3 ANALYSIS

In order to evaluate and compare the performance of our reconstruction method and MATLAB® *iradon* function, the function *evaluation* defines Mean Squared Error (MSE):

$$MSE = \sum_i value_{i,reconstruction} - value_{i,original}$$

a measurement standard for difference between the reconstructed and original image, which sums up the difference of two image matrix and is then divided by number of pixels. Also the time cost of the reconstruction methods is recorded using command *tic toc*.

3.1 Number of Projections

The number of projections used on the data pertains to the number of slices that are processed through the object. As the number of projection slices decrease, the image becomes blurry and contains some artifacts. In comparison, as the number of projection slices increase, the quality of the image becomes sharper and clearer. This can all be illustrated in Figure 4. Figure 3 demonstrates that when number of projections is greater than 70 there is little to no difference between to the two methods which shows that our method is as good as iradon. Also in this range MSE of both methods are relatively stable, meaning further increase in projection numbers won't give us significantly better results. Moreover, as number of projections continues to decrease, our reconstruction method generates much smaller MSE compared with iradon. However, as the number of projections increase, in terms of time cost, iradon is much faster than our reconstruction method and as the number of projections increase the gap between the two

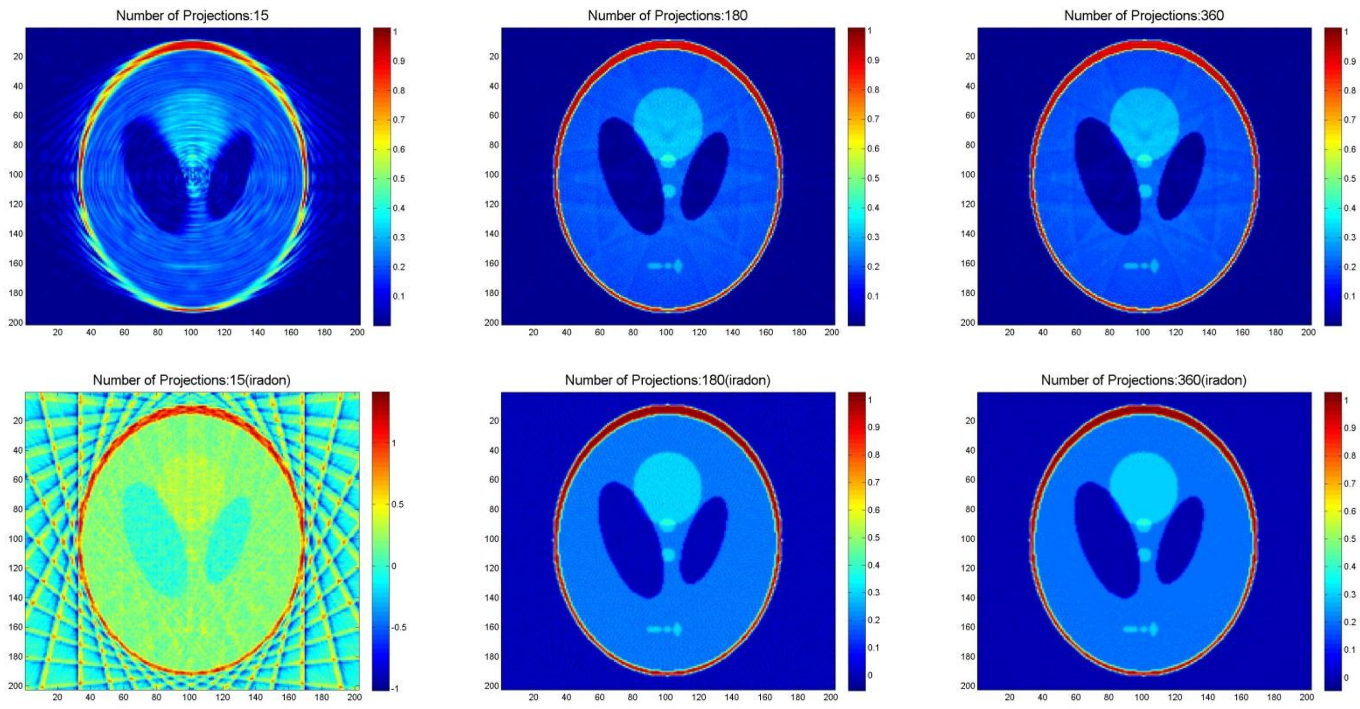

Figure 3: Images of DFR vs. iradon for Different Number of Projections. Top row: DFR, bottom row: iradon. Column: number of projections (from left to right) 15,180,360

