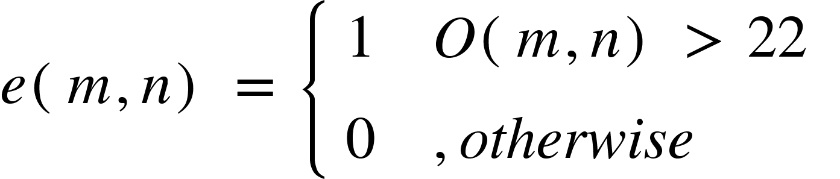
Now, we will apply these filters to the input image(*I*) and we get the following result:



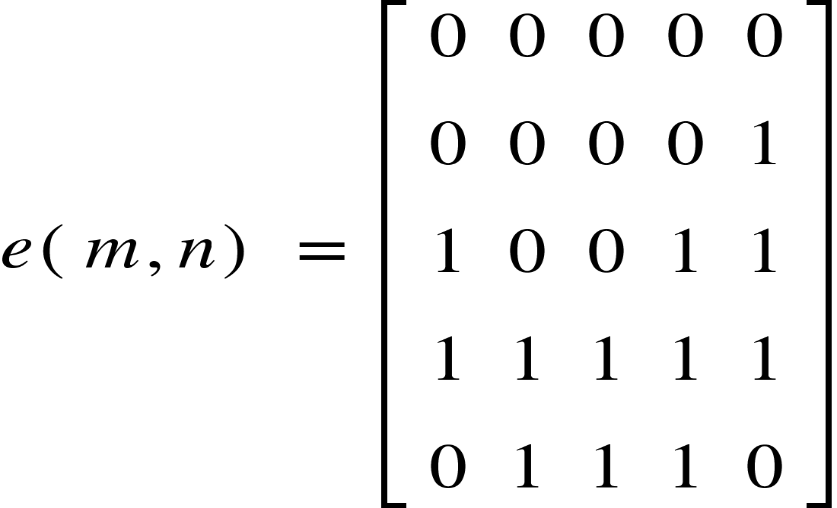
We will approximate the gradient magnitude by adding the modulus of *O1(m,n)* and *O2(m,n)* as shown below:



Applying the threshold, T = 22 on the output image, we get



Hence, the edge map is given as:



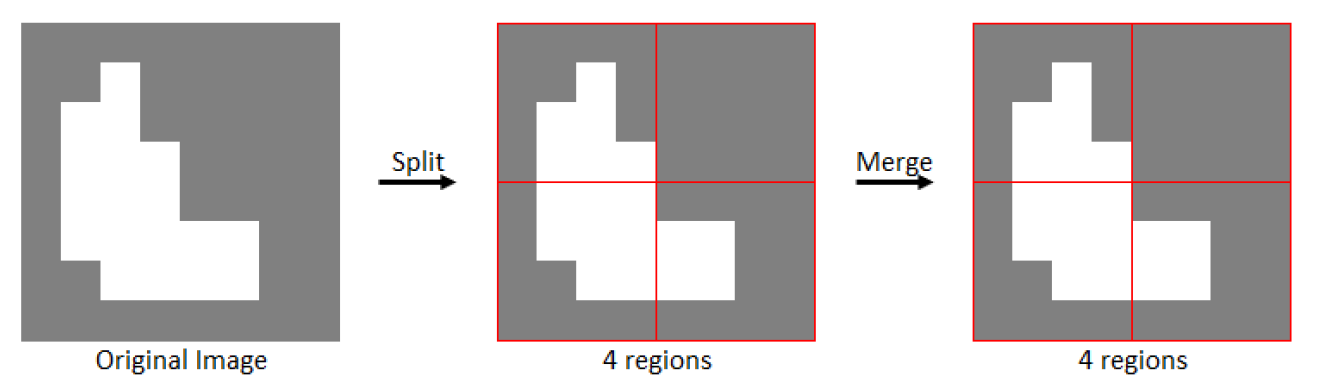
**Q12 Solution:**

Input Image(*I*):



In each iteration, there is a 1 split operation which is followed by a merge operation in the iterative quadtree split-and-merge algorithm. We can perform the split operation only on non-uniform regions. This process is continued till no more non-uniform regions remain.

