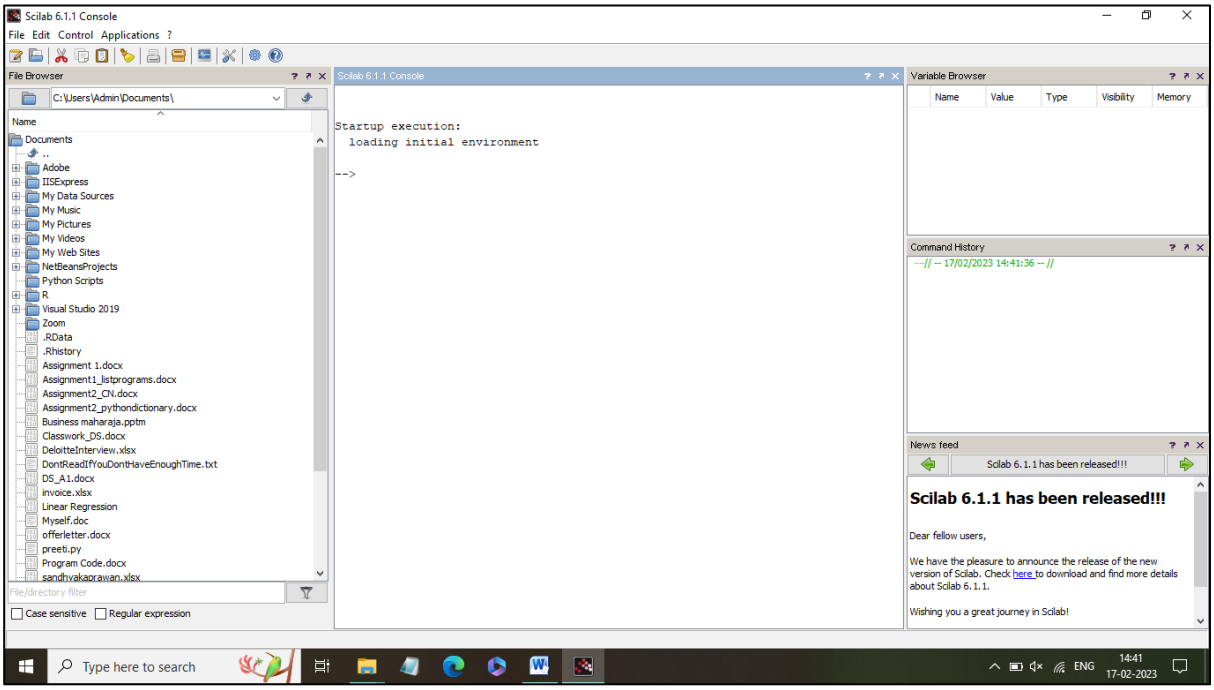
INDEX

	b. Program for Image deblurring using inverse, Weiner filters	28.03.23		
5.	**Colour Image Processing**		31	
	a. Program to read a color image and segment into RGB planes, histogram of color Image.	11.04.23		
	b. Program for Converting from one color model to another model.	11.04.23		
	c. Program to apply false colouring(pseudo) on a gray scale image.	11.04.23		
6.	**Fourier Related Transforms**		33	
	a. Program to compute Discrete Cosine Transform	18.04.23		
7.	**Image Compression**		34	
	a. Program to apply compression and decompression algorithm on an image using Huffman coding technique.	18.04.23		
8.	**Morphological Image Processing**		39	
	a. Program to apply erosion, dilation, opening, closing	18.04.23		
	b. Program for detecting boundray of an Image.	18.04.23		
	c. Program to apply Hit-or-Miss transform	25.04.23		
	d. Program to apply morphological gradient on an image	25.04.23		
	e. Program to apply Top-Hat/Bottom-Hat Transformations	25.04.23		
9.	**Image Segmentation**		43	
	a. Program for Edge detection using. i. Sobel ii. Prewitt iii. Canny iv. Marr-Hildreth	02.05.23		
	b. Illustrate Watershed segmentation algorithm	02.05.23		
10.	**Feature Extraction**		47	
	a. Apply Principal components for image description.	02.05.23		
	b. Apply Harris-Stephen’s corner detector algorithm.	02.05.23		

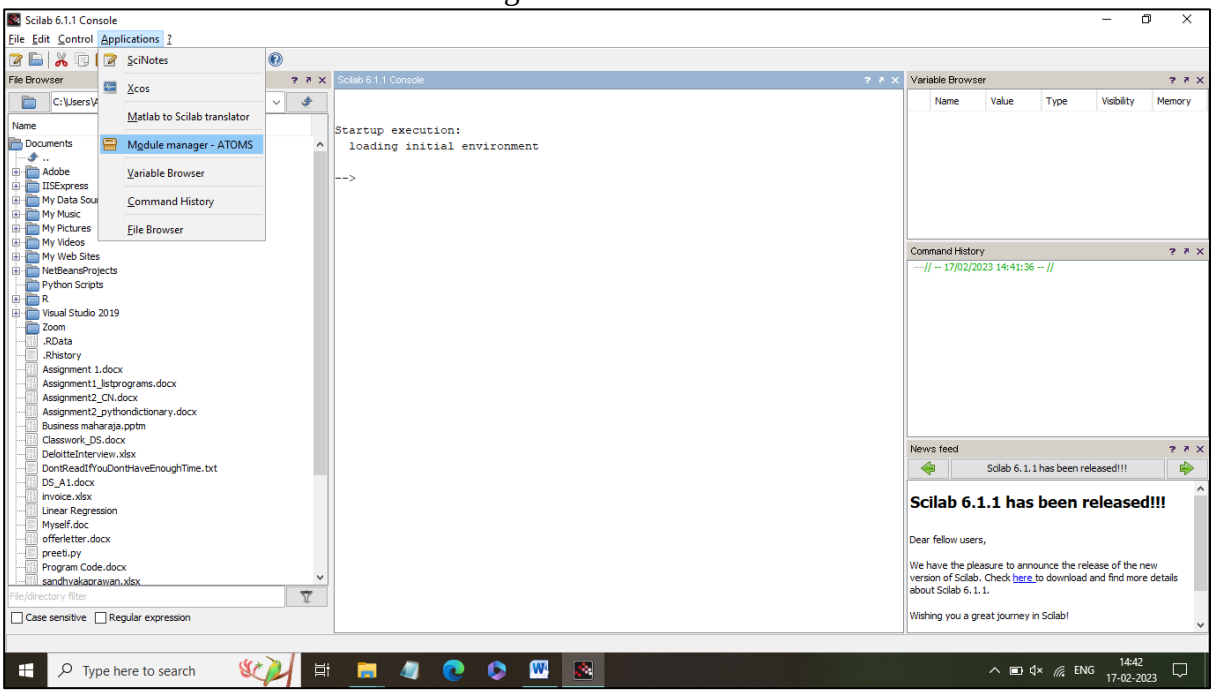
Practical 1

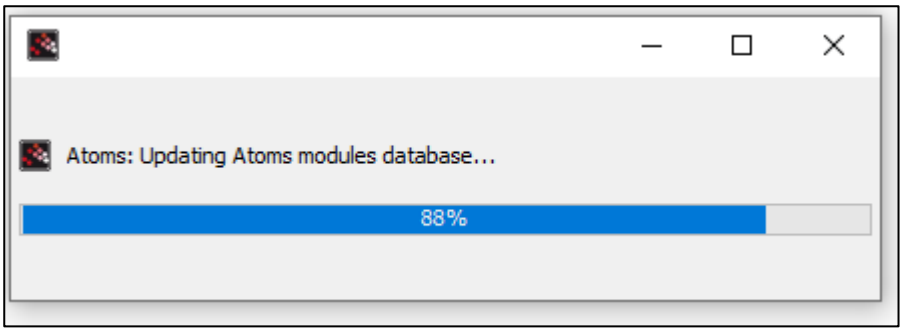
A. Program to calculate number of samples required for an image.

1. Install Scilab 6.1.1, open scilab console

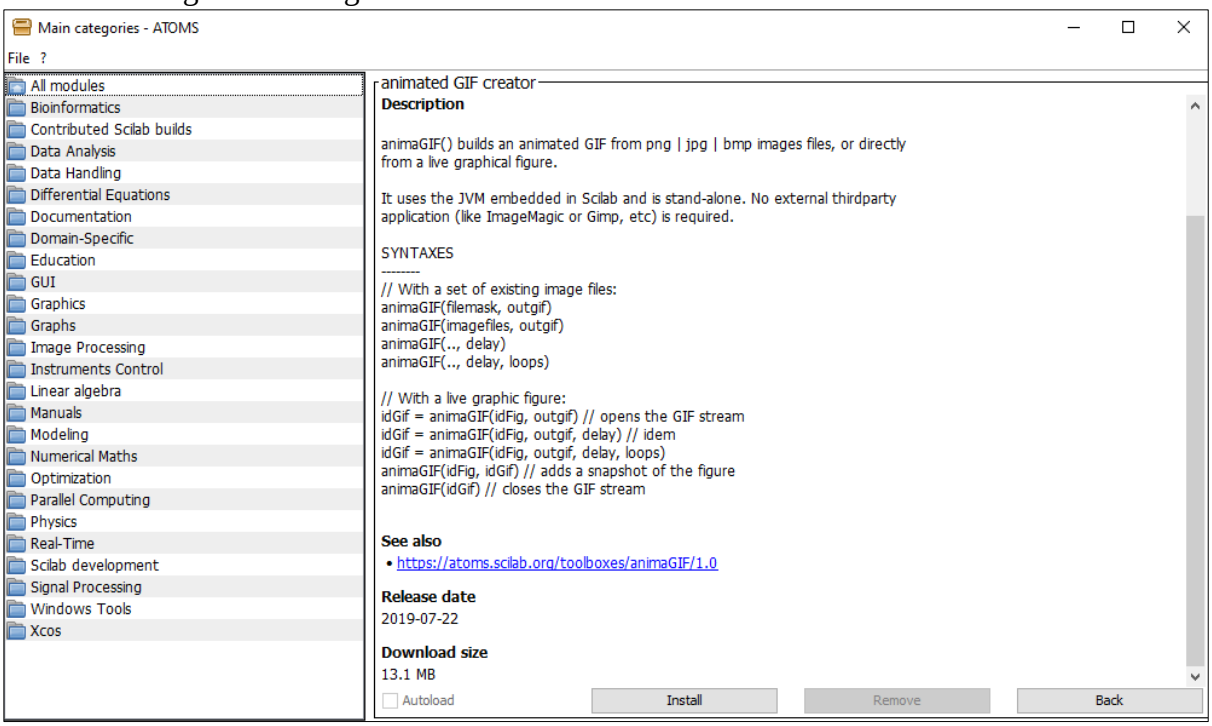


2. We require Modules for performing Digital Image processing Practical. Click on Applications from toolbar then click on Module manager – ATOMS.

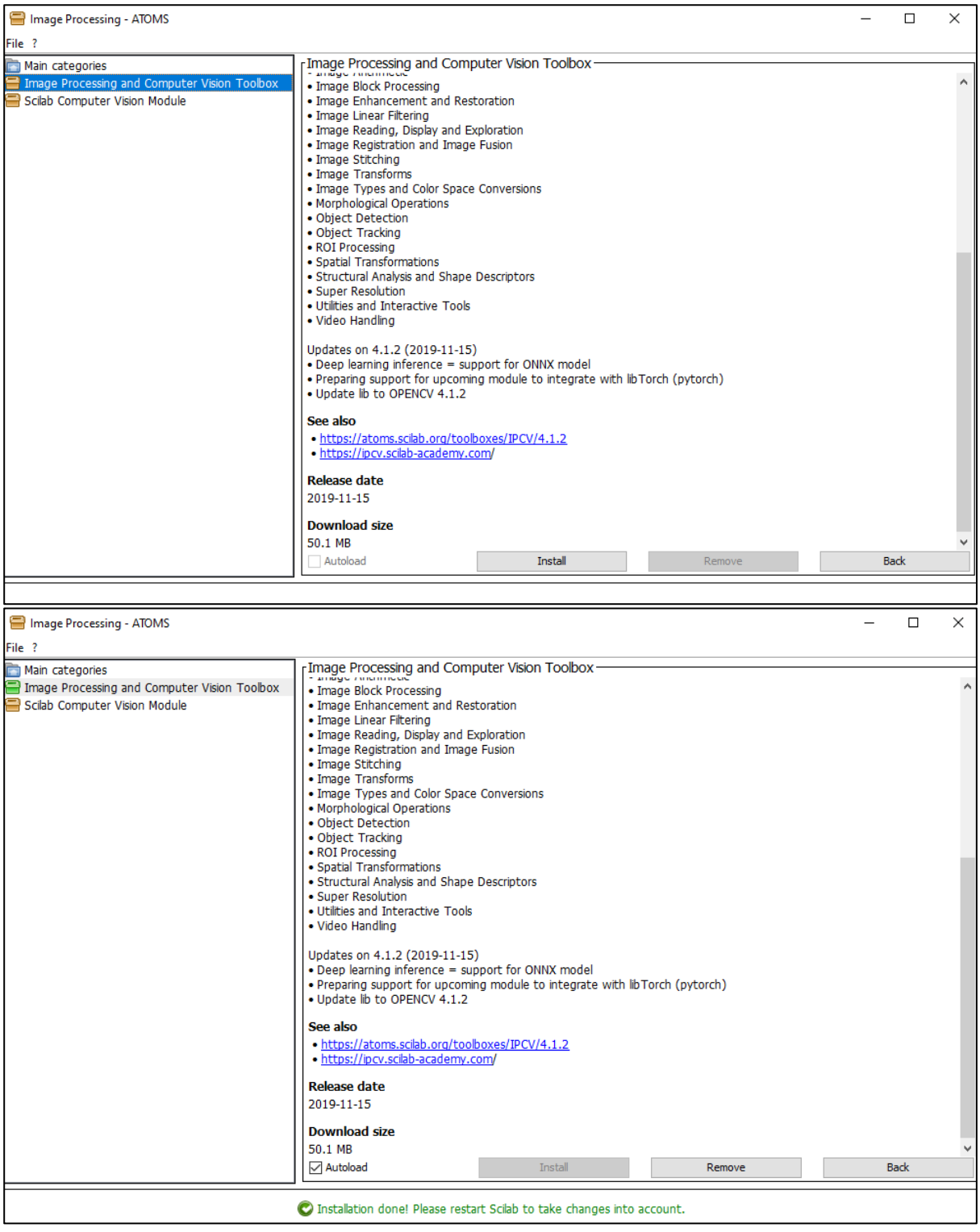




3. Click on Image Processing from list of Modules.



4. Install Image Processing and Computer Vision Toolbox and after installation restart the scilab so, the changes take place in ATOMS Module.

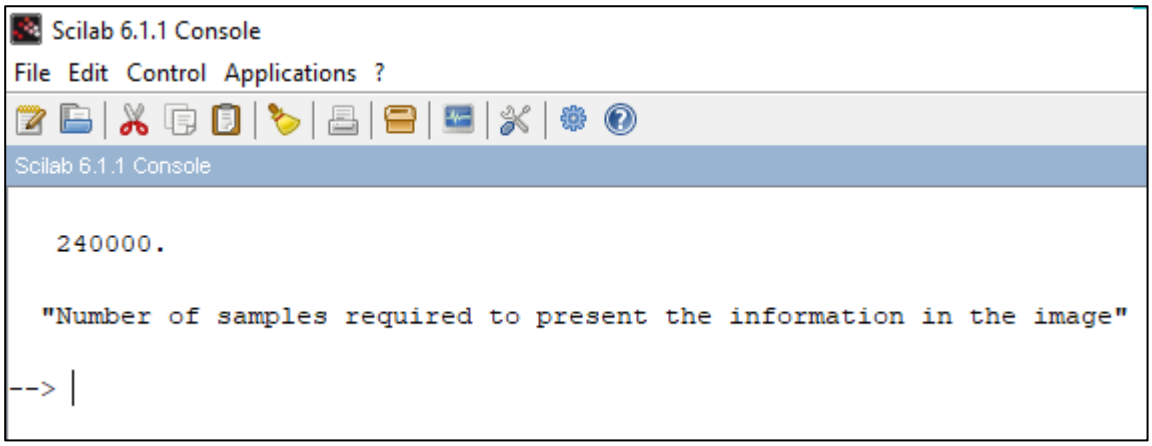


5. Write the code for calculating number of samples required for image

Code:

```
clc;
close;
//dimension of image in inches
m=6;
n=4;
N=100;
N2=N*N
Fs=m*n*N2
disp(Fs,'Number of samples required to present the information in the image')
```

Output:



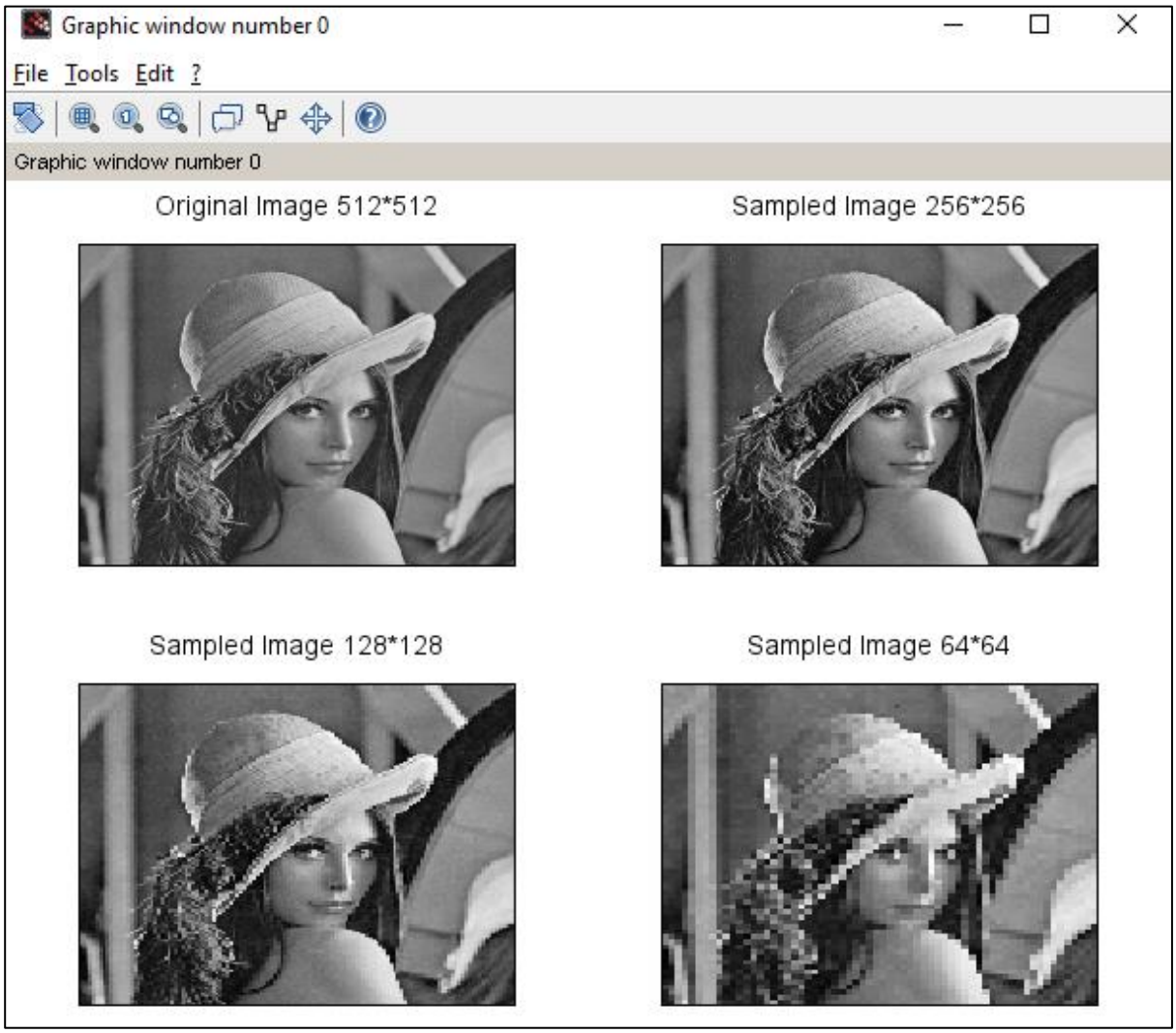
B. Program to study the effects of reducing the spatial resolution of a digital image.

