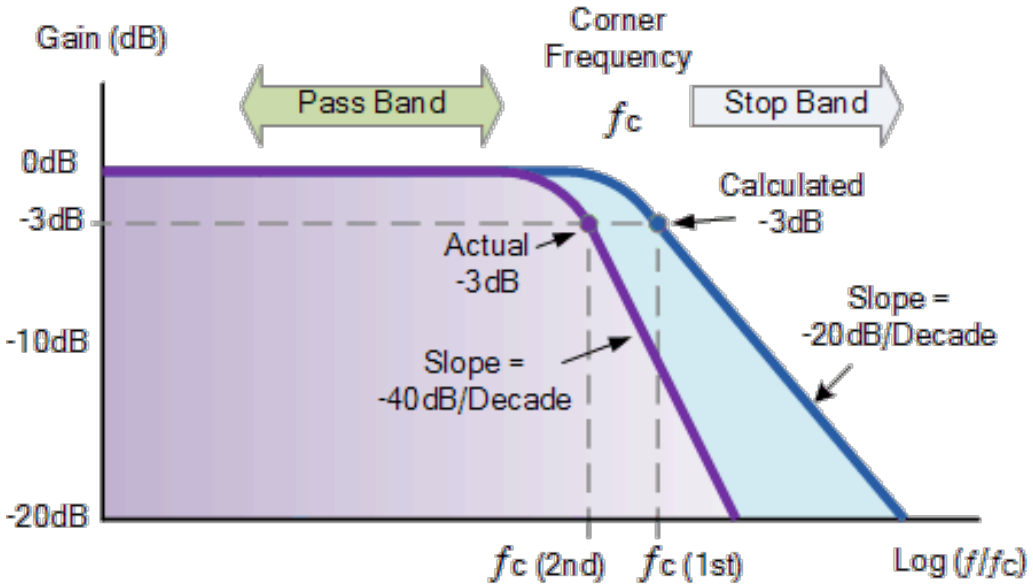
EBME 361
Jamie Yu
03/07/18

Homework 6: Fourier Transforms

1.

a. This is a low-pass filter. We can tell because low-frequency components are shown unaffected near the center of the amplitude spectrum in the Fourier domain while the outer data is much depressed. When we apply a low-pass filter in the Fourier domain, much of the information about the edges and contrast is removed.

b. The generic plot of the transfer function of a low-pass filter can be seen below:



As we can see, the cutoff frequency is represented as f_c, and for a first-order low-pass filter, the slope of the stop-band is -20 dB/Decade with a -3dB drop at f_c.

c.

i. The cutoff occurs at approximately 10mm since the diameter of the circle of unaffected data is 20mm.

ii. There are 36 cycles (36 black lines and 36 white lines)

iii. $36\ cycles * \frac{0.5\frac{mm}{pixel}}{10mm} = \mathbf{1.8\ cycles/pixel}$

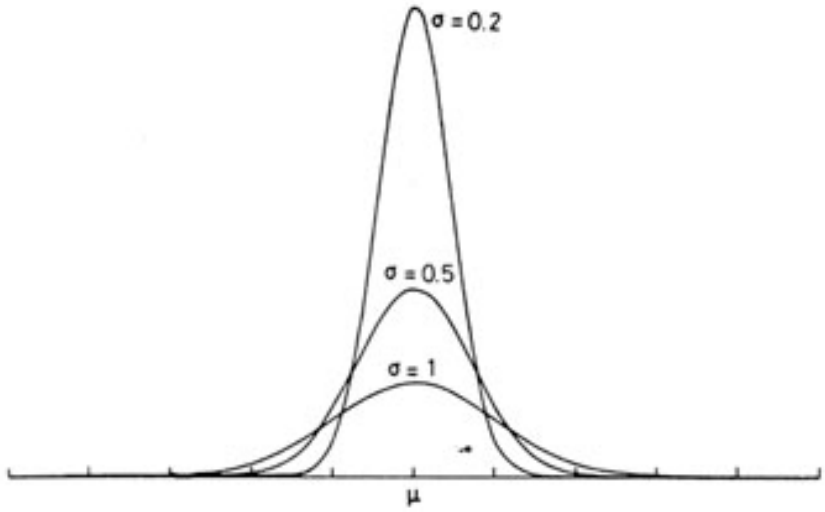
2.

