```
% Problem 1: Extraction Bitplanes from Image and Image Reconstruction from
% Bitplanes.
clear all;

%-----Part 1 : Read the original image :slope.tif', which is an 8 bits/pixel gray scale
image.
slope_image = imread('slope.tif'); % reading the image from an external source with the
help of imread funciton.

%-----Part 2 : Extract its 8 bitplanes as 8 binary images. storing image
%information in new variable
converted_slope_image = double(slope_image); % converting each pixel of the image in
double.

% extracting all bit one by one from 1st to 8th in variable from bitplane_1 to
bitplane_8 respectively.
% We are converting all the images into binary format. That is why we have
% used mod here.
bit_plane_8 = mod(converted_slope_image,2); % Least Significant Bitplane
bit_plane_7 = mod(floor(converted_slope_image/2),2);
bit_plane_6 = mod(floor(converted_slope_image/4),2);
bit_plane_5 = mod(floor(converted_slope_image/8),2);
bit_plane_4 = mod(floor(converted_slope_image/16),2);
bit_plane_3 = mod(floor(converted_slope_image/32),2);
bit_plane_2 = mod(floor(converted_slope_image/64),2);
bit_plane_1 = mod(floor(converted_slope_image/128),2); % Most Significant Bitplane

%imshow(a)

%-----Part 3 : Reconstruct 8 versions of the original image.
% Reconstructing the image using the most significant (upper) bitplane only.
reconstructed_image_1 = (2 * (2 * (2 * (2 * (2 * (2 * (2 * bit_plane_1))))))); % 128 *
bitplane1
reconstructed_image_1 = uint8(reconstructed_image_1); % It will convert each and every
pixel value of reconstructed_image_1 into the range of 0 to 255.

%reconstructing the image using upper 2 bitplanes only
reconstructed_image_2 = (2 * (2 * (2 * (2 * (2 * (2 * (2 * bit_plane_1 +
bit_plane_2))))))); % 128 * bitplane1 + 64 * bitplane2
reconstructed_image_2 = uint8(reconstructed_image_2); % It will convert each and every
pixel value of reconstructed_image_2 into the range of 0 to 255.

%reconstructing the image using upper 3 bitplanes only
reconstructed_image_3 = (2 * (2 * (2 * (2 * (2 * (2 * (2 * bit_plane_1 + bit_plane_2) +
bit_plane_3)))))); % 128 * bitplane1 + 64 * bitplane2 + 32 * bitplane3
reconstructed_image_3 = uint8(reconstructed_image_3); % It will convert each and every
pixel value of reconstructed_image_3 into the range of 0 to 255.

%reconstructing the image using upper 4 bitplanes only
reconstructed_image_4 = (2 * (2 * (2 * (2 * (2 * (2 * (2 * bit_plane_1 + bit_plane_2) +
```

```
bit_plane_3) + bit_plane_4))))); % 128 * bitplane1 + 64 * bitplane2 + 32 * bitplane3 +↙
16 * bitplane4
reconstructed_image_4 = uint8(reconstructed_image_4); % It will convert each and every↙
pixel value of reconstructed_image_4 into the range of 0 to 255.

%reconstructing the image using upper 5 bitplanes only
reconstructed_image_5 = (2 * (2 * (2 * (2 * (2 * (2 * (2 * bit_plane_1 + bit_plane_2) +↙
bit_plane_3) + bit_plane_4) + bit_plane_5) ))); % 128 * bitplane1 + 64 * bitplane2 + 32↙
* bitplane3 + 16 * bitplane4 + 8 * bitplane5
reconstructed_image_5 = uint8(reconstructed_image_5); % It will convert each and every↙
pixel value of reconstructed_image_5 into the range of 0 to 255.

%reconstructing the image using upper 6 bitplanes only
reconstructed_image_6 = (2 * (2 * (2 * (2 * (2 * (2 * (2 * bit_plane_1 + bit_plane_2) +↙
bit_plane_3) + bit_plane_4) + bit_plane_5) + bit_plane_6) )); % 128 * bitplane1 + 64 *↙
bitplane2 + 32 * bitplane3 + 16 * bitplane4 + 8 * bitplane5 + 4 * bitplane6
reconstructed_image_6 = uint8(reconstructed_image_6); % It will convert each and every↙
pixel value of reconstructed_image_6 into the range of 0 to 255.

%reconstructing the image using upper 7 bitplanes only
reconstructed_image_7 = (2 * (2 * (2 * (2 * (2 * (2 * (2 * bit_plane_1 + bit_plane_2) +↙
bit_plane_3) + bit_plane_4) + bit_plane_5) + bit_plane_6) + bit_plane_7) ); % 128 *↙
bitplane1 + 64 * bitplane2 + 32 * bitplane3 + 16 * bitplane4 + 8 * bitplane5 + 4 *↙
bitplane6 + 2 * bitplane7
reconstructed_image_7 = uint8(reconstructed_image_7); % It will convert each and every↙
pixel value of reconstructed_image_7 into the range of 0 to 255.

%reconstructing the image using upper 8 bitplanes only
reconstructed_image_8 = (2 * (2 * (2 * (2 * (2 * (2 * (2 * bit_plane_1 + bit_plane_2) +↙
bit_plane_3) + bit_plane_4) + bit_plane_5) + bit_plane_6) + bit_plane_7) +↙
bit_plane_8); % 128 * bitplane1 + 64 * bitplane2 + 32 * bitplane3 + 16 * bitplane4 + 8↙
* bitplane5 + 4 * bitplane6 + 2 * bitplane7 + 1 * bitplane8
reconstructed_image_8 = uint8(reconstructed_image_8); % It will convert each and every↙
pixel value of reconstructed_image_8 into the range of 0 to 255.

%-----Part 4 : Create 4 figures, each with 4 sub-figures of 2x2 layout
% In Figure 1, show the upper 4 bitplanes in 4 sub-figures.
figure; % figure creates figure graphics objects. figure objects are the individual↙
windows on the screen in which MATLAB displays graphical output.
subplot(2,2,1) % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the position specified by p. Here, m=n=2, p=1.
imshow(bit_plane_1) % It will display the gray-scale image in the figure.
title('Bitplane_1 image') % It will add the specified title for the current plot.
subplot(2,2,2) % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the position specified by p. Here, m=n=p=2.
imshow(bit_plane_2) % It will display the gray-scale image in the figure.
title('Bitplane_2 image') % It will add the specified title for the current plot.
subplot(2,2,3) % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the position specified by p. Here, m=n=2, p=3.
imshow(bit_plane_3) % It will display the gray-scale image in the figure.
```

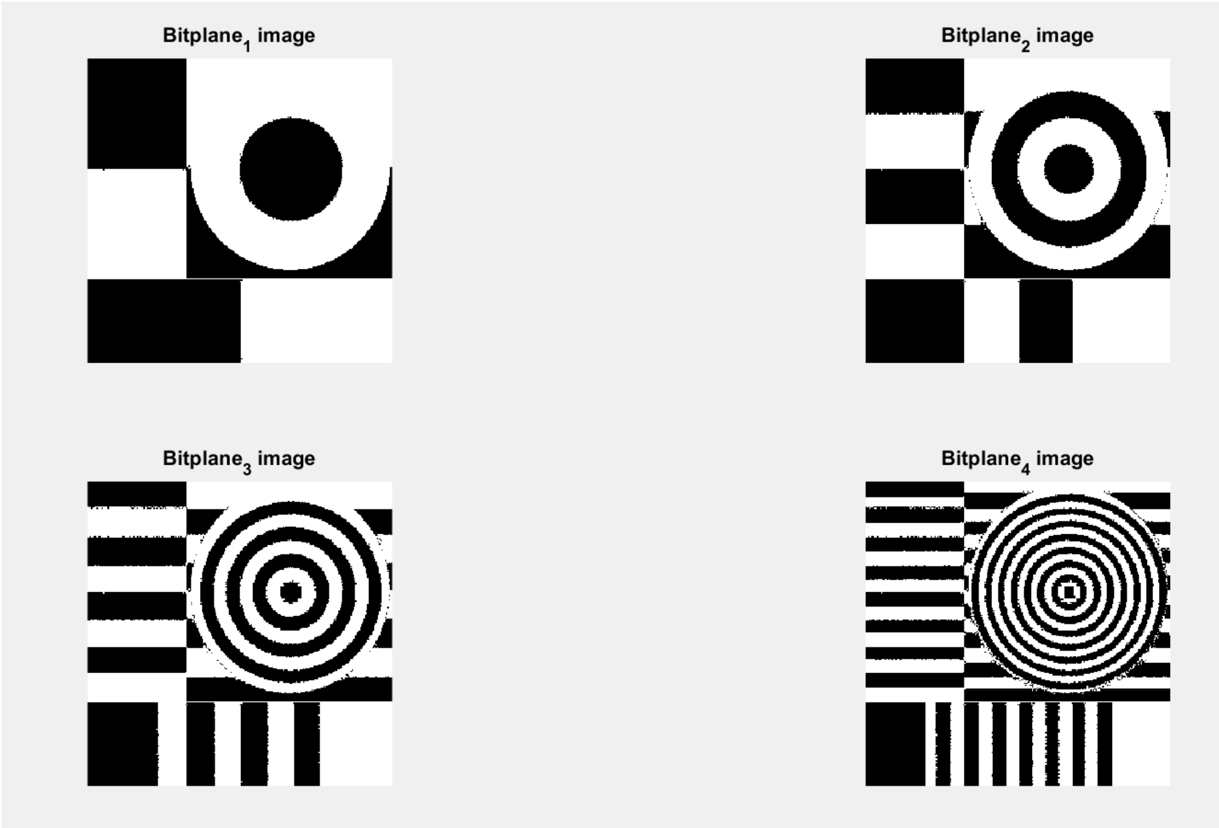
```
title('Bitplane_3 image') % It will add the specified title for the current plot.
subplot(2,2,4) % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the position specified by p. Here, m=n=2, p=4.
imshow(bit_plane_4) % It will display the gray-scale image in the figure.
title('Bitplane_4 image') % It will add the specified title for the current plot.

% In Figure 2, show the lower 4 bitplanes in 4 sub-figures.
figure; % figure creates figure graphics objects. figure objects are the individual↙
windows on the screen in which MATLAB displays graphical output.
subplot(2,2,1) % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the position specified by p. Here, m=n=2, p=1.
imshow(bit_plane_5) % It will display the gray-scale image in the figure.
title('Bitplane_5 image') % It will add the specified title for the current plot.
subplot(2,2,2) % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the position specified by p. Here, m=n=p=2.
imshow(bit_plane_6) % It will display the gray-scale image in the figure.
title('Bitplane_6 image') % It will add the specified title for the current plot.
subplot(2,2,3) % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the position specified by p. Here, m=n=2, p=3.
imshow(bit_plane_7) % It will display the gray-scale image in the figure.
title('Bitplane_7 image') % It will add the specified title for the current plot.
subplot(2,2,4) % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the position specified by p. Here, m=n=2, p=4.
imshow(bit_plane_8) % It will display the gray-scale image in the figure.
title('Bitplane_8 image') % It will add the specified title for the current plot.

% Figure 3, show the first 4 reconstructed images in 4 sub-figures, corresponding to↙
using the upper 1, 2, 3, and 4 bitplanes, respectively.
figure; % figure creates figure graphics objects. figure objects are the individual↙
windows on the screen in which MATLAB displays graphical output.
subplot(2,2,1) % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the position specified by p. Here, m=n=2, p=1.
imshow(reconstructed_image_1) % It will display the gray-scale image in the figure.
title('Reconstructed Image using bitplane 1') % It will add the specified title for the↙
current plot.
subplot(2,2,2) % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the position specified by p. Here, m=n=p=2.
imshow(reconstructed_image_2) % It will display the gray-scale image in the figure.
title('Reconstructed Image using bitplanes 1-2') % It will add the specified title for↙
the current plot.
subplot(2,2,3) % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the position specified by p. Here, m=n=2, p=3.
imshow(reconstructed_image_3) % It will display the gray-scale image in the figure.
title('Reconstructed Image using bitplanes 1-3') % It will add the specified title for↙
the current plot.
subplot(2,2,4) % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the position specified by p. Here, m=n=2, p=4.
imshow(reconstructed_image_4) % It will display the gray-scale image in the figure.
title('Reconstructed Image using bitplanes 1-4') % It will add the specified title for↙
the current plot.
```

