
# Biomedical Signal and Image Processing Lab

## Lab 7

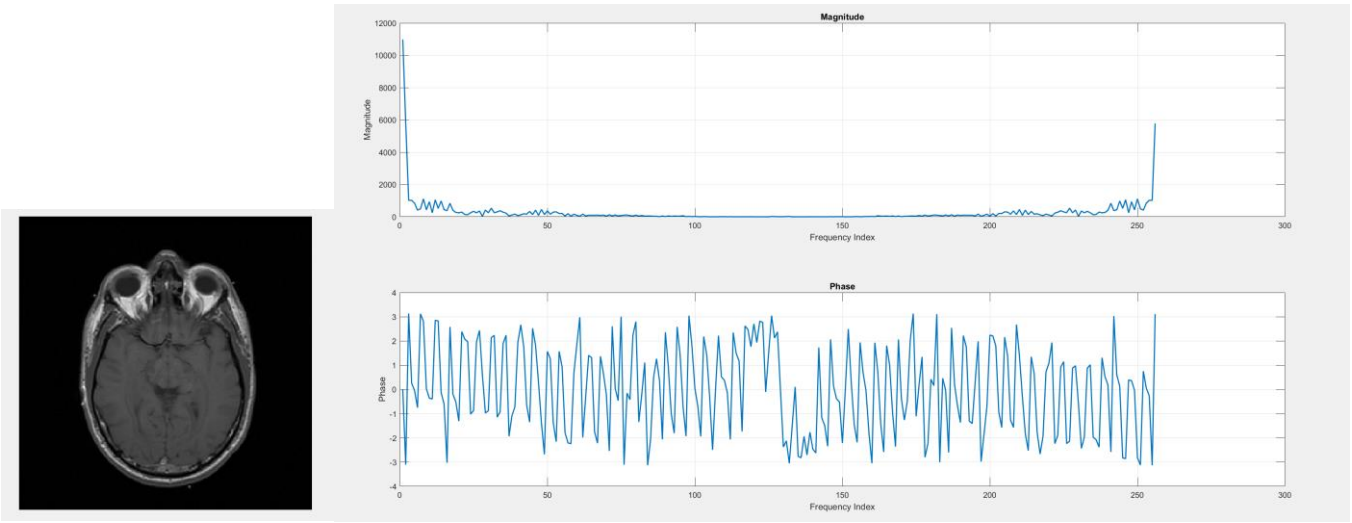
**Ali Khosravipour - 99101502**

**Mohmmadhossein Faramarzi - 99104095**

**Sara Rezanejad - 99101643**

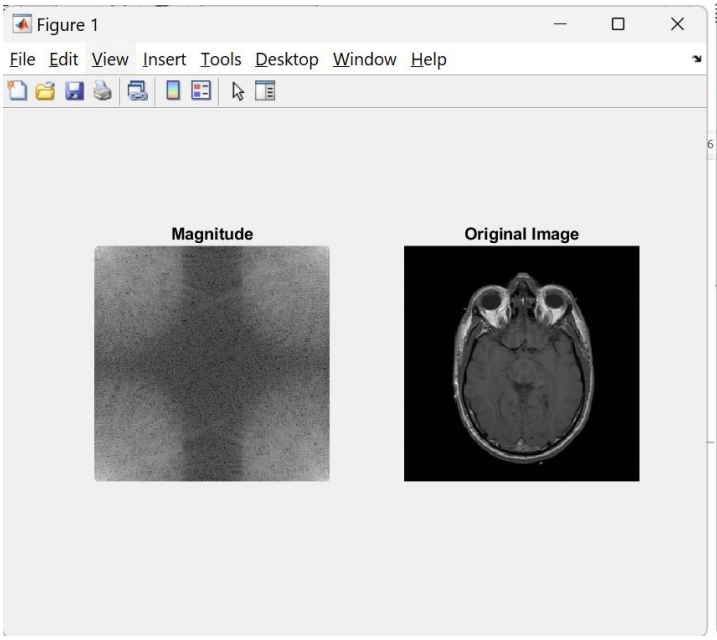
## Question 1)

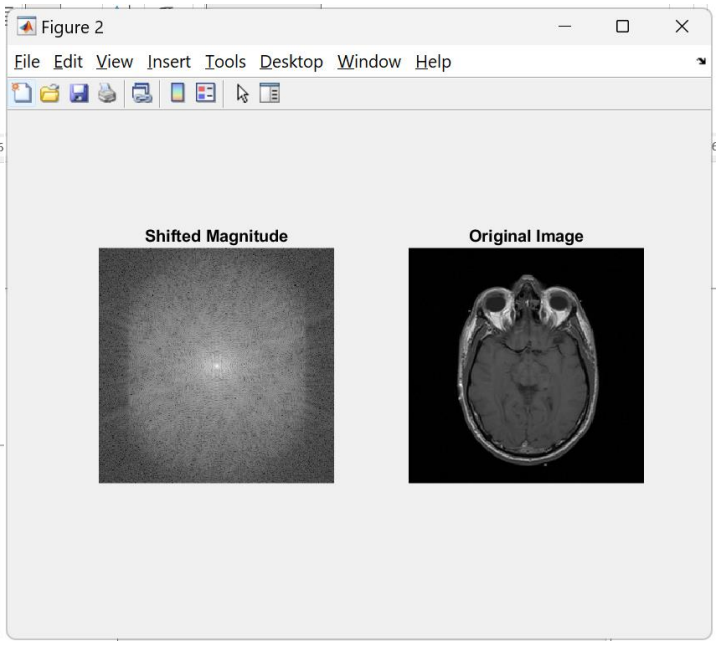
```
% Lab 7
% Ali Khosravipour 99101502 - Mohamad Hosein Faramarzi 99104095 - Sara
% Rezanejad 99101643
%% Q1
clc;
t1_image = imread("t1.jpg");
first_slice = t1_image(:,:,1);
figure;
imshow(first_slice);
%%
row_128 = first_slice(128,:);
dft_128 = fft(row_128);
mag_128 = abs(dft_128);
phase_128 = angle(dft_128);
disp('128th Row DFT Magnitude:'), disp(mag_128);
disp('128th Row DFT Phase:'), disp(phase_128);
figure;
subplot(2, 1, 1);
plot(mag_128, 'LineWidth', 1.5);
title('Magnitude');
xlabel('Frequency Index');
ylabel('Magnitude');
grid on;
subplot(2, 1, 2);
plot(phase_128, 'LineWidth', 1.5);
title('Phase');
xlabel('Frequency Index');
ylabel('Phase');
grid on;
```




The top plot shows the **magnitude spectrum** of the row, which indicates the frequency components' intensities. The high magnitude at the start corresponds to the low-frequency content, while the spike at the end may indicate a sharp feature or aliasing in the data. The bottom plot shows the **phase spectrum**, which represents the phase angle of each frequency component. The phase values oscillate between −π-\pi−π and π\piπ, capturing the alignment of sinusoidal components. This analysis highlights both the amplitude and phase information of the spatial frequency content in the image row.

```
%%
first_slice_double = double(first_slice);
dft_img = fft2(first_slice_double);
magnitude = abs(dft_img);
figure;
subplot(1,2,1);
imshow(log(1 + magnitude), []);
title('Magnitude');
subplot(1,2,2);
imshow(first_slice);
title('Original Image');
%% fftshift
dft_shifted = fftshift(dft_img);
magnitude2 = abs(dft_shifted);
figure;
subplot(1,2,1);
imshow(log(1 + magnitude2), []);
title('Shifted Magnitude');
subplot(1,2,2);
imshow(first_slice);
title('Original Image');
```



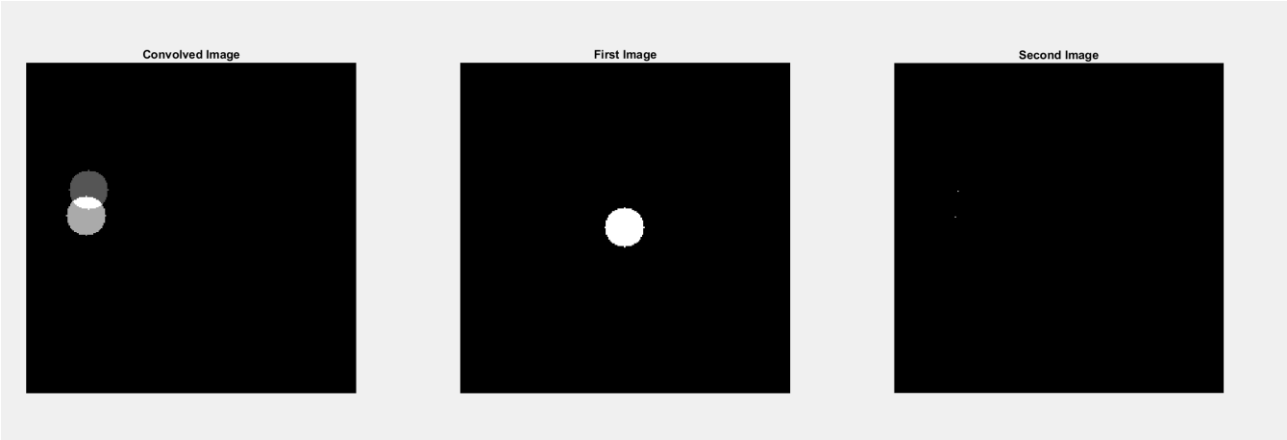




The first image displays the **magnitude spectrum** of the Discrete Fourier Transform (DFT) of a 2D MRI image slice on the left, with the original image on the right. The magnitude spectrum is shown on a logarithmic scale to enhance visibility of the frequency components, and it highlights the strong low-frequency content concentrated near the center.

The second image shows the **shifted magnitude spectrum** after applying fftshift, which centers the low-frequency components for better visualization. The center of the magnitude spectrum now corresponds to the zero frequency, providing a clear representation of the spatial frequency components' distribution. The original image is displayed on the right in both cases for comparison. These steps are commonly used for image analysis and enhancement in the frequency domain.

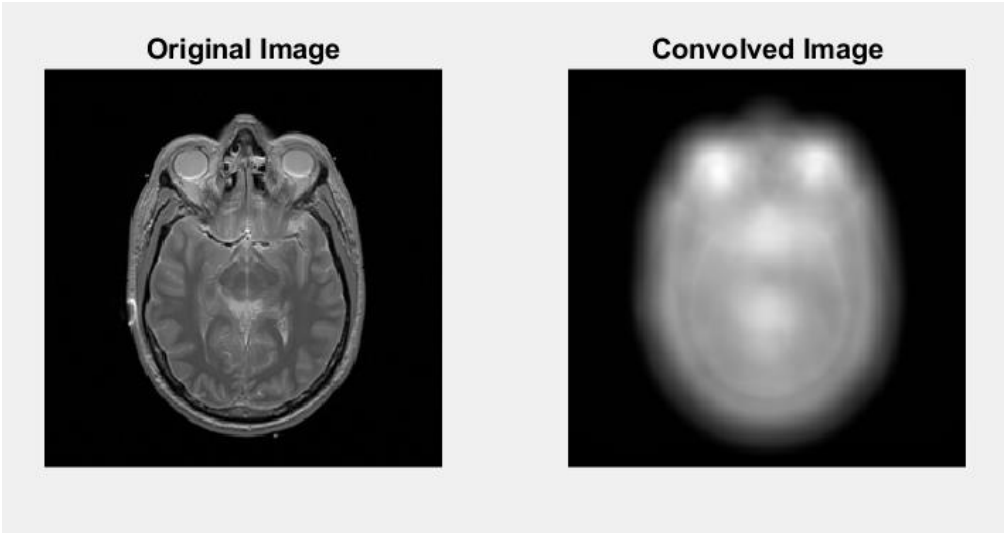
**Question 2)**

```
%% Q2
rows = 256;
cols = 256;
centerX = rows / 2;
centerY = cols / 2;
radius = 15;
[x, y] = ndgrid(1:rows, 1:cols);
G = ((x - centerX).^2 + (y - centerY).^2) <= radius^2;
G = double(G);
F = zeros(256,256);
F(100,50)=1;
F(120,48)=2;
G_fft = fft2(G);
F_fft = fft2(F);
P = G_fft .* F_fft;
P_inverse = ifft2(P);
P_inverse = P_inverse/max(max(P_inverse));
P_inverse = ifftshift(P_inverse);
figure;
subplot(1,3,1);
imshow(P_inverse);
title("Convolved Image");
subplot(1,3,2);
imshow(G);
title("First Image");
subplot(1,3,3);
imshow(F);
title("Second Image");
```




In this question, we performed a 2D convolution in the frequency domain by first creating a circular binary mask G and a sparse matrix F with two high-intensity points. We computed the Fourier transforms of F and F, multiplied their frequency representations, and applied an inverse Fourier transform to obtain the convolved result. The left plot shows the convolved image, where the circular mask spreads the intensity of the points in F, creating overlapping circular patterns. The middle plot represents F, a binary circular mask, while the right plot shows F, containing the two non-zero points.

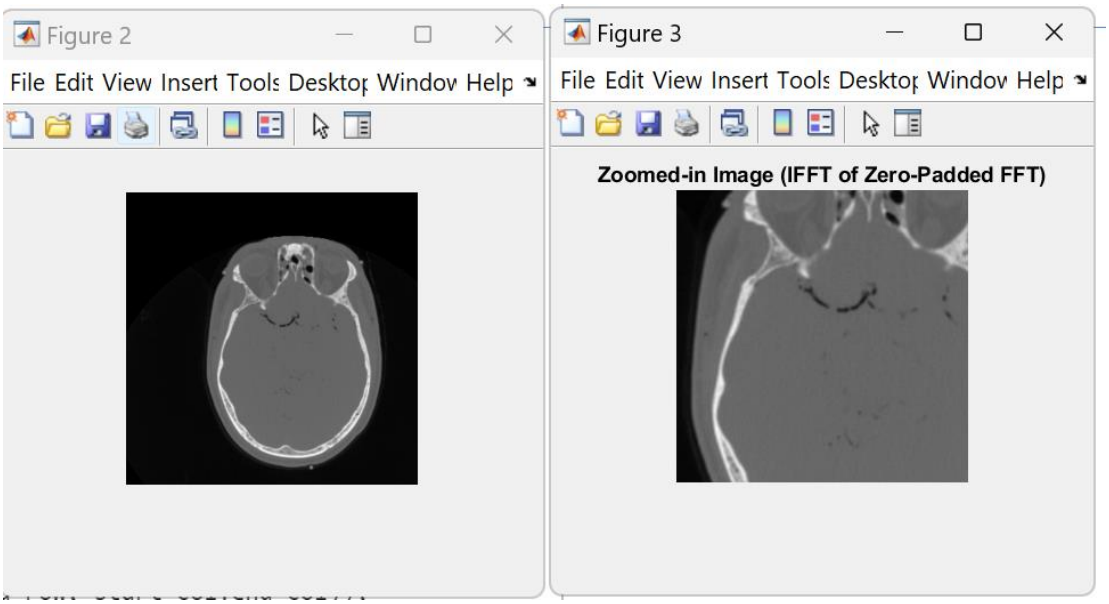
```
%% pd.jpg
my_pd = imread("pd.jpg");
first_slice_pd = my_pd(:,:,1);
first_slice_pd = double(first_slice_pd);
first_slice_pd_fft = fft2(first_slice_pd);
pd_conv = first_slice_pd_fft .* G_fft;
pd_conv_inverse = ifft2(pd_conv);
pd_conv_inverse = pd_conv_inverse/max(max(pd_conv_inverse));
pd_conv_inverse = ifftshift(pd_conv_inverse);
figure;
subplot(1,2,1);
imshow(my_pd);
title("Original Image");
subplot(1,2,2);
imshow(pd_conv_inverse);
title("Convolved Image");
```




In this example, we convolved an MRI image with a predefined kernel in the frequency domain. First, we loaded the original image my_pd and extracted its first slice. We computed the Fourier transform of this slice and multiplied it element-wise with the Fourier transform of the kernel G_fft. Afterward, we applied the inverse Fourier transform, normalized the result, and shifted it back to produce the spatial domain representation of the convolved image. The left plot shows the **Original Image**, and the right plot displays the **Convolved Image**, which appears blurred due to the convolution process that smoothens and spreads the intensity based on the kernel.

**Question 3)**

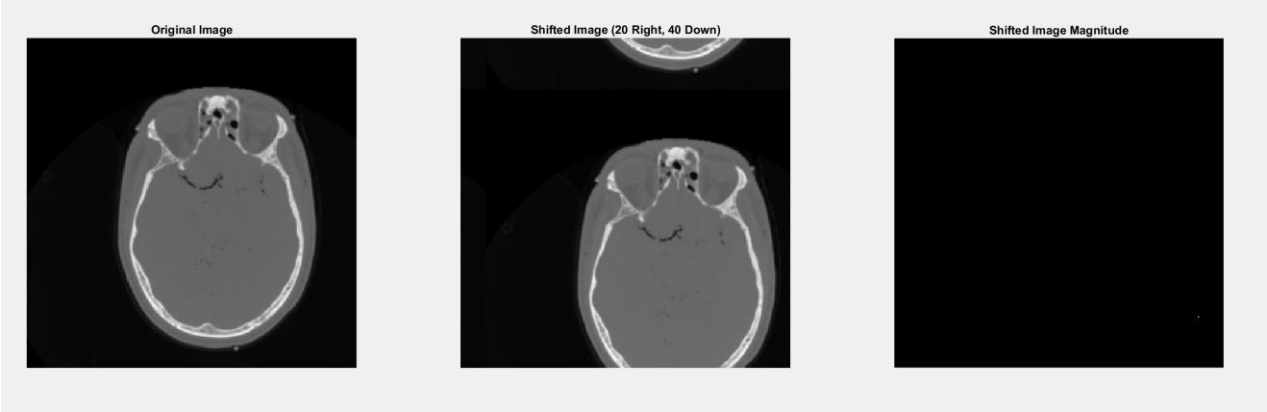
```
95  %% Q3
96  ct = imread("ct.jpg");
97  figure;
98  imshow(ct);
99  ct_first_slice = ct(:,:,1);
100 ct_fft = fft2(ct_first_slice);
101 ct_fft = fftshift(ct_fft);
102 zoomed_ct = zeros(512,512);
103 start_row = rows / 2 + 1;
104 end_row = 3 * rows / 2;
105 start_col = cols / 2 + 1;
106 end_col = 3 * cols / 2;
107 zoomed_ct(start_row:end_row, start_col:end_col) = ct_fft;
108 zoomed_ct = ifftshift(zoomed_ct);
109 ifft_zoomed_ct = ifft2(zoomed_ct);
110 ifft_zoomed_ct = ifft_zoomed_ct / max(max(ifft_zoomed_ct));
111 figure;
112 imshow(ifft_zoomed_ct(start_row:end_row, start_col:end_col));
113 title('Zoomed-in Image (IFFT of Zero-Padded FFT)');
```




In this example, we performed a zoom-in operation on a CT image using the Fourier domain. First, the 2D Fourier transform of the image slice was computed and shifted for better centering of low frequencies. Then, zero-padding was applied in the frequency domain to enhance spatial resolution in the inverse transform, effectively focusing on the center portion of the image. After applying the inverse Fourier transform and normalizing the result, we displayed the zoomed-in region of the CT image. The left plot shows the **original CT image**, while the right plot demonstrates the **zoomed-in image** generated by manipulating the frequency domain data, highlighting more detail in the selected region.

**Question 4.1)**

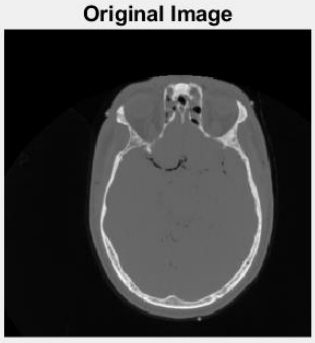
```
%% Q4.1
ct = imread("ct.jpg");
ct_first_slice = ct(:,:,1);
[rows, cols] = size(ct_first_slice);
ct_fft = fft2(ct_first_slice);
shift_x = 20; % Shift to the right
shift_y = 40; % Shift down
[x, y] = meshgrid(1:cols, 1:rows);
shift_kernel = exp(-2*pi*1i * ((x - 1) * shift_x / cols + (y - 1) * shift_y / rows));
shifted_fft = ct_fft .* shift_kernel;
shifted_image = ifft2(shifted_fft);
shifted_image = real(shifted_image);
shifted_image = mat2gray(shifted_image);
figure;
subplot(1,3,1);
imshow(ct_first_slice);
title('Original Image');
subplot(1,3,2);
imshow(shifted_image);
title('Shifted Image (20 Right, 40 Down)');
subplot(1,3,3);
shift_kernel_fft = fft2(shift_kernel);
subplot(1,3,3);
imshow(log(1 + abs(shift_kernel_fft)), []);
title('Shifted Image Magnitude');
```

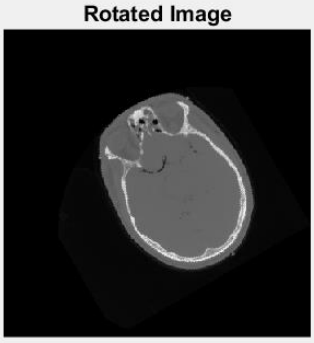


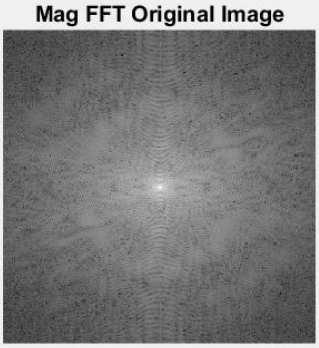

This code demonstrates how to shift an image in the spatial domain using its Fourier Transform. First, the Fourier Transform of the original CT image is computed, and a shifting kernel is created in the frequency domain. The kernel applies a phase shift corresponding to a 20-pixel shift to the right and a 40-pixel shift downward. The Fourier-transformed image is multiplied by the shifting kernel, and the inverse Fourier Transform is applied to generate the shifted image in the spatial domain. The first subplot displays the original image, the second subplot shows the spatially shifted image, and the third subplot visualizes the magnitude spectrum of the shifting kernel in the frequency domain, using a logarithmic scale for better visibility of the frequency components. This demonstrates how phase manipulation in the frequency domain translates to spatial shifts in the image.

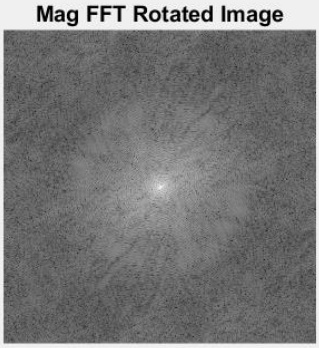
**Question 4.2)**

```
%% Q4.2
ct = imread("ct.jpg");
ct_first_slice = ct(:,:,1);
ct_rotate = imrotate(ct_first_slice,30);
figure;
subplot(1,2,1);
imshow(ct);
title('Original Image');
subplot(1,2,2);
imshow(ct_rotate);
title('Rotated Image');
figure;
subplot(1,2,1);
ct_fft = fft2(ct_first_slice);
ct_fft = fftshift(ct_fft);
magnitude = abs(ct_fft);
magnitude_log = log(1 + magnitude);
magnitude_log = magnitude_log / max(magnitude_log(:));
imshow(magnitude_log, []);
title('Mag FFT Original Image');
subplot(1,2,2);
ct_rotate_fft = fft2(ct_rotate);
ct_rotate_fft = fftshift(ct_rotate_fft);
magnitude = abs(ct_rotate_fft);
magnitude_log = log(1 + magnitude);
magnitude_log = magnitude_log / max(magnitude_log(:));
imshow(magnitude_log, []);
title('Mag FFT Rotated Image');
```



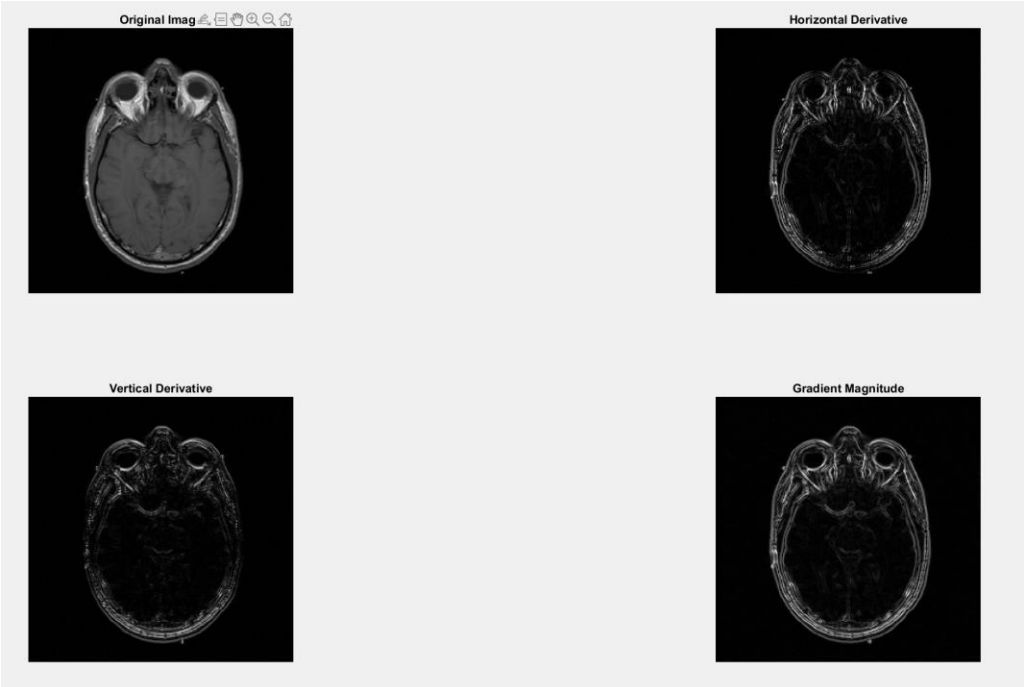




In this question, we analyzed the Fourier Transform of an original and rotated CT image. First, we loaded the CT image and extracted its first slice. We then rotated the image by 30 degrees and displayed both the original and rotated images side by side. Next, we computed the Fourier Transform of both the original and rotated images, shifted the zero frequency component to the center, and calculated the magnitude spectrum. We visualized the log-transformed magnitude spectra for better detail and normalized them for comparison. The results showed that while the spatial domain rotation affects the image's orientation, the magnitude spectrum in the frequency domain captures rotationally invariant features, as evidenced by the similar patterns in the magnitude spectra of the original and rotated images. This demonstrates how the Fourier domain is robust to certain spatial transformations.

**Question 5)**

```
%% Q5
t1_img = imread("t1.jpg");
t1_img_first_slice = double(t1_img(:,:,1));
y = circshift(t1_img_first_slice, 1, 2);
y1 = circshift(t1_img_first_slice, -1, 2);
horizontal_derivative = abs(y - y1);
z = circshift(t1_img_first_slice, 1, 1);
z1 = circshift(t1_img_first_slice, -1, 1);
vertical_derivative = abs(z - z1);
gradient_magnitude = sqrt(horizontal_derivative.^2 + vertical_derivative.^2);
figure;
subplot(2,2,1);
imshow(t1_img_first_slice, []);
title('Original Image');
subplot(2,2,2);
imshow(horizontal_derivative, []);
title('Horizontal Derivative');
subplot(2,2,3);
imshow(vertical_derivative, []);
title('Vertical Derivative');
subplot(2,2,4);
imshow(gradient_magnitude, []);
title('Gradient Magnitude');

```

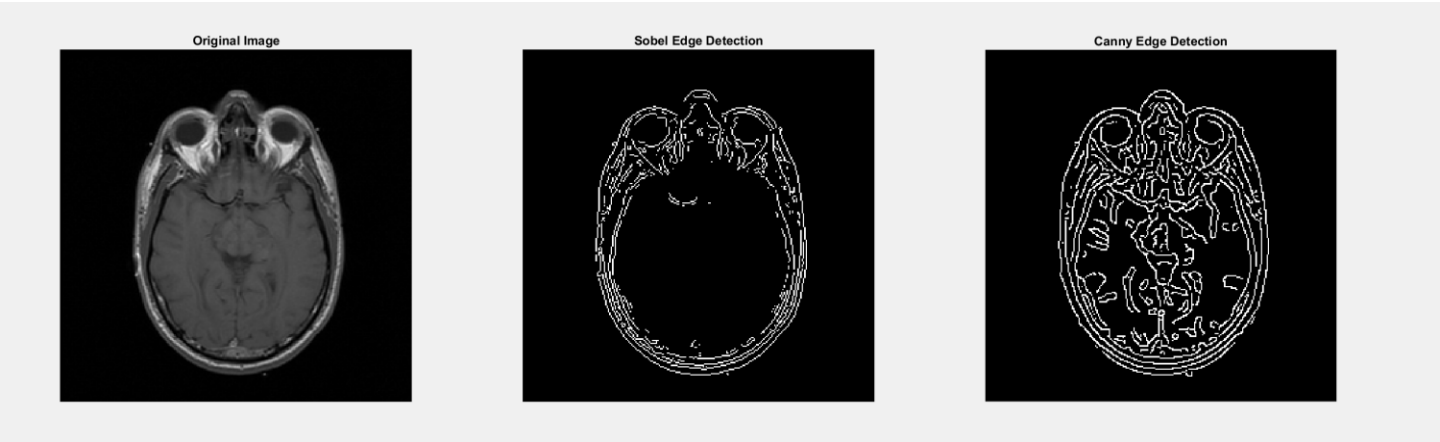


In this question, we performed edge detection on the first slice of an image using central difference approximations to compute the horizontal and vertical derivatives. By applying the circshift function, we shifted the image by one pixel to the left and right (for horizontal derivatives) and up and down (for vertical derivatives) and calculated the absolute differences. The gradient magnitude was then computed as the square root of the sum of squared horizontal and vertical derivatives, providing a measure of overall edge strength. The resulting visualizations include the original image (top-left), the horizontal derivative highlighting horizontal edges (top-right), the vertical derivative emphasizing vertical edges (bottom-left), and the gradient magnitude combining both directions to show the overall edges (bottom-right). This approach allows us to identify and analyze the edges and transitions in the image effectively.

**Question 6)**

- Sobel is a gradient-based method that calculates derivatives in horizontal and vertical directions using convolution filters. It highlights edges with a high-intensity gradient, particularly suited for detecting general edges.
- Canny is a more advanced edge-detection technique that uses Gaussian smoothing to reduce noise, computes the gradient magnitude, applies non-maximum suppression to thin edges, and finally uses double-thresholding to identify strong and weak edges. This method is more robust and detects finer details compared to Sobel.

```
197  %% Q6: Sobel & Canny
198  t1_img = imread("t1.jpg");
199  t1_img_first_slice = double(t1_img(:,:,1));
200  sobel_edges = edge(t1_img_first_slice, 'Sobel');
201  canny_edges = edge(t1_img_first_slice, 'Canny');
202  figure;
203  subplot(1,3,1);
204  imshow(t1_img_first_slice, []);
205  title('Original Image');
206  subplot(1,3,2);
207  imshow(sobel_edges, []);
208  title('Sobel Edge Detection');
209  subplot(1,3,3);
210  imshow(canny_edges, []);
211  title('Canny Edge Detection');
212
```



1. **Sobel vs. Central Differences:**
    - Sobel uses fixed convolutional kernels to compute gradients in the horizontal and vertical directions. This method inherently applies smoothing, reducing sensitivity to noise compared to central differences.
    - Sobel edges are less sensitive to fine details but emphasize strong, prominent edges, making them suitable for detecting structural boundaries in the image.
2. **Canny vs. Central Differences:**
    - Canny is significantly more advanced, combining Gaussian smoothing for noise reduction, edge thinning using non-maximum suppression, and double-thresholding for edge linking. This results in detailed and precise edges.
    - Compared to central differences, the Canny method detects finer details and is more robust to noise. It identifies weaker edges that might be missed by the central difference method.
3. **Gradient Magnitude (Central Differences) vs. Sobel and Canny:**
    - The gradient magnitude computed with central differences captures edges in all directions but can be noisy and lacks thinning, resulting in thicker edges.
    - Sobel produces cleaner edges, while Canny is the most robust, capturing fine and weak edges with better localization.

**The Sobel method is an improvement over central differences for edge detection, offering smoother and more prominent edges. The Canny method, however, outperforms both by providing finer details and robustness to noise, making it ideal for applications requiring high precision.**