a. I wrote a separate helper function to apply an ideal low-pass filter to an image at a specified cutoff frequency. I performed this by taking the specified cutoff frequency and constructing an ideal filter consisting of a circle of 0s (outside the cutoff frequency) and 1s (inside the cutoff frequency). According to the convolution theorem, multiplication in the frequency domain is equivalent to convolution in the time domain, so I multiplied this filter by the frequency domain image to apply the filter. The code used can be found below:

```
function [H, filteredImageDFT, filteredImage] =
idealLPF(image,f_c)
    % Get image dimensions
    [M, N] = size(image);

    % Transform image to Fourier Domain.
    imageDFT = fft2(double(image));

    % Initialize new set of point coordinates.
    u = 0:(M-1);
    v = 0:(N-1);

    % Find indices of x- and y- coordinates outside
half the dimensions.
    idx = find(u > M/2);
    idy = find(v > N/2);
    u(idx) = u(idx) - M;
    v(idy) = v(idy) - N;

    % Create a grid from the new coordinates.
    [V,U] = meshgrid(v,u);

    % Use the formula of a circle to construct the
logical bounds of the
    % filter.
    D = sqrt(U.^2+V.^2);

    % Construct the filter by locating points that are
within the radius of
    % the cutoff frequency.
    H = double(D <= f_c);

    % Apply the filter by multiplying with the image
in Fourier Domain
    % (which is equal to convolving in the Time
Domain).
    filteredImageDFT = H .* imageDFT;

    % Transform the image back into the Time Domain.
    filteredImage =
real(ifft2(double(filteredImageDFT)));
end
```

Similar to part a., I created a separate helper function to apply a Gaussian low-pass filter. I used the fspecial() function in MATLAB to construct a filter of type 'gaussian'. For this problem, the half-width at half maximum amplitude was specified as 1/3 the maximum frequency. The sigma value specified in MATLAB can then specify the width of the Gaussian distribution (see figure below). I used

the calculated maximum frequencies from part a. to specify the sigma values for this section. The code can be found below:



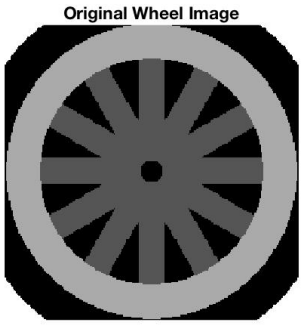

```
%% Problem 2
close all; clear; clc;

% Load the images.
wheel = imread('wheel.tif');
edges = rgb2gray(imread('edges.tif'));
[m1, n1] = size(wheel);

wheelFT = fft2(double(wheel));
edgesFT = fft2(double(edges));

% a. Apply an ideal lpf at a frequency which is 1/3
the maximum frequency.
% Maximum frequency occurs at +/- N/2 in each dir. of
an unshifted NxN DFT.
magWheelFT = abs(wheelFT);
magEdgesFT = abs(edgesFT);
maxFreqWheel = magWheelFT(m1/2, n1/2);
maxFreqEdges = magEdgesFT(m1/2, n1/2);

% Compute cutoff frequencies as 1/3 the maximum
frequencies.
fcWheel = 1/3 * maxFreqWheel;
fcEdges= 1/3 * maxFreqEdges;

% Construct an ideal low-pass filter which is a circle
shape of 0s and 1s.
[~, lpfWheelDFT, lpfWheel] = idealLPF(wheel, fcWheel);
[edgesH, lpfEdgesDFT, lpfEdges] = idealLPF(edges,
fcEdges);
```

