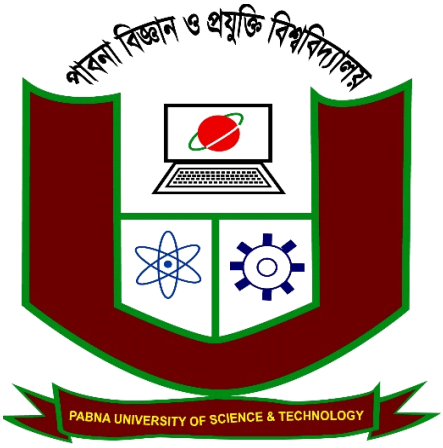
Pabna University of Science and Technology



Faculty of **Engineering and Technology**

Department of **Information and Communication Engineering**

Lab Report

Course Code: ICE-3208

Course Title: Digital Image and Speech Processing Sessional

Submitted by:

Md. Mujahid Hossain
ID: 170621
Session: 2016-17
Department of ICE
Pabna University of
Science & Technology

Submitted to:

Md: Anwar Hossain
Associate Professor
Department of ICE
Pabna University of
Science & Technology

Index

Problem No: 1
Problem Title: Write a MATLAB program for (i) negative (ii) Log (iii) power-law transformation of a gray level image.

(i). Negative transformation of a gray level image:

Negative transformation is the opposite of identity transformation. Here, each value of the input image is subtracted from L-1.
The second linear transformation is negative transformation, which is invert of identity transformation. In negative transformation, each value of the input image is subtracted from the L-1 and mapped onto the output image.

In this case the following transition has been done.

$s = (L – 1) – r$

since the input image of Einstein is an 8 bpp image, so the number of levels in this image are 256. Putting 256 in the equation, we get this

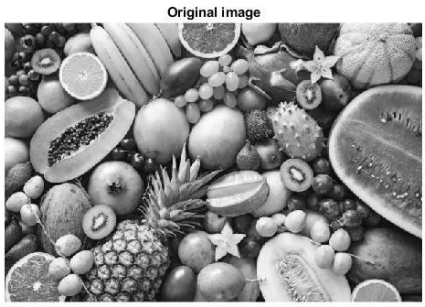
$s = 255 – r$

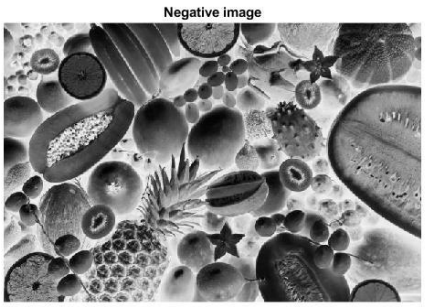
So each value is subtracted by 255 and the result image has been shown above. So what happens is that, the lighter pixels become dark and the darker picture becomes light. And it results in image negative.

Matlab source code:

```
clc;
clear all;
close all;
i1=imread('fruits.jpg');
i2=rgb2gray(i1);
figure(1)
subplot(121);
imshow(i2)
title('Original image')
b=255-i2;
subplot(122);
imshow(b);
title('Negative image')
```

Input and Output:





(ii). Log transformation of a gray level image:

The formula for Logarithmic transformation,

$$s = c \log(r + 1)$$

Where s and r are the pixel values of the output and the input image and c is a constant. The value 1 is added to each of the pixel value of the input image because if there is a pixel intensity of 0 in the image, then log (0) is equal to infinity. So 1 is added, to make the minimum value at least 1.

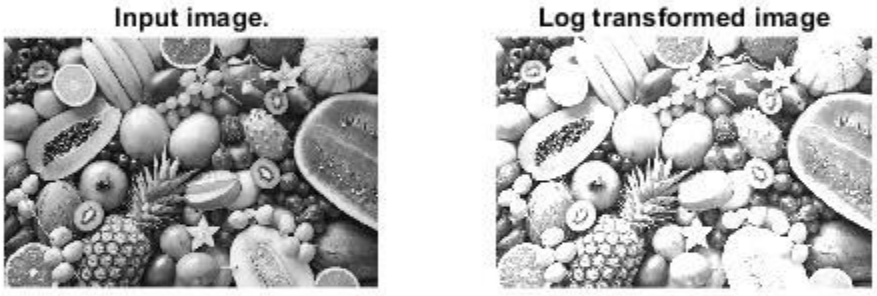
During log transformation, the dark pixels in an image are expanded as compare to the higher pixel values. The higher pixel values are kind of compressed in log transformation. This result in following image enhancement.

The value of c in the log transform adjust the kind of enhancement you are looking for.

Matlab source code:

```
clc;
clear all;
close all;
im=imread('fruits.jpg');
img=rgb2gray(im);
figure(1);
subplot(1,2,1)
imshow(img)
title('Input image.');
img2=im2double(img);
c=input('Input constant c:');
trans_img = c*log(1+img2);
subplot(1,2,2)
imshow(trans_img)
title('Log transformed image');
```

Input and Output:
Input constant c: 2



(iii). Power-law transformation of a gray level image:

There are further two transformation is power law transformations, that include nth power and nth root transformation. These transformations can be given by the expression:

$s=cr^{\gamma}$

This symbol γ is called gamma, due to which this transformation is also known as gamma transformation.

Variation in the value of γ varies the enhancement of the images. Different display devices / monitors have their own gamma correction, that's why they display their image at different intensity.

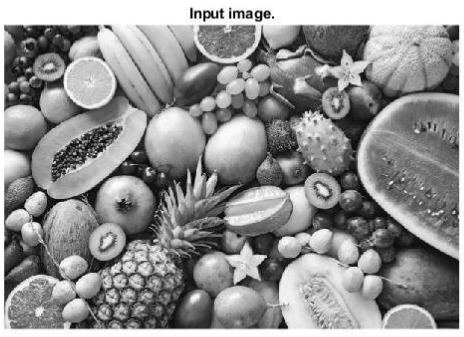
This type of transformation is used for enhancing images for different type of display devices. The gamma of different display devices is different. For example Gamma of CRT lies in between of 1.8 to 2.5, that means the image displayed on CRT is dark.

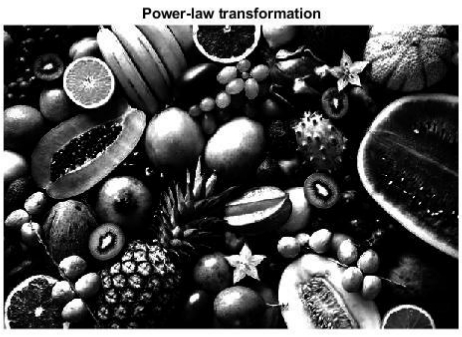
Matlab source code:

```
clc;
im=imread('fruits.jpg');
img=rgb2gray(im);
figure(1);
subplot(121);
imshow(img);
title('Input image.');
img2=im2double(img);
c=input('Input constant c:');
gamma=input('Input constant gamma:');
trans_img = c*(img2.^gamma);
subplot(122);
imshow(trans_img);
title('Power-law transformation')
```

Input and Output:

```
Input constant c: 2
Input constant gamma: 5
```





Problem No: 2
Problem Title: Write a MATLAB program for (i) showing histogram (ii) contrast stretching (iii) histogram equalization of a gray level image.

(i). Showing histogram of a gray level image:

Histogram of an image, like other histograms also shows frequency. But an image histogram, shows frequency of pixels intensity values. In an image histogram, the x axis shows the gray level intensities and the y axis shows the frequency of these intensities.

Histograms has many uses in image processing. The first use as it has also been discussed above is the analysis of the image. We can predict about an image by just looking at its histogram. Its like looking an x ray of a bone of a body.

The second use of histogram is for brightness purposes. The histograms has wide application in image brightness. Not only in brightness, but histograms are also used in adjusting contrast of an image.

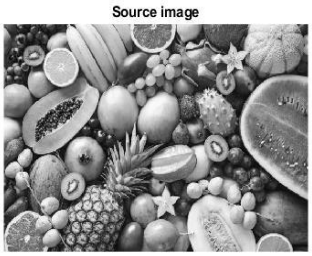
Another important use of histogram is to equalize an image.

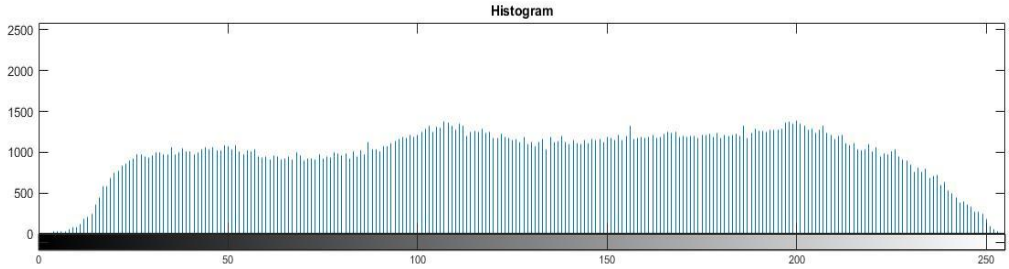
And last but not the least, histogram has wide use in thresholding. This is mostly used in computer vision.

Matlab source code:

```
clc;
clear all;
close all;
im=imread('fruits.jpg');
img=rgb2gray(im);
figure(1);
subplot(2,1,1);
imshow(img);
title('Source image')
subplot(2,1,2);
imhist(img);
title('Histogram');
```

Input and Output:





(ii). Contrast stretching of a gray level image:

Contrast stretching (often called normalization) is a simple image enhancement technique that attempts to improve the contrast in an image by `stretching' the range of intensity values it contains to span a desired range of values, *e.g.* the the full range of pixel values that the image type concerned allows. It differs from the more sophisticated histogram equalization in that it can only apply a *linear* scaling function to the image pixel values

Before the stretching can be performed it is necessary to specify the upper and lower pixel value limits over which the image is to be normalized. Often these limits will just be the minimum and maximum pixel values that the image type concerned allows. For example for 8-bit graylevel images the lower and upper limits might be 0 and 255. Call the lower and the upper limits *a* and *b* respectively.

The simplest sort of normalization then scans the image to find the lowest and highest pixel values currently present in the image. Call these *c* and *d*. Then each pixel *P* is scaled using the following function:

$$P_{out} = (P_{in} - c)\left(\frac{b-a}{d-c}\right) + a$$

Values below 0 are set to 0 and values about 255 are set to 255.

Matlab source code:

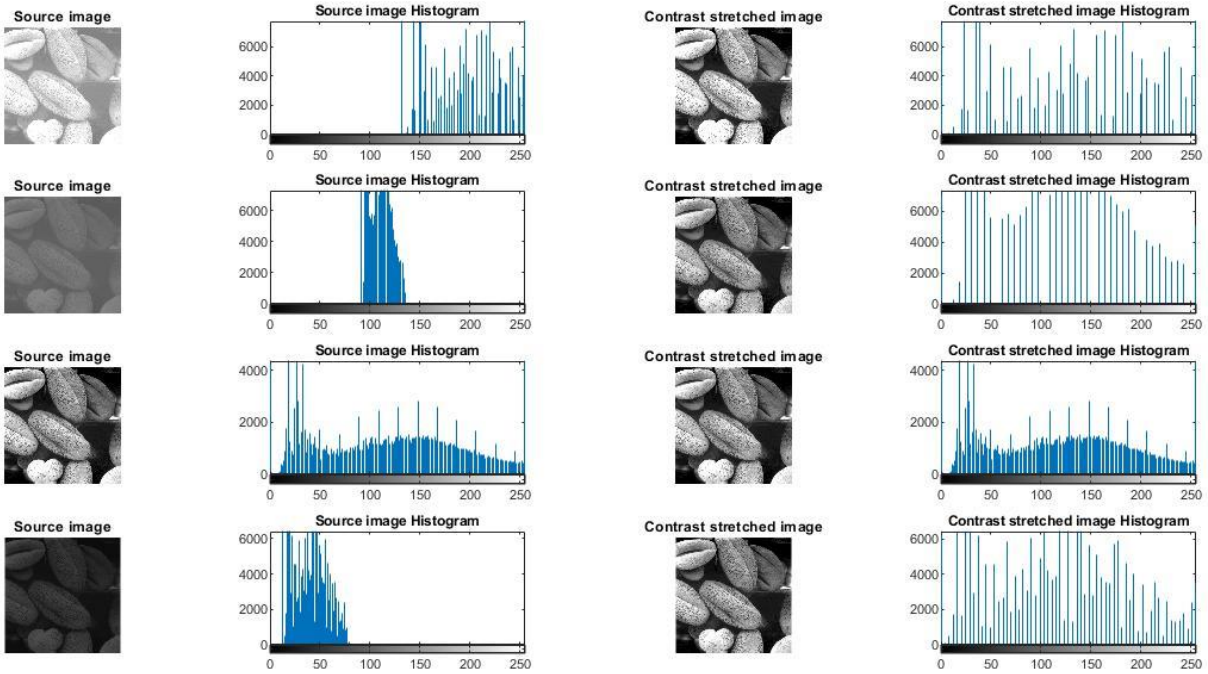
```
clc;
i1=imread('C:\Users\Shihab\Downloads\Compressed\DIP3E_CH03_Original_Images\Fig0316(1)(top_left).tif');

figure(1);
subplot(441);
imshow(i1);
title('Source image')

subplot(442);
imhist(i1);
title('Source image Histogram')
```

```
cs_i1=imadjust(i1);

subplot(443);
imshow(cs_i1);
title('Contrast stretched image');

subplot(444);
imhist(cs_i1);
title('Contrast stretched image
Histogram');

i1=imread('C:\Users\Shihab\Downloads\Compre
ssed\DIP3E_CH03_Original_Images\Fig0316(2)(
2nd_from_top).tif');

subplot(445);
imshow(i1);
title('Source image')

subplot(446);
imhist(i1);
title('Source image Histogram')

cs_i1=imadjust(i1);

subplot(447);
imshow(cs_i1);
title('Contrast stretched image');

subplot(448);
imhist(cs_i1);
title('Contrast stretched image
Histogram');

i3=imread('C:\Users\Shihab\Downloads\Compre
ssed\DIP3E_CH03_Original_Images\Fig0316(3)(
third_from_top).tif');
```

```
subplot(449);
imshow(i3);
title('Source image')

subplot(4,4,10);
imhist(i3);
title('Source image Histogram')

cs_i3=imadjust(i3);

subplot(4,4,11);
imshow(cs_i3);
title('Contrast stretched image');

subplot(4,4,12);
imhist(cs_i3);
title('Contrast stretched image
Histogram');

i4=imread('C:\Users\Shihab\Downloads\Compre
ssed\DIP3E_CH03_Original_Images\Fig0316(4)(
bottom_left).tif');

subplot(4,4,13);
imshow(i4);
title('Source image')

subplot(4,4,14);
imhist(i4);
title('Source image Histogram')

cs_i4=imadjust(i4);

subplot(4,4,15);
imshow(cs_i4);
title('Contrast stretched image');

subplot(4,4,16);
```

```
imhist(cs_i4);
title('Contrast stretched image Histogram');
```

Input and Output:



(iii). Histogram equalization of a gray level image:

Histogram Equalization is a computer image processing technique used to improve contrast in images. It accomplishes this by effectively spreading out the most frequent intensity values, i.e. stretching out the intensity range of the image. This method usually increases the global contrast of images when its usable data is represented by close contrast values. This allows for areas of lower local contrast to gain a higher contrast.
Histogram equalization is used for equalizing all the pixel values of an image. Transformation is done in such a way that uniform flattened histogram is produced. Histogram equalization increases the dynamic range of pixel values and makes an equal count of pixels at each level which produces a flat histogram with high contrast image. While stretching histogram, the shape of histogram remains the same whereas in Histogram equalization, the shape of histogram changes and it generates only one image.

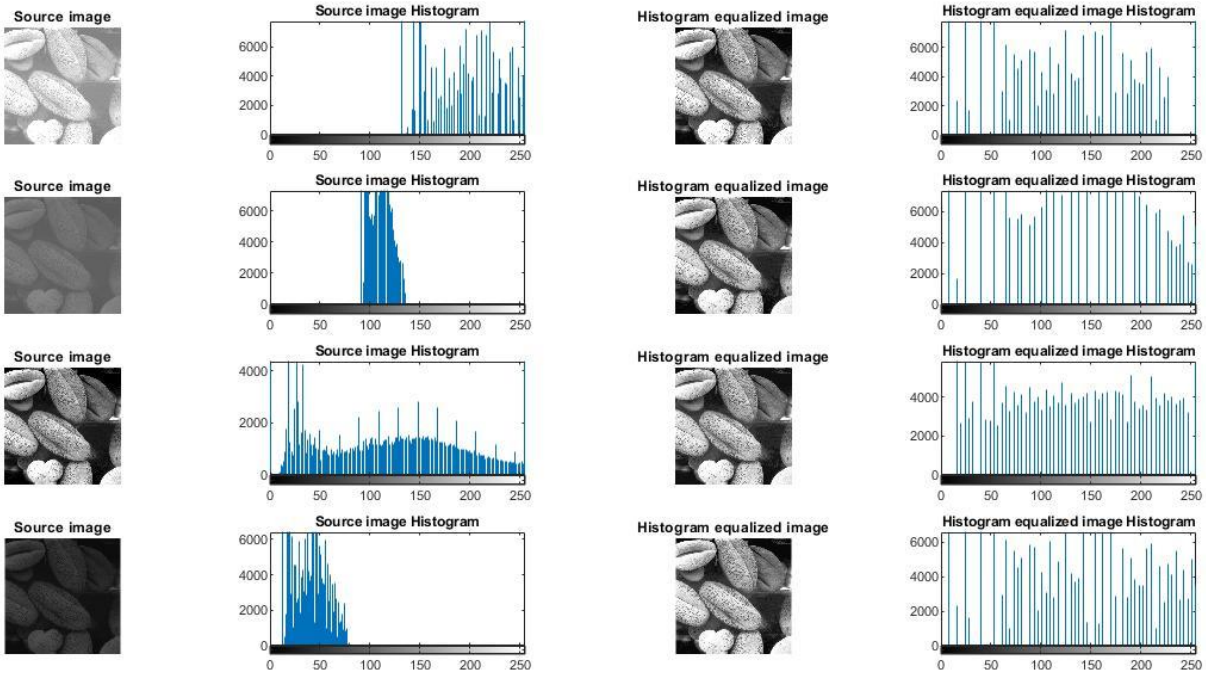
Matlab source code:

```
clc;
i1=imread('C:\Users\Shihab\Downloads\Compressed\DIP3E_CH03_Original_Images\Fig0316(1)(top_left).tif');

figure(1);
subplot(441);
imshow(i1);
title('Source image')

subplot(442);
imhist(i1);
title('Source image Histogram')

cs_i1=histeq(i1);

subplot(443);
imshow(cs_i1);
title('Histogram equalized image');

subplot(444);
imhist(cs_i1);
```

```
title('Histogram equalized image
Histogram');

i1=imread('C:\Users\Shihab\Downloads\Compre
ssed\DIP3E_CH03_Original_Images\Fig0316(2)(
2nd_from_top).tif');

subplot(445);
imshow(i1);
title('Source image')

subplot(446);
imhist(i1);
title('Source image Histogram')

cs_i1=histeq(i1);

subplot(447);
imshow(cs_i1);
title('Histogram equalized image');

subplot(448);
imhist(cs_i1);
title('Histogram equalized image
Histogram');

i3=imread('C:\Users\Shihab\Downloads\Compre
ssed\DIP3E_CH03_Original_Images\Fig0316(3)(
third_from_top).tif');

subplot(449);
imshow(i3);
title('Source image')

subplot(4,4,10);
imhist(i3);
title('Source image Histogram')
```

```
cs_i3=histeq(i3);

subplot(4,4,11);
imshow(cs_i3);
title('Histogram equalized image');

subplot(4,4,12);
imhist(cs_i3);
title('Histogram equalized image
Histogram');

i4=imread('C:\Users\Shihab\Downloads\Compre
ssed\DIP3E_CH03_Original_Images\Fig0316(4)(
bottom_left).tif');

subplot(4,4,13);
imshow(i4);
title('Source image')

subplot(4,4,14);
imhist(i4);
title('Source image Histogram')

cs_i4=histeq(i4);

subplot(4,4,15);
imshow(cs_i4);
title('Histogram equalized image');

subplot(4,4,16);
imhist(cs_i4);
title('Histogram equalized image
Histogram');
```

Input and Output:



Problem No: 3
Problem Title: Write a MATLAB program for (i) high pass and lowpass filter (ii) average filter (iii) median, max and min filter of a gray level image.

(i). Highpass and Lowpass filter of a gray level image:

Lowpass filter (smoothing): A low pass filter is used to pass low-frequency signals. The strength of the signal is reduced and frequencies which are passed is higher than the cut-off frequency. The amount of strength reduced for each frequency depends on the design of the filter. Smoothing is low pass operation in the frequency domain.

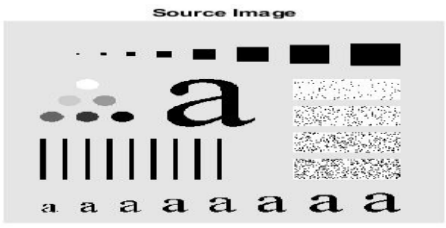
Highpass filters (sharpening): A high pass filter is the basis for most sharpening methods. An image is sharpened when contrast is enhanced between adjoining areas with little variation in brightness or darkness. A high pass filter tends to retain the high frequency information within an image while reducing the low frequency information.

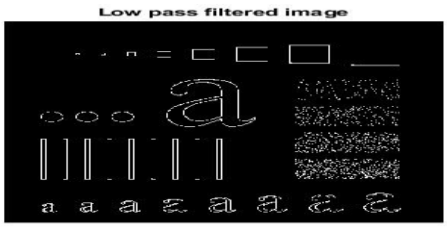
Matlab source code:

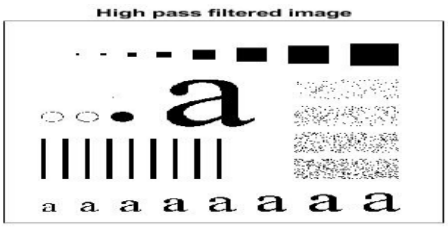
```
clc;
i1=imread('C:\Users\Shihab\Downloads\Compressed
\DIP3E_CH03_Original_Images\Fig0333(a)(test_pat
tern_blurring_orig).tif');

figure(1);
subplot(131);
imshow(i1);
title('Source Image');

% Lowpass filter
LowKernel = [ 1 1 1; 1 -10 1; 1 1 1 ];
lpfi = conv2(LowKernel,i1);
```

```
subplot(132)
imshow(lpfi);
title('Low pass filtered image');


% Highpass filter
HighKernel = [ -1 -1 -1; -1 10 -1; -1 -1 -1 ];
hpfi = conv2(HighKernel, i1);
subplot(133);
imshow(hpfi);
title('High pass filtered image');
```

Input and Output :







(ii). Average filter of a gray level image:

Average (or mean) filtering is a method of 'smoothing' images by reducing the amount of intensity variation between neighbouring pixels.

The average filter works by moving through the image pixel by pixel, replacing each value with the average value of neighbouring pixels, including itself.

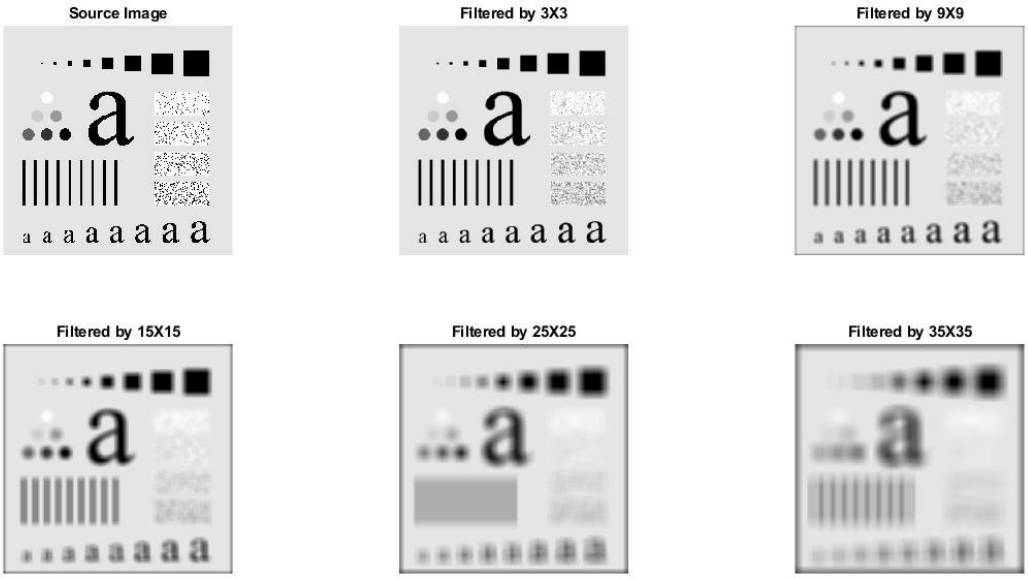
There are some potential problem:
A single pixel with a very unrepresentative value can significantly affect the average value of all the pixels in its neighborhood. 9 ¾ When the filter neighborhood straddles an edge, the filter will interpolate new values for pixels on the edge and so will blur that edge. This may be a problem if sharp edges are required in the output.

Matlab source code:

```
clc;
close all;
clear all;
i1=imread('C:\Users\Shihab\Downloads\Compre
ssed\DIP3E_CH03_Original_Images\Fig0333(a)(
test_pattern_blurring_orig).tif');

figure(1);
subplot(231);
imshow(i1);
title('Source Image');

M3=fspecial('average',3);
M9=fspecial('average',9);
M15=fspecial('average',15);
M25=fspecial('average',25);
M35=fspecial('average',35);

J3=imfilter(i1,M3);
J9=imfilter(i1,M9);
J15=imfilter(i1,M15);
J25=imfilter(i1,M25);
J35=imfilter(i1,M35);

subplot (232);
imshow(J3);
title('Filtered by 3X3');

subplot (233);
imshow(J9);
title('Filtered by 9X9');

subplot (234);
imshow(J15);
title('Filtered by 15X15');
```

```
subplot (235);
imshow(J25);
title('Filtered by 25X25');

subplot (236);
imshow(J35);
title('Filtered by 35X35');
```

Input and Output:



(iii). Median, max and min filter filter of a gray level image:

Median filter: Median filtering is a nonlinear method used to remove noise from images. It is widely used as it is very effective at removing noise while preserving edges. It is particularly effective at removing 'salt and pepper' type noise. The median filter works by moving through the image pixel by pixel, replacing each value with the median value of neighbouring pixels. The pattern of neighbours is called the "window", which slides, pixel by pixel over the entire image 2 pixel, over the entire image.

The median is calculated by first sorting all the pixel values from the window into numerical order, and then replacing the pixel being considered with the middle (median) pixel value.

Max Filter: The maximum filter is defined as the maximum of all pixels within a local region of an image. The maximum filter is typically applied to an image to remove negative outlier noise.

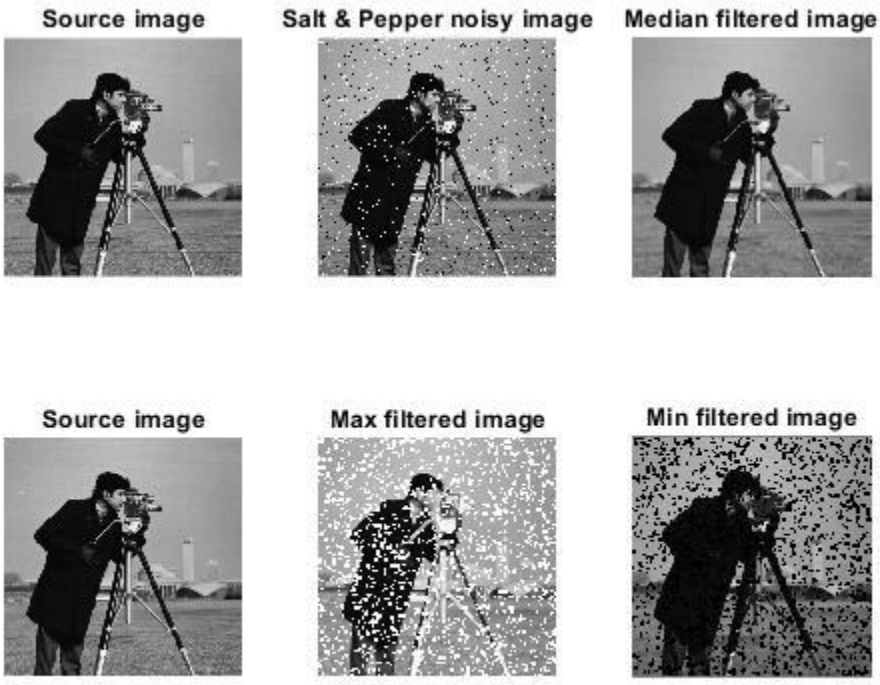
Min Filter: The minimum filter is defined as the minimum of all pixels within a local region of an image. The minimum filter is typically applied to an image to remove positive outlier noise.

Matlab source code:

```
clc;
clear all;
close all;

% Median Filter
img=imread('cameraman.tif');
[r,c]=size(img);
img=im2double(img);
subplot(231);imshow(img);title('Source
image');

%%% Adding Salt & Pepper Noise
noisy_img=imnoise(img,'salt & pepper');
subplot(232);imshow(noisy_img);title('Salt
& Pepper noisy image');

%%% Filtering , window 3*3
mf_img=ordfilt2(noisy_img,5,ones(3,3));
subplot(233);imshow(mf_img);title('Median
filtered image');

% Max & Min Filter
subplot(234);
```

```
imshow(img);
title('Source image');

maxf_img=ordfilt2(noisy_img,9,ones(3,3));
subplot(235);imshow(maxf_img);
title('Max filtered image');

minf_img=ordfilt2(noisy_img,1,ones(3,3));
subplot(236);
imshow(minf_img);
title('Min filtered image');
```

Input and Output:



Problem No: 4
Problem Title: Write a MATLAB program for (i) ideal (ii) Butterworth (iii) Gaussian lowpass and highpass filter of a gray level image in frequency domain.

(i). Ideal lowpass and highpass filter of a gray level image in frequency domain:

Low Pass Filtering

A low pass filter is the basis for most smoothing methods. An image is smoothed by decreasing the disparity between pixel values by averaging nearby pixels (see Smoothing an Image for more information).

Using a low pass filter tends to retain the low frequency information within an image while reducing the high frequency information. An example is an array of ones divided by the number of elements within the kernel, such as the following 3 by 3 kernel.

High Pass Filtering

A high pass filter is the basis for most sharpening methods. An image is sharpened when contrast is enhanced between adjoining areas with little variation in brightness or darkness (see Sharpening an Image for more detailed information).

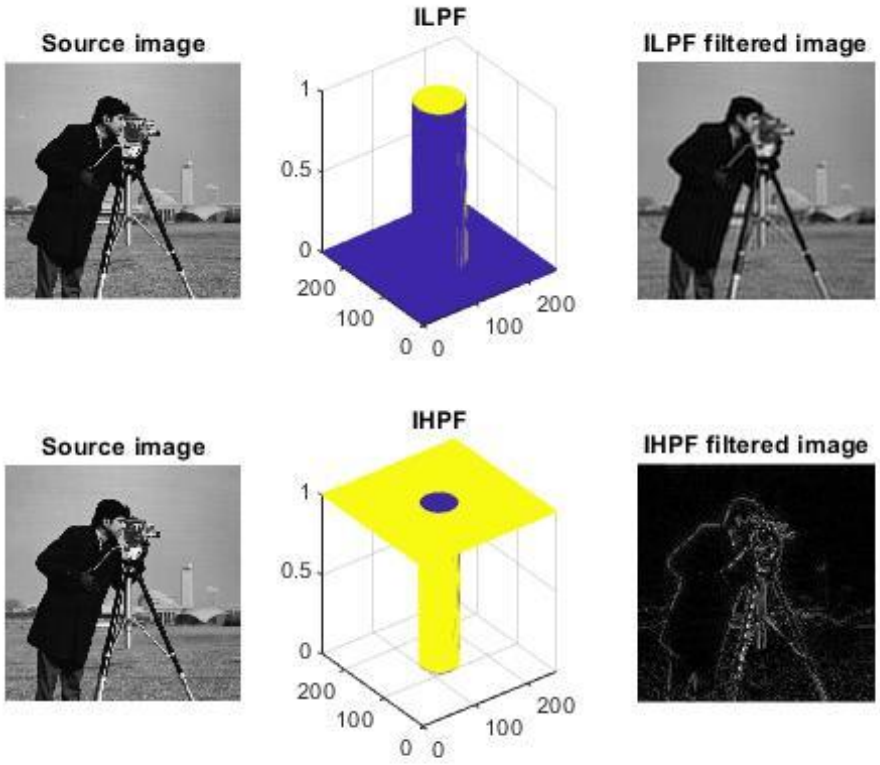
A high pass filter tends to retain the high frequency information within an image while reducing the low frequency information. The kernel of the high pass filter is designed to increase the brightness of the center pixel relative to neighboring pixels. The kernel array usually contains a single positive value at its center, which is completely surrounded by negative values. The following array is an example of a 3 by 3 kernel for a high pass filter

Matlab source code:

```
clc;
clear all;
close all;

% Ideal Lowpass Filter(ILPF)
img=imread('cameraman.tif');
[r,c]=size(img);
subplot(231);
imshow(img);
title('Source image');
IMG=fftshift(fft2(img));

%%%Creating filter
[u,v]=meshgrid(-floor(c/2):floor((c-1)/2),-
floor(r/2):floor((r-1)/2));
D=sqrt(u.^2+v.^2);
D0=40;
ILPF=(D<=D0);
subplot(232);
mesh(double(ILPF));
title('ILPF');

ILPF_IMG=IMG.*ILPF;
ilpf_img=ifft2(ILPF_IMG);
subplot(233);
imshow(mat2gray(abs(ilpf_img)));
title('ILPF filtered image');

% Ideal Highpass Filter(IHPF)
subplot(234);
imshow(img);
title('Source image');

D0=30;
IHPF=(D>D0);
subplot(235);
mesh(double(IHPF));
```

```
title('IHPF');

IHPF_IMG=IMG.*IHPF;
ihpf_img=ifft2(IHPF_IMG);
subplot(236);
imshow(mat2gray(abs(ihpf_img)));
title('IHPF filtered image');
```

Input and Output:



(ii). Butterworth lowpass and highpass filter of a gray level image in frequency domain:

Lowpaas Butterworth filter:

In the field of Image Processing, **Butterworth Lowpass Filter (BLPF)** is used for image smoothing in the frequency domain. It removes high-frequency noise from a digital image and preserves low-frequency components However, one main

disadvantage of the Butterworth filter is that it achieves this pass band flatness at the expense of a wide transition band as the filter changes from the pass band to the stop band. It also has poor phase characteristics as well. The ideal frequency response, referred to as a "brick wall" filter, and the standard Butterworth approximations.

The transfer function of BLPF of order is defined as-

Where,

- is a positive constant. BLPF passes all the frequencies less than value without attenuation and cuts off all the frequencies greater than it.
- This is the transition point between H(u, v) = 1 and H(u, v) = 0, so this is termed as *cutoff frequency*. But instead of making a sharp cut-off (like, **Ideal Lowpass Filter (ILPF)**), it introduces a smooth transition from 1 to 0 to reduce ringing artifacts.
- is the Euclidean Distance from any point (u, v) to the origin of the frequency plane, i.e,

Highpaas Butterworth filter:

In the field of Image Processing, **Butterworth Highpass Filter (BHPF)** is used for image sharpening in the frequency domain. Image Sharpening is a technique to enhance the fine details and highlight the edges in a digital image. It removes low-frequency components from an image and preserves high-frequency components.

This Butterworth highpass filter is the reverse operation of the Butterworth lowpass filter. It can be determined using the relation

This Butterworth highpass filter is the reverse operation of the Butterworth lowpass filter. It can be determined using the relation

-

where, is the transfer function of the highpass filter and is the transfer function of the corresponding lowpass filter.

The transfer function of BHPF of order is defined as-

Where,

- ➓ is a positive constant. BHPF passes all the frequencies greater than value without attenuation and cuts off all the frequencies less than it.
- ➓ This is the transition point between H(u, v) = 1 and H(u, v) = 0, so this is termed as *cutoff frequency*. But instead of making a sharp cut-off (like, **Ideal Highpass Filter (IHPF)**), it introduces a smooth transition from 0 to 1 to reduce ringing artifacts.

  is the Euclidean Distance from any point (u, v) to the origin of the frequency plane.

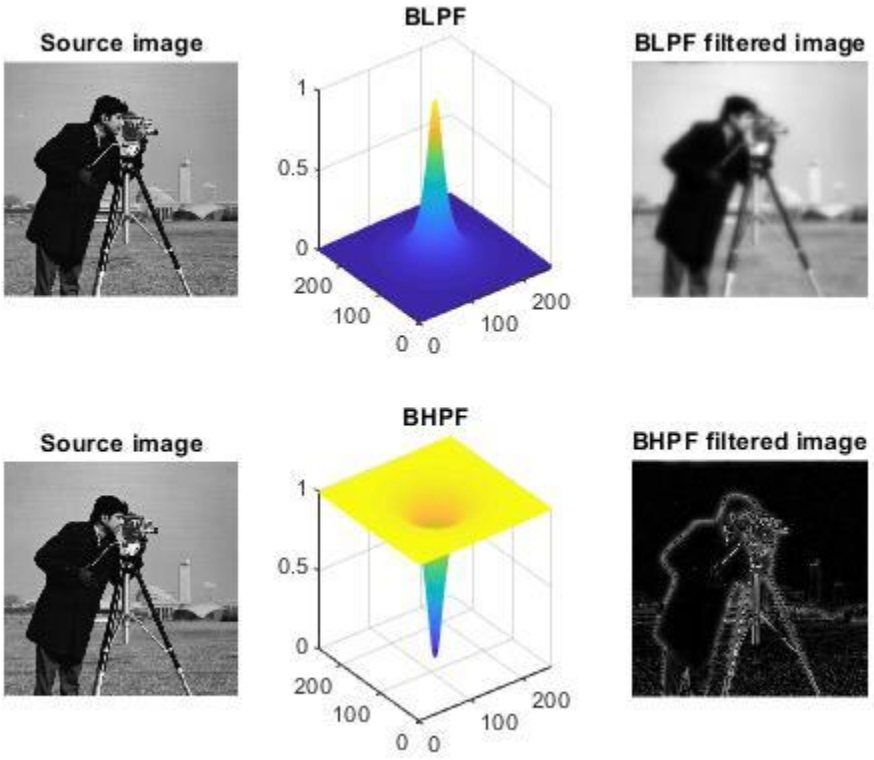
  .

Matlab source code:

```
clc;
clear all;
close all;

% Butterworth Lowpass Filter(BLPF)
img=imread('cameraman.tif');
[r,c]=size(img);
subplot(231), imshow(img), title('Source
image');
IMG=fftshift(fft2(img));

%%%Creating filter
[u, v]=meshgrid(-floor(c/2):floor((c-
1)/2),-floor(r/2):floor((r-1)/2));
D=sqrt(u.^2+v.^2);
D0=15;
```

```
n=1;
BLPF = 1./( 1.+ (D./D0).^(2*n) );
subplot(232), mesh(BLPF), title('BLPF')

BLPF_IMG=IMG.*BLPF;
blpf_img=ifft2(BLPF_IMG);
subplot(233),
imshow(mat2gray(abs(blpf_img))),
title('BLPF filtered image')

% Butterworth Highpass Filter(BHPF)
subplot(234), imshow(img), title('Source
image')
D0=15;
n=1;
BHPF=1./( 1.+ (D0./D).^(2*n) );
subplot(235), mesh(BHPF), title('BHPF')

BHPF_IMG=IMG.*BHPF;
bhpf_img=ifft2(BHPF_IMG);
subplot(236),
imshow(mat2gray(abs(bhpf_img))),
title('BHPF filtered image')
```

Input and Output:



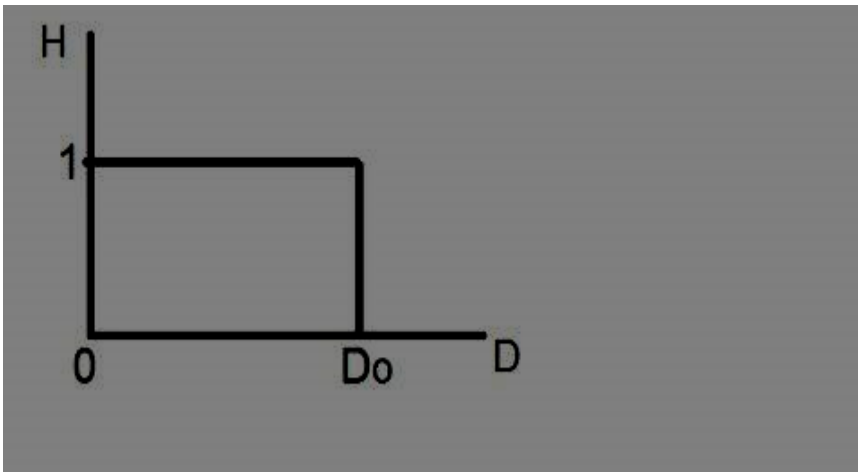
(iii). Gaussian lowpass and highpass filter of a gray level image in frequency domain:

Gaussian Highpass filter:

Gaussian low pass and Gaussian high pass filter minimize the problem that occur in ideal low pass and high pass filter.

This problem is known as ringing effect. This is due to reason because at some points transition between one color to the other cannot be defined precisely, due to which the ringing effect appears at that point.

Have a look at this graph



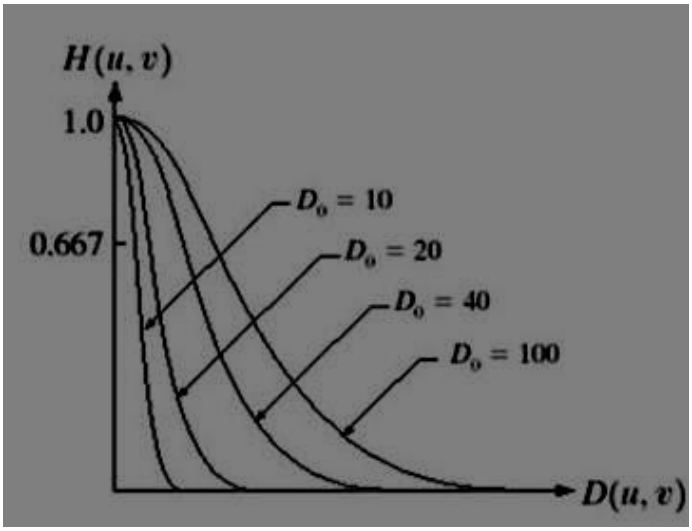
This is the representation of ideal low pass filter. Now at the exact point of Do, you cannot tell that the value would be 0 or 1. Due to which the ringing effect appears at that point.

So in order to reduce the effect that appears is ideal low pass and ideal high pass filter, the following Gaussian low pass filter and Gaussian high pass filter is introduced.

Gaussian Lowpass filter:

The concept of filtering and low pass remains the same, but only the transition becomes different and become more smooth.

The Gaussian low pass filter can be represented as



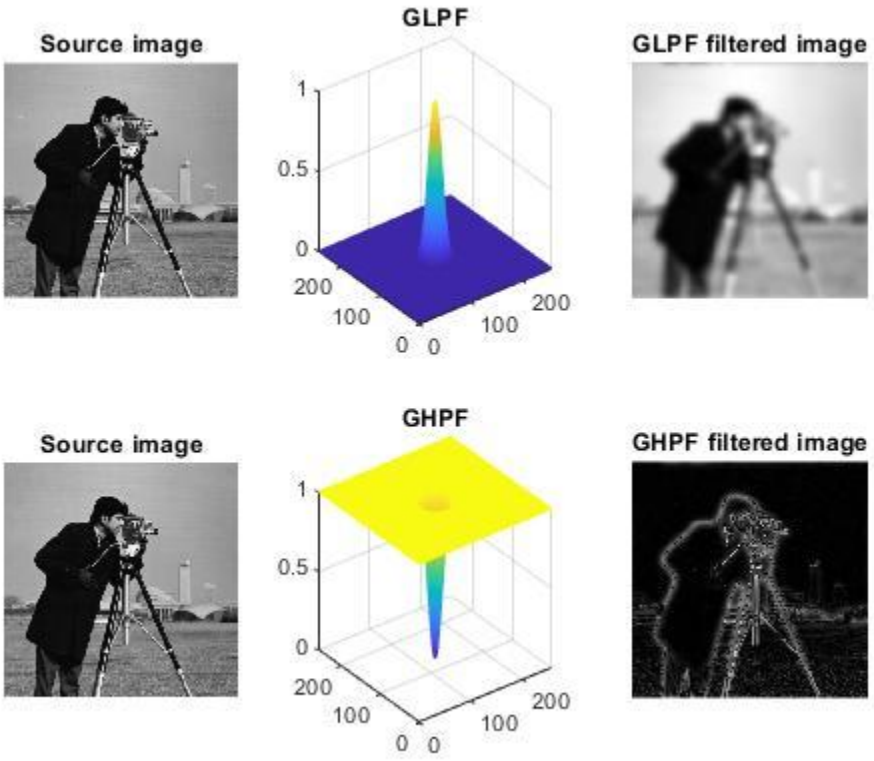
Note the smooth curve transition, due to which at each point, the value of Do, can be exactly defined.

Matlab source code:

```
clc;
clear all;
close all

% Gaussian Lowpass Filter(GLPF)
img=imread('cameraman.tif');
[r,c]=size(img);
subplot(231), imshow(img), title('Source
image')
IMG=fftshift(fft2(img));

%%%Creating filter
[u,v]=meshgrid(-floor(c/2):floor((c-1)/2),-
floor(r/2):floor((r-1)/2));
D=sqrt(u.^2+v.^2);
D0=10;
GLPF = exp (-(D.^2)./(2*D0^2)) ;
subplot(232), mesh(GLPF), title('GLPF')
GLPF_IMG=IMG.*GLPF;
glpf_img=ifft2(GLPF_IMG);
subplot(233),
imshow(mat2gray(abs(glpf_img))),
title('GLPF filtered image')

% Gaussian Highpass Filter(GHPF)
subplot(234), imshow(img), title('Source
image')
D0=10;
GHPF=1 - exp (-(D.^2)./(2*D0^2)) ;
subplot(235), mesh(GHPF), title('GHPF')
```

```
GHPF_IMG=IMG.*GHPF;
ghpf_img=ifft2(GHPF_IMG);
subplot(236),
imshow(mat2gray(abs(ghpf_img))),
title('GHPF filtered image')
```

Input and Output:



Problem No: 5
Problem Title: Write a MATLAB program for (i) Laplacian (ii) homomorphic filter of a gray level image in frequency domain.

(i). Laplacian filter of a gray level image:

Laplacian Operator is also a derivative operator which is used to find edges in an image. The major difference between Laplacian and other operators like Prewitt, Sobel, Robinson and Kirsch is that these all are first order derivative masks but Laplacian is a second order derivative mask. In this mask we have two further classifications one is Positive Laplacian Operator and other is Negative Laplacian Operator.

Another difference between Laplacian and other operators is that unlike other operators Laplacian didn't take out edges in any particular direction but it take out edges in following classification.

- Inward Edges
- Outward Edges

How it works :

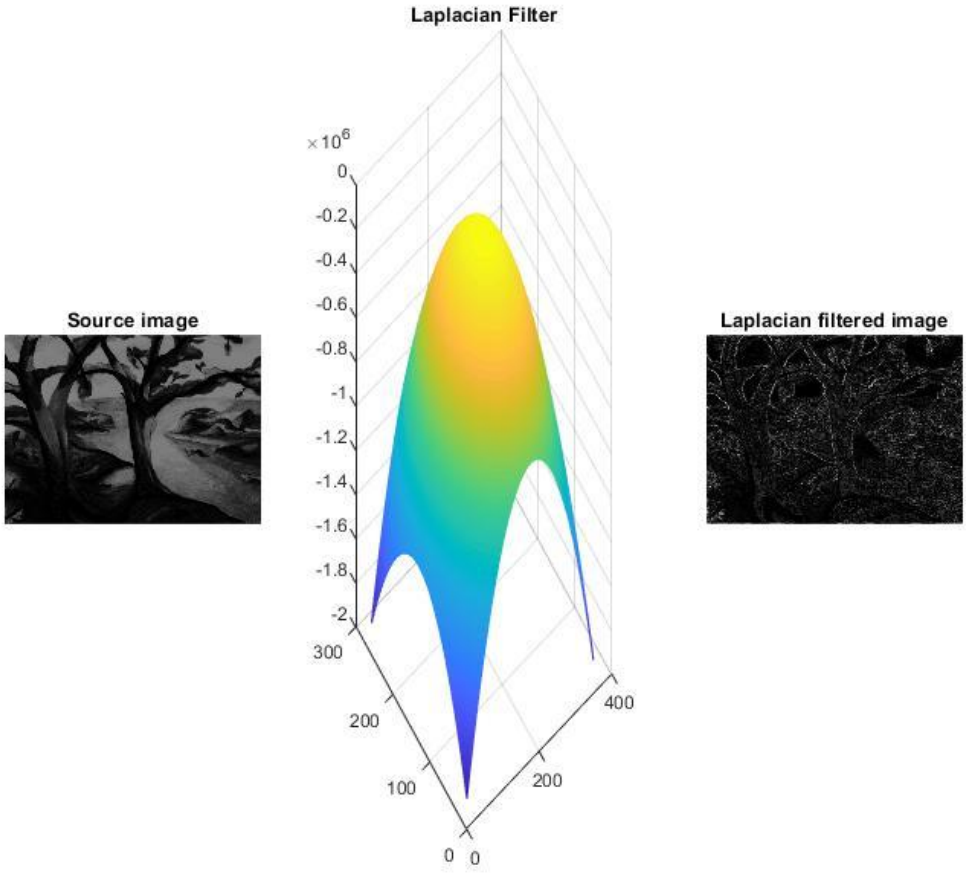
Laplacian is a derivative operator; its uses highlight gray level discontinuities in an image and try to deemphasize regions with slowly varying gray levels. This operation in result produces such images which have grayish edge lines and other discontinuities on a dark background. This produces inward and outward edges in an image

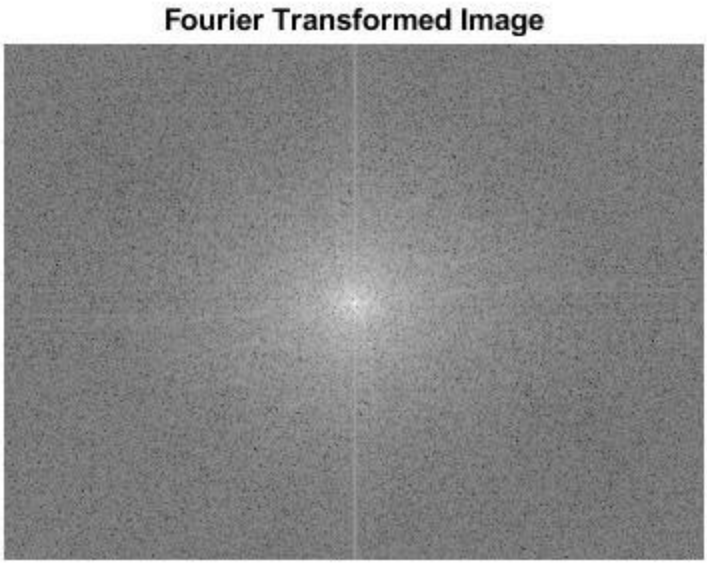
The important thing is how to apply these filters onto image. Remember we can't apply both the positive and negative Laplacian operator on the same image. we have to apply just one but the thing to remember is that if we apply positive Laplacian operator on the image then we subtract the resultant image from the original image to get the sharpened image. Similarly if we apply negative Laplacian operator then we have to add the resultant image onto original image to get the sharpened image.

Matlab source code:

```
clc;
clear all;
close all;

% Laplacian Filter(LF)
img=imread('trees.tif');
[r,c]=size(img);
subplot(131), imshow(img), title('Source
image')
IMG=fftshift(fft2(img));

%Dispaly Fourier Transformed Image
IMG1=log(1+abs(IMG));
m=max(IMG1(:));
figure(2), imshow(im2uint8(IMG1/m)),
title('Fourier Transformed Image');

%%%Creating filter
[u,v]=meshgrid(-floor(c/2):floor((c-1)/2),-
floor(r/2):floor((r-1)/2));
LF = -4*pi^2*(u.^2+v.^2) ;
figure(1), subplot(132), mesh(LF),
title('Laplacian Filter')

LF_IMG=IMG.*LF;
lf_img=ifft2(LF_IMG);
subplot(133),
imshow(mat2gray(abs(lf_img))),
title('Laplacian filtered image')
```

Input and Output:





(ii). Homomorphic filter of a gray level image:

Homomorphic filtering is sometimes used for image enhancement. It simultaneously normalizes the brightness across an image and increases contrast. Here homomorphic filtering is used to remove multiplicative noise. Illumination and reflectance are not separable, but their approximate locations in the frequency domain may be located. Since illumination and reflectance combine multiplicatively, the components are made additive by taking the logarithm of the image intensity, so that these multiplicative components of the image can be separated linearly in the frequency domain. Illumination variations can be thought of as a multiplicative noise, and can be reduced by filtering in the log domain.

Homomorphic filtering can be used for improving the appearance of a grayscale image by simultaneous intensity range compression (illumination) and contrast enhancement (reflection).

Where,

m = image,

i = illumination,

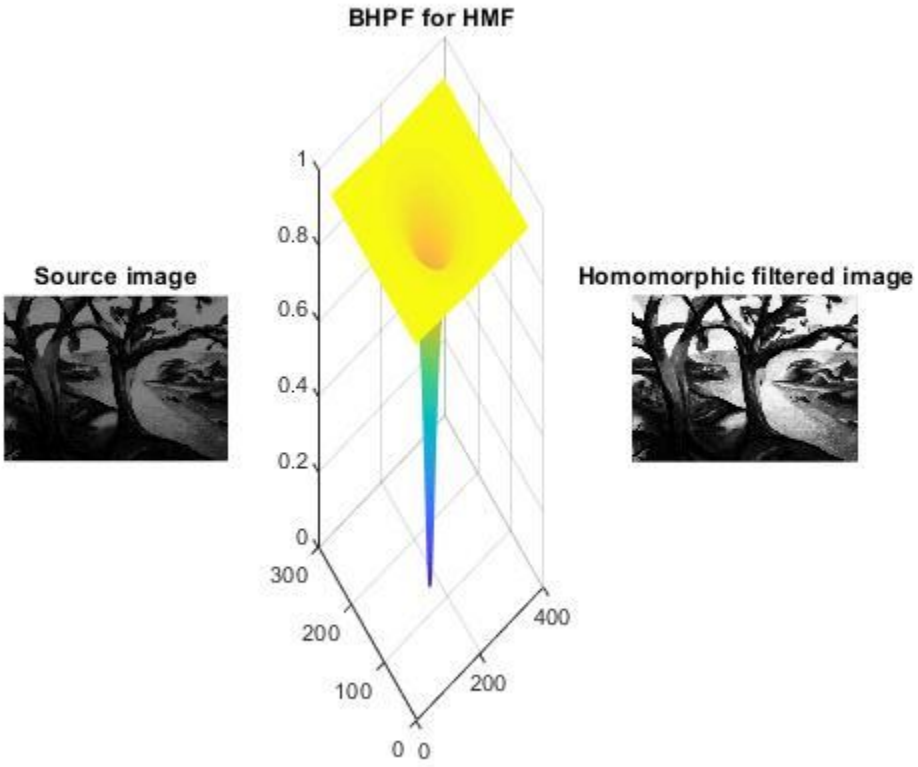
r = reflectance

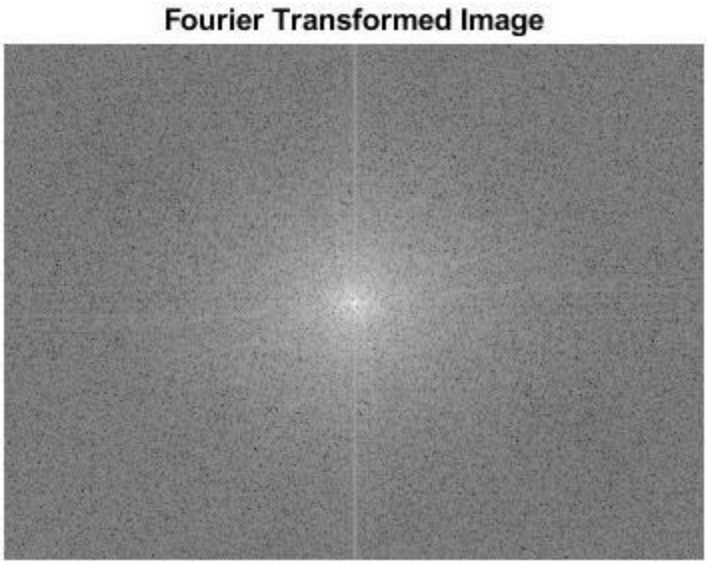
Matlab source code:

```
clc;
clear all;
close all;

% Homomorphic Filter(HMF)
img=imread('trees.tif');
[r,c]=size(img);
subplot(131), imshow(img), title('Source
image')
IMG=fftshift(fft2(img));

%Dispaly Fourier Transformed Image
IMG1=log(1+abs(IMG));
m=max(IMG1(:));
figure(2), imshow(im2uint8(IMG1/m)),
title('Fourier Transformed Image');
img=im2double(img);
IMG=fft2(log(img+0.01));

%%%Creating Butterworth High Pass Filter
for HMF
%%%Creating filter
[u,v]=meshgrid(-floor(c/2):floor((c-1)/2),-
floor(r/2):floor((r-1)/2));
D=sqrt(u.^2+v.^2);
D0=15;
n=1;
BHPF = 1./( 1.+ (D0./D).^(2*n) );
figure(1), subplot(132), mesh(BHPF),
title('BHPF for HMF')

BHPF_IMG=IMG.*BHPF;
real_img=real(ifft2(BHPF_IMG));
exp_img=exp(real_img);
```

```
subplot(133),
imshow(mat2gray(abs(exp_img))),
title('Homomorphic filtered image')
```

Input and Output:





Problem No: 6
Problem Title: Write a MATLAB program for (i) arithmetic and geometric (ii) harmonic and contraharmonic (iii) midpoint and alpha-trimmed mean filter of a gray level image.

(i). Arithmetic and geometric filter of a gray level image:

Arithmetic mean filter :

This is the simplest of the mean filters. Let Sxv represent the set of coordinates in a rectangular subimage window of size m X n, centered at point (x, y).The arithmetic mean filtering process computes the average value of the corrupted image g(x, y) in the area defined by Sxy.The value of the restored image at any point (x, y) is simply the arithmetic mean computed using the pixels in the region defined by S. In other words.

$$\hat{f}(x,y) = \frac{1}{mn} \sum_{(s,t)\in S_{xy}} g(s,t). \qquad (5.3\text{-}3)$$

This operation can be implemented using a convolution mask in which ail coefficients have value 1/mm. Noise is reduced as a result of blurring.