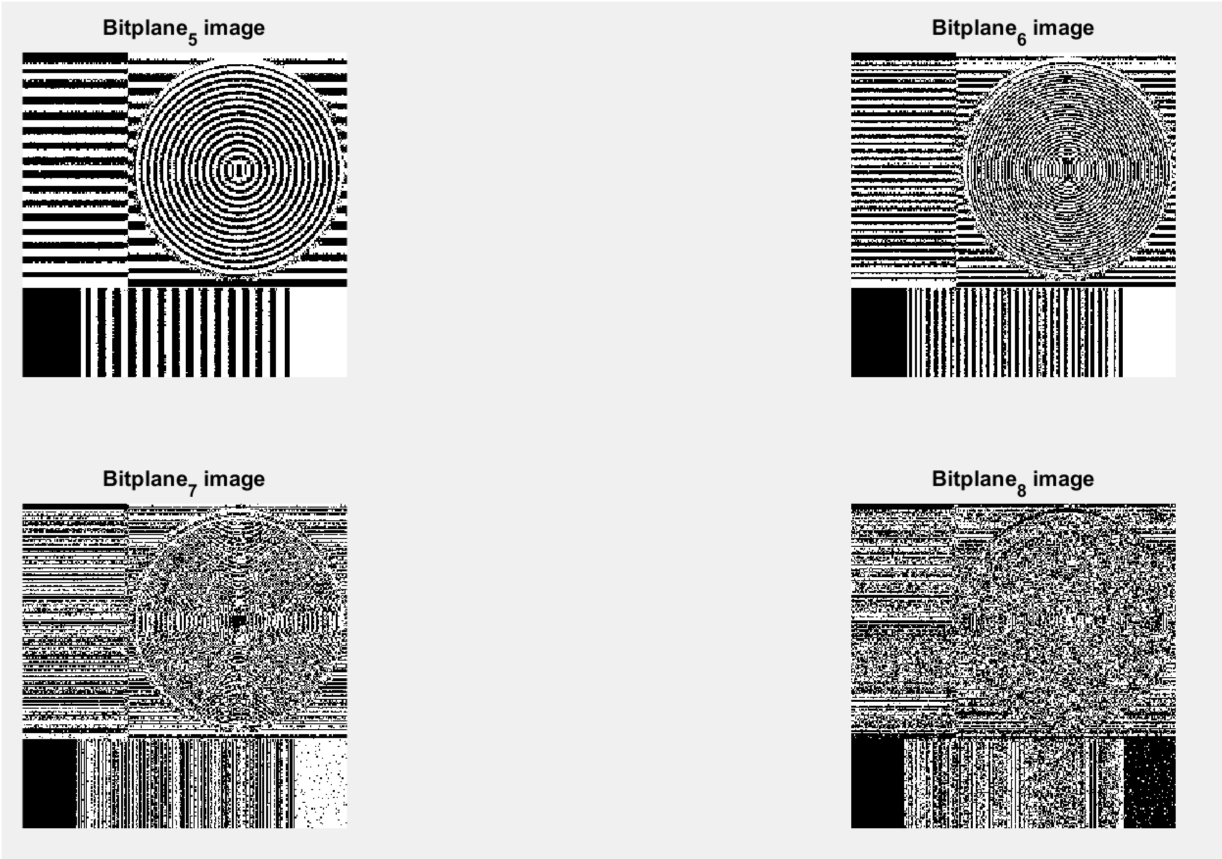
```
% In Figure 4, show the remaining 4 reconstructed images in 4 sub-figures,↙
corresponding to using the upper 5, 6, 7 and 8 bitplanes, respectively.
figure; % figure creates figure graphics objects. figure objects are the individual↙
windows on the screen in which MATLAB displays graphical output.
subplot(2,2,1) % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the position specified by p. Here, m=n=2, p=1.
imshow(reconstructed_image_5) % It will display the gray-scale image in the figure.
title('Reconstructed Image using bitplanes 1-5') % It will add the specified title for↙
the current plot.
subplot(2,2,2) % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the position specified by p. Here, m=n=p=2.
imshow(reconstructed_image_6) % It will display the gray-scale image in the figure.
title('Reconstructed Image using bitplanes 1-6') % It will add the specified title for↙
the current plot.
subplot(2,2,3) % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the position specified by p. Here, m=n=2, p=3.
imshow(reconstructed_image_7) % It will display the gray-scale image in the figure.
title('Reconstructed Image using bitplanes 1-7') % It will add the specified title for↙
the current plot.
subplot(2,2,4) % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the position specified by p. Here, m=n=2, p=4.
imshow(reconstructed_image_8) % It will display the gray-scale image in the figure.
title('Reconstructed Image using bitplanes 1-8') % It will add the specified title for↙
the current plot.
```

Figure 1:



Bitplane$_1$ image
Bitplane$_2$ image
Bitplane$_3$ image
Bitplane$_4$ image

Figure 2:



Bitplane$_5$ image
Bitplane$_6$ image
Bitplane$_7$ image
Bitplane$_8$ image

Figure 3:

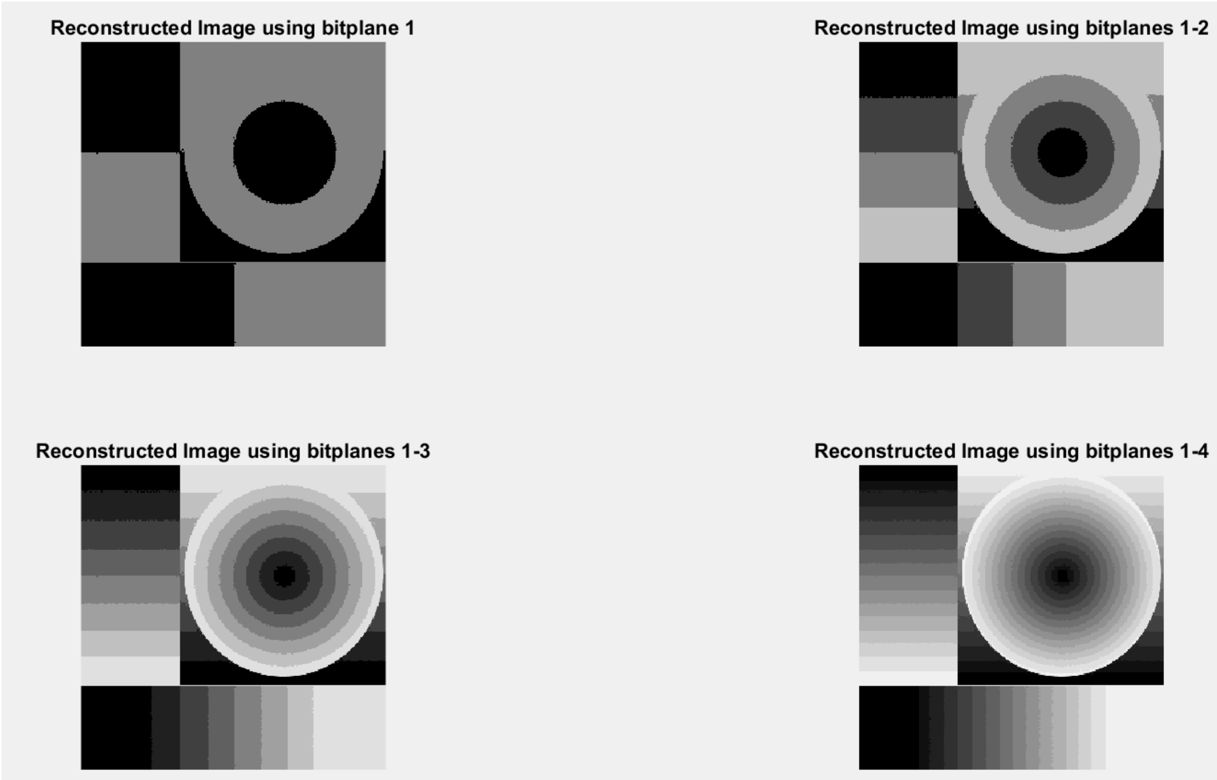
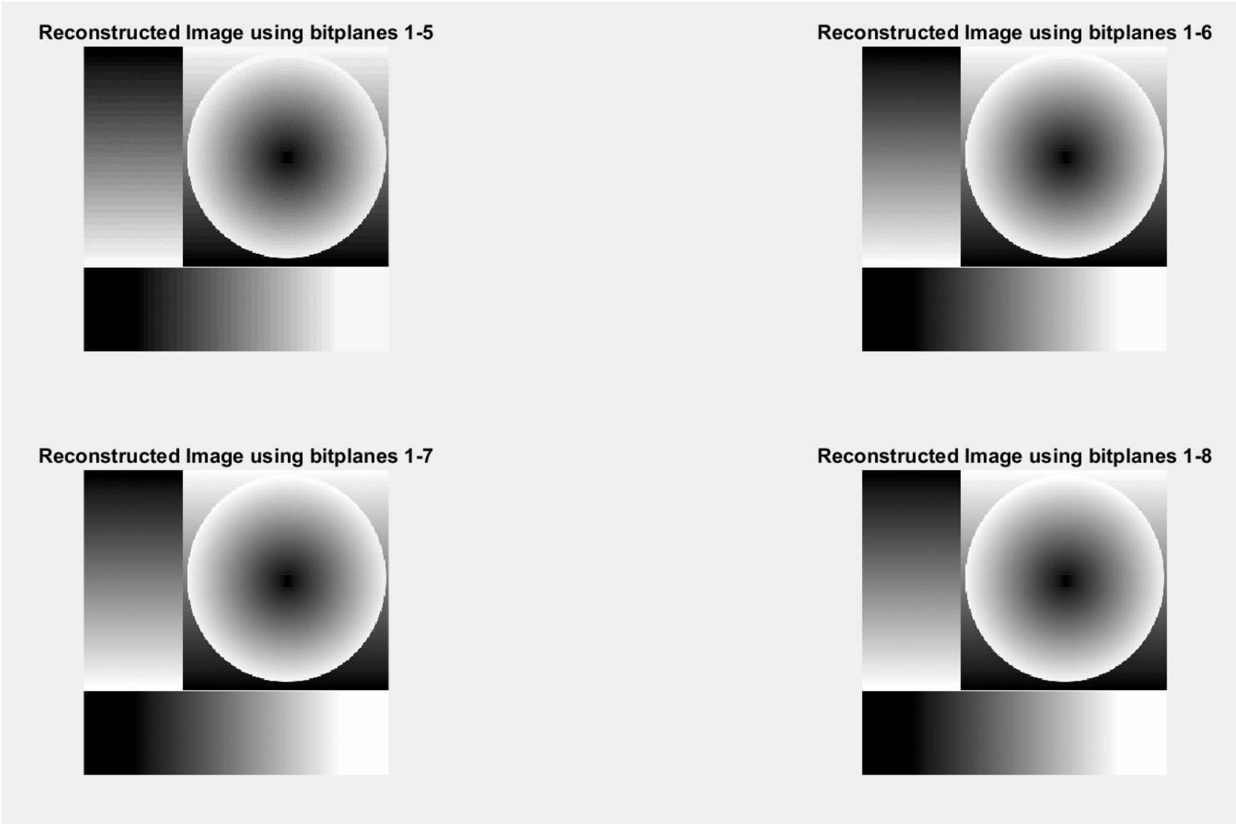

Reconstructed Image using bitplane 1
Reconstructed Image using bitplanes 1-2
Reconstructed Image using bitplanes 1-3
Reconstructed Image using bitplanes 1-4

Figure 4:



Q1 Observation:

The original image is broken into multiple bitplanes (8 in all) in Figures 1 and 2, creating the appearance that the image has been fractured into 8 layers. After we recreated the image from these different bitplanes one at a time, two at a time, and so on, we can see the picture recovering its prior look bitplane by bitplane. The picture takes almost all of its shape in the first four reconstructed photographs in Figure 3, while the remaining rebuilt photos in Figure 4 are merely adding to the finer definitions. Because the first four Bitplanes 1-4 are the most significant, while Bitplanes 5-8 are the least crucial, we observe this. As demonstrated in the reconstruction, the upper bitplanes contain the bulk of the structural data, while the lower bitplanes contribute the distortions and gray areas. The reconstruction shows that a 6 bitplane reconstruction can almost precisely match the original image.

wrong normalization

```
% Problem 2: Intensity Transformations: Gamma Mapping, Full-scale Contrast
% Stretch, and Histogram Equalization.
clear all;

input_image = double(imread('books.tif')); % reading the original image. This image is
8 bits/pixel gray-scale
% image.

% Plotting the original image.
figure % figure creates figure graphics objects. figure objects are the individual
windows on the screen
% in which MATLAB displays graphical output.
subplot(2,2,1); % subplot(m, n, p) divides the current figure into an m-by-n grid and
creates axes in the
% position specified by p. Here, m=n=2, p=1.
imshow(uint8(input_image)); % It will display the gray-scale image in the figure and
% it will convert each and every pixel value of the input image into the range of 0 to
255.
title('Original Books Image'); % It will add the specified title for the current plot.

max_val = max(input_image(:)); % It will calculate the maximum value from the input
image. It
% will be further used for image scaling for generating gamma mapped image.
min_val = min(input_image(:)); % It will calculate the minimum value from the input
image. It
% will be further used for image scaling for generating gamma mapped image.
input_image_scaled = (input_image - min_val) / (max_val - min_val); % Here, we are
scalling the pixel
% intensity. It is basically called the normalixation of all the pixel
% values in the image. We are scalling this image from [0, 255] to [0, 1]
% to apply power law.
gamma_transformed_image = (input_image_scaled.^0.5) * 255; % Here, we applied power law
with gamma = 0.5 and
% after that we again need to rescale the image from [0,1] to [0, 255].Now, our gamma
mapped image is ready.

subplot(2,2,2); % subplot(m, n, p) divides the current figure into an m-by-n grid and
creates axes in the
% position specified by p. Here, m=n=p=2.
imshow(uint8(gamma_transformed_image)); % It will display the gray-scale image in the
figure and
% it will convert each and every pixel value of the input image into the range of 0 to
255.
title('Gamma Transformed Image'); % It will add the specified title for the current
plot.

full_scale_contrast_stretched_image = round((2^8-1) * input_image_scaled); % Here, we
created the full scale
% contrast stretched image from the original input scaled image.
subplot(2,2,3); % subplot(m, n, p) divides the current figure into an m-by-n grid and
```

```
creates axes in the
% position specified by p. Here, m=n=2, p=3.
imshow(uint8(full_scale_contrast_stretched_image)); % It will display the gray-scale↙
image in the figure and
% it will convert each and every pixel value of the input image into the range of 0 to↙
255.
title('Full Scale Contrast Streched Transformed Image'); % It will add the specified↙
title for the current
% plot.


cummuluative_histogram_equalized_image = cumsum(hist(reshape(input_image,[],1),↙
linspace(0,255,256))); % It will
% create a cummulative histogram equalized image. hist() function will
% create a histogram bar chart of the elements in the given vector.
% reshape() method will change the shape of our image.
intermediate_image = cummuluative_histogram_equalized_image(uint8(input_image) + 1); %↙
It will create an
% intermediate image to create final histogram equalized image. uint8() will convert↙
the value
% of each pixel from 0 to 255.
hei = round((2^8 - 1)*((intermediate_image - min(intermediate_image(:)))/(max↙
(intermediate_image(:))- ...
    min(intermediate_image(:))))); % It will simply do the scalling of the image to↙
generate the final histogram
% equalized image.
subplot(2,2,4); % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the
% position specified by p. Here, m=n=2, p=4.
imshow(uint8(hei)); % It will display the gray-scale image in the figure and
% it will convert each and every pixel value of the input image into the range of 0 to↙
255.
title('Histogram Equalized Image'); % It will add the specified title for the current
% plot.


figure; % figure creates figure graphics objects. figure objects are the individual↙
windows on the screen
% in which MATLAB displays graphical output.
original_image_histogram = hist(reshape(input_image,[],1), linspace(0,255,256));
subplot(2,2,1); % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the
% position specified by p. Here, m=n=2, p=1.
bar(original_image_histogram); % It will create a bar chart of the given image.
title('Histogram of Original Books Image'); % It will add the specified title for the↙
current
% plot.

gamma_transformed_image_histogram = hist(reshape(gamma_transformed_image,[],1),↙
linspace(0,255,256));
```

```
subplot(2,2,2); % subplot(m, n, p) divides the current figure into an m-by-n grid and
creates axes in the
% position specified by p. Here, m=n=p=2.
bar(gamma_transformed_image_histogram); % It will create a bar chart of the given
image.
title('Histogram of Gamma Transformed Image'); % It will add the specified title for
the current
% plot.

fscs_image_histogram = hist(reshape(full_scale_contrast_stretched_image,[],1), linspace
(0,255,256));
subplot(2,2,3); % subplot(m, n, p) divides the current figure into an m-by-n grid and
creates axes in the
% position specified by p. Here, m=n=2, p=3.
bar(fscs_image_histogram); % It will create a bar chart of the given image.
title('Histogram of Full Scale Contrast Streched Image'); % It will add the specified
title for the current
% plot.

hei_image_histogram = hist(reshape(hei,[],1), linspace(0,255,256));
subplot(2,2,4); % subplot(m, n, p) divides the current figure into an m-by-n grid and
creates axes in the
% position specified by p. Here, m=n=2, p=4.
bar(hei_image_histogram); % It will create a bar chart of the given image.
title('Histogram of Histogram Equalized Image'); % It will add the specified title for
the current
% plot.
```

Figure 1:

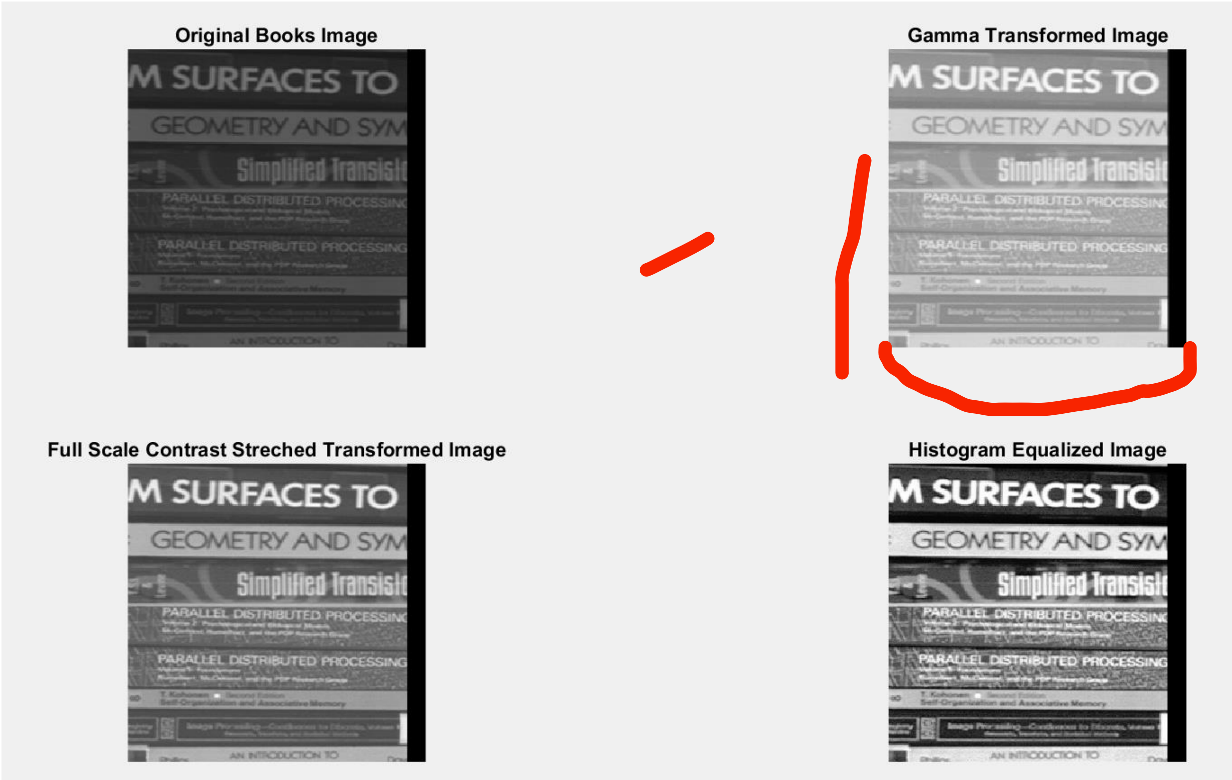
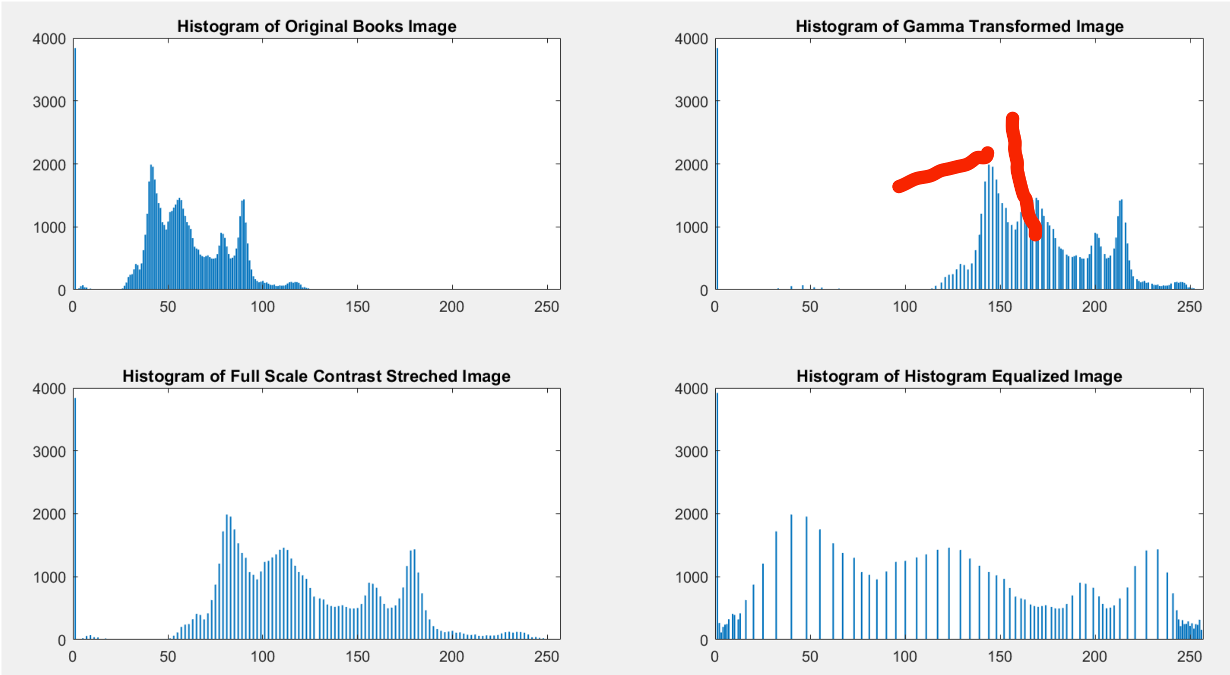

Original Books Image
Gamma Transformed Image
Full Scale Contrast Streched Transformed Image
Histogram Equalized Image

Figure 2:



Q2 Observation:

The image sharpened as we moved from Gamma intensity change to full-scale contrast stretch to histogram equalization. The histograms also revealed that the pixel intensities in the histogram equalized image are more balanced than the original image, implying that the histogram equalized image is the clearest of the four images. After applying the gamma rule, the pixel values (intensity) moved towards the center, slightly increasing the brightness. The contrast and brightness of the full-size extended image are higher, and the histogram reveals more uniformly distributed pixel values.

```
% Problem 3: Frequency Domain Low-pass, Band-pass and High-pass Filtering.
clear all;

% functionality of a particular functions.
% fft2() - It will return the two-dimensional fourier transform of a
% matrix using the fast fourier trtansform.
% fftshift() - It will rearrange a Fourier transform x by shifting the
% zero-frequency component to the center of the array.
% ifft2() - It will return the two-dimensional inverse fourier transform of a
% matrix using the fast fourier trtansform.

input_image = double(imread("bridge.tif")); % reading the original image. This image is
8 bits/pixel gray-scale
% image.
[n_rows, n_cols] = size(input_image); % It will return a x * y dimension of the image.
x represents the number of
% rows in the image and y represents the number of columns of a matrix formed from the
input image.
% for example if we have a 3-by-4 image then size function will return [3 4].

% The 2-D DFT of the original image and shifting the DC component.
input_image_2ddft = fft2(input_image); % 2D-DFT of the original image.
input_image_2ddft_shift = fftshift(input_image_2ddft); % Shifting of the DC component.

%center of image
mid_value = n_rows/2; % finding the middle of the image.

% We want to do the indexing of the matrix, so we will create rectuangluar
% structures from the given array. It is basically done with the help of
% meshgrid function.
[vector_col,vector_row] = meshgrid(1:n_cols,1:n_rows); % It will return 2-D grid
coordinates based on the
% coordinates in vectors (1:n_row) and (1:n_col).

% Evaluating distance over the grid. Calculating the euclidean distance(distance
between two points).
distance = ((vector_row-mid_value).^2+(vector_col-mid_value).^2).^0.5; % simply
calculating the distance between
% two points.

% Here, we will design 3 filters in total. 1] Low-pass filter, 2] high-pass
% filter, and 3] bandpass filter.

% Designing low pass filter

distance_low =  (1/8) * mid_value; % It is given in the problem statement that
frequency response of the ideal
% low pass filter is equal to one-eighth of the distance from the center to the
horizontal or vertical edge.
low_pass_filter = (distance <= distance_low); % here the distance must be less than or
```

```
equal to the frequency
% response of the ideal low pass filter.
low_pass_filtered_image = input_image_2ddft_shift.*low_pass_filter; % Here, the '.'↙
after input_image_2ddft_shift
% indicates that it will take all the pixel values. i.e. it will take all the values of↙
each and every rows and
% columns from the matrix. Also, we are applying a low pass filter on the shifted↙
image.
low_pass_filtered_spatial_image = real(ifft2(fftshift(low_pass_filtered_image))); %↙
Here we are applying inverse
% 2D-DFT on the low pass filtered image to obtain the image in spatial domain. % So,↙
first we apply fftshift and
% then ifft2 function for inverse 2D-DFT.

% Designing high pass filter

distance_high = (1/2) * mid_value; % It is given in the problem statement that↙
frequency response of the ideal
% high pass filter is equal to one-twoth of the distance from the center to the↙
horizontal or vertical edge.
high_pass_filter = (distance >= distance_high); % here the distance must be greater↙
than or equal to the frequency
% response of the ideal high pass filter.
high_pass_filtered_image = input_image_2ddft_shift.*high_pass_filter; % Here, the '.'↙
after input_image_2ddft_shift
% indicates that it will take all the pixel values. i.e. it will take all the values of↙
each and every rows and
% columns from the matrix. Also, we are applying a high pass filter on the shifted↙
image.
high_pass_filtered_spatial_image = real(ifft2(fftshift(high_pass_filtered_image))); %↙
Here we are applying inverse
% 2D-DFT on the high pass filtered image to obtain the image in spatial domain. % So,↙
first we apply fftshift and
% then ifft2 function for inverse 2D-DFT.

% Deigning band pass filter

band_pass_filter = (distance >= distance_low & distance <= distance_high); % here the↙
distance must be greater
% than or equal to the frequency response of the ideal low pass filter and less than or↙
equal to the frequency
% response of the ideal high pass filter.
band_pass_filtered_image = input_image_2ddft_shift.*band_pass_filter; % Here, the '.'↙
after input_image_2ddft_shift
% indicates that it will take all the pixel values. i.e. it will take all the values of↙
each and every rows and
% columns from the matrix. Also, we are applying a band pass filter on the shifted↙
image.
band_pass_filtered_spatial_image = real(ifft2(fftshift(band_pass_filtered_image)));  %↙
Here we are applying inverse
```

```
% 2D-DFT on the band pass filtered image to obtain the image in spatial domain. % So,↙
first we apply fftshift and
% then ifft2 function for inverse 2D-DFT.

f = figure; % figure creates figure graphics objects. figure objects are the individual↙
windows on the screen
% in which MATLAB displays graphical output.
subplot(2,2,1); % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the
% position specified by p. Here, m=n=2, p=1.
imshow(uint8(input_image)); % It will display the gray-scale image in the figure and
% it will convert each and every pixel value of the input image into the range of 0 to↙
255.
title('Original_Image'); % It will add the specified title for the current plot.
subplot(2,2,2); % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the
% position specified by p. Here, m=n=p=2.
imshow(low_pass_filtered_spatial_image,[]); % It will display the gray-scale image in↙
the figure.
title('Low-pass filtered Image'); % It will add the specified title for the current↙
plot.
subplot(2,2,3); % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the
% position specified by p. Here, m=n=2, p=3.
imshow(band_pass_filtered_spatial_image,[]); % It will display the gray-scale image in↙
the figure.
title('Band-pass filtered Image'); % It will add the specified title for the current↙
plot.
subplot(2,2,4); % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the
% position specified by p. Here, m=n=2, p=4.
imshow(high_pass_filtered_spatial_image,[]); % It will display the gray-scale image in↙
the figure.
title('High-pass filtered Image'); % It will add the specified title for the current↙
plot.

f = figure; % figure creates figure graphics objects. figure objects are the individual↙
windows on the screen
% in which MATLAB displays graphical output.
max_amplitude = max(input_image(:)); % It will simply find the maximum amplitude of the↙
image.
log_shifted_img = log(1+abs(input_image_2ddft_shift)); % Apply log transform on the↙
2ddft shifted image
% and make it log shifted image.
subplot(2,2,1); % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the
% position specified by p. Here, m=n=2, p=1.
imshow(log_shifted_img./max_amplitude,[]); % It will display the gray-scale image in↙
the figure.
title('Original Image'); % It will add the specified title for the current plot.
```

```
log_shifted_low = log(1+abs(low_pass_filtered_image)); % Apply log transform on the↙
2ddft shifted image
% and make it log shifted image.
subplot(2,2,2); % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the
% position specified by p. Here, m=n=p=2.
imshow(log_shifted_low./max_amplitude, []); % It will display the gray-scale image in↙
the figure.
title('Low-pass filtered'); % It will add the specified title for the current plot.
log_shifted_band = log(1+abs(band_pass_filtered_image)); % Apply log transform on the↙
2ddft shifted image
% and make it log shifted image.
subplot(2,2,3); % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the
% position specified by p. Here, m=n=2, p=3.
imshow(log_shifted_band./max_amplitude, []); % It will display the gray-scale image in↙
the figure.
title('Band-pass filtered'); % It will add the specified title for the current plot.
log_shifted_high = log(1+abs(high_pass_filtered_image)); % Apply log transform on the↙
2ddft shifted image
% and make it log shifted image.
subplot(2,2,4); % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the
% position specified by p. Here, m=n=2, p=4.
imshow(log_shifted_high./max_amplitude, []); % It will display the gray-scale image in↙
the figure.
title('High-pass filtered'); % It will add the specified title for the current plot.
```