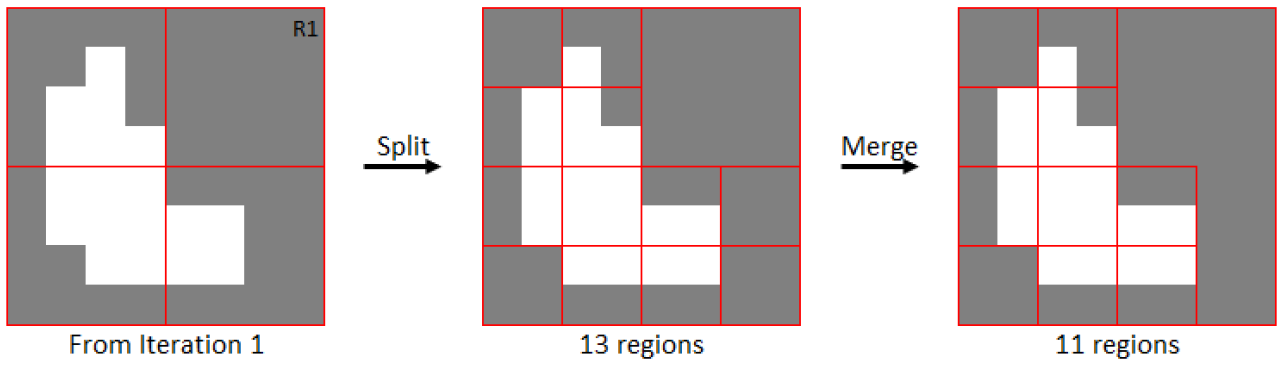
**Iteration 1:**



There are 3 things which we need to consider:

1. The input image is divided into four equal parts.
2. There are no more multiple regions so there is no way to perform merge operation.
3. The output is passed as the input to the next iteration.

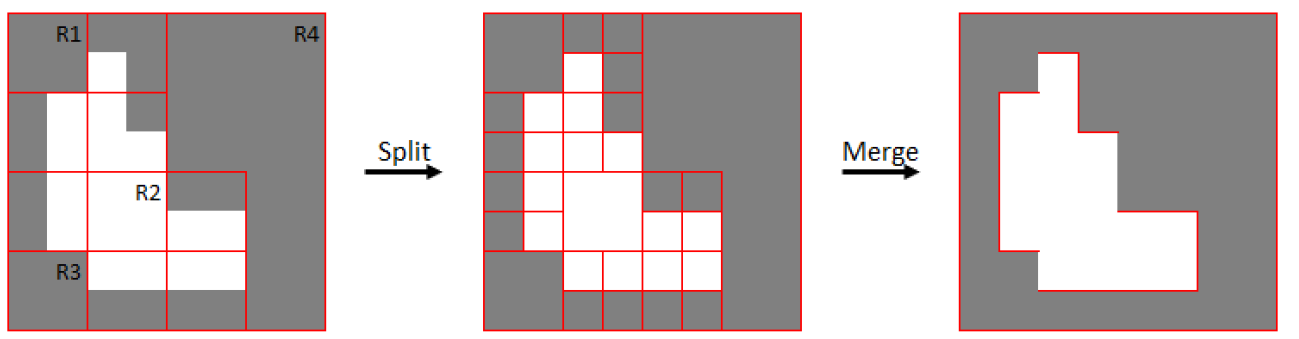
**Iteration 2:**



There are 3 things which we need to consider:

1. Here, R1 is the only region which is uniform and the other regions are divided into 4 equal parts as shown in step 2 (with 13 regions).
2. All the uniform regions are merged together after the split.
3. The output is passed to the input of the next iteration.

**Iteration 3:**



There are 3 things which we need to consider:

1. Here, R1, R2, R3, and R4, are the uniform regions so we will not split them. All the other regions will be split into 4 equal regions as shown in the step 2 of the above figure (with 32 regions).
2. The uniform regions are split and merged together.
3. So, the output of this iteration has correctly segmented the image into 2 parts as shown in step 3 (with 2 regions). This is the final output image.

blank