**Lab : Fourier Analysis and Image Filtering.**

**Exercise 1.**

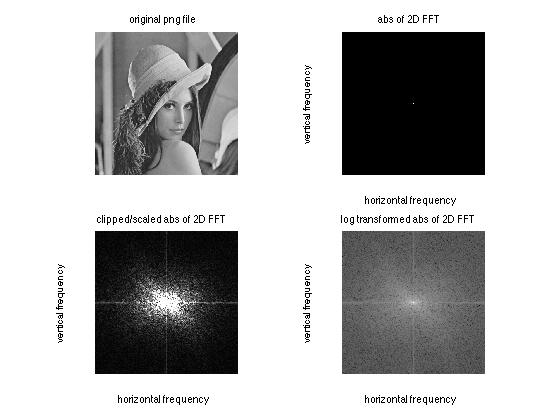
Using the image file **lena.png,** read the image and compute the 2-D DFT magnitude of the image. On a single plot (using a grid of 2 x 2 images), plot the following:

* the original gray scale image in the upper left cell
* the image representation of the 2-D DFT magnitude of the image being studied in the

upper right cell

* a clipped and scaled version of the 2-D DFT magnitude that is clipped and scale at a level where you can see the nature of the 2-D DFT magnitude; this plot should be placed in the lower left cell
* the log transformed 2-D DFT magnitude plotted again on a scale that enables you to see the structure of the transform magnitude; this plot should be placed in the lower right cell

It should look like this  for each image



**Exercice 2 : Phase matters !!!**

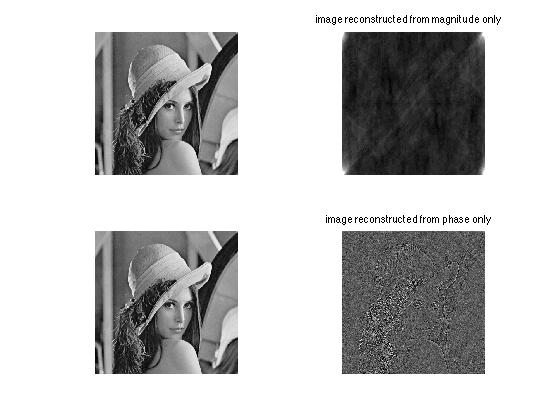
**A)** **Write a Matlab program to compute** the magnitude only and phase only versions

of the two gray scale images, Barbara.png and lena.png, and plot the results in a 2 x

2 grid with the following format:

* the upper left and lower left images should be the original gray scale image files (so that you can compare them to the transformed images directly).
* the upper right image should be the image reconstructed from magnitude only information
* the lower right image should be the image reconstructed from phase only information

It should look like this  for ‘Lena.png’:



**(B)** Next perform the following experiment.

Compute the 2-D DFTs of the two images used in part (a) of this exercise. From these transforms, form two new DFTs in which the magnitudes and phases are interchanged. Compute the corresponding two images and display them, along with the original gray scale images, in a 2 x 2 grid of the form:

* the upper left image should be the first original image
* the upper right image should be the image reconstructed with the cross magnitude and the correct image phase signal.
* the lower left image should be the second original image
* the lower right image should be the image reconstructed with the cross magnitude and the correct image phase signal.

It should look like this



**Exercise 3**Linear Filtering of Images

(a) Using the Matlab (see appendix) routine H=lpfilter(type, M, N, D0, n) for designing lowpass filters, where the input arguments are:

type='gaussian', 'ideal' or 'btw' (for Butterworth filters)

M,N = dimensionality of filter frequency response

D0 = filter cutoff frequency (normalized to range [0, M] or [0

N])

n = filter order for Butterworth filters

and the output H is the frequency response of the filter,

**write a Matlab m-file** that accepts as input the filter type (ftype='gaussian','ideal','btw'), the cutoff frequency D0, and (optionally for the Butterworth filter) the filter order n.

Within the Matlab program plot the filter frequency response (**using the mesh command**) along with the filter impulse response.