Figure 1:

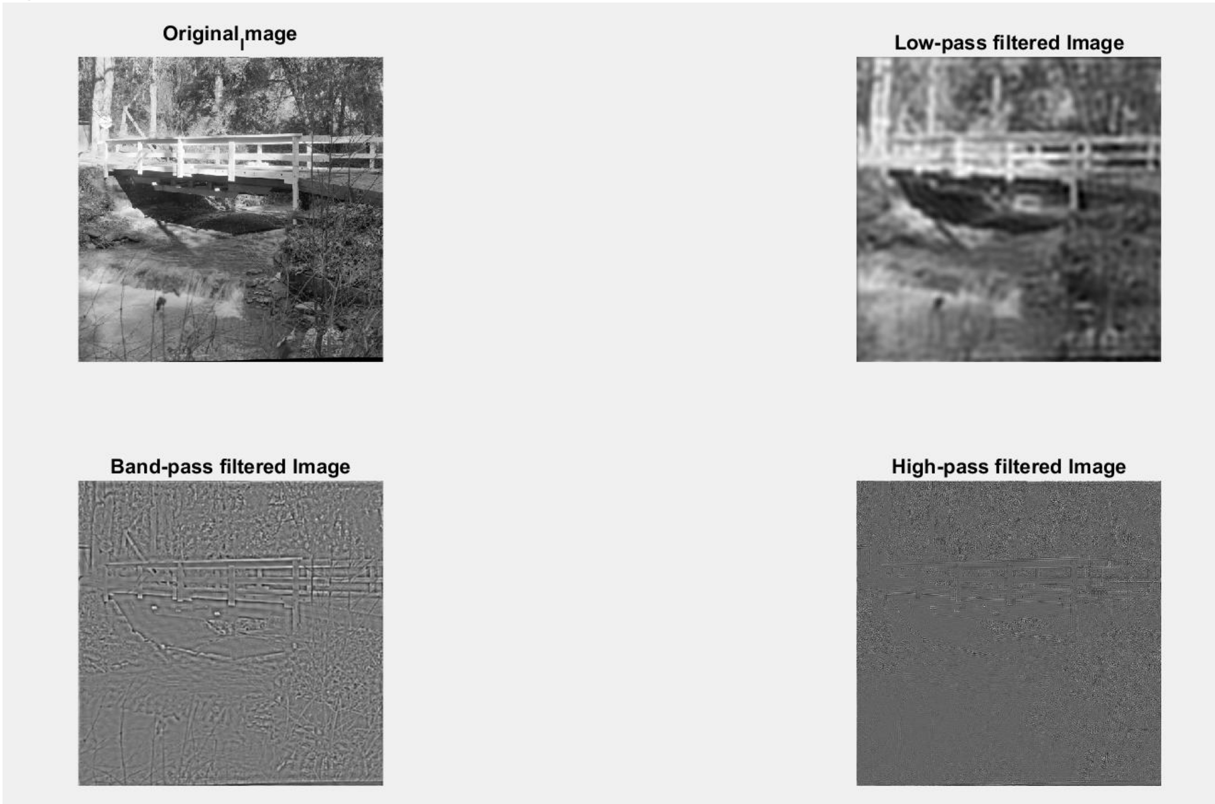
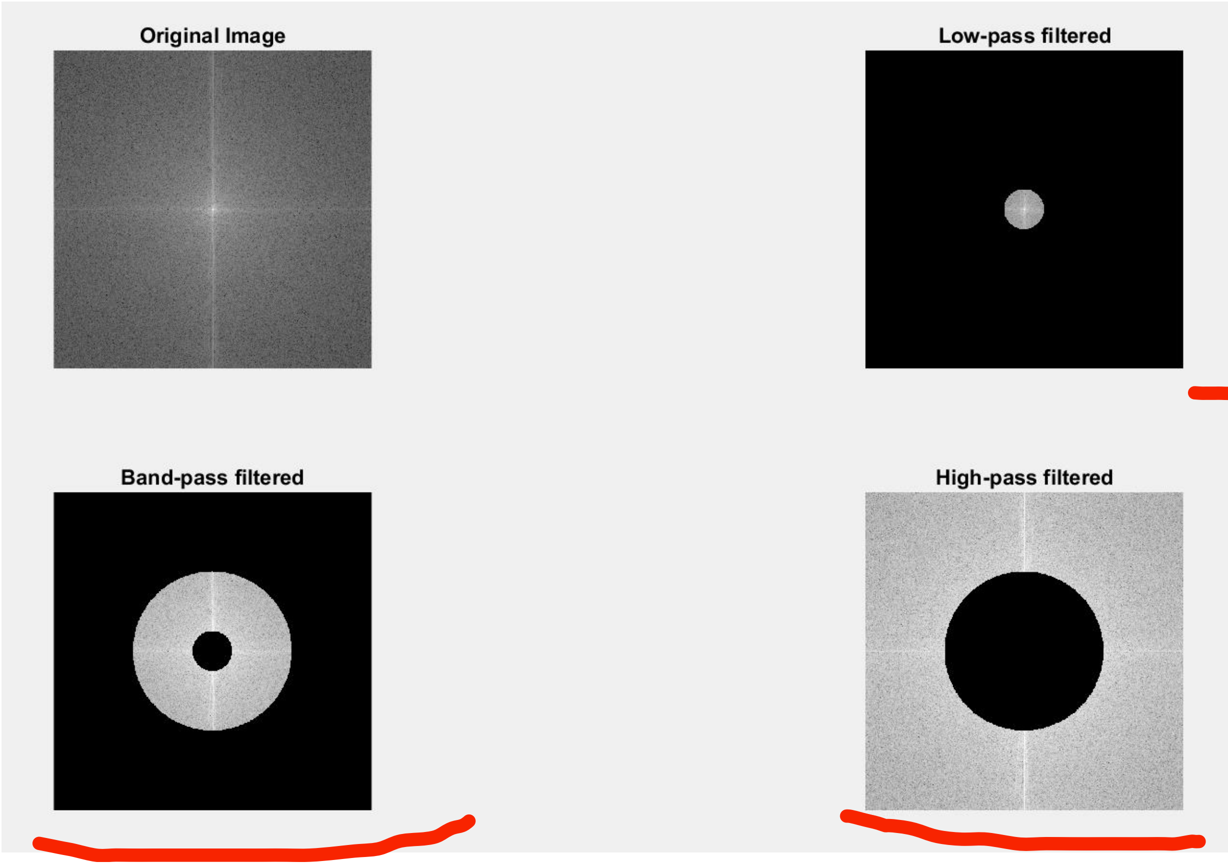

Original Image
Low-pass filtered Image
Band-pass filtered Image
High-pass filtered Image

Figure 2:



-2

Q3 Observation:

The low pass filter smoothens out the images by reducing the high-intensity pixels while leaving the low-intensity pixels alone. As can be seen from the amplitude response, pixels with a percent higher intensity than the cut-off value are masked out. Similarly, the high pass filter sharpens the image. By suppressing low Pixel intensities and reducing the percent intensity, the band pass filter improves the edges while minimizing noise. Extremely high pixel intensities smoothens out the image by filtering out all of the high frequency components after sending it through a low pass filter. The image is stripped of all low frequency components after going through a high pass filter, leaving just the edges.

```
% Problem 4 - Image Deblurring by Frequency Domain Inverse Filter Design.
clear all;

% functionality of a particular functions.
% fft2() - It will return the two-dimensional fourier transform of a
% matrix using the fast fourier trtansform.
% fftshift() - It will rearrange a Fourier transform x by shifting the
% zero-frequency component to the center of the array.
% ifft2() - It will return the two-dimensional inverse fourier transform of a
% matrix using the fast fourier transform.
% mesh() - It will create a 3D surface that has a sloid edge colors and no face colors.

input_image = double(imread("text.tif")); % reading the original image. This image is 8↙
bits/pixel gray-scale

size_of_filter = 21; % We are simply defining the filter size and it is used further in↙
the below code.
standard_deviation = 1; % We are defining the value of standard deviation as per the↙
given problem.

[vector_col,vector_row] = meshgrid(-floor(size_of_filter/2):floor(size_of_filter/2), -↙
...
    floor(size_of_filter/2):floor(size_of_filter/2)); % It will return 2-D grid↙
coordinates based on the
% coordinates in vectors. This basically means that we are using the cropped image of↙
the central part, so that
% input image and the output image, bith has the same size.
gaussian_filter = exp(-(vector_col.^2 + vector_row.^2)/(2*standard_deviation^2))/↙
(2*pi*standard_deviation^2);
% Here, we are just
% normilizing the gaussian filter such that the sum of all the coefficients will be↙
equal to 1. This is given in
% the problem statement. We are just normalizing the gaussian filter.
blurring_filter = gaussian_filter./sum(gaussian_filter(:)); % created a blurring filter↙
which will be used
% to give blurring effect to an image.

blurred_image = conv2(input_image, blurring_filter, 'same'); % we applied 2D↙
convolution between the original
% input image and the gaussian filter to get a blurred image as a resulting image.↙
Hence, we have applied
% conv2() function on the input image.

% Frequency domain of the gaussian filter.
gaussian_filter_freq_domain = fft2(blurring_filter); % Here, we are applying 2D-DFT to↙
the spatial domain
% Gaussian filter to obtain frequency domain gaussian filter.  Hence, we have applied↙
fft2() function on the
% blurred image.
```

```
%Inverse Gaussian Filter
freq_domain_inverse_gaussian_filter = 1./gaussian_filter_freq_domain; % Here, we have↙
created a frequency
% domain inverse gaussian filter by simply taking reciprocal value of each and every↙
pixel value of frequency
% domain Gaussian filter coefficients. The '.'indicates that we have taken all the↙
values from the image matrix.
deblurring_spatial_domain_inverse_filter = real(ifft2↙
(freq_domain_inverse_gaussian_filter)); % Here, as per the
% problem statement we have applied the inverse 2D-DFT to generate spatial domain↙
inverse gaussian filter. For
% this we have implemented ifft2() method.

% convolution between the blurred image and the deblurred image obtained from spatial↙
domain inverse filter.
deblurred_image = conv2(blurred_image, deblurring_spatial_domain_inverse_filter,↙
'same'); % we applied 2D
% convolution between the spatial domain gaussian filtered image and the spatial domain↙
inverse gaussian filter
% to get a spatial domain blurred image as a resulting image. Hence, we have applied↙
conv2() function on the
% input image.

% Intensity transformation and DC shift for corresponding images.
original_image_fft = fft2(input_image); % Here, we are applying 2D-DFT to the input↙
image to obtain
% fourier transformed image.  Hence, we have applied fft2() function on the input↙
image.
original_image_intensity_transform = log(1 + abs(original_image_fft)); % Apply log↙
transform on the 2ddft
% image and make it log shifted image.
original_image_frequency = fftshift(original_image_intensity_transform./ ...
    max(original_image_intensity_transform(:))); % DC shift for the corresponding↙
images.

blurred_image_fft = fft2(blurred_image); % Here, we are applying 2D-DFT to the blurred↙
image to obtain
% fourier transformed image.  Hence, we have applied fft2() function on the blurred↙
image.
blurred_image_intensity_transform = log(1 + abs(blurred_image_fft)); % Apply log↙
transform on the 2ddft
% image and make it log shifted image.
blurred_image_frequency = fftshift(blurred_image_intensity_transform./max↙
(blurred_image_intensity_transform(:)));
% DC shift for the corresponding images.

deblurred_image_fft = fft2(deblurred_image); % Here, we are applying 2D-DFT to the↙
deblurred image to obtain
% fourier transformed image.  Hence, we have applied fft2() function on the deblurred↙
image.
```

```
deblurred_image_intensity_transform = log(1 + abs(deblurred_image_fft)); % Apply log↙
transform on the 2ddft
% image and make it log shifted image.
deblurred_image_frequency = fftshift(deblurred_image_intensity_transform./ ...
    max(deblurred_image_intensity_transform(:))); % DC shift for the corresponding↙
images.


% Plot for images and figures

figure % figure creates figure graphics objects. figure objects are the individual↙
windows on the screen
% in which MATLAB displays graphical output.
subplot(2,3,1); % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the
% position specified by p. Here, m=2, n=3, p=1.
imshow(uint8(input_image)); % It will display the gray-scale image in the figure and
% it will convert each and every pixel value of the input image into the range of 0 to↙
255.
title('Original Text Image'); % It will add the specified title for the current plot.

subplot(2,3,2); % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the
% position specified by p. Here, m=2, n=3, p=2.
imshow(uint8(blurred_image)); % It will display the gray-scale image in the figure and
% it will convert each and every pixel value of the input image into the range of 0 to↙
255.
title('Blurred Text Image'); % It will add the specified title for the current plot.

subplot(2,3,3); % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the
% position specified by p. Here, m=2, n=p=3.
imshow(uint8(deblurred_image)); % It will display the gray-scale image in the figure↙
and
% it will convert each and every pixel value of the input image into the range of 0 to↙
255.
title('Deblurred Text Image'); % It will add the specified title for the current plot.

subplot(2,3,4); % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the
% position specified by p. Here, m=2, n=3, p=4.
imshow(original_image_frequency,[]); % It will display the gray-scale image in the↙
figure and
% it will convert each and every pixel value of the input image into the range of 0 to↙
255.
title('Original Image Frequency'); % It will add the specified title for the current↙
plot.

subplot(2,3,5); % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the
```

```
% position specified by p. Here, m=2, n=3, p=5.
imshow(blurred_image_frequency,[]); % It will display the gray-scale image in the↙
figure and
% it will convert each and every pixel value of the input image into the range of 0 to↙
255.
title('Blurred Image Frequency'); % It will add the specified title for the current↙
plot.

subplot(2,3,6); % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the
% position specified by p. Here, m=2, n=3, p=6.
imshow(deblurred_image_frequency,[]); % It will display the gray-scale image in the↙
figure and
% it will convert each and every pixel value of the input image into the range of 0 to↙
255.
title('Deblurred Image Frequency'); % It will add the specified title for the current↙
plot.


%%%%%
figure % figure creates figure graphics objects. figure objects are the individual↙
windows on the screen
% in which MATLAB displays graphical output.
subplot(2,2,1); % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the
% position specified by p. Here, m=n=2, p=1.
mesh(blurring_filter); % It will create a 3D surface that has a sloid edge colors and↙
no face colors.
title('Spatial domain Gaussian Blur Filter'); % It will add the specified title for the↙
current plot.

subplot(2,2,2); % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the
% position specified by p. Here, m=n=p=2.
mesh(abs(fftshift(fft2(blurring_filter)))); % It will create a 3D surface that has a↙
sloid edge colors and no
% face colors.
title('Magnitude of Fourier Domain GBF'); % It will add the specified title for the↙
current plot.


subplot(2,2,3); % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the
% position specified by p. Here, m=n=2, p=3.
mesh(deblurring_spatial_domain_inverse_filter); % It will create a 3D surface that has↙
a sloid edge colors and no
% face colors.
title('Spatial domain Gaussian Deblur Filter'); % It will add the specified title for↙
the current plot.
```

```
subplot(2,2,4); % subplot(m, n, p) divides the current figure into an m-by-n grid and
creates axes in the
% position specified by p. Here, m=n=2, p=4.
mesh(abs(fftshift(freq_domain_inverse_gaussian_filter))); % It will create a 3D surface
that has a sloid edge colors and
% no face colors.
title('Magnitude of Fourier Domain GDF'); % It will add the specified title for the
current plot.
```

Figure 1:

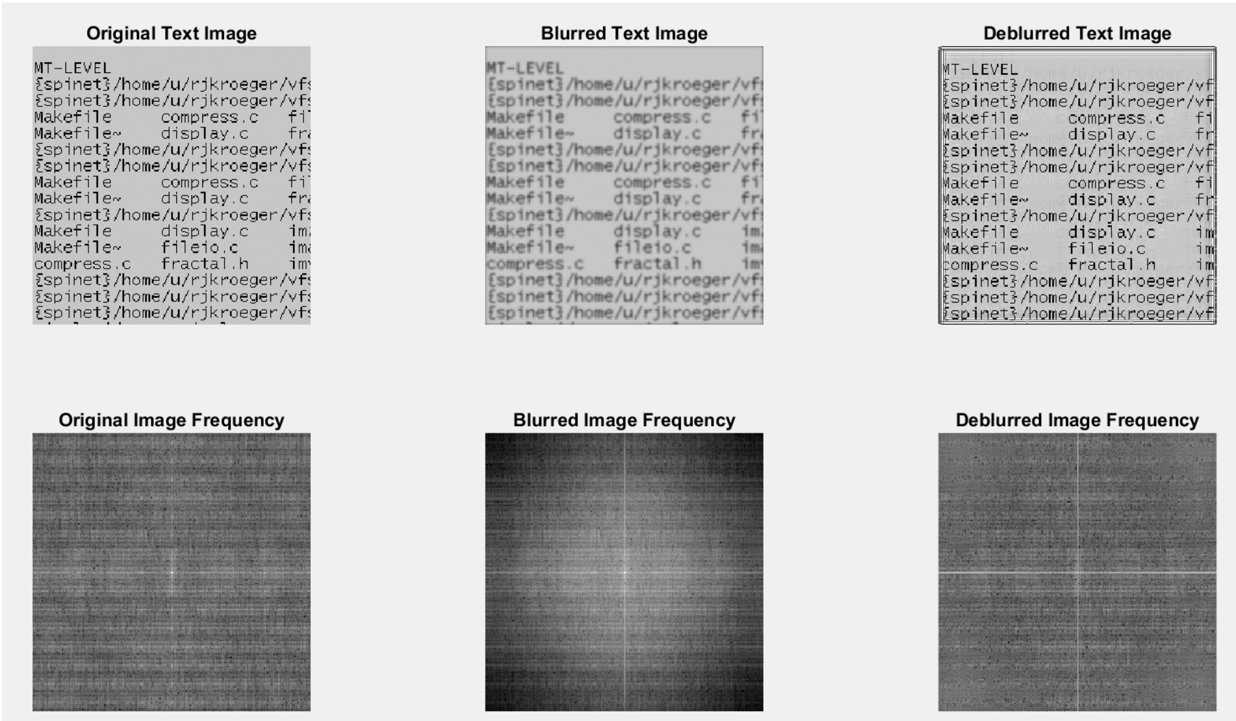
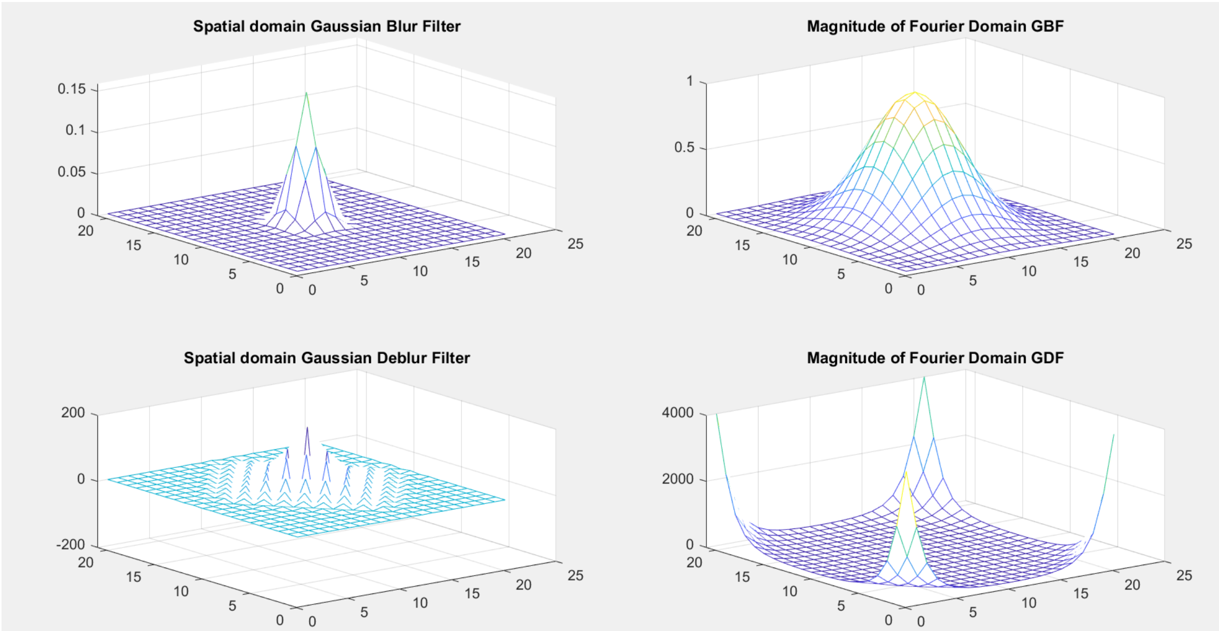

Original Text Image
Blurred Text Image
Deblurred Text Image
Original Image Frequency
Blurred Image Frequency
Deblurred Image Frequency

Figure 2:



Q4 Observation:

The blurring effect on the picture was caused by Gaussian noise. Darker points were found farther distant from the brightest point in the middle of the blurred picture's amplitude spectrum. When the original and deblurred images amplitude spectrums are examined, we can see that all of the spots are equally intense and similar in both. A low-pass filter and the Gaussian filter blurs an image (smoothening). The deblurred picture is almost similar to the original, with the high frequency components enhanced for a little gain in sharpness.

```
% Problem 5: Image Interpolation: Nearest Neighbor and Bilinear Interpolation
clear all;

input_image = imread('einstein.tif'); % reading the original image. This image is 8
bits/pixel gray-scale
% image.
size_of_image = size(input_image); % It will simply calculate the size of the image.

downsampling_factor = 3; % we are just defining the downsampling factor. It is as per
given in the problem
% statement.

% Downsampling the image in horizontal and vertical directions
downsampled_image_d3 = input_image((downsampling_factor+1)/2:downsampling_factor:end,
...
    (downsampling_factor+1)/2:downsampling_factor:end); % it will downsample the image
by a factor of D in
% horizontal and vertical directions, where the top-left sample should be at the
location of ((D+1)/2, (D+1)/2)
% in the original image.

% upsampling
upsampled_image_d3 = zeros(size_of_image); % It will simply upsample the image.
Upsampling of the downsampled
% image to have exactly the size of the original image by filling zeros in the missing
pixels.

p = 1; % initializing the value of p.
for i = (downsampling_factor+1)/2:downsampling_factor:size_of_image % for loop as per
the given condition.
    q = 1; % initializing the value of q.
    for j = (downsampling_factor+1)/2:downsampling_factor:size_of_image
    upsampled_image_d3(i,j) = downsampled_image_d3(p,q); % for loop as per the given
condition.
        q = q + 1; % incrementing the value of q.
    end % end for loop.
    p = p + 1; % incrementing the value of q.
end % end for loop.


% Convolution with block shape for D=3
block_filter_d3 = [1 1 1]; % creating the block filter of shape and size D.
block_conv_image_d3 = upsampled_image_d3; % the block convolutional filter shold be of
the same size as of the
% upsampled image.
for i = 1:size_of_image % for loop condition
block_conv_image_d3(i,:) = conv(block_conv_image_d3(i,:), block_filter_d3, 'same'); %
Here, the convolutions
% of the rows of the matrix. convolution operation for rows.
end % for loop ends
```

```
for j = 1:size_of_image % for loop condition
block_conv_image_d3(:,j) = conv(block_conv_image_d3(:,j), block_filter_d3.','same'); %↙
Here, the convolutions
% of the rows of the matrix. convolution operation for columns.
end % for loop ends

% Convolution with triangle shape for D=3
triangle_filter_d3 = (1/3) * [1 2 3 2 1]; % defining the filter as per the given↙
problem.
triangle_conv_image_d3 = upsampled_image_d3; % the block convolutional filter shold be↙
of the same size as of the
% upsampled image.
for i = 1:size_of_image % for loop condition
    triangle_conv_image_d3(i,:) = conv(triangle_conv_image_d3(i,:),↙
triangle_filter_d3,'same'); % Here, the
    % convolutions of the rows of the matrix. convolution operation for rows.
end % for loop ends
for j = 1:size_of_image % for loop condition
    triangle_conv_image_d3(:,j) = conv(triangle_conv_image_d3(:,j),↙
triangle_filter_d3.', 'same'); % Here, the
    % convolutions of the rows of the matrix. convolution operation for columns.
end % for loop ends



% Figure plotting for D=3
figure; % figure creates figure graphics objects. figure objects are the individual↙
windows on the screen
% in which MATLAB displays graphical output.
subplot(2,2,1); % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the
% position specified by p. Here, m=n=2, p=1.
imshow(input_image); % It will display the gray-scale image in the figure.
title('Original image'); % It will add the specified title for the current plot.
subplot(2,2,2); % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the
% position specified by p. Here, m=n=p=2.
imshow(uint8(upsampled_image_d3)); % It will display the gray-scale image in the figure↙
and
% it will convert each and every pixel value of the input image into the range of 0 to↙
255.
title('Upsampled image (D=3)'); % It will add the specified title for the current plot.
subplot(2,2,3); % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the
% position specified by p. Here, m=n=2, p=3.
imshow(uint8(block_conv_image_d3)); % It will display the gray-scale image in the↙
figure and
% it will convert each and every pixel value of the input image into the range of 0 to↙
255.
title('Interpolated image using Nearest Neighbour (D=3)'); % It will add the specified↙
```

```
title for the current plot.
subplot(2,2,4); % subplot(m, n, p) divides the current figure into an m-by-n grid and
creates axes in the
% position specified by p. Here, m=n=2, p=4.
imshow(uint8(triangle_conv_image_d3)); % It will display the gray-scale image in the
figure and
% it will convert each and every pixel value of the input image into the range of 0 to
255.
title('Image using Bilinear interpolation (D=3)'); % It will add the specified title
for the current plot.



% D = 7
downsampling_factor = 7; % we are just defining the downsampling factor. It is as per
given in the problem
% statement.

%Downsampling the image in horizontal and vertical directions
downsampled_image_d7 = input_image((downsampling_factor+1)/2:downsampling_factor:end,
...
    (downsampling_factor+1)/2:downsampling_factor:end); % it will downsample the image
by a factor of D in
% horizontal and vertical directions, where the top-left sample should be at the
location of ((D+1)/2, (D+1)/2)
% in the original image.

%upsampling
upsampled_image_d7 = zeros(size_of_image); % It will simply upsample the image.
Upsampling of the downsampled
% image to have exactly the size of the original image by filling zeros in
% the missing pixels.

p = 1; % initializing the value of p.
for i = (downsampling_factor+1)/2:downsampling_factor:size_of_image % for loop as per
the given condition.
    q = 1; % initializing the value of q.
    for j = (downsampling_factor+1)/2:downsampling_factor:size_of_image
    upsampled_image_d7(i,j) = downsampled_image_d7(p,q); % for loop as per the given
condition.
        q = q + 1; % incrementing the value of q.
    end % end for loop.
    p = p + 1; % incrementing the value of q.
end % end for loop.


% Convolution with block shape for D=7
block_filter_d7 = [1 1 1 1 1 1 1];  % defining the filter as per the given problem.
block_convoluted_image_d7 = upsampled_image_d7; % the block convolutional filter shold
be of the same size as
```

```
% of the upsampled image.
for i = 1:size_of_image % for loop condition
block_convoluted_image_d7(i,:) = conv(block_convoluted_image_d7(i,:), block_filter_d7,
'same'); % Here, the
    % convolutions of the rows of the matrix. convolution operation for rows.
end % for loop ends
for j = 1:size_of_image % for loop condition
block_convoluted_image_d7(:,j) = conv(block_convoluted_image_d7(:,j),
block_filter_d7.','same'); % Here, the
    % convolutions of the rows of the matrix. convolution operation for columns.
end % for loop ends

% Convolution with triangle shape for D=7
triangle_filter_d7 = (1/7) * [1 2 3 4 5 6 7 5 4 3 2 1]; % defining the filter as per
the given problem.
triangle_convoluted_image_d7 = upsampled_image_d7; % the block convolutional filter
shold be of the same size
% as of the upsampled image.
for i = 1:size_of_image % for loop condition
    triangle_convoluted_image_d7(i,:) = conv(triangle_convoluted_image_d7(i,:),
triangle_filter_d7,'same');
    % Here, the
    % convolutions of the rows of the matrix. convolution operation for rows.
end % for loop ends
for j = 1:size_of_image % for loop condition
    triangle_convoluted_image_d7(:,j) = conv(triangle_convoluted_image_d7(:,j),
triangle_filter_d7.', 'same');
    % Here, the
    % convolutions of the rows of the matrix. convolution operation for columns.
end % for loop ends



% Figure plotting for D=7
figure; % figure creates figure graphics objects. figure objects are the individual
windows on the screen
% in which MATLAB displays graphical output.
subplot(2,2,1); % subplot(m, n, p) divides the current figure into an m-by-n grid and
creates axes in the
% position specified by p. Here, m=n=2, p=1.
imshow(input_image); % It will display the gray-scale image in the figure.
title('Original image'); % It will add the specified title for the current plot.
subplot(2,2,2); % subplot(m, n, p) divides the current figure into an m-by-n grid and
creates axes in the
% position specified by p. Here, m=n=p=2.
imshow(uint8(upsampled_image_d7)); % It will display the gray-scale image in the figure
and
% it will convert each and every pixel value of the input image into the range of 0 to
255.
title('Upsampled image (D=7)'); % It will add the specified title for the current plot.
```

```
subplot(2,2,3); % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the
% position specified by p. Here, m=n=2, p=3.
imshow(uint8(block_convoluted_image_d7)); % It will display the gray-scale image in the↙
figure and
% it will convert each and every pixel value of the input image into the range of 0 to↙
255.
title('Interpolated image using Nearest Neighbour (D=7)'); % It will add the specified↙
title for the current plot.
subplot(2,2,4); % subplot(m, n, p) divides the current figure into an m-by-n grid and↙
creates axes in the
% position specified by p. Here, m=n=2, p=4.
imshow(uint8(triangle_convoluted_image_d7)); % It will display the gray-scale image in↙
the figure and
% it will convert each and every pixel value of the input image into the range of 0 to↙
255.
title('Image using Bilinear interpolation (D=7)'); % It will add the specified title↙
for the current plot.
```