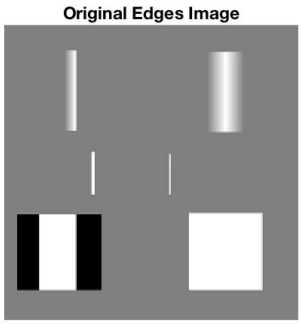
```
% b. Apply a Gaussian lpf with a 1/2 width at half
maximum amplitude which is
% 1/3 the maximum frequency.
[~, gaussWheelDFT, gaussWheel] = gaussLPF(wheel,
fcWheel);
[gaussEdgesH, gaussEdgesDFT, gaussEdges] =
gaussLPF(edges, fcEdges);

% c. Print out these images for the report.
% i. Original images in the cartesian domain.
fig1 = figure(1);
subplot(1,2,1); imshow(wheel, []); title('Original
Wheel Image');
subplot(1,2,2); imshow(edges, []); title('Original
Edges Image');
saveas(fig1, 'fig1_hw6.jpg');

% ii. Original images in the frequency domain.
fig2 = figure(2);
subplot(1,2,1); imshow(log(abs(fftshift(wheelFT))),
[]); title('Wheel Image in FD');
subplot(1,2,2); imshow(log(abs(fftshift(edgesFT))),
[]); title('Edges Image in FD');
saveas(fig2, 'fig2_hw6.jpg');

% iii. Images with Ideal LPF in cartesian domain.
fig3 = figure(3);
subplot(1,2,1); imshow(lpfWheel, []); title('Ideal LPF
Wheel Image');
subplot(1,2,2); imshow(lpfEdges, []); title('Ideal LPF
Edges Image');
saveas(fig3, 'fig3_hw6.jpg');

% iv. Images with Ideal LPF in frequency domain.
fig4 = figure(4);
subplot(1,2,1);
imshow(log(abs(fftshift(lpfWheelDFT))), []);
title('Ideal LPF Wheel Image in FD');
subplot(1,2,2);
imshow(log(abs(fftshift(lpfEdgesDFT))), []);
title('Ideal LPF Edges Image in FD');
saveas(fig4, 'fig4_hw6.jpg');

% v. Images with Gaussian LPF in cartesian domain.
fig5 = figure(5);
subplot(1,2,1); imshow(gaussWheel, []); title('Gauss
LPF Wheel Image');
subplot(1,2,2); imshow(gaussEdges, []); title('Gauss
```

```
LPF Edges Image');
saveas(fig5, 'fig5_hw6.jpg');

% vi. Images with Gaussian LPF in frequency domain.
fig6 = figure(6);
subplot(1,2,1);
imshow(log(abs(fftshift(gaussWheelDFT))), []);
title('Gauss LPF Wheel Image in FD');
subplot(1,2,2);
imshow(log(abs(fftshift(gaussEdgesDFT))), []);
title('Gauss LPF Edges Image in FD');
saveas(fig6, 'fig6_hw6.jpg');

% vii. Ideal LPF in frequency domain.
fig7 = figure(7);
imshow(fftshift(edgesH)); title('Ideal Low-Pass
Filter');
saveas(fig7, 'fig7_hw6.jpg');

% viii. Gaussian LPF in frequency domain.
fig8 = figure(8);
imshow(gaussEdgesH); title('Gaussian Low-Pass
Filter');
saveas(fig8, 'fig8_hw6.jpg');
```

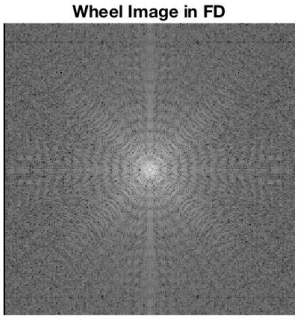
b.

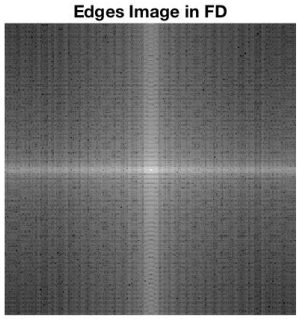
i.



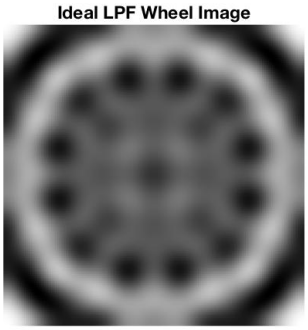


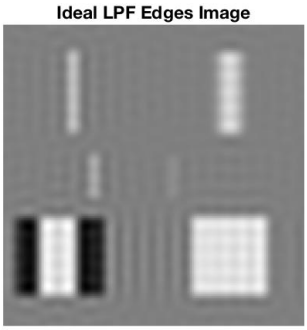
ii.



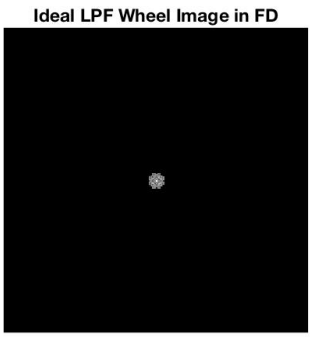


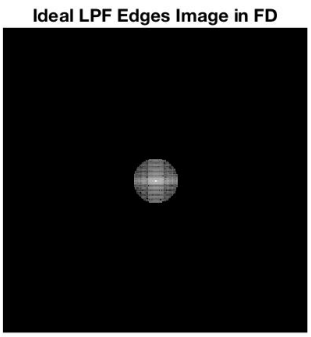
iii.


