**IVP PROJECT REPORT**

**JHANVI BHALODIA**

**E281**

**70552200006**

**Radiological Image Data Compression and Diagonstic Enhancement Framework (RIDC-DEF)**

**Problem Definition:**

The existing challenges in radiological imaging encompass two key aspects:

data compression and diagnostic image enhancement. While data compression aims to reduce the storage and transmission requirements of radiological images, diagnostic image enhancement aims to improve the quality and interpretability of these images. To address these challenges, I propose the development of the Radiological Image Data Compression and Diagnostic Enhancement Framework (RIDC-DEF).

**Proposed Solution:**

**PART 1 : IMAGE COMPRESSION**

* Discrete Cosine Transform (DCT): DCT is a mathematical technique used for signal and image compression. It transforms an image from the spatial domain to the frequency domain, making it possible to represent the image using fewer coefficients while preserving its essential information.
* Fast Fourier Transform (FFT): FFT is another mathematical tool used for transforming an image into the frequency domain. It's commonly used in image processing to analyze and manipulate images in the frequency space. In this context, it may be used alongside DCT for comparison purposes or as part of the compression process.
* Image Compression: The primary concept here is image compression, which involves reducing the amount of data required to represent an image. Lossy and lossless compression techniques, such as DCT and FFT, are fundamental to this concept.
* Compression Levels: The code explores the effects of different compression levels on image quality. Adjusting compression levels involves varying the amount of information retained, highlighting the trade-off between image quality and compression ratio.

**PART 2 :** **IMAGE RECONSTRUCTION**

* Sinogram: A sinogram is a representation of data obtained from a CT scanner. It's created by projecting X-ray measurements taken at various angles through the patient's body. Understanding sinograms is essential for CT image reconstruction.
* Backpropagation: In this context, backpropagation refers to the process of taking the sinogram data and using it to reconstruct an image in the spatial domain. It's a fundamental concept in CT image reconstruction.
* Central Slice Theorem: This theorem is a mathematical principle that explains how to reconstruct an image from its Fourier domain data, which is closely related to CT imaging. It's used to reconstruct CT images based on data collected in the Fourier domain.

**Source Code:  
PART 1:**

%% Read and Resize

img=imread('Brain.jpg');

img=imresize(img,[100,100]);

%% Find DCT

Z(:,:,1)=dct2(img(:,:,1));

Z(:,:,2)=dct2(img(:,:,2));

Z(:,:,3)=dct2(img(:,:,3));

for i=1:100

for j=1:100

if((i+j)>60)

Z(i,j,1)=0;

Z(i,j,2)=0;

Z(i,j,3)=0;

end

end

end

%% find inverse DCT

K(:,:,1)=idct2(Z(:,:,1));

K(:,:,2)=idct2(Z(:,:,2));

K(:,:,3)=idct2(Z(:,:,3));

%% Plot

FigHandle = figure;

set(FigHandle, 'Position', [100, 100, 1000, 400]);

subplot(1,4,1),imshow(uint8(K)),title('70% compression');

%% Next

Z(:,:,1)=dct2(img(:,:,1));

Z(:,:,2)=dct2(img(:,:,2));

Z(:,:,3)=dct2(img(:,:,3));

for i=1:100

for j=1:100

if((i+j)>100)

Z(i,j,1)=0;

Z(i,j,2)=0;

Z(i,j,3)=0;

end

end

end

K(:,:,1)=idct2(Z(:,:,1));

K(:,:,2)=idct2(Z(:,:,2));

K(:,:,3)=idct2(Z(:,:,3));

%% Plot

subplot(1,4,2),imshow(uint8(K)),title('50% compression');

%% Next

Z(:,:,1)=dct2(img(:,:,1));

Z(:,:,2)=dct2(img(:,:,2));

Z(:,:,3)=dct2(img(:,:,3));

for i=1:100

for j=1:100

if((i+j)>140)

Z(i,j,1)=0;

Z(i,j,2)=0;

Z(i,j,3)=0;

end

end

end

K(:,:,1)=idct2(Z(:,:,1));

K(:,:,2)=idct2(Z(:,:,2));

K(:,:,3)=idct2(Z(:,:,3));

% Plot

subplot(1,4,3),imshow(uint8(K)),title('30% compression');

%% Next

Z(:,:,1)=dct2(img(:,:,1));

Z(:,:,2)=dct2(img(:,:,2));

Z(:,:,3)=dct2(img(:,:,3));

for i=1:100

for j=1:100

if((i+j)>180)

Z(i,j,1)=0;

Z(i,j,2)=0;

Z(i,j,3)=0;

end

end

end

K(:,:,1)=idct2(Z(:,:,1));

K(:,:,2)=idct2(Z(:,:,2));

K(:,:,3)=idct2(Z(:,:,3));

subplot(1,4,4);

%% Plot

subplot(1,4,4),imshow(uint8(K)),title('10% compression');

%% Fast Fourier Transform

Z(:,:,1)=fft2(img(:,:,1));

Z(:,:,2)=fft2(img(:,:,2));

Z(:,:,3)=fft2(img(:,:,3));

for i=1:100

for j=1:100

if((i+j)>60)

Z(i,j,1)=0;

Z(i,j,2)=0;

Z(i,j,3)=0;

end

end

end

K(:,:,1)=ifft2(Z(:,:,1));

K(:,:,2)=ifft2(Z(:,:,2));

K(:,:,3)=ifft2(Z(:,:,3));

%% Plot

FigHandle = figure;

set(FigHandle, 'Position', [100, 100, 1000, 400]);

subplot(1,4,1),imshow(uint8(K));

title('70% compression FFT');

%% Next

Z(:,:,1)=fft2(img(:,:,1));

Z(:,:,2)=fft2(img(:,:,2));

Z(:,:,3)=fft2(img(:,:,3));

for i=1:100

for j=1:100

if((i+j)>100)

Z(i,j,1)=0;

Z(i,j,2)=0;

Z(i,j,3)=0;

end

end

end

K(:,:,1)=ifft2(Z(:,:,1));

K(:,:,2)=ifft2(Z(:,:,2));

K(:,:,3)=ifft2(Z(:,:,3));

%% Plot

subplot(1,4,2),imshow(uint8(K)),title('50% compression FFT');

%% Next

Z(:,:,1)=fft2(img(:,:,1));

Z(:,:,2)=fft2(img(:,:,2));

Z(:,:,3)=fft2(img(:,:,3));

for i=1:100

for j=1:99

if((i+j)>140)

Z(i,j,:)=0;

end

end

end

K(:,:,1)=ifft2(Z(:,:,1));

K(:,:,2)=ifft2(Z(:,:,2));

K(:,:,3)=ifft2(Z(:,:,3));

%% Plot

subplot(1,4,3),imshow(uint8(K)),title('30% compression FFT');

%% Next

Z(:,:,1)=fft2(img(:,:,1));

Z(:,:,2)=fft2(img(:,:,2));

Z(:,:,3)=fft2(img(:,:,3));

for i=1:100

for j=1:100

if((i+j)>180)

Z(i,j,1)=0;

Z(i,j,2)=0;

Z(i,j,3)=0;

end

end

end

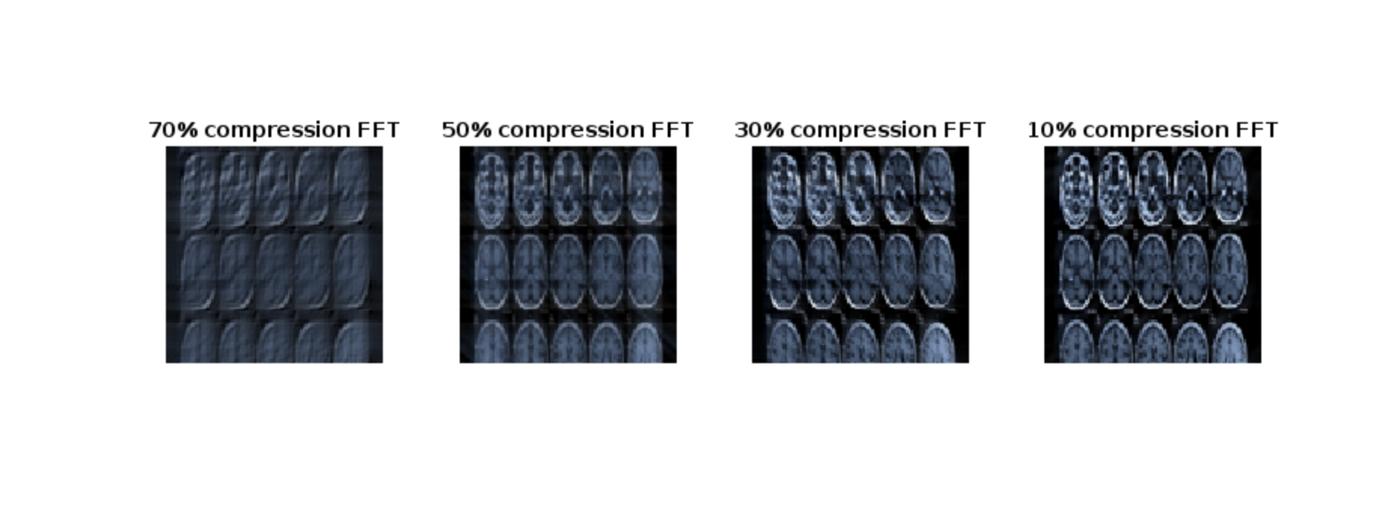
K(:,:,1)=ifft2(Z(:,:,1));

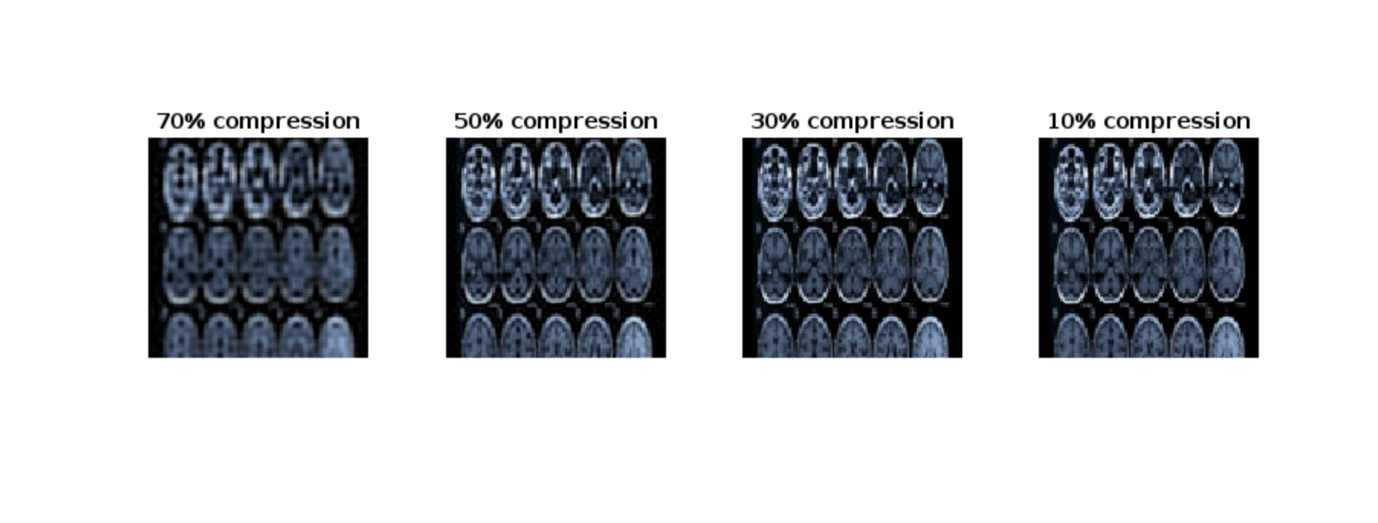
K(:,:,2)=ifft2(Z(:,:,2));

K(:,:,3)=ifft2(Z(:,:,3));

%% Plot

subplot(1,4,4),imshow(uint8(K)),title('10% compression FFT');

**OUTPUT IMAGES: **

****

**PART 2:**

%% The following program computes the sinogram of an image, and

% then accordinly reconstructs the imgage using the methods specified in

% the question.

clear;

%% Load the image and compute sinogram

image=phantom(256);

figure

imshow(image);

thetas=[0:0.5:179.5];

angularPCount = length(thetas);

parallelPCount = size(image,1);

sinogram = zeros(parallelPCount,angularPCount);

% Start the loop

figure

title ('Sinogram');

% make a new figure to store the final output

for i = 1:length(thetas)

% rotate image

tmpImage = imrotate(image,-thetas(i),'bilinear','crop');

% Make the sinogram

sinogram(:,i) = sum(tmpImage,2);

% Display the sinogram

imagesc(sinogram);

drawnow

end

% sinogram matrix now stores the sinogram of the image under specifined

% angles

%% Backpropagation (unfiltered)

% figure out how big our picture is going to be.

pCount = size(sinogram,1);

aCount = length(thetas);

% Angle conversion according to matlab conventions

thetas = (pi/180)\*thetas;

% This is the vector in which the backpropagated is going to be stored

backPImage = zeros(pCount,pCount);

% Middle index

midI = floor(pCount/2) + 1;

[x,y] = meshgrid(ceil(-pCount/2):ceil(pCount/2-1));

% Start the loop. Make a new figure for the backpropagated image display

figure

for i = 1:aCount

rotCoords = round(midI + x\*sin(thetas(i)) + y\*cos(thetas(i)));

indices = find((rotCoords > 0) & (rotCoords <= pCount));

newCoords = rotCoords(indices);

backPImage(indices) = backPImage(indices) + sinogram(newCoords,i)./aCount;

% Draw the backpropagated image

imagesc(backPImage)

drawnow

end

%% Reconstruction using central sliceing theorem

% slice output image stored in this matrix

fourierSliceImage = complex(zeros(pCount,pCount));

% Middle index

midCoord = (pCount+1)/2;

yFBP = ([1:pCount]) - (pCount+1)/2;

% set up filter

rampF = [floor(pCount/2):-1:0 1:ceil(pCount/2-1)];%linspace(-1,1,numOfParallelProjections);

% loop begins. Make new figure again for output

figure

for i = 1:aCount

% figure out which projections to add to which spots

xCR = round(midCoord - yFBP\*cos(thetas(i)));

yCR = round(midCoord - yFBP\*sin(thetas(i)));

% Convert 2D coords to indices. Also transpose

newInd = sub2ind(pCount\*[1 1],xCR,yCR);

% Filter in fourier domain

ffDomain = rampF.\*fftshift( fft( fftshift(sinogram(:,i)') ) );

% Summation in fourier domain

fourierSliceImage(newInd) = fourierSliceImage(newInd) + ffDomain./aCount;

% Draw the image

imagesc(real(fourierSliceImage))

title('Filling in fourier domain')

drawnow

end

% Conversion to to spatial domain

finBPI = real( ifftshift(fft2(fourierSliceImage)) );

% visualization on the fly

imagesc(finBPI)

drawnow

%% Reconstruction using filtered backpropagation

% set new image for filtered back propagation

filteredBP = zeros(pCount,pCount);

% Mid index like found above

midFBP = floor(pCount/2) + 1;

[xFBP,yFBP] = meshgrid(ceil(-pCount/2):ceil(pCount/2-1));

% Filter for spatial doman. One can use either 'sheppLogan' or 'ramLak'

filterMode = 'ramLak';

if mod(pCount,2) == 0

halfFilSz = floor(1 + pCount);

else

halfFilSz = floor(pCount);

end

% Proceed accourind to the input

if strcmp(filterMode,'ramLak')

filter = zeros(1,halfFilSz);

filter(1:2:halfFilSz) = -1./([1:2:halfFilSz].^2 \* pi^2);

filter = [fliplr(filter) 1/4 filter];

elseif strcmp(filterMode,'sheppLogan')

filter = -2./(pi^2 \* (4 \* (-halfFilSz:halfFilSz).^2 - 1) );

end

% Loop

figure

for i = 1:aCount

rotCoords = round(midFBP + xFBP\*sin(thetas(i)) + yFBP\*cos(thetas(i)));

% Check the boundedness

indices = find((rotCoords > 0) & (rotCoords <= pCount));

newCoords = rotCoords(indices);

% Setup new filter

filteredProfile = conv(sinogram(:,i),filter,'same');

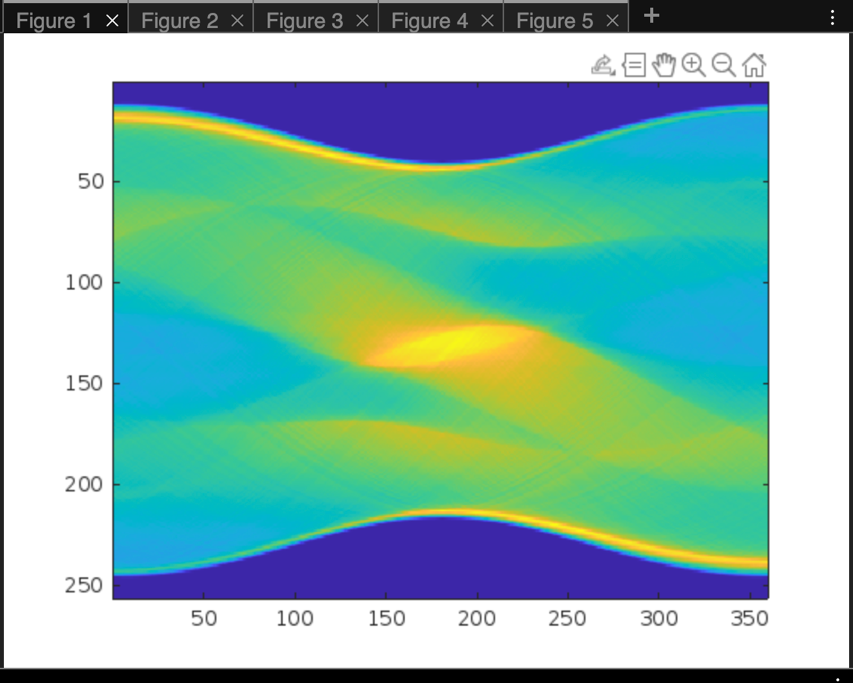
filteredBP(indices) = filteredBP(indices) + filteredProfile(newCoords)./aCount;

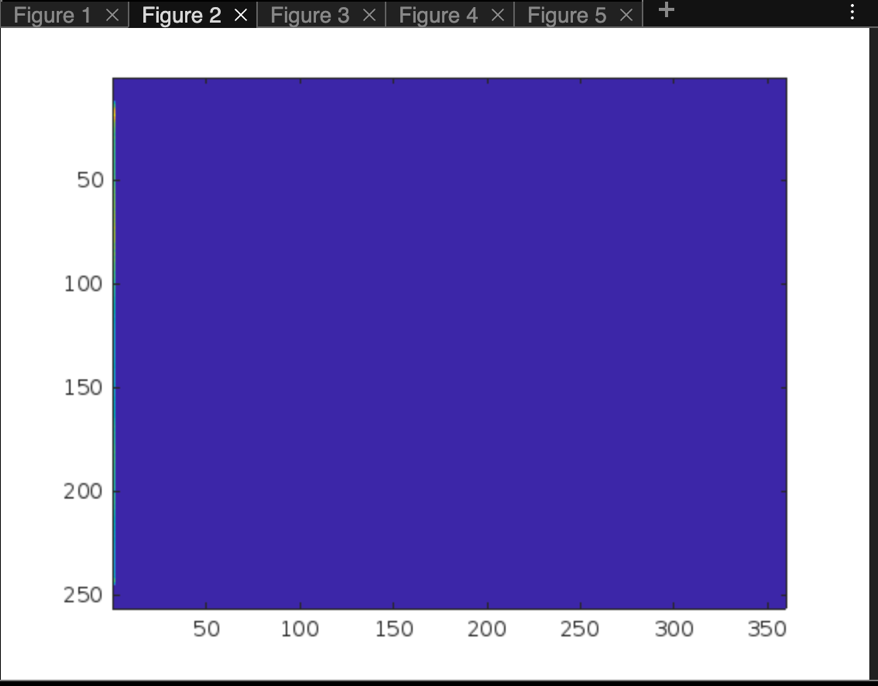
imagesc(filteredBP)

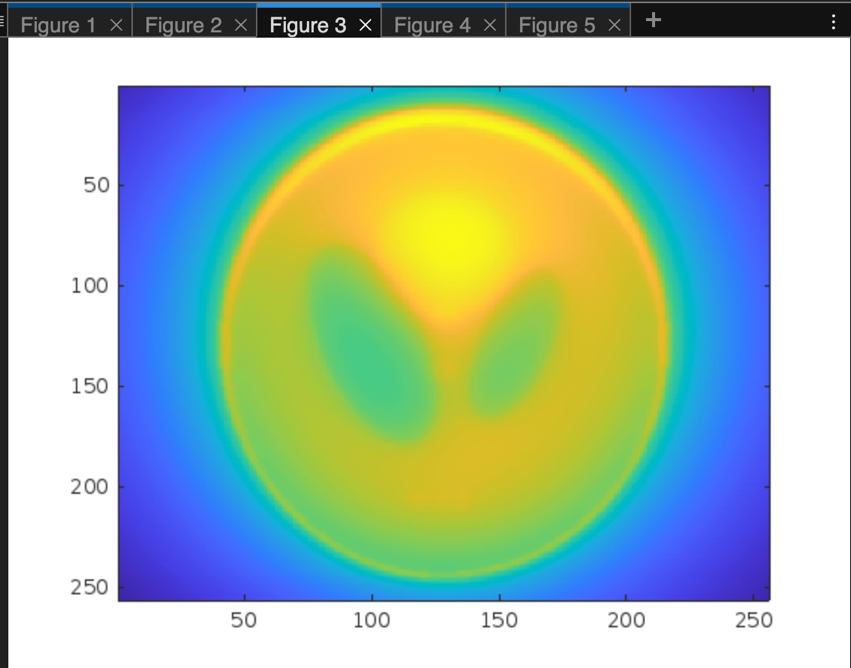
drawnow

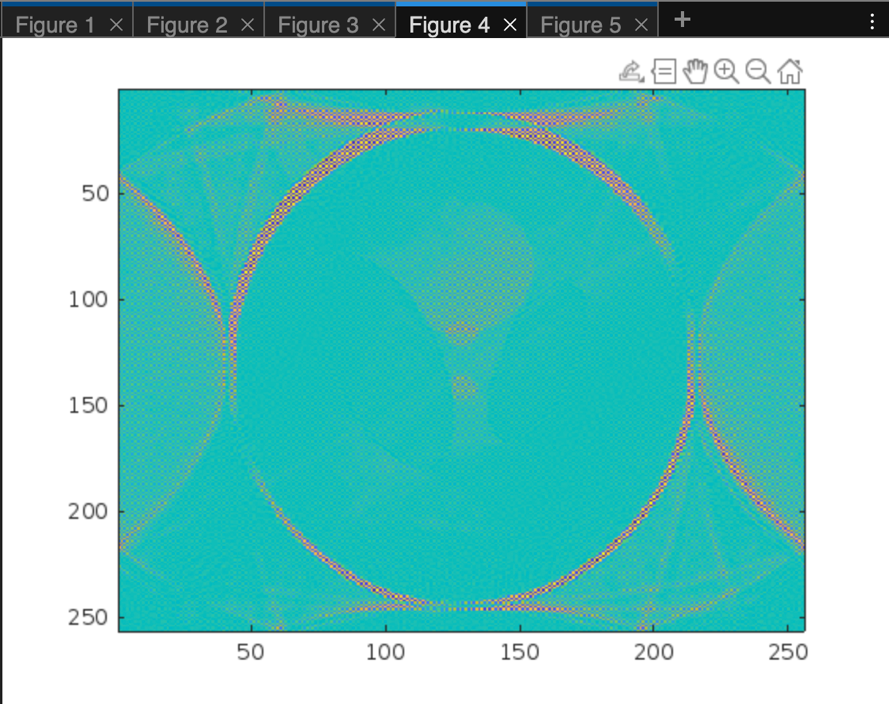
end

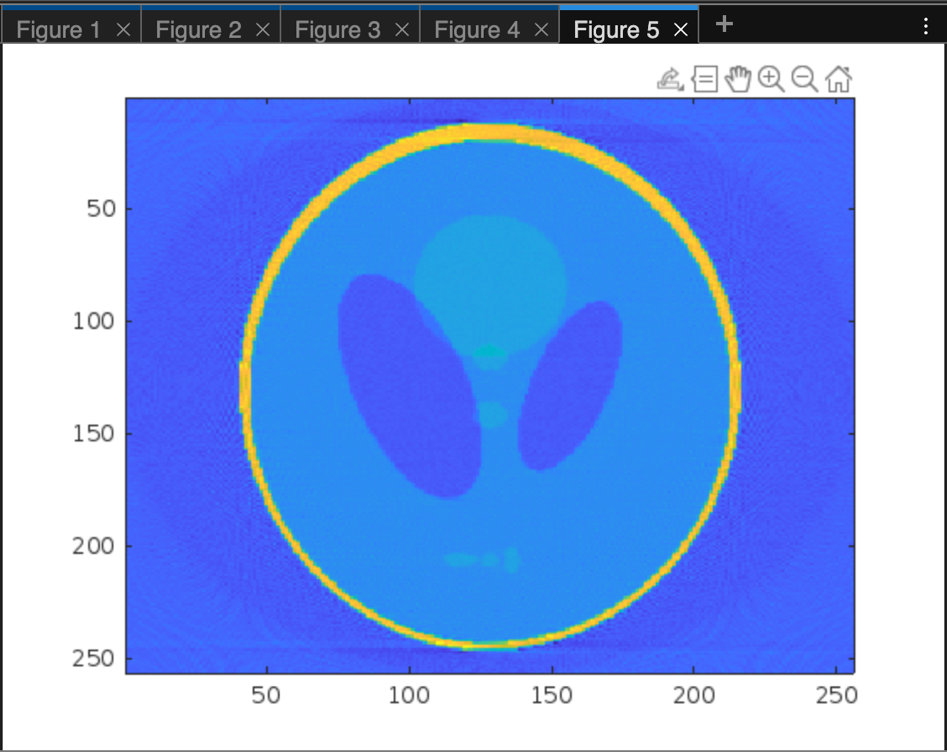
**OUTPUT IMAGES:**

****

****

****

****

****