# ECE 7410/ENGI 9804 IMAGE PROCESSING AND APPLICATIONS

# Assignment 2

**STUDENT NAME:**

**NAYEM AL TAREQ : 202293442**

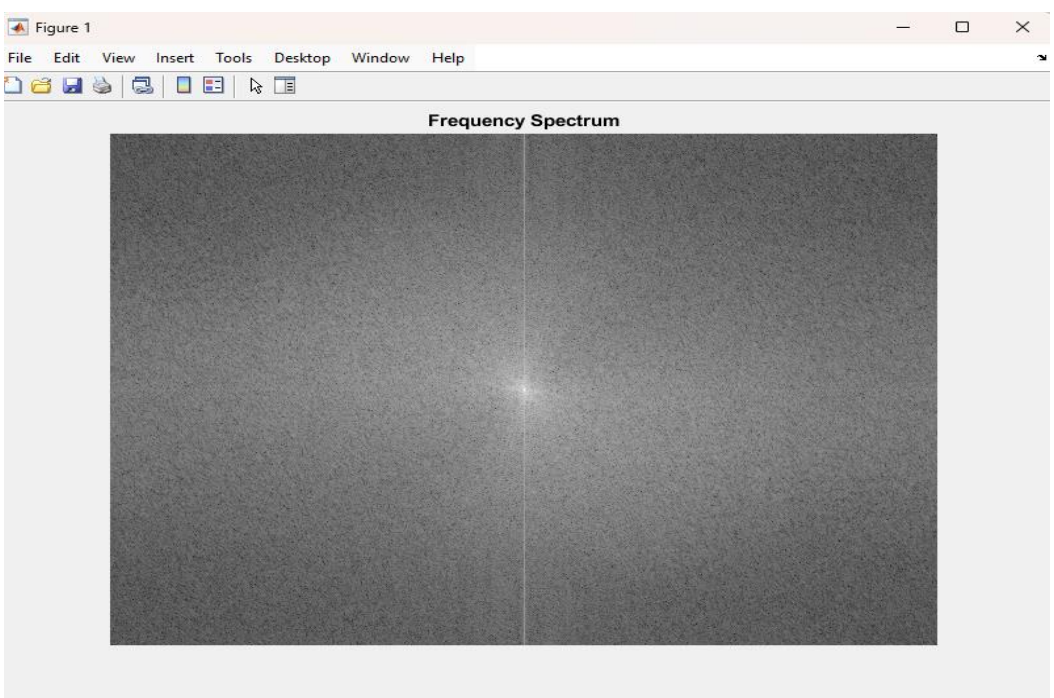
1. [5] Convert to the frequency domain and center the spectrum. Calculate the maximum value of the frequency spectrum (use MATLAB command $FS_{max} = \max(FS(:))$, where $FS$ is the frequency spectrum and $FS_{max}$ is its maximum value).

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

% Read the image file 'moonlanding.png'
img = imread('moonlanding.png');

% Convert the image to double precision for FFT
img_double = double(img);

% Perform the 2-dimensional Fast Fourier Transform (FFT) on the image
fft_img = fft2(img_double);

% Shift the zero-frequency component to the center of the Fourier Transform
fft_shifted = fftshift(fft_img);

% Compute the magnitude of the Fourier Transform
fft_mag = abs(fft_shifted);

% Calculate the maximum value of the frequency spectrum
FSmax = max(fft_mag(:));

% Display the magnitude of the Fourier Transform as an image with automatic scaling and add a title
figure, imshow(log(1 + fft_mag), []), title('Frequency Spectrum');

% Display the maximum value of the frequency spectrum
disp(['Maximum value of the frequency spectrum: ', num2str(FSmax)]);
```

Command Window

```
Maximum value of the frequency spectrum: 23229905
```

2. [5] Display the frequency spectrum obtained in Question 1 as an image.



3. [25] Using the centered frequency spectrum obtained in Question 1, manually introduce noise at eight points. The points *must* be located on the circle at a distance of 100 from the center point of the image and having 45° angle increments at the four cardinal and intercardinal directions (N,S,E,W,NE,NW,SE,SW). Find the $3 \times 3$ neighborhoods of these eight elements and set their values to $FS_{max}/10$ where $FS_{max}$ was obtained in Question 1. Display the resultant spectrum as an image.

```
clc; clear all; close all;

% Read the image file 'moonlanding.png'
img = imread('moonlanding.png');

% Convert the image to double precision for FFT
img_double = double(img);

% Perform the 2-dimensional Fast Fourier Transform (FFT) on the image
fft_img = fft2(img_double);

% Shift the zero-frequency component to the center of the Fourier Transform
centered_fft_image = fftshift(fft_img);

% Compute the magnitude of the Fourier Transform
fft_mag = abs(centered_fft_image);

% Calculate the maximum value of the frequency spectrum
max_frequency_value = max(fft_mag(:));

% Get the size of the centered Fourier transform image
[rows, cols] = size(centered_fft_image);

% Calculate the center coordinates of the image
center = [rows / 2, cols / 2];

% Define the distance for the noise points from the center
dist = 100;

% Define the angle in radians for diagonal points
angle = pi / 4;

% Reduce the maximum frequency spectrum value to introduce noise
new_max_freq = max_frequency_value / 10;

% Define the coordinates for the points in the north, east, south, and west directions
N = [center(1) - dist, center(2)];
E = [center(1), center(2) + dist];
S = [center(1) + dist, center(2)];
W = [center(1), center(2) - dist];