Ideal LPF Wheel Image

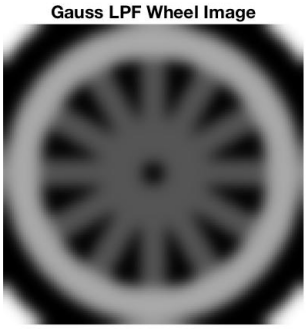

Ideal LPF Edges Image

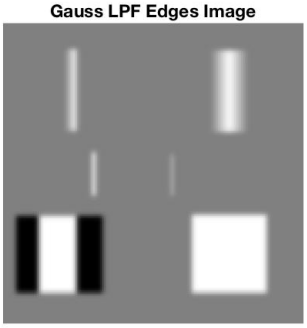
iv.


Ideal LPF Wheel Image in FD

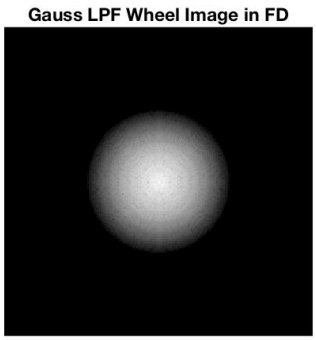

Ideal LPF Edges Image in FD

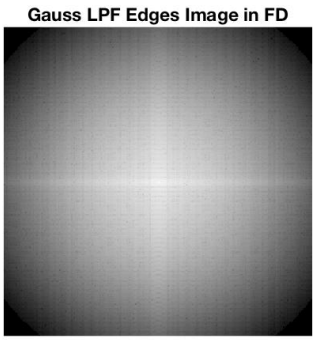
v.


Gauss LPF Wheel Image

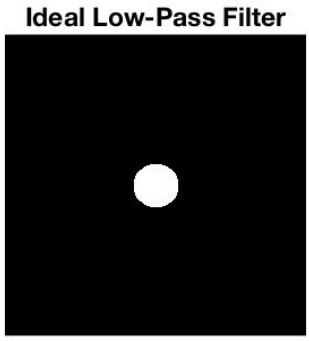

Gauss LPF Edges Image

vi.

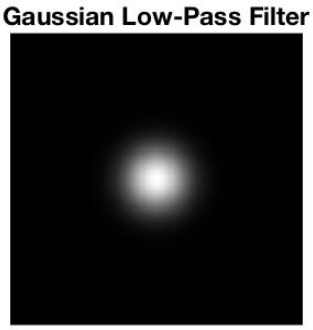

Gauss LPF Wheel Image in FD


Gauss LPF Edges Image in FD

vii.



viii.



c. From comparing the two filtered images, we find that the ideal LPF produces ringing artifacts more than the Gaussian LPF. This is clear to see in both the wheel edges images (part b, image iii). This makes sense because an ideal LPF in the time domain is a sinc function, and as the sinc function extends towards negative and positive infinity, ringing can be observed (it does not go to 0 instantaneously). In contrast, the Gaussian LPF is non-negative and non-oscillatory, thus causes no ringing artifacts.

   Additionally, the ideal LPF produces more smoothing and blurring. This occurs simultaneously due to ringing artifact mentioned above. The narrower the filter in the frequency domain, the more severe the blurring and ringing results.

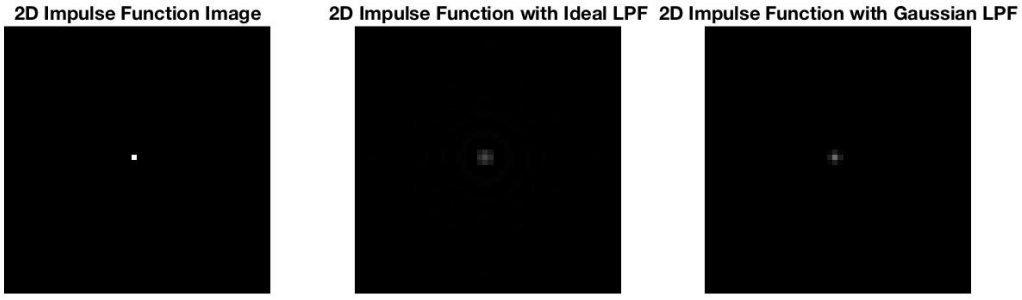
d. The high frequencies in these images are the sharp edges and areas of high contrast. In the wheel image, this would be the edges between the wheel spokes and the borders of the circle. In the edges image, this would be all of the distinctive edges against the background.