Code:

```
clc;
clear all;
Img1=imread('D:\ IP_Practical_Images\lena.jpeg');
Img = rgb2gray(Img1);
//512*512
subplot (2,2,1),imshow(Img),title('Original Image 512*512');
//256*256
Samp=zeros(256);
m=1;
n=1;
for i=1:2:512
    for j=1:2:512
         Samp(m,n)=Img(i,j);
           n=n+1;
        end
        n=1;
        m=m+1;
end
SampImg256=mat2gray(Samp);
subplot(2,2,2);
imshow(SampImg256);
title('Sampled Image 256*256')
Samp=zeros(128);
m=1;
n=1;
for i=1:4:512
    for j=1:4:512
         Samp(m,n)=Img(i,j);
           n=n+1;
        end
        n=1;
        m=m+1;
end
SampImg128=mat2gray(Samp);
subplot(2,2,3),imshow(SampImg128),title('Sampled Image 128*128')
Samp=zeros(64);
m=1;
n=1;
for i=1:8:512
    for j=1:8:512
         Samp(m,n)=Img(i,j);
           n=n+1;
        end
        n=1;
```

```
        m=m+1;
end
SampImg64=mat2gray(Samp);
subplot(2,2,4),imshow(SampImg64),title('Sampled Image 64*64')
```

Output:



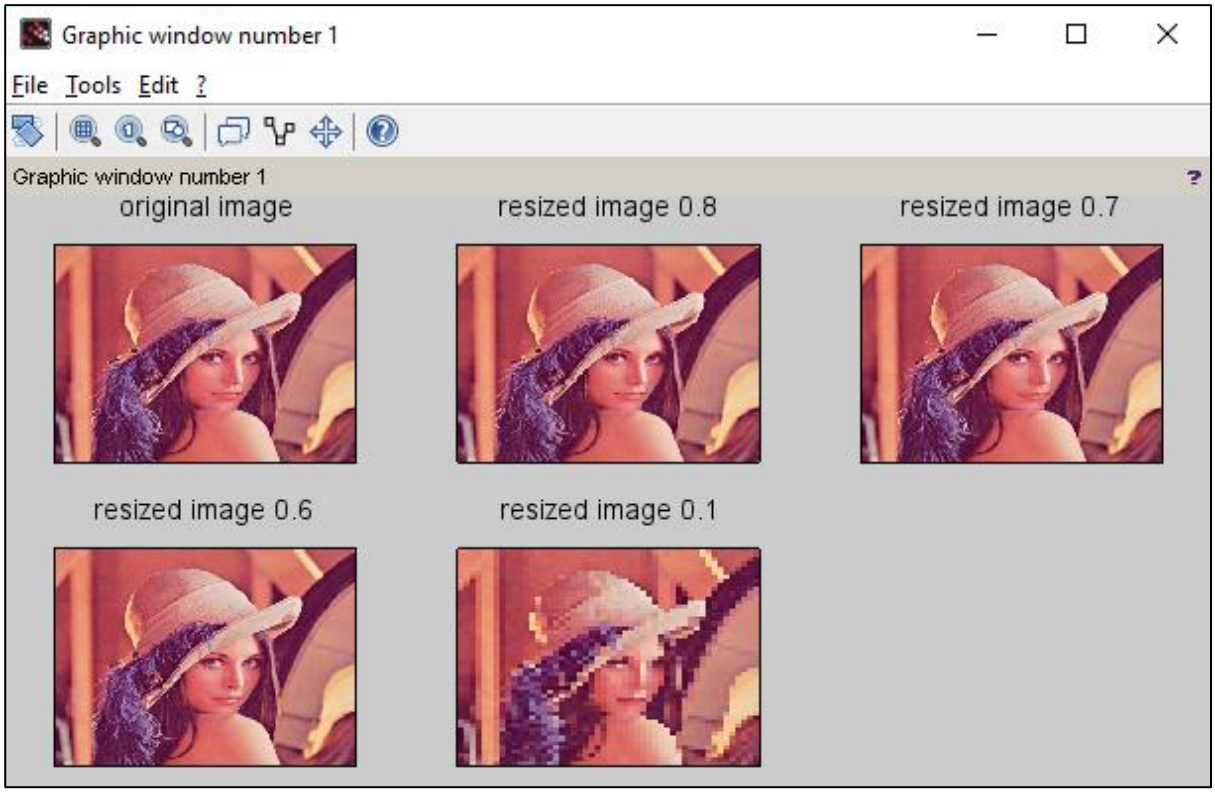
C. Program to study the effects of varying the number of intensity levels in a digital image.

Code:

```
clc;
clear all;
figure(1)

subplot(3,3,1);
i=imread('D:\MSC IT\Part I\Sem II\Image Processing\image-20230324T094412Z-
001\image\lena.jpeg');
imshow(i);
title('original image');
subplot(3,3,2);
j1=imresize(i,0.8);
imshow(j1);
title('resized image 0.8');

subplot(3,3,3);
j2=imresize(i,0.7);
imshow(j2);
```

```
title('resized image 0.7');

subplot(3,3,4);
j3=imresize(i,0.6);
imshow(j3);
title('resized image 0.6');

subplot(3,3,5);
j4=imresize(i,0.1);
imshow(j4);
title('resized image 0.1');
```

Output:



D. Program to perform image averaging (image addition) for noise reduction.

Code:

```
clc;
clear all;
i=imread("D:\MSC IT\Part I\Sem II\Image Processing\IP_Practical\flower.jpg");
subplot(2,1,1);
title('Original Image');
imshow(i);
b=imnoise(i,'salt & pepper');
subplot(2,1,2);
title('Salt and Pepper Noise Image');
imshow(b);
imwrite(b,'flowersalt&pepper.jpg')
```

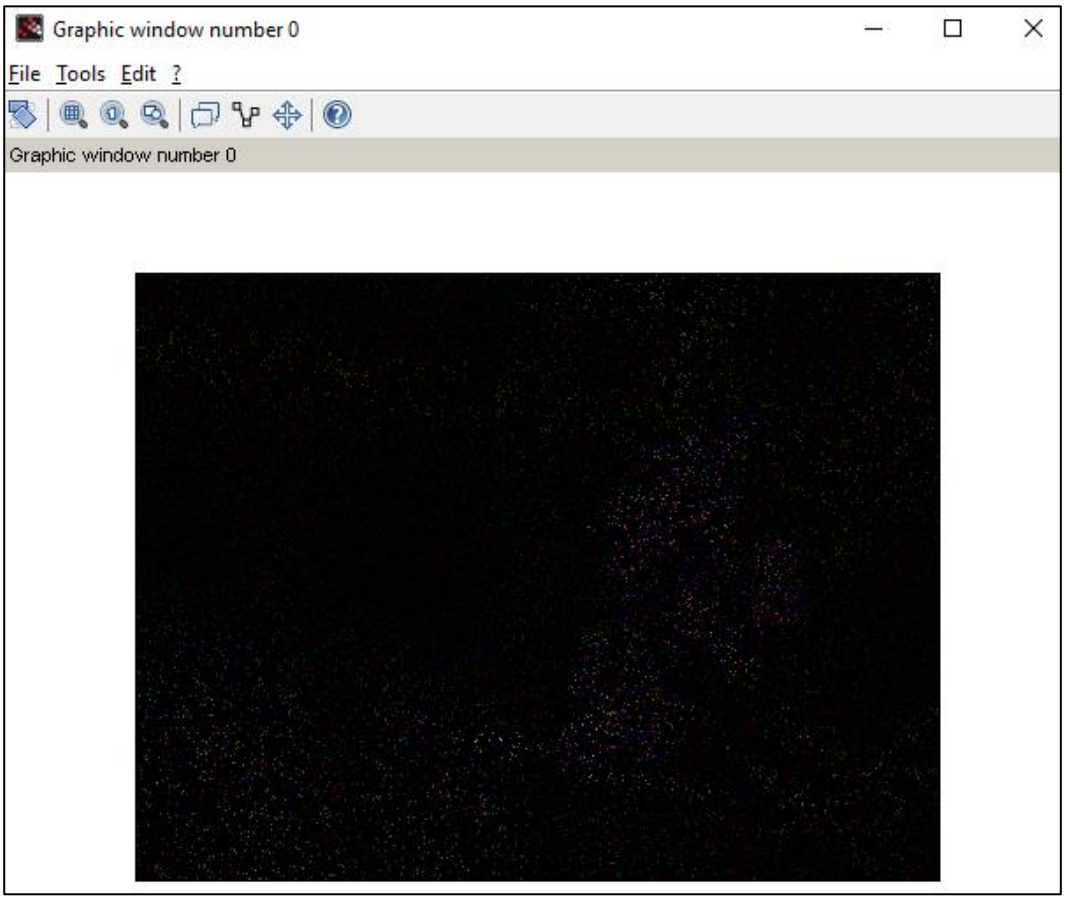
Output:

E. Program to compare images using subtraction for enhancing the difference between images.

Code:

```
clc;
a=imread('D:\MSC IT\Part I\Sem II\Image Processing\IP_Practical\flower.jpg');
b=imread('D:\MSC IT\Part I\Sem II\Image
Processing\IP_Practical\flowersalt&pepper.jpg');
c=imsubtract(a,b);
imshow(c);
```

Output:



Practical 2

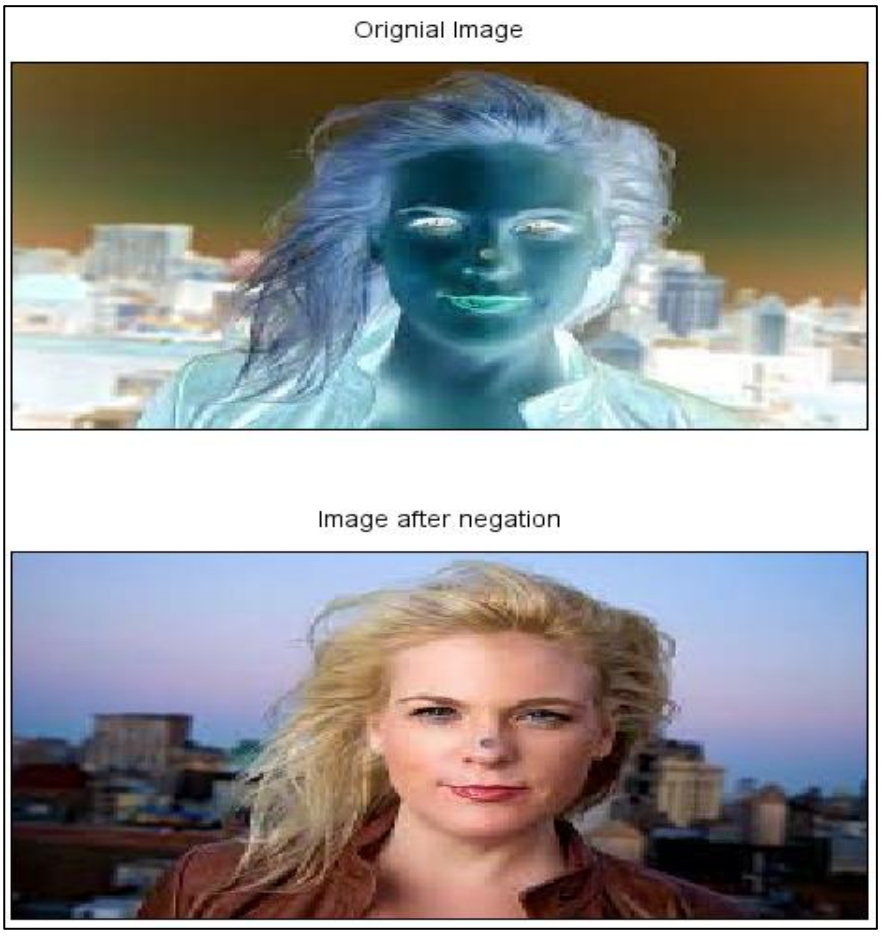
A. Basic Intensity Transformation functions

i. Program to perform Image negation.

Code:

```
clc;
clear all;
A = imread('D:\MSC IT\Part I\Sem II\Image Processing\IP_Practical\IP_Practical_Images\negimg.jpg');
subplot(2,1,1);
imshow(A);
title('Orignial Image');
R = A(:,:,1);
G = A(:,:,2);
B = A(:,:,3);
[row col]=size(A);
for x=1:row
    for y=1:col
       R(x,y)=255-R(x,y);
       G(x,y)=255-G(x,y);
       B(x,y)=255-B(x,y);
    end
end
A(:,:,1)=R;
A(:,:,2)=G;
A(:,:,3)=B;
subplot(2,1,2);
imshow(A);
title('Image after negation');
```

Output:

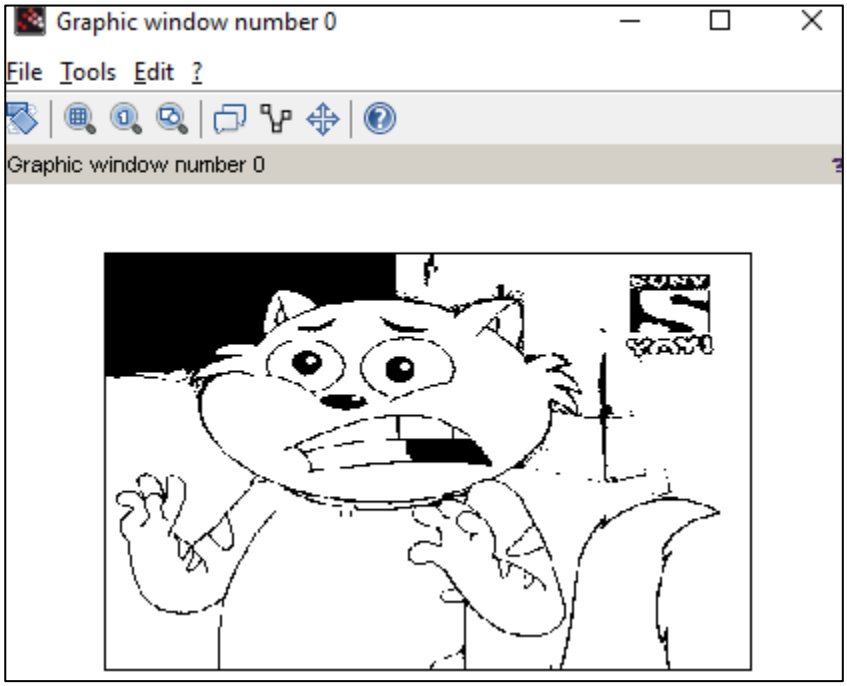


ii. Program to perform threshold on an image.

Code:

```
clc;
a=imread('D:\MSC IT\Part I\Sem II\Image Processing\IP_Practical\Honey.jpg');
b=double(a);
[m,n]=size(b);
T=100;
for i=1:m
    for j=1:n
        if(b(i,j)<T)
            c(i,j)=0;
        else
            c(i,j)=255;
        end
    end
end
imshow(uint8(c));
```

Output:

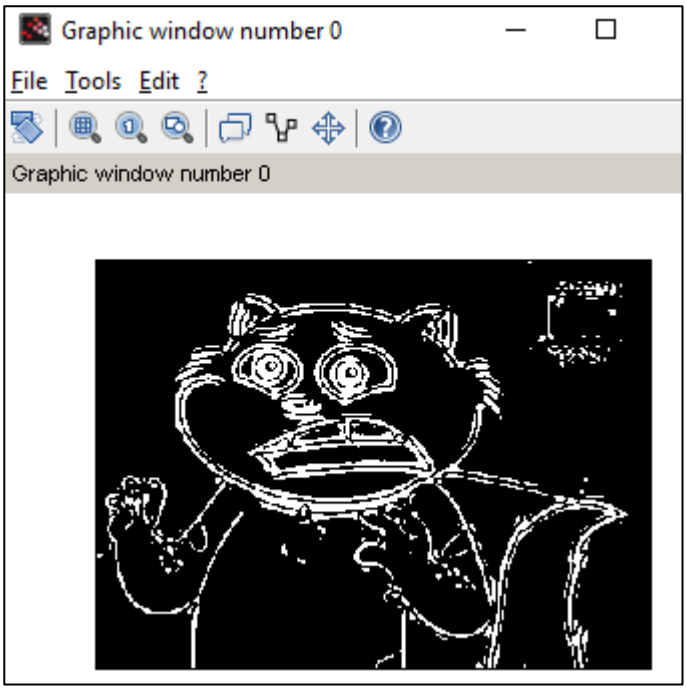


iii. Program to perform Log transformation.

Code:

```
//LOG
a=imread('D:\MSC IT\Part I\Sem II\Image Processing\IP_Practical\Honey.jpg');
b=rgb2gray(a);
//Log operator
c=edge(b,'log');
imshow(c)
```

Output:



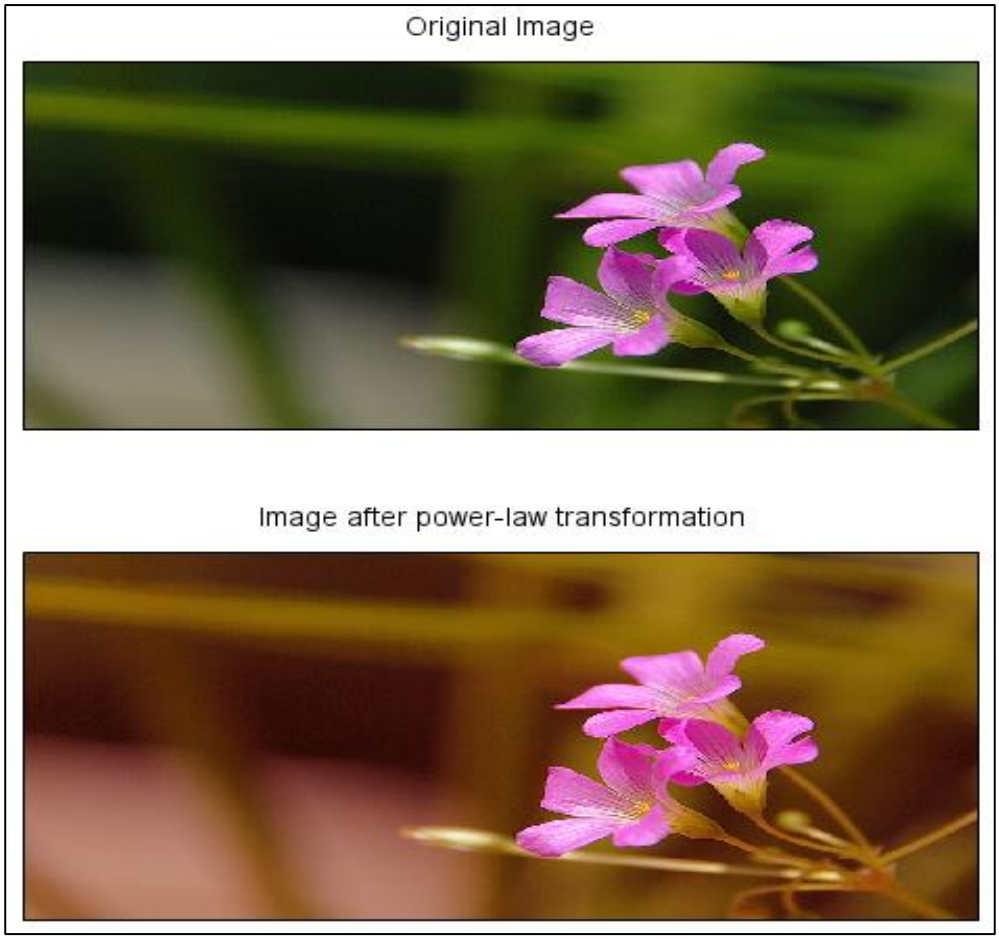
iv. Power-law transformations

Code:

```
//Power Law transformation
clear all;
clc;
close all;
i=imread('D:\MSC              IT\Part              I\Sem              II\Image
Processing\IP_Practical\IP_Practical_Images\flower.jpg');
subplot(2,1,1);
imshow(i);
title('Original Image');
i=im2double(i);
c=1;
[row col]=size(i);
for x=1:row
    for y=1:col
        i(x,y)=c*i(x,y)^0.5; //1.5
    end
end
i=im2uint8(i);
subplot(2,1,2);
imshow(i);
title('Image after power-law transformation');
```

Output:

0.5


Original Image
Image after power-law transformation

1.5


Original Image
Image after power-law transformation

v. Piecewise linear transformations

a. Contrast Stretching

Code:

```
clc
a = imread ("D:\MSC IT\Part I\Sem II\Image Processing\IP_Practical\IP_Practical_Images\lena.png");
a = rgb2gray ( a ) ;
b = double ( a ) *0.5;
b = uint8 ( b );
c = double ( b ) *2;
c = uint8 ( c );
subplot(1,3,1)
imshow(a) ;
title ( "Original Image " )
subplot(1,3,2)
imshow(b) ;
title ( "Decrease in Contrast" )
subplot(1,3,3)
imshow(c) ;
title ( "Increase in Contrast")
```