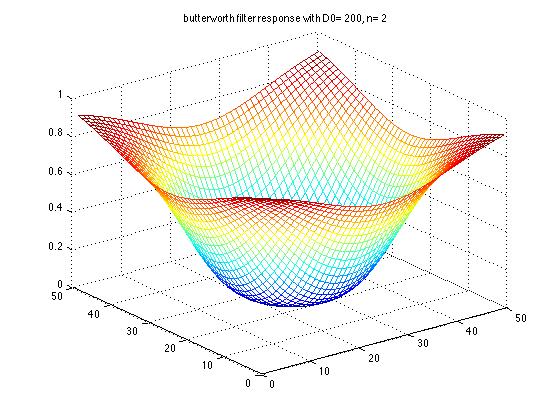
Test your program for the following conditions:

1. ftype='gaussian', D0=50

2. ftype='ideal', D0=50

3. ftype='btw', n=4, D0=50

It should loo like that  (here different btw):



We can now use the filters designed above for filtering an image file. We use the file lena.png for this exercise. The Matlab code for choosing the appropriate lowpass filter and fitering an image file (in the frequency domain) is given in the Matlab code below:

% lp\_hp\_filter\_image

%

% read in png image lena and size the image

f=imread('lena.png');

[M,N]=size(f);

% pad image for convolution

PQ=paddedsize(size(f));

% enter filter type and design parameters

D0=50; % cutoff frequency (range 0 to 250)

D0=input('value for cutoff frequency (range 0 to 250):');

ftype=input('filter type (gaussian, ideal, btw (butterworth)):','s');

if (ftype(1:3) == 'gau')

Hp=lpfilter('gaussian',PQ(1),PQ(2),D0);

elseif (ftype(1:3) == 'ide')

Hp=lpfilter('ideal',PQ(1),PQ(2),D0);

elseif (ftype(1:3) == 'btw')

n=input('butterworth filter order (1 to 6):');

Hp=lpfilter('btw',PQ(1),PQ(2),D0,n);

else

error('improper filter type specified');

end

ilphp=input('lowpass (1) or highpass (0) filter:');

if (ilphp == 0) Hp=1-Hp;

end

% transform image to frequency domain using zero-padded image

Fp=fft2(f,PQ(1),PQ(2));

% display impulse response of lowpass filter using mesh plot

H=fftshift(Hp);

figure,mesh(H(1:10:PQ(1), 1:10:PQ(2)));

axis([0 round(PQ(1)/10) 0 round(PQ(2)/10) 0 1]);

% filter in frequency domain by multiplying 2-D FFTs

Gp=Hp.\*Fp;

% convert back to image plane, using real value of inverse transform

gp=real(ifft2(Gp));

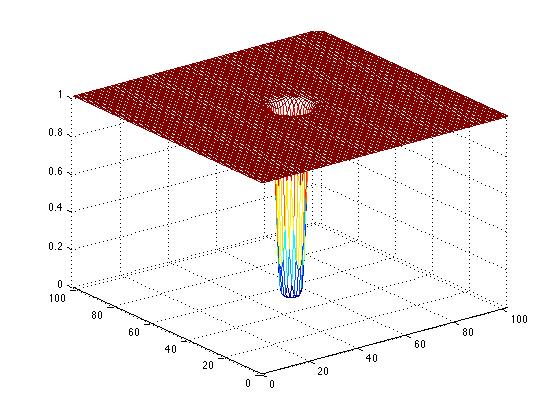
% crop image to original size and display uncropped and cropped image

gpc=gp(1:size(f,1),1:size(f,2));

figure,imshow(gp,[ ]);

figure,imshow(gpc,[ ]);

