Berfin Kavşut
21602459
EEE475
April 2, 2020

REPORT FOR HOMEWORK #2

Part 1

Q1.1 - Spiral Trajectory

In part 1, we are reconstructing images taken with spiral trajectory in k-domain. spiral.mat has data: ktraj and kdata. The spiral trajectory is interleaved spiral trajectory. There are 6 spiral trajectories inside each other. Each interleaf is taken in separate repetition times. Each trajectories has 2048 data points.

1. Code

```
load('spiral.mat');
%"interleaved spiral trajectory"
% with 6 interleaves

figure; set(gcf, 'WindowState', 'maximized'); plot(ktraj);
xlabel('k_{x}'); ylabel('k_{y}');
title('2D k-space Trajectory');
```

2. Plots

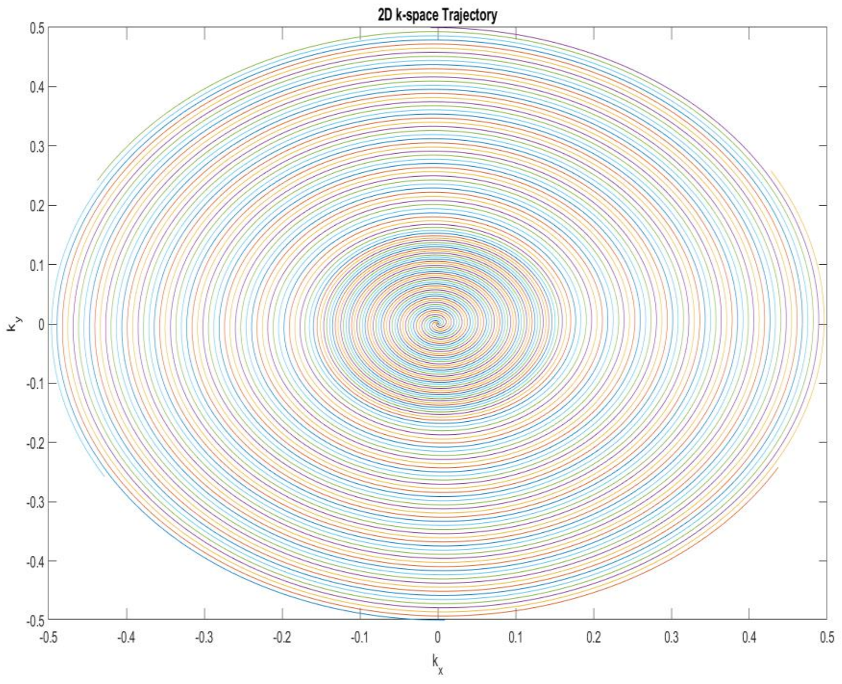

Fig. 1. 2D k-space Trajectory

Q1.2 – Calculating Density Compensation Filter

As it can be seen from Fig. 1, the density of data points is higher as we come to the origin in k-space. This density difference between center and outer parts causes more emphasis on center in k-domain. We want to get rid of density problem, hence we filter our k-space data before going back to image domain. The filter should lower low-frequencies and amplify higher frequencies.

For density compensation filter, we used the given function voronoidens(). Voronoid technique simply takes the area related to each data point. For each data point, it looks at the nearest data point and draws a line. From the middle point of the line, it draws an orthogonal line. With drawing all orthogonal lines around the data point, data point is surrounded by a closed area. The areas around data points after voronoid technique are shown in Fig. 2.

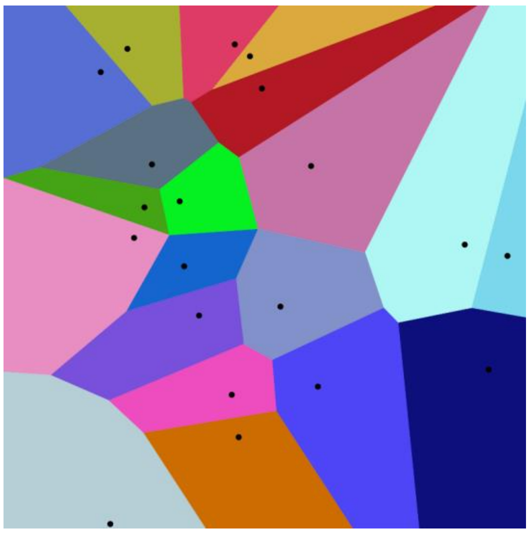

Fig. 2.Voronoid Technique

1. Code

```
%density compensation filter = 1/density = area
area = voronoidens(ktraj);
%density compensation filter until 2000th Data Point
area_zoom = area(1:2000,:);

%number of NaN's in area
count_nan = sum(isnan(area));
display(strcat('# of NaN for each trajectory: ',
num2str(count_nan(1))));

figure; set(gcf, 'WindowState', 'maximized');
subplot(2,1,1); plot(area(:));
xlabel('i^{th} Data Point'); ylabel('Area');
title('Density Compensation Filter')

subplot(2,1,2); plot(area_zoom(:));
xlabel('i^{th} Data Point'); ylabel('Area');
title('Density Compensation Filter until 2000^{th} Data Point');
```

2. Display Results

```
1.2
Calculating Density Compensation Filter
# of NaN for each trajectory:31
```

3. Plots

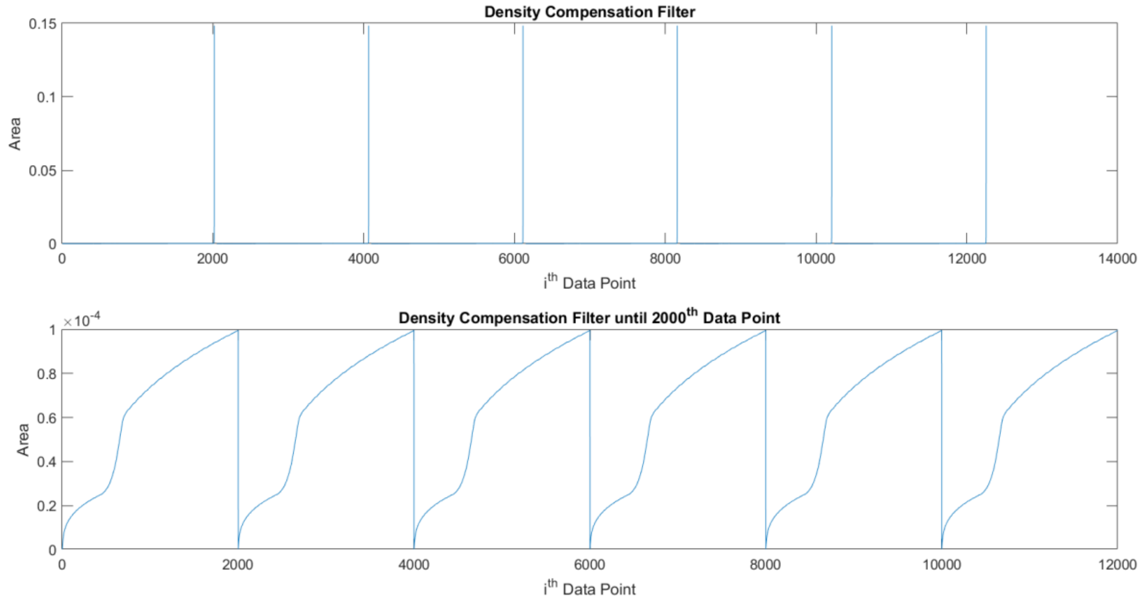

Fig. 3. Density Compensation Filter

First, we had undesired jumps and NaN's when we came to the end of trajectory from the center. The areas at the end cannot be calculated, because the data points needs another closer point to calculate its corresponding area, but there is no data point nearby. Therefore, we have NaN's. The number of NaN's in one interleaf is 31.

Q1.3 – Correcting the Density Compensation Filter

To avoid unreasonably large values, we used simple thresholding. The area for 2000th data point in each spiral trajectory was 0.0001, therefore I chose 0.0001 as the threshold for maximum value. There are no undesired values at the end.

1. Code

```
%NaN's and unreasonably large values are eliminated by simple
tresholding
area = voronoidens(ktraj);
corrected_dens = correct_dens(area);

correct_dens function
function corrected = correct_dens(area)
%
% function corrected = correct_dens(area);
%
% input:  area calculated for density compensation filter
% output: density compensation filter without NaN's
%         and unreasonably large values

[m,n] = size(area);
corrected = 0.0001*ones(size(area));
```

```
nan_check = isnan(area);
for m = 1:m
    for n = 1:n
        if (nan_check(m,n) == 0)
            corrected(m,n) = area(m,n);
        end
    end
end
corrected(corrected >= 0.0001) = 0.0001;
end
```

2. Plots

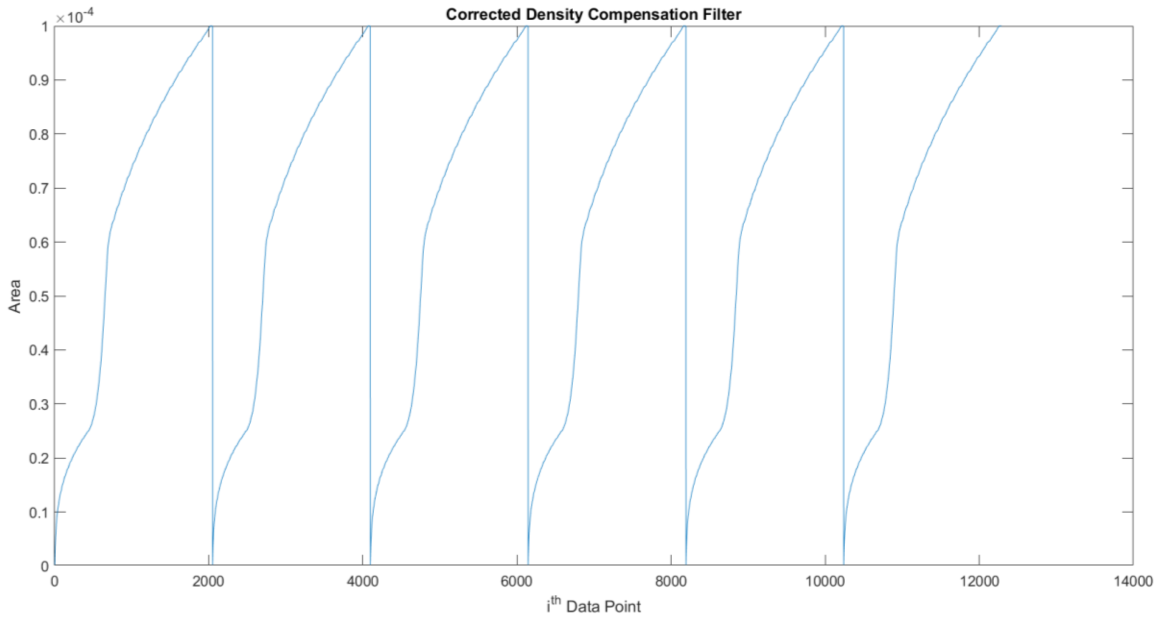

Fig. 4. Corrected Density Compensation Filter

Q1.4 – Direct Summation Technique

We will use the result image of direct summation reconstruction as our reference image. Although its image quality is very good, it consumes too much time. Hence, this technique is used for having reference images. We will implement gridding reconstruction with two different oversampling ratios and compare their results with direct summation image.

CPU time is 23.5469, which is a lot higher compared to 1X gridding reconstruction and 2X gridding reconstruction.

1. Code

```
load('spiral.mat');
%density compensation filter
area = voronoidens(ktraj);
d = correct_dens(area);
%for 128x128 image
N = 128;
%normalized image calculated by direct summation technique
```

```
%reference image for the rest of Part 1
[ima_direct,cpu_time] = direct_summation(d,N,ktraj,kdata);
%flip the image to have correct orientation
%on y-axis with imshow function
ima_direct = flipud(ima_direct);

%display the magnitude image
imshow(ima_direct,[]);

%display the central horizontal cross-section of the image
plot(ima_direct(end/2,:));
%display the central vertical cross-section of this image
plot(flip(ima_direct(:,end/2)));


disp(strcat('CPU time needed for the direct summation technique: ',
num2str(cpu_time), ' seconds'));
```

direct summation function

```
function [image,time] = direct_summation(d,N,ktraj,kdata)
%
% function [ima_direct,time] = direct_summation(d,N,ktraj,kdata)
%
% input:  d = density compensation filter
%         N = #of pixels, image has dimensions of NxN
%         ktraj = kx + i ky, k-space trajectory
%         kdata = k-space data
% output: ima_direct = normalized magnitude image

%start the timer
t=cputime;

%(kx,ky) coordinates
kx = real(ktraj);
ky = imag(ktraj);

%tau is the center of field-of-view relative to the image
%for going from -64 to +64 in x-y coordinates
tau =  N/2;

%direct summation technique
ima_direct = zeros(N,N);
for m = 0:N-1
    for  n = 0:N-1
      ima_direct(m+1,n+1) = sum(sum(d.*kdata.*exp(1i*2*pi*(kx.*(m-
tau) + ky.*(n-tau)))));
    end
end

%take magnitude image
image = abs(ima_direct);
%normalize the magnitude image
image = image/max(image(:));
```

```
time = cputime-t;

end
```

2. Display Results

```
1.4
Direct Summation
CPU time needed for the direct summation technique:23.5469
```

3. Images and Plots

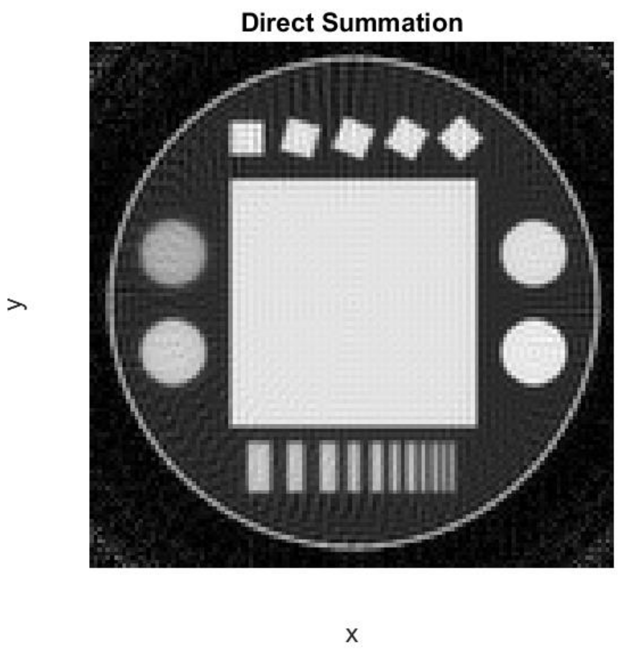

Fig. 5. Direct Summation Image

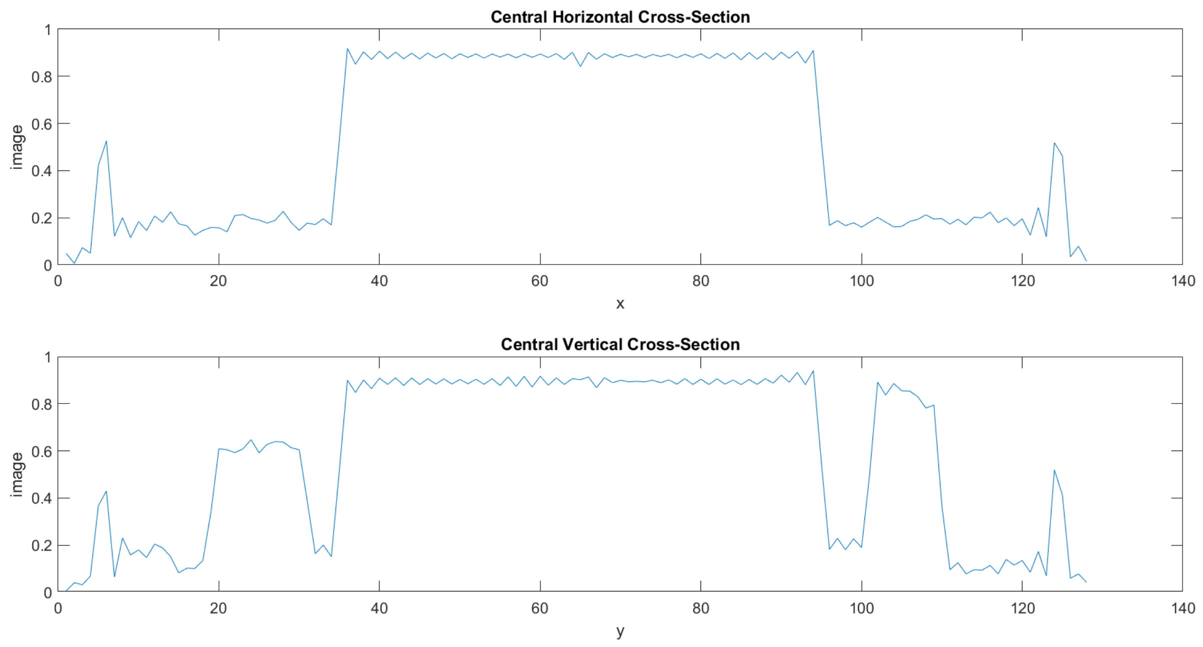

Fig. 6. Central Cross-Sections of the Image

From Fig. 5 and Fig. 6, we can see the image has blurring. The blurring is not like the aliasing in Cartesian images, and the spiral trajectory has blurring like turning around the center. It is hard to detect aliasing in one specific direction with our visual abilities.

Q1.5 - Simple Gridding Without Density Compensation

Let us summarize gridding reconstruction verbally. We have k-space data and we are sampling it by multiplying with a sampling function in k-space. Sampling function depends on the Non-Cartesian imaging technique. We are convolving the sampled k-space data with a suitable kernel. Rect-kernel is the perfect kernel, but the image domain has sinc as dual of rect function. This is not desired because sinc do not decay rapidly and our image inside FOV is affected by replicas, therefore we need another approach for kernels. The popular choice in MRI is Kaiser-Bessel window. It is a low-pass filter. It decays faster than sinc function, and it is smooth. It gives reasonable weighting to lower frequencies and higher frequencies.

We talked about the density compensation filter before. In this question, compensation filter is not used. We should have lowered the low frequencies by multiplying with a ramp function, which is the density compensation filter in Q1.3. Therefore, low frequency components of image dominates high frequencies. We see a smooth, blurry image. This can also be seen from Fig. 8. The cross-sections are very smooth and their high-frequency components are attenuated.

The advantage of gridding reconstruction can be seen on the change of CPU time. CPU time needed for direct summation is 16.6563 and CPU time needed for 1X gridding reconstruction is 0.046875. CPU time decreased to 0.281% of direct summation.

PSNR is 12.1761 and SSIM is 0.30184. They will get higher values as we improve our gridding reconstruction.

1. Code

```
%no density compensation
t = cputime;
w = ones(size(kdata));
ima = gridkb(kdata,ktraj,w,128,1,2,'image');
time_simpleGrid = cputime-t;

%correct orientation on y-axis for imshow function
image = flipud(ima);
%normalize the image
image = abs(image);
image = image./max(image(:));

%error image is difference of reference image
%and simple gridding image
error = ref-image;

%display reference image
imshow(ref,[]);

%display result image
imshow(image,[]);

%display magnitude of error image
```

```
%to set the zero-error pixel to colour black
imshow(abs(error),[]);

%display central horizontal cross-sections of the image
plot(ref(end/2,:),'m');
hold on;  plot(image(end/2,:),'b');


%display the central vertical cross-section of the image
plot(flip(ref(:,end/2)), 'm');
hold on;  plot(flip(image(:,end/2)),'b');

%IQA results
PSNR = psnr(image,ref);
SSIM = ssim(image,ref);
disp(strcat('PSNR: ', num2str(PSNR)));
disp(strcat('SSIM: ', num2str(SSIM)));

%CPU times time
disp(strcat('CPU time needed for direct summation: ',
num2str(time_directSum)));
disp(strcat('CPU time needed for 1X gridding reconstruction: ',
num2str(time_simpleGrid)));
```

2. Display Results

```
1.5
Simple Gridding Without Density Compensation
PSNR:12.1761
SSIM:0.30184
CPU time needed for direct summation:16.6563
CPU time needed for 1X gridding reconstruction:0.046875
```

3. Images and Plots

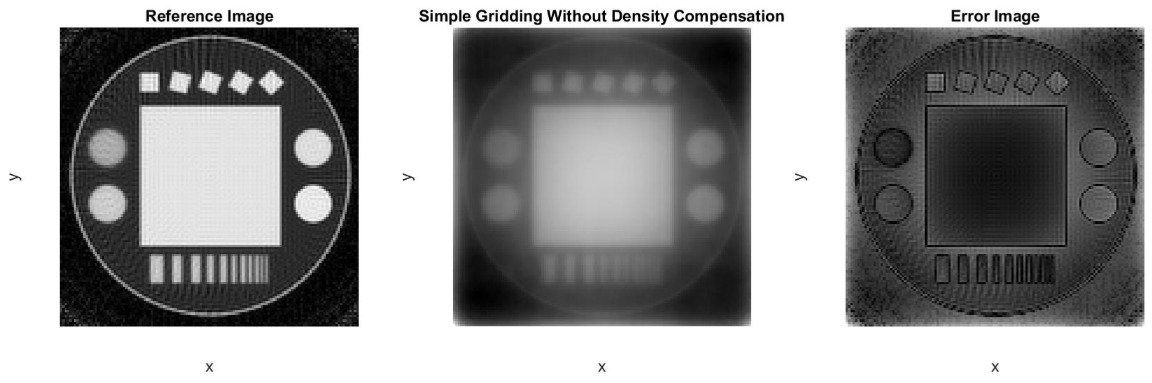

Fig. 7. Simple Gridding Without Density Compensation

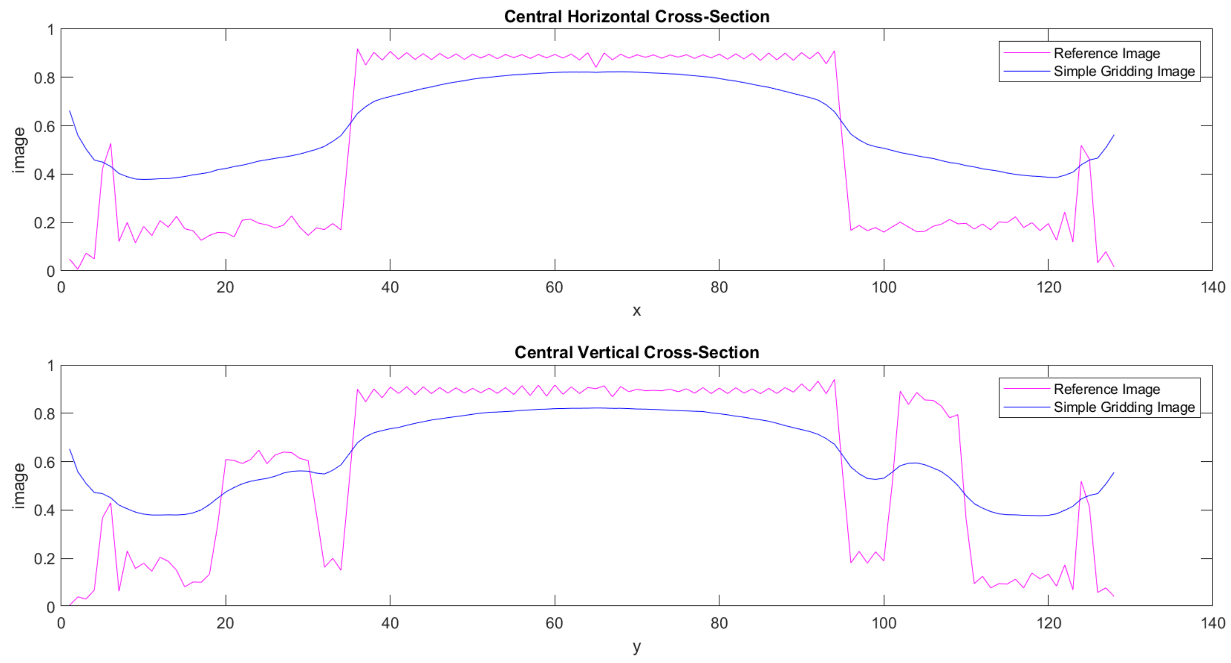

Fig. 8. Central Cross-Sections of 1X Gridding without Density Compensation

Q1.6 - Simple Gridding with Density Compensation

The weighting of low and high frequencies are not as we desire without density compensation. From Fig. 10, we can see the low frequency component, which is the big square in the middle, has less weighting compared to the small shapes around it. Still, the shapes are distinguishable and this does not affect visual performance of image a lot.

We are using ideal case for density compensation filters. We have pre-compensation of k-space data before convolving with kernel. We are correcting the density in k-space directly with the filter.

CPU time needed for direct summation is 15.3906, and CPU time needed for 1X gridding reconstruction is 0.0625. CPU time increased with density compensation, but still this increase is negligible.

PSNR is 23.5805 and SSIM is 0.7578, which are higher than the simple gridding results without density compensation. PSNR and SSIM increased considerably.

1. Code

```
%simple gridding with density compensation
t = cputime;
w = d;
ima = gridkb(kdata,ktraj,w,128,1,2,'image');
time_simpleGrid = cputime-t;

%correct orientation on y-axis for imshow function
image = flipud(ima);
%normalize the image
image = abs(image);
image = image./max(image(:));
```

```
%error image is difference of reference image
%and simple gridding image
error = ref-image;

%display reference image
imshow(ref,[]);

%display result image
imshow(image,[]);

%display magnitude of error image
%to set the zero-error pixel to colour black
imshow(abs(error),[]);

%display central horizontal cross-sections of the image
plot(ref(end/2,:),'m');
hold on;  plot(image(end/2,:),'b');


%display the central vertical cross-section of the image
plot(flip(ref(:,end/2)), 'm');
hold on;  plot(flip(image(:,end/2)),'b');

%IQA results
PSNR = psnr(image,ref);
SSIM = ssim(image,ref);
disp(strcat('PSNR: ', num2str(PSNR)));
disp(strcat('SSIM: ', num2str(SSIM)));

%CPU times time
disp(strcat('CPU time needed for direct summation: ',
num2str(time_directSum)));
disp(strcat('CPU time needed for 1X gridding reconstruction: ',
num2str(time_simpleGrid)));
```

2. Display Results

```
1.6

Simple Gridding With Density Compensation

PSNR:23.5805

SSIM:0.7578

CPU time needed for direct summation:15.3906

CPU time needed for 1X gridding reconstruction:0.0625
```

3. Images and Plots

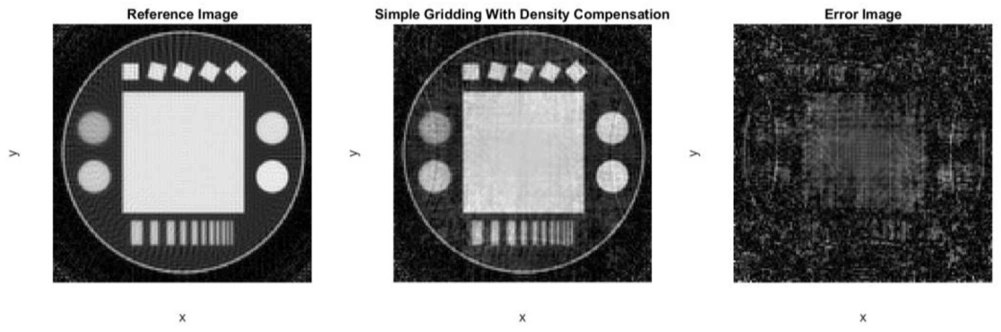

Fig. 9. Simple Gridding with Density Compensation

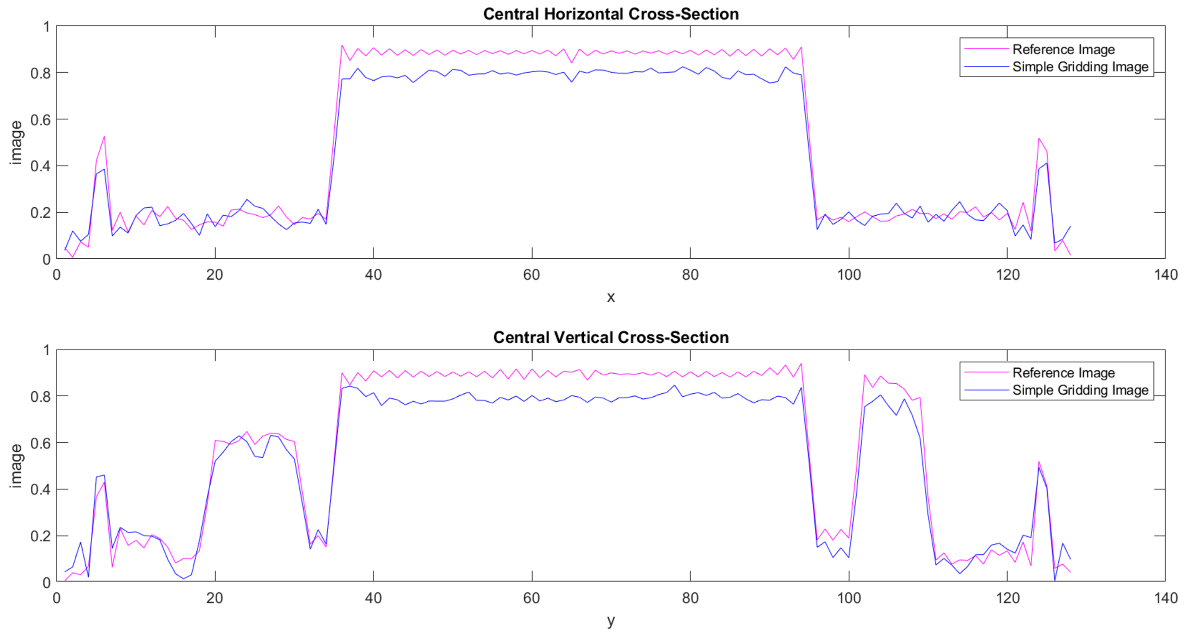

Fig. 10. Central Cross-Sections of 1X Gridding with Density Compensation

Q1.7 – 2X Gridding

Our image is better with density compensation now, but we have aliasing on the image. This can be seen from its error image in Fig. 8. We want to get rid of aliasing by separating replicas in image domain, and this means oversampling k-space data. When sampling frequency increases in k-space, the repeated replicas will get further away from each other in image domain.

PSNR is 51.0358, and SSIM is 0.99689. PSNR and SSIM increased drastically compared to 1X gridding reconstruction.

CPU time needed for 2X gridding reconstruction is 0.078125, it was 0.0625 for 1X gridding reconstruction. We have 25% increase in time. This did not affect our results very much, but it would be important for calculations with larger data.

1. Code

```
%2X Gridding
t = cputime;
%density compensation
w = d;
ima = gridkb(kdata,ktraj,w,128,2,4,'image');
time_2Xgrid = cputime-t;

%normalize full image
full_image = flipud(ima);
full_image = abs( full_image);
full_image =  full_image./max( full_image(:));

%crop the image by taking central 128x128
ima = ima(end/4+1:(3*end/4),end/4+1:(3*end/4));
%normalize the image
image = flipud(ima);
image = abs(image);
image = image./max(image(:));

%error image
error = ref - image;
```

2. Display Results

```
1.7
2X Gridding
PSNR:51.0358
SSIM:0.99689
CPU time needed for direct summation:15.9531
CPU time needed for 2X gridding reconstruction:0.078125
```

3. Images and Plots

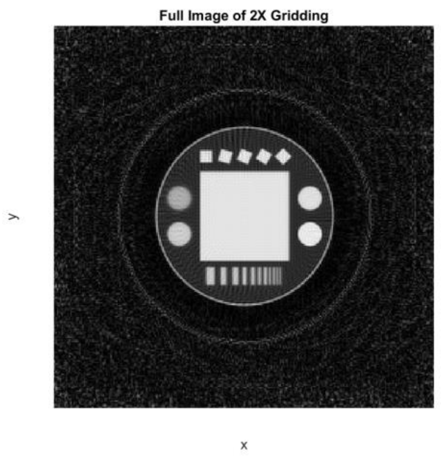

Fig. 11. Full Image of 2X Gridding

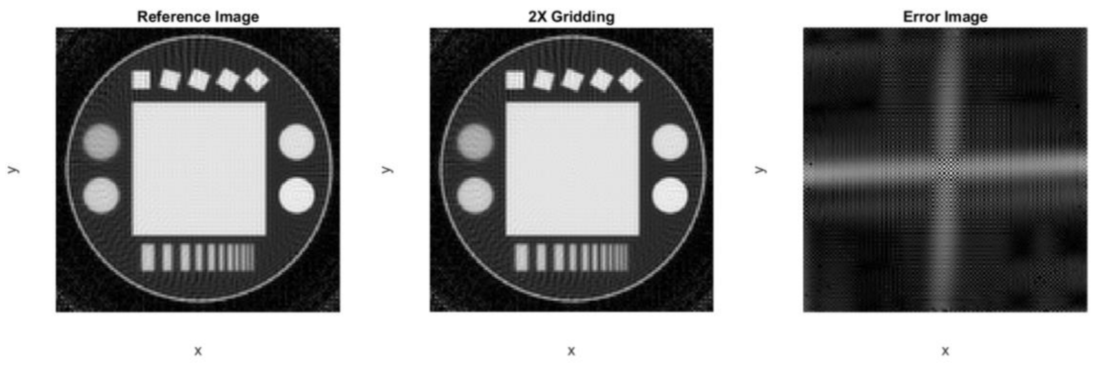

Fig. 12. 2X Gridding

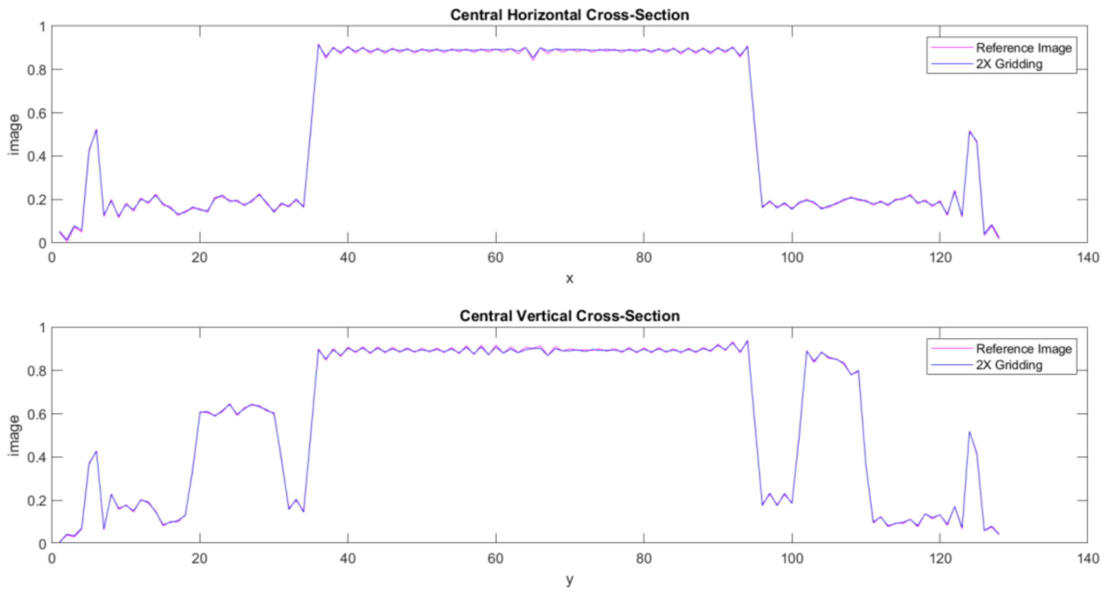

Fig. 13. Central Cross-Sections of 2X Gridding

Q1.8 – Reduced Oversampling for Gridding

The kernel widths of 5.5 and 6 are chosen from the figure in Beatty paper. The results are displayed for both kernel width of 5.5 and 6. Their visual results and IQA measurements are very close. But, CPU time needed for kernel width of 6 is 75.01% higher than CPU time needed for kernel width of 5.5. Therefore, I chose kernel width of 5.5 as our kernel width. In Q1.9, kernel width is 5.5, as well.

In question PSNR was 51.0358 and SSIM was 0.99689 in 2X gridding. In gridding with reduced oversampling ratio, PSNR is 45.0788 and SSIM is 0.99196. Even though IQA measures decreased a little, the difference in visual results cannot be detected by human eye. Also, central cross-sections are almost fully sitting on the reference image's central cross-sections.

1. Code

```
%reduced oversampling for gridding
t = cputime;
%density compensation
w = d;
```

```
%for oversampling factor 1.25, I have chosen kernel width to be 6
%so that aliasing amplitude is less than 10^-3
oversamp_factor = 1.25;
kernel_width = 5.5;
ima = gridkb(kdata,ktraj,w,N,oversamp_factor,kernel_width,'image');
time_rog = cputime-t;

%normalize full image
full_image = flipud(ima);
full_image = abs( full_image);
full_image =  full_image./max( full_image(:));

%crop the image by taking central 128x128
size_ima = 1.25*N;
ima = ima(size_ima/2-N/2+1:size_ima/2+N/2,size_ima/2-
N/2+1:size_ima/2+N/2);
%%normalize the image
image = flipud(ima);
image = abs(image);
image = image./max(image(:));

%error image
error = ref - image;
```

2. Display Results

*** Display results for kernel width 5.5:**

```
1.8

Reduced Oversampling for Gridding

PSNR:45.0788

SSIM:0.99196

CPU time needed for direct summation:15.7969

CPU time needed for gridding reconstruction with oversampling ratio
1.25: 0.0625
```

*** Display results for kernel width 6:**

```
1.8

Reduced Oversampling for Gridding

PSNR:44.3217

SSIM:0.99106

CPU time needed for direct summation:16.2813

CPU time needed for gridding reconstruction with oversampling ratio
1.25: 0.10938
```

3. Images and Plots

<u>For kernel width of 5.5:</u>

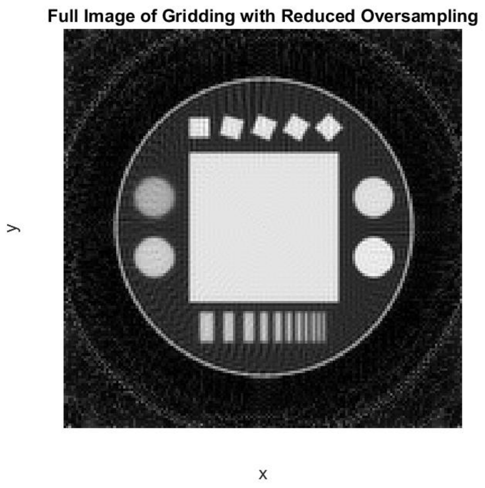

Fig. 14. Full Image of Gridding with Reduced Oversampling, Kernel Width = 5.5

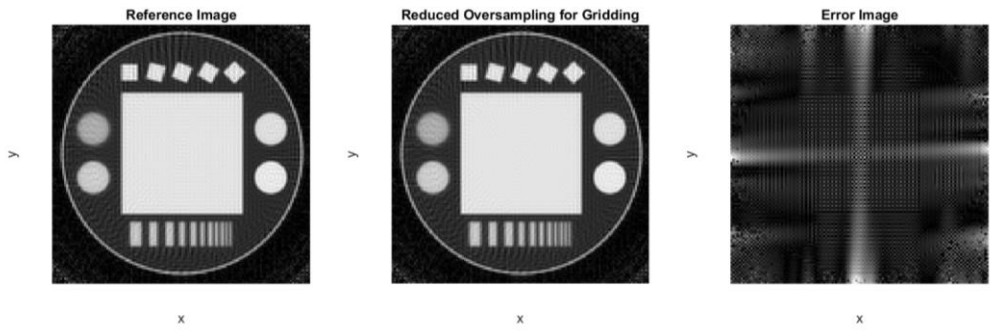

Fig. 15. Gridding with Reduced Oversampling, Kernel Width = 5.5

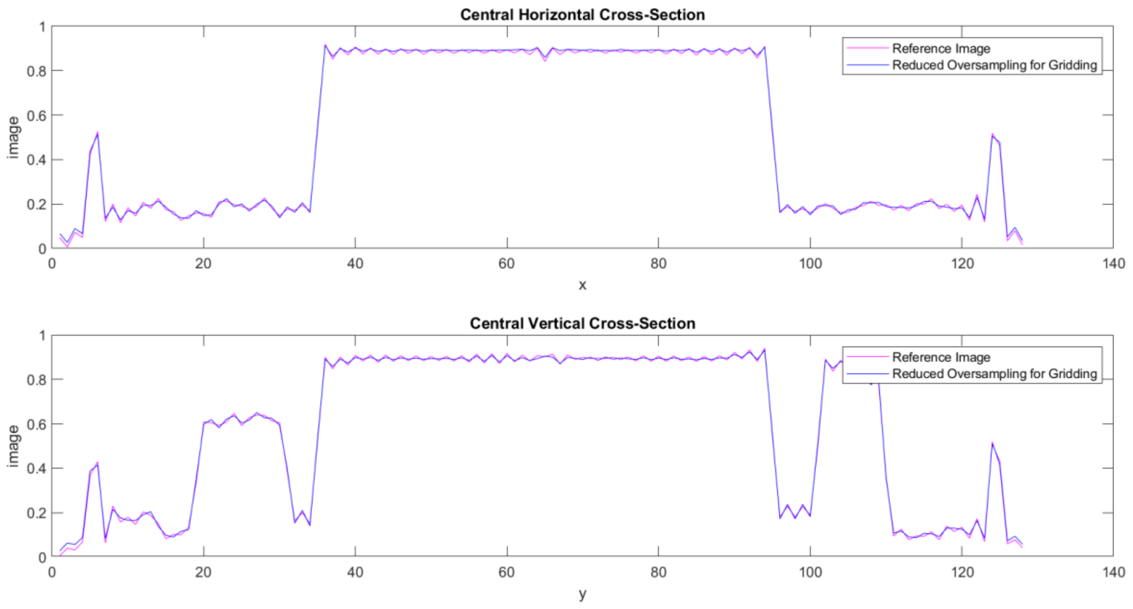

Fig. 16. Central Cross-Sections of Gridding with Reduced Oversampling, Kernel Width = 5.5

For kernel width of 6:

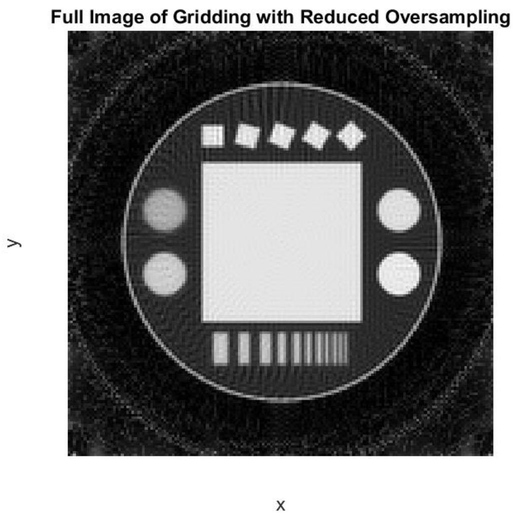

Fig. 17. Full Image of Gridding with Reduced Oversampling, Kernel Width = 6

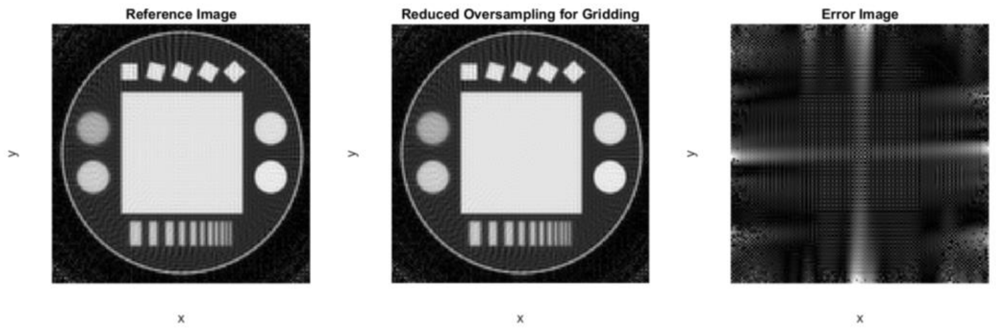

Fig. 18. Gridding with Reduced Oversampling, Kernel Width = 6

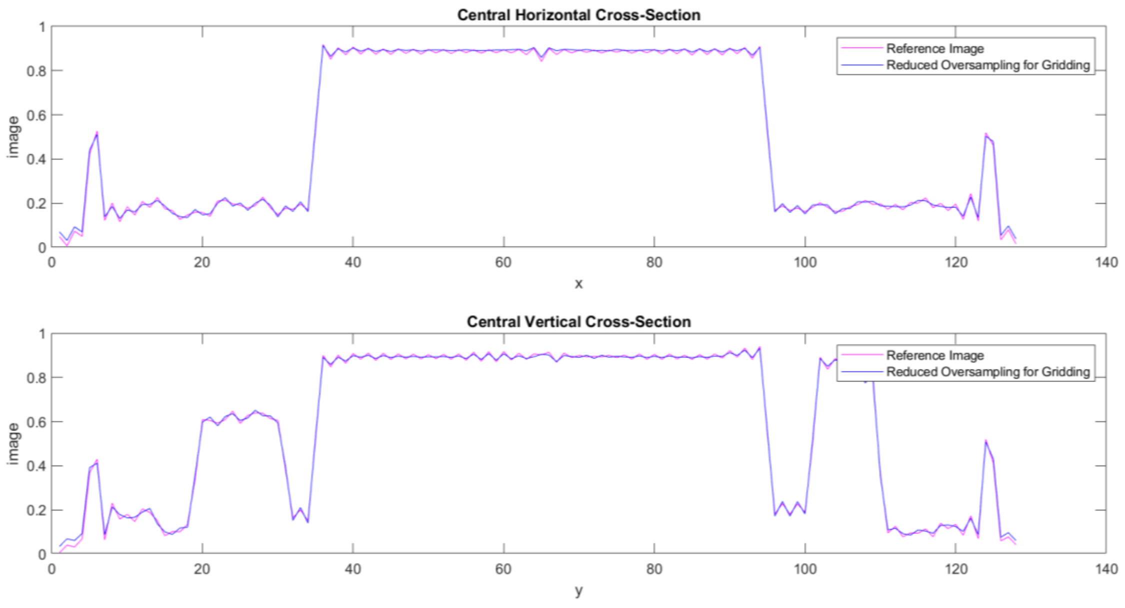

Fig. 19. Central Cross-Sections of Gridding with Reduced Oversampling, Kernel Width = 6

Q1.9 – Effect of Deapodization

We have apodization in image domain due to multiplication with the kernel in image domain. It has two effects on our image: side lobes are apodized by gridding kernel, which is positive and the main image is affected, which is negative. We want to divide out the apodization inside FOV after we calculate the image in the last step and get rid of this negative effect of apodization. If we do not deapodize the image, the image will decay down when we go away from the center of image, which can be seen in Fig. 21. The effect of apodization can be seen clearer in Fig. 22 when we look at the central cross-sections of result image. PSNR and SSIM values decreased drastically due to the effect of deapodization.

1. Code

```
load('spiral.mat');
%reference image
area = voronoidens(ktraj);
d = correct_dens(area);
N = 128;
[ref,time_directSum] = direct_summation(d,N,ktraj,kdata);
ref = flipud(ref);

%reduced oversampling for gridding
t = cputime;
%density compensation
w = d;
%for oversampling factor 1.25, I have chosen kernel width to be 6
%so that aliasing amplitude is less than 10^-3
oversamp_factor = 1.25;
kernel_width = 5.5;
%'k-space' is for eliminatig the effedct of deapodization
kd = gridkb(kdata,ktraj,w,N,oversamp_factor,kernel_width,'k-space');
time_rog = cputime-t;
ima = ifft2c(kd);

%normalize full image
full_image = flipud(ima);
full_image = abs( full_image);
full_image =  full_image./max( full_image(:));

%crop the image by taking central 128x128
size_ima = 1.25*N;
ima = ima(size_ima/2-N/2+1:size_ima/2+N/2,size_ima/2-
N/2+1:size_ima/2+N/2);
%%normalize the image
image = flipud(ima);
image = abs(image);
image = image./max(image(:));

%error image
error = ref - image;
```

2. Display Results

```
1.9

Effect of Deapodization
```

```
PSNR:13.3772

SSIM:0.35743

CPU time needed for direct summation:16.0313

CPU time needed for gridding reconstruction without
deapodization:0.10938
```

3. Images and Plots

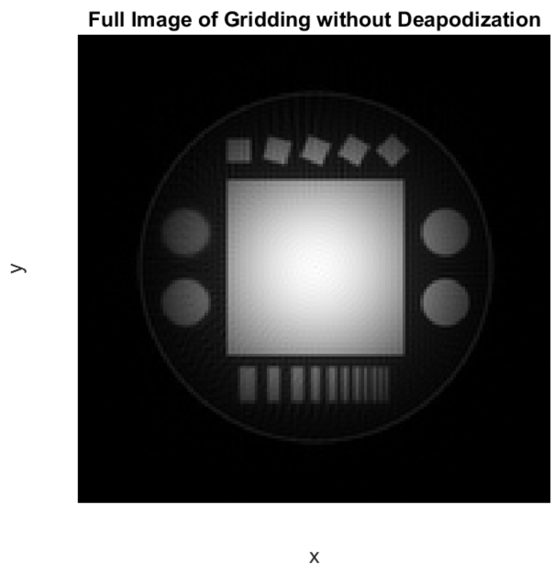

Fig. 20. Full Image of Gridding without Deapodization, Kernel Width = 5.5

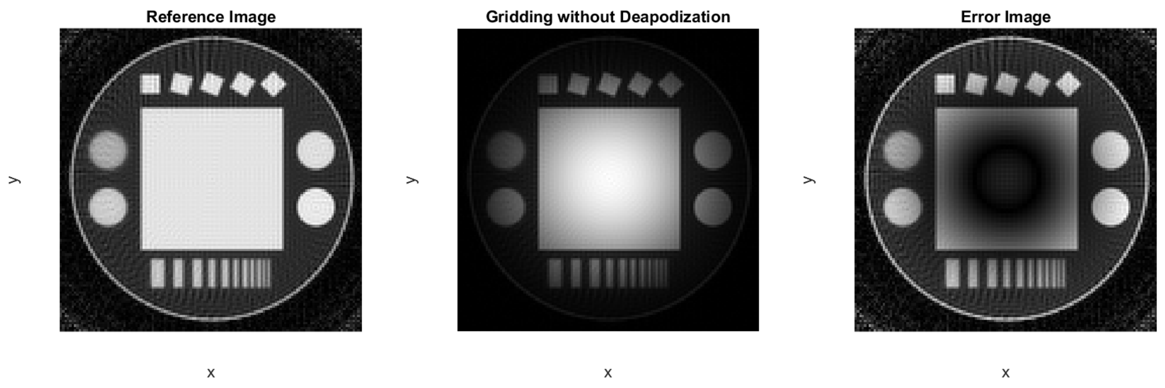

Fig. 21. Gridding without Deapodization, Kernel Width = 5.5

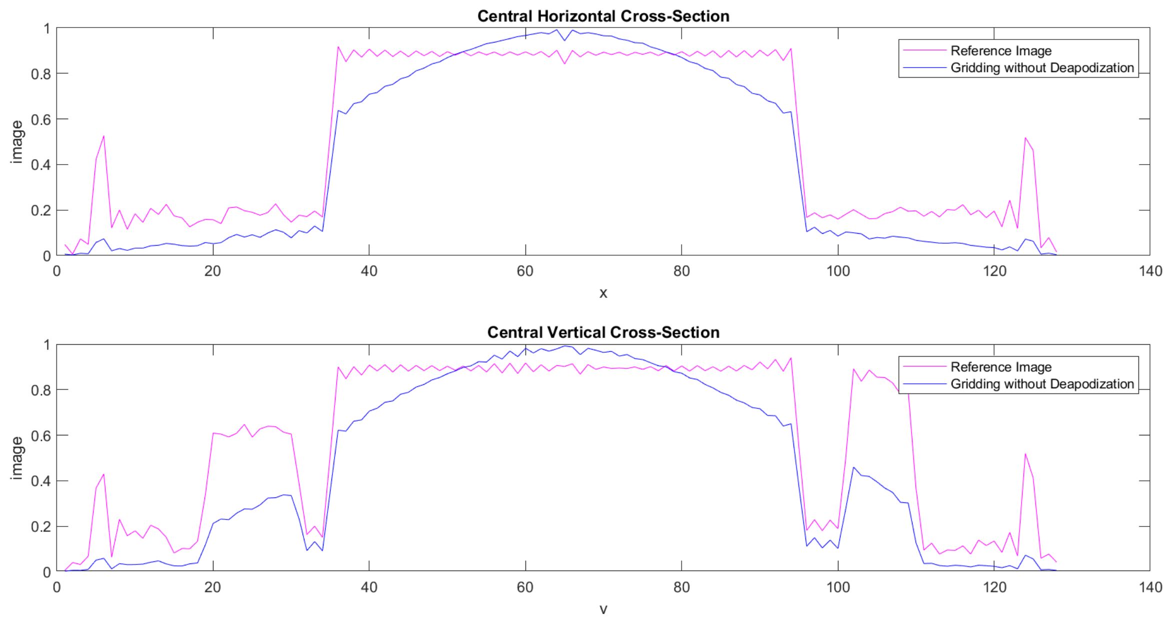

Fig. 22. Central Cross-Sections of Gridding without Deapodization, Kernel Width = 5.5

<u>Part 2</u>

Q2.1 – Generating Projections

We are using Modified Shepp-Logan image of MATLAB as our reference image throughout Part 2 and 3. We had the projections of the image by using radon function in MATLAB. The dimensions of proj is 367x180, and we have 180 projections.

In sinogram, θ increases upwards and *l* increases from left to right. The proj is transposed to have θ on y-axis and *l* on x-axis.

1. Code

```
P = phantom('Modified Shepp-Logan',256);
proj = radon(P,[0:179]);

%normalize phantom image to use as reference image
ref = abs(P);
ref = ref/max(ref(:));

%take transpose to have L in x-axis
%and Theata in y-axis
%then flip up-down to correct axis orientation
sinogram = flipud(transpose(proj));

%display reference image
figure; set(gcf, 'WindowState', 'maximized');
subplot(1,2,1);
imshow(abs(ref),[]);
xlabel('x'); ylabel('y');
title('Reference Image');

%display sinogram
```

```
subplot(1,2,2);
imshow(sinogram,[]);
xlabel('{\it l}'); ylabel('\theta')
title('Sinogram');
saveas(gcf,'2.1.png');

%display the central horizontal cross-section of the image
figure; set(gcf, 'WindowState', 'maximized');
subplot(2,1,1);
plot(ref(end/2,:),'m');
xlabel('x'); ylabel('image');
title('Central Horizontal Cross-Section');

%display the central vertical cross-section of the image
subplot(2,1,2);
plot(flip(ref(:,end/2)), 'm');
xlabel('y'); ylabel('image');
title('Central Vertical Cross-Section');
```

2. Images and Plots

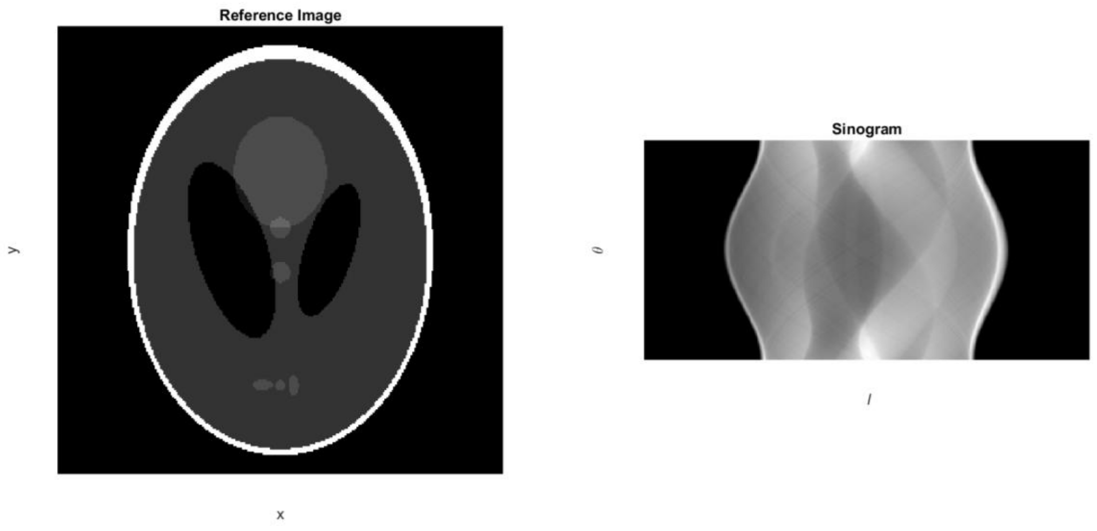

Fig. 23. Reference Image and Sinogram g(*l*,θ)

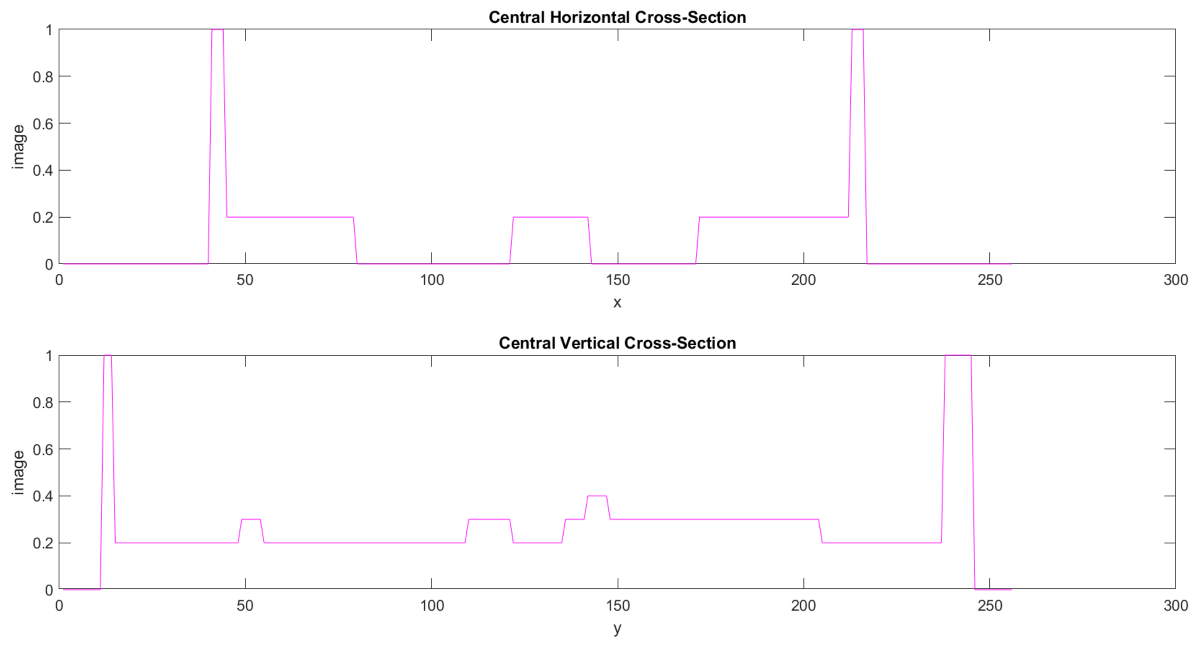

Fig. 24. Central Cross-Sections of Reference Image

Q2.2 – Backprojection

We will reconstruct the image from its projections, which is proj matrix. We will use the default filter named as Ram-Lak in MATLAB. This filter is the ramp filter. When we calculate the density compensation filter later, we will see this means using density compensation filter on k-space data.

PSNR is 26.8242 and SSIM is 0.67993. These IQA values are acceptable. We had error on edges, therefore their values are affected by these errors on edges. Unfortunately, we will not be able to get rid of the error on edges as we move on, but it does not affect the visual performance of images a lot.

1. Code

```
%reference image
P = phantom('Modified Shepp-Logan',256);
proj = radon(P,[0:179]);
ref = abs(P);
ref = ref/max(ref(:));

%filtered backprojection
image = iradon(proj,0:179);
image = image(2:end-1,2:end-1);
%normalize the image
image = abs(image);
image = image/max(image(:));

%error image
error = ref - image;

%display reference image
figure; set(gcf, 'WindowState', 'maximized');
```

```
subplot(1,3,1);
imshow(ref,[]);
xlabel('x'); ylabel('y');
title('Reference Image');

%display result image
subplot(1,3,2);
imshow(image,[]);
xlabel('x'); ylabel('y');
title('Filtered Backprojection');

%display magnitude of error image
%to set the zero-error pixel to colour black
subplot(1,3,3);
imshow(abs(error),[]);
xlabel('x'); ylabel('y');
title('Error Image');

%display central cross-sections of the image
%compare reference image cross-sections with filtered backprojection
figure; set(gcf, 'WindowState', 'maximized');
subplot(2,1,1);
plot(ref(end/2,:),'m');
hold on;  plot(image(end/2,:),'b');
xlabel('x'); ylabel('image');
title('Central Horizontal Cross-Section');
legend({'Reference Image','Filtered
Backprojection'},'Location','NorthEast');

subplot(2,1,2);
plot(flip(ref(:,end/2)), 'm');
hold on;  plot(flip(image(:,end/2)),'b');
xlabel('y'); ylabel('image');
title('Central Vertical Cross-Section');
legend({'Reference Image','Filtered
Backprojection'},'Location','NorthEast');


%IQA results
PSNR = psnr(image,ref);
SSIM = ssim(image,ref);
disp(strcat('PSNR: ', num2str(PSNR)));
disp(strcat('SSIM: ', num2str(SSIM)));
```

2. Display Results

```
2.2

Backprojection

PSNR:26.8242

SSIM:0.67993
```

3. Images and Plots

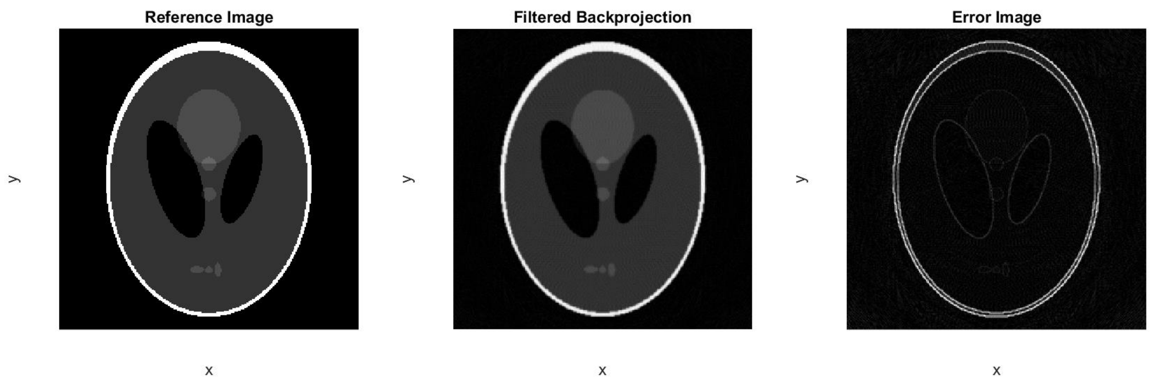

Fig. 25. Filtered Backprojection

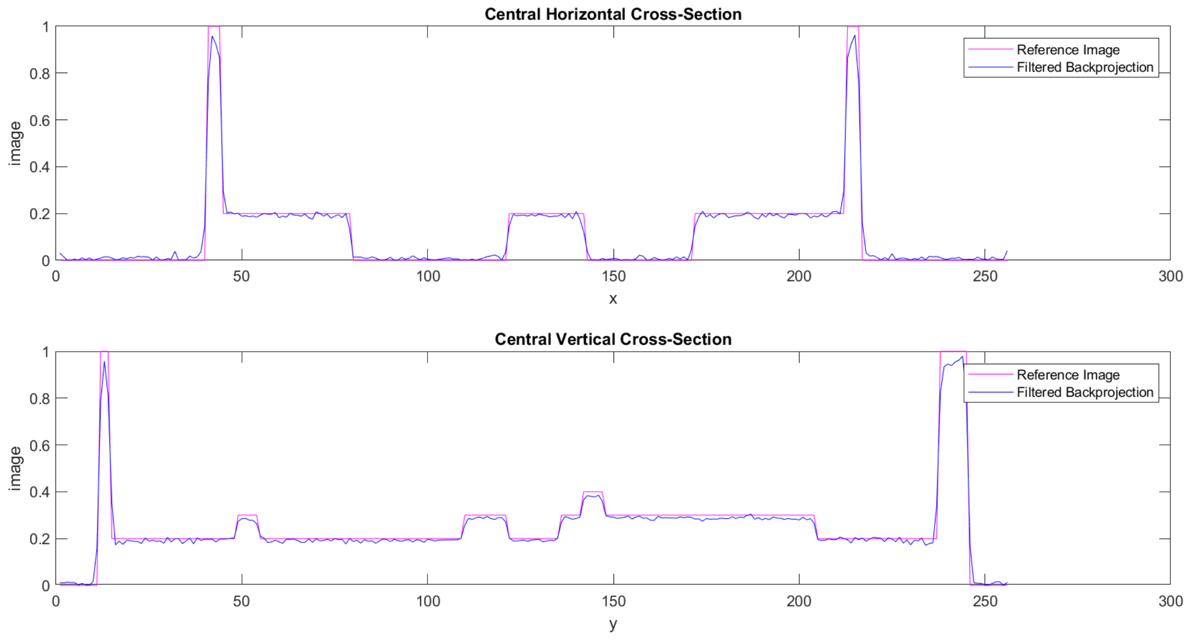

Fig.26. Central Cross-Sections of Filtered Backprojection

Q2.3 - Naive Backprojection

Now, we will see the backprojection without filtering projections with ramp filter. Similar to spiral trajectory case, low frequencies are dominating high frequencies. The image became blurry.

PSNR is 4.7234 and SSIM is 0.15346. PSNR has decreased, because we have more noise on background. The decrease in SSIM can be seen from error image visually, as well.

1. Code

```
%reference image
P = phantom('Modified Shepp-Logan',256);
proj = radon(P,[0:179]);
ref = abs(P);
ref = ref/max(ref(:));
```

```
%naive backprojection
image = iradon(proj,0:179,'none');
image = image(2:end-1,2:end-1);
%normalize the image
image = abs(image);
image = image/max(image(:));
%error image
error = ref - image;
```

2. Display Results

```
2.3

Naive Backprojection

PSNR:4.7234

SSIM:0.15346
```

3. Images and Plots

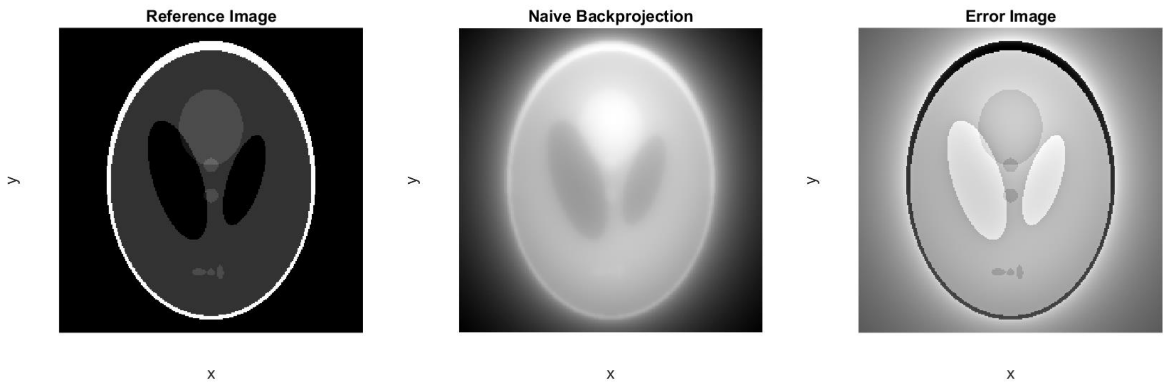

Fig. 27. Naive Backprojection

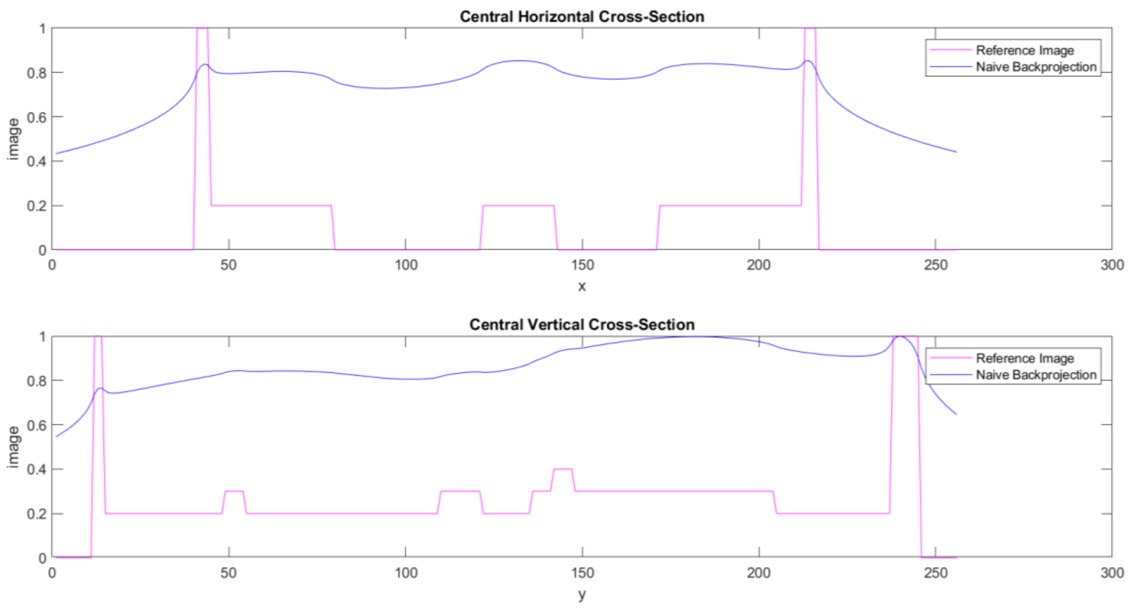

Fig.28. Central Cross-Sections of Naive Backprojection

Q2.4 – Generating k-space Data from Projections

By projection slice theorem, we are taking 1D Fourier Transform of projections. FFT is for each projection, which means for each θ. fft1c function operates on each column, each column is projection with an angle θ.

1. Code

```
%projection slice theorem
P = phantom('Modified Shepp-Logan',256);
proj = radon(P,[0:179]);
[ktraj,kspace] = projection_slice(proj);
kspace = flipud(transpose(kspace));

%180 radial line in k-space
%theta is between [0,179]
%radius is between [-0.5,0.5]
figure; set(gcf, 'WindowState', 'maximized');
subplot(1,2,1); imshow(log(abs(kspace)+1),[]);
title('K-space Data');
xlabel('k_{\rho}'); ylabel('\theta');

subplot(1,2,2); plot(ktraj);
xlabel('k_{x}'); ylabel('k_{y}');
title('Radial Trajectory');
```

projection slice function

```
function [ktraj,kspace] = projection_slice(proj)
%
% function [ktraj,kspace] = projection_slice(proj)
%
% input:  proj = g(l,theta)
% output: ktraj = kx + j*ky
%         kspace = G(l,theta)

%line_no is equal to number of angles we are projecting with
%datapoint_no is number of values sampled for each projection
[datapoint_no,line_no] = size(proj);

%radius is from -0.5 to 0.5
radius = linspace(-0.5,0.5,datapoint_no);
%theta is from 0 to 180
%convert degrees to radians
theta = [0:line_no]/180*pi;

%ktraj: (kx,ky) = (r*cos(theta),r*sin(theta))
ktraj = zeros(datapoint_no,line_no);
for i = 1:180
    ktraj(:,i) = radius*exp(j*theta(i));
end

%take 1D FFT of proj for k-space
%fft() is applied to each column,
%which means we take FFT of each projection
```

```
kspace = fft1c(proj);

end
```

fft1c function

```
function d = fft1c(f)
% d = fft21(f)
%
% fft1c performs a centered fft
d = fftshift(fft(ifftshift(f)));
end
```

2. Plots

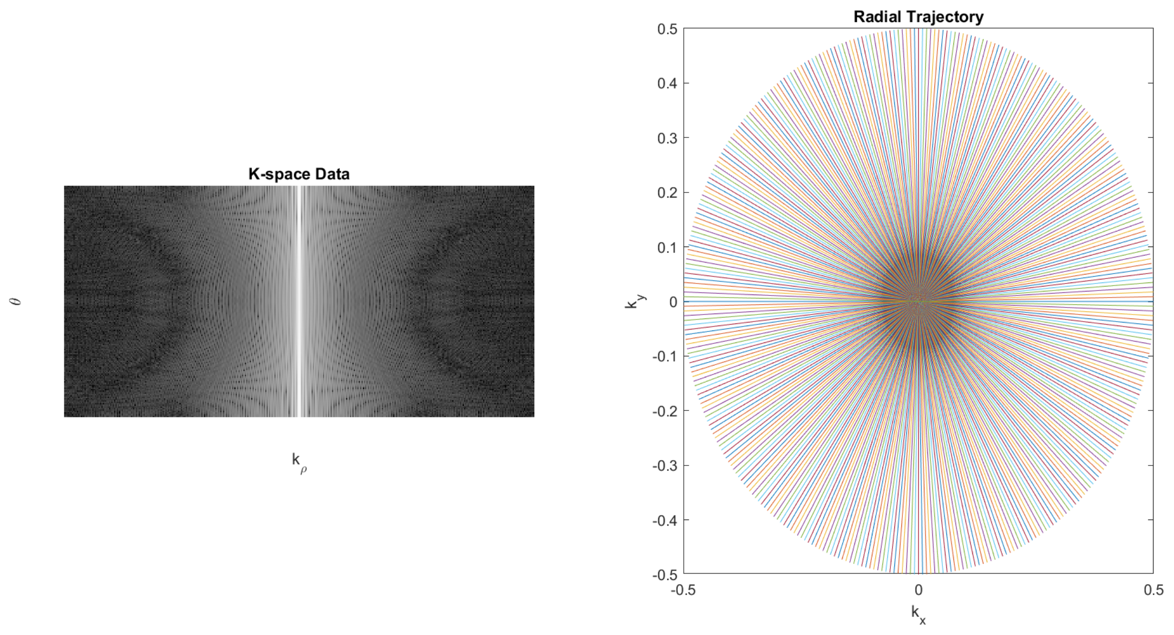

Fig. 29. K-Space Data and Radial Trajectory

Q2.5 – Density Compensation Filter for Radial Trajectory

With iradon, we used the default filter of MATLAB. Now, we will compute the density compensation filter by geometric approach. We will divide the circle in k-space into discs between consecutive k points in one radial line. Calculate the area for the disc, and assign the area for the density compensation filter for each sample point inside that disc.

There are two options for density compensation, since in Part 3, k values on radial lines do not start from 0.5, but from origin. Here, k values on radial lines start from origin.

1. Code

```
P = phantom('Modified Shepp-Logan',256);
proj = radon(P,[0:179]);

%density compensation filter calculated by geometric approach
[filter,rho] = density_compensate(proj,'opt1');
```

```
filter_first = filter(:,1);

%display the density compensation filter for radila trajectory
figure; set(gcf, 'WindowState', 'maximized');
plot(rho,filter_first);
xlabel('\rho'); ylabel('Area');
title('Density Compensation Filter for Radial Trajectory');
```

density_compensate_function

```
function [filter,rho] = density_compensate(proj,opt)
%
% function [filter,rho] = density_compensate(proj,opt)
%
% input:  proj = g(l,theta)
%         opt =   'opt1', 'opt2'
% output: filter = calculated area for each sample point
%         rho = k-rho vector for given projectory

%'opt1' : radial lines extend from -0.5 to 0.5
%'opt2' : radial extend from the center, i.e. from 0 to 0.5
switch opt

case 'opt1'

[datapoint_no,line_no] = size(proj);

%N is number of lines
N = line_no;

%k is between -0.5 and 0.5
k_interval = 1/(datapoint_no-1);

%each sample is in n^th disk
limit = (datapoint_no - 1)/2;
rho = [-limit:limit];
n = abs(rho);

%calculate area for each sample
area = zeros(1,datapoint_no);
%n^th sample
area = (pi/N)*(k_interval)^2.*n;
%DC disk
area(rho == 0) = (pi/(4*N))*(k_interval/2)^2;

%create filter for each data point
%columns are  equal, because filter is equal each projection
filter = transpose(repmat(area,[line_no 1]));

case 'opt2'

[datapoint_no,line_no] = size(proj);

%we treat lines as if 2 opposite lines are combined to 1 line
N = line_no/2;
```

```
%k is between 0 and 0.5
k_interval = 0.5/datapoint_no;

%each sample is in n^th disk
limit = datapoint_no-1;
rho = [0:limit];
n = rho;

%calculate area for each sample
area = zeros(1,datapoint_no);
%n^th sample
area = (pi/N)*(k_interval)^2.*n;
%DC disk
area(n == 0)  = (pi/(4*N))*(k_interval/2)^2;

%create filter for each data point
%columns are  equal, because filter is equal each projection
filter = transpose(repmat(area,[line_no 1]));

end

end
```

2. Plots

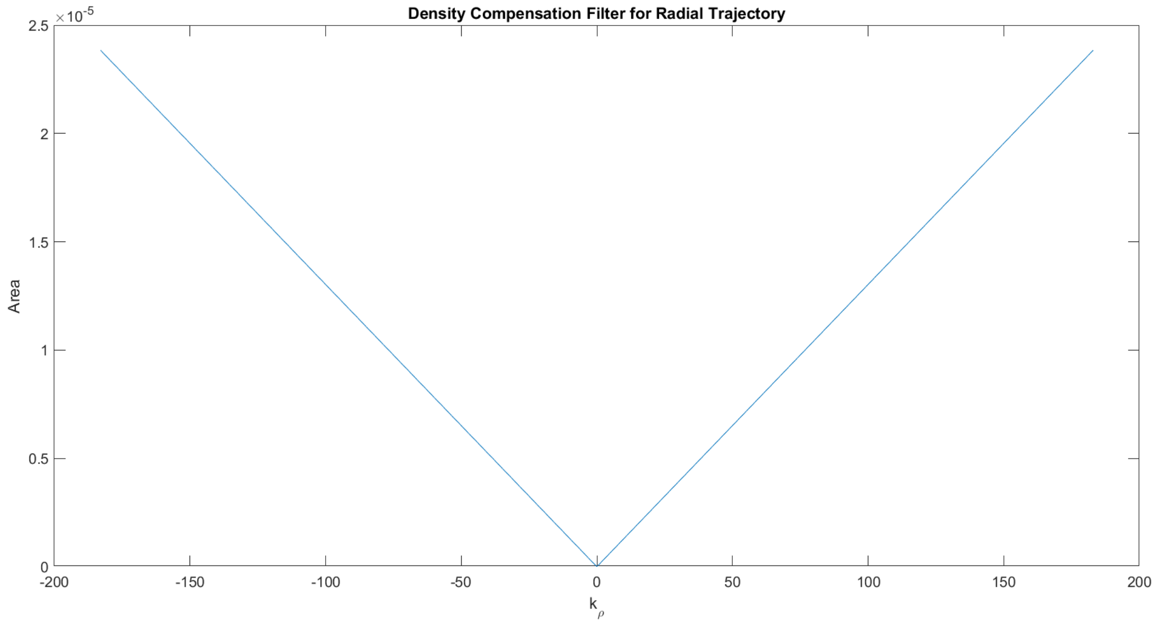

Fig. 30. Density Compensation Filter for Radial Trajectory

Computed with Geometric Approach

Q2.6 – Direct Summation

PSNR is 20.0632, and SSIM is 0.40186. We take the phantom itself as the reference image here. PSNR was 26.8242 and SSIM was 0.67993 in filtered backprojection. Our IQA assessments performed worse in direct summation. Also, this direct summation technique consumes much more time than iradon function.

When it is look carefully to the result image, there are radial ringing artifacts like circles starting from center and going outwards. We multiply k-space data with a ramp function, this corresponds to convolving with sinc^2 in image domain. Convolution with sinc functions results in ringing artifacts, like waves on images.

1. Code

```
%reference image
P = phantom('Modified Shepp-Logan',256);
proj = radon(P,[0:179]);
ref = abs(P);
ref = ref/max(ref(:));

%density compensation filter
[filter,rho] = density_compensate(proj,'opt1');
%projection slice theorem
[ktraj,kspace] = projection_slice(proj);
%filter
d = filter;
%256x256 image
N = 256;
%direct summation
[ima_direct,time] = direct_summation(d,N,ktraj,kspace);

%normalize the result image
image = abs(ima_direct);
image = image/max(image(:));
image = flipud(transpose(image));

%error image
error = ref - image;
```

2. Display Results

```
2.6

Direct Summation for Radial Trajectory

PSNR:20.0632

SSIM:0.40186
```

3. Images and Plots

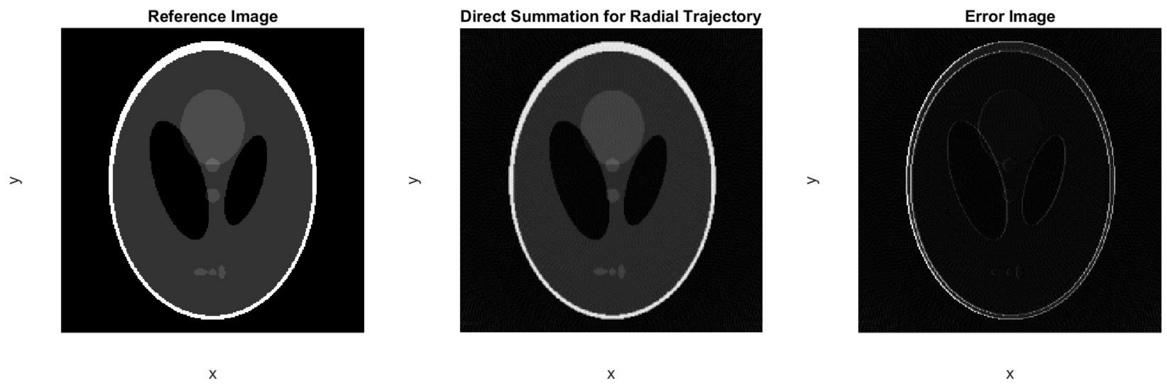

Fig. 31. Direct Summation for Radial Trajectory

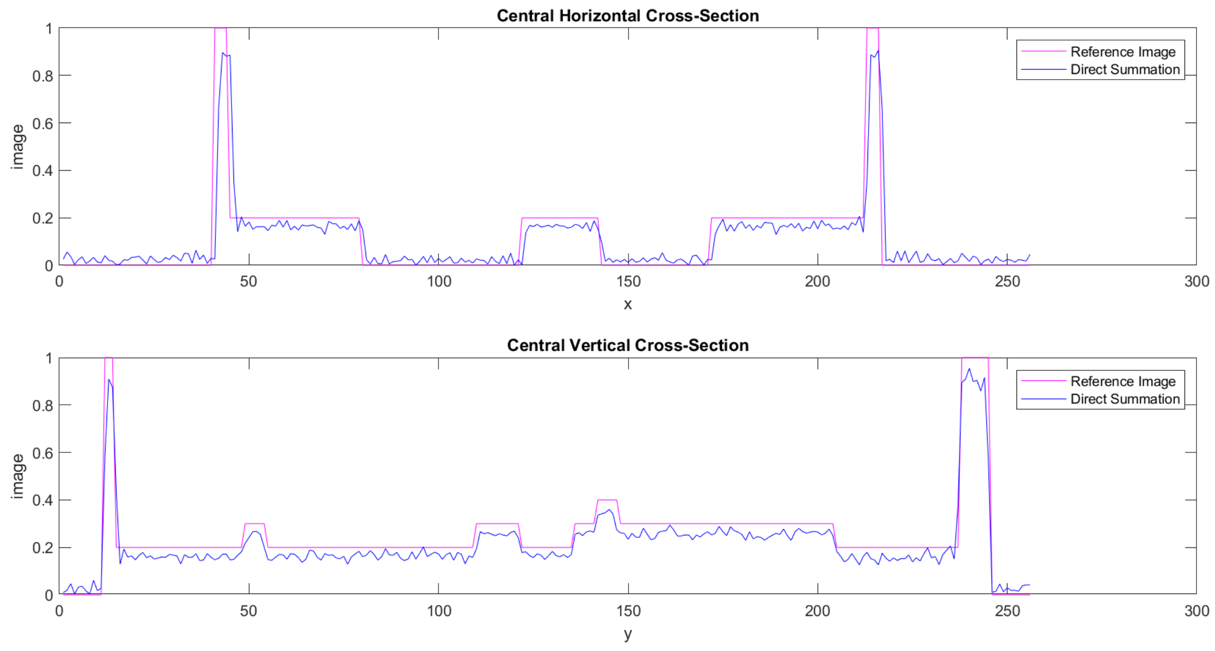

Fig. 32. Central Cross-Sections of Direct Summation for Radial Trajectory

Q2.7 – 1X Gridding Reconstruction

PSNR is 19.8416 and SSIM is 0.36023, which are lower than direct summation. The quality of the image is visually bad with 1X gridding. The aliasing can be detected by human eye. There are parts of replicas inside the region of interest (ROI). The solution for it will be oversampling k-space data in order to separate replica images in image domain.

1. Code

```
%reference image
P = phantom('Modified Shepp-Logan',256);
proj = radon(P,[0:179]);
ref = abs(P);
ref = ref/max(ref(:));

%density compensation filter
[filter,rho] = density_compensate(proj,'opt1');
%projection slice theorem
[ktraj,kdata] = projection_slice(proj);
%filter
w = filter;
%256x256 image
N = 256;
%1X gridding reconstruction
ima_grid = gridkb(kdata,ktraj,w,256,1,2,'image');

%normalize the result image
image = abs(ima_grid);
image = image/max(image(:));
image = flipud(transpose(image));

%error image
error = ref - image;
```

2. Display Results

```
2.7

1X Gridding Reconstruction

PSNR:19.8416

SSIM:0.36023
```

3. Images and Plots

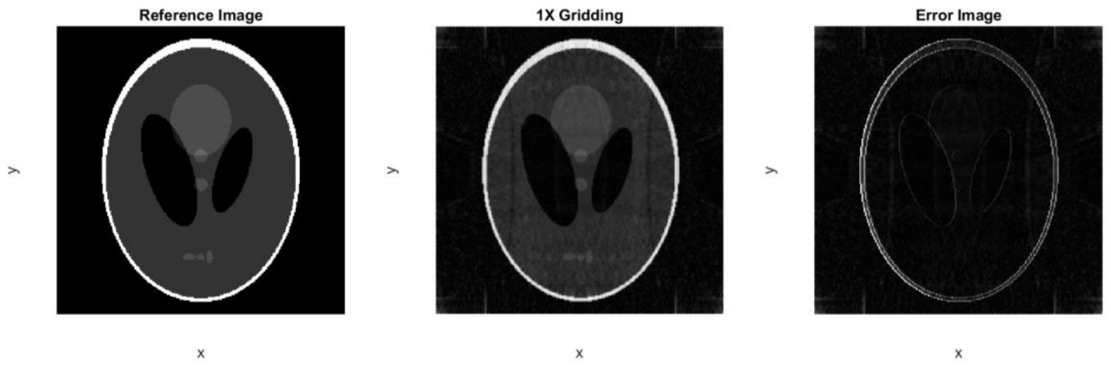

Fig. 33. 1X Gridding

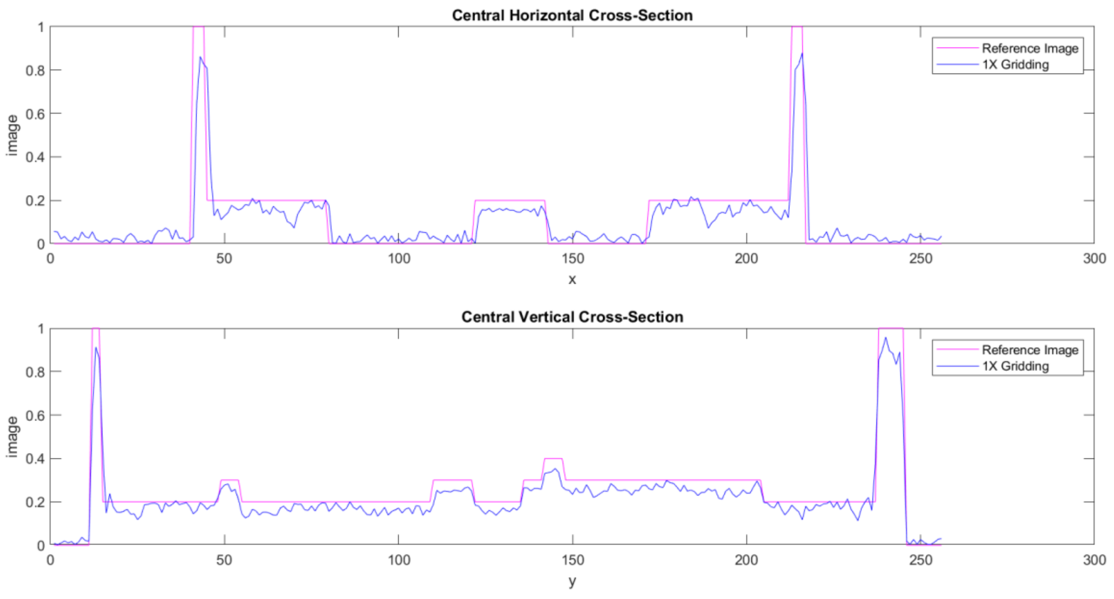

Fig. 34. Central Cross-Sections of 1X Gridding

Q2.8 – 2X Gridding

PSNR was 19.8416 and SSIM was 0.36023 in 1X gridding. Now, PSNR increased to 20.0652 and SSIM increased to 0.40174 with 2X gridding. PSNR and SSIM values are very close to direct summation results. 2X gridding is better than direct summation, because visual results and IQA results are very close to each other, but direct summation is quite slower than 2X gridding.

The aliasing decreased by oversampling, as well. This can be detected visually. We do not have parts of replicas inside FOV now.

1. Code

```
%reference image
P = phantom('Modified Shepp-Logan',256);
proj = radon(P,[0:179]);
ref = abs(P);
ref = ref/max(ref(:));

%density compensation filter
[filter,rho] = density_compensate(proj,'opt1');
%filter
w = filter;
%projection slice theorem
[ktraj,kdata] = projection_slice(proj);

%2X gridding reconstruction
ima_grid = gridkb(kdata,ktraj,w,256,2,4,'image');
%crop the image by taking central 128x128
ima_grid = ima_grid(end/4+1:(3*end/4),end/4+1:(3*end/4));
%normalize the result image
image = abs(ima_grid);
image = image/max(image(:));
image = flipud(transpose(image));

%error image
error = ref - image;
```

2. Display Results

```
2.8

2X Gridding Reconstruction

PSNR: 20.0652

SSIM:0.40174
```

3. Images and Plots

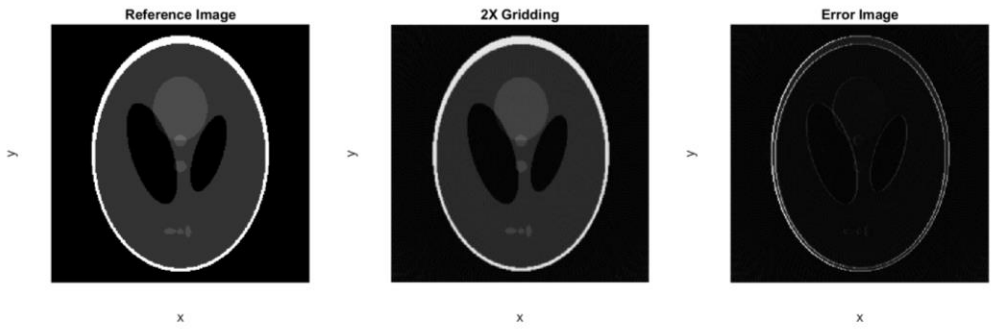

Fig. 35. 2X Gridding

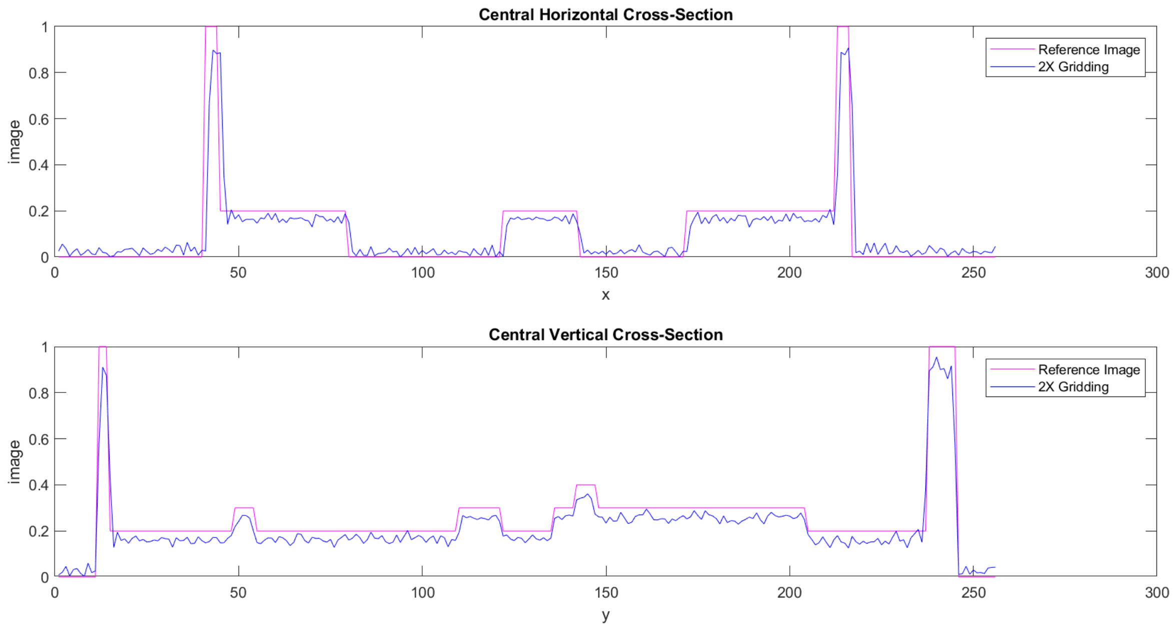

Fig. 36. Central Cross-Sections of 1X Gridding

Table 1. Comparison of IQA Measures of Reconstructions

	Direct Summation	1X Gridding	2X Gridding
PSNR	20.0632	19.8416	20.0652
SSIM	0.40186	0.36023	0.40174

Q3.1 – Displaying 2D K-space Trajectory

k-space data and ktraj are 129x300 matrix. There are 300 radial lines, and 129 data points for one radial line. Radial lines start from the origin unlike Part 2.

1. Code

```
%ktraj:129x300 matrix, kdata:129x300 matrix
load('shepplogan_radial_data.mat');

figure; set(gcf, 'WindowState', 'maximized'); plot(ktraj);
xlabel('k_{x}'); ylabel('k_{y}');
title('2D k-space Trajectory');
saveas(gcf,'3.1.png');
```

2. Plots

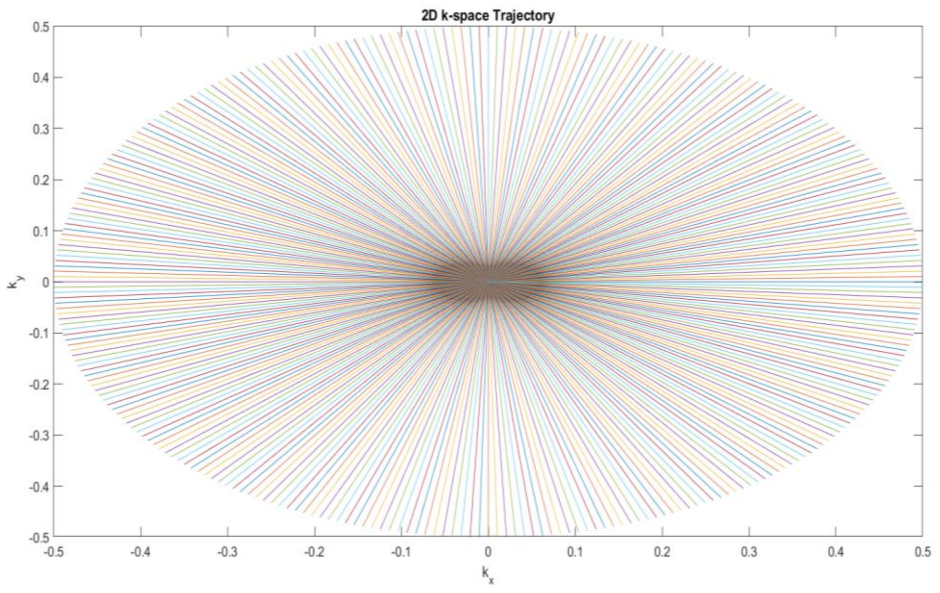

Fig. 37. 2D K-Space Trajectory

Q3.2 Calculating Density Compensation Filter

The density compensate function is the same function in Part 2, the modifications due to data acquisition for radial lines are explained in the function.

1. Code

```
load('shepplogan_radial_data.mat');
%density compensation filter calculated by geometric approach
[filter,rho] = density_compensate(ktraj,'opt2');
filter_first = filter(:,1);

%display the density compensation filter for radial trajectory
figure; set(gcf, 'WindowState', 'maximized');
plot(rho,filter_first);
```

```
xlabel('\rho'); ylabel('Area');
title('Density Compensation Filter for Radial Trajectory');
saveas(gcf,'3.2.png');
```

density compensate function

```
function [filter,rho] = density_compensate(proj,opt)
%
% function [filter,rho] = density_compensate(proj,opt)
%
% input:  proj = g(l,theta)
%         opt =   'opt1', 'opt2'
% output: filter = calculated area for each sample point
%         rho = k-rho vector for given projectory

%'opt1' : radial lines extend from -0.5 to 0.5
%'opt2' : radial extend from the center, i.e. from 0 to 0.5
switch opt

case 'opt1'

[datapoint_no,line_no] = size(proj);

%N is number of lines
N = line_no;

%k is between -0.5 and 0.5
k_interval = 1/(datapoint_no-1);

%each sample is in n^th disk
limit = (datapoint_no - 1)/2;
rho = [-limit:limit];
n = abs(rho);

%calculate area for each sample
area = zeros(1,datapoint_no);
%n^th sample
area = (pi/N)*(k_interval)^2.*n;
%DC disk
area(rho == 0) = (pi/(4*N))*(k_interval/2)^2;

%create filter for each data point
%columns are  equal, because filter is equal each projection
filter = transpose(repmat(area,[line_no 1]));

case 'opt2'

[datapoint_no,line_no] = size(proj);

%we treat lines as if 2 opposite lines are combined to 1 line
N = line_no/2;

%k is between 0 and 0.5
k_interval = 0.5/datapoint_no;

%each sample is in n^th disk
```

```
limit = datapoint_no-1;
rho = [0:limit];
n = rho;

%calculate area for each sample
area = zeros(1,datapoint_no);
%n^th sample
area = (pi/N)*(k_interval)^2.*n;
%DC disk
area(n == 0) = (pi/(4*N))*(k_interval/2)^2;

%create filter for each data point
%columns are  equal, because filter is equal each projection
filter = transpose(repmat(area,[line_no 1]));

end

end
```

2. Plots

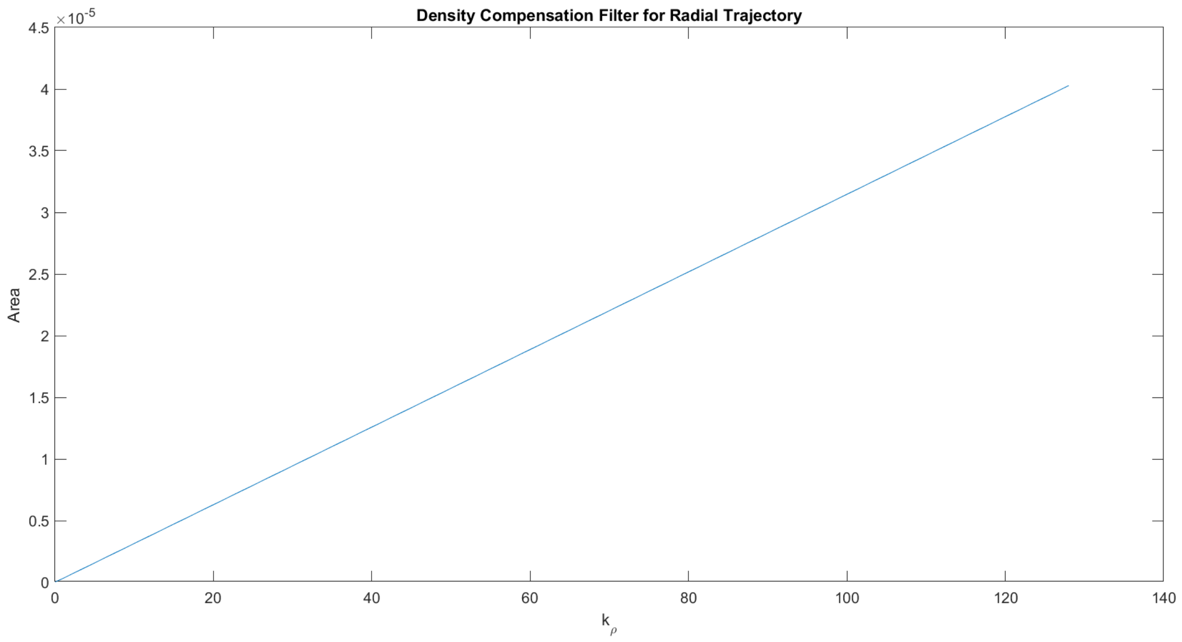

Fig. 38. Density Compensation Filter for Radial Trajectory

Q3.3 – Direct Summation

There is an artifact appearing like a ring outside the ROI. The image looks pretty well even though we lost small amount of contrast.

PSNR is 17.0303 and SSIM is 0.39524. These IQA measure are lower when compared to direct summation in Part 2. I think this is due to the artifact appearing as a ring, and also being undersampled. Our data points in k-space decreased with the new radial trajectory. This undersampling on radial lines may have caused the artifacts to appear closer to ROI.

1. Code

```
%reference image
P = phantom('Modified Shepp-Logan',256);
ref = abs(P);
ref = ref/max(ref(:));

load('shepplogan_radial_data.mat');
%density compensation filter
[filter,rho] = density_compensate(ktraj,'opt2');
%256x256 image
N = 256;
%filter
d = filter;
%direct summation reconstruction
[ima_direct,time] = direct_summation(d,N,ktraj,kdata);

%normalize the result image
image = abs(ima_direct);
image = image/max(image(:));
image = flipud(transpose(image));

%error image
error = ref-image;
```

2. Display Results

```
3.3

Direct Summation for Radial Trajectory

PSNR:17.0303

SSIM:0.39524
```

3. Images and Plots

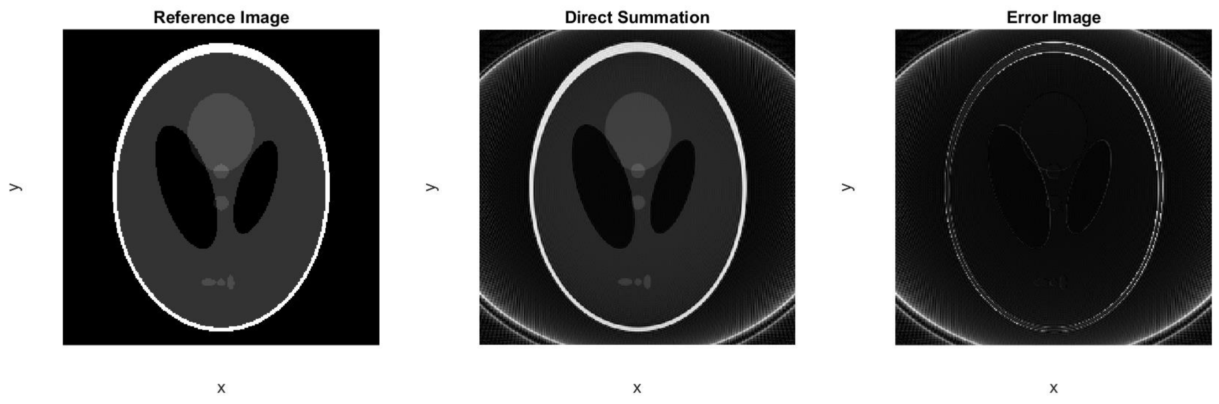

Fig. 39. Direct Summation

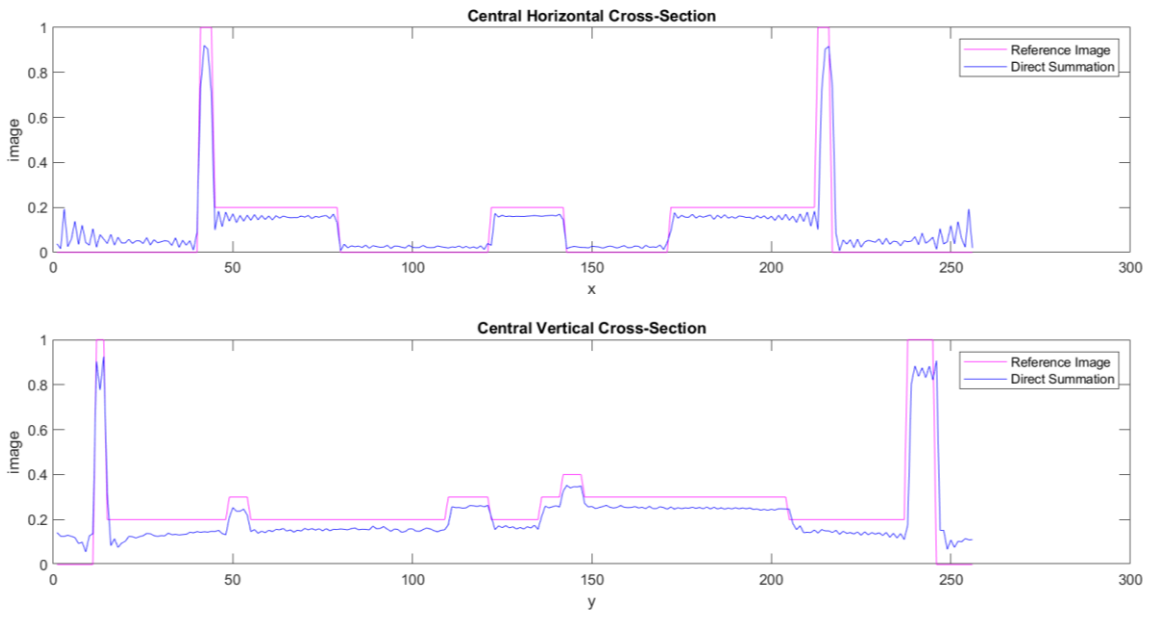

Fig. 40. Central Cross-Sections of Direct Summation

Q3.4 – 1X Gridding Reconstruction

PSNR is 15.608 and SSIM is 0.27012, which are lower than 1X gridding reconstruction. The images are unacceptable and small shapes inside ROI cannot be detected easily. We will try remedying this by 2X gridding again.

1. Code

```
%reference image
P = phantom('Modified Shepp-Logan',256);
ref = abs(P);
ref = ref/max(ref(:));

load('shepplogan_radial_data.mat');
%density compensation filter
[filter,rho] = density_compensate(ktraj,'opt2');
%filter
w = filter;
%1X gridding reconstruction
ima_grid = gridkb(kdata,ktraj,w,256,1,2,'image');

%normalize the result image
image = abs(ima_grid);
image = image/max(image(:));
image = flipud(transpose(image));

%error image
error = ref - image;
```

2. Display Results

```
3.4

1X Gridding Reconstruction

PSNR: 15.608

SSIM: 0.27012
```

3. Images and Results

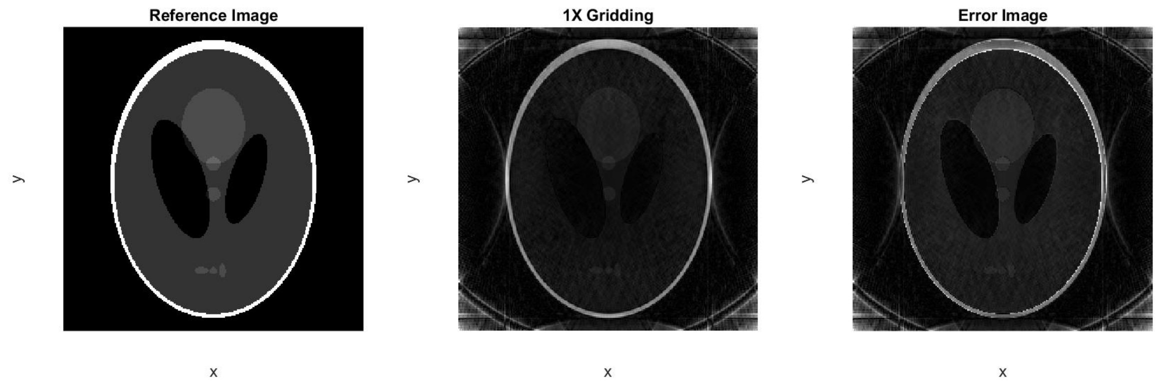

Fig. 41. 1X Gridding Reconstruction

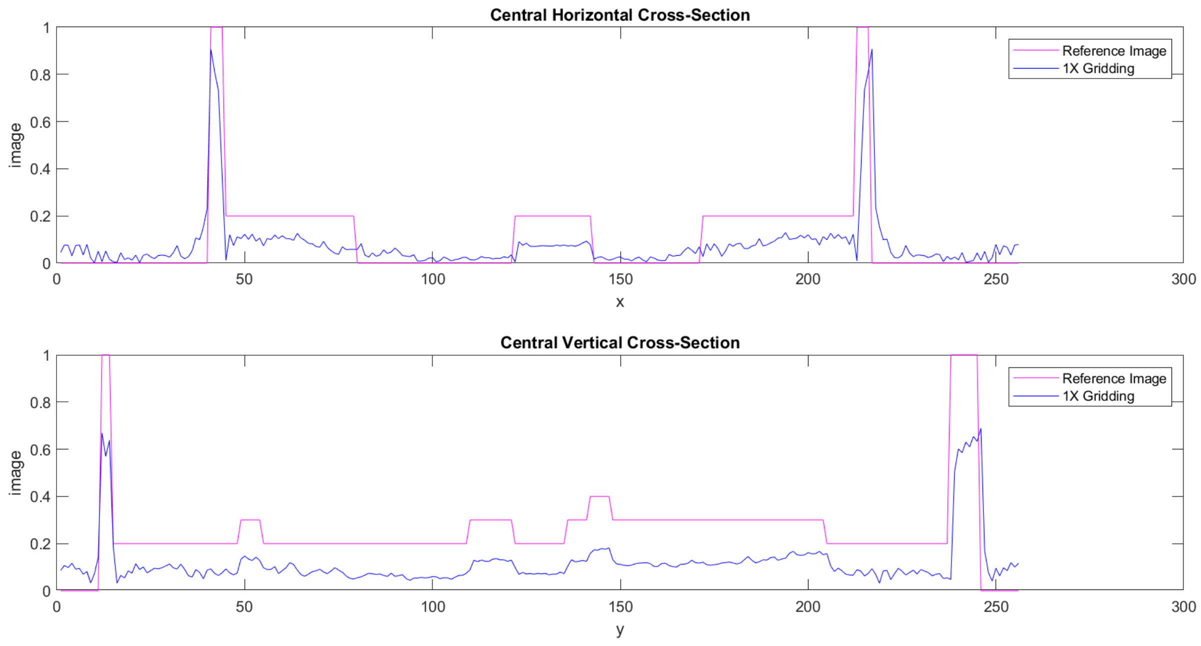

Fig. 42. Central Cross-Sections of 1X Gridding Reconstruction

Q3.5 – 2X Gridding Reconstruction

PSNR is 17.0329 and SSIM is 0.39586. PSNR was 17.0303 and SSIM was 0.39524 in direct summation. 2X gridding is better because of time issue. IQA measures and visual results are better now compared to 1X gridding. IQA measure are lower than Part 2 because of the artifact inside FOV appearing as a ring.

1. Code

```
%reference image
P = phantom('Modified Shepp-Logan',256);
ref = abs(P);
ref = ref/max(ref(:));

load('shepplogan_radial_data.mat');
%density compensation filter
[filter,rho] = density_compensate(ktraj,'opt2');
%filter
w = filter;
%2X gridding reconstruction
ima_grid = gridkb(kdata,ktraj,w,256,2,4,'image');
%crop the image by taking central 128x128
ima_grid = ima_grid(end/4+1:(3*end/4),end/4+1:(3*end/4));
%normalize the result image
image = abs(ima_grid);
image = image/max(image(:));
image = flipud(transpose(image));

%error image
error = ref - image;
```

2. Display Results

```
3.5

2X Gridding Reconstruction

PSNR:17.0329

SSIM:0.39586
```

3. Results

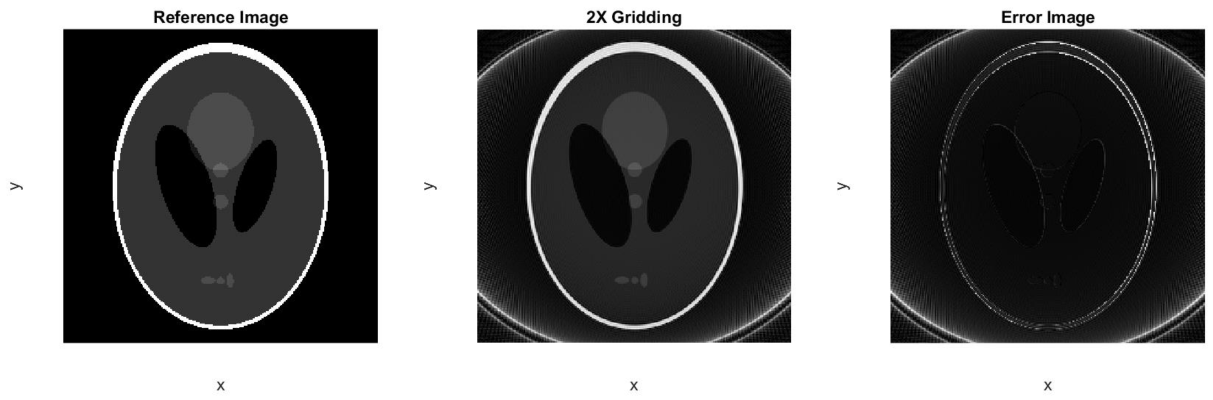

Fig. 43. 2X Gridding Reconstruction

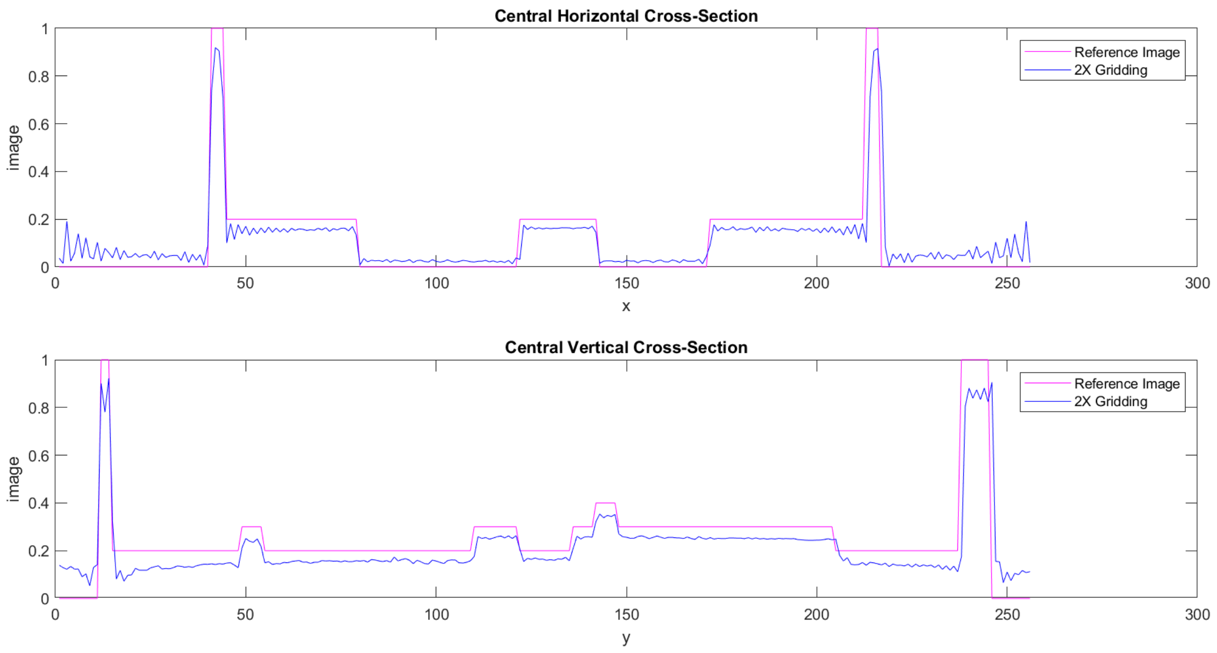

Fig. 44. Central Cross-Sections of 2X Gridding Reconstruction

Table 2. Comparison of IQA Measures of Reconstructions

	Direct Summation	1X Gridding	2X Gridding
PSNR	17.0303	15.608	17.0329
SSIM	0.27012	0.27012	0.39586