Output:

b. Gray-level slicing with and without background

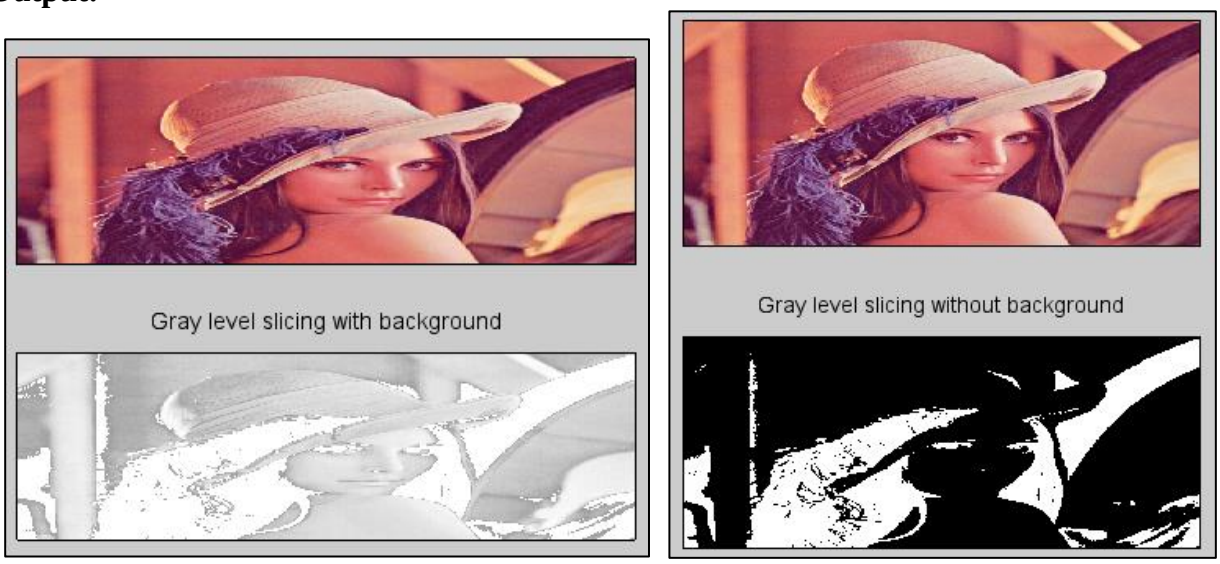
Code with background:

```
clc;
clear all;
a=imread('D:\MSC IT\Part I\Sem II\Image Processing\IP_Practical\IP_Practical_Images\lena.png');
a1=58; // This value is user defined
b1=158; // This value is user defined
[r,c]=size(a);
figure(2);
subplot(2,1,1);
imshow(a);
for i=1:r
    for j=1:c
        if (a(i,j)>a1 & a(i,j)<b1)
            x(i,j)=255;
        else
            x(i,j)=a(i,j);
        end
    end
end
x=uint8(x);
subplot(2,1,2);
title('Gray level slicing with background')
imshow(x);
```

Code without background:

```
clc;
clear all;
a=imread('D:\MSC              IT\Part              I\Sem              II\Image
Processing\IP_Practical\IP_Practical_Images\lena.png');

a1=50; // This value is user defined
b1=150; // This value is user defined
[r,c]=size(a);
figure(1)
subplot(2,1,1);
imshow(a);
for i=1:r
    for j=1:c
        if (a(i,j)>a1 & a(i,j)<b1)
            x(i,j)=255;
        else
            x(i,j)=0;
        end
    end
end
x=uint8(x);
subplot(2,1,2);
title('Gray level slicing without background');
imshow(x);
```

Output:



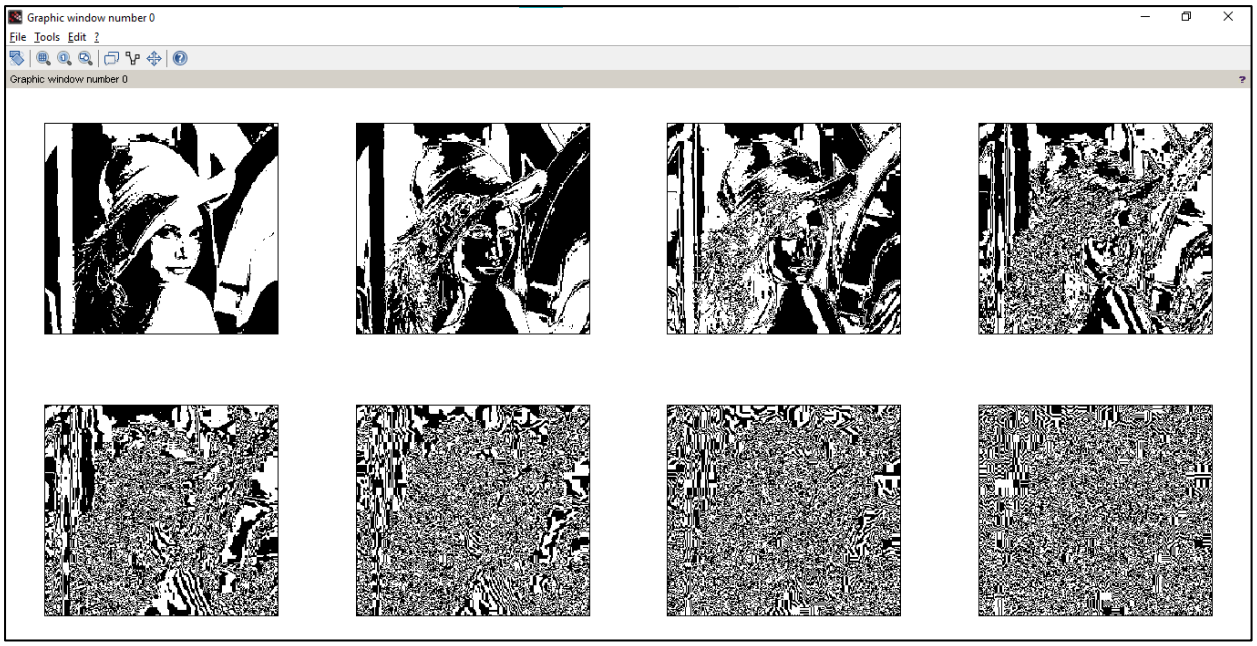
c. Bit-plane slicing

Code:

```
clc;
clear all;
f=imread('D:\MSC IT\Part I\Sem II\Image
Processing\IP_Practical\IP_Practical_Images\lenag.jpeg');
f=double(f);
[r,c]=size(f);
com=[128 64 32 16 8 4 2 1];

for k=1:1:length(com);
    for i=1:r
        for j=1:c
        new(i,j)=bitand(f(i,j),com(k));
    end
    subplot(2,4,k);
    imshow(new);
```

```
    end
end
```

Output:

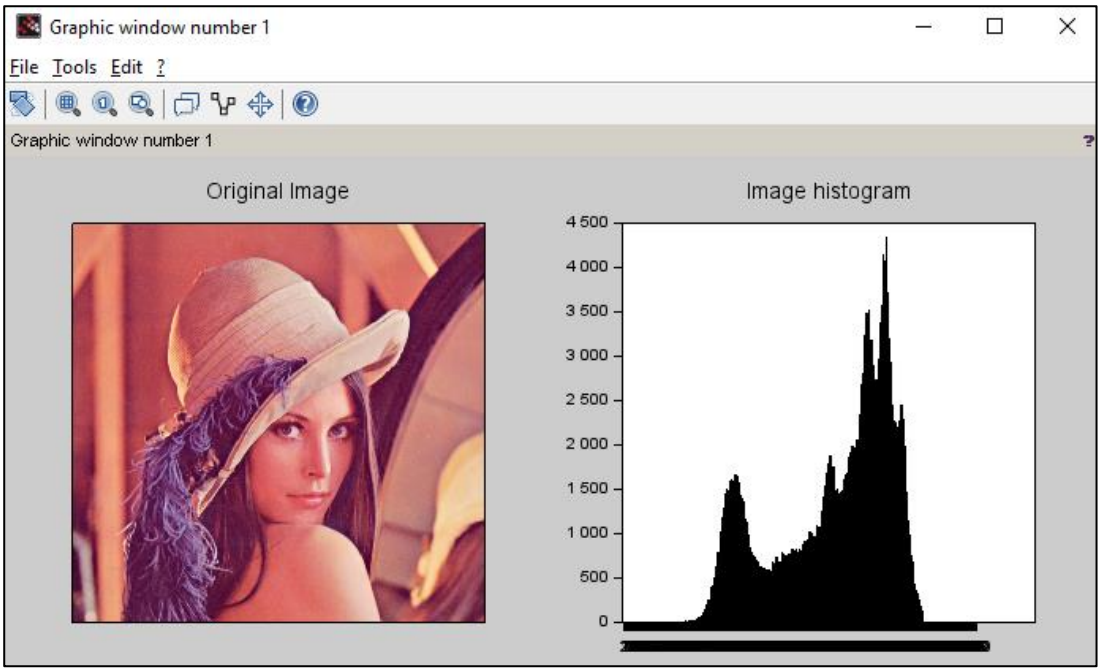


B 1. Program to plot the histogram of an image and categorise.

Code:

```
clear all;
clc;
a=imread('D:\MSC              IT\Part              I\Sem              II\Image
Processing\IP_Practical\IP_Practical_Images\lena.png');
a=double(a);
[row col]=size(a);
h=zeros(1,300);
for n=1:1:row
for m=1:1:col
if a(n,m)==0
a(n,m)=1;
end
end
end
for n=1:1:row
for m=1:1:col
t=a(n,m);
h(t)=h(t)+1;
end
end
figure(1);
subplot(1,2,1);
imshow(uint8(a));
title('Original Image')
subplot(1,2,2);
bar(h);
title('Image histogram');
```

Output:



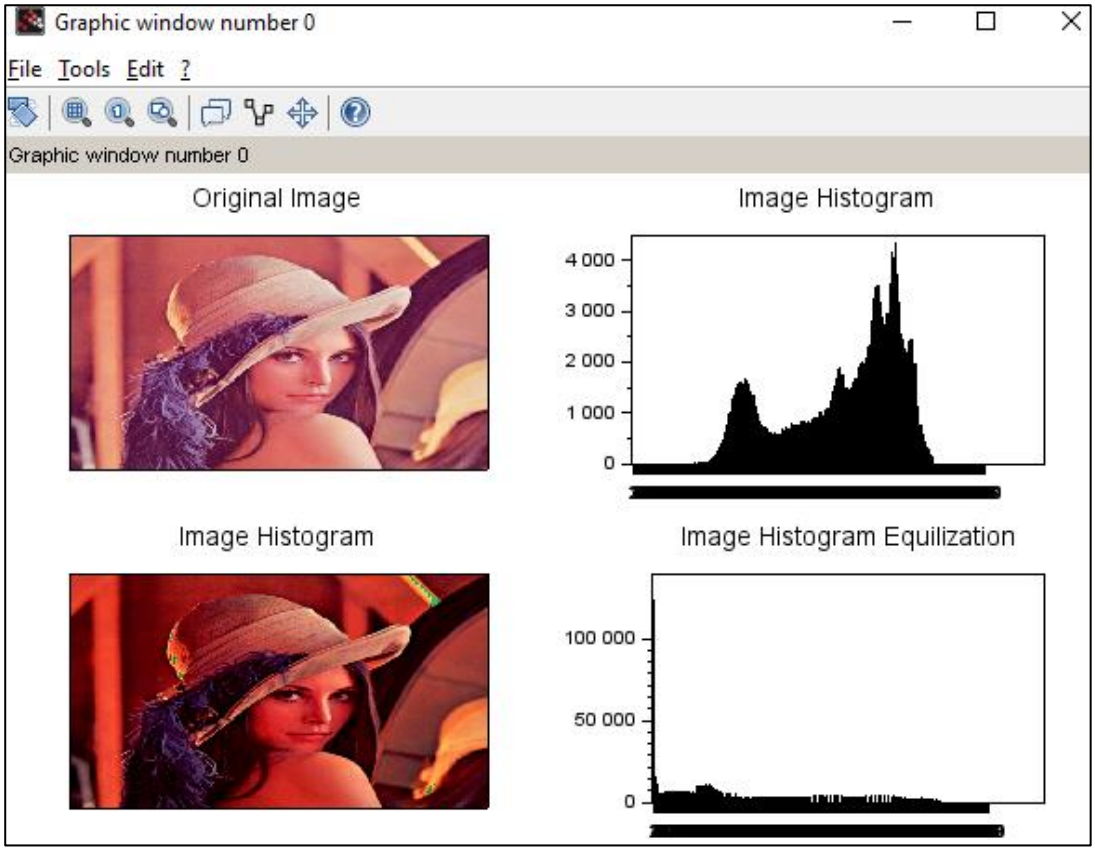
B 2. Program to apply histogram equalization.

Code:

```
clear all;
clc;
a          =imread('D:\MSC          IT\Part          I\Sem          II\Image
Processing\IP_Practical\IP_Practical_Images\lena.png');
a=double(a);
big=256;
[row col d]=size(a);
c=row*col;
h=zeros(1,300);
z=zeros(1,300);
for e=1:1:d
for n=1:1:row
for m=1:1:col
if a(n,m,e)==0
a(n,m,e)=1;
end
end
end
end
for n=1:1:row
for m=1:1:col
t=a(n,m);
h(t)=h(t)+1;
end
end
pdf=h/c;
cdf(1)=pdf(1);
for x=2:1:big
cdf(x)=pdf(x)+cdf(x-1);
end
new=round(cdf*big);
new=new+1;
for r=1:1:d
for p=1:1:row
for q=1:1:col
temp=a(p,q,r);
b(p,q,r)=new(temp);
```

```
t=b(p,q,r);
z(t)=z(t)+1;
       end
    end
end
b=b-1;
subplot(2,2,1);
imshow(uint8(a));
title('Original Image');
subplot(2,2,2);
bar(h);
title('Image Histogram');
subplot(2,2,3);
imshow(uint8(b));
title('Image Histogram');
subplot(2,2,4);
bar(z);
title('Image Histogram Equilization');
```

Output:



C. Write a program to perform convolution and correlation.

Code:

```
//Caption : Linear convolution of any signal with an impulse signal gives
rise to the same signal
clc;
x=[1,2,3;4,5,6;7,8,9];
h=[1,1;1,1;1,1];
y=conv2(x,h);
disp("Linear 2D Convolution result y=",y)
//Caption: Linear cross correlation of a 2D matrix
clc;
x=[3,1;2,4];
h1 = [1,5;2,3];
h2=h1(:,$:-1:1);
h = h2($ :-1:1,:);
```

```
y = conv2(x,h)
disp("Linear cross correlation result y=",y)
```

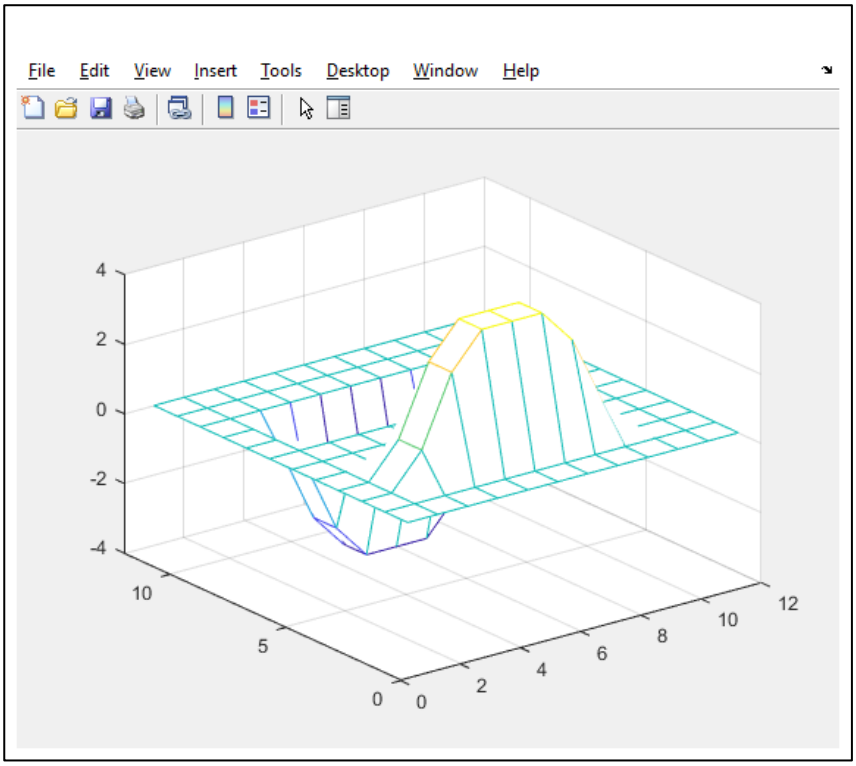
2D-Convolution:

```
%conv2 % MATLAB
%Two-dimensional convolution
s = [1 2 1; 0 0 0; -1 -2 -1];
A = zeros(10);
A(3:7,3:7) = ones(5);
H = conv2(A,s);
mesh(H)
```

Output:

```
"Linear 2D Convolution result y="

 1.    3.    5.    3.
 5.    12.   16.   9.
 12.   27.   33.   18.
 11.   24.   28.   15.
 7.    15.   17.   9.
```

```
"Linear cross correlation result y="

 9.    9.    2.
 21.   24.   9.
 10.   22.   4.
```



D. Write a program to apply smoothing and sharpening filters on grayscale and colour images.

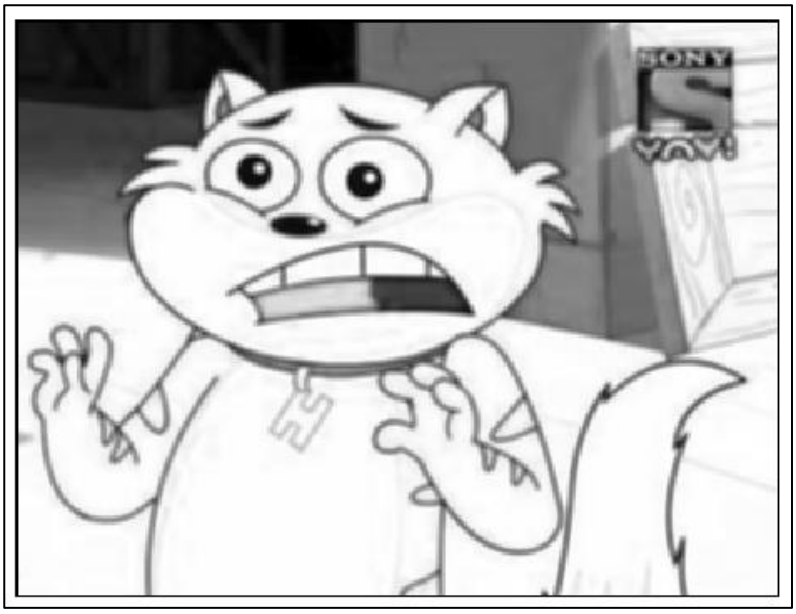
a. Low Pass

Code:

```
a1=imread("D:\MSC IT\Part I\Sem II\Image Processing\IP_Practical\Honey.jpg");
a=double(a1);
[m,n]=size(a);
w=[1 1 1;1 1 1;1 1 1];
for i=2:m-1
    for j=2:n-1
        b(i,j)=[w(1)*a(i-1,j+1)+w(2)*a(i,j+1)+w(3)*a(i+1,j+1)+w(4)*a(i-
1,j)+w(5)*a(i,j)+w(6)*a(i+1,j)+w(7)*a(i-1,j-1)+w(8)*a(i,j-1)+w(9)*a(i+1,j-
1)]/9
    end
```

```
end
c=uint8(b)
imshow(c);
```

Output:

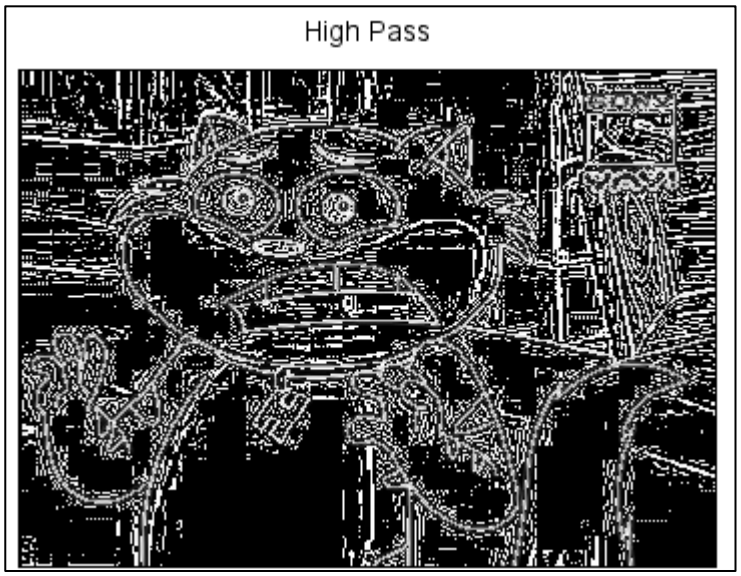


b. High Pass

Code:

```
clc;
a1=imread("D:\MSC IT\Part I\Sem II\Image Processing\IP_Practical\honey.jpg");
a=double(a1);
[m,n]=size(a);
w=[-1 -1 -1;-1 8 -1;-1 -1 -1];
for i=2:m-1
    for j=2:n-1
        b(i,j)=[w(1)*a(i-1,j+1)+w(2)*a(i,j+1)+w(3)*a(i+1,j+1)+w(4)*a(i-
1,j)+w(5)*a(i,j)+w(6)*a(i+1,j)+w(7)*a(i-1,j-1)+w(8)*a(i,j-1)+w(9)*a(i+1,j-
1)]/9
    end
end
c=uint8(b)
title('High Pass')
imshow(c);
```

Output:



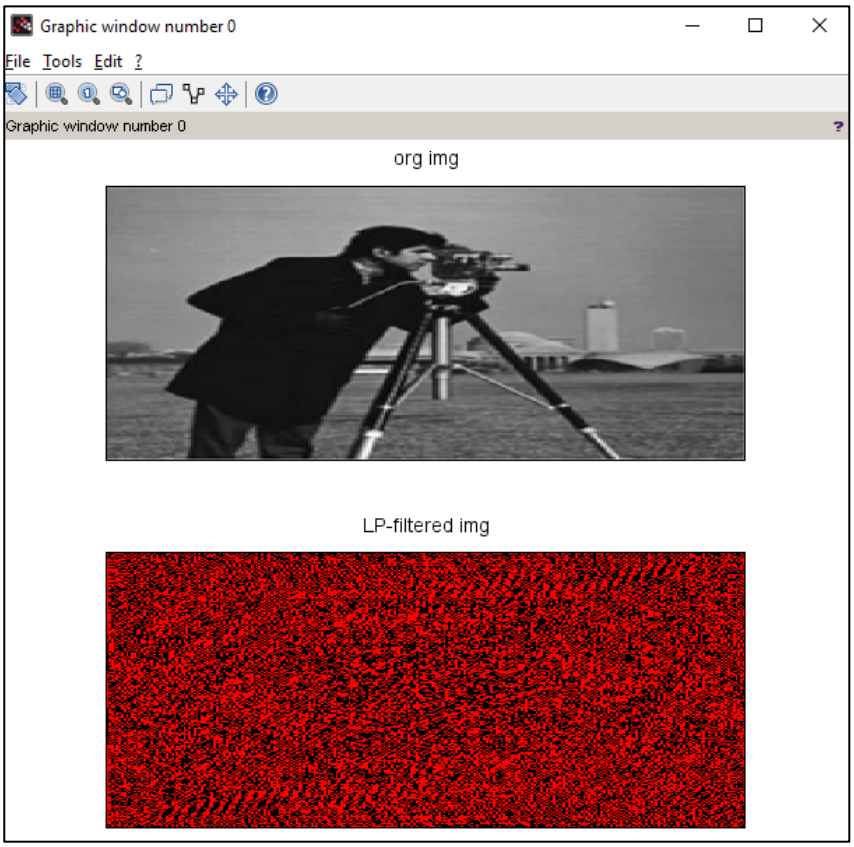
Practical 3

A. Program to apply Discrete Fourier Transform on an image.

Code:

```
//3A-DFT
a=imread('D:\MSC                IT\Part                I\Sem                II\Image
Processing\IP_Practical\IP_Practical_Images\camera.png');
subplot(2,1,1);
imshow(a);
title('org img');
b=double(a);
c=fft(b);
subplot(2,1,2);
imshow(c);
title('LP-filtered img');
```

Output:



B. Program to apply Low pass and High pass filters in frequency domain.

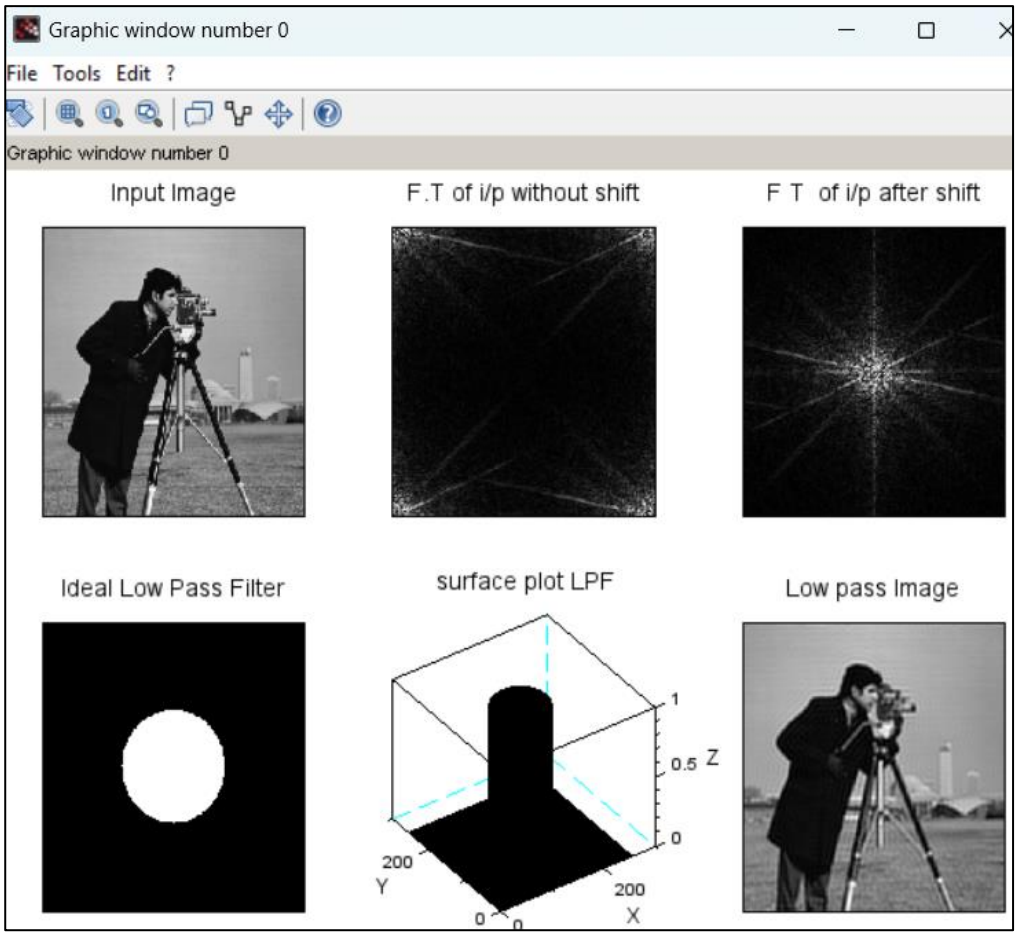
i. Ideal Low Pass filter

Code:

```
clc;
clear;
a=imread('C:\Users\sandhya\Desktop\3\cameraman.tif')
a=im2double(a);
subplot(2,3,1)
imshow(a)
title('Input Image')
[m,n]=size(a);
D0 =50;
A=fft2(a);
subplot(2,3,2)
imshow(uint8(abs(A)))
title('F.T of i/p without shift');
A_shift=fftshift(A);
A_real=abs(A_shift);
```

```
subplot(2,3,3)
imshow(uint8(A_real))
title('F T  of i/p after shift');
A_low=zeros(m,n);
d=zeros(m,n);
for u=1:m
    for v=1:n
        d(u,v)=sqrt((u-(m/2))^2+(v-(n/2))^2);
        if d(u,v)<=D0
            A_low(u,v)=A_shift(u,v);
            filt(u,v)=1;
        else
            A_low(u,v)=0;
            filt(u,v)=0;
        end
    end
end
subplot(2,3,4),imshow(filt),title('Ideal Low Pass Filter')
subplot(2,3,5),mesh(filt),title('surface plot LPF')
B=fftshift(A_low);
B_inverse=ifft(B);
B_real=abs(B_inverse);
subplot(2,3,6),imshow(B_real),title('Low pass Image')
```

Output:



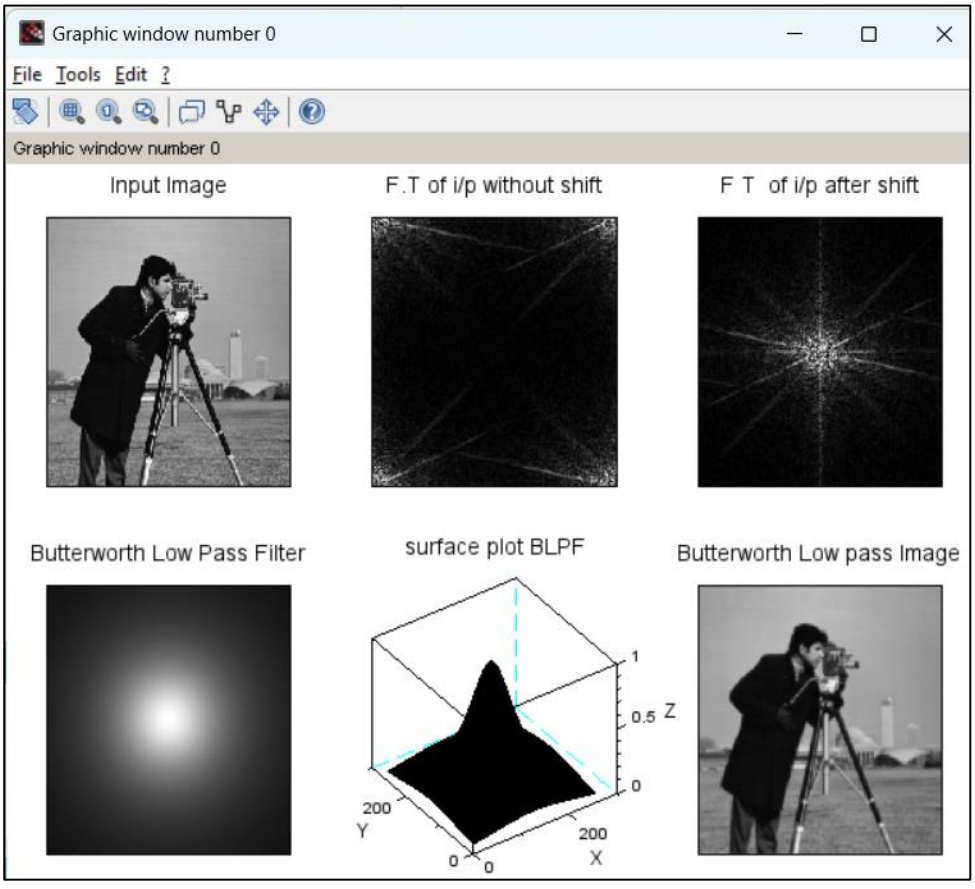
ii. Butterworth Low Pass filter

Code:

```
clc;
clear;
a=imread('C:\Users\sandhya\Desktop\3\cameraman.tif')
a=im2double(a);
subplot(2,3,1)
imshow(a)
title('Input Image')
```

```
[m,n]=size(a);
A=fft2(a);
subplot(2,3,2)
imshow(uint8(abs(A)))
title('F.T of i/p without shift');
A_shift=fftshift(A);
A_real=abs(A_shift);
subplot(2,3,3)
imshow(uint8(A_real))
title('F T  of i/p after shift');
D0=50;
d=zeros(m,n);
order=1;
for u=1:m
    for v=1:n
        d(u,v)=sqrt((u-(m/2))^2+(v-(n/2))^2);
        h(u,v)=1/((1+(d(u,v)/D0)^(2*order)));
    end
end
subplot(2,3,4),imshow(h),title('Butterworth Low Pass Filter')
subplot(2,3,5),mesh(h),title('surface plot BLPF')
B=A_shift.*h;
B_inverse=ifft(B);
B_real=abs(B_inverse);
subplot(2,3,6),imshow(B_real),title('Butterworth Low pass Image')
```

Output:



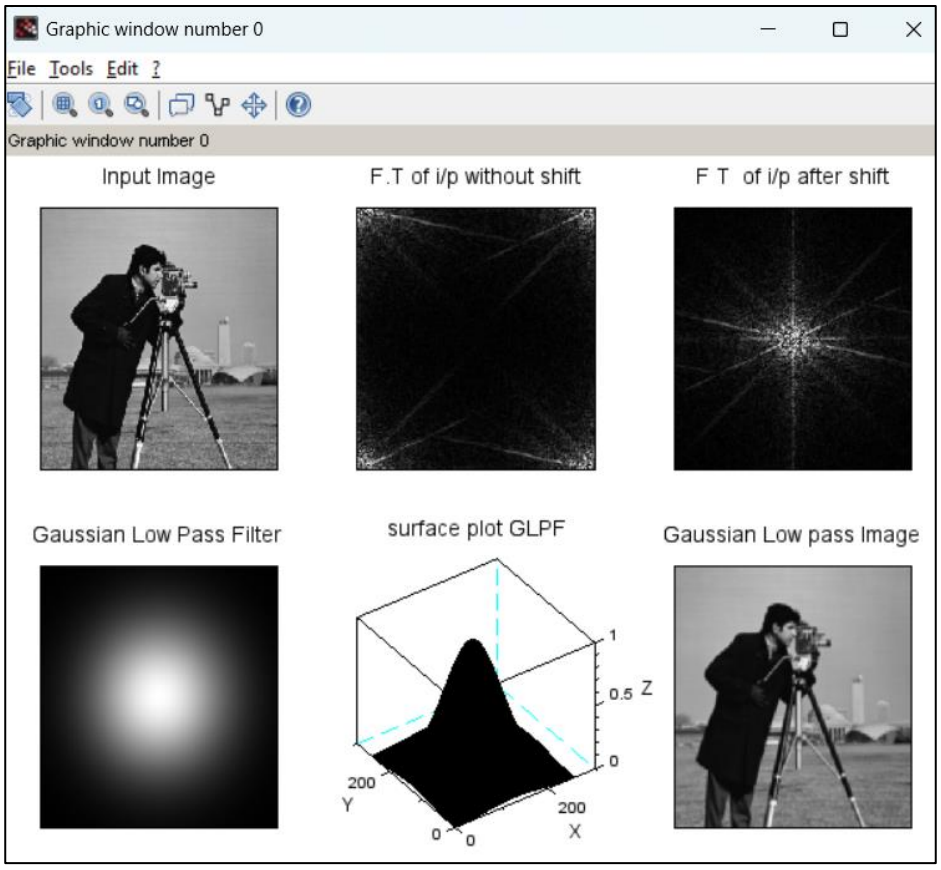
iii. Gaussian Low Pass filter

Code:

```
clc;
clear;
a=imread('C:\Users\sandhya\Desktop\3\cameraman.tif')
a=im2double(a);
subplot(2,3,1)
imshow(a)
```

```
title('Input Image')
[m,n]=size(a);
A=fft2(a);
subplot(2,3,2)
imshow(uint8(abs(A)))
title('F.T of i/p without shift');
A_shift=fftshift(A);
A_real=abs(A_shift);
subplot(2,3,3)
imshow(uint8(A_real))
title('F T  of i/p after shift');
D0=50;
d=zeros(m,n);
order=1;
for u=1:m
    for v=1:n
        d=sqrt((u-(m/2)).^2+(v-(n/2)).^2);
        h(u,v)=exp(-(d^2)/(2*D0.^2));
    end
end
subplot(2,3,4),imshow(h),title('Gaussian Low Pass Filter')
subplot(2,3,5),mesh(h),title('surface plot GLPF')
H_low=A_shift.*h;
H_low_shift=fftshift(H_low);
H_low_shift=ifft(H_low_shift);
B_real=abs(H_low_shift);
subplot(2,3,6),imshow(B_real),title('Gaussian Low pass Image')
```

Output:



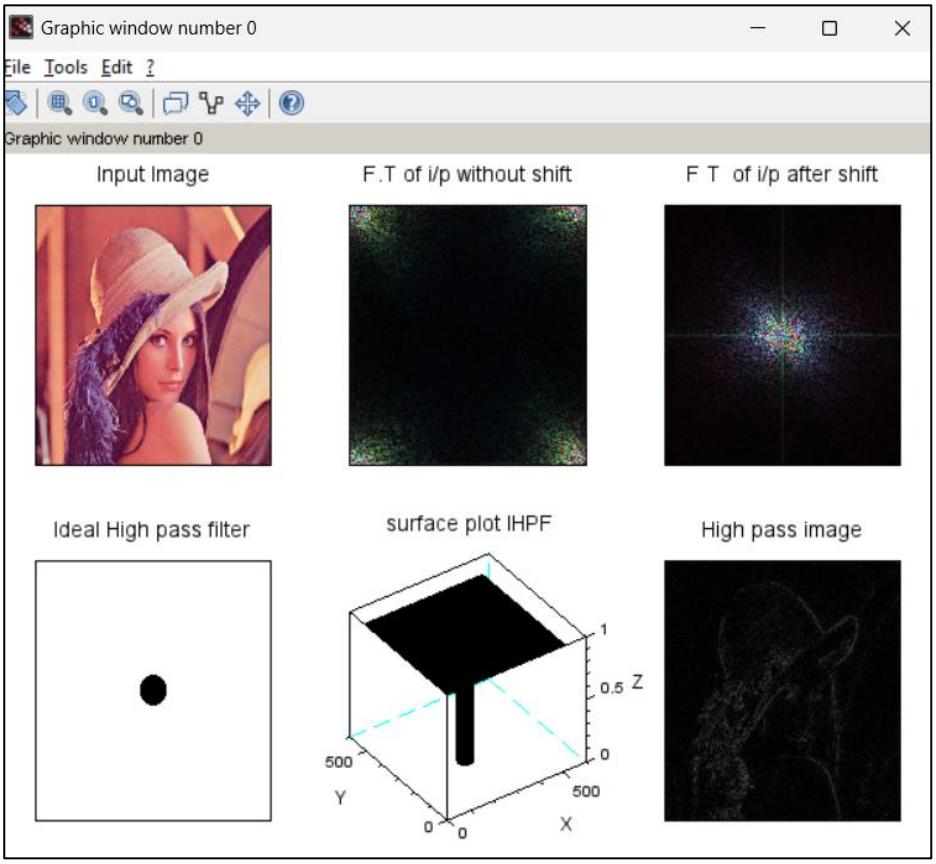
iv. Ideal High Pass filter

Code:

```
clc;
clear all;
a=imread('C:\Users\sandhya\Desktop\3\lena.jpeg')
a=im2double(a);
subplot(2,3,1)
```

```
imshow(a)
title('Input Image')
[m,n]=size(a);
D0=30
A=fft2(a);
subplot(2,3,2)
imshow(uint8(abs(A)))
title('F.T of i/p without shift');
A_shift=fftshift(A);
A_real=abs(A_shift);
subplot(2,3,3)
imshow(uint8(A_real))
title('F T  of i/p after shift');
A_low=zeros(m,n)
d=zeros(m,n)
for u=1:m
    for v=1:n
        d(u,v)=sqrt((u-(m/2))^2+(v-(n/2))^2)
        if d(u,v)<=D0
            A_high(u,v)=0
            H(u,v)=0
        else
            A_high(u,v)=A_shift(u,v)
            H(u,v)=1
        end
    end
end
subplot(2,3,4),imshow(H),title('Ideal High pass filter')
subplot(2,3,5),mesh(H),title('surface plot IHPF')
B=fftshift(A_high);
B_inverse=ifft(B);
B_real=abs(B_inverse);
subplot(2,3,6),imshow(B_real),title('High pass image')
```

Output:

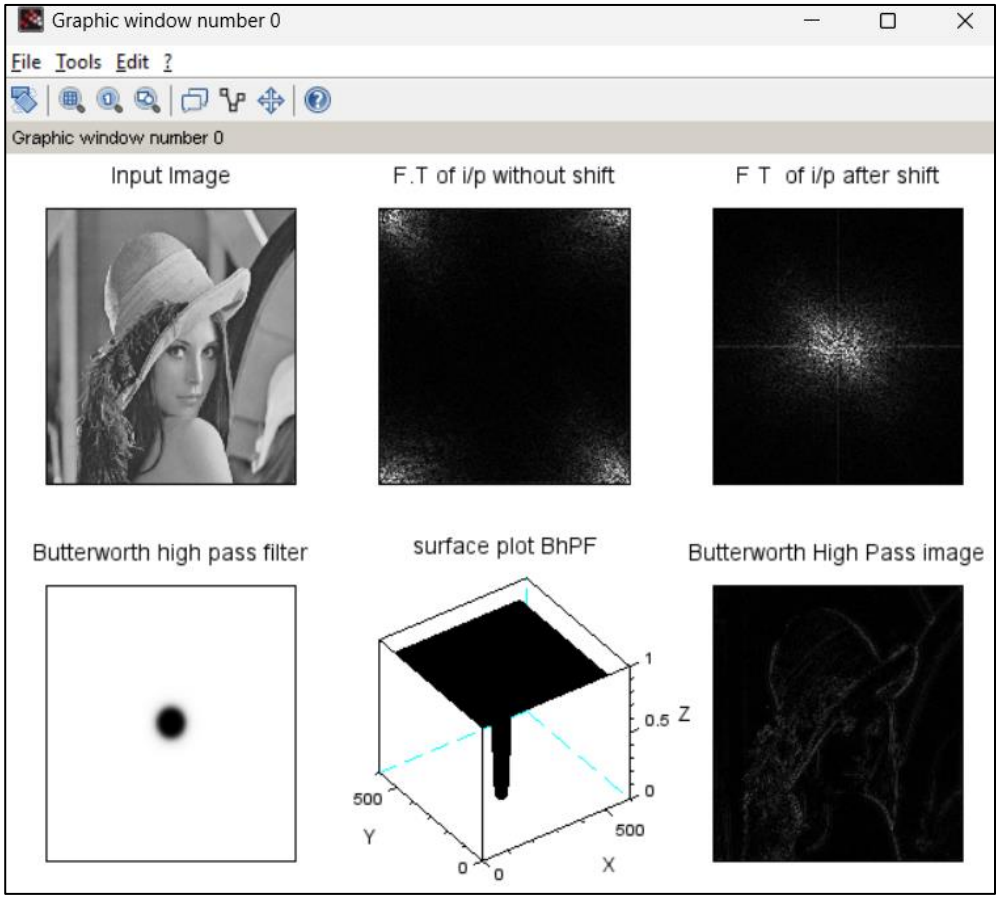


v. Butterworth High Pass filter

Code:

```
clc;
clear;
a=imread('C:\Users\sandhya\Desktop\3\lena_gray.bmp')
a=im2double(a);
subplot(2,3,1)
imshow(a)
title('Input Image')
[m,n]=size(a);
A=fft2(a);
subplot(2,3,2)
imshow(uint8(abs(A)))
title('F.T of i/p without shift');
A_shift=fftshift(A);
A_real=abs(A_shift);
subplot(2,3,3)
imshow(uint8(A_real))
title('F T  of i/p after shift');
D0=30
d=zeros(m,n)
order=4
for u=1:m
    for v=1:n
        d(u,v)=sqrt((u-(m/2))^2+(v-(n/2))^2)
        h(u,v)=1/((1+(D0/d(u,v))^(2*order)))
    end
end
subplot(2,3,4),imshow(h),title('Butterworth high pass filter')
subplot(2,3,5),mesh(h),title('surface plot BhPF')
B=A_shift.*h
B_inverse=ifft(B)
B_real=abs(B_inverse)
subplot(2,3,6),imshow(B_real),title('Butterworth High Pass image')
```