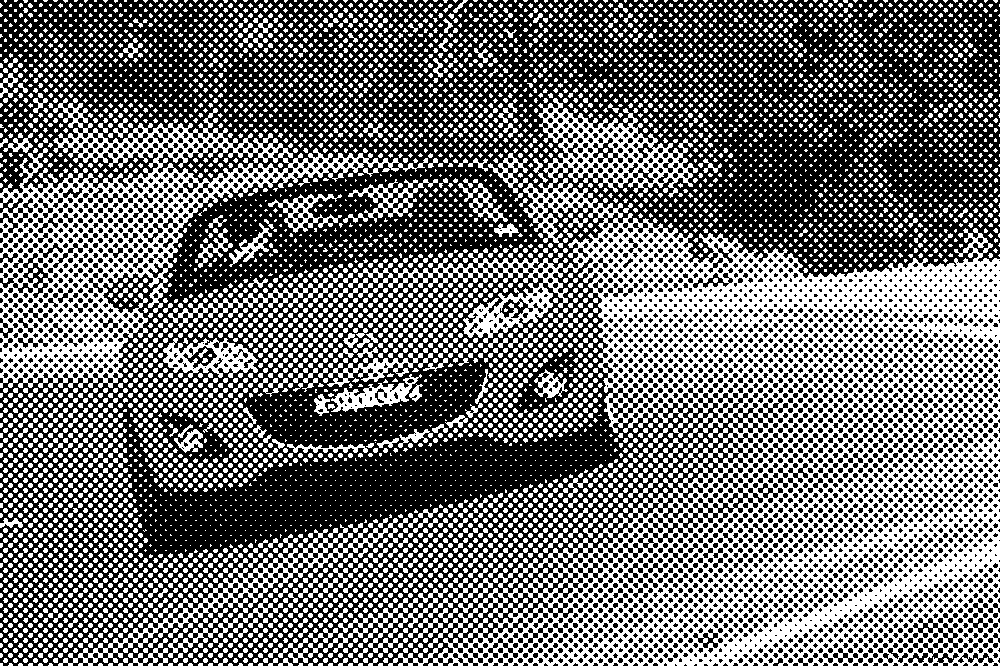
It should look like this :



**Exercise 4**Image Enhancement, CSI !!

You are the image processing expert for a local police department. A detective has reopened a cold case and part of his evidence is a newpaper print of a car. You have been asked to do some CSI style magic to see if you can learn the suspect's license plate number or see his face.

Download the image "**halftone.png**" and store it in MATLAB's "Current Directory".



Use a set of notch filters to remove the peaks from the image's Fourier transform.

***TIPS:***

create a function that takes a list of peaks as an argument the peaks form a repetitive pattern. Figure out the pattern to save time.

Fine tune your results by trying varying widths of the three notch filter types. Provide a best effort

One of the possible solutions :

****

**Appendix**

**How the Mesh and Images were Created:**

The following MATLAB code relies on M-Files: [dftuv.m](http://www.cs.uregina.ca/Links/class-info/425/Lab5/M-Functions/dftuv.m), [lpfilter.m](http://www.cs.uregina.ca/Links/class-info/425/Lab5/M-Functions/lpfilter.m), and [hpfilter.m](http://www.cs.uregina.ca/Links/class-info/425/Lab5/M-Functions/hpfilter.m)

**Lowpass Filters**

1. **Ideal**
   * HLPF\_ideal=fftshift(lpfilter('ideal', 500, 500, 50));
   * mesh(HLPF\_ideal(1:10:500,1:10:500))
   * colormap([0 0 0])
   * axis off
   * grid off
   * axis([0 50 0 50 0 1])
   * figure, imshow(HLPF\_ideal)
2. **Butterworth**
   * HLPF\_btw=fftshift(lpfilter('btw', 500, 500, 50));
   * figure, mesh(HLPF\_btw(1:10:500,1:10:500))
   * colormap([0 0 0])
   * axis off
   * grid off
   * axis([0 50 0 50 0 1])
   * figure, imshow(HLPF\_btw)
3. **Gaussian**
   * HLPF\_gauss=fftshift(lpfilter('gaussian', 500, 500, 50));
   * figure, mesh(HLPF\_gauss(1:10:500,1:10:500))
   * colormap([0 0 0])
   * axis off
   * grid off
   * axis([0 50 0 50 0 1])
   * figure, imshow(HLPF\_gauss)