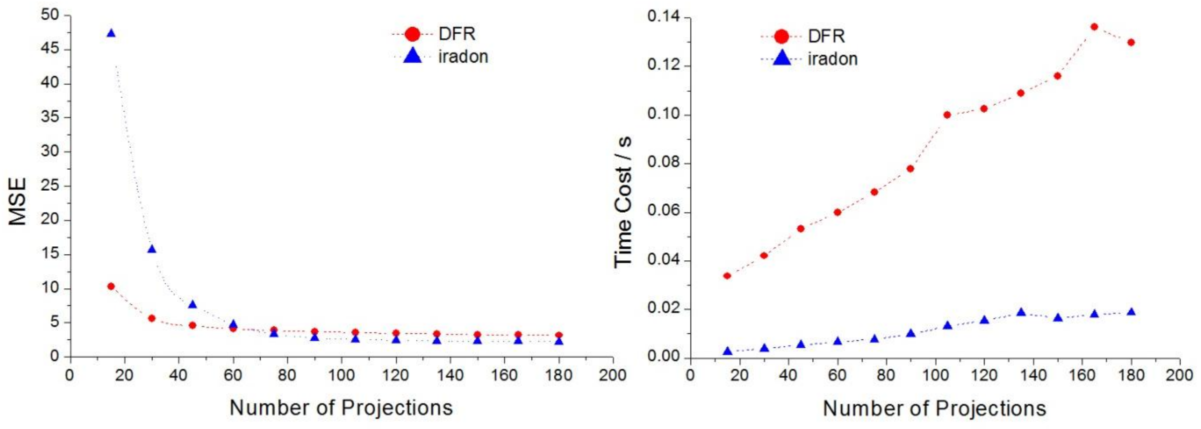

Figure 4: Comparison of Reconstruction Method vs. iradon for Different Number of Projections.

continues to increase. Although our method is accurate in comparison to iradon it is not as efficient since it takes a longer amount of time.

3.2 Image Size

Image size is the size of the matrix that the data is contained in. Here we use number of pixels in one dimension to denote image size, e.g. 201 means image with 201X201 pixels. Decreasing the image size leads to a smaller matrix and a smaller portion of the data that will be analyzed whereas the increasing the image size has the opposite effect. Since MATLAB® built-in *radon* function always set distance between adjacent beam as one pixel, so larger image size here means more X-ray detectors and smaller distance between adjacent X-ray beams as in clinical the head size of patient is constant. Larger image gives more detail and generally higher resolution than the pixelated smaller image (Figure 5). When looking at MSE, our reconstruction method vs. iradon is very close at a small image size. As the image size increases, the gap starts to widen, then come together, widen and come together again with our method being consistently higher. Figure 5 shows that the MSE of the image is independent of the image size. In terms of time cost, our reconstruction method increases exponentially as the image size whereas the iradon is fairly linear and smaller. Again our method is mostly as accurate as iradon but has a longer time cost.

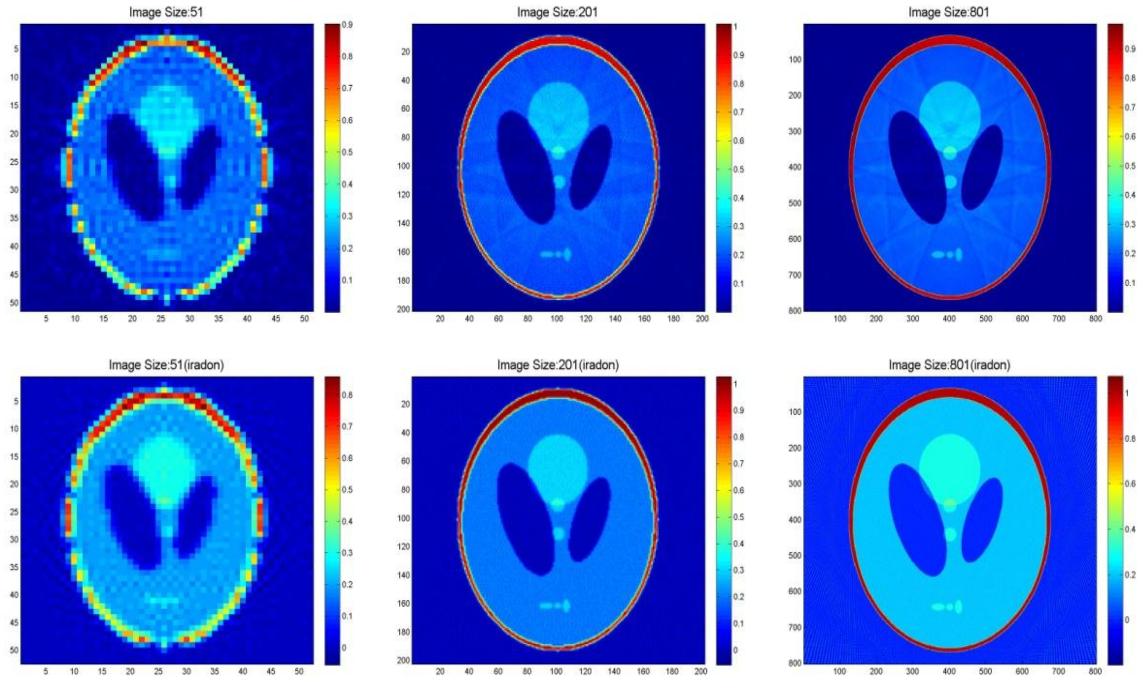

Figure 5: Example Images of DFR vs. iradon for Various Image Sizes. Top row: DFR, bottom row: iradon. Column: image size (from left to right) 51,201,801