e. To demonstrate what happens to a point spread function, I coded the following below:

```
%% f. Apply an ideal LPF and a Gaussian LPF to an
image consisting of a
```

```
% single white dot on a black background.

% Create a 256x256 black background image.
dotImg = zeros(50, 50);

% Set the center pixel to a white dot.
dotImg(50/2, 50/2) = 1;

fig0 = figure(13);
subplot(1,3,1); imshow(dotImg); title('2D Impulse
Function Image');

% (i) Apply an ideal LPF (select a cutoff frequency of
20)
fc = 15;
[~, ~, lpfDotImg] = idealLPF(dotImg, fc);
figure(13);
subplot(1,3,2); imshow(lpfDotImg); title('2D Impulse
Function with Ideal LPF');

% (ii) Apply a Gaussian LPF.
[~, ~, gaussDotImg] = gaussLPF(dotImg, fc);
figure(13)
subplot(1,3,3); imshow(gaussDotImg); title('2D Impulse
Function with Gaussian LPF');
saveas(fig0, 'fig13_hw6.jpg');
```

The resulting images were found:



From observing the effects of both filters, we see that the ideal LPF spreads the impulse and blurs it significantly more than the Gaussian. Additionally, the intensity (amplitude) of the pixel is greatly reduced. This make sense because an impulse function has high frequency content in the form of high contrast and sharp edges. When we apply both filters, the ideal LPF will affect the impulse more than the Gaussian LPF because its truncation has a sharper cutoff at the cutoff frequency, and less high frequencies remain in the image after filtering.

3.

a. Below is the code I used to convert the images to grayscale format.

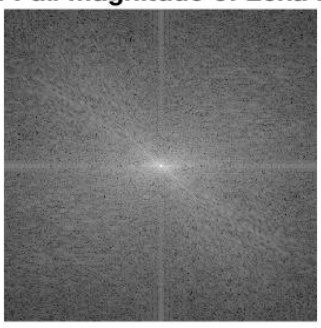
```
%% Problem 3
```

```
close all; clear; clc;

lena = imread('lena.bmp');
iris = imread('iris-illustration.bmp');

% Convert images to grayscale format.
lena = rgb2gray(lena);
iris = rgb2gray(iris);
```

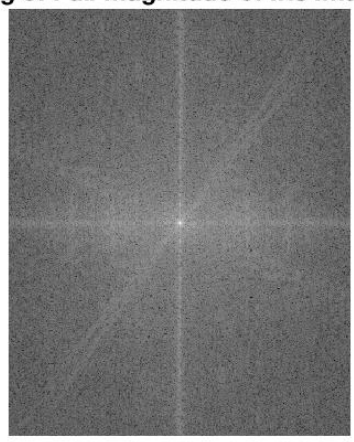
b. I found the 2D DFT of the grayscale images using fft2() in MATLAB. Next, I used fftshift() to center shift the plot, and lastly took the magnitude and log using abs() and log(). The following code produced the images shown below:

```
% Calculate DFT.
lenaDFT = fft2(lena);
irisDFT = fft2(iris);

% b. Plot the log magnitude of the 2D DFT of the
grayscale image, with
% center shifted.
fig8 = figure(8);
subplot(1,2,1); imshow(log(abs(fftshift(lenaDFT))),
[]); title('Log of Magnitude of Lena Image');
subplot(1,2,2); imshow(log(abs(fftshift(irisDFT))),
[]); title('Log of Magnitude of Iris Image');
saveas(fig8, 'hw6_fig8.jpg');
```