Output:

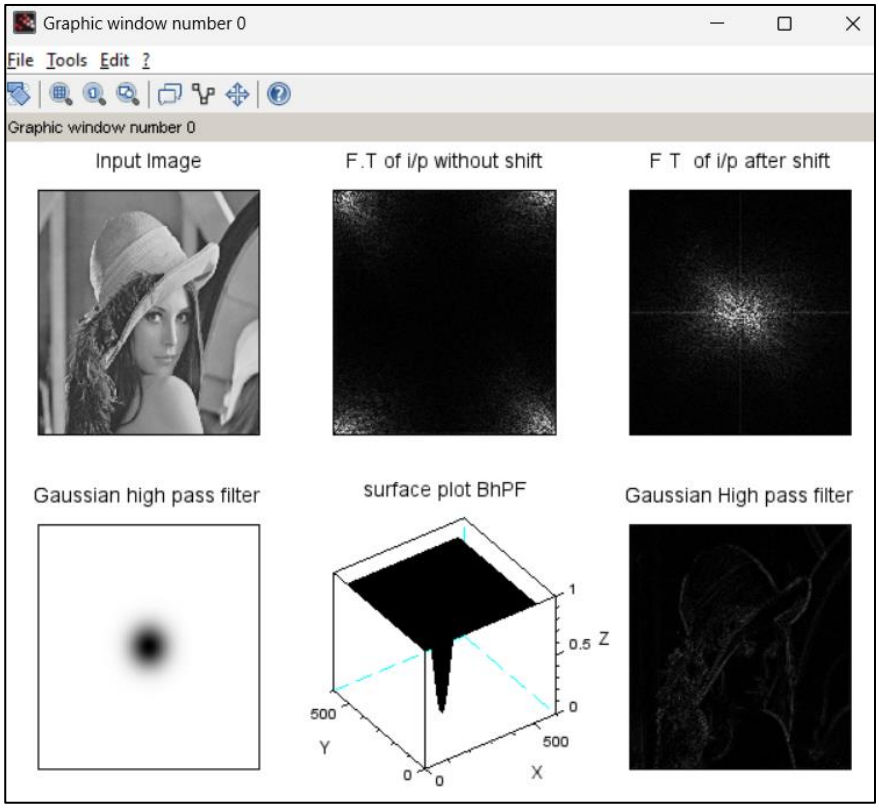


vi. Gaussian High Pass filter

Code:

```
clc;
clear;
a=imread('C:\Users\sandhya\Desktop\3\lena_gray.bmp')
a=im2double(a);
subplot(2,3,1)
imshow(a)
title('Input Image')
[m,n]=size(a);
A=fft2(a);
subplot(2,3,2)
imshow(uint8(abs(A)))
title('F.T of i/p without shift');
A_shift=fftshift(A);
A_real=abs(A_shift);
subplot(2,3,3)
imshow(uint8(A_real))
title('F T  of i/p after shift');
D0=30
d=zeros(m,n)
order=1
for u=1:m
    for v=1:n
        d=sqrt((u-(m/2)).^2+(v-(n/2)).^2)
        h(u,v)=1-exp(-(d^2)/(2*D0.^2))
    end
end
subplot(2,3,4),imshow(h),title('Gaussian high pass filter')
subplot(2,3,5),mesh(h),title('surface plot BhPF')
H_high=A_shift.*h
H_high_shift=fftshift(H_high)
H_high_shift=ifft(H_high_shift)
B_real=abs(H_high_shift)
subplot(2,3,6),imshow(B_real),title('Gaussian High pass filter')
```

Output:

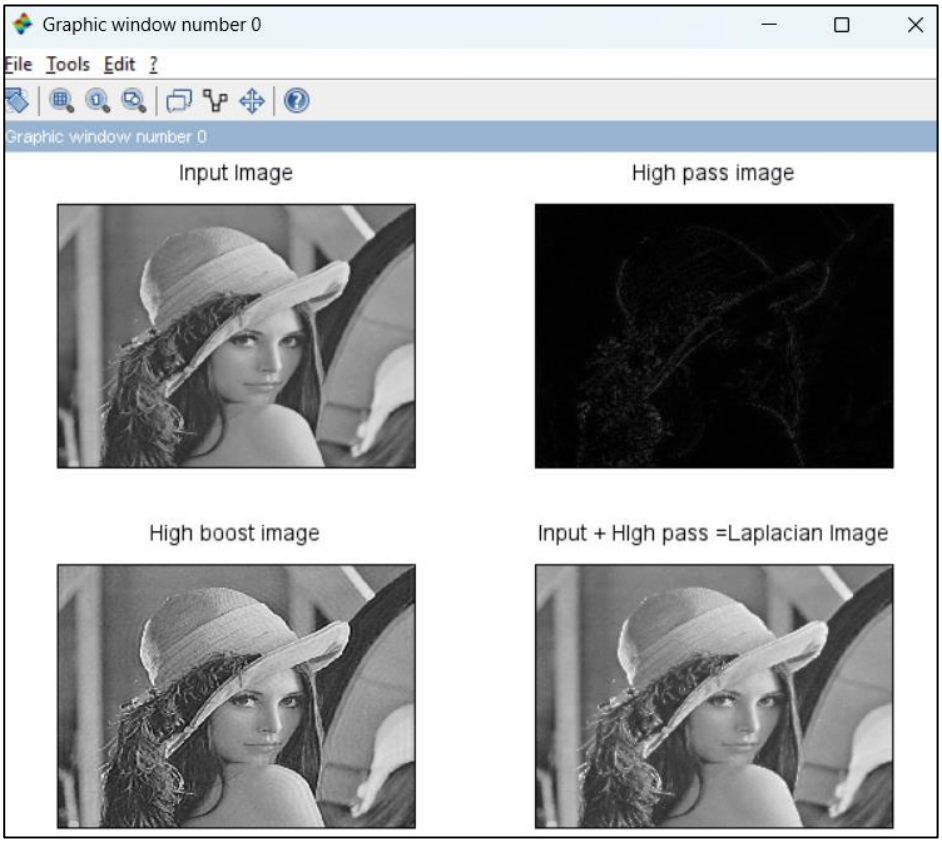


C. Program to apply Laplacian filter in frequency domain.

Code:

```
clc
clear all;
a=imread('C:\Users\sandhya\Desktop\3\lena_gray.BMP')
a=im2double(a)
subplot(2,2,1),imshow(a),title('Input Image')
[m,n]=size(a)
D0=50
A=fft2(a)
A_shift=fftshift(A)
A_real=abs(A_shift)
H=zeros(m,n)
D=zeros(m,n)
for u=1:m
    for v=1:n
        D(u,v)=sqrt((u-(m/2))^2+(v-(n/2))^2)
        if D(u,v)<=D0
            H(u,v)=0
        else
            H(u,v)=1
        end
    end
end
AHB=2.0
H1=(AHB-1)+H
X=A_shift.*H
X1=A_shift.*H1
XA=abs(ifft(X))
XB=abs(ifft(X1))
subplot(2,2,2),imshow(XA),title('High pass image')
subplot(2,2,3),imshow(XB),title('High boost image')
subplot(2,2,4),imshow(a+XA),title('Input + HIgh pass =Laplacian Image')
```

Output:

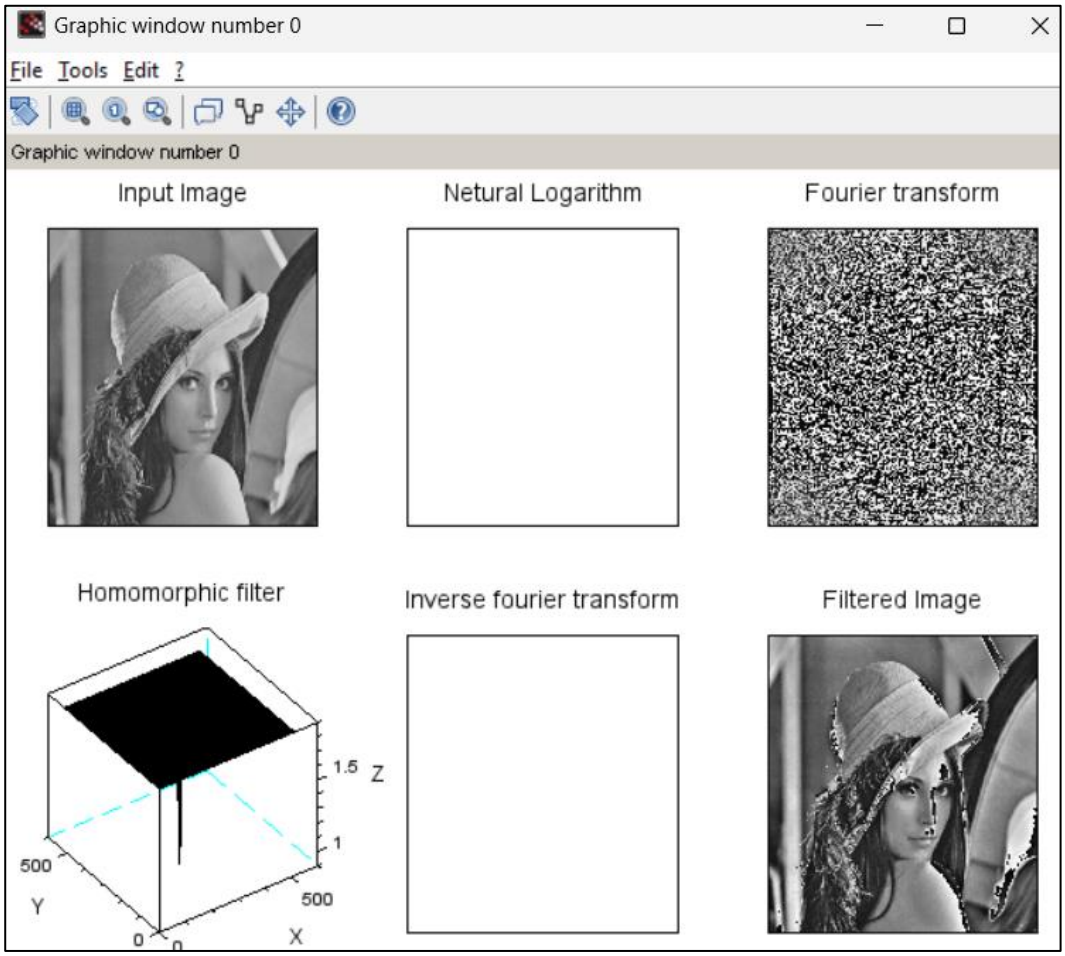


D. Program for homomorphic filtering

Code:

```
clc
clear  all
a=imread('C:\Users\sandhya\Desktop\3\lena_gray.BMP')
subplot(2,3,1),imshow(a),title('Input Image')
a=double(a)
b=a
D0=50
GL=0.9
GH=1.9
[m,n]=size(a)
b=b+1
log_b=log(b)
subplot(2,3,2),imshow(log_b),title('Netural Logarithm')
c=fft2(log_b)
subplot(2,3,3),imshow(uint8(c)),title('Fourier transform')
dd=fftshift(c)
for u=1:m
    for v=1:n
        H(u,v)=(GH -GL)*(1-exp(-1*(sqrt((u-m/2)^2+(v-n/2)^2))^2/D0)^2)+GL
        end
end
subplot(2,3,4),mesh(H),title('Homomorphic filter')
x=dd.*H
real_x=abs(ifft(x))
subplot(2,3,5),imshow(real_x),title('Inverse fourier transform')
Final = exp(real_x)
subplot(2,3,6),imshow(uint8(Final)),title('Filtered Image')
```

Output:



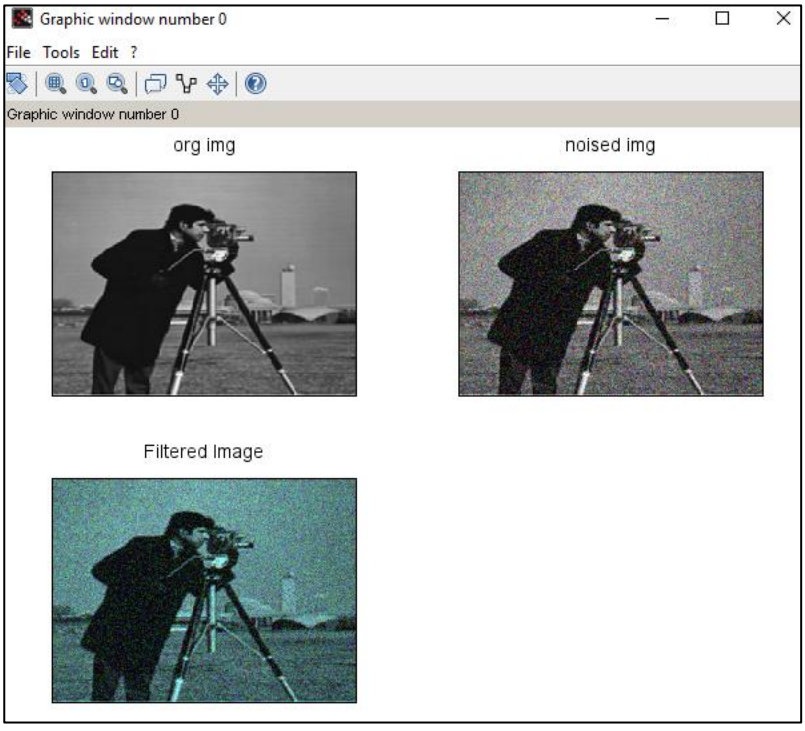
Practical 4

A. Program to denoise using spatial mean, median filtering.

Code for mean filtering:

```
clc;
clear all;
a=imread('D:\MSC IT\Part I\Sem II\Image
Processing\IP_Practical\IP_Practical_Images\camera.png');
b1=double(a);
c=imnoise(a,'gaussian');
d=double(c);
b=d;
m=(1/9)*(ones(3,3));
[r1,c1]=size(a);
subplot(2,2,1);
imshow(a);
title('org img');
subplot(2,2,2);
imshow(c);
title('noised img');
for i=2:r1-1
for j=2:c1-1
a1=d(i-1,j-1)+d(i-1,j)+d(i-1,j+1)+d(i,j-1)+d(i,j)+d(i,j+1)
+d(i+1,j-1)+d(i+1,j)+d(i+1,j+1);
b(i,j)=a1*(1/9);
end end
subplot(2,2,3);
imshow(uint8(b));
title('Filtered Image');
```

Output:

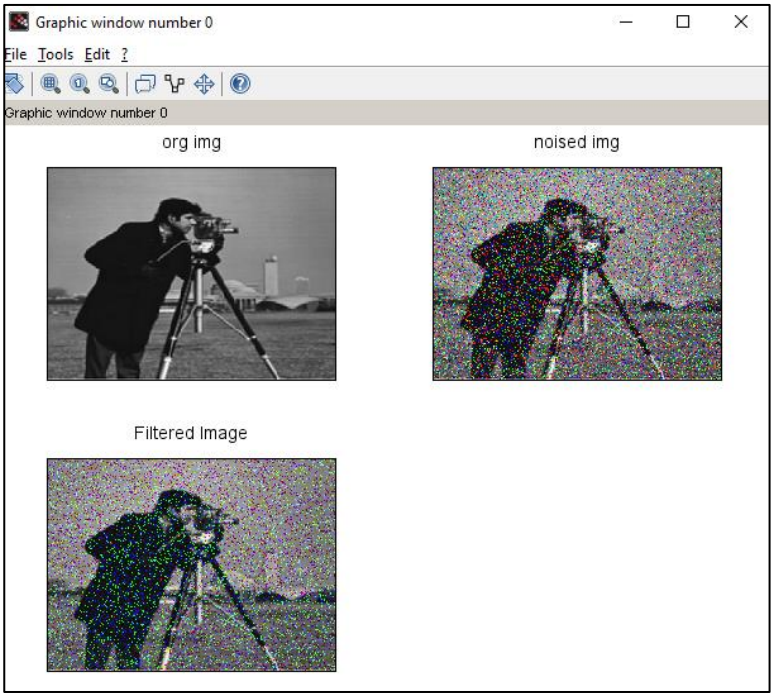


Code for Median filtering:

```
clc;
clear all;
a=imread('D:\MSC IT\Part I\Sem II\Image
Processing\IP_Practical\IP_Practical_Images\camera.png');
b1 = double(mtlb_double(a));
c = imnoise(a,"salt & pepper",0.2);
d = double(mtlb_double(c));
```

```
b = d;
m = (1/9)*ones(3,3);
subplot(2,2,1);
imshow(a);
title('org img');
subplot(2,2,2);
imshow(c);
title('noised img');
[r1,c1] = size(mtlb_double(a));
for i = 2:r1-1
        for j = 2:c1-1
        a1          =          [d(i-1,j-1),d(i-1,j),d(i-1,j+1),d(i,j-1),d(i,j),
d(i,j+1),d(i+1,j-1),d(i+1,j),d(i+1,j+1)];
        a2  =  gsort(a1,"g","i");//gsort(A,'g','i')  sort  the  elements  of  the
array A in the increasing order.
        med = a2(5);
        b(i,j) = med;
        end;
end;
subplot(2,2,3);
imshow(uint8(b));
title('Filtered Image');
```

Output:

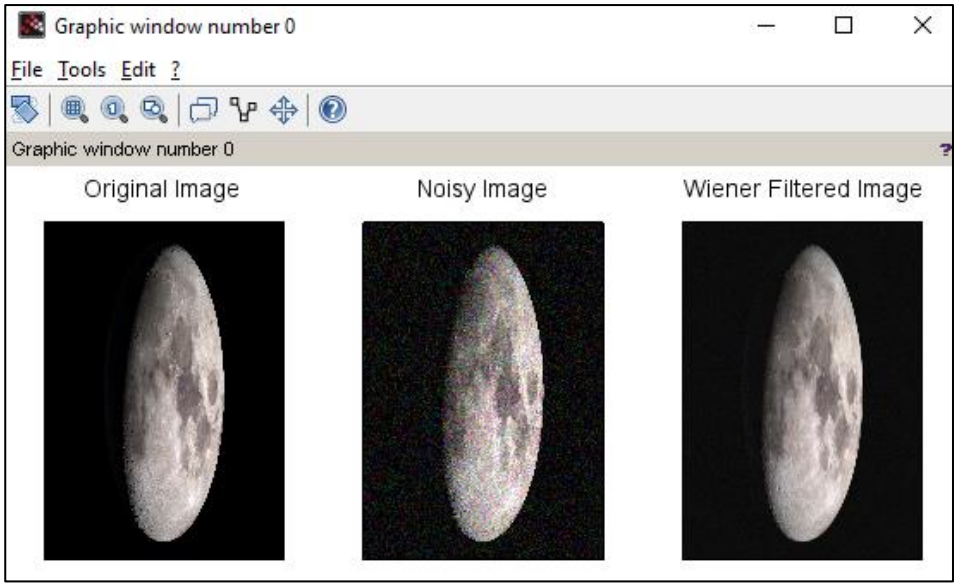


B. Program for Image deblurring using inverse, Wiener filters.

Code for Wiener filter:

```
clc;
clear all;
Image=imread('D:\MSC             IT\Part             I\Sem             II\Image
Processing\IP_Practical\IP_Practical_Images\moon.tif')
NoisyImage= imnoise(Image,'gaussian',0.02);
wienerfilter=imwiener2(NoisyImage,[5,5],0.2);
subplot(1,3,1);
imshow(Image);
title('Original Image');
subplot(1,3,2);
imshow(NoisyImage);
title('Noisy Image');
subplot(1,3,3);
imshow(wienerfilter);
title('Wiener Filtered Image');
```