Figure 1:

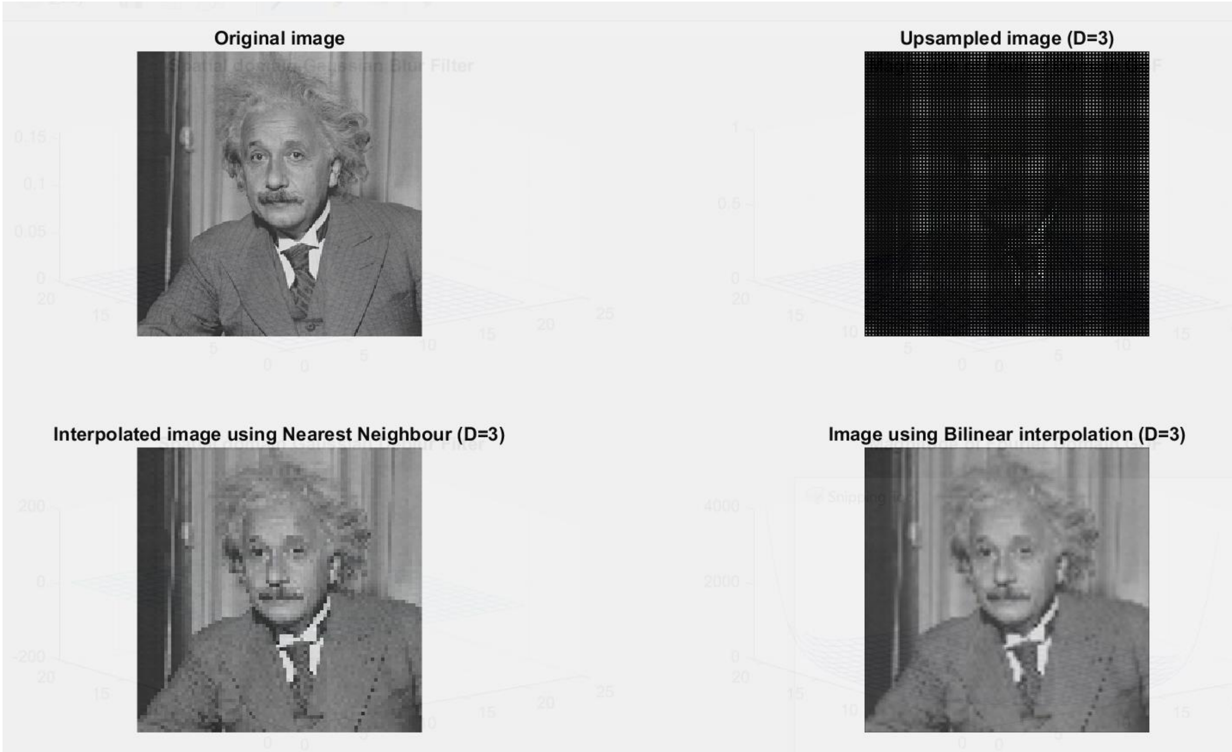
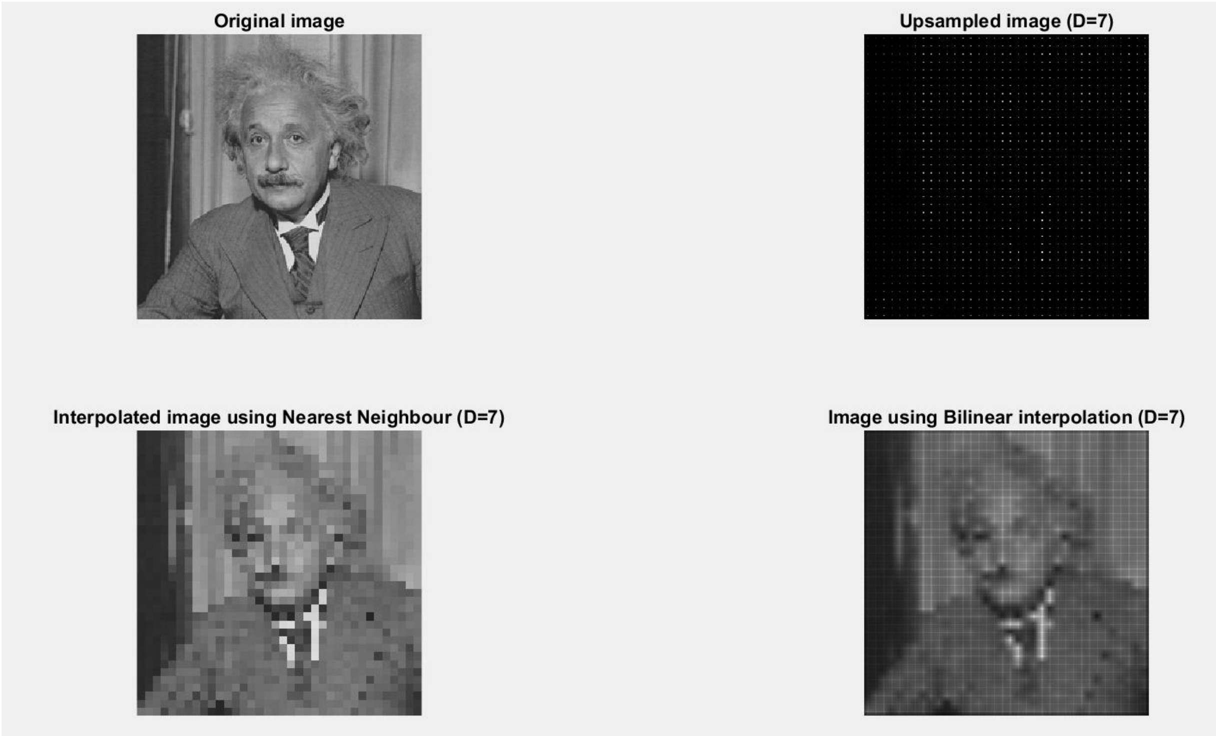

Original image
Upsampled image (D=3)
Interpolated image using Nearest Neighbour (D=3)
Image using Bilinear interpolation (D=3)

Figure 2:



Q5 Observation:

After downsampling the original image, we determined that the Nearest Neighbor Interpolation strategy was the most effective in restoring the image to its original condition. The bilinear interpolation is followed by the up sampled image. When applied to the original image, the Nearest Neighbor algorithm creates blocky results. This is because in the closest neighbor method, we find the next-closest pixel and assume its intensity value. As a result, the image takes on a blocky appearance. When compared to closest neighbor interpolation, bilinear interpolation estimates a new pixel value using a distance weighted average of the D - nearest pixel values, resulting in a smoother image.