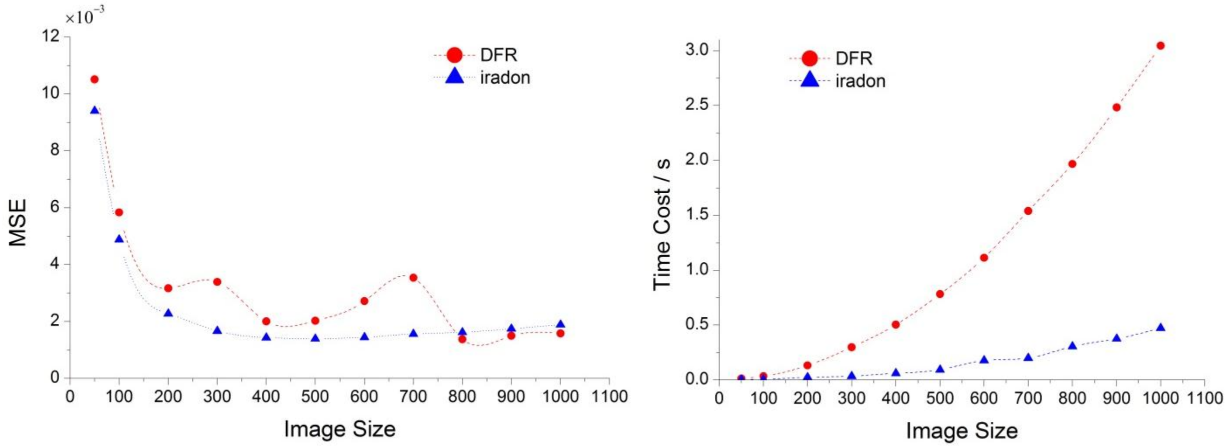

Figure 6: Comparison of Reconstruction Method vs. iradon for Various Image Sizes.

3.3 Different Interpolation Schemes

Interpolation is used to convert points from the radial grid to the Cartesian grid. For this application we compared three different methods: nearest neighbor, linear and cubic. The nearest neighbor method assigns the nearest neighbor value to the Cartesian point of interest. The linear method assigns the value of neighbors with weight. The distance is inversely proportional to the weight so the longer the distance the less weight it has and vice versa. The cubic method provides more curvature in approximation but on the flip side it also needs more points to compute the interpolation. The nearest neighbor method is the worst out of the three and contains many artifacts. In comparison, cubic and nearest neighbor both create clear and sharp images with the cubic method giving the highest contrast. All these characteristics can be seen in Figure 7. Looking at SSE, our reconstruction method is much higher for all three methods than iradon, which indicates that our method isn't comparable with iradon. However, the difference between the two methods is greatest only in nearest neighbor but the two are fairly close for the cubic and linear method. In terms of efficiency our reconstruction method did much worse than iradon for all three schemes (Figure 8). Since iradon algorithm does not use interpolation, MSE and time cost are the same for all three interpolation methods.

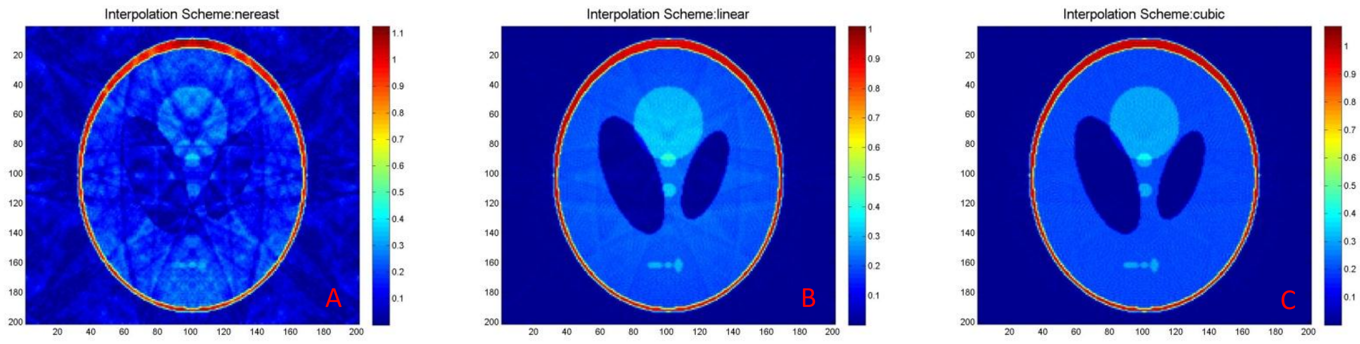
Interpolation Scheme:nereast
Interpolation Scheme:linear
Interpolation Scheme:cubic
A
B
C

Figure 7: Different Interpolation Schemes. A) Nearest. B) Linear. C) Cubic -Method

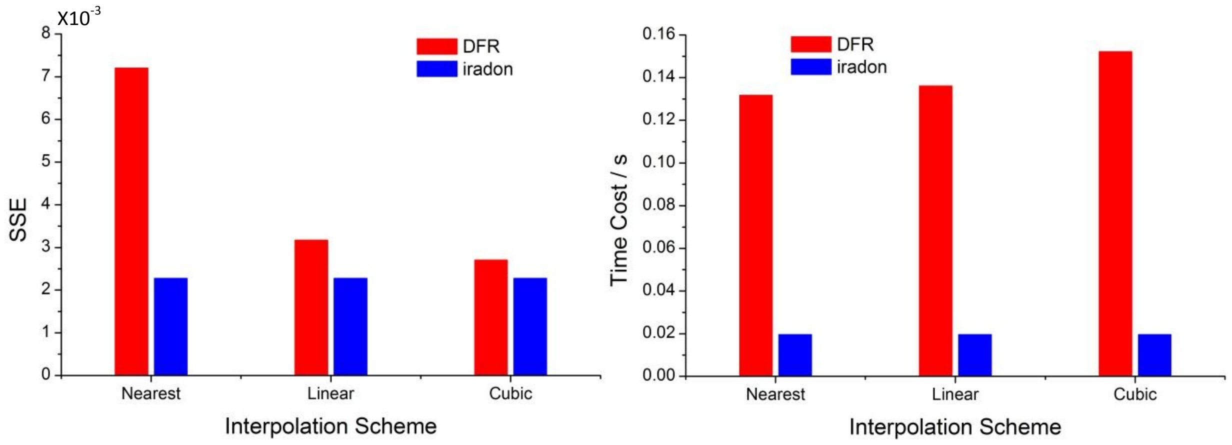

Figure 8: Comparison of Reconstruction Method vs. iradon for Different Interpolation Schemes.

3.4 Noise Tolerance

Random Gaussian noise with mean zero was added right after the radon transform to all the images to observe its effect on the reconstruction. The variance of noise is reflected by signal to noise ratio (SNR) expressed in unit of decibel (dB):

$$SNR = 20 \log_{10} \frac{mean\ signal}{noise\ variance}$$

where mean signal is average of the radon transform matrix elements. When the is no noise, SNR=inf. We should carefully interpret SNR since it varies the opposite way with noise variance, i.e. the lower SNR, the larger noise variance hence noisier source data. Also the relationship is not linear. Figure 9 illustrates that the images become nosier as the SNR decreases. When comparing our reconstruction method vs. iradon, one can see in Figure 10 that they are highly alike in terms of MSE and stable over a long range of SNR. And MSE becomes significantly large when SNR is below 10 dB. Therefore the reconstruction method is relatively noise resistive. In terms of time cost, our method was much higher than iradon but both remained constant because noise does not change the matrix size which mainly influence computation time.

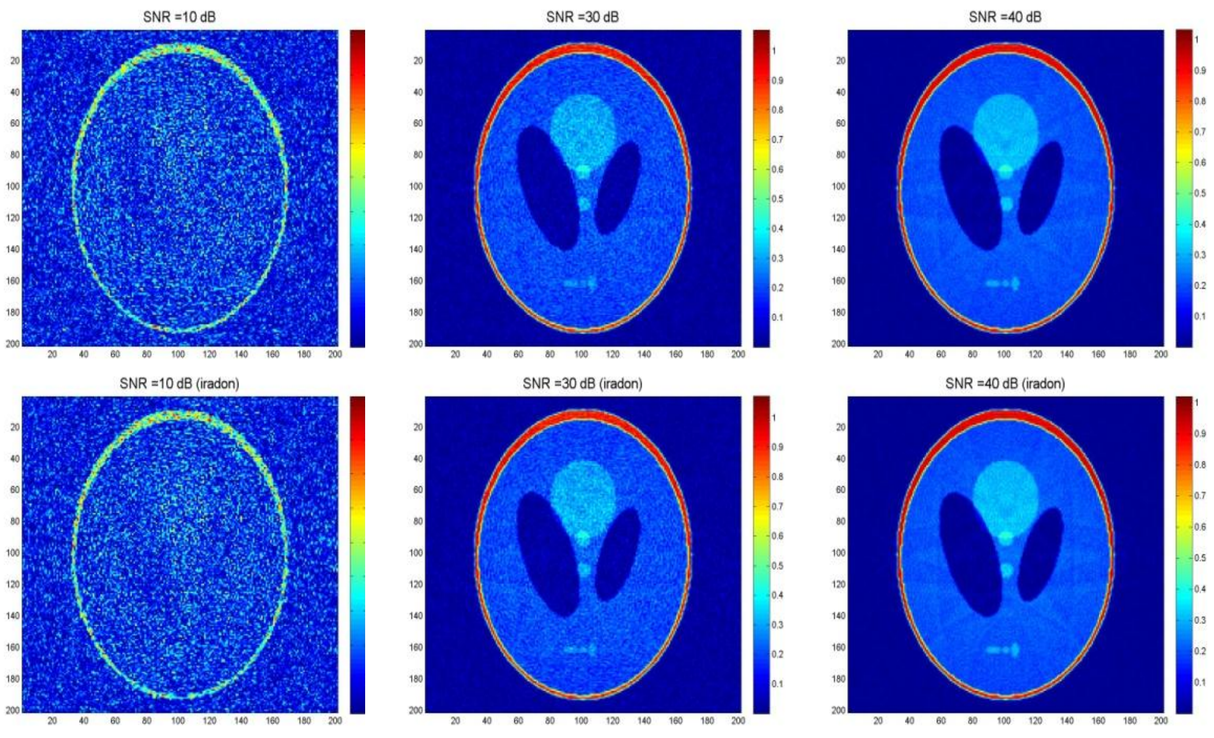

Figure 9: Example Images of DFR vs. iradon for Various Noise Level. Top row: DFR, bottom row: iradon. Column: SNRdB (from left to right) 10,30,40

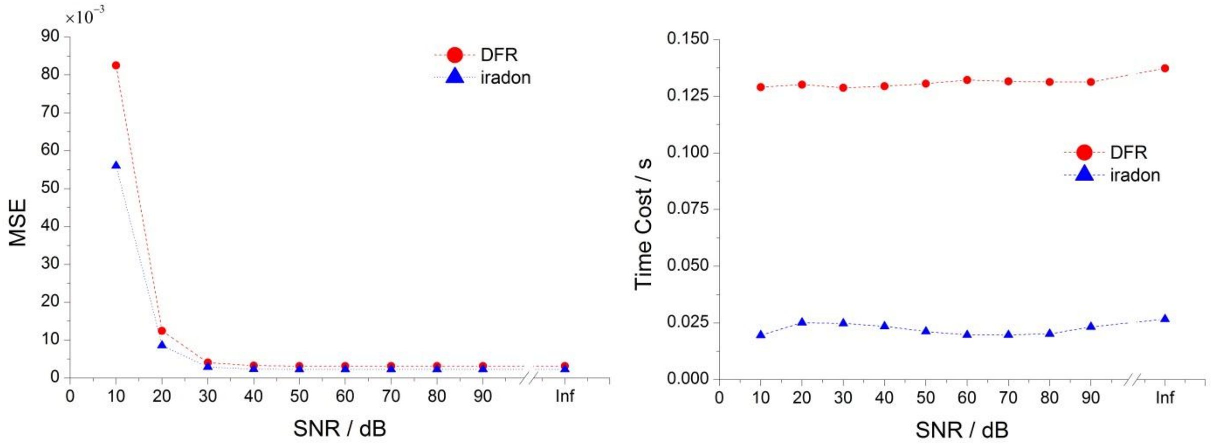

Figure 10: Comparison of Reconstruction Method vs. iradon for Different Noise Levels.

\

3.5 Resolution Limits

There are many ways to test the resolution limits. To quantify the resolution limits of reconstruction methods, we draw two straight lines separated by certain distance Δx, then decrease Δx until the two lines cannot be distinguished and we define this Δx_{min} as the resolution limits. Moreover, various level of noise is added to the image to examine how the resolution limits is resistive to noise. Typical results are shown in Figure 10. When there is no noise, the resolution limit Δx_{min}=1 pixel. As SNR decreases (more noise) down to 0, the resolution limit is still 1 pixel. However, if the SNR is very low, such as 0 dB, the two lines near the edge of the image are not as clearly separated as that in the center. In conclusion, the resolution limits is relatively stable in a relatively large noise level range, and it’s more influenced by location of the two objects of interest (Figure 11). Generally, the center of the image may contain more information; hence the resolution is better than that of the edge.