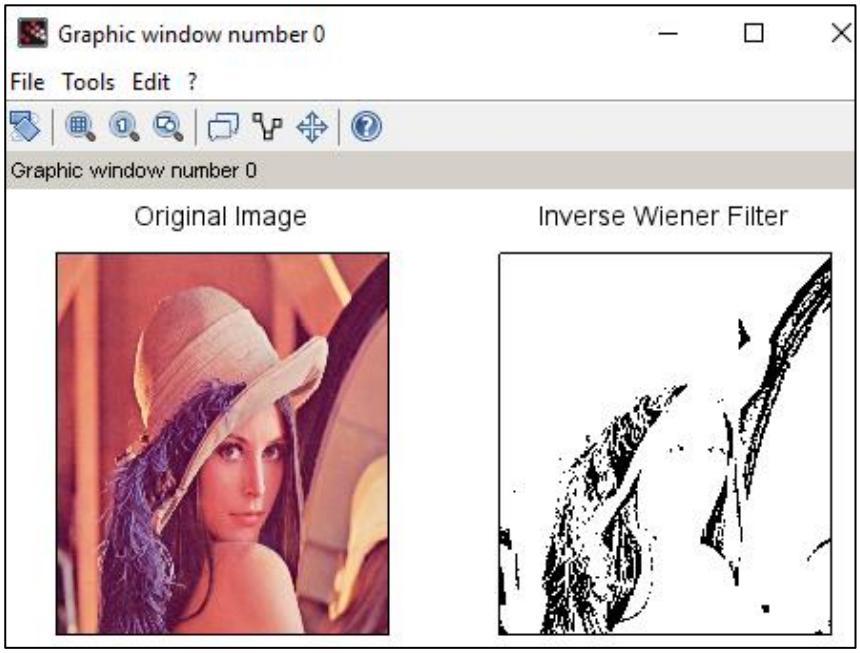
Output:



Code for Inverse Wiener Filter:

```
a=imread('D:\MSC IT\Part I\Sem II\Image Processing\IP_Practical\IP_Practical_Images\lena.png');
subplot(1,2,1);
title("Original Image")
imshow(a);
b=double(a);
[m,n]=size (b);
T=100;
for I=1: m
    for J=1:n
        if (b(I,J) <T)
            c(I,J)=0;
        else
            c(I,J)=255;
        end
    end
end
subplot(1,2,2);
title("Inverse Wiener Filter")
imshow(uint8(c));
```

Output:



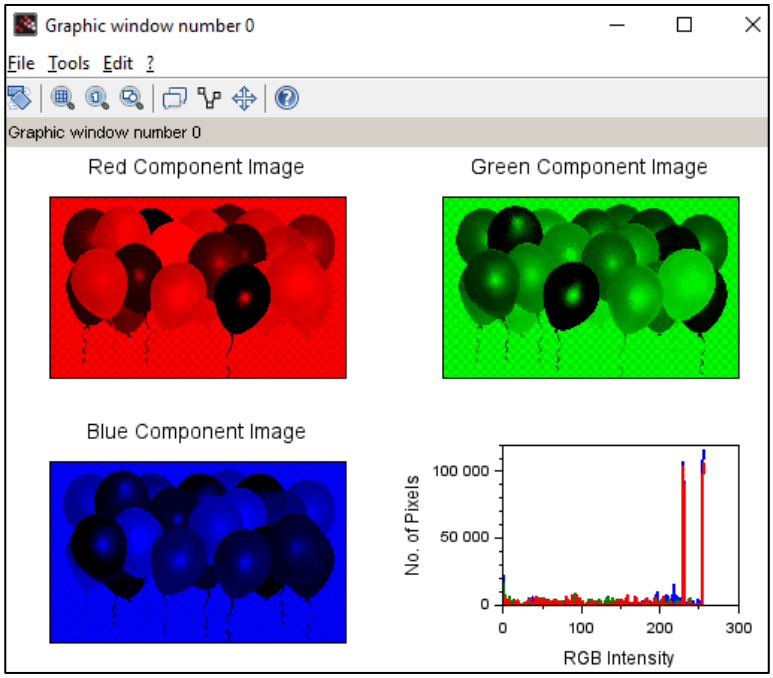
Practical 5

A. Program to read a colour image and segment into RGB planes, histogram of colour image.

Code:

```
clc;
clear all;
Image=imread('D:\MSC IT\Part I\Sem II\Image Processing\IP_Practical\IP_Practical_Images\rgb.png');
r=size(Image,1);
c=size(Image,2);
R=zeros(r,c,3);
G=zeros(r,c,3);
B=zeros(r,c,3);
R(:,:,1)=Image(:,:,1);
G(:,:,2)=Image(:,:,2);
B(:,:,3)=Image(:,:,3);
subplot(2,2,1);imshow(uint8(R));title('Red Component Image');
subplot(2,2,2);imshow(uint8(G));title('Green Component Image');
subplot(2,2,3);imshow(uint8(B));title('Blue Component Image');
nBins = 256;
[yR,x] = imhist(Image(:,:,1),nBins);
[yG,x] = imhist(Image(:,:,2),nBins);
[yB,x] = imhist(Image(:,:,3),nBins);
subplot(2,2,4)
plot(x,yR,x,yG,x,yB,"Linewidth",2);
xlabel("RGB Intensity");
ylabel("No. of Pixels");
```

Output:



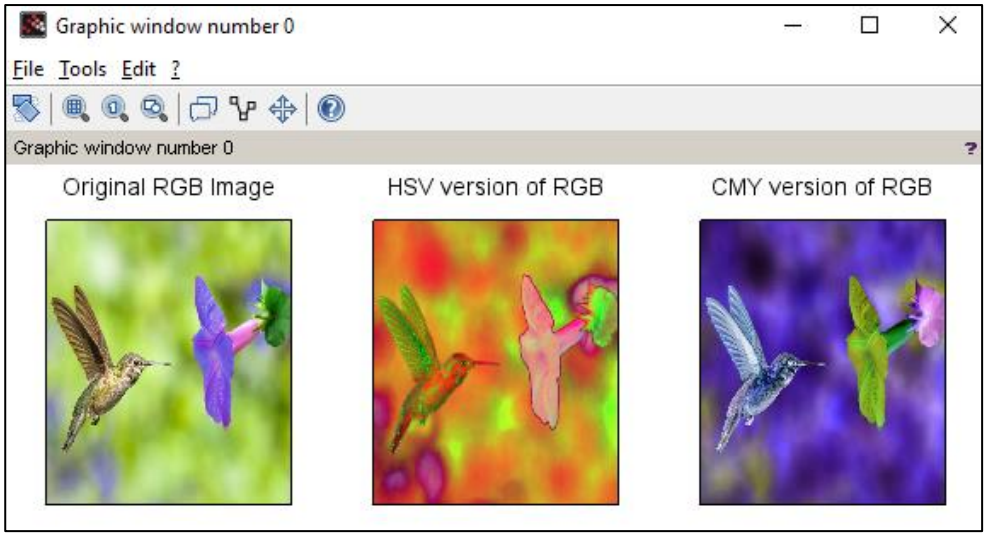
B. Program for converting from one colour model to another model.

Code:

```
clc;
clear all;
x=imread('C:\Users\Admin\Pictures\Nature\hummingbird-visiting-morning-glory.jpg')
//Displayimg RGB image
subplot(1,3,1);
imshow(x);
title('Original RGB Image')
```

```
//Displaying HSV image
HSV = rgb2hsv(x);
subplot(1,3,2);
imshow(HSV);
title('HSV version of RGB')
CMY=imcomplement(x);
subplot(1,3,3)
imshow(CMY)
title('CMY version of RGB')
```

Output:

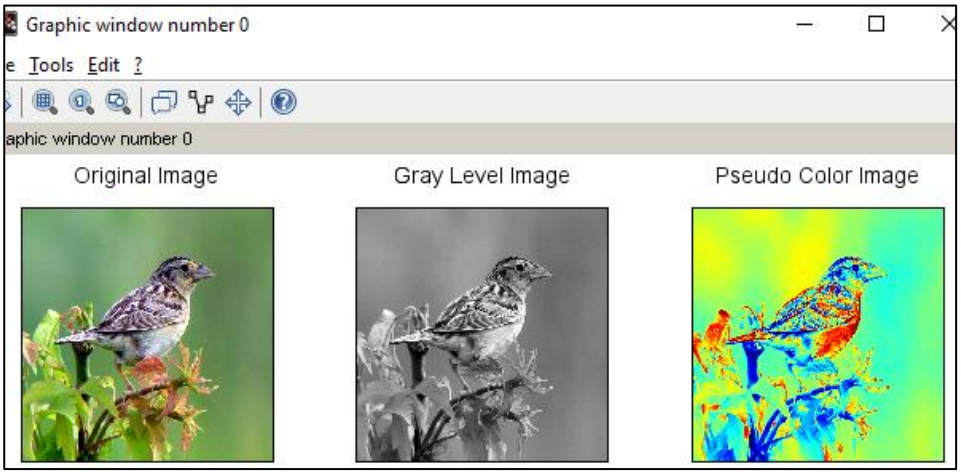


C. Program to apply false colouring(pseudo) on a gray scale image.

Code:

```
clc;
close;
a = imread('D:\MSC IT\Part I\Sem II\Image Processing\IP_Practical\sparrow1.jpg');
//Displaying Original RGB image
subplot(1,3,1)
imshow(a);
title("Original Image")
//Displaying Gray level image
b = rgb2gray(a);
subplot(1,3,2)
imshow(b);
title("Gray Level Image")
//Displaying False coloring(Pseudo) image
subplot(1,3,3)
imshow(b,jetcolormap(256));
title("Pseudo Color Image");
```

Output:



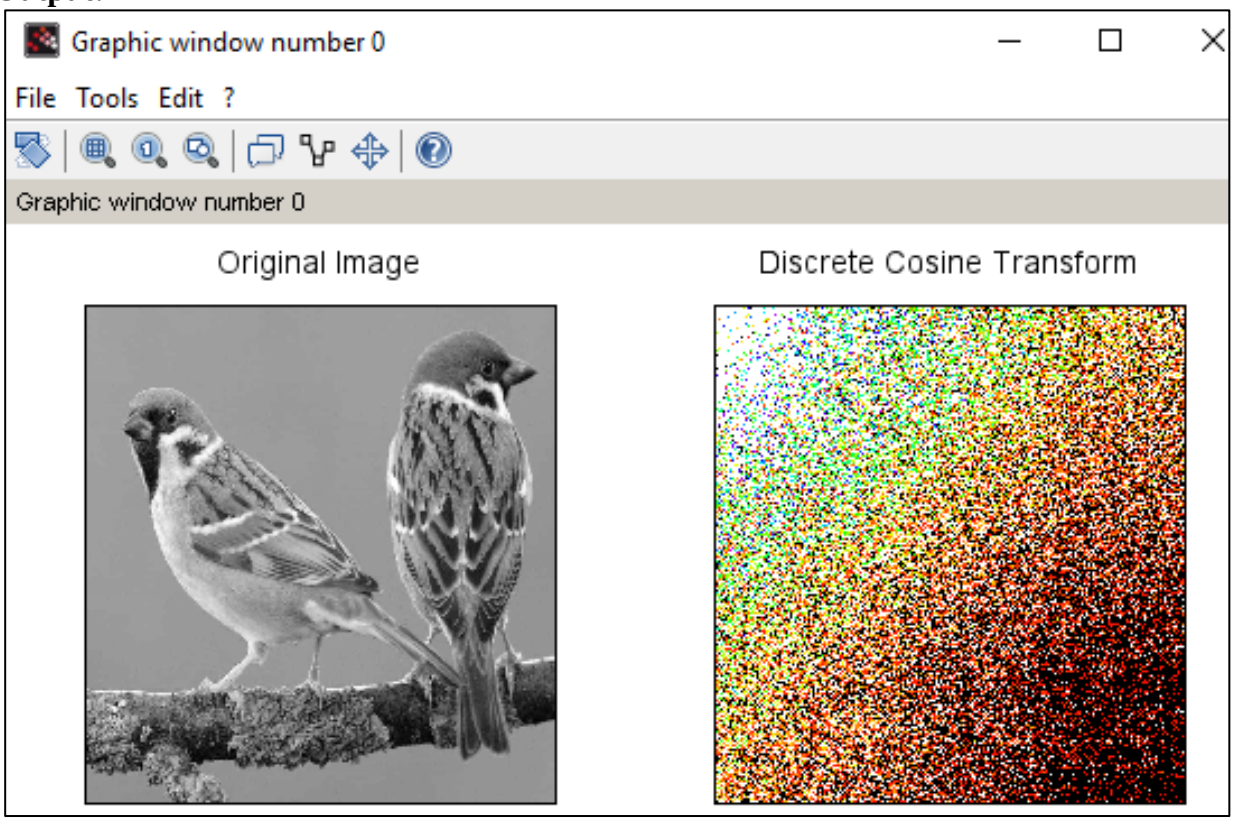
Practical 6

A. Program to compute Discrete Cosine Transform.

Code:

```
clc;
clear all;
a = imread('D:\MSC IT\Part I\Sem II\Image Processing\IP_Practical\IP_Practical_Images\Gray_sparrow.png');
dctImg = imdct(a);
subplot(1,2,1)
imshow(a);
title('Original Image');
subplot(1,2,2);
imshow(dctImg,rainbowcolormap(32));
title('Discrete Cosine Transform');
```

Output:



Practical 7

A. Program to apply compression and decompression algorithm on an image using Huffman Coding Technique.

Code:

Huffman.py:

```
import re
import numpy as np
from PIL import Image
print("Huffman Compression Program")
print("================================================================")
h = int(input("Enter 1 if you want to input an colour image file, 2 for
default gray scale case:"))
if h == 1:
    file = input("Enter the filename:")
    my_string = np.asarray(Image.open(file),np.uint8)
    shape = my_string.shape
    a = my_string
    print ("Enetered string is:",my_string)
    my_string = str(my_string.tolist())
elif h == 2:
    array = np.arange(0, 737280, 1, np.uint8)
    my_string = np.reshape(array, (1024, 720))
    print ("Enetered string is:",my_string)
    a = my_string
    my_string = str(my_string.tolist())
else:
     print("You  entered  invalid  input")                    #  taking  user
input
letters = []
only_letters = []
for letter in my_string:
    if letter not in letters:
         frequency  =  my_string.count(letter)              #frequency  of  each
letter repetition
        letters.append(frequency)
        letters.append(letter)
        only_letters.append(letter)
nodes = []
while len(letters) > 0:
    nodes.append(letters[0:2])
     letters  =  letters[2:]                                # sorting according
to frequency
nodes.sort()
huffman_tree = []
huffman_tree.append(nodes)                                 #Make   each   unique
character as a leaf node
def combine_nodes(nodes):
    pos = 0
    newnode = []
    if len(nodes) > 1:
        nodes.sort()
        nodes[pos].append("1")                          # assigning values 1 and
0
        nodes[pos+1].append("0")
        combined_node1 = (nodes[pos] [0] + nodes[pos+1] [0])
        combined_node2 = (nodes[pos] [1] + nodes[pos+1] [1])  # combining the
nodes to generate pathways
        newnode.append(combined_node1)
        newnode.append(combined_node2)
        newnodes=[]
        newnodes.append(newnode)
        newnodes = newnodes + nodes[2:]
```

```
        nodes = newnodes
        huffman_tree.append(nodes)
        combine_nodes(nodes)
     return  huffman_tree                                    #  huffman  tree
generation
newnodes = combine_nodes(nodes)
huffman_tree.sort(reverse = True)
print("Huffman tree with merged pathways:")
checklist = []
for level in huffman_tree:
    for node in level:
        if node not in checklist:
            checklist.append(node)
        else:
            level.remove(node)
count = 0
for level in huffman_tree:
    print("Level", count,":",level)             #print huffman tree
    count+=1
print()
letter_binary = []
if len(only_letters) == 1:
    lettercode = [only_letters[0], "0"]
    letter_binary.append(letter_code*len(my_string))
else:
    for letter in only_letters:
        code =""
        for node in checklist:
             if  len  (node)>2  and  letter  in  node[1]:           #genrating
binary code
                 code = code + node[2]
        lettercode =[letter,code]
        letter_binary.append(lettercode)
print(letter_binary)
print("Binary code generated:")
for letter in letter_binary:
    print(letter[0], letter[1])
bitstring =""
for character in my_string:
    for item in letter_binary:
        if character in item:
            bitstring = bitstring + item[1]
binary ="0b"+bitstring
print("Your message as binary is:")
                                        # binary code generated
uncompressed_file_size = len(my_string)*7
compressed_file_size = len(binary)-2
print("Your  original  file  size  was",  uncompressed_file_size,"bits.  The
compressed size is:",compressed_file_size)
print("This      is      a      saving      of      ",uncompressed_file_size-
compressed_file_size,"bits")
output = open("compressed.txt","w+")
print("Compressed file generated as compressed.txt")
output = open("compressed.txt","w+")
print("Decoding.......")
output.write(bitstring)
bitstring = str(binary[2:])
uncompressed_string =""
code =""
for digit in bitstring:
    code = code+digit
    pos=0                                    #iterating and decoding
    for letter in letter_binary:
        if code ==letter[1]:
```

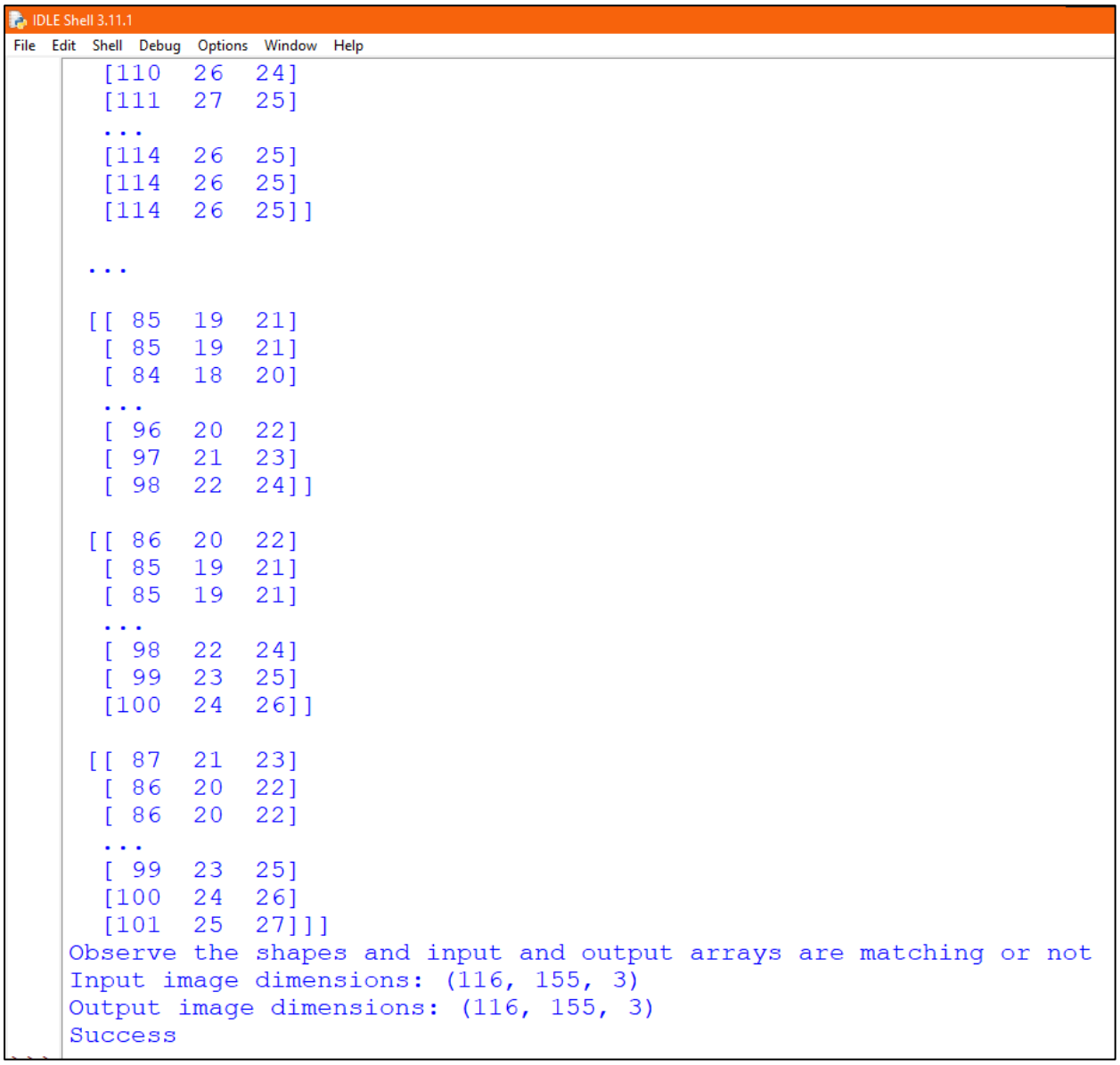
```
            uncompressed_string=uncompressed_string+letter_binary[pos] [0]
            code=""
        pos+=1
print("Your UNCOMPRESSED data is:")
if h == 1:
    temp = re.findall(r'\d+', uncompressed_string)
    res = list(map(int, temp))
    res = np.array(res)
    res = res.astype(np.uint8)
    res = np.reshape(res, shape)
    print(res)
    print("Observe the shapes and input and output arrays are matching or
not")
    print("Input image dimensions:",shape)
    print("Output image dimensions:",res.shape)
    data = Image.fromarray(res)
    data.save('uncompressed.png')
    if a.all() == res.all():
        print("Success")
if h == 2:
    temp = re.findall(r'\d+', uncompressed_string)
    res = list(map(int, temp))
    print(res)
    res = np.array(res)
    res = res.astype(np.uint8)
    res = np.reshape(res, (1024, 720))
    print(res)
    data = Image.fromarray(res)
    data.save('uncompressed.png')
    print("Success")
```