**Highpass Filters**

1. **Ideal**
   * HHPF\_ideal=fftshift(hpfilter('ideal', 500, 500, 50));
   * mesh(HHPF\_ideal(1:10:500,1:10:500))
   * colormap([0 0 0])
   * axis off
   * grid off
   * axis([0 50 0 50 0 1])
   * figure, imshow(HHPF\_ideal)
2. **Butterworth**
   * HHPF\_btw=fftshift(hpfilter('btw', 500, 500, 50));
   * figure, mesh(HHPF\_btw(1:10:500,1:10:500))
   * colormap([0 0 0])
   * axis off
   * grid off
   * axis([0 50 0 50 0 1])
   * figure, imshow(HHPF\_btw)
3. **Gaussian**
   * HHPF\_gauss=fftshift(hpfilter('gaussian', 500, 500, 50));
   * figure, mesh(HHPF\_gauss(1:10:500,1:10:500))
   * colormap([0 0 0])
   * axis off
   * grid off
   * axis([0 50 0 50 0 1])
   * figure, imshow(HHPF\_gauss)

function [U, V] = dftuv(M, N)

%DFTUV Computes meshgrid frequency matrices.

% [U, V] = DFTUV(M, N) computes meshgrid frequency matrices U and

% V. U and V are useful for computing frequency-domain filter

% functions that can be used with DFTFILT. U and V are both M-by-N.

% Set up range of variables.

u = 0:(M-1);

v = 0:(N-1);

% Compute the indices for use in meshgrid

idx = find(u > M/2);

u(idx) = u(idx) - M;

idy = find(v > N/2);

v(idy) = v(idy) - N;

% Compute the meshgrid arrays

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

function H = hpfilter(type, M, N, D0, n)

%HPFILTER Computes frequency domain highpass filters

% H = HPFILTER(TYPE, M, N, D0, n) creates the transfer function of

% a highpass filter, H, of the specified TYPE and size (M-by-N).

% Valid values for TYPE, D0, and n are:

%

% 'ideal' Ideal highpass filter with cutoff frequency D0. n

% need not be supplied. D0 must be positive

%

% 'btw' Butterworth highpass filter of order n, and cutoff D0.

% The default value for n is 1.0. D0 must be positive.

%

% 'gaussian' Gaussian highpass filter with cutoff (standard deviation)

% D0. n need not be supplied. D0 must be positive.

%

% The transfer function Hhp of a highpass filter is 1 - Hlp,

% where Hlp is the transfer function of the corresponding lowpass

% filter. Thus, we can use function lpfilter to generate highpass

% filters.

if nargin == 4

n = 1; % Default value of n.

end

% Generate highpass filter.

Hlp = lpfilter(type, M, N, D0, n);

H = 1 - Hlp;

function H = lpfilter(type, M, N, D0, n)

%LPFILTER Computes frequency domain lowpass filters

% H = LPFILTER(TYPE, M, N, D0, n) creates the transfer function of

% a lowpass filter, H, of the specified TYPE and size (M-by-N). To

% view the filter as an image or mesh plot, it should be centered

% using H = fftshift(H).

%

% Valid values for TYPE, D0, and n are:

%

% 'ideal' Ideal lowpass filter with cutoff frequency D0. n need

% not be supplied. D0 must be positive

%

% 'btw' Butterworth lowpass filter of order n, and cutoff D0.

% The default value for n is 1.0. D0 must be positive.

%

% 'gaussian' Gaussian lowpass filter with cutoff (standard deviation)

% D0. n need not be supplied. D0 must be positive.

% Use function dftuv to set up the meshgrid arrays needed for

% computing the required distances.

[U, V] = dftuv(M, N);

% Compute the distances D(U, V).

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

% Begin fiter computations.

switch type

case 'ideal'

H = double(D <=D0);

case 'btw'

if nargin == 4

n = 1;

end

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

case 'gaussian'

H = exp(-(D.^2)./(2\*(D0^2)));

otherwise

error('Unknown filter type.')

end