Methods

Based on the algorithm above, the MATLAB® code is mainly comprised of four portions:

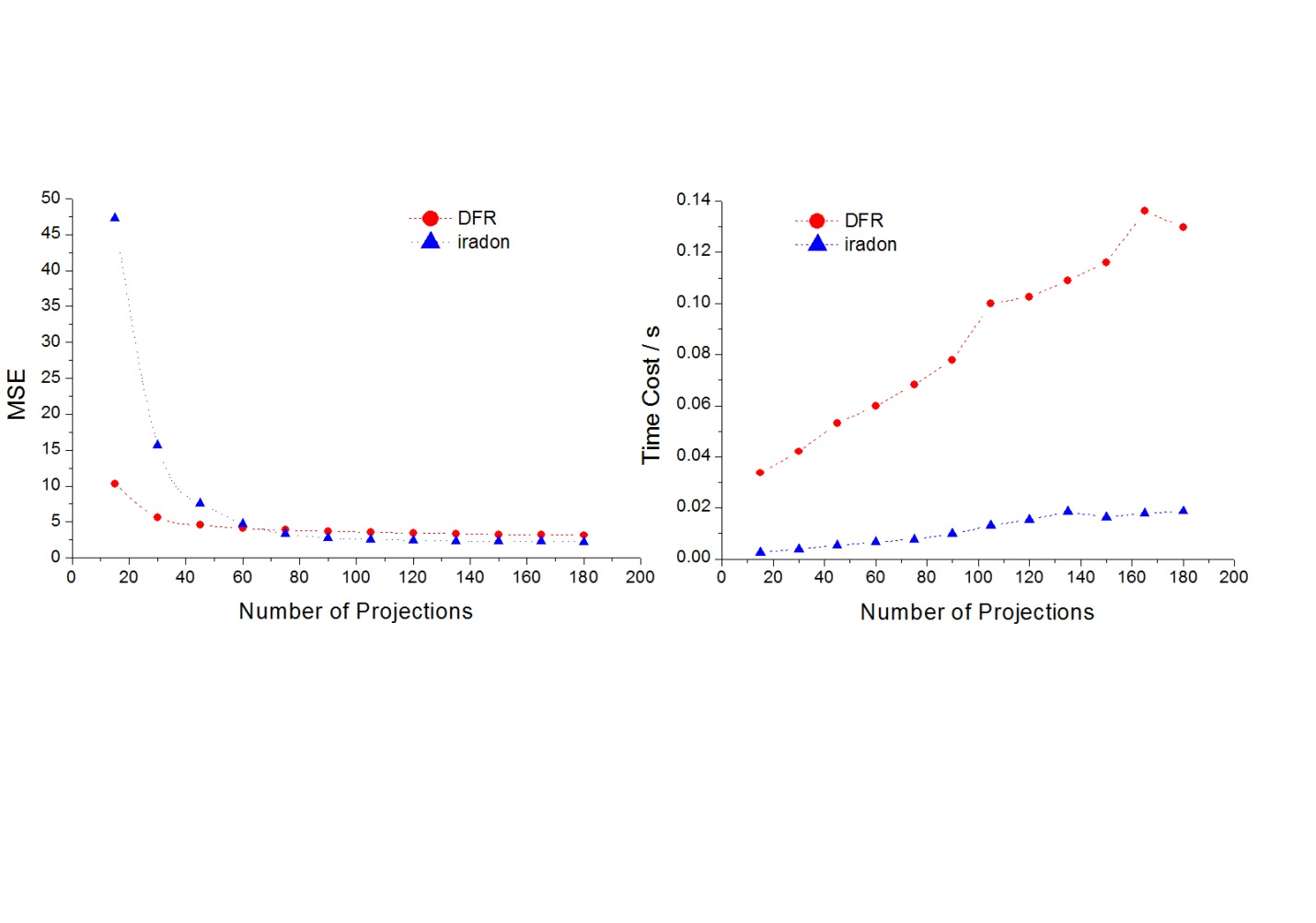
1. Source data preparation. First a head phantom image is made using function *Phantom= phantom (def,n)*, then the CT scan process is simulated by the built-in Radon transform function *Radon=radon(Phantom,theta)*. The result (Figure 1) is our source data for image reconstruction.
2. Since each column of Radon matrix is the projection at certain angle θ, we apply 1D Fourier transform to each column, i.e. fft across the dimension 1, shift the zero-frequency component to the center and get 2D Fourier space arranged in radial grid, using the user-defined function *Fourier\_radial=ft1D(Radon\_pad)* (Figure 2). Each column of matrix *Fourier\_radial* is a central slice in 2D Fourier space at certain angle θ. Prior to 1D FT, the *Radon* matrix is zeropadded by function *zeropad* to make the length of the column next power of 2 to improve FFT efficiency. For instance, if the original length of the column is 287, it would be elongated length of 512 by adding zeros to both sides of the column array.
3. A rectangular grid of the same size as original image matrix is defined in 2D Fourier space, and then converted to polar coordinates using function *cart2pol*. The value in the matrix *Fourier\_2D* (Figure 3) is obtained by interpolating the matrix *Fourier\_radial*, using function *interp2*.
4. The zero-frequency component of the 2D Fourier matrix, *Fourier\_2D*, is first shifted to upper left corner by function *ifftshift*, then undergoes 2D inverse Fourier transform, and finally shifted back by function *fftshift*. Generally the result may contain complex value, so absolute value of the matrix is taken to be the reconstructed image matrix and plotted.

|  |  |
| --- | --- |
| Figure 1 Radon transform | Figure 2 1D Fourier transform |
| Figure 3 Interpolated 2D Fourier space |  |

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), 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*.

Number of Projections Used



|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

Image Size

Different Interpolation Schemes

Noise Tolerance

Resolution Limits

% 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,mode)

% 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\_position+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);