Geometric mean filter

An image restored using a geometric mean filter is given by the expression

$$\hat{f}(x,y) = \left[\prod_{(s,t)\in S_{xy}} g(s,t) \right]^{\frac{1}{mn}}. \qquad (5.3\text{-}4)$$

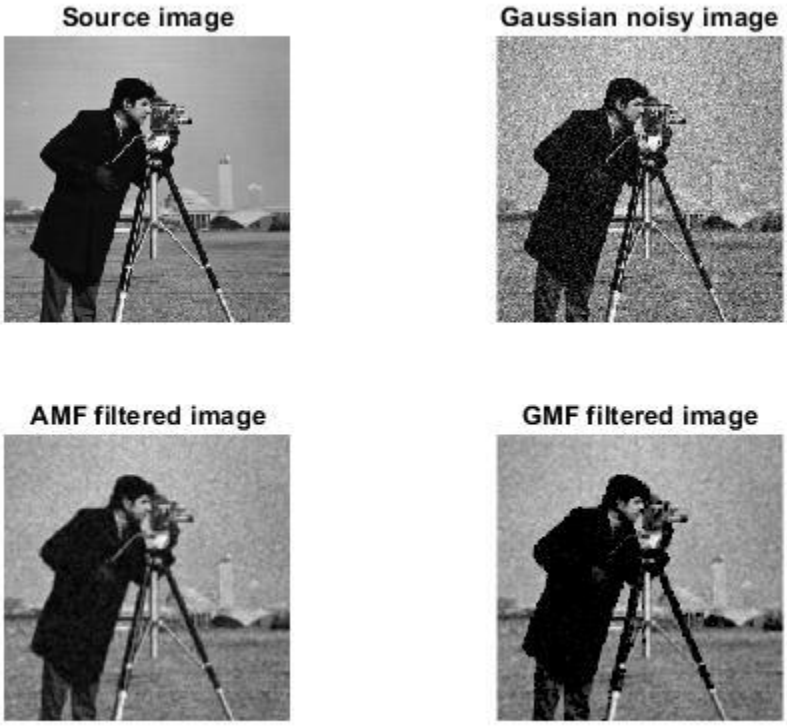
Here, each restored pixel is given by the product of the pixels in the subimage window, raised to the power 1/mn.

Matlab source code:

```
clc;
clear all;
close all;

% Arithmetic Mean Filter and Geometric Mean
Filter
img=imread('cameraman.tif');
[r,c]=size(img);
img=im2double(img);
subplot(221);imshow(img);title('Source
image');

%%%Adding Gaussian Noise
noisy_img=imnoise(img,'gaussian');
subplot(222);imshow(noisy_img);title('Gauss
ian noisy image');

%%% 'valid' convolution(3*3) , so image
dimension will be reduced
for i=1:r-2
 for j=1:c-2
 window = noisy_img(i:i+2,j:j+2);
```

```
 amf_img(i,j)= mean( window(:) );
 gmf_img(i,j)= geomean( window(:) );
 end
end

subplot(223);imshow(amf_img);title('AMF
filtered image');
subplot(224);imshow(gmf_img);title('GMF
filtered image');
```

Input and Output:



(ii). Harmonic and contraharmonic filter of a gray level image:

Harmonic mean filter :

The harmonic mean filtering operation is given by the expression

$$\hat{f}(x,y) = \frac{mn}{\sum_{(s,t)\in S_{xy}} \frac{1}{g(s,t)}}. \qquad (5.3\text{-}5)$$

The harmonic mean filter works well for salt noise, but fails for pepper noise. It does well also with other types of noise tike Gaussian noise.

Contraharmonic mean filter :

The contraharmonic mean filtering operation yields a restored image based on the expression

$$\hat{f}(x,y) = \frac{\sum_{(s,t)\in S_{xy}} g(s,t)^{Q+1}}{\sum_{(s,t)\in S_{xy}} g(s,t)^{Q}} \qquad (5.3\text{-}6)$$

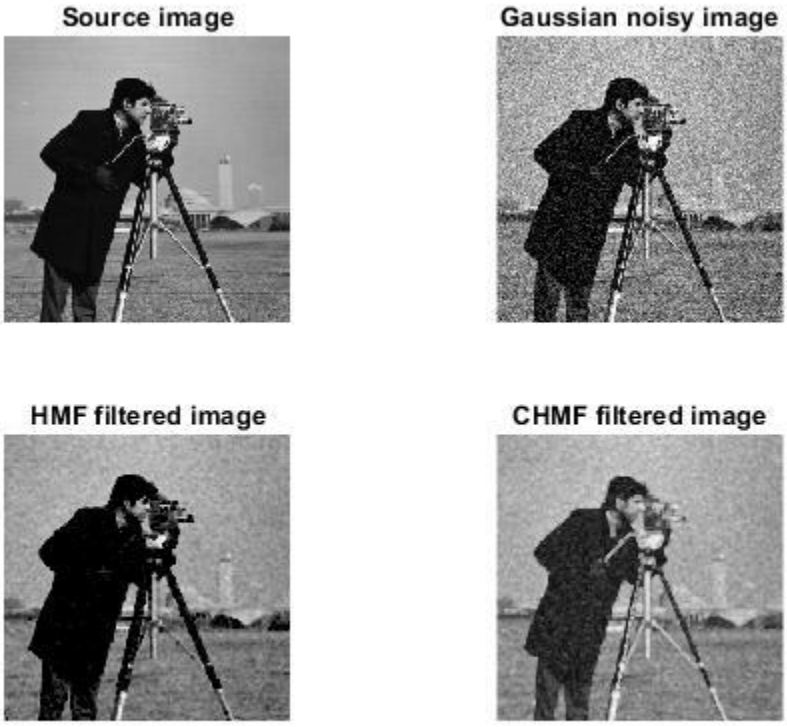
where Q is called the order of the filter. This filter is well suited for reducing or virtually eliminating the effects of salt-and-pepper noise. For positive values of Q, the filter eliminates pepper noise. For negative values of Q it eliminates salt noise. It cannot do both simultaneously. Note that the contraharmonic filter reduces to the arithmetic mean filter if Q = 0, and to the harmonic mean filter if Q = - 1.

Matlab source code:

```
clc;
clear all;
close all;

% Harmonic Mean Filter and Conrtaharmonic
Mean Filter
img=imread('cameraman.tif');
[r,c]=size(img);
img=im2double(img);
subplot(221);
imshow(img);
title('Source image');

%%%Adding Gaussian Noise
noisy_img=imnoise(img,'gaussian');
subplot(222);
imshow(noisy_img);
title('Gaussian noisy image');

%%% 'valid' convolution(3*3) , so image
dimension will be reduced
Q=1.5;
for i=1:r-2
 for j=1:c-2
 window = noisy_img(i:i+2,j:j+2);
 hmf_img(i,j)= harmmean( window(:) );
 chmf_img(i,j)= sum( window(:).^(Q+1) ) ./
sum( window(:).^Q );
 end
end

subplot(223);
imshow(hmf_img);
title('HMF filtered image');

subplot(224);
imshow(chmf_img);
```

```
title('CHMF filtered image');
```

Input and Output:



(iii). Midpoint and alpha-trimmed mean filter of a gray level image:

Midpoint Filter :

The midpoint filter is typically used to filter images containing short tailed noise such as Gaussian and uniform type noises. The midpoint filter is defined as : where the coordinate (x+i, y+j) is defined over the image A and the coordinate (i, j) is defined over the N x N size square mask.

Alpha-Trimmed Mean Filter :

The modified alpha mean filter is similar to the method of removing the highest score and removing the lowest score to evaluate a player's level, that is, sort the data in the filtering range, remove d data from large to small, and remove d from small to large Data, calculate the average of the remaining data. Such filters are very good at removing pictures that have been contaminated by salt and pepper noise along with other types of noise

Matlab source code:

```
clc;
clear all;
close all;

% Midpoint Filter ( MF )
img=imread('cameraman.tif');
[r,c]=size(img);
img=im2double(img);
```

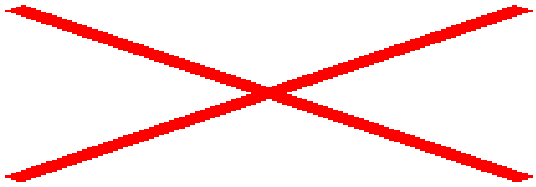
```
subplot(221);imshow(img);title('Source
image');

%%% Adding Gaussian Noise
noisy_img=imnoise(img,'Gaussian');
subplot(222);imshow(noisy_img);title('Gauss
ian noisy image');
%%% Filtering , window 3*3

midf_img= ( ordfilt2(noisy_img,9,ones(3,3))
+ ordfilt2(noisy_img,1,ones(3,3)) )./2;
subplot(223);imshow(midf_img);title('Midpoi
nt filtered image');

% Alpha-trimmed Mean Filter ( ATMF )
%%% 'valid' convolution(3*3) , so image
dimension will be reduced
d=25; %percent

for i=1:r-2
 for j=1:c-2
 window = noisy_img(i:i+2,j:j+2);
 atmf_img(i,j)= trimmean( window(:),d );
 end
end

subplot(224);
imshow(atmf_img);
title('ATMF filtered image');
```

Input and Output:

Problem No: 7
Problem Title: Write a MATLAB program for (i) ideal (ii) Butterworth (iii) Gaussian bandreject and bandpass filter of a gray level image.

(i). Ideal bandreject and bandpass filter of a gray level image:

Bandpass and bandreject filter transfer functions in the frequency domain can be constructed by combining lowpass and highpass filter transfer functions. In other words, the lowpass filter transfer functions are the basis for forming highpass, bandreject and bandpass filter functions. A bandpass filter transfer function is obtained from a band-reject function:



A low-pass filter passes frequencies that are below the cutoff frequency, and a high-pass filter passes frequencies that are above the cutoff frequency. A band-pass filter, in contrast, passes frequencies that fall only within a relatively narrow range, and a band-reject filter (also called a band-stop or notch filter) passes all frequencies except those that fall within a relatively narrow range.

Band-pass filters are widely used in communications systems because they can separate a received signal from other received signals that occupy adjacent frequency bands. Band-pass filters can also be used to identify the pitch of an incoming audio signal.

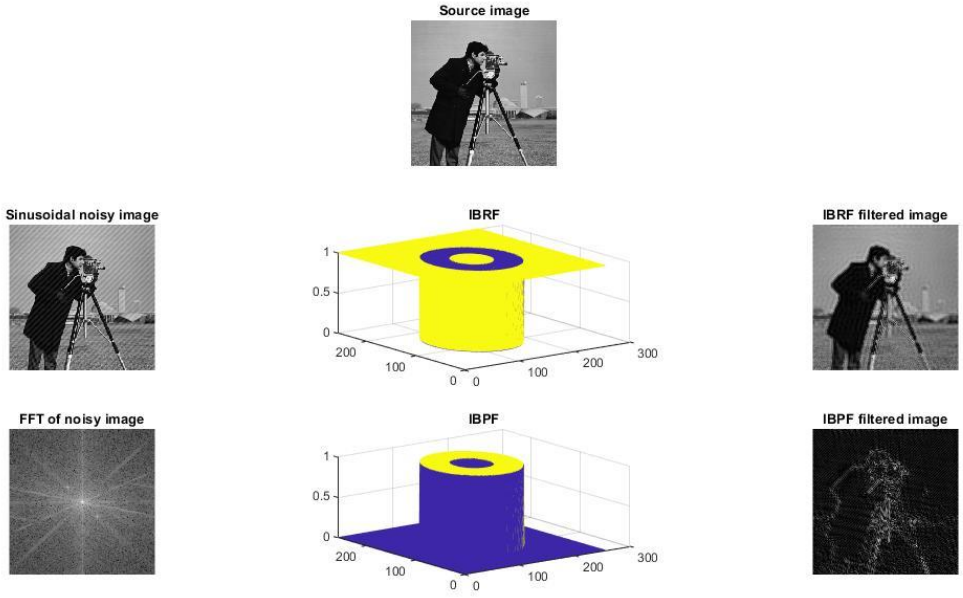
Matlab source code:

```
clc;
clear all;
close all;
```

```
% Ideal Bandreject Filter(IBRF)
img=imread('cameraman.tif');
[r,c]=size(img);
figure(1);
subplot(332)
imshow(img);
title('Source image');
[u,v]=meshgrid(-floor(c/2):floor((c-1)/2),-
floor(r/2):floor((r-1)/2));

%%%Adding Noise
sin_noise= 15*sin( 2*pi*1/10*u +
2*pi*1/10*v);
noisy_img=double(img)+sin_noise;
NOISY_IMG=fftshift(fft2(noisy_img));
subplot(334);
imshow(noisy_img,[]);
title('Sinusoidal noisy image')

subplot(337);
imshow(mat2gray(log(1+abs(NOISY_IMG))));
title('FFT of noisy image');

%%%Creating filter
D=sqrt(u.^2+v.^2);
D0=50;
W=40;
IBRF= ( D<(D0-W/2) | D>(D0+W/2) );
subplot(335);mesh(IBRF);title('IBRF')
IBRF_IMG=NOISY_IMG.*IBRF;
ibrf_img=ifft2(IBRF_IMG);
subplot(336);
imshow(mat2gray(abs(ibrf_img)));
title('IBRF filtered image');

% Ideal Bandpass Filter(IBPF)
IBPF= 1 - IBRF ;
subplot(338);
```

```
mesh(IBPF);
title('IBPF')

IBPF_IMG=NOISY_IMG.*IBPF;
ibpf_img=ifft2(IBPF_IMG);
subplot(339);
imshow(mat2gray(abs(ibpf_img)));
title('IBPF filtered image');
```

Input and Output:



(ii). Butterworth bandreject and bandpass filter of a gray level image:

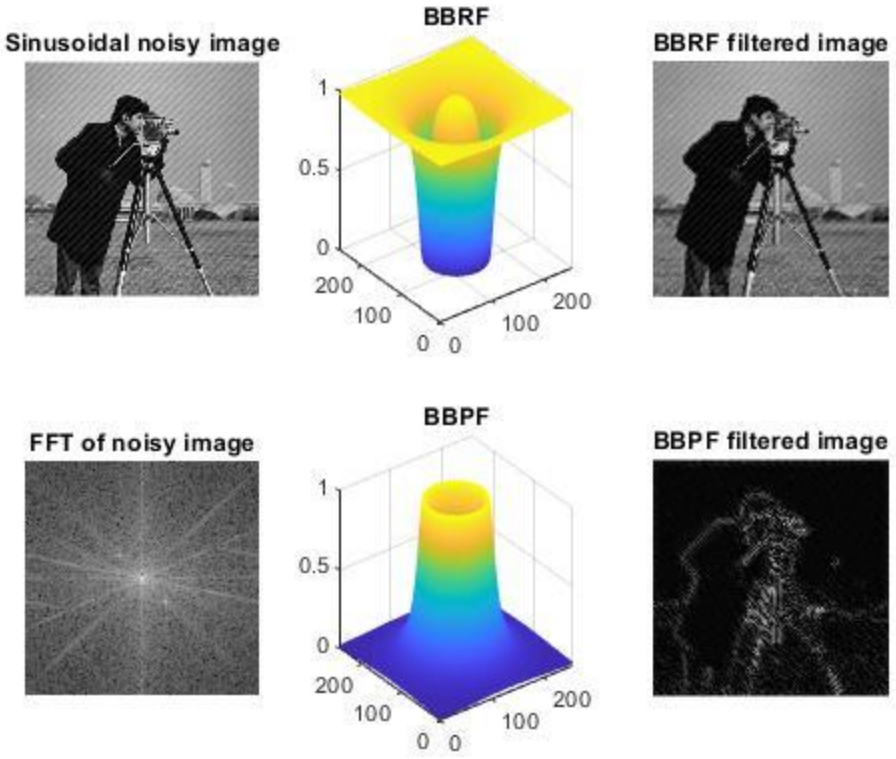
Bandpass and bandreject filter transfer functions in the frequency domain can be constructed by combining lowpass and highpass filter transfer functions. In other words, the lowpass filter transfer functions are the basis for forming highpass, bandreject and bandpass filter functions. A bandpass filter transfer function is obtained from a band-reject function

Matlab source code:

```
clc;
clear all;
close all;

% Butterworth Bandreject Filter(BBRF)
img=imread('cameraman.tif');
[r,c]=size(img);
imshow(img);title('Source image');
[u,v]=meshgrid(-floor(c/2):floor((c-1)/2),-
floor(r/2):floor((r-1)/2));

%%%Adding Noise
sin_noise= 15*sin( 2*pi*1/10*u +
2*pi*1/10*v);
```

```
noisy_img=double(img)+sin_noise;
NOISY_IMG=fftshift(fft2(noisy_img));
figure(2)
subplot(231);
imshow(noisy_img,[]);
title('Sinusoidal noisy image');

subplot(234);
imshow(mat2gray(log(1+abs(NOISY_IMG))));
title('FFT of noisy image');

%%%Creating filter
D=sqrt(u.^2+v.^2);
D0=50;
n=1;
W=20;
BBRF=1./( 1.+ ( (D.*W) ./ (D.^2-D0.^2) )
.^(2*n) );
subplot(232);mesh(BBRF);title('BBRF')
BBRF_IMG=NOISY_IMG.*BBRF;
bbrf_img=ifft2(BBRF_IMG);
subplot(233);
imshow(mat2gray(abs(bbrf_img)));
title('BBRF filtered image')

% Butterworth Bandpass Filter(BBPF)
BBPF= 1 - BBRF;
subplot(235);
mesh(BBPF);
title('BBPF')

BBPF_IMG=NOISY_IMG.*BBPF;
bbpf_img=ifft2(BBPF_IMG);
subplot(236);
imshow(mat2gray(abs(bbpf_img)));
title('BBPF filtered image')
```

Input and Output:



(iii). Gaussian bandreject and bandpass filter of a gray level image:

Bandpass and bandreject filter transfer functions in the frequency domain can be constructed by combining lowpass and highpass filter transfer functions. In other words, the lowpass filter transfer functions are the basis for forming highpass, bandreject and bandpass filter functions. A bandpass filter transfer function is obtained from a band-reject function

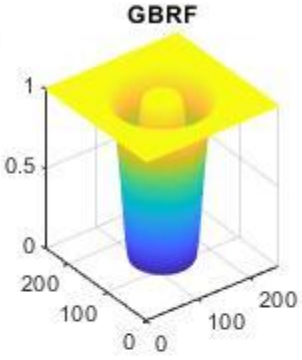
The key requirements of a bandpass transfer function are: (1) the values of the function must be in the range [0,1]; (2) the value of the function must be zero at a distant from the origin of the function; and (3) we must be able to specify a value for .

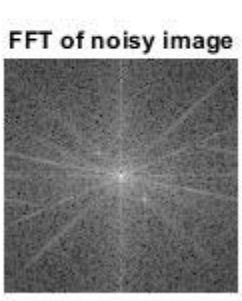
Matlab source code:

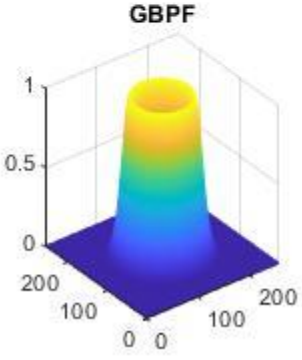
```
clc;
clear all;
close all;

% Gaussian Bandreject Filter(GBRF)
img=imread('cameraman.tif');
[r,c]=size(img);
imshow(img);
title('Source image')
[u,v]=meshgrid(-floor(c/2):floor((c-1)/2),-
floor(r/2):floor((r-1)/2));

%%%Adding Noise
sin_noise= 15*sin( 2*pi*1/10*u +
2*pi*1/10*v);
noisy_img=double(img)+sin_noise;
NOISY_IMG=fftshift(fft2(noisy_img));
figure(2)
subplot(231);
imshow(noisy_img,[]);
title('Sinusoidal noisy image');

subplot(234);
imshow(mat2gray(log(1+abs(NOISY_IMG))));
title('FFT of noisy image');

%%%Creating filter
D=sqrt(u.^2+v.^2);
D0=50;
```

```
W=20;
GBRF= 1 - exp ( -(1/2).* ( ((D.^2)-(D0.^2))
./ (D.*W) ).^2 ) ;
subplot(232);mesh(GBRF);title('GBRF');
GBRF_IMG=NOISY_IMG.*GBRF;
gbrf_img=ifft2(GBRF_IMG);
subplot(233);
imshow(mat2gray(abs(gbrf_img)));
title('GBRF filtered image');

% Gaussian Bandpass Filter(GBPF)
GBPF=1 - GBRF;
subplot(235);
mesh(GBPF);
title('GBPF');

GBPF_IMG=NOISY_IMG.*GBPF;
gbpf_img=ifft2(GBPF_IMG);
subplot(236);
imshow(mat2gray(abs(gbpf_img)));
title('GBPF filtered image');
```

Input and Output:







Problem No: 8
Problem Title: Write a MATLAB program for Wiener filter of a gray level image.

The most important technique for removal of blur in images due to linear motion or unfocussed optics is the Wiener filter. From a signal processing standpoint, blurring due to linear motion in a photograph is the result of poor sampling. Each pixel in a digital representation of the photograph should represent the intensity of a single stationary point in front of the camera. Unfortunately, if the shutter speed is too slow and the camera is in motion, a given pixel will be an amalgram of intensities from points along the line of the camera's motion. This is a two-dimensional analogy to

$$G(u,v)=F(u,v).H(u,v)$$

where F is the Fourier transform of an "ideal" version of a given image, and H is the blurring function. In this case H is a since function: if three pixels in a line contain info from the same point on an image, the digital image will seem to have been convolved with a three-point boxcar in the time domain. Ideally one could reverse-engineer a Fest, or F estimate, if G and H are known. This technique is inverse filtering .

It should be noted that the image restoration tools described here work in a similar manner for cases with blur due to incorrect focus. In this case the only difference is in the selection of H. The 2-d Fourier transform of H for motion is a series of sinc functions in parallel on a line perpendicular to the direction of motion; and the 2-d Fourier transform of H for focus blurring is the sombrero function, described elsewhere.

In the real world, however, there are two problems with this method. First, H is not known precisely. Engineers can guess at the blurring function for a given circumstance, but determination of a good blurring function requires lots of trial and error. Second, inverse filtering fails in some circumstances because the sinc function goes to 0 at some values of x and y. Real pictures contain noise which becomes amplified to the point of destroying all attempts at reconstruction of an Fest.The best method to solve the second problem is to use Wiener filtering. This tool solves an estimate for F according to the following equation:

$$Fest(u,v) = |H(u,v)|^2.G(u,v)/(|H(u,v)|^2.H(u,v) + K(u,v))$$

Matlab source code:

```
clc;
clear all;
close all;

% Minimum Mean Square Error Filter( Wiener
Filter )
img=imread('cameraman.tif');
subplot(131)
imshow(img)
title('Source image')

%%%Adding Gaussian Noise
noisy_img=imnoise(img,'gaussian');
subplot(132);
imshow(noisy_img);
title('Gaussian noisy image');

wiener_img=wiener2(noisy_img,[5 5]);
subplot(133)
imshow(wiener_img);
title('Wiener filtered image')
```

Input and Output:

Problem No: 9
Problem Title: Write a MATLAB program for separating RGB and HSI components of a color image.

separating RGB components of a color image:

A color image \(f \in \RR^{N \times 3}\) is made of three independent images, one for each channel red, green and blue (RGB color space).

Size \(N= n \times n\) of the image.

```
n = 256;
N = n*n;
```

Loading an image \(f \in \RR^{N \times 3}\).

```
name = 'hibiscus';
f = rescale( load_image(name,n) );
```

One can display on screen a color image in RGB space using the rule of additive color mixing.

separating HIS components of a color image:

Color spaces provide a way to specify order and manipulate colors. While there are several color spaces, the choice of a suitable space for color representation remains a challenge for scientists researching processing and analysis of color images . The most of the color spaces have been developed for specific applications, but all come from the same concept: the trichromatic theory of primary colors red, green and blue. The HSI color space (Hue, Saturation and Intensity) define a model in terms of its components. This space has the ability to separate the intensity of the intrinsic color information, which would refer to the hue and saturation.

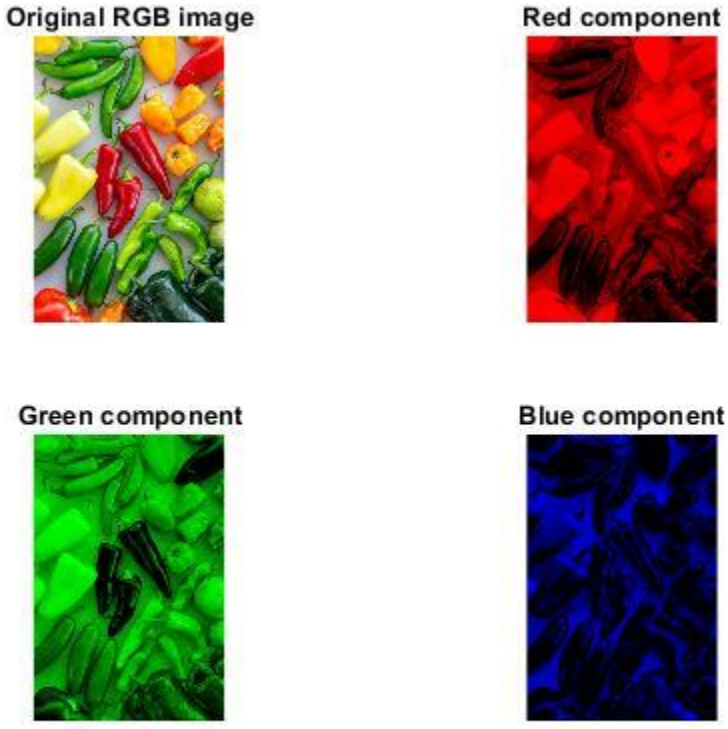
Matlab source code:

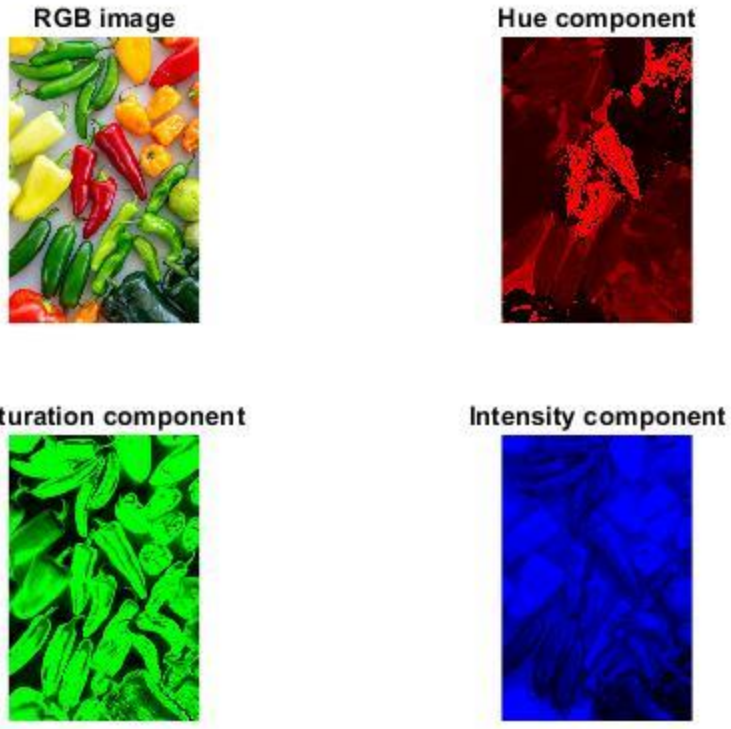
```
clc;
clear all;c
lose all;

rgb_img=imread('lenna.png');
rgb_img=im2double(rgb_img);
figure(1)
subplot(221);
imshow(rgb_img);
```

```
title('Original RGB image');

% Separating RGB components
R=rgb_img; R(:,:,2)=0; R(:,:,3)=0;
G=rgb_img; G(:,:,1)=0; G(:,:,3)=0;
B=rgb_img; B(:,:,1)=0; B(:,:,2)=0;

subplot(222);
imshow(R);
title('Red component');

subplot(223);
imshow(G);
title('Green component');

subplot(224);
imshow(B);
title('Blue component');

figure(2)
subplot(221);
imshow(rgb_img);
title('RGB image');

hsi_img=rgb2hsv(rgb_img);
% Separating HSI components
H=hsi_img; H(:,:,2)=0; H(:,:,3)=0;
S=hsi_img; S(:,:,1)=0; S(:,:,3)=0;
I=hsi_img; I(:,:,1)=0; I(:,:,2)=0;

subplot(222);
imshow(H);
title('Hue component');

subplot(223);
imshow(S);
title('Saturation component');
```

```
subplot(224);
imshow(I);
title('Intensity component');
```

Input and Output:





Problem No: 10
Problem Title: Write a MATLAB program for smoothing and sharpening of a color image.