Log of Full Magnitude of Lena Image



Log of Full Magnitude of Iris Image

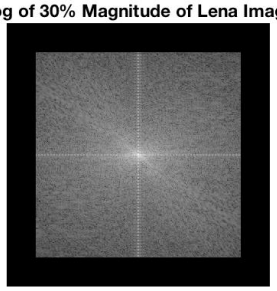


c. I applied the truncation windows by capturing only 30%, 15%, and 5% of the DFT magnitudes. To calculate this, I first found the x- and y- coordinates representing the ranges in which we want to capture the DFT (centered around the centered, shifted signal). I did so by solving a triangle equation to find the four corners of a rectangle scaled to 30%, 15%, and 5% within the original dimensions. Once I found the x- and y-coordinates for each truncation amount, I constructed

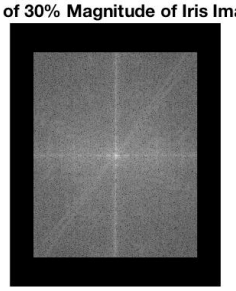
truncation windows, which consisted of 0s and 1s, and multiplied these windows by the original DFTs to produce the truncated spectrums.

The following truncated spectrums are shown below with the code used to create them:

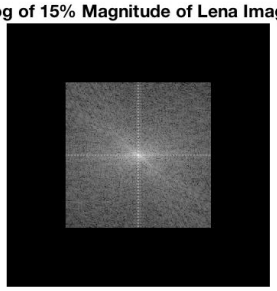
Log of 30% Magnitude of Lena Image



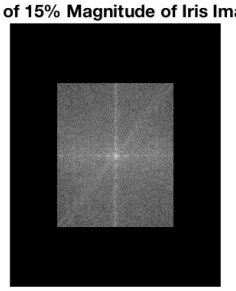
Log of 30% Magnitude of Iris Image



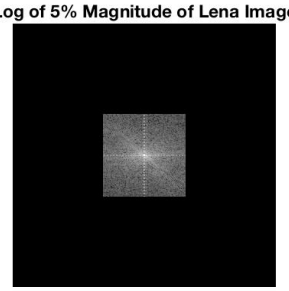
Log of 15% Magnitude of Lena Image



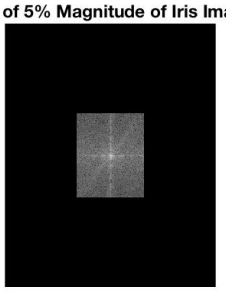
Log of 15% Magnitude of Iris Image



Log of 5% Magnitude of Lena Image



Log of 5% Magnitude of Iris Image



```
%% c. Apply truncation windows to keep 30%, 15%, and
5% of the DFT
% coefficients.

% Find the size of the original DFT.
[m1, n1] = size(lenaDFT);
[m2, n2] = size(irisDFT);

% Lena x- and y-coordinates
xMinLena30 = ceil(m1/2 - m1 * sqrt(0.3 / 2));
xMaxLena30 = ceil(m1/2 + m1 * sqrt(0.3 / 2));
```

```
yMinLena30 = ceil(n1/2 - n1 * sqrt(0.3 / 2));
yMaxLena30 = ceil(n1/2 + n1 * sqrt(0.3 / 2));
xMinLena15 = ceil(m1/2 - m1 * sqrt(0.15 / 2));
xMaxLena15 = ceil(m1/2 + m1 * sqrt(0.15 / 2));
yMinLena15 = ceil(n1/2 - n1 * sqrt(0.15 / 2));
yMaxLena15 = ceil(n1/2 + n1 * sqrt(0.15 / 2));
xMinLena5 = ceil(m1/2 - m1 * sqrt(0.05 / 2));
xMaxLena5 = ceil(m1/2 + m1 * sqrt(0.05 / 2));
yMinLena5 = ceil(n1/2 - n1 * sqrt(0.05 / 2));
yMaxLena5 = ceil(n1/2 + n1 * sqrt(0.05 / 2));

% Iris x- and y-coordinates
xMinIris30 = ceil(m2/2 - m2 * sqrt(0.3 / 2));
xMaxIris30 = ceil(m2/2 + m2 * sqrt(0.3 / 2));
yMinIris30 = ceil(n2/2 - n2 * sqrt(0.3 / 2));
yMaxIris30 = ceil(n2/2 + n2 * sqrt(0.3 / 2));
xMinIris15 = ceil(m2/2 - m2 * sqrt(0.15 / 2));
xMaxIris15 = ceil(m2/2 + m2 * sqrt(0.15 / 2));
yMinIris15 = ceil(n2/2 - n2 * sqrt(0.15 / 2));
yMaxIris15 = ceil(n2/2 + n2 * sqrt(0.15 / 2));
xMinIris5 = ceil(m2/2 - m2 * sqrt(0.05 / 2));
xMaxIris5 = ceil(m2/2 + m2 * sqrt(0.05 / 2));
yMinIris5 = ceil(n2/2 - n2 * sqrt(0.05 / 2));
yMaxIris5 = ceil(n2/2 + n2 * sqrt(0.05 / 2));

% Construct truncatation windows and multiply them by
the original DFT in
% order to truncate.

% 30%.
hLena30 = zeros(m1, n1);
hLena30(xMinLena30:xMaxLena30, yMinLena30:yMaxLena30)
= 1;
lenaDFT30 = fftshift(hLena30) .* lenaDFT;

% 15%.
hLena15 = zeros(m1, n1);
hLena15(xMinLena15:xMaxLena15, yMinLena15:yMaxLena15)
= 1;
lenaDFT15 = fftshift(hLena15) .* lenaDFT;

% 5%.
hLena5 = zeros(m1, n1);
hLena5(xMinLena5:xMaxLena5, yMinLena5:yMaxLena5) = 1;
lenaDFT5 = fftshift(hLena5) .* lenaDFT;

% 30%.
hIris30 = zeros(m2, n2);
```