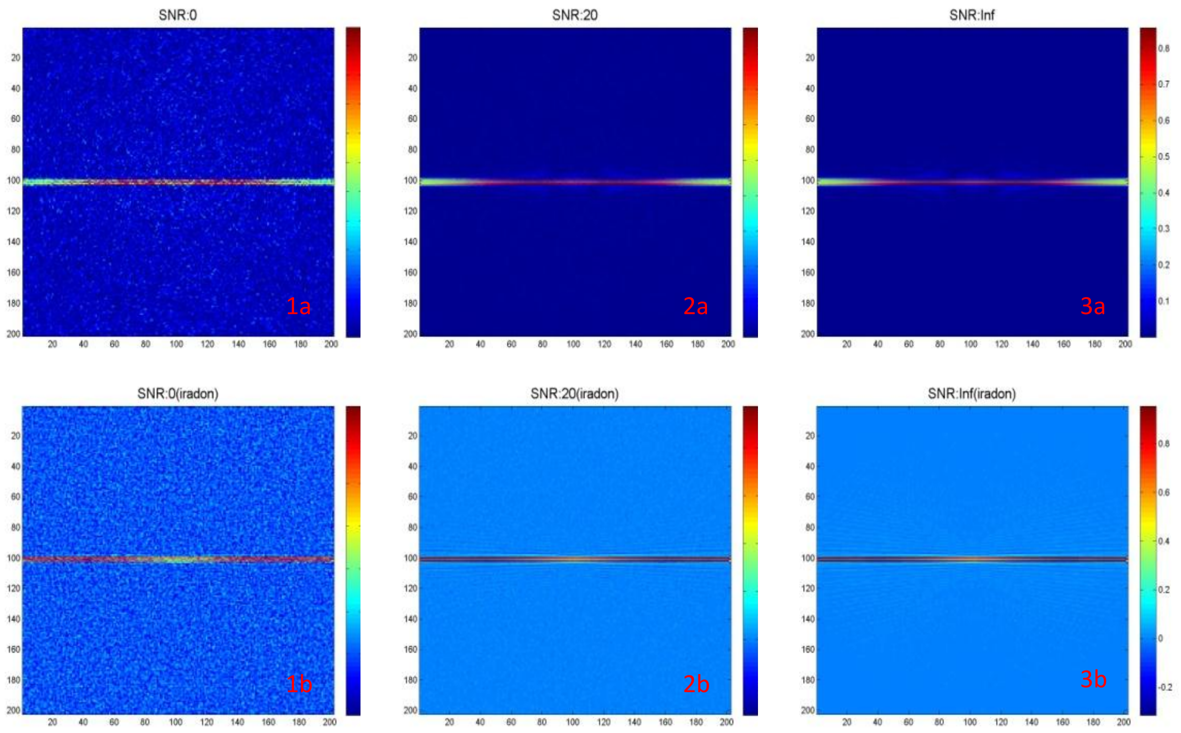

Figure 11: Comparison of Reconstruction Method vs. iradon for Different Resolution Limits at Different SNR's. Top row: DFR, bottom row: iradon. Column: SNRdB (from left to right) 0,20,inf

References

1. Lecture 4 Notes, BMEN4894 class material
2. Image reconstruction in radiology. J. Anthony Parker. CRC Press 1990

Acknowledgement

We thank Professor Andreas H. Hielscher and TA Michael A. Khalil for their help in this project.

APPENDIX MATLAB Code

```
% BMEN E4984 Biomedical Imaging Project
% CT image reconstruction using Direct Fourier Reconstruction
% Siheng He, Natalie Delpratt

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Evaluate the effct of number of projections
num_proj = [45 90 180 360];
SSE_proj = zeros(length(num_proj),2);
time_proj = zeros(length(num_proj),2);
N_image=201;
for i=1:length(num_proj)
    N_theta = num_proj(i);
    tic
    Reconstruct_image =
FourierReconstruction(N_theta,N_image,'linear',inf,0);
    time_proj(i,1) = toc;
    save_result(Reconstruct_image,strcat('Number of Projections:  ',...
        num2str(N_theta)));
    SSE_proj(i,1) = evaluation(N_image,Reconstruct_image);

    d_theta = 180/N_theta;
    theta = linspace(0,180-d_theta,N_theta);
    rad = radon(phantom(N_image),theta);
    tic
    irad = iradon(rad,theta);
    time_proj(i,2) = toc;
    save_result(irad,strcat('Number of Projections:  ',num2str(N_theta),...
        '(iradon)'));
    SSE_proj(i,2) = evaluation(N_image,irad(1:N_image,1:N_image));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Evaluate the effct of image size
N_theta = 180;
image_size = [201 301 401 501];
SSE_size = zeros(length(image_size),2);
time_size = zeros(length(image_size),2);
d_theta = 180/N_theta;
theta = linspace(0,180-d_theta,N_theta);
for i=1:length(image_size)
    % evaluate DFR
    N_image = image_size(i);
    tic
    Reconstruct_image =
FourierReconstruction(N_theta,N_image,'linear',inf,0);
    time_size(i,1) = toc;
    save_result(Reconstruct_image,strcat('Image Size:  ',...
        num2str(N_image)));
    SSE_size(i,1) = evaluation(N_image,Reconstruct_image)/(N_image/201)^2;
    % evaluate iradon
    rad = radon(phantom(N_image),theta);
    tic
    irad = iradon(rad,theta);
    time_size(i,2) = toc;
    save_result(irad,strcat('Image Size:  ',num2str(N_image),'(iradon)'));
    SSE_size(i,2) =
evaluation(N_image,irad(1:N_image,1:N_image))/(N_image/201)^2;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Evaluate the effct of Interpolation Scheme
interp_m = cell(1,4);
interp_m{1}='nereast';
interp_m{2}='linear';
interp_m{3}='spline';
interp_m{4}='cubic';
SSE_interp = zeros(length(interp_m),2);
time_interp = zeros(length(interp_m),2);
N_theta = 180;
theta = 0:179;
N_image = 201;
for i=1:length(interp_m)
    % evaluate DFR
    tic
    Reconstruct_image =
FourierReconstruction(N_theta,N_image,interp_m{i},inf,0);
    time_interp(i,1) = toc;
    save_result(Reconstruct_image,strcat('Interpolation Scheme:  ',...
        interp_m{i}));
    SSE_interp(i,1) = evaluation(N_image,Reconstruct_image);
    % evaluate iradon
    rad = radon(phantom(N_image),theta);
    tic
    irad = iradon(rad,theta);
    time_interp(i,2) = toc;
    save_result(irad,strcat('Interpolation Scheme:
',interp_m{i},'(iradon)'));
    SSE_interp(i,2) = evaluation(N_image,irad(1:N_image,1:N_image));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Evaluate noise tolerance
noise_level = [inf 60 40 20];
SSE_noise = zeros(length(noise_level),2);
time_noise = zeros(length(noise_level),2);
N_image=201;
N_theta=180;
theta=0:179;
for i=1:length(noise_level)
    SNRdB = noise_level(i);
    tic
    Reconstruct_image =
FourierReconstruction(N_theta,N_image,'linear',SNRdB,0);
    time_noise(i,1) = toc;
    save_result(Reconstruct_image,strcat('SNR =  ',num2str(SNRdB),' dB'));
    SSE_noise(i,1) = evaluation(N_image,Reconstruct_image);


    rad = radon(phantom(N_image),theta);
    tic
    rad = add_noise(rad,SNRdB);
    irad = iradon(rad,theta);
    time_noise(i,2) = toc;
    save_result(Reconstruct_image,strcat('SNR =  ',num2str(SNRdB),' dB
(iradon)'));
    SSE_noise(i,2) = evaluation(N_image,irad(1:N_image,1:N_image));
end


%-------------------------------------------------------
% @param N_theta: Number of projections
% @param N_image: Size of image (N_image X N_image pixels)
% @param method: Interpolation method
% @param SNRdB: Signal to Noise ratio in decibel
% @param mode: 1-print all results; 0-print no results
% @retval Reconstructed_image: Matrix of the reconstructed image
function
Reconstructed_image=FourierReconstruction(N_theta,N_image,method,SNRdB,mod
e)

% Source data preparation
d_theta = 180 / N_theta;
theta = linspace(0,180-d_theta,N_theta);

Phantom = phantom(N_image);
axis = linspace(-(N_image-1)/2,(N_image-1)/2,N_image);
Radon = radon(Phantom,theta);          % Radon transform.
if ~isinf(SNRdB)
    Radon = add_noise(Radon,SNRdB);    % add noise to Radon transform data
end
axis_s = linspace(-(size(Radon,1)-1)/2,(size(Radon,1)-1)/2,size(Radon,1));
if mode
    save_result(axis,fliplr(axis),Phantom,'Original Image','x','y');
    save_result(theta,axis_s,Radon,'Radon Transform','\theta','s');
end
Radon_pad = zeropad(Radon);    % zeropadding to improve speed of code

% 1D FOURIER TRANSFORM
[Fourier_radial w_s] = ft1D(Radon_pad);
if mode
    save_result(theta, w_s, log(abs(Fourier_radial)),...
        'Fourier Transform of Radon Space (Absolute Value in Log Scale)'...
        ,'\theta','\omega_s');
end

% INTERPOLATION
[Fourier_2D w_xy] = pol_cart(theta,w_s,Fourier_radial,N_image,method);
if mode
    save_result(w_xy,fliplr(w_xy),log(abs(Fourier_2D)),...
    'Interpolated Fourier Space From Raidal Grid to Cartesian Grid (Log
Scale)'...
    ,'\omega_x','\omega_y')
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% INVERSE 2D FOURIER TRANSFORM
Reconstructed_image = ift2D(Fourier_2D);
Reconstructed_image = abs(Reconstructed_image);
if mode
    save_result(axis,fliplr(axis),abs(Reconstructed_image),'Reconstructed
Image'...
    ,'x','y');
end


%-------------------------------------------------------
function NoisyRadon = add_noise(Radon,SNRdB)
signal = mean(mean(Radon));
SNR = 10^(SNRdB/20);
sigma = signal/SNR;     % noise variance
noise = randn(size(Radon))*sigma;   % random Gaussian noise
NoisyRadon = Radon+noise;


%-------------------------------------------------------
% Make the dimesion to power of 2 to improve FFT algorithm efficiency
function Radon_pad = zeropad(Radon)
[size_s size_theta] = size(Radon);
next_power_of_2 = pow2(nextpow2(size_s));
% Shift the DC to the left
shifted_Radon = ifftshift(Radon,1);
% Estimate the size of zeropad required
size_zeropad = next_power_of_2 - size_s;
zeropad = zeros(size_zeropad,size_theta);
if(mod(size_s,2)) % if length is an odd number, the 'middle' is between
(size_s + 1)/2 and (size_s + 1)/2+1
        mid_position = (size_s + 1)/2;
else    % if length is an even number, the 'middle' is between size_s/2
and size_s/2+1
        mid_position = size_s / 2;
end
% Add zeros to the middle of the shifted signal
shifted_Radon2 =
vertcat(shifted_Radon(1:mid_position,:),zeropad,shifted_Radon((mid_positio
n+1):size_s,:));
% Shift the DC back to the centre
Radon_pad = fftshift(shifted_Radon2,1);


%-------------------------------------------------------
% 1D Fourier Transform of Radon Transform
function [Fourier_Radon w_s] = ft1D(Radon)
```

```
% Apply 1D FT to each column of the Radon transform data matrix
Fourier_Radon = fft(ifftshift(Radon,1));
[size_omega_s size_theta] = size(Fourier_Radon);
ds=1;
d_omega = pi/((size_omega_s-1)/2*ds);
% define the frequency array
w_s = linspace(-size_omega_s/2,size_omega_s/2,size_omega_s+1)* d_omega;
% Shift the DC back to centre
Fourier_Radon = fftshift(Fourier_Radon,1);
% add value corresponding to the w_s_max
Fourier_Radon = vertcat(Fourier_Radon,conj(Fourier_Radon(1,:)));


%-------------------------------------------------------
% Interpolate from radial grid to rectangular grid, preparing data for
% inverse 2D Fourier Transform
function [Fourier_2D axis_omega_xy] =
pol_cart(theta,omega_s,Fourier_Radon,N_image,interp_m)
% Check correctness of input data
[size_omega_s size_theta] = size(Fourier_Radon);
length_theta = length(theta);
length_omega_s = length(omega_s);

if(length_theta ~= size_theta)
 error('size of theta does not match with the size of Fourier_Radon!')
elseif(length_omega_s ~= size_omega_s)
 error('size of omega_s does not match with the size of Fourier_Radon!')
end

% Label each elements in the matrix Fourier_Radon with the corresponding
theta and omega_s:
theta=[theta 180];
% Add flipped 0 degree data as 180 degree data
Fourier_Radon=horzcat(Fourier_Radon,flipud(Fourier_Radon(:,1)));


% Define the desired scale of the rectangular coordinates
x = linspace(-(N_image-1)/2,(N_image-1)/2,N_image);
d_omega = 2*pi/(N_image-1);
omega_x = x * d_omega;
omega_y= fliplr(omega_x);

% Label each (omega_x, omega_y) to (omega_s, theta)
[OMEGA_X OMEGA_Y] = meshgrid(omega_x, omega_y);
[THETA_I OMEGA_SI] = cart2pol(OMEGA_X,OMEGA_Y);

sign_theta=sign(THETA_I);
ind=find(sign_theta==0);
sign_theta(ind)=1;
OMEGA_SI = OMEGA_SI .* sign_theta;
THETA_I = mod( THETA_I * (180/pi), 180);

% interpolation
```

```
Fourier_2D =
interp2(theta,omega_s,Fourier_Radon,THETA_I,OMEGA_SI,interp_m);
Fourier_2D(find(isnan(Fourier_2D)))=0;
if mod(size(Fourier_2D,1),2)
    mid=(size(Fourier_2D,1)+1)/2;
    Fourier_2D(mid,1:mid-1) = fliplr(conj(Fourier_2D(mid,(mid+1):end)));
end
axis_omega_xy = omega_x;


%-------------------------------------------------------
% Two dimensional inverse Fourier transform
function Final_image = ift2D(Fourier_2D)
Final_image = fftshift(ifft2(ifftshift(Fourier_2D)));


%-------------------------------------------------------
function save_result(varargin)
if nargin == 6
    axis_x = varargin{1};
    axis_y = varargin{2};
    result = varargin{3};
    Title = varargin{4};
    Xlabel = varargin{5};
    Ylabel = varargin{6};
    figure
    imagesc(axis_x,axis_y,result)
    axis xy
    colorbar
    title(Title,'fontsize',16)
    xlabel(Xlabel,'fontsize',14)
    ylabel(Ylabel,'fontsize',14)
    if length(axis_x)==length(axis_y)
        axis square
    end
    print('-djpeg',strcat(Title,'.jpg'))
elseif nargin == 2
    result = varargin{1};
    Title = varargin{2};
    figure
    imagesc(result)
    colorbar
    title(Title,'fontsize',16)
    print('-djpeg',strrep(strcat(Title,'.jpg'),':','_'))
else
    error('Inappropriate number of arguments');
end


%-------------------------------------------------------
% Compare the reconstructed image with original phantom image
function SSE = evaluation(N_image,Reconstructed_image)

result = abs(Reconstructed_image);
```

```
original = phantom(N_image);
dif = result - original;
figure;imagesc(dif);colormap(jet);colorbar
SSE=sum(sum(dif.^2));


%----------------------------------------------------------
function [Reconstructed_image noise_radon] =
resolution(N_theta,N_image,method,dx,SNRdB)

% Source data preparation
d_theta = 180 / N_theta;
theta = linspace(0,180-d_theta,N_theta);

line2 = zeros(N_image);
mid = (N_image-1)/2;
line2(mid,:)=1;
line2(mid+dx,:)=1;
Radon = radon(line2,theta);          % Radon transform.
if ~isinf(SNRdB)
    Radon = add_noise(Radon,SNRdB);    % add noise to Radon transform data
end
noise_radon = Radon;
Radon_pad = zeropad(Radon);    % zeropadding to improve speed of code

% 1D FOURIER TRANSFORM
[Fourier_radial w_s] = ft1D(Radon_pad);

% INTERPOLATION
[Fourier_2D w_xy] = pol_cart(theta,w_s,Fourier_radial,N_image,method);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% INVERSE 2D FOURIER TRANSFORM
Reconstructed_image = ift2D(Fourier_2D);
Reconstructed_image = abs(Reconstructed_image);
```

References:

1.) http://www.mssl.ucl.ac.uk/www_solar/moses/moses-web/Pages/fourier-slice-theorem.jpg