Smoothing:
Image smoothing techniques have the goal of preserving image quality. In other words, to remove noise without losing the principal features of the image. However, there are several types of noise. The main three types are: impulsive, additive, and multiplicative. Impulsive noise is usually characterized by some portion of image pixels that are corrupted, leaving the others unchanged. Additive noise appears when the values of the original image have been modified by adding random values which follow a certain probability distribution. Finally, multiplicative noise is more difficult to be removed from images than additive noise, because in this case intensities vary along with signal intensity (e.g., speckle noise). There are different sources of noise and plenty of de noising methods for each kind of noise. The most common one is probably the so-called thermal noise. This impulsive noise is due to CCD sensor malfunction in the image acquisition process. Another interesting case is Gaussian noise, in which each pixel of the image will be changed from its original value by some small amount that follows a Gaussian distribution. This kind of noise is

modelled as an additive white Gaussian noise. So that, its presence can be simulated by adding random values from a zero-mean Gaussian distribution to the original pixel intensities in each image channel independently, where the standard deviation σ of the Gaussian distribution characterizes the noise intensity [44]. The elimination of this type of noise is known as smoothing, and this will be the type of noise elimination considered in this work. There are plenty of nonlinear methods for smoothing. In the rest of the section, we will review some of them

Sharpening:

Image enhancement process consists of a collection of techniques whose purpose is to improve image visual appearance and to highlight or recover certain details of the image for conducting an appropriate analysis by a human or a machine. During the acquisition process, several factors can influence on the quality of the image such as illumination conditions, ambient pressure or temperature fluctuations. In order to enhance the image, we try to convert it for getting details that are obscured, or to sharpen certain features of interest. There is a large number of applications.
of these techniques that include medical image analysis, remote sensing, high definition television, microscopic imaging, etc. The existence of such a variety implies that there will also be very different goals within image enhancement, according to each particular application. In some cases, the purpose is to enhance the contrast, in others, to emphasize details and/or borders of the image. We will refer to this last process as sharpening, although the difference is not always clear. The choice of the most suitable techniques for each purpose will be a function of the specific task to be conducted, the image content, the observer characteristics, and the viewing conditions

Matlab source code:

```
clc;
clear all;
close all;

rgb_img=imread('peppers.jpg');
rgb_img=im2double(rgb_img);
figure(1)
subplot(221);imshow(rgb_img);title('Origina
l RGB image');
```

```
% Creating filter
avg_filter=fspecial('average',[5 5]);
laplacian_filter=fspecial('laplacian',0.2);

% Filtering RGB components independently
R=rgb_img(:,:,1);
avg_R=imfilter(R,avg_filter);
lap_R=imfilter(R,laplacian_filter);
G=rgb_img(:,:,2);
avg_G=imfilter(G,avg_filter);
lap_G=imfilter(G,laplacian_filter);
B=rgb_img(:,:,3);
avg_B=imfilter(B,avg_filter);
lap_B=imfilter(B,laplacian_filter);

% Combining 3 channels after filtering
avg_rgb_img=cat(3,avg_R,avg_G,avg_B);
lap_rgb_img=cat(3,lap_R,lap_G,lap_B);

%%%% Converting to HSI from RGB
hsi_img=rgb2hsv(rgb_img);

% Filtering intensity component only
H=hsi_img(:,:,1);
S=hsi_img(:,:,2);
I=hsi_img(:,:,3);
avg_I=imfilter(I,avg_filter);
lap_I=imfilter(I,laplacian_filter);

% Combining 3 channels after filtering
avg_hsi_img=cat(3,H,S,avg_I);
lap_hsi_img=cat(3,H,S,lap_I);

% Converting to RGB from HSI
avg_rgb_img_from_hsi=hsv2rgb(avg_hsi_img);
lap_rgb_img_from_hsi=hsv2rgb(lap_hsi_img);
```

```
% Calculate difference
diff_avg_img=avg_rgb_img-
avg_rgb_img_from_hsi;
diff_lap_img=lap_rgb_img-
lap_rgb_img_from_hsi;
% Displaying
subplot(222);
imshow(avg_rgb_img);
title('Smoothing using RGB model');

subplot(223);
imshow(avg_rgb_img_from_hsi);
title('Smoothing using HSI model');

subplot(224);
imshow(diff_avg_img);
title('Difference');

figure(2)
subplot(221);
imshow(rgb_img);
title('Original RGB image');

subplot(222);
imshow(lap_rgb_img);
title('Sharpening using RGB model');

subplot(223);
imshow(lap_rgb_img_from_hsi);
title('Sharpening using HSI model');

subplot(224);
imshow(diff_lap_img);
title('Difference');
```

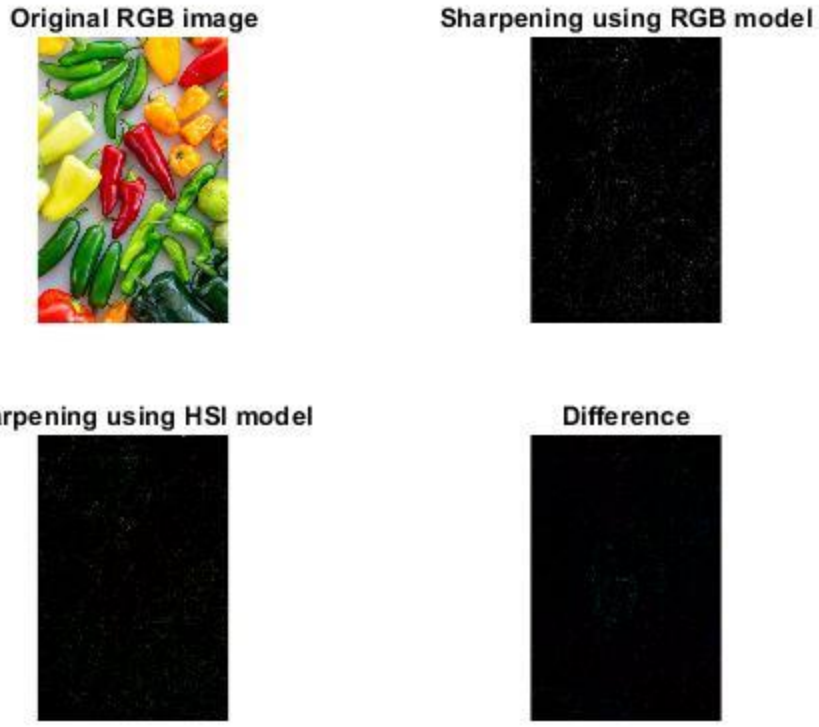
Input and Output:

Original RGB image

Smoothing using RGB model

Smoothing using HSI model

Difference



Problem No: 11
Problem Title: Write a MATLAB program for (i) Haar Transform (ii) one dimensional Discrete Wavelet Transform (iii) Discrete Cosine Transform of an image.

(i). Haar Transform of an image:

A complete orthogonal system of functions in L p [0, 1], p ∈ [0, ∞] which take values from the set {0, 2 j : j ∈ N} was defined by Haar [1]. This system of functions has property that each function continuous on interval [0, 1] may be represented by a uniformly and convergent series in terms of elements of this system. Nowadays, in the literature, there are some other definitions of the Haar functions [16]. Those definitions are mutually differing with respect to the values of Haar functions at the points of discontinuity. For example the original Haar definition is as follows [4]:.

$$haar(0,t) = 1, \text{ for } t \in [0,1); \quad haar(1,t) = \begin{cases} 1, & \text{for } t \in [0, \frac{1}{2}), \\ -1, & \text{for } t \in [\frac{1}{2}, 1) \end{cases}$$

Matlab source code:

```
%haartransfrom
clc;
clear all;
close all;

i=imread('peppers.png');
subplot(231); imshow(i); title('original
image');

%2D discrete wavelt transformations (single
level) using Haar Basic
%functions
[ia1, ih1, iv1, id1]=dwt2(i, 'haar');

%display different coefficients
a1=ia1/255; h1=log10(ih1)*0.3;
v1=log10(iv1)*0.3; d1=log10(id1)*0.3;
subplot(232); imshow(real(a1));
title('Approximation');
subplot(233); imshow(abs(h1));
title('Horizontal details');
subplot(234); imshow(abs(v1));
title('Vertical detail');
subplot(235); imshow(abs(d1));
title('diagonal detail');

%combined different coefficients in one
image
transformed_i_level_1=[a1 v1 ; h1 d1];
subplot(236);
imshow(abs(transformed_i_level_1));
```

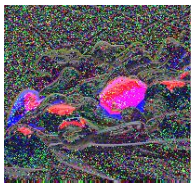
```
title('level-1 transformation');

%reconstruction from level-1 transformation
rec_i=idwt2(ia1, ih1, iv1, id1, 'haar');
figure(2);
subplot(211);imshow(rec_i/255);
title('Reconstruction from level-1
transformation');

%leve-2 transformation
[ia2, ih2, iv2, id2]=dwt2(ia1, 'haar');

%combined different coefficients in one
image
a2=ia2/255; h2=log10(ih2)*0.3;
v2=log10(iv2)*0.3; d2=log10(id2)*0.3;
tranformed_i_level_2=[[a2 v2 ; h2 d2] v1 ;
h1 d1];
subplot(212);
imshow(abs(tranformed_i_level_2));
title('Level-2 transformation');
```

Input and Output:

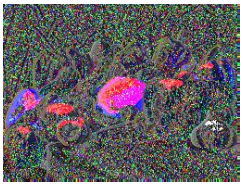
original image

Approximation

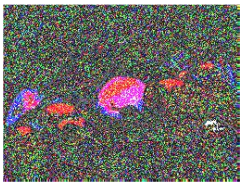
Horizontal deta



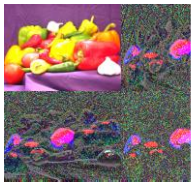
Vertical detail



diagonal detail

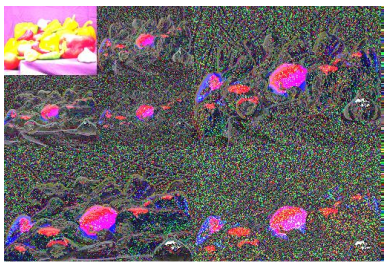


level-1 transform



Reconstruction from level-1 transformation

Level-2 transformation



(ii). One dimensional Discrete Wavelet Transform of an image:

An outstanding property of the Haar functions is that except function haar(0, t), the i–th Haar function can be generated by the restriction of the (j − 1)–th function to be half of the interval where it is different from zero, by multiplication with $\sqrt{2}$ and scaling over the interval [0, 1]. These properties give considerable interest of the Haar function, since they closely relate them to the wavelet theory. In this setting, the first two Haar functions are called the global functions, while all the others are denoted as the local functions. Hence, the Haar function, which is an odd rectangular pulse pair, is the simplest and oldest wavelet. The motivation for using the discrete wavelet transform is to obtain information that is more discriminating by providing a different resolution at different parts of the time–frequency plane. The wavelet transforms allow the partitioning of the time-frequency domain into nonuniform tiles in connection with the time–spectral contents of the signal. The wavelet methods are strongly connected with classical basis of the Haar functions; scaling and dilation of a basic wavelet can generate the basis Haar functions.

$$\Psi(t) = \begin{cases} 1, & \text{for } t \in [0, \frac{1}{2}), \\ -1, & \text{for } t \in [\frac{1}{2}, 1), \\ 0, & \text{otherwise.} \end{cases}$$

Matlab source code:

```
clc;
close all;
clear all;
X=imread('peppers.png');
figure(1);
subplot(1,2,1);
imshow(X);
title('Original Image');
```

```
X= double(X);
i=imresize(X,[512 512]);
subplot(1,2,2);
imshow(i);
title('Resize Image');

sX=size(X);
[LL,LH,HL,HH]= dwt2(X,'db1');

figure(2)
subplot(2,2,1);imshow(LL);title('LL band of
image');
subplot(2,2,2);imshow(LH);title('LH band of
image');
subplot(2,2,3);imshow(HL);title('HL band of
image');
subplot(2,2,4);imshow(HH);title('HH band of
image');
```

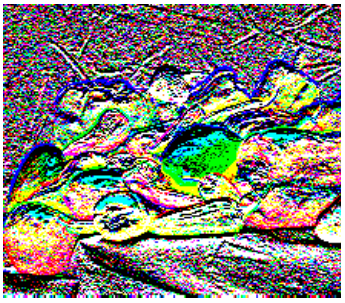
Input and Output:

Original Image

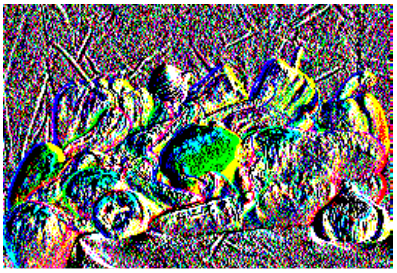
Resize Image

LL band of image

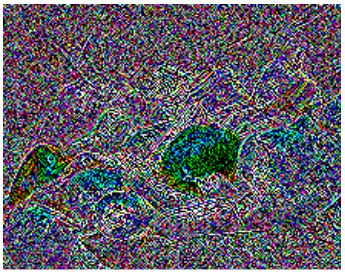
LH band of image



HL band of image



HH band of image



(iii). Discrete Cosine Transform of an image:

In Discrete Cosine Transformation, coefficients carry information about the pixels of the image. Also, much information is contained using very few coefficients, and the remaining coefficient contains minimal information. These coefficients can be removed without losing information. By doing this, the file size is reduced in the DCT domain. DCT is used for lossy compression.

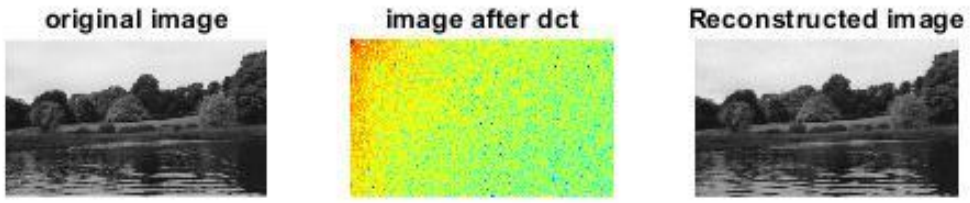
$$X_k = \frac{1}{2}(x_0 + (-1)^k x_{N-1}) + \sum_{n=1}^{N-2} x_n \cos\left[\frac{\pi}{N-1} nk\right] \qquad k = 0, \ldots, N-1.$$

Matlab source code:

```
%discrete cosine transformation
clc;
clear all;
close all;

img=imread('autumn.tif');
img=rgb2gray(img);
subplot(131); imshow(img); title('original
image');

%2d discrete cosine transformation
dct_img=dct2(img);
subplot(132);
imshow(log(abs(dct_img)), []);
colormap(gca, jet);
```

```
title('image after dct');

%inverse discrete cosine transormation
dct_img(abs(dct_img)<10)=0;
idct_img=idct2(dct_img);
subplot(133);
imshow(idct_img, [0, 255]);
title('Reconstructed image');
```

Input and Output:



Problem No: 12
Problem Title: Write a MATLAB program for edge detection using Sobel, Canny, Prewitt, Roberts, log, zero cross filter.

Edges are significant local changes of intensity in a digital image. An edge can be defined as a set of connected pixels that forms a boundary between two disjoint regions. There are three types of edges:

- Horizontal edges
- Vertical edges
- Diagonal edges

Edge Detection is a method of segmenting an image into regions of discontinuity. It is a widely used technique in digital image processing like

- pattern recognition
- image morphology
- feature extraction

Edge detection allows users to observe the features of an image for a significant change in the gray level. This texture indicating the end of one region in the image and the beginning of another. It reduces the amount of data in an image and preserves the structural properties of an image.