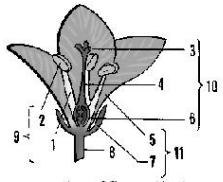
```
hIris30(xMinIris30:xMaxIris30, yMinIris30:yMaxIris30)
= 1;
irisDFT30 = fftshift(hIris30) .* irisDFT;

% 15%.
hIris15 = zeros(m2, n2);
hIris15(xMinIris15:xMaxIris15, yMinIris15:yMaxIris15)
= 1;
irisDFT15 = fftshift(hIris15) .* irisDFT;

% 5%.
hIris5 = zeros(m2, n2);
hIris5(xMinIris5:xMaxIris5, yMinIris5:yMaxIris5) = 1;
irisDFT5 = fftshift(hIris5) .* irisDFT;

% Plot truncated DFTs.
fig10 = figure(10);
subplot(3,2,1); imshow(log(abs(fftshift(lenaDFT30))),
[]); title('Log of 30% Magnitude of Lena Image');
subplot(3,2,3); imshow(log(abs(fftshift(lenaDFT15))),
[]); title('Log of 15% Magnitude of Lena Image');
subplot(3,2,5); imshow(log(abs(fftshift(lenaDFT5))),
[]); title('Log of 5% Magnitude of Lena Image');
subplot(3,2,2); imshow(log(abs(fftshift(irisDFT30))),
[]); title('Log of 30% Magnitude of Iris Image');
subplot(3,2,4); imshow(log(abs(fftshift(irisDFT15))),
[]); title('Log of 15% Magnitude of Iris Image');
subplot(3,2,6); imshow(log(abs(fftshift(irisDFT5))),
[]); title('Log of 5% Magnitude of Iris Image');
saveas(fig10, 'fig10_hw6.jpg');
```

d. I used the ifft2() command in MATLAB to compute the 2D inverse DFT. Below is the code used to produce the images shown below:

```
%% d. Apply the 2D Inverse DFT to reconstruct the
image for each of the
% truncated spectra. Print out both images
reconstructed using each of
% these truncation windows.

% Apply inverse DFT and bring back to original
dimensions.
lenaReconstruct = ifft2(lenaDFT);
lenaReconstruct = lenaReconstruct(1:256, 1:256);
lenaReconstruct30 = ifft2(lenaDFT30);
lenaReconstruct30 = lenaReconstruct30(1:256, 1:256);
lenaReconstruct15 = ifft2(lenaDFT15);
lenaReconstruct15 = lenaReconstruct15(1:256, 1:256);
```

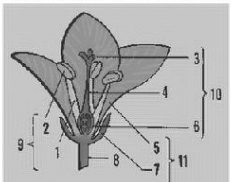
```
lenaReconstruct5 = ifft2(lenaDFT5);
lenaReconstruct5 = lenaReconstruct5(1:256, 1:256);

irisReconstruct = ifft2(irisDFT);
irisReconstruct = irisReconstruct(1:372, 1:298);
irisReconstruct30 = ifft2(irisDFT30);
irisReconstruct30 = irisReconstruct30(1:372, 1:298);
irisReconstruct15 = ifft2(irisDFT15);
irisReconstruct15 = irisReconstruct15(1:372, 1:298);
irisReconstruct5 = ifft2(irisDFT5);
irisReconstruct5 = irisReconstruct5(1:372, 1:298);

fig11 = figure(11);
subplot(2,2,1); imshow(real(lenaReconstruct), []);
title('Reconstructed Image');
subplot(2,2,2); imshow(real(lenaReconstruct30), []);
title('Reconstructed 30% Truncated Image');
subplot(2,2,3); imshow(real(lenaReconstruct15), []);
title('Reconstructed 15% Truncated Image');
subplot(2,2,4); imshow(real(lenaReconstruct5), []);
title('Reconstructed 5% Truncated Image');
saveas(fig11, 'fig11_hw6.jpg');

fig12 = figure(12);
subplot(2,2,1); imshow(real(irisReconstruct), []);
title('Reconstructed Image');
subplot(2,2,2); imshow(real(irisReconstruct30), []);
title('Reconstructed 30% Truncated Image');
subplot(2,2,3); imshow(real(irisReconstruct15), []);
title('Reconstructed 15% Truncated Image');
subplot(2,2,4); imshow(real(irisReconstruct5), []);
title('Reconstructed 5% Truncated Image');
saveas(fig12, 'fig12_hw6.jpg');
```

Reconstructed Image



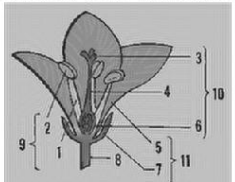
cross section of flower 1b: *1* filament, *2* anther, *3* stigma, *4* style, *5* petal, *6* ovary, *7* sepal, *8* pedicel, *9* stamen, *10* pistil, *11* perianth

Reconstructed 30% Truncated Image



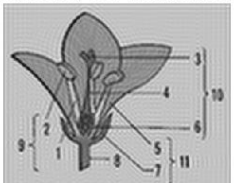
cross section of flower 1b: *1* filament, *2* anther, *3* stigma, *4* style, *5* petal, *6* ovary, *7* sepal, *8* pedicel, *9* stamen, *10* pistil, *11* perianth

Reconstructed 15% Truncated Image



cross section of flower 1b: *1* filament, *2* anther, *3* stigma, *4* style, *5* petal, *6* ovary, *7* sepal, *8* pedicel, *9* stamen, *10* pistil, *11* perianth

Reconstructed 5% Truncated Image



cross section of flower 1b: *1* filament, *2* anther, *3* stigma, *4* style, *5* petal, *6* ovary, *7* sepal, *8* pedicel, *9* stamen, *10* pistil, *11* perianth

Reconstructed Image

Reconstructed 30% Truncated Image

Reconstructed 15% Truncated Image

Reconstructed 5% Truncated Image

e. The formula for SNR is found as:

$$\text{SNR} = \frac{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\hat{f}(x,y)^2}{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[f(x,y)-\hat{f}(x,y)]^2}$$

where $\hat{f}$ is the noisy image and f is the original image. The following code was used to get the SNR values in the table below:

% DFT Coefficients	SNR Lena Image	SNR Iris Image
30%	1358.4	199.4252
15%	386.4698	71.3428
5%	129.15	29.2394

From the table above, we see that in both cases, truncating more data from the signal decreases the SNR. This makes sense, because as we lose more of the DFT coefficients, we reconstruct a "noisier" signal, which would reduce the signal to noise ratio.

From visually observing the two images, we see that the "lena" image is reconstructed better than the iris-illustration. We also observe this in the SNR values. The SNR values are significantly higher for the "lena" image in comparison to the "iris" image.

f. I compared the energy distributions produced from the natural photo (lena) and diagram (iris). In the natural photo (lena), the energy diagram has distinctive vertical and horizontal frequencies, indicating strong edges in directions perpendicular to these lines. Additionally, we see a greater low frequency content in the natural photo, indicating less sharp edges and contrast in comparison to the diagram. We can confirm this when observing the original images, since the diagram has greater contrast between the white background and darker foreground image.

   In comparing natural photos and diagrams, we see that natural photos contain mostly smooth variations in spatial domain, i.e., more low frequency response, while the diagram contains more contrast, i.e., more high frequency response. This explains why truncation of DFT coefficients affects the diagram more than the natural photo, and why the reconstructed "lena" images are better than the reconstructed "iris" images.