Output:

```
>>>
================== RESTART: C:/Users/admin/Desktop/huffman.py ==================
Huffman Compression Program
================================================================
Enter 1 if you want to input an colour image file, 2 for default gray scale case:1
Enter the filename:hanuman1.jpg
Enetered string is: [[[111  27  25]
  [111  27  25]
  ...
  [115  27  26]
  [115  27  26]
  [115  27  26]]

 [[110  26  24]
  [111  27  25]
  [111  27  25]
  ...
  [115  27  26]
  [115  27  26]
  [115  27  26]]

 [[110  26  24]
  [110  26  24]
  [111  27  25]
  ...
  [114  26  25]
  [114  26  25]
  [114  26  25]]

 ...

 [[ 85  19  21]
  [ 85  19  21]
  [ 84  18  20]
  ...
  [ 96  20  22]
  [ 97  21  23]
  [ 98  22  24]]
```

```
  [ 85  19  21]
  [ 84  18  20]
  ...
  [ 96  20  22]
  [ 97  21  23]
  [ 98  22  24]]

 [[ 86  20  22]
  [ 85  19  21]
  [ 85  19  21]
  ...
  [ 98  22  24]
  [ 99  23  25]
  [100  24  26]]

 [[ 87  21  23]
  [ 86  20  22]
  [ 86  20  22]
  ...
  [ 99  23  25]
  [100  24  26]
  [101  25  27]]]
Huffman tree with merged pathways:
Level 0 : [[273712, ',216509[]7843 ']]
Level 1 : [[108983, ',21', '1'], [164729, '6509[]7843 ', '0']]
Level 2 : [[69943, '6509[]', '1'], [94786, '7843 ', '0']]
Level 3 : [[53939, ',', '1'], [55044, '21', '0'], [94786, '7843 ', '0']]
Level 4 : [[40847, '7843', '1'], [53939, ' ', '0'], [55044, '21', '0']]
Level 5 : [[33749, '6509', '1'], [36194, '[]', '0'], [53939, ' ', '0'], [55044, '21', '0']]
Level 6 : [[24786, '2', '1'], [30258, '1', '0'], [36194, '[]', '0'], [53939, ' ', '0']]
Level 7 : [[18498, '78', '1'], [22349, '43', '0'], [30258, '1', '0'], [36194, '[]', '0'], [53939, ',', '1']]
Level 8 : [[18097, '[', '1'], [18097, ']', '0'], [22349, '43', '0'], [30258, '1', '0'], [53939, ' ', '0']]
Level 9 : [[16365, '65', '1'], [17384, '09', '0'], [18097, ']', '0'], [22349, '43', '0'], [30258, '1', '0'], [53939, ',', '1']]
Level 10 : [[9555, '4', '1'], [12794, '3', '0'], [17384, '09', '0'], [18097, ']', '0'], [24786, '2', '1'], [53939, ' ', '0']]
Level 11 : [[9076, '7', '1'], [9422, '8', '0'], [12794, '3', '0'], [17384, '09', '0'], [18097, ']', '0'], [30258, '1', '0'], [53939, ',', '1'
]]
Level 12 : [[8533, '0', '1'], [8851, '9', '0'], [9422, '8', '0'], [12794, '3', '0'], [18097, '[', '1'], [24786, '2', '1'], [53939, ' ', '0']]
Level 13 : [['7885, '6', '1'], [8480, '5', '0'], [8851, '9', '0'], [9422, '8', '0'], [12794, '3', '0'], [18097, ']', '0'], [30258, '1', '0'],
[53939, ',', '1']]

[['[', '0101'], ['1', '100'], [',', '11'], [' ', '000'], ['2', '101'], ['7', '00111'], ['5', '01110'], [']', '0100'], ['8', '00110'], ['6', '
01111'], ['3', '00100'], ['4', '00101'], ['9', '01100'], ['0', '01101']]
```

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
[['[', '0101'], ['1', '100'], [',', '11'], [' ', '000'], ['2', '101'], ['7', '00111'], ['5', '01110'], [']', '0100'], ['8', '00110'], ['6', '
01111'], ['3', '00100'], ['4', '00101'], ['9', '01100'], ['0', '01101']]
Binary code generated:
[ 0101
1 100
, 11
  000
2 101
7 00111
5 01110
] 0100
8 00110
6 01111
3 00100
4 00101
9 01100
0 01101
Your message as binary is:
Your original file size was 1915984 bits. The compressed size is: 952583
This is a saving of  963401 bits
Compressed file generated as compressed.txt
Decoding.......
Your UNCOMPRESSED data is:
[[[111  27  25]
  [111  27  25]
  [111  27  25]
  ...
  [115  27  26]
  [115  27  26]
  [115  27  26]]

 [[110  26  24]
  [111  27  25]
  [111  27  25]
  ...
  [115  27  26]
  [115  27  26]
  [115  27  26]]

 [[110  26  24]
  [110  26  24]
  [111  27  25]
```


IDLE Shell 3.11.1
File Edit Shell Debug Options Window Help
[110 26 24]
[111 27 25]
...
[114 26 25]
[114 26 25]
[114 26 25]]
...
[[85 19 21]
[85 19 21]
[84 18 20]
...
[96 20 22]
[97 21 23]
[98 22 24]]
[[86 20 22]
[85 19 21]
[85 19 21]
...
[98 22 24]
[99 23 25]
[100 24 26]]
[[87 21 23]
[86 20 22]
[86 20 22]
...
[99 23 25]
[100 24 26]
[101 25 27]]]
Observe the shapes and input and output arrays are matching or not
Input image dimensions: (116, 155, 3)
Output image dimensions: (116, 155, 3)
Success

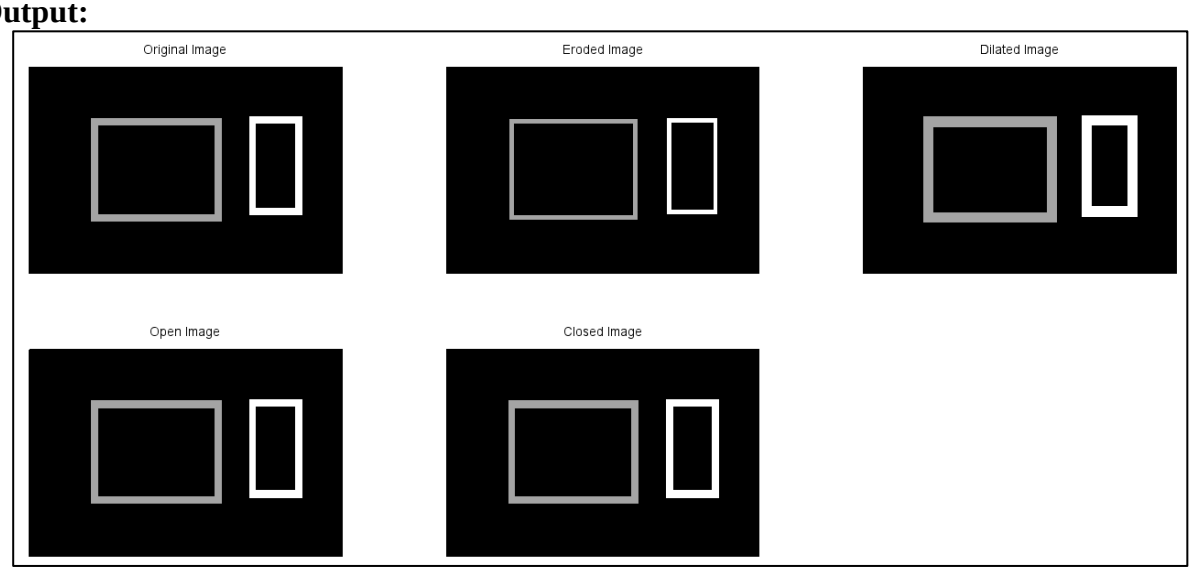
Practical 8

A. Program to apply erosion, dilation, opening, closing.

Code:

```
8A-Opening
clear;
clc;
a = imread('D:\MSC IT\Part I\Sem II\Image Processing\IP_Practical\IP_Practical_Images\rectb.png')
I=rgb2gray(a);
subplot (2,3,1);imshow(I);title("Original Image ") ;
se = imcreatese('rect',3,3);
erode= imerode (I, se);
subplot(2,3,2);imshow (erode);title ("Eroded Image");
dilate = imdilate (I,se);
subplot (2,3,3); imshow (dilate); title ("Dilated Image");
open= imopen(I, se);
subplot (2,3,4); imshow (open); title ("Open Image");
closed= imclose (I,se);
subplot (2,3,5) ; imshow (closed); title ("Closed Image") ;
```

Output:



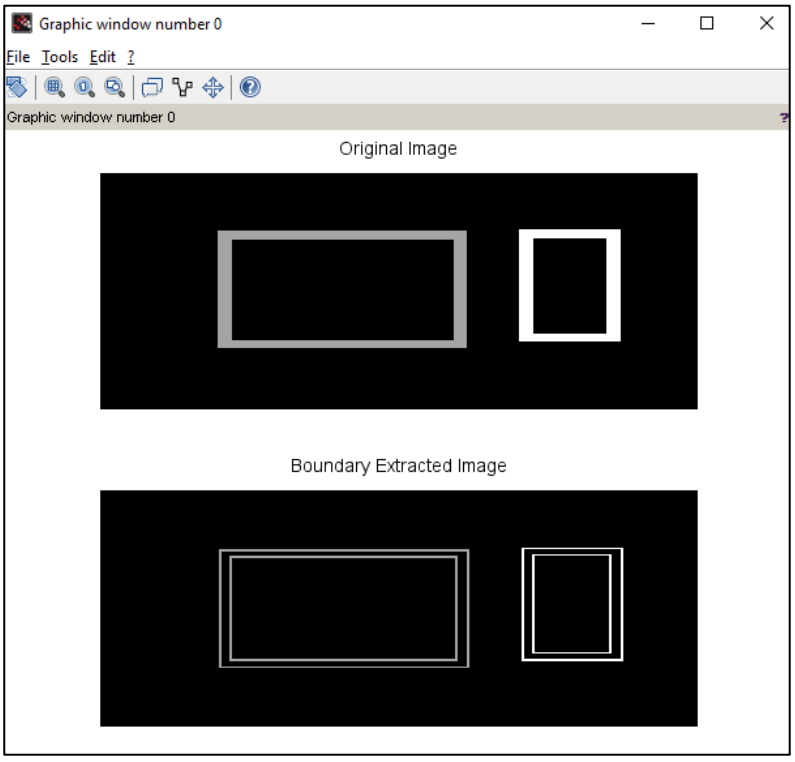
B. Program for detecting boundary of an Image.

Code:

```
clc;
clear all;
a=imread('D:\MSC IT\Part I\Sem II\Image Processing\IP_Practical\IP_Practical_Images\rectb.png');
a=rgb2gray(a);
subplot(2,1,1);
imshow(a);
title('Original Image');
d=a;
[r,c]=size(d);
m=[1 1 1;1 1 1;1 1 1];
for i=2:1:r-1
for j=2:1:c-1
new=[(m(1)*d(i-1,j-1)) (m(2)*d(i-1,j)) (m(3)*d(i-1,j+1))
(m(4)*d(i,j-1)) (m(5)*d(i,j)) (m(6)*d(i,j+1))
(m(7)*d(i+1,j-1)) (m(8)*d(i+1,j)) (m(9)*d(i+1,j+1))];
A2(i,j)=min(new);
aa(i,j)=d(i,j)-A2(i,j);
end
```

```
end
subplot(2,1,2);
imshow(aa);title('Boundary Extracted Image');
```

Output:

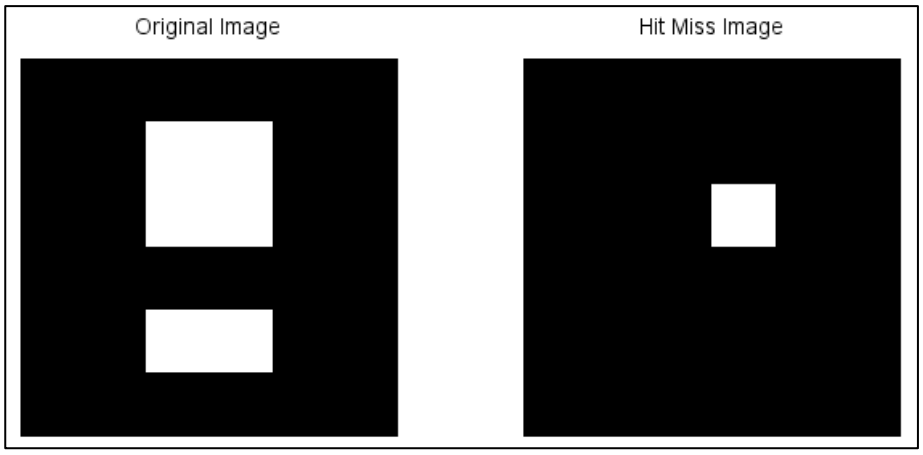


C. Program to apply Hit-or-Miss transform.

Code:

```
clear;
clc;
a=[0 0 0 0 0 0 ;
   0 0 1 1 0 0 ;
   0 0 1 1 0 0 ;
   0 0 0 0 0 0 ;
   0 0 1 1 0 0 ;
   0 0 0 0 0 0 ];
a = im2bw(a,0.5);
se = [0 0 0 0;
      0 1 1 0;
      0 1 1 0;
      0 0 0 0];
s2 = imhitmiss(a,se);
subplot(1,2,1);imshow(a);title("Original Image  ");
subplot(1,2,2);imshow(s2);title("Hit Miss Image");
```

Output:

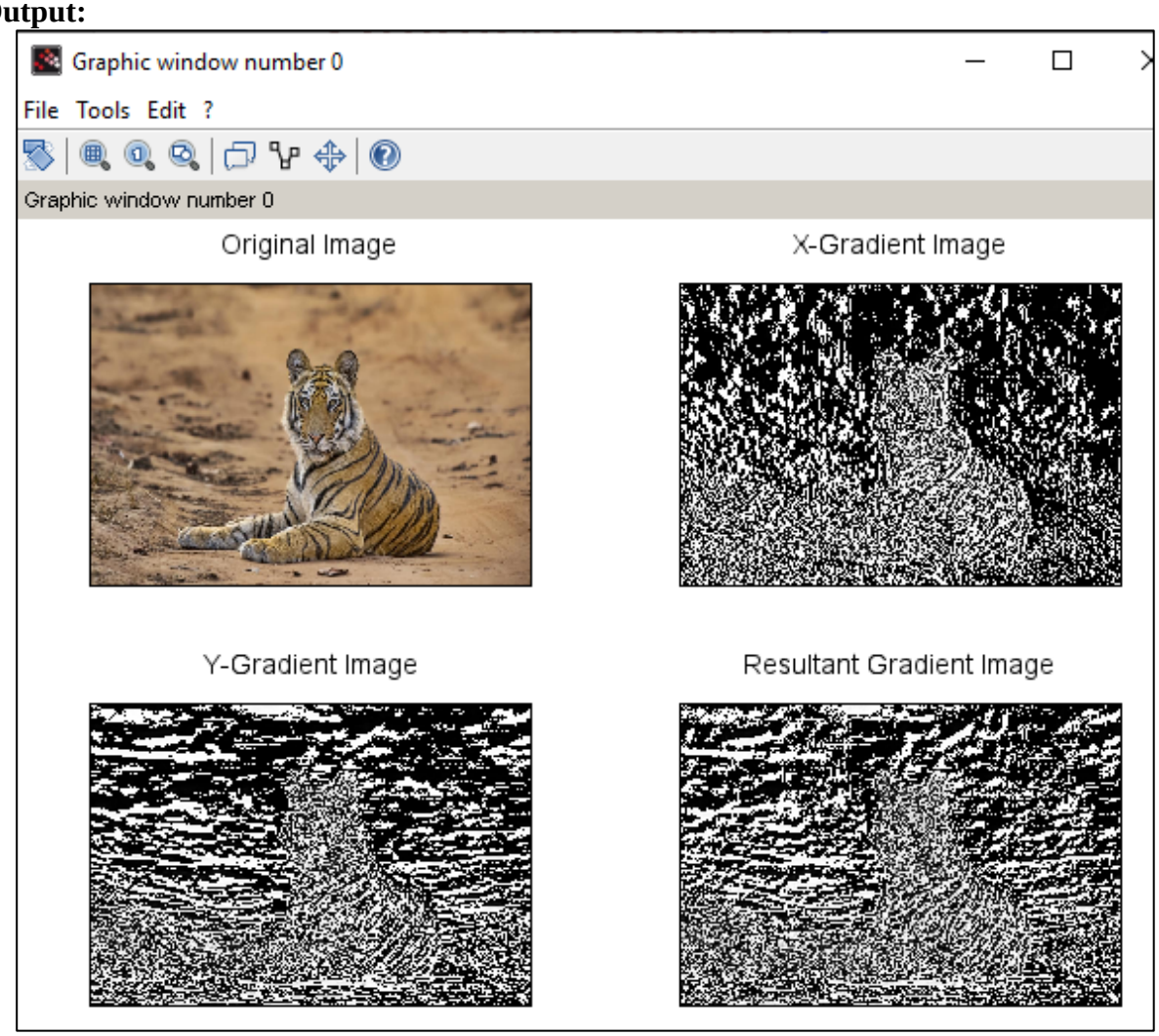


D. Program to apply morphological gradient on an image.

Code:

```
clear all;
close all;
aa=imread('D:\Basic Soft\tiger.jpg');
a=double(aa);
[row col]=size(a);
w1=[1 0;-1 0];
w2=[1 -1;0 0];
for x=2:1:row-1
 for y=2:1:col-1
a1(x,y)=w1(1)*a(x,y)+w1(2)*a(x,y+1)+w1(3)*a(x+1,y)+w1(4)*a(x+1,y
+1);
a2(x,y)=w2(1)*a(x,y)+w2(2)*a(x,y+1)+w2(3)*a(x+1,y)+w2(4)*a(x+1,y
+1);
 end
end
a3=a1+a2;
subplot(221),imshow(uint8(a)),title('Original Image');
subplot(222),imshow(uint8(a1)),title('X-Gradient Image');
subplot(223),imshow(uint8(a2)),title('Y-Gradient Image');
subplot(224),imshow(uint8(a3)),title('Resultant          Gradient
Image');
```

Output:

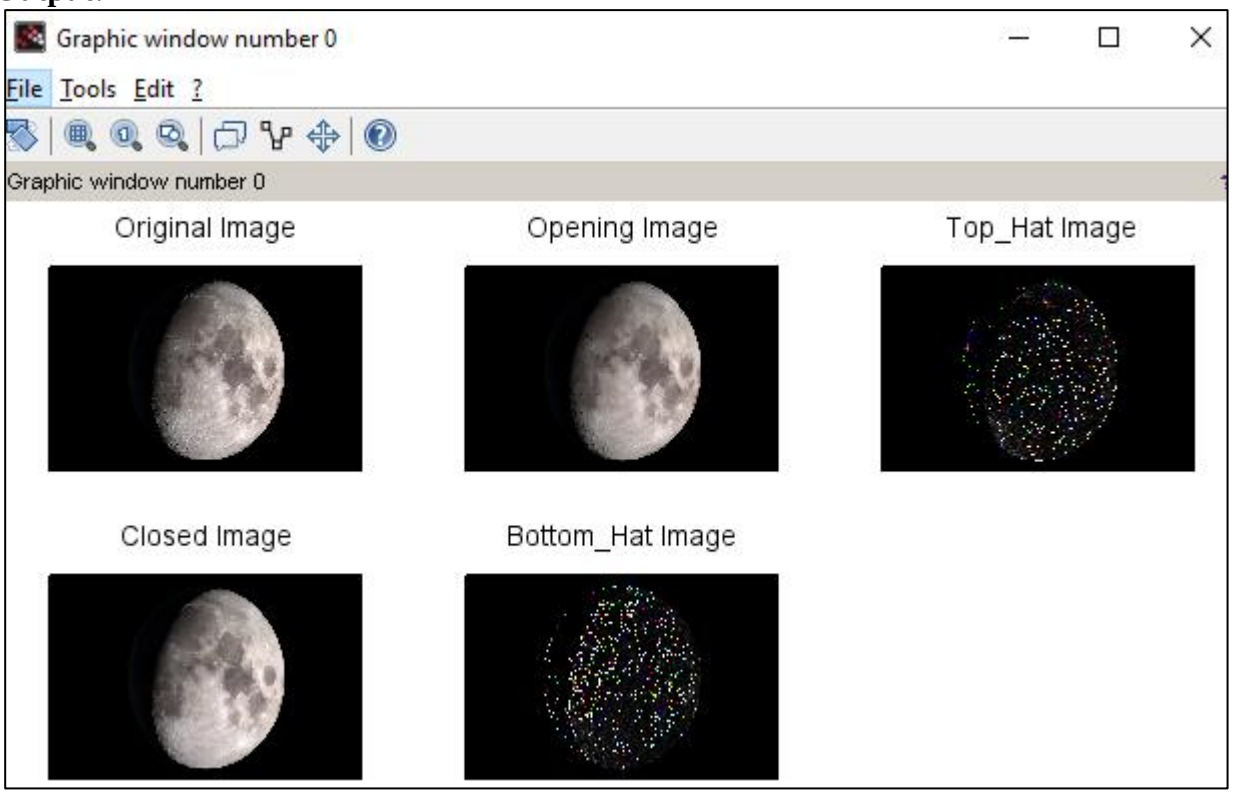


E. Program to apply Top-Hat/Bottom-Hat Transformations

Code:

```
clear;
clc;

a = imread('D:\MSC IT\Part I\Sem II\Image Processing\IP_Practical\IP_Practical_Images\moon.tif')
se = imcreatese('ellipse',10,10);
opening = imopen(a,se);
Top_Hat = a-opening
closed = imclose(a,se);
Bottom_Hat = closed-a
subplot(2,3,1);imshow(a);title("Original Image  ");
subplot(2,3,2);imshow(opening);title("Opening Image");
subplot(2,3,3);imshow(Top_Hat);title("Top_Hat Image");
subplot(2,3,4);imshow(closed);title("Closed Image");
subplot(2,3,5);imshow(Bottom_Hat);title("Bottom_Hat Image");
```

Output:



Practical 9

A. Program for Edge detection using.

i. Sobel

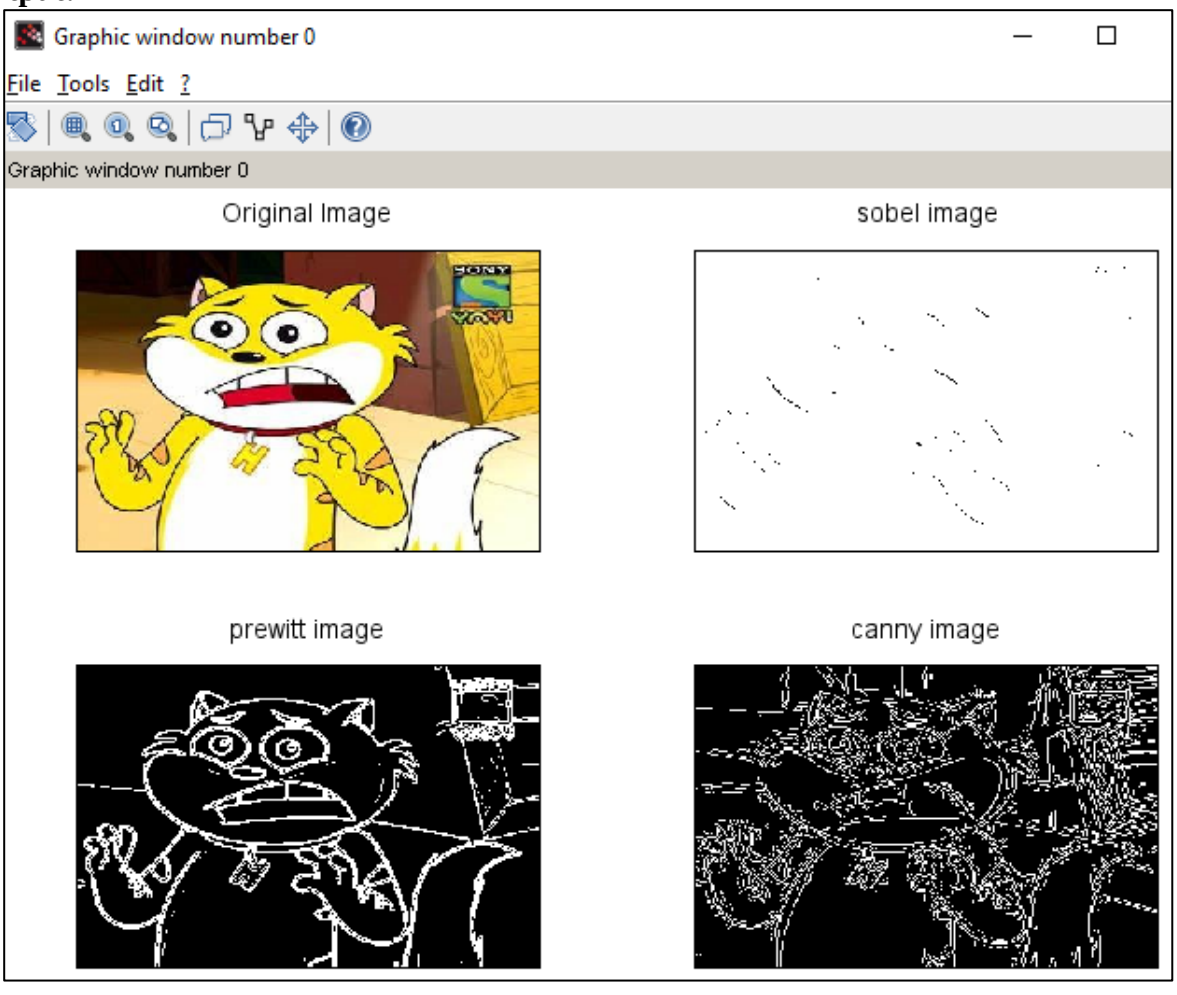
ii. Prewitt

iii. Canny

Code:

```
clc;
image=imread('D:\MSC IT\Part I\Sem II\Image
Processing\IP_Practical\Honey.jpg');
gray=rgb2gray(image);
//sobel operator
edge1 = edge(gray,'sobel');
//prewitt operator
edge2 = edge(gray,'prewitt');
//canny operator
edge3 = edge(gray,'canny');
subplot(2,2,1)
imshow(image);
title("Original Image ");
subplot(2,2,2);
imshow(edge1);
title('sobel image');
subplot(2,2,3);
imshow(edge2);
title('prewitt image');
subplot(2,2,4);
imshow(edge3);
title('canny image');
```

Output:

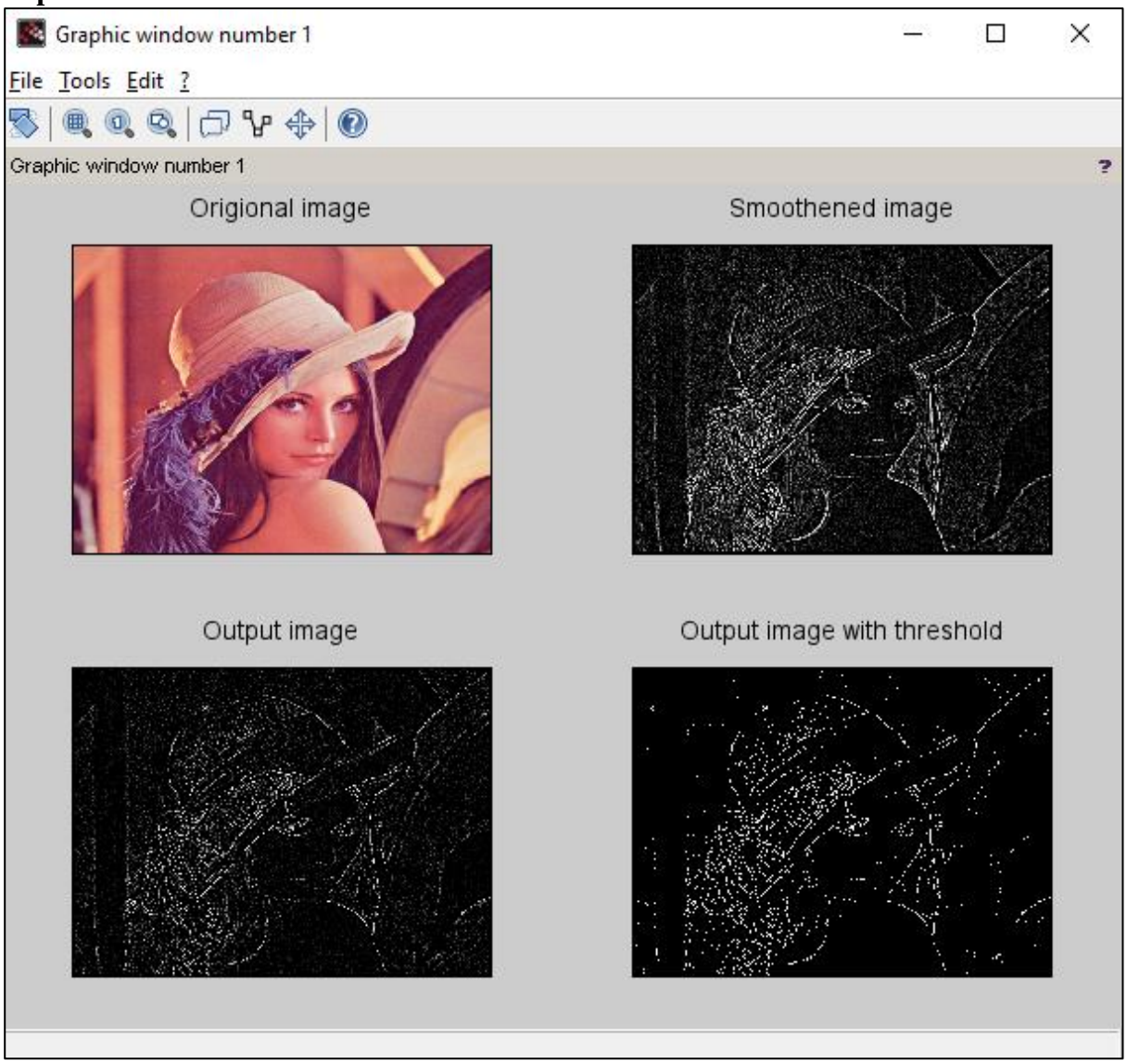


iv. Marr-Hildreth

Code:

```
clear all
im=imread('D:\MSC              IT\Part              I\Sem              II\Image
Processing\IP_Practical\IP_Practical_Images\lena.jpeg');
im=im2double(im);
gfilter= [0 0 1 0 0;
       0 1 2 1 0;
       1 2 -16 2 1;
       0 1 2 1 0;
       0 0 1 0 0];
smim=conv2(im,gfilter)
[rr,cc]=size(smim);
zc=zeros([rr,cc]);
for i=2:rr-1
    for j=2:cc-1
        if (smim(i,j)>0)
             if   (smim(i,j+1)>=0   &&   smim(i,j-1)<0)   ||   (smim(i,j+1)<0   &&
smim(i,j-1)>=0
                zc(i,j)= smim(i,j+1);

            elseif  (smim(i+1,j)>=0  &&  smim(i-1,j)<0)  ||  (smim(i+1,j)<0  &&
smim(i-1,j)>=0)
                    zc(i,j)= smim(i,j+1);
            elseif  (smim(i+1,j+1)>=0  &&  smim(i-1,j-1)<0)  ||  (smim(i+1,j+1)<0
&& smim(i-1,j-1)>=0)
                  zc(i,j)= smim(i,j+1);
            elseif  (smim(i-1,j+1)>=0  &&  smim(i+1,j-1)<0)  ||  (smim(i-1,j+1)<0
&& smim(i+1,j-1)>=0)
                  zc(i,j)=smim(i,j+1);
            end

        end

    end
end
otpt=im2uint8(zc);
otptth= otpt>105;
figure;
  subplot(2,2,1);imshow(im);title('Origional image');
  subplot(2,2,2);imshow(smim);title('Smoothened image');
  subplot(2,2,3);imshow(otpt);title('Output image');
  subplot(2,2,4);imshow(otptth);title('Output image with threshold');
   figure, imshow(otptth);
```

Output:



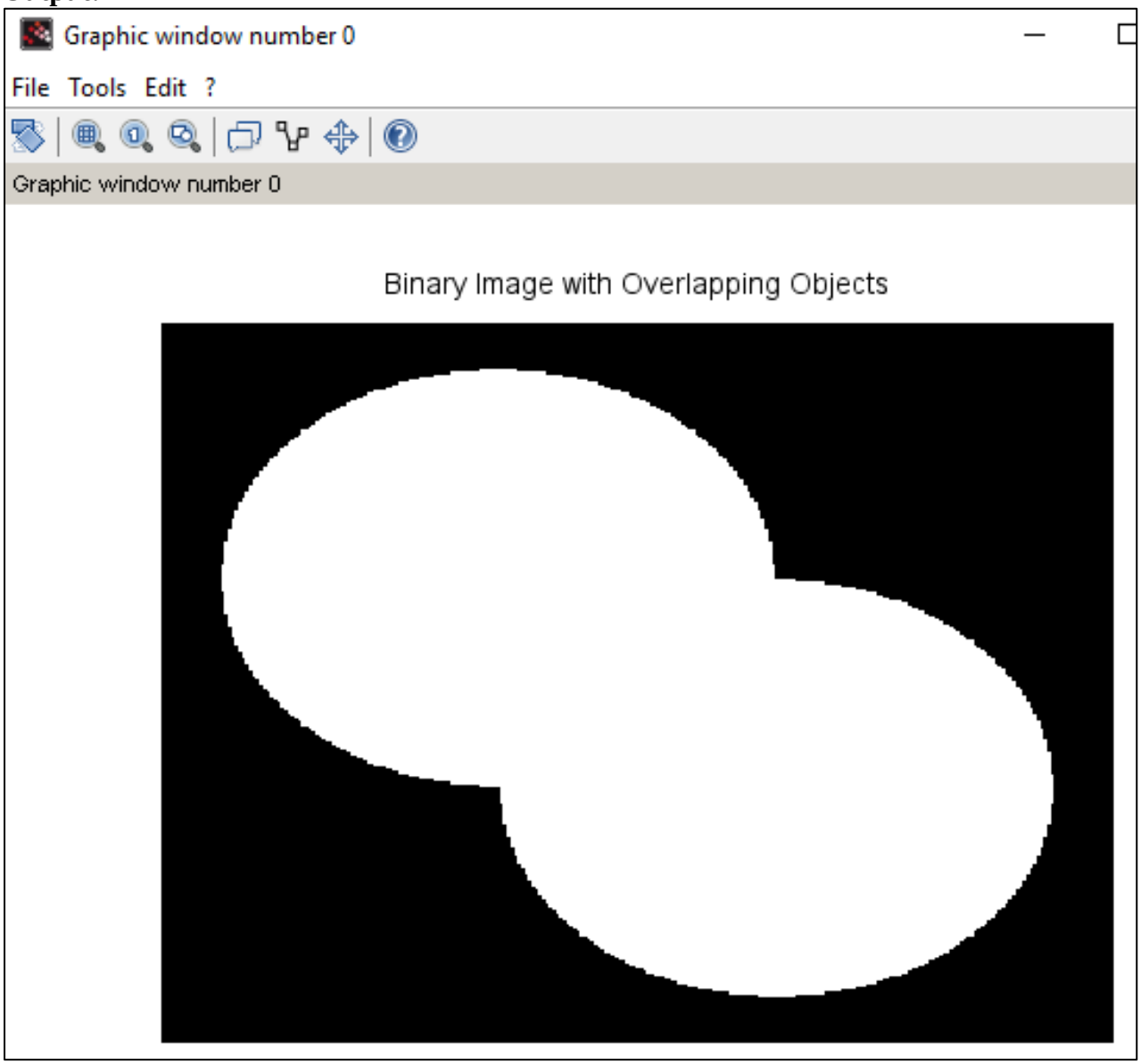
B. Illustrate Watershed segmentation algorithm.

Code:

```
clear all;
clc;
center1 = -40;
center2 = -center1;
dist = sqrt(2*(2*center1)^2);
radius = dist/2 * 1.4;
lims = [floor(center1-1.2*radius) ceil(center2+1.2*radius)];
[x,y] = meshgrid(lims(1):lims(2));
bw1 = sqrt((x-center1).^2 + (y-center1).^2) <= radius;
bw2 = sqrt((x-center2).^2 + (y-center2).^2) <= radius;
bw = bw1 | bw2;
imshow(bw)
title('Binary Image with Overlapping Objects')
D = bwdist(~bw);
imshow(D,[])
title('Distance Transform of Binary Image')
D = -D;
imshow(D,[])
title('Complement of Distance Transform')
L = watershed(D);
L(~bw) = 0;
```

```
rgb = label2rgb(L,'jet',[.5 .5 .5]);
imshow(rgb)
title('Watershed Transform')
```

Output:



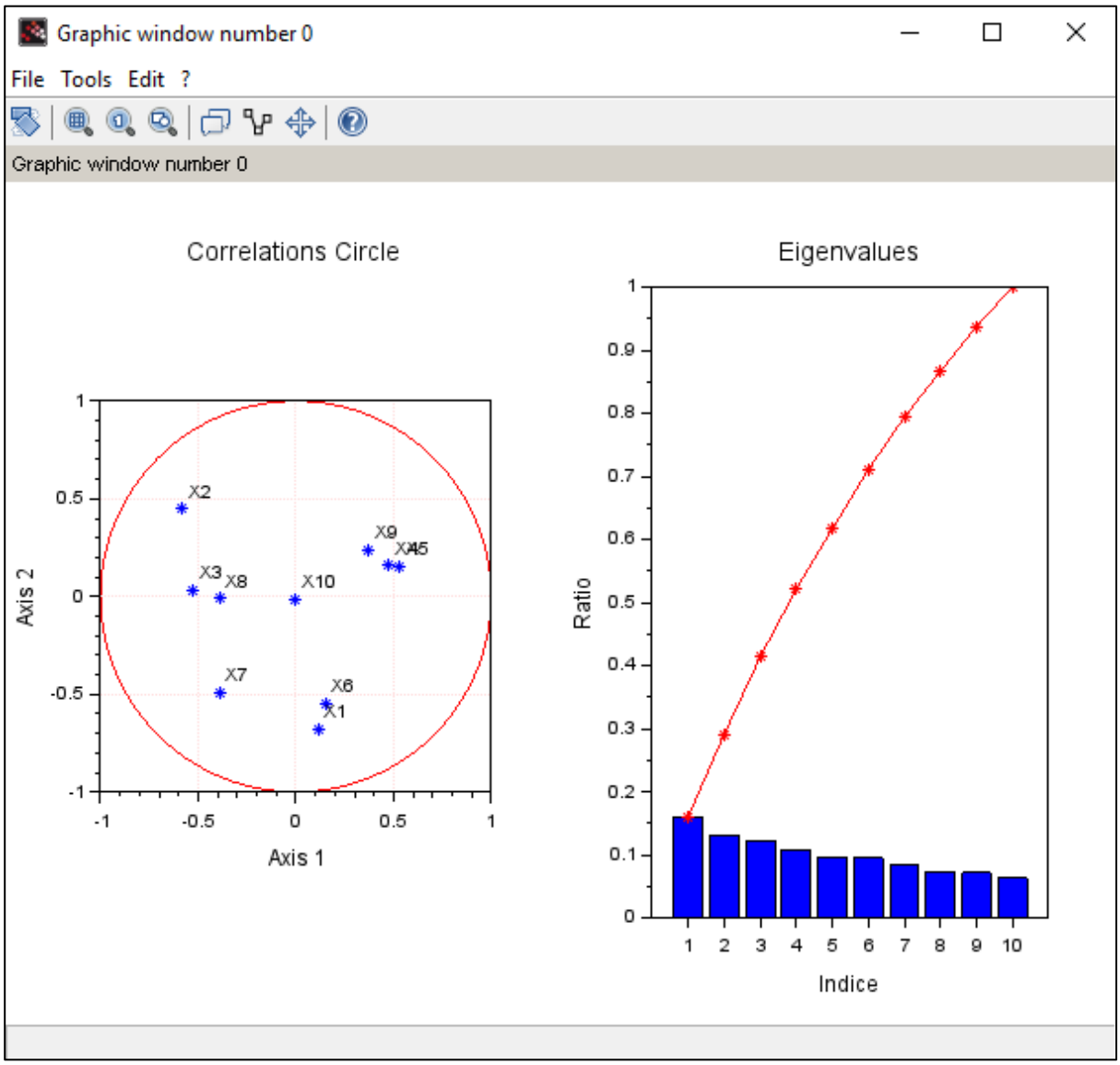
Practical 10

A. Principal components for image description

Code:

```
%MATLAB
a=rand(100,10,'n');
[lambda,facpr,comprinc]=pca(a);
show_pca(lambda,facpr)
```

Output:



B. Apply Harris-Stephen's corner detector algorithm.

Code:

```
%MATLAB
Image_in = checkerboard;
cornerDetector = detectHarrisFeatures(Image_in);
imshow(Image_in);
hold on;
plot(cornerDetector.selectStrongest(50));
```

Output: