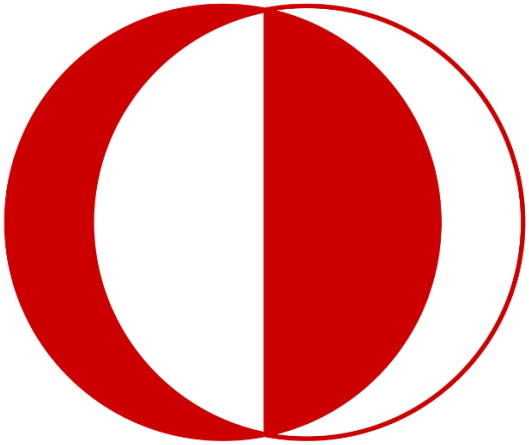


MIDDLE EAST TECHNICAL UNIVERSITY

ELECTRICAL-ELECTRONICS ENGINEERING DEPARTMENT

EE415 TERM PROJECT

X-RAY CT-IMAGING

Student Name: Ali Mertcan Karaman

Student No: 2595031

Submission Date: 20.01.2022

TABLE OF CONTENTS

1. ABSTRACT

In this project, I implemented x-ray tomographic imaging. In medical terms, this is called CT-scan or computed tomography scan. To achieve this goal, I first calculate the projection functions using the Radon Transform. The projection function corresponds to the received signal after the rays passed through an object in tomography. When the projections are obtained, I implemented a backward projection algorithm to construct the object or its image back using inverse Radon Transform and plot the resulting image.

2. INTRODUCTION

2.1. History of Computed Tomography

The history begins with the discovery of x-rays by Wilhelm Konrad Rontgen in 1895 and he is famous with a radiograph, which is an x-ray image of Mrs. Roentgen's hand. In 1896, one year later, Dr. Edwin Frost is credited with making the first diagnostic radiograph.

The history of computed tomography goes back to 1917, the year of the first findings in the theory of Radon Transform. The first clinical CT scan was first performed in 1971 using a scanner invented by Sir, Godfrey Hounsfield.

2.2. Purpose

In this report, the theory of x-ray propagation through a media and the calculation steps of the projection function of the x-rays, which travels through a media or an object, are explained. Obtaining the projection functions is the first step, which we call the forward problem and this part is done using the Radon Transform. Then by using the projection functions, we can reconstruct the image back. This is the second step which we call the inverse problem and it is done using the inverse Radon Transform.

After the theory, I put the results of both the forward and inverse problem as plots using three different images and compared each by changing the projection angles or whether or not a filtering is done on the image.

At the end, I discussed about the results of the reconstructed images and finished with a brief conclusion.

3. THEORY

X-rays attenuates as they propagates through biological tissues and the amount of attenuation is dependent on the photon energy, atomic numbers of the biological tissues and also the electron density. Therefore, x-ray attenuates differently for every tissue and by using the dependence of the attenuation rate on tissue characteristics can be used to reconstruct medical images.

In this project, the forward and backward projection algorithm for an object with known attenuation coefficients is implemented in order to be able to calculate the projection functions and reconstruct the images using these functions.

3.1. Forward Problem

First, we have to determine the projections of x-rays after they propagated through a medium. The simplest way to calculate the projection function is to send a number of x-rays in parallel to a medium and measure the intensity after they passed the medium or an object. However, this works for real applications, here we have to form the projection function ourselves.

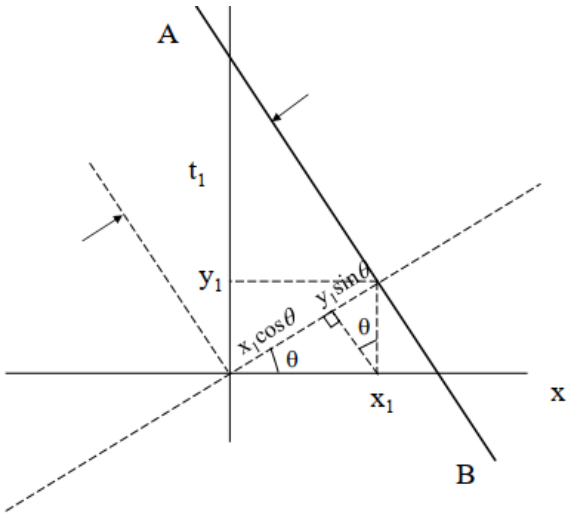
In the algorithm, since we work with objects with known attenuation coefficients, we need to use line integrals to calculate the projection function of the x-ray along that line as in Radon Transform. A line integral is the integral of a physical parameter of the object along a line. For the case of x-ray computed tomography, the line integral is the total attenuation of an x-ray beam when the ray passes through the object along a straight line. If we use a set of x-rays sent in parallel and they pass through the object, the combination of their line integrals can be used to form a projection function. Calculation of projections for known attenuation coefficient distribution and incident beam intensity after propagation of the rays through the object is defined as forward problem of x-ray imaging.

The projection function P(θ,t) is also determined by the angle of view theta (θ). The projection function can be expressed mathematically:

$$\mathrm{P}(\theta, t) = \int_{-L/2}^{L/2} f(x, y) dl$$

Where L is the integration path along the line defined by:

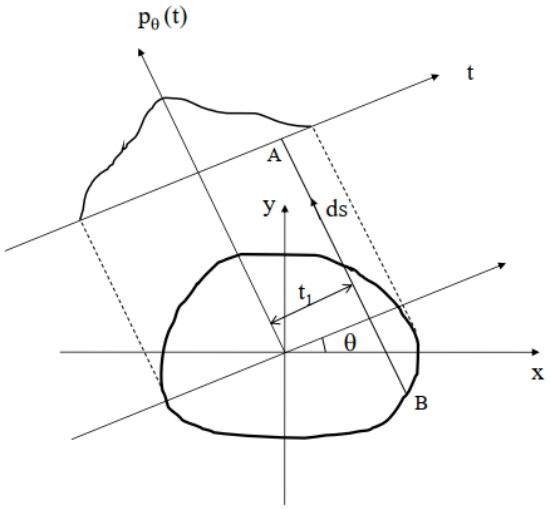
$$x\cos(\theta) + y\sin(\theta) = t$$



The equation of line AB is $x\cos(\theta) + y\sin(\theta) = t$

Then in Cartesian coordinates projection function can be calculated as:

$$\mathrm{P}(\theta, t) = \iint_{-\infty}^{\infty} f(x, y)\delta(x\cos(\theta) + y\sin(\theta) - t)\, dxdy$$

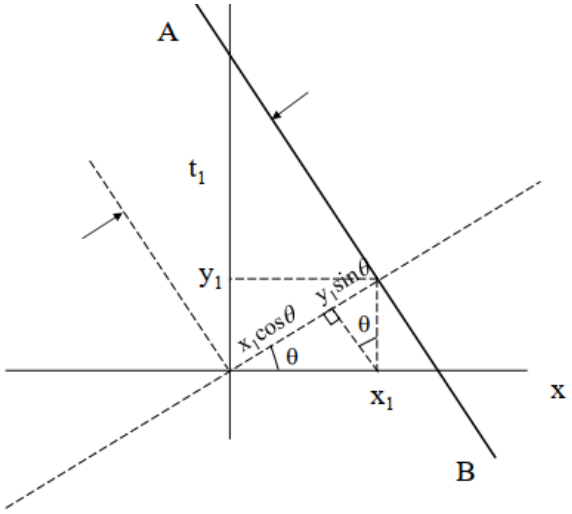


In order to calculate the line integral of any beam at any angle of view; the distance traveled by the corresponding beam within each pixel and the attenuation constant of that pixel should be multiplied. The multiplication for each pixel along the beam where it crosses the object should be added up and stored.

3.2. Inverse Problem

The back projection Pb(θ,t) is also determined by the angle of view theta (θ). The back projection function can be expressed in the same coordinate system where:

$$x\cos(\theta) + y\sin(\theta) = t$$



The equation of line AB is $x\cos(\theta) + y\sin(\theta) = t$

Then the back projection function can be calculated using the projections as given below:

(Radon Transform)

$$\mathrm{Pb(x,y)} = \iint_{0,-\infty}^{\pi,\ \infty} \mathrm{P}(\theta, t)\,\delta(x\cos(\theta) + y\sin(\theta) - t)\,dtd\theta$$

3.2.1. Filtering

The back projected image has a lot of blurry effects. It is the blurred version of the original image. In order to remove the blurring effect filtering is used in frequency domain mostly using a high-pass filter.

The relation between the back projected image and the actual image in Fourier domain is given as:

$$F_b(\rho, \beta) = F(\rho, \beta) \, . \, F\{1/r\}$$

where $F\{1/r\} = 1 / \rho$

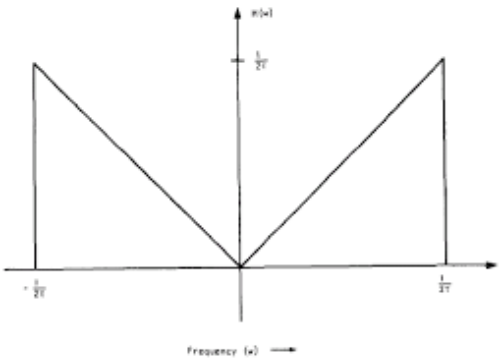
$$F_b(\rho, \beta) = F(\rho, \beta) / \rho$$

Then according to the above relation in frequency domain, we can see that the blurring effect can removed simply by multiplying the back projected image with ρ in frequency domain which is actually a high-pass filter. as given below:

$$\mathbf{P_{bf}(x, y) = F^{-1}\{ \rho * F\{ P_b(x, y)\} \}}$$

where $P_{bf}(x, y)$ is our filtered back projected image.

One example filter for this job is Ram-Lak Filter as shown below:

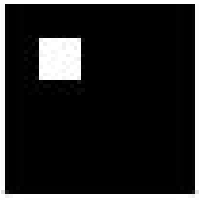


4. RESULTS

Three different images are used to test the algorithms. These images are: the square image, Shepp-Logan image and the lena image. The outputs include projections of the images from different angles and comparisons with the matlab's built-in radon() function ouputs. Also the backprojected images are displayed which are the outputs of the inverse problem and they are compared with the matlab's built-in inverse radon (iradon()) function with and without filtering.

4.1. Square Image

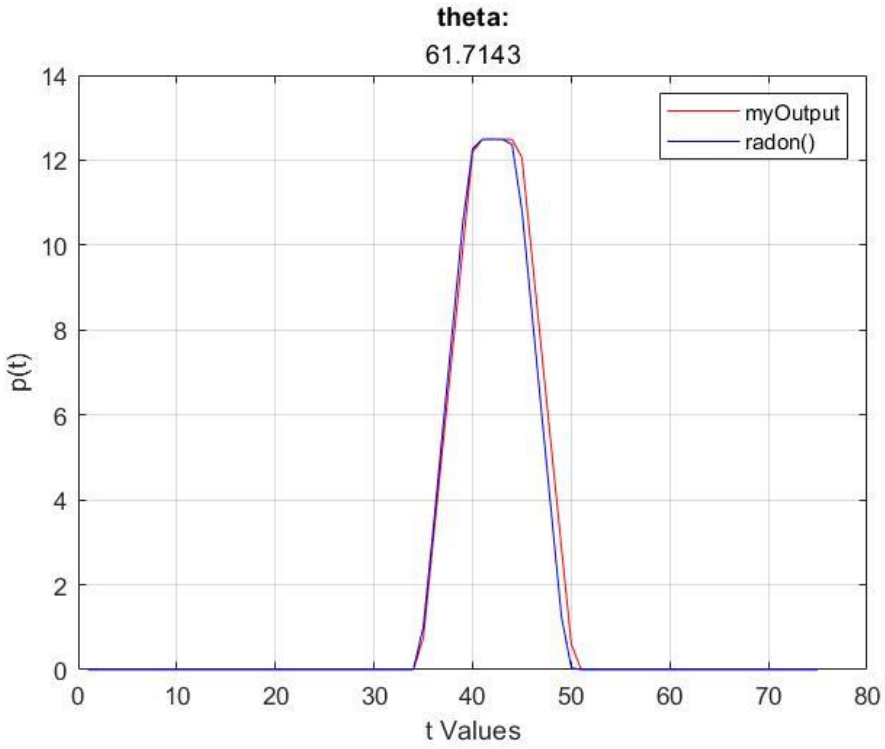
The square image is a 50 by 50 matrix. It can be displayed using the ***imshow()*** function:

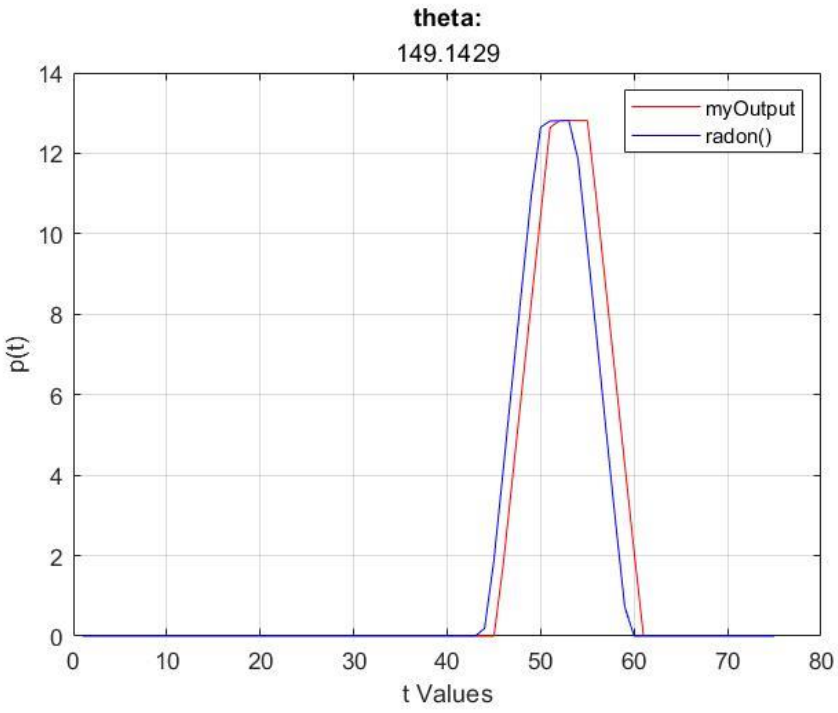


4.1.1. Forward Problem Outputs

As an example, I selected the number of projections as 70 which means the step size is 2.5714 degrees and the number of beams for each projection as 75.

Here is the projection results and their comparisons with Matlab's radon() function for the angles 61.7143 and 149.1429 degrees.

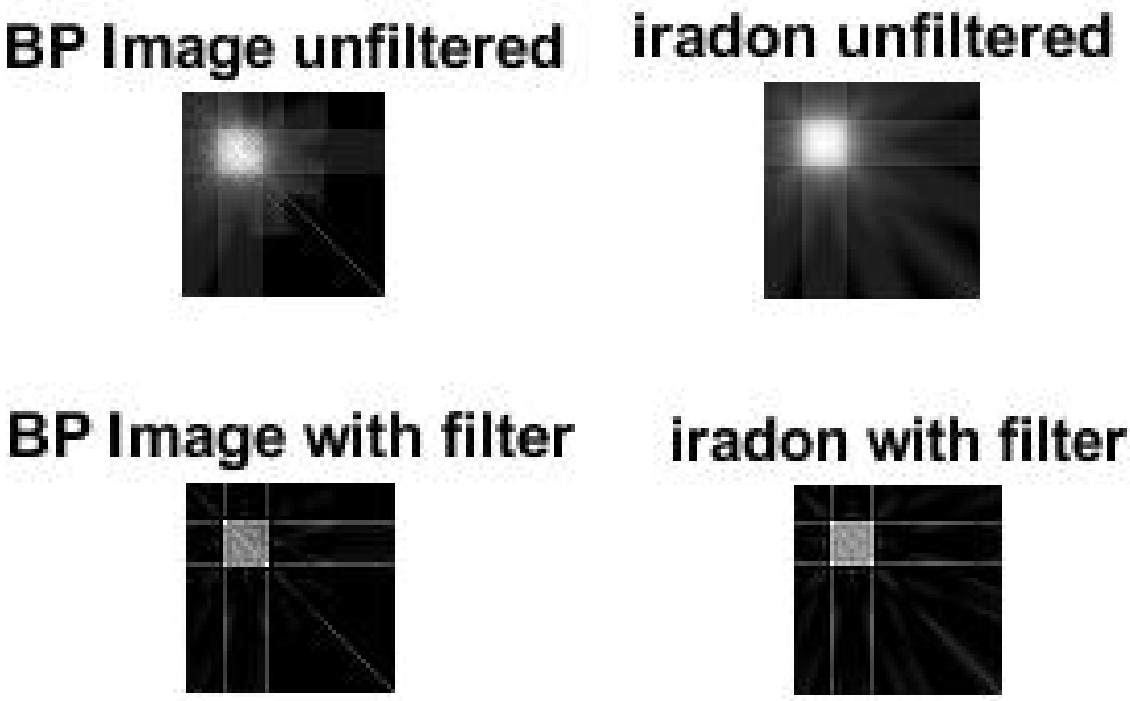




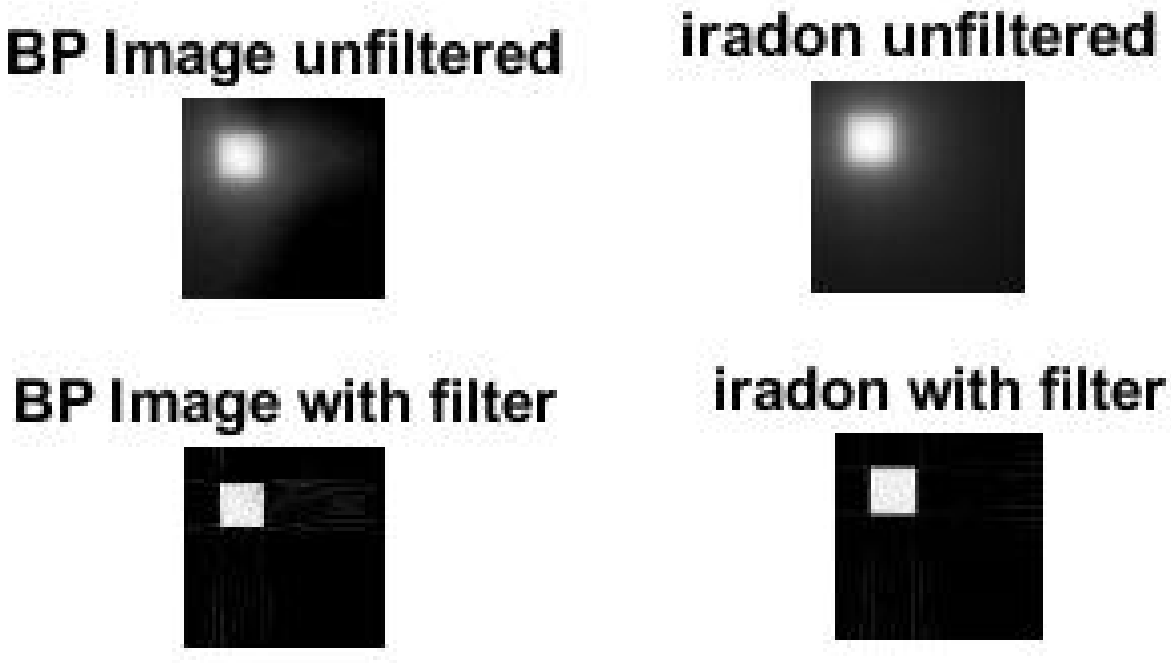
4.1.2. Inverse Problem Outputs

Here is the back projection results with and without filter and their comparisons with Matlab's iradon() function.

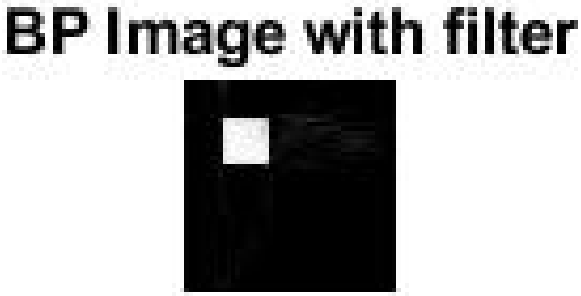
Number of projections is selected as **8** and the number of beams for each projection is **75**.

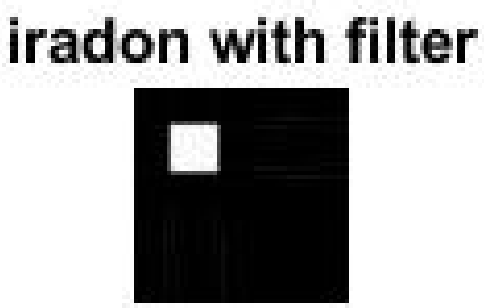


Number of projections is selected as **41** and the number of beams for each projection is **75**.



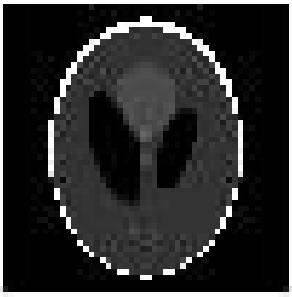
Number of projections is selected as **90** and the number of beams for each projection is **75**.





4.2. Shepp-Logan Image

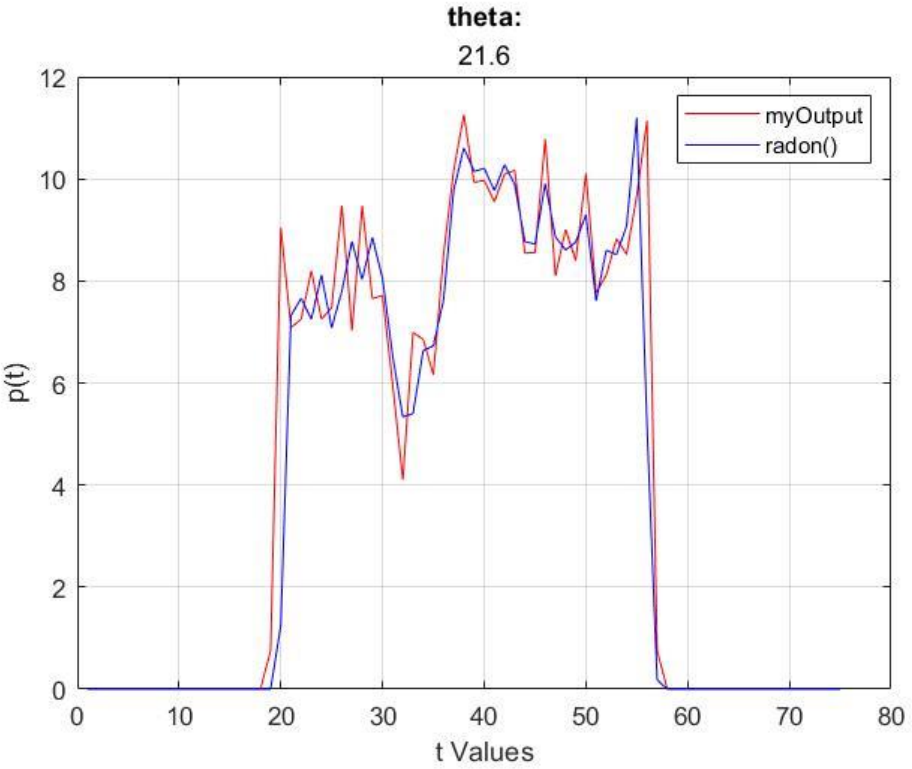
The Shepp-Logan image is a 50 by 50 matrix. It can be displayed using the ***imshow()*** function:

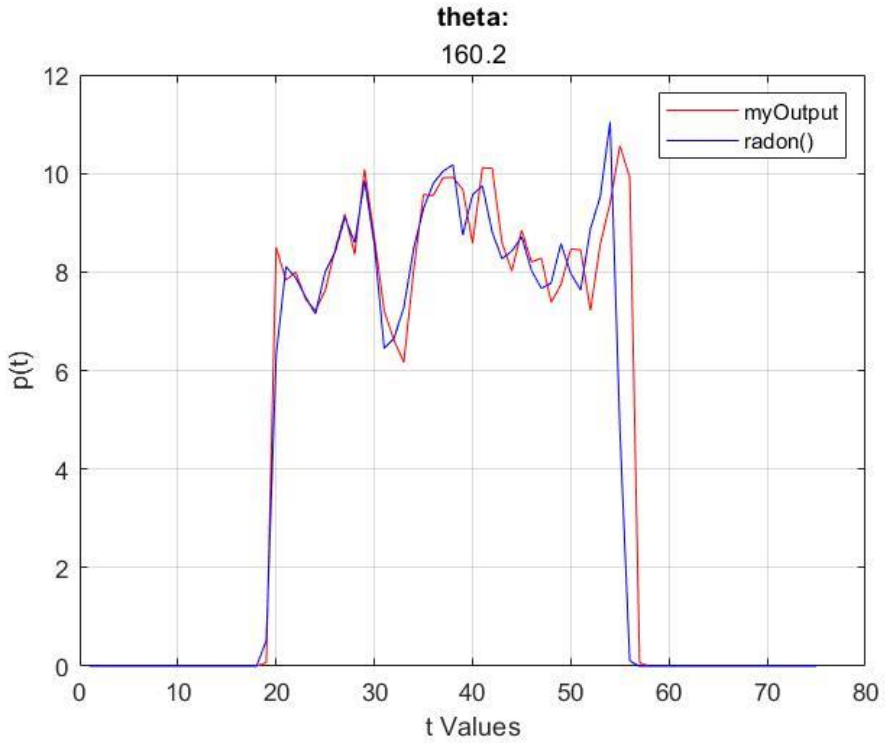


4.2.1. Forward Problem Outputs

I selected the number of projections as 100 which means the step size is 1.8 degrees and the number of beams for each projection as 75.

Here is the projection results and their comparisons with Matlab's radon() function for the angles 21.6 and 160.2 degrees.


theta:
21.6
myOutput
radon()
p(t)
t Values
0
2
4
6
8
10
12
0
10
20
30
40
50
60
70
80

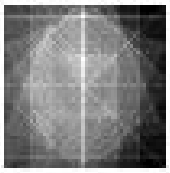

theta:
160.2
myOutput
radon()
p(t)
t Values
0
2
4
6
8
10
12
0
10
20
30
40
50
60
70
80

4.2.2. Inverse Problem Outputs

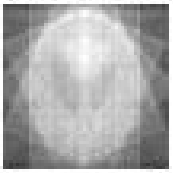
Here is the back projection results with and without filter and their comparisons with Matlab's iradon() function.

Number of projections is selected as **8** and the number of beams for each projection is **75**.

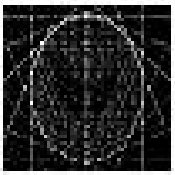
BP Image unfiltered



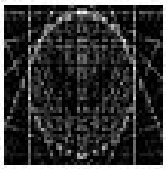
iradon unfiltered



BP Image with filter



iradon with filter



Number of projections is selected as **41** and the number of beams for each projection is **75**.

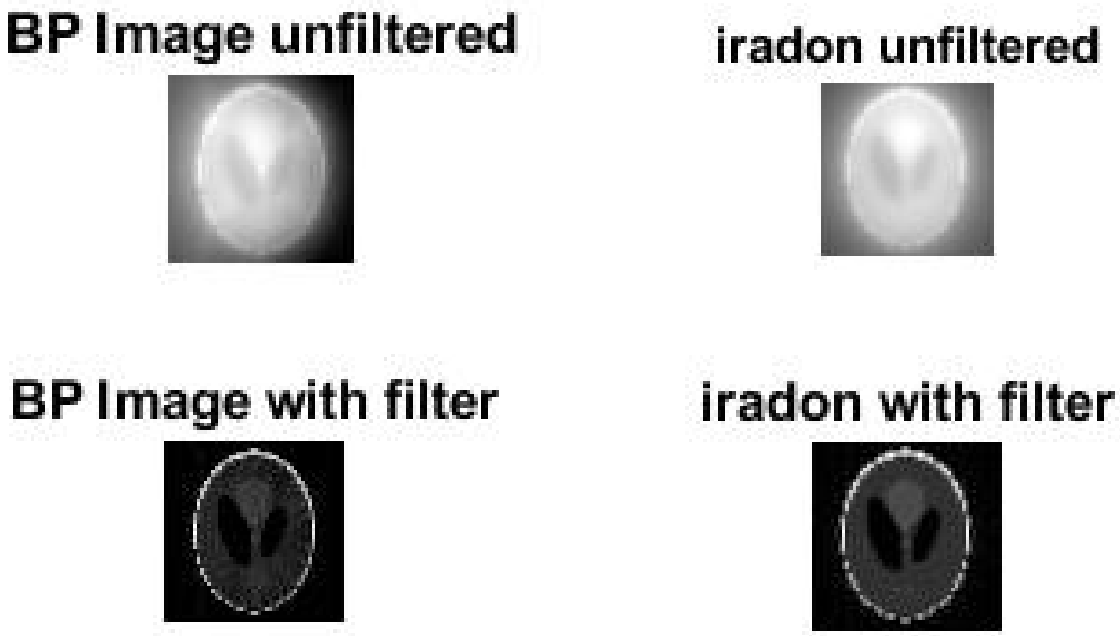
BP Image unfiltered

iradon unfiltered

BP Image with filter

iradon with filter

Number of projections is selected as **180** and the number of beams for each projection is **75**.



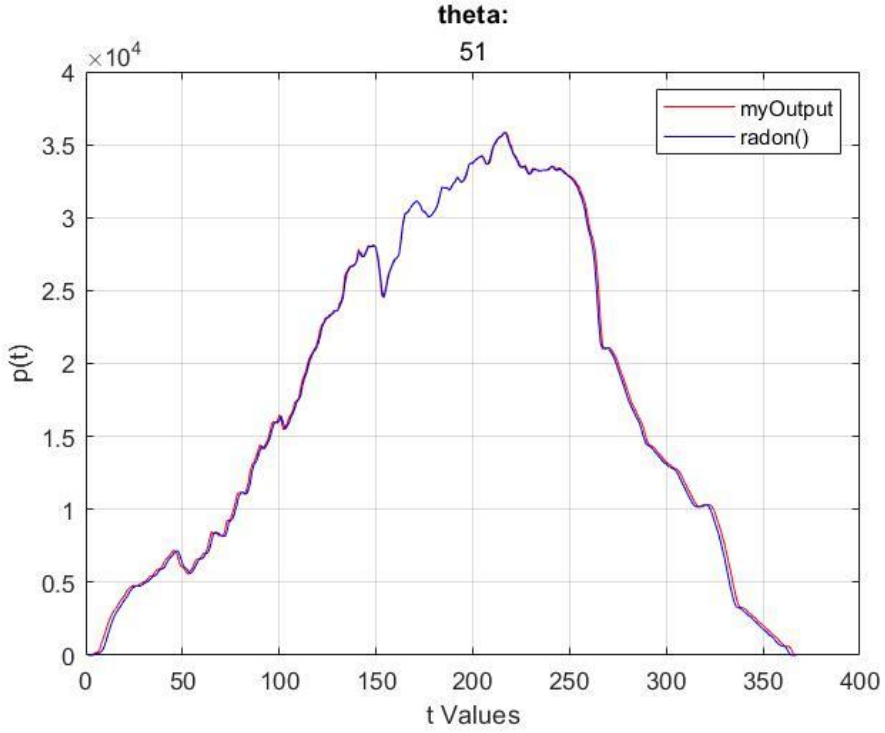
4.3. Lena Image

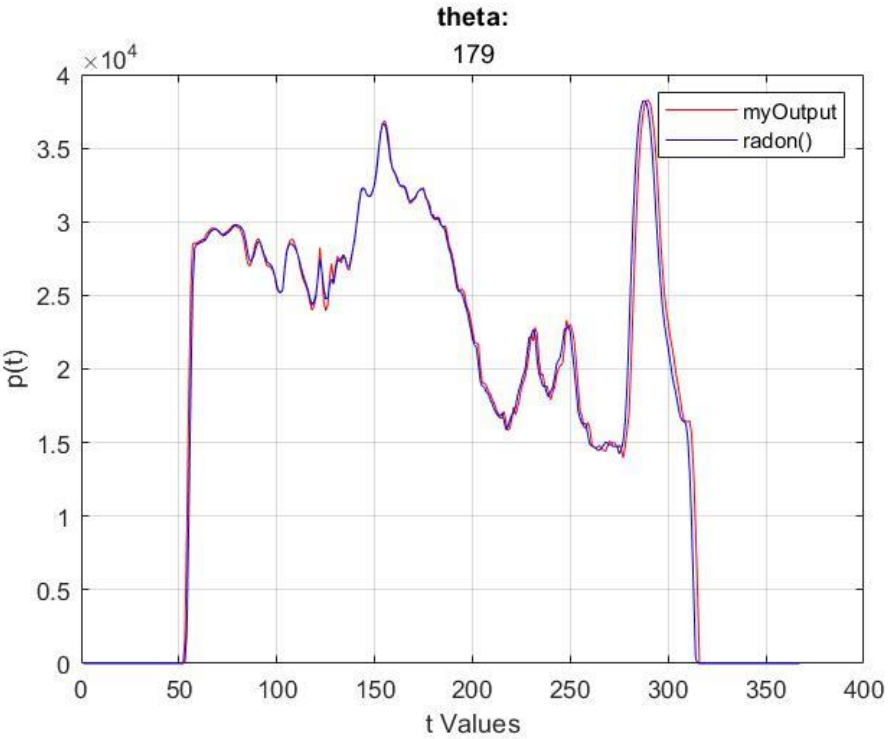
The lena image is a 256 by 256 matrix. It can be displayed using the ***imshow()*** function:

4.3.1. Forward Problem Outputs

I selected the number of projections as 180 which means the step size is 1 degree and the number of beams for each projection as 367.

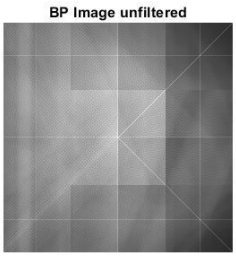
Here is the projection results and their comparisons with Matlab's radon() function for the angles 51 and 179 degrees.

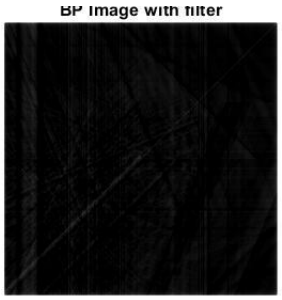


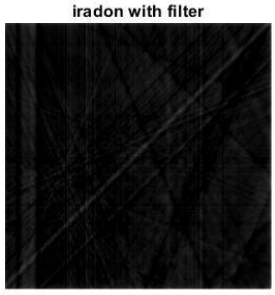


4.3.2. Inverse Problem Outputs

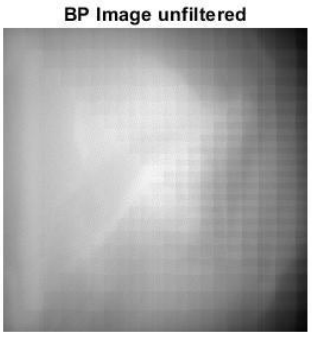
Number of projections is selected as **8** and the number of beams for each projection is **367**.



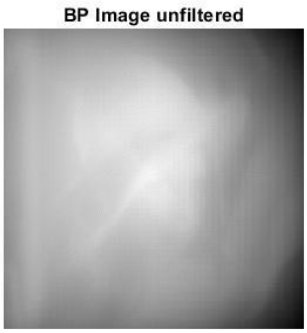


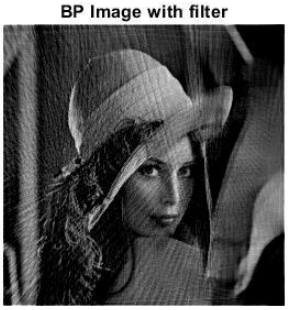


Number of projections is selected as **41** and the number of beams for each projection is **367**.

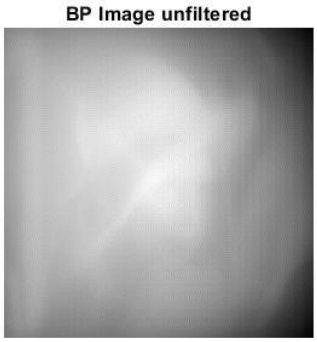


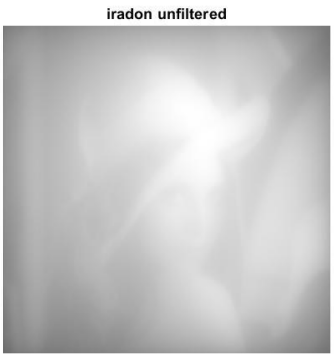
Number of projections is selected as **90** and the number of beams for each projection is **367**.

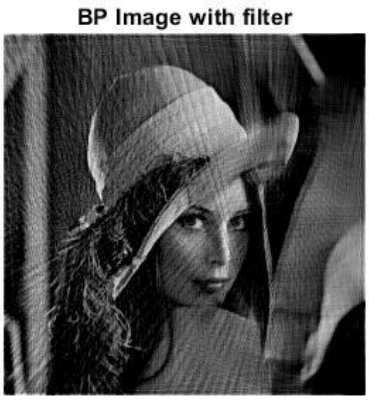




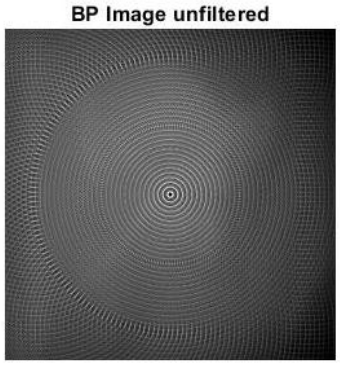
Number of projections is selected as **90** and the number of beams for each projection is **555**.

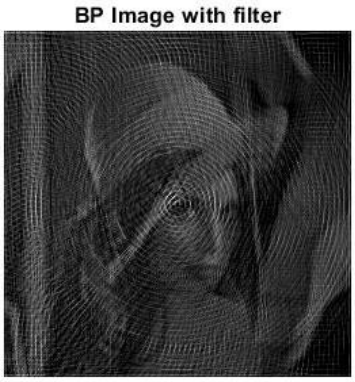






Number of projections is selected as **90** and the number of beams for each projection is **100**.





5. DISCUSSION

I want to start the discussion with my previous small mistake I made in the algorithm of the forward projection part. I was just not pairing up x-points and y-points accurately. Due to that mistake, the projection functions were not truly accurate but I fixed that mistake and in the final algorithm everything works just fine as seen in this report.

In this report, I used three different images. In the forward projection algorithm, I load one of the images by entering their name such as “lena” or “SheppLogan” as they stored in my folder with these names. After that I enter the number of projections and the number of beams used for each projections and then the algorithm runs obtaining the projections. In the backward projection, having the projections from the first part and knowing the distances that the rays traveled from each pixel, I was able to reconstruct the image.

I compared my results with the radon and iradon functions. In the forward problem, the outputs which are actually the projection functions are presented for different angle of views. My outputs and the outputs of the radon() function are almost exactly the same for each image which proves the forward problem is successful.

In the inverse problem, to reconstruct the images, we need the projections and I used the forward projection algorithm with different number of projections; 8, 41, 90 and 180 are used as an example. When the number of projections is small (8), reconstructing the complicated images was not possible but for the square image, since it is a basic image, the output of the algorithm was similar to the original one but for lena and SheppLogan images, their backprojected images do not resemble the original images. When we

increase the number of projections, the quality or the resolution of the reconstructed images becomes higher for every images since we can extract more information from the objects.

The number of beams for each projection also effect the resolution of the reconstructed image. When 100 beams are used for lena image, since its size is 256x256, not every pixel can be sampled. So for the same number of projections (90), as we increase the number of beams, the reconstructed image's resolution becomes higher. However, increasing it too much does not make big differences but it costs time and it needs more calculation power.

As the final discussion, filtering makes a big difference on the back projected image. Without using a high-pass filter, the original image can not be fully available for us even if we use a high number of projections and beam numbers. But when we use filtering, we can see the back projected images very clearly.

At the end, the outputs from my algorithms and the built-in Matlab functions are quite similar. So x-ray computed tomographic imaging is implemented successfully.

6. CONCLUSION

In this project, a complete x-ray computed tomography imaging is implemented with all the necessary steps. I learned how to implement radon transform both forward and backwards. I saw that without filtering, the reconstructed image is very blurry even if we use a lot of projections. So it is extremely important for x-ray CT but the number of projections should be adequate while increasing the number of projections at some point is meaningless since the difference between the reconstructed images is very small. Therefore, with adequate number of projections and a well designed filter, we can obtain x-ray tomographic images.

7. REFERENCES

[1] Sample Algorithm for the Term Project, Damla Alptekin, November 2020

[2] Principles of Medical Imaging, K. Kirk Shung, 1992

[3] https://en.wikipedia.org/wiki/History_of_computed_tomography

8. APPENDIX

8.1. ForwardProjection.m

```
clear; clc;

%% INPUTS

imageName = input('Enter the name of the image: ','s');
img = load(imageName);

[M, N] = size(img.(imageName));

numberOfAngles = str2double(input('Enter the number of projections: ', 's'));
numberOfBeams = str2double(input('Enter the number of sampling points in each projection: ', 's'));

stepSize = (180 - 0) / numberOfAngles;
thetaValues = linspace(0, (180 - stepSize), numberOfAngles);
```

```
tValues = linspace((-M*sqrt(2)/2), (M*sqrt(2)/2), numberOfBeams);

xCoor = floor(-M/2):ceil(M/2);
yCoor = floor(-N/2):ceil(N/2);

pt = zeros(numberOfAngles,numberOfBeams);
distances_teta = zeros(numberOfBeams, (length(xCoor)*2-1), numberOfAngles);
rows_teta = zeros(numberOfBeams, (length(xCoor)*2-1), numberOfAngles);
cols_teta = zeros(numberOfBeams, (length(xCoor)*2-1), numberOfAngles);

%% FORWARD PROJECTION

for teta = 1:numberOfAngles

   distances = zeros(numberOfBeams, (length(xCoor)*2-1));
   rows = zeros(numberOfBeams, (length(xCoor)*2-1));
   cols = zeros(numberOfBeams, (length(yCoor)*2-1));

   if(0 == thetaValues(teta))
      %Find the midpoints and length of the line segments ignoring the irrelevant points
      for t =  1:numberOfBeams
         for i = 1:(length(yCoor)-1)
            %Calculate length and address of the line segment if it intersects the image
            if(M/2 >= abs(tValues(t)))
               distances(t,i) = yCoor(i+1) - yCoor(i);
               rows(t,i) = M/2 - yCoor(i);
               cols(t,i) = M/2 + ceil(tValues(t));
            end
            %Calculation of projection function
            if((0 < rows(t,i)) && (0 < cols(t,i)))
               pt(teta,t) = pt(teta,t) + img.(imageName)(rows(t,i), cols(t,i)) * distances(t,i);
            end
         end
      end
   elseif(90 == thetaValues(teta))
      %Find the midpoints and length of the line segments ignoring the irrelevant points
      for t =  1:numberOfBeams
         for i = 1:(length(xCoor)-1)
            %Calculate length and address of the line segment if it intersects the image
            if(M/2 >= abs(tValues(t)))
               distances(t,i) = xCoor(i+1) - xCoor(i);
               cols(t,i) = M/2 - xCoor(i);
               rows(t,i) = M/2 - floor(tValues(t));
            end
            %Calculation of projection function
            if((0 < rows(t,i)) && (0 < cols(t,i)))
               pt(teta,t) = pt(teta,t) + img.(imageName)(rows(t,i), cols(t,i)) * distances(t,i);
            end
         end
      end
   else
      xValues = zeros(1, length(xCoor));
      yValues = zeros(1, length(yCoor));
      midXPoints = zeros(1, length(xCoor)-1);
      midYPoints = zeros(1, length(yCoor)-1);
      tempx = zeros(1,length(xCoor)*2);
      tempy = zeros(1,length(xCoor)*2);
      xPoints = zeros(1,length(xCoor)*2);
      yPoints = zeros(1,length(xCoor)*2);
      combined = zeros(length(xCoor)*2,2);
      points = zeros(length(xCoor)*2,2);

      for t =  1:numberOfBeams
         for i = 1:(length(xCoor))
            %Find the x and y intersection points
```

```
            xValues(i) = (tValues(t) - yCoor(i) * sind(thetaValues(teta))) / cosd(thetaValues(teta));
            yValues(i) = (tValues(t) - xCoor(i) * cosd(thetaValues(teta))) / sind(thetaValues(teta));
        end

        tempx = [xCoor xValues];
        tempy = [yValues yCoor];
        xPoints = tempx;
        yPoints = tempy;

        %Check if the x and y points are on the image otherwise, remove from the matrix
        xPoints(abs(tempx) > M/2) = [];
        xPoints(abs(tempy) > M/2) = [];
        yPoints(abs(tempx) > M/2) = [];
        yPoints(abs(tempy) > M/2) = [];

        %1st column is x points and 2nd column is y points
        combined = [xPoints' yPoints'];

        %Sort the relevant points
        points = unique(round(sortrows(combined, 1), 5), 'rows');

        if(1 < (size(points,1)))
            for i = 1:(size(points,1)-1)
                %Calculation of distance, row and column data
                distances(t,i) = sqrt( (abs(points(i+1,1)) - abs(points(i,1)))^2 + ...
                                (abs(points(i+1,2)) - abs(points(i,2)))^2 );
                midXPoints(i) = (points(i,1) + points(i+1,1)) / 2;
                midYPoints(i) = (points(i,2) + points(i+1,2)) / 2;
                rows(t,i) = M/2 - floor(midYPoints(i));
                cols(t,i) = M/2 + ceil(midXPoints(i));

                %Calculation of projection function
                if((0 < rows(t,i)) && (0 < cols(t,i)))
                    pt(teta,t) = pt(teta,t) + img.(imageName)(rows(t,i), cols(t,i)) * distances(t,i);
                end
            end
        end
    end

    %Store each distance, row and column data for each projection
    distances_teta(:,:,teta) = distances;
    rows_teta(:,:,teta) = rows;
    cols_teta(:,:,teta) = cols;
end

%save('lena90_555', 'pt','distances_teta', 'rows_teta', 'cols_teta')

%% VISUALIZATION
% Comparison with the radon() function.

[y,xp] = radon(img.(imageName),thetaValues);

for teta = 1:length(thetaValues)
    ptNormalized(teta,:) = pt(teta,:) / max(pt(teta,:));
    yNormalized(:,teta) = y(:, teta) / max(y(:,teta));
end

projectionNumber = round(numberOfAngles/2);
figure;
g(1) = plot(pt(projectionNumber,:), "-r");
hold on
g(2) = plot(y(:,projectionNumber),"-b");
grid on
```

```
xlabel("t Values");
ylabel("p(t)");
legend(g, 'myOutput', 'radon()')
title("theta:", thetaValues(projectionNumber));
%}
```

8.2. BackProjection.m

```
clear; close all; clc;

%% INPUTS

projectionName = input('Enter the name of the projected data: ','s');
data = load(projectionName);
projections = data.pt;
M = max(max(max(data.rows_teta)));
N = max(max(max(data.cols_teta)));

[numberOfProjections, numberOfSamples] = size(projections);

stepSize = (180 - 0) / numberOfProjections;
thetaValues = linspace(0, (180 - stepSize), numberOfProjections);

xCoor = floor(-M/2):ceil(M/2);
yCoor = floor(-N/2):ceil(N/2);

%% FILTER

projectionsFiltered = filterRamlak(projections);

%% BACKPROJECTION

image = zeros(M,N);
imageFiltered = zeros(M,N);
distances = zeros(numberOfSamples, (length(xCoor)-1));
rows = zeros(numberOfSamples, (length(xCoor)-1));
cols = zeros(numberOfSamples, (length(yCoor)-1));
pt = zeros(numberOfSamples);
ptFiltered = zeros(numberOfSamples);

for teta = 1:numberOfProjections
   distances = data.distances_teta(:,:,teta);
   rows = data.rows_teta(:,:,teta);
   cols = data.cols_teta(:,:,teta);
   pt = projections(teta, :);
   ptFiltered = projectionsFiltered(teta, :);

   for t = 1:numberOfSamples
      for i = 1:(length(xCoor)-1)
         if((0 < rows(t,i)) && (0 < cols(t,i)))
            image(rows(t,i), cols(t,i)) = image(rows(t,i), cols(t,i)) + pt(t) * distances(t,i);
            imageFiltered(rows(t,i), cols(t,i)) = imageFiltered(rows(t,i), cols(t,i)) + ptFiltered(t) * distances(t,i);
         end
      end
   end
end

%% VISUALIZATION

load lena
normalized = lena / max(lena(:));
imageNormalized = image / max(image(:));
imageFilteredNormalized = imageFiltered / max(imageFiltered(:));
```

```
%[R,xp] = radon(normalized,thetaValues);
I = iradon(projections', thetaValues','linear','none');
I = I / max(I(:));
I2 = iradon(projections', thetaValues');
I2 = I2 / max(I2(:));

% figure;
% imshow(normalized);
% title('Original Image')
figure;
imshow(I);
title('iradon unfiltered')
figure;
imshow(I2);
title('iradon with filter')
figure;
imshow(imageNormalized);
title('BP Image unfiltered')
figure;
imshow(imageFilteredNormalized);
title('BP Image with filter')
```

8.3. filterRamlak.m

```
function sinogram = filterRamlak(sinogram)
% Filter image for before backprojection in frequency domain
% using ramlak triangular filter
%
% sinogram: image converted to radon space
% make a Ram-Lak filter. it's just abs(f).

sinogram = sinogram';
N1 = size(sinogram,1);
freqs=linspace(-1, 1, N1).';
myFilter = abs(freqs);
%myFilter = repmat(myFilter, [1 180]);

% FT domain filtering
ft_R = fftshift(fft(sinogram, [], 1), 1);
filteredProj = ft_R .* myFilter;
filteredProj = ifftshift(filteredProj, 1);
sinogram = real(ifft(filteredProj,[],1));
sinogram = sinogram';
```