% Define the coordinates for the diagonal points (northeast, northwest, southeast, southwest)
NE = [round(center(1) - (sin(angle) * dist)), round(center(2) + (cos(angle) * dist))];
NW = [round(center(1) - (sin(angle) * dist)), round(center(2) - (cos(angle) * dist))];
```

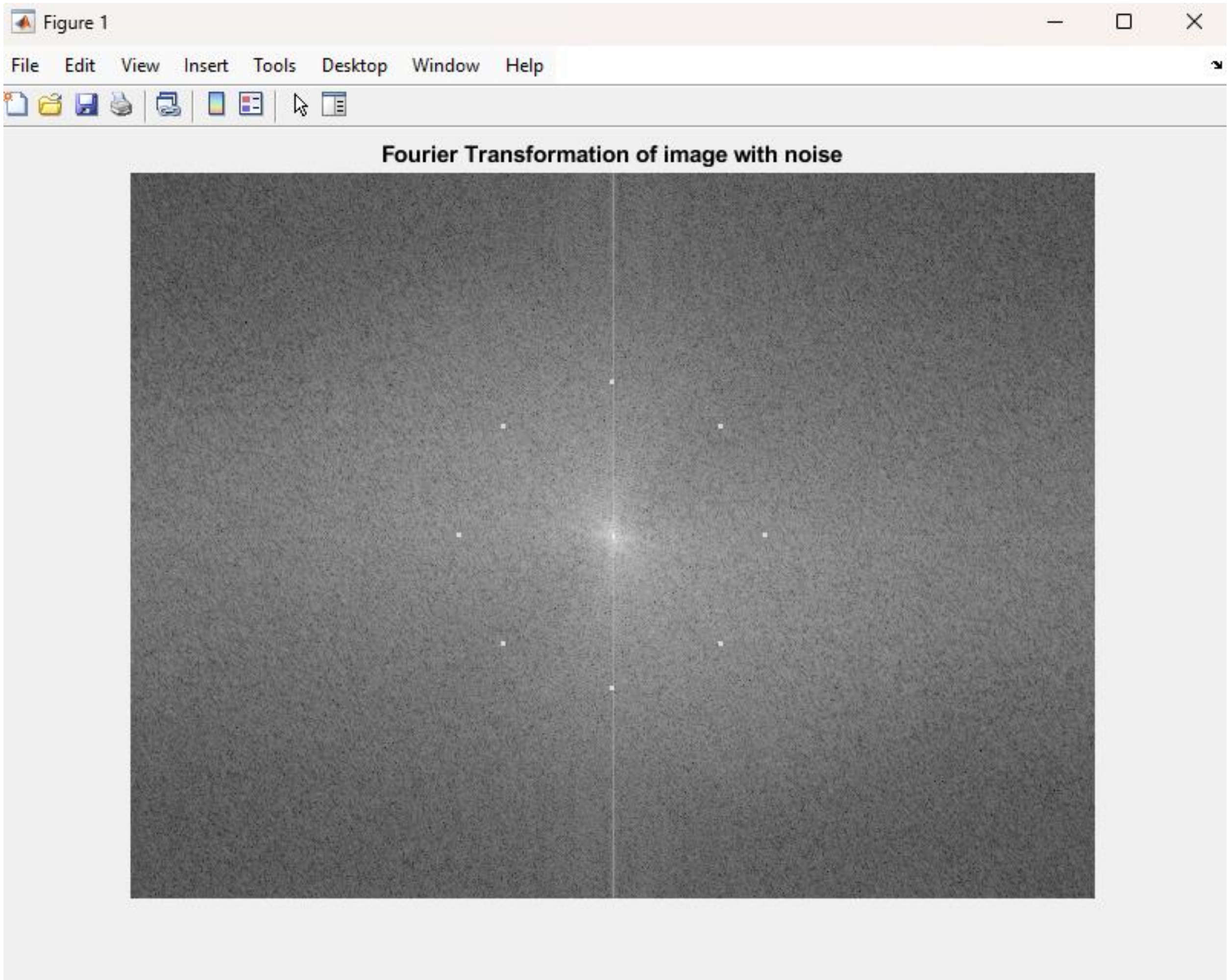
```
SE = [round(center(1) + (sin(angle) * dist)), round(center(2) + (cos(angle) * dist))];
SW = [round(center(1) + (sin(angle) * dist)), round(center(2) - (cos(angle) * dist))];

% Copy the original centered Fourier transform image to add noise
noisy_fft = centered_fft_image;

% Define a placeholder function for adding noise; replace 'noise_func' with the actual function name
% noise_func should be a function that adds noise to the specific point in the frequency domain

for idx = 1:8
   % Combine all the points into a matrix
   points = [N; E; S; W; NE; NW; SE; SW];

   % Select the current point
   point = points(idx, :);

   % Apply manual noise to the selected point in the Fourier transform image
   % You need to replace 'noise_func' with the actual noise-adding function
   noisy_fft = noise_func(noisy_fft, point, new_max_freq); % Replace 'manual_noise' with the actual function if different
end

% Compute the magnitude of the noisy Fourier transform and apply logarithmic scaling
noisy_fft_mag = log(1 + abs(noisy_fft));

% Display the noisy Fourier transform as an image with automatic scaling and add a title
figure, imshow(noisy_fft_mag, []), title('Fourier Transformation of image with noise');
```

## Noise_function

noise_func.m

```
function [result] = noise_func(img, point, noise)
% This function gets an image and desired point,
% then apply noise(value) on the 3*3 neighboring pixels
for i = -1:1
    for j =-1:1
        img((point(1)+i), (point(2)+j)) = noise;
    end
end
result = img;
end
```

Output:

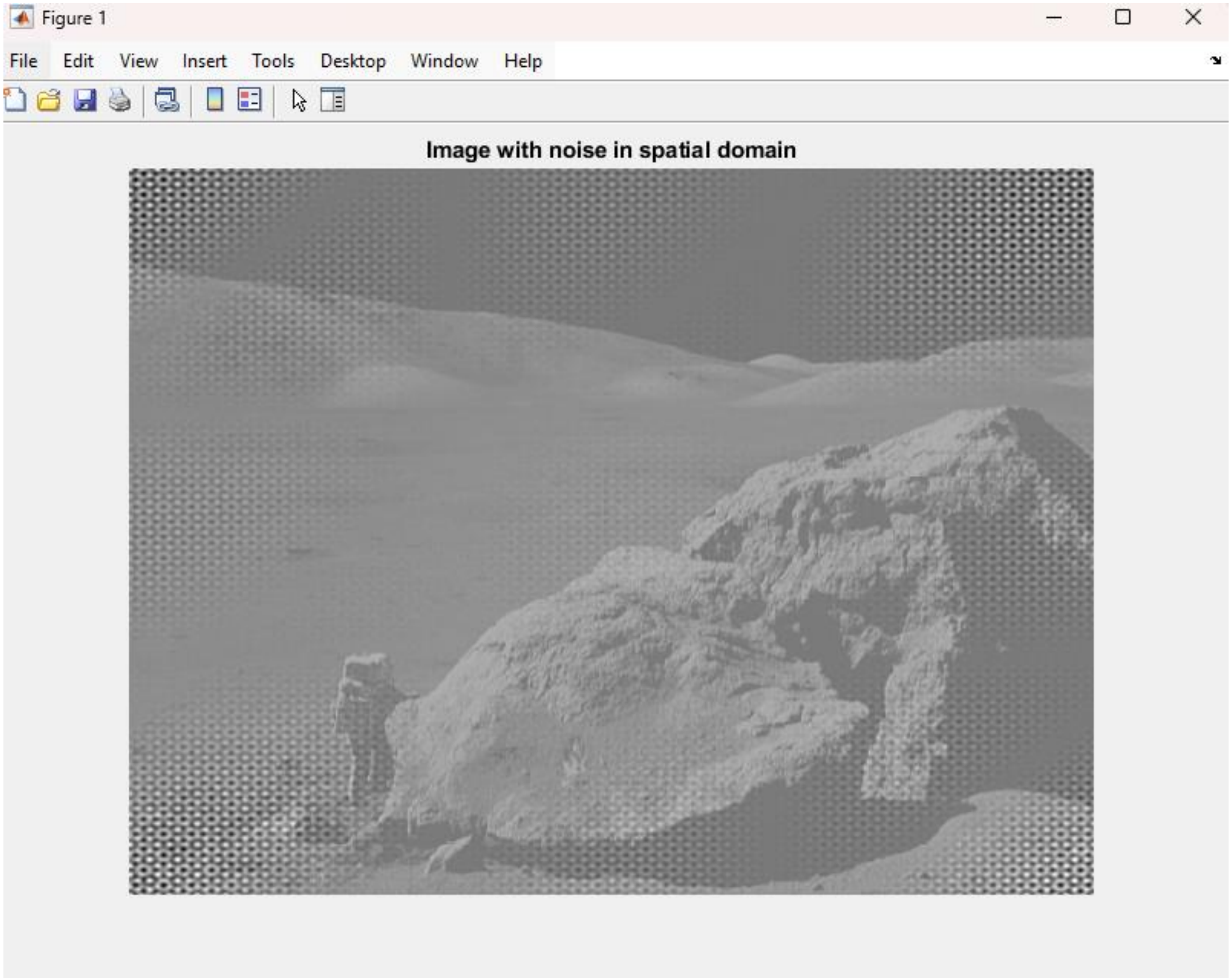



4. [5] Convert the frequency domain representation in Question 3 to the spatial domain, scale it and display the result. Observe the result of adding noise, and comment briefly on the noisy image.

Q4.m

```
% Shift the noisy Fourier transform image to center the zero-frequency component
fft_noise_centered = fftshift(noisy_fft);

% Perform the inverse 2-dimensional Fast Fourier Transform to convert back to the spatial domain
spatial_img = real(ifft2(fft_noise_centered));

% Display the noisy image in the spatial domain with automatic scaling and add a title
figure, imshow(spatial_img, []), title('Image with noise in spatial domain');
```

Output :



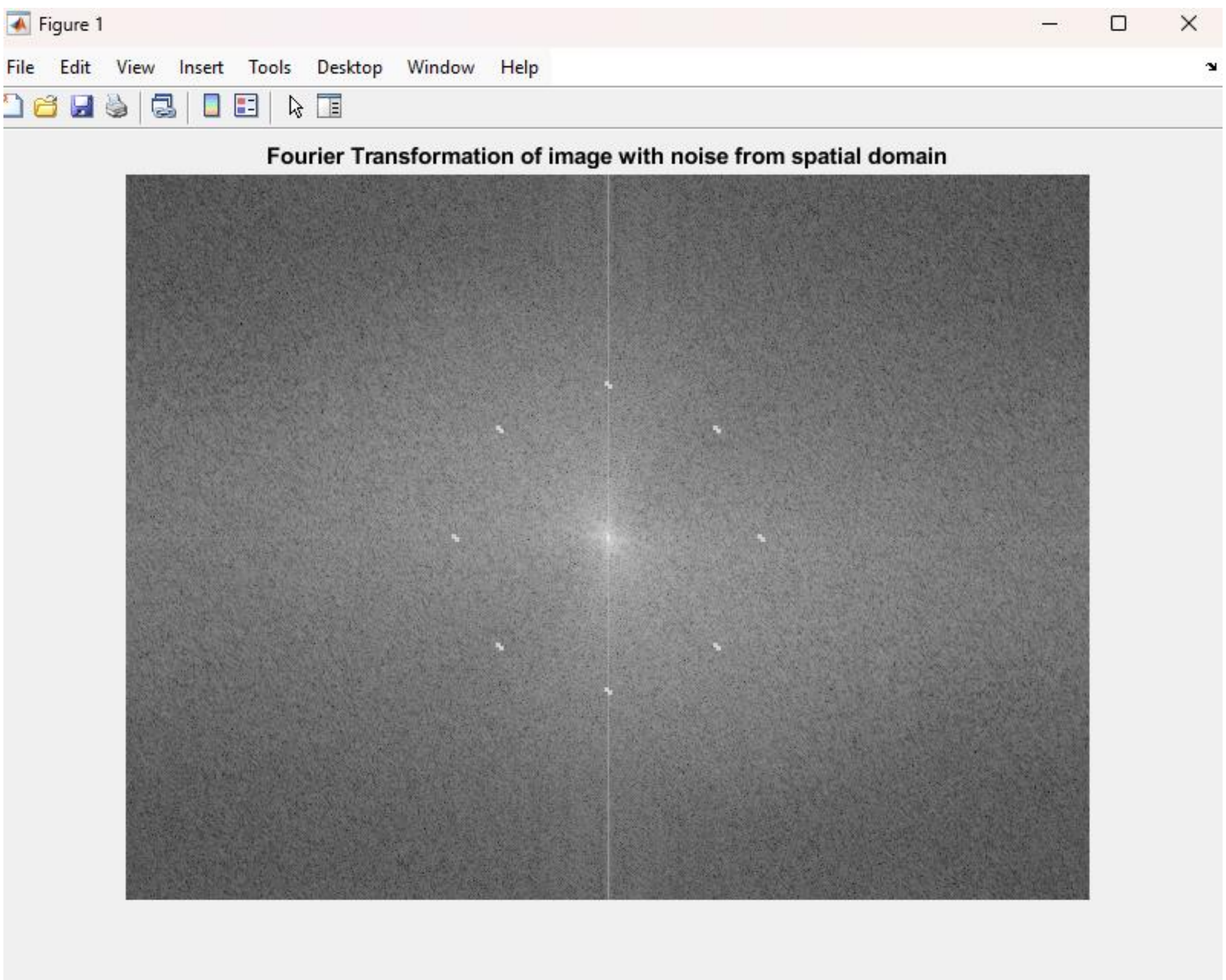

**Comment:** The noisy image displays a regular pattern of circular artifacts and blurring, resulting from the specific points where noise was added in the frequency domain. Despite this, the astronaut and rock formations remain visible, though less sharp than in the original image. The noise patterns create a grainy texture, disrupting the smoothness of the original image and degrading its overall quality. This demonstrates how frequency domain manipulations can significantly impact the spatial domain representation, introducing high-frequency components that manifest as artifacts and blurring, thereby emphasizing the importance of understanding these effects.

5. [5] Now, using the image from Question 4 (which is corrupted with periodic noise), obtain the frequency domain representation, center the spectrum and display it. Comment on whether or not it looks like the frequency spectrum you manually created in Question 3.

Q5.m

```
% Perform the 2-dimensional Fast Fourier Transform on the noisy spatial domain image
fft_spatial = fft2(spatial_img);

% Shift the zero-frequency component to the center of the Fourier Transform of the noisy spatial image
fft_shifted_spatial = fftshift(fft_spatial);

% Compute the magnitude of the Fourier Transform and apply logarithmic scaling to enhance visibility
fft_mag_spatial = log(1 + abs(fft_shifted_spatial));

% Display the magnitude of the Fourier Transform as an image with automatic scaling and add a title
figure, imshow(fft_mag_spatial, []), title('Fourier Transformation of image with noise from spatial domain');
```

Output:



**Comment:** The frequency domain representation of the noisy spatial domain image shows distinct bright spots around the center, indicating periodic noise. These spots correspond to specific frequency components added to the image, matching the expected pattern from manually adding noise in the frequency domain. The centered spectrum clearly reveals these components, confirming the consistency between spatial domain manipulations and their frequency domain representations. This image closely resembles the manually created frequency spectrum, validating the effects of periodic noise and demonstrating the alignment between spatial and frequency domain alterations.

6. [30] Assuming the cutoff frequency $D_0$=100 and the width $W$= 8, design:

   (a) Ideal

   (b) Butterworth with order 4 (i.e. n=4)

   (c) Gaussian

Q6.m

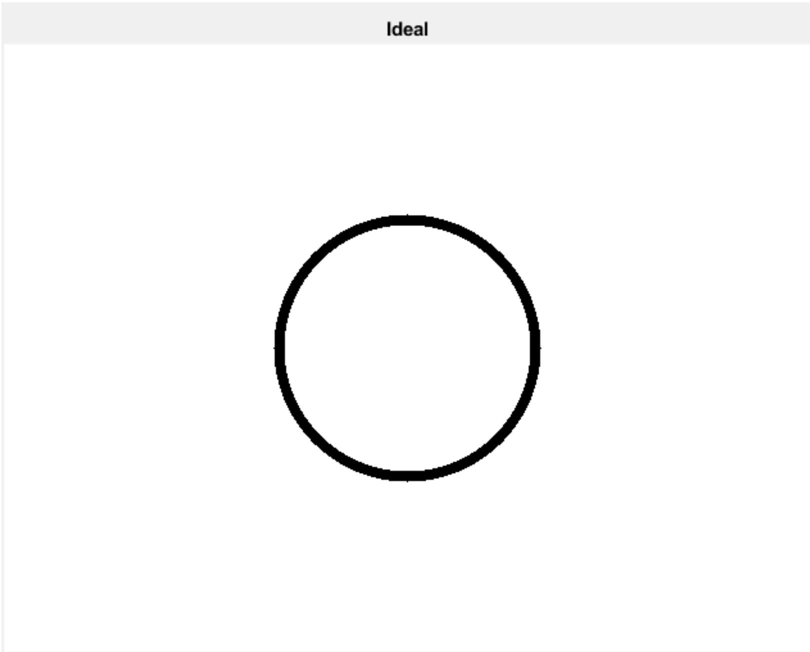
```
% Read the image file 'moonlanding.png'
img = imread('moonlanding.png');

% Perform the 2-dimensional Fast Fourier Transform (FFT) on the image matrix img
fft_img = fft2(img);

% Shift the zero-frequency component to the center of the Fourier Transform
fft_shifted = fftshift(fft_img);

% Compute the magnitude of the Fourier Transform and apply logarithmic scaling to enhance visibility
fft_mag = log(1 + abs(fft_shifted));

% Calculate the maximum value in the frequency spectrum
max_freq = max(fft_shifted(:));

% Display the magnitude of the Fourier Transform as an image with automatic scaling and add a title
figure, imshow(fft_mag, []), title('Fourier Transformation of Original');

% Shift the noisy Fourier transform image to center the zero-frequency component
fft_noise_centered = fftshift(noisy_fft);

% Perform the inverse 2-dimensional Fast Fourier Transform to convert back to the spatial domain
spatial_img = real(ifft2(fft_noise_centered));

% Display the noisy image in the spatial domain with automatic scaling and add a title
figure, imshow(spatial_img, []), title('Image with noise in spatial domain');

% Perform the 2-dimensional Fast Fourier Transform on the noisy spatial domain image
fft_spatial = fft2(spatial_img);

% Shift the zero-frequency component to the center of the Fourier Transform of the noisy spatial image
fft_shifted_spatial = fftshift(fft_spatial);

% Compute the magnitude of the Fourier Transform and apply logarithmic scaling to enhance visibility
fft_mag_spatial = log(1 + abs(fft_shifted_spatial));

% Display the magnitude of the Fourier Transform as an image with automatic scaling and add a title
figure, imshow(fft_mag_spatial, []), title('Fourier Transformation of image with noise from spatial domain');

% Define the cutoff frequency and the bandwidth for the filters
cutoff_freq = 100;
bandwidth = 8;
```

```
% Get the size of the Fourier transform magnitude image
[rows, cols] = size(fft_mag_spatial);

% Initialize the ideal filter with ones
ideal_filter = ones(rows, cols);

% Calculate the center coordinates of the filter
center_x = round(rows / 2);
center_y = round(cols / 2);

% Create the ideal bandstop filter
for i = 1:rows
   for j = 1:cols
       % Calculate the distance from the center
       dist_x = i - center_x;
       dist_y = j - center_y;
       distance = sqrt(dist_x^2 + dist_y^2);

       % Apply the ideal bandstop filter condition
       if (cutoff_freq - bandwidth / 2) <= distance && distance <= (cutoff_freq + bandwidth / 2)
           ideal_filter(i, j) = 0;
       end
   end
end

% Initialize the Butterworth filter with ones
butterworth_filter = ones(rows, cols);

% Define the order of the Butterworth filter
order = 4;

% Create the Butterworth bandstop filter
for i = 1:rows
   for j = 1:cols
       % Calculate the distance from the center
       dist_x = i - center_x;
       dist_y = j - center_y;
       distance = sqrt(dist_x^2 + dist_y^2);

       % Apply the Butterworth bandstop filter formula
       butterworth_filter(i, j) = 1 / (1 + (distance * bandwidth / (distance^2 - cutoff_freq^2)^2 * order));
   end
end
```

```
% Initialize the Gaussian filter with ones
gaussian_filter = ones(rows, cols);

% Create the Gaussian bandstop filter
for i = 1:rows
   for j = 1:cols
       % Calculate the distance from the center
       dist_x = i - center_x;
       dist_y = j - center_y;
       distance = sqrt(dist_x^2 + dist_y^2);

       % Apply the Gaussian bandstop filter formula
       gaussian_filter(i, j) = 1 - exp(-((distance^2 - cutoff_freq^2) / distance * bandwidth)^2);
   end
end
```

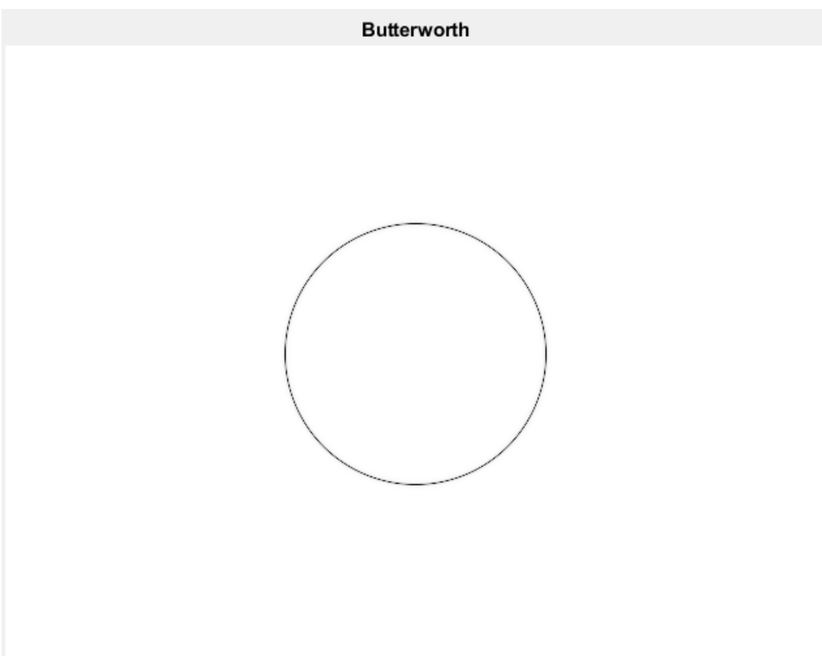
a)Ideal

```
% Display the ideal filter
figure, imshow(ideal_filter), title('Ideal');
```



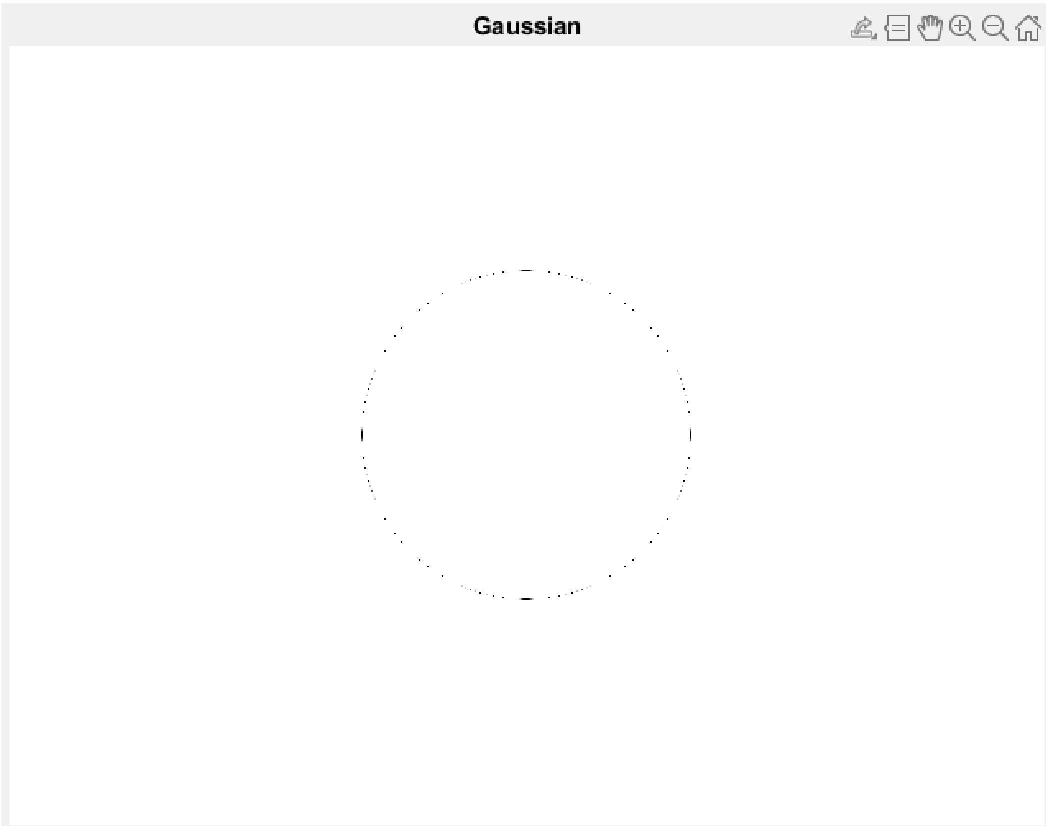

(b) Butterworth with order 4 (i.e. n=4)

```
% Display the Butterworth filter with order n=4
figure, imshow(butterworth_filter), title('Butterworth');
```



c) Gaussian

```
% Display the Gaussian filter
figure, imshow(gaussian_filter), title('Gaussian');
```



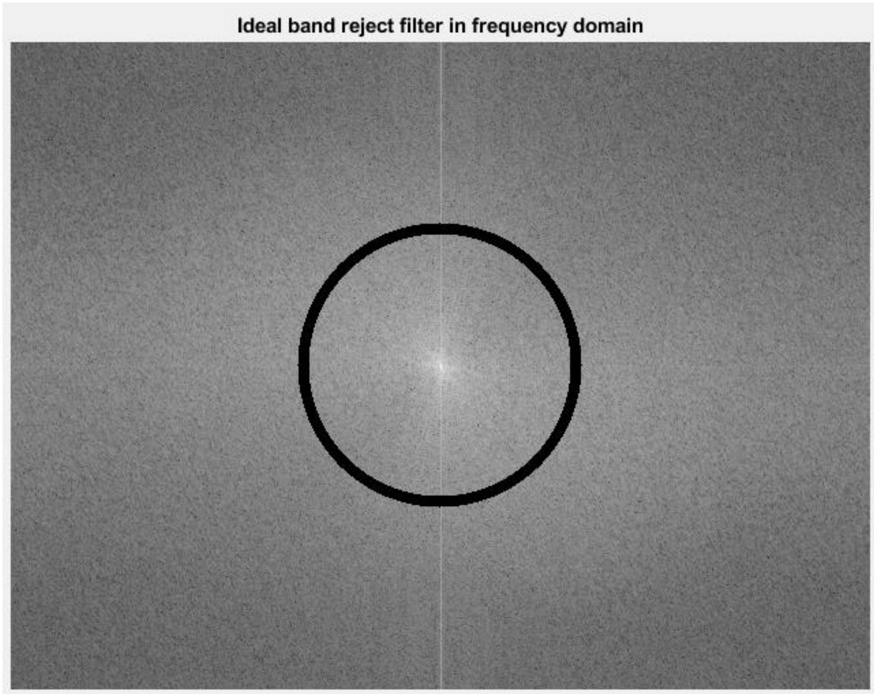

7. [10] Implement the band reject filters you developed in Question 6 to remove the noise you also introduced using element-wise multiplication. Display the results in the frequency domain for each filter individually.

Q7.m

```
% Apply the ideal band reject filter to the noisy Fourier transform image
img_ideal_filtered = noisy_fft_mag .* ideal_filter;

% Apply the Butterworth band reject filter to the noisy Fourier transform image
img_butter_filtered = noisy_fft_mag .* butterworth_filter;

% Apply the Gaussian band reject filter to the noisy Fourier transform image
img_gaussian_filtered = noisy_fft_mag .* gaussian_filter;
```

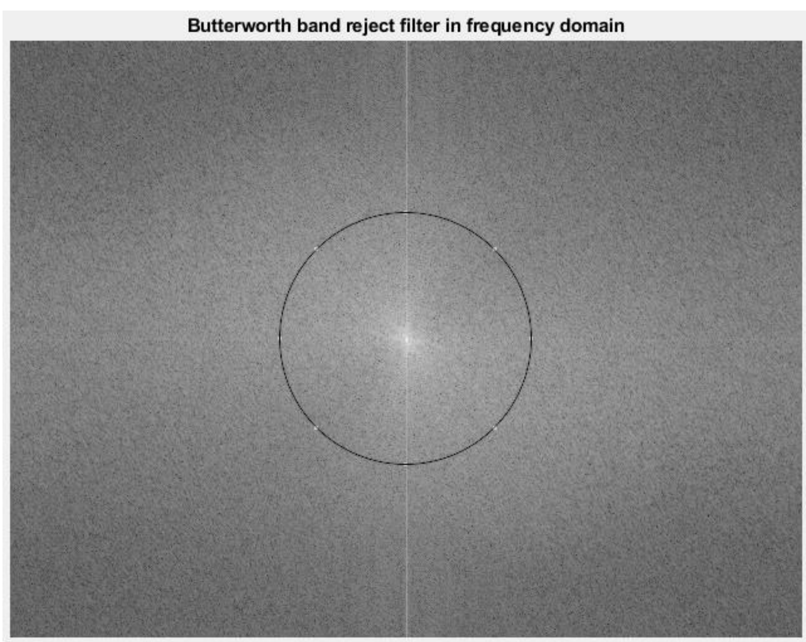
**a)Ideal Filter**

```
% Display the result of applying the ideal band reject filter in the frequency domain
figure, imshow(img_ideal_filtered, []), title('Ideal band reject filter in frequency domain');
```



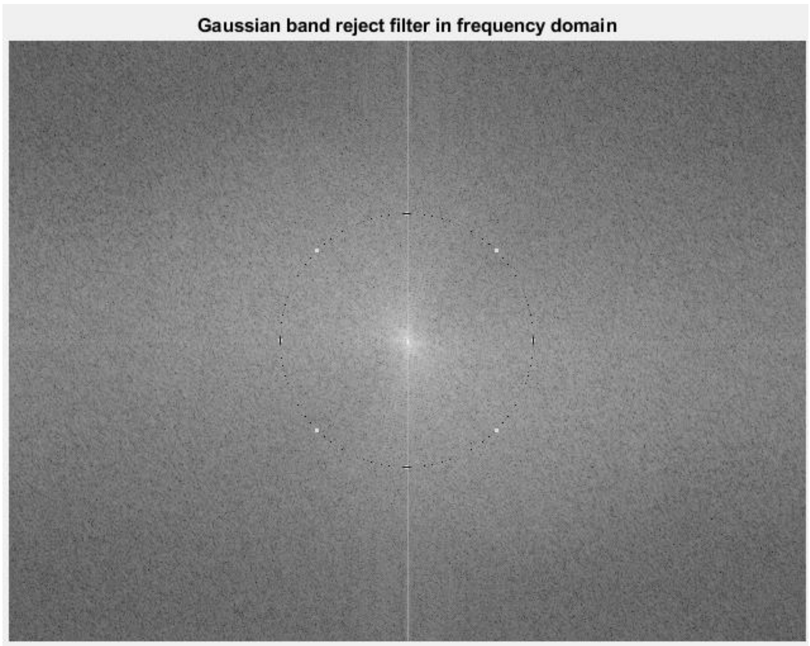

## b) Butterworth Filter

```
% Display the result of applying the Butterworth band reject filter in the frequency domain
figure, imshow(img_butter_filtered, []), title('Butterworth band reject filter in frequency domain');
```



c)Gaussian Filter

```
% Display the result of applying the Gaussian band reject filter in the frequency domain
figure, imshow(img_gaussian_filtered, []), title('Gaussian band reject filter in frequency domain');
```



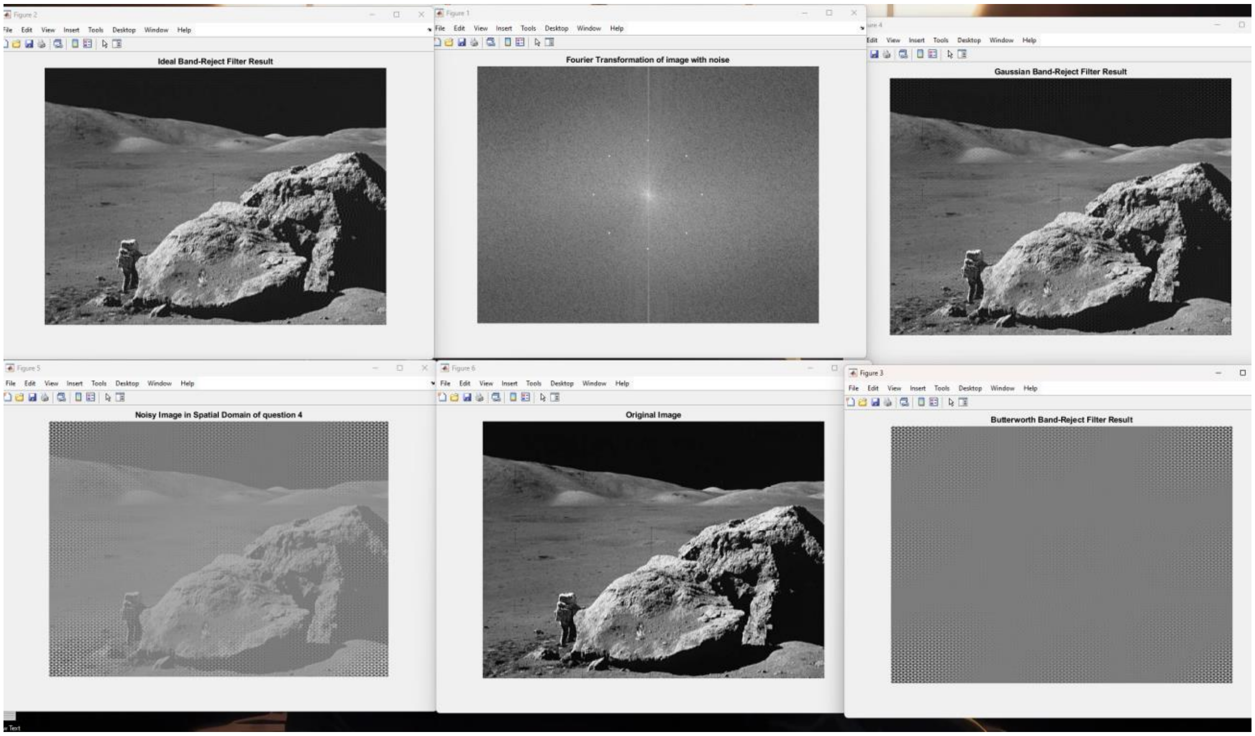
8. [10] Convert the three frequency domain representations of the filtered image found in Question 7 to the spatial domain, scale them and display the results. Compare the resultant images:

   (a) with the noisy image in Question 4.

   (b) with the original image, *moonlanding.png*.

Solution:

```
% Apply the ideal band reject filter to the noisy Fourier transform image
img_ideal_filtered = noisy_fft .* ideal_filter;

% Apply the Butterworth band reject filter to the noisy Fourier transform image
img_butter_filtered = noisy_fft .* butterworth_filter;

% Apply the Gaussian band reject filter to the noisy Fourier transform image
img_gaussian_filtered = noisy_fft .* gaussian_filter;

% Convert the filtered images back to the spatial domain
ideal_spatial = real(ifft2(ifftshift(img_ideal_filtered)));
butter_spatial = real(ifft2(ifftshift(img_butter_filtered)));
gaussian_spatial = real(ifft2(ifftshift(img_gaussian_filtered)));

% Display the results of applying the filters in the spatial domain
figure, imshow(ideal_spatial, []), title('Ideal Band-Reject Filter Result');
figure, imshow(butter_spatial, []), title('Butterworth Band-Reject Filter Result');
figure, imshow(gaussian_spatial, []), title('Gaussian Band-Reject Filter Result');

% Display the noisy image from Question 4 for comparison
spatial_img = real(ifft2(ifftshift(noisy_fft)));
figure, imshow(spatial_img, []), title('Noisy Image in Spatial Domain of question 4')

% Display the original image for comparison
figure, imshow(img_double, []), title('Original Image');
```




9. [5] Comment on the differences in the performance of the three band-reject filters used, with respect to their ability to recover the original image.

Solution:

The Ideal, Butterworth, and Gaussian band-reject filters each handle noise removal differently. The Ideal filter is very effective at removing specific frequencies but often introduces noticeable ringing artifacts due to its abrupt cutoff. The Butterworth filter provides a smoother transition, effectively reducing noise while minimizing artifacts, striking a balance between noise removal and image quality preservation. The Gaussian filter offers the smoothest transition, reducing noise with minimal artifacts and best preserving overall image quality. Thus, the Ideal filter is best for complete noise removal at the cost of artifacts, the Butterworth filter offers a good compromise, and the Gaussian filter provides the best overall performance with minimal artifacts.

Referance : Google, Chatgpt ,Matlab website,gemni .