Edge Detection Operators are of two types:

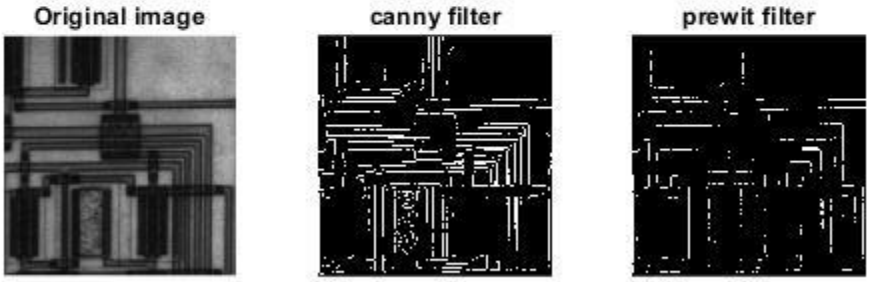
- **Gradient –** based operator which computes first-order derivations in a digital image like, Sobel operator, Prewitt operator, Robert operator
- **Gaussian –** based operator which computes second-order derivations in a digital image like, Canny edge detector, Laplacian of Gaussian

Matlab source code:

```
%edge detection
clc;
clear all;
close all;

I=imread('coins.png');
figure(1),
subplot(231), imshow(I), title('Original
image');
```

```
BW1=edge(I, 'sobel');
BW2=edge(I, 'canny');

subplot(232), imshow(BW1), title('Soble
filter');
subplot(233), imshow(BW2), title('Canny
filter');

I=imread('circuit.tif');
subplot(234), imshow(I), title('Original
image');

BW1=edge(I, 'canny');
BW2=edge(I, 'prewitt');

subplot(235), imshow(BW1), title('canny
filter');
subplot(236), imshow(BW2), title('prewit
filter');
```

Input and Output:



Problem No: 13
Problem Title: Write a MATLAB program to implement a LPF (FIR) with cutoff 8KHz and to denoise audio.

Low pass filter: For different low-pass filter, there is different weakening of frequency in each signal. When using it in audio applications, sometimes it is called high-cut filter or treble cut filter [5] . The application of low-pass filter in signal processing is equal to the effect of other areas. There are many types of low-pass filter; the most common one is Butterworth filter and Chebyshev filter.

The study aimed to filter and remove noise from audio, and in this study researcher followed the descriptive analytical method. The study sample consisted of an audio file and has been save Audio of on a formula (WAV), and the study used matlab 7.10.0 to read sound and design low pass filter, then insert the audio signal with the noise signal into the filter and output a signal audio without noise. The researcher has reached findings that differed according to the rank of the filter. The low pass filter has a great effect on removing noise during the registration process together with the times of implementation.

Matlab source code:

```
clc;
clear all;
close all;
[filename, pathname]=uigetfile(".", 'select
the input audio');
[x, Fs]=audioread(num2str(filename));

%filter implementation
Fsf=44100; %sampling frequency
Fp=8e3; %passband frequency in Hz
Fst=8.4e3; %stopband frequency in Hz
Ap=1; %passband ripple in db
Ast=95; %stopband attenuation in db

%design the filter
df=designfilt('lowpassfir',
'PassbandFrequency', Fp,
```

```
'StopbandFrequency', Fst, 'PassbandRipple',
Ap, 'StopbandAttenuation', Ast,
'SampleRate', Fsf);

fvtool(df) %visualize frequency response
fvtool(df);
xn=awgn(x,15, 'measured'); %signal
corrupted by white gaussian noise
y=filter(df, xn);
subplot(3,1,1), plot(x(1:450));
title('original signal');
subplot(3,1,2), plot(xn(1:450));
title('noisy signal');
subplot(3,1,3), plot(y(1:450));
title('filtered signal');
audiowrite('noisy_signal.wav',xn,Fs);
audiowrite('filtered_signal.wav',xn,Fs);
```

Input and Output:

Input audio:

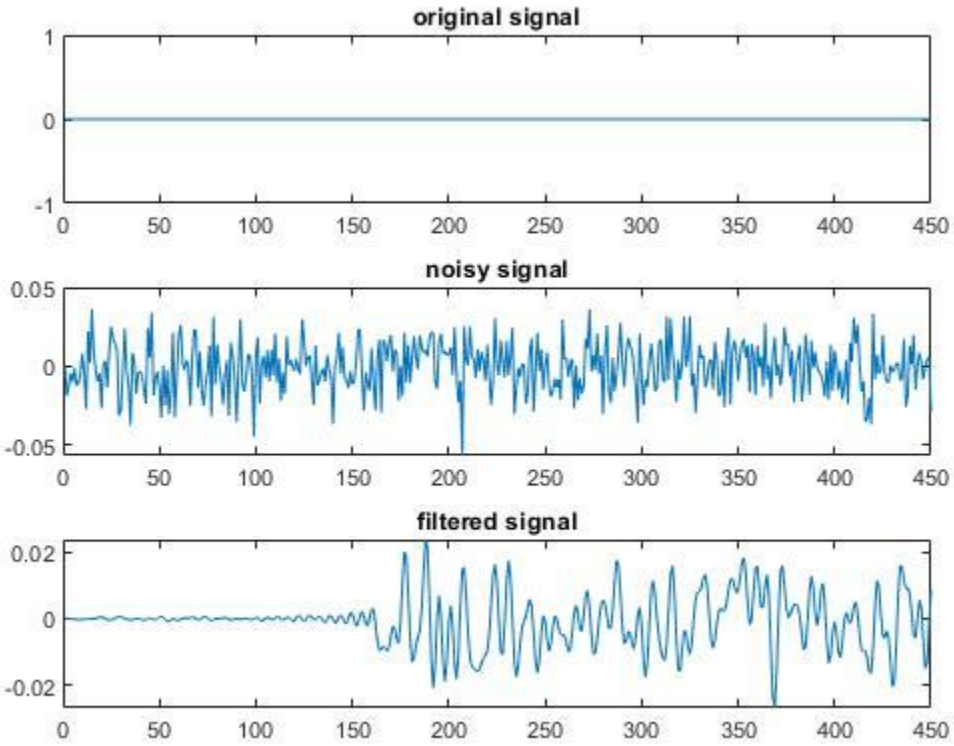
Output audio:

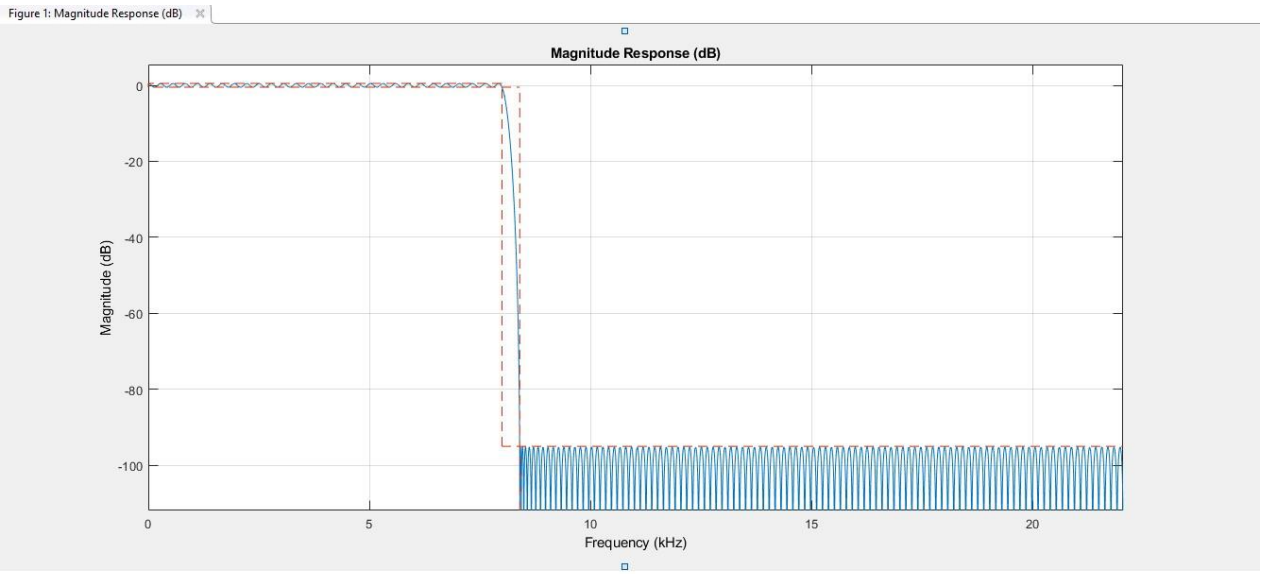
Noisy signal:

Filtered signal:

filtered_signal.wav

Outputs:





Problem No: 14
Problem Title: Write a MATLAB program to implement Echo of audio signal.

A frequently occurring distortion in speech telecommunication scenarios is echoes, which have either an electric or acoustic cause. Electric echoes appear in analogue networks at points of impedance mismatch. Since a majority of telecommunication networks today are digital, such electric echoes are mostly of historical interest and not discussed here further. Acoustic echoes however is a problem which has become more important, especially with the increasing use of teleconferencing services. Such acoustic echoes appear when the speech of a person A is played through the loudspeakers for a person B, such that the loudspeaker sound is picked up by the microphone of person B and transmitted back to person A. If the delay would be mere milliseconds, then person A would perceive the feedback signal as reverberation, which is *not* too disturbing (in some circumstances such an effect, known as *side-talk*, could actually be a useful feature indicating that the microphone is recording). However, typically the transmission delay is above 100ms, such that the feedback is perceived as an echo, which is usually highly disconcerting and disturbing. Even worse, sometimes the acoustic echo signal completes the feedback loop and goes in a circle, again and again. If the feedback signal is attenuated on each loop, then the feedback slowly diminishes, but sometimes the feedback loop can amplify the signal

such that the signal quickly escalates up to the physical limit of hardware or until the loudspeakers blow up. It is thus clear that echoes must be avoided in real-life systems.

Matlab source code:

```
clc;
clear all;
close all;
[filename, pathname]=uigetfile(".", 'select
the input audio');
[x, Fs]=audioread(num2str(filename));
n=length(x); %get the length of the audio
file
a=0.8; %attenuation factor
d=2000; %delay
y=zeros(n+d, 1); %initialize the output
audio file
xn=padarray(x, [d,0], 0, 'Pre');

for i=(d+1):1:n
y(i-d, 1)=x(i)+a*xn(i-d);
end

audiowrite('echo_signal.wav',y,Fs);
```

Input and Output:

Input audio:

sample_audio.wav

Output Audio:

echo_signal.wav

Problem No: 15
Problem Title: Write a MATLAB program to record and save single and double channel audio.

There are many times where I need to convert a multichannel clip into separate mono clips to simplify a mix. For example:

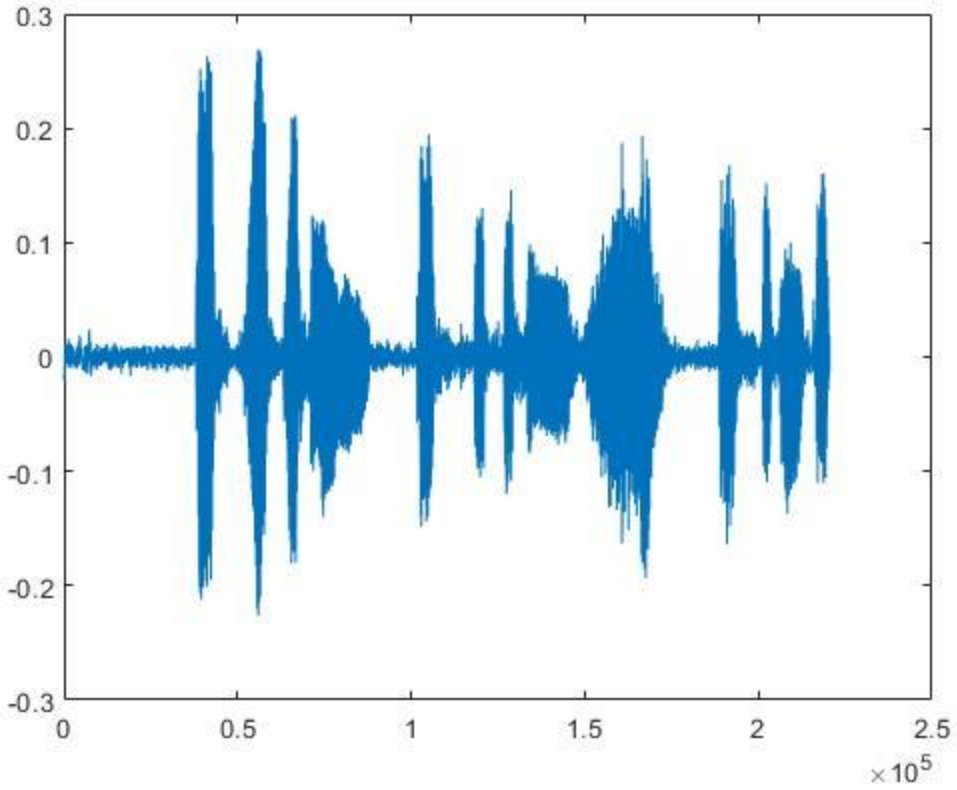
- I recorded dual-channel (or more) mono audio in the field and need to separate the interviewer from the guest into two separate clips.
- The master clip is stereo while the session is mono
- There is phase cancellation between the two channels
- I want to avoid the audio level differences between channels in a stereo and mono clip

Matlab source code:

```
clc;
clear all;
close all;
Fs=44100;
noc=1;
nob=16;
recObj=audiorecorder(Fs,nob,noc);

recordblocking(recObj,5);
play(recObj);
myRecording=getaudiodata(recObj);
plot(myRecording);

sound(myRecording,Fs);
audiowrite('Recorded_audio.wav',myRecording
,Fs);
```

Input and Output:

Recorded_audio.wa
v



Problem No: 16
Problem Title: Write a MATLAB program to implement Text to Speech signal.

Although speech-to-text conversion (STT) machines aim at providing benefits for the deaf or people who can't speak, it is difficult to review, retrieve and reuse speech transcripts. Hence, when the speech to text conversion module is combined with the summarization, the applications further increase in educational fields as well. This chapter discussed the need for speech summarization, various issues in the summarization of a spoken document, supervised, and unsupervised summarization algorithms. Isolated Tamil speech recognition was performed using a sample set of Tamil spoken words. In addition, state-of-the-art recognition techniques were used, and analysis was performed. Also, the summarization of speech data in Tamil language is explored, along with related work on text summarization. The features used in the summarization of a spoken document are analyzed and compared, based on the various forms of input into the spoken document.

Matlab source code:

```
clc;
clear all;
close all;
NET.addAssembly('System.speech');
mySpeaker=System.Speech.Synthesis.SpeechSyn
thesizer;

mySpeaker.Rate=0;
mySpeaker.Volume=100;

Speak(mySpeaker, 'this is digital Image and
Speech Processing');
```

Input and Output:

The programme run successfully and speaks the sentence.