EE 610 Image Processing Project 2

Swrangsar Basumatary Roll 09d07040

Implementing the ‘A’ matrix

The plots the initial image estimates (the data) and the final image estimates(the output after carrying out the TV constraint and so on) are as follows:

Initial image estimate using 8 spokes

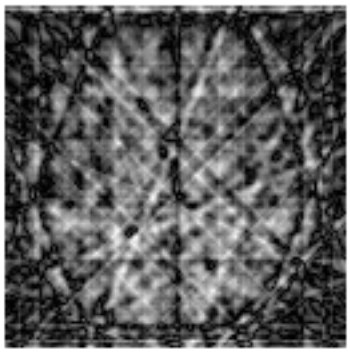
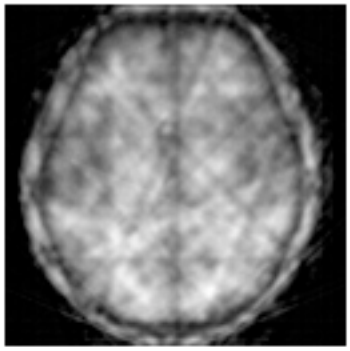


Image estimate using 8 spokes



Initial image estimate using 16 spokes

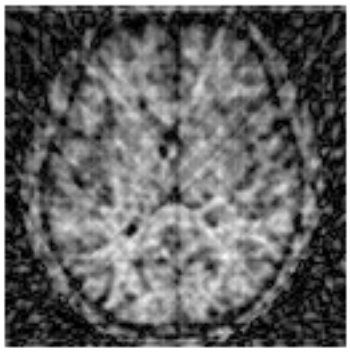
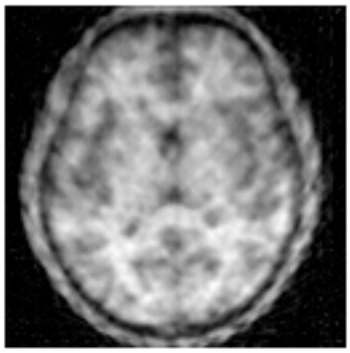


Image estimate using 16 spokes



Initial image estimate using 32 spokes

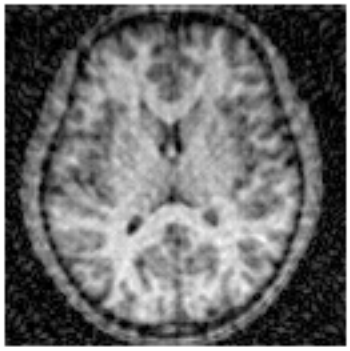
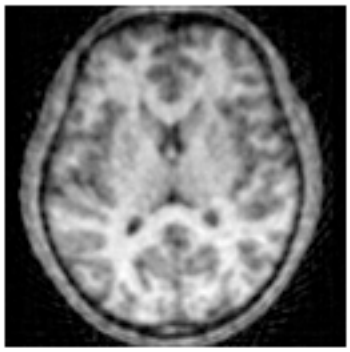


Image estimate using 32 spokes



Initial image estimate using 64 spokes

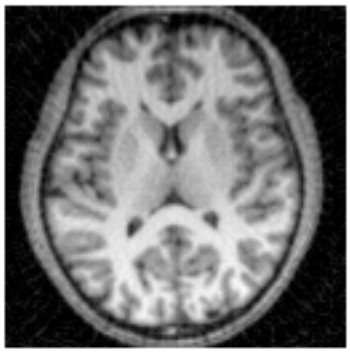
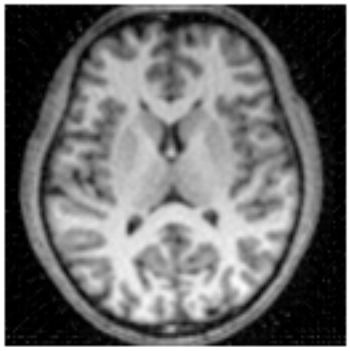


Image estimate using 64 spokes



The functions:

```
function [x, repetitionCounter] = fnlCg(x0,numberOfSpokes,data,
param)
%-------------------------------------------------------------------
--------
%-------------------------------------------------------------------
----------
% the nonlinear conjugate gradient method
disp('running fnlcg');

x = x0;
```

```
% line search parameters - Dont touch..leave alone
maxlsiter = 150;
gradToll = 1.0000e-030;
alpha = 0.0100;
beta = 0.6000;
t0 = 1;
Itnlim = 16;

%%%%%%%%%%
repetitionCounter = 0;
previousObjective = 0;
%%%%%%%%%%


k = 0;

% copmute g0  = grad(Phi(x))
g0 = wGradient(x,numberOfSpokes,data, param);

dx = -g0;

% iterations
while(1)

% backtracking line-search

    % pre-calculate values, such that it would be cheap to
compute the objective
    % many times for efficient line-search
    f0 = objective(x,dx, 0, numberOfSpokes,data, param);
    t = t0;

        [f1]  =  objective(x,dx, t,numberOfSpokes,data, param);

    lsiter = 0;

    while (f1 > f0 - alpha*t*abs(g0(:)'*dx(:)))^2 &
(lsiter<maxlsiter)
        lsiter = lsiter + 1;
        t = t * beta;
        [f1]  =  objective(x,dx, t,numberOfSpokes,data, param);
    end

    if lsiter == maxlsiter
        disp('Reached max line search,.... not so good... might
have a bug in operators. exiting... ');
```

```
        return;
    end

    % control the number of line searches by adapting the
initial step search
    if lsiter > 2
        t0 = t0 * beta;
    end

    if lsiter<1
        t0 = t0 / beta;
    end

    x = (x + t*dx);

    %--------- uncomment for debug purposes
------------------------
    disp(sprintf('%d   , obj: %f ', k,f1));

%-------------------------------------------------------------

    %conjugate gradient calculation- Dont touch

    g1 = wGradient(x,numberOfSpokes,data, param);
    bk = g1(:)'*g1(:)/(g0(:)'*g0(:)+eps);
    g0 = g1;
    dx =  - g1 + bk* dx;
    k = k + 1;

    %TODO: need to "think" of a "better" stopping criteria ;-)

    if f1 == previousObjective
        repetitionCounter = repetitionCounter + 1;
    else
        repetitionCounter = 0;
    end
    previousObjective = f1;

    if (k > Itnlim) | (norm(dx(:)) < gradToll)
        break;
    end
    if repetitionCounter > 7
        break;
    end

end
```

```
repetitionCounter
return;

end


%% the objective function

function [res] = objective(x,dx,t,numberOfSpokes,data, param)
%DEFINE obj
x = x + (t * dx);
b = data;
Ax = getDataMatrix(x, numberOfSpokes);
obj = (Ax - b);
res= (obj(:)'*obj(:)) + (param.TVWeight * getTotalVariation(x))
+ ...
    (param.FOVWeight * fov(x)) + (param.POSWeight *
getPosResidual(x)) + ...
    (param.LaplacianWeight * getLaplacianResidual(x));

end


%% the grad function

function grad = wGradient(x,numberOfSpokes,data, param)

%Define this function
gradObj=gOBJ(x,numberOfSpokes,data);
grad = (gradObj) + (param.TVWeight * gradTotalVariation(x))
+ ...
    (param.FOVWeight * gradFOV(x)) + (param.POSWeight *
getPosGradient(x)) ...
    + (param.LaplacianWeight * getLaplacianGradient(x));

end

%% calculating the gradient of the Objective

function gradObj = gOBJ(x,numberOfSpokes,data)

% computes the gradient of the data consistency
b = data;
inputSize = size(x, 1);
```

```
Ax = getDataMatrix(x, numberOfSpokes);
AhAx = getImageMatrix(Ax, numberOfSpokes, inputSize);
Ahb = getImageMatrix(b, numberOfSpokes, inputSize);
gradObj = 2 * (AhAx - Ahb);

end

%% the function implementing system matrix A

function dataMatrix = getDataMatrix(imageMatrix, numberOfSpokes)

theta = 0:numberOfSpokes-1;
theta = theta .* (180/numberOfSpokes);
dataMatrix = radon(imageMatrix, theta);
dataMatrix = fft(dataMatrix, [], 1); % column fft of the matrix

end

%% function implementing the adjoint system matrix A*

function imageMatrix = getImageMatrix(dataMatrix,
numberOfSpokes, inputSize)

theta = 0:numberOfSpokes-1;
theta = theta .* (180/numberOfSpokes);
imageMatrix = ifft(dataMatrix, [], 1);
imageMatrix = iradon(imageMatrix, theta, 'linear', 'Ram-Lak', 1,
inputSize);

end


%% the total variation penalty function

function totalVariation = getTotalVariation(imageMatrix)

variationX = filter2([1 -1 0], imageMatrix);
variationY = filter2([1; -1; 0], imageMatrix);
mag = sqrt( (abs(variationX) .^ 2) + abs((variationY) .^ 2));
totalVariation = sum(mag(:));

end


%% gradient of the total variation
```

```
function gradTV = gradTotalVariation(imageMatrix)

gradTV=filter2([0 -1 1],filter2([1 -1 0], imageMatrix))
+filter2([0;-1;1],filter2([1; -1; 0], imageMatrix));

end


%% the FOV Mask function

function fovMask = fovMask(inputMatrix)

[rows, cols] = size(inputMatrix);
xRadius = ceil(rows/2);
yRadius = ceil(cols/2);
[X, Y] = meshgrid(-(xRadius):(rows-(xRadius+1)), -yRadius:(cols-
(yRadius + 1)));
a = 4;
b = 5;
term1 = (X./a).^2;
term2 = (Y./b).^2;
radius = sqrt(term1 + term2);
normalizer = sqrt((xRadius/a)^2 + (0/b)^2);
normalizedRadius = radius/normalizer;
mask = normalizedRadius < 0.93;

inputMatrix(mask) = 0;
fovMask = inputMatrix;

end


%% the FOV penalty function

function fov = fov(x)

x = fovMask(x);
mag = abs(x) .^ 2;
fov = sum(mag(:));

end


%% the gradient of the FOV function

function gradFOV = gradFOV(x)
```

```
x = fovMask(x);
gradFOV = 2 .* x;

end

%% the POS function

function posResidual = getPosResidual(x)

mask = x < 0;
posMatrix = (x .* mask) .^ 2;
posResidual = sum(posMatrix(:));

end

%% gradient of the POS function

function posGradient = getPosGradient(x)

mask = x < 0;
posGradient = 2 * (x .* mask);

end

%% the laplacian function

function laplacianResidual = getLaplacianResidual(imageMatrix)

laplacianX = filter2([1 -2 1], imageMatrix);
laplacianY = filter2([1; -2; 1], imageMatrix);
mag = sqrt( (abs(laplacianX) .^ 2) + abs((laplacianY) .^ 2));
laplacianResidual = sum(mag(:));

end

%% the gradient of the laplacian

function laplacianGradient = getLaplacianGradient(imageMatrix)

laplacianGradient = filter2([0 -1 1],filter2([1 -2 1],
imageMatrix))+filter2([0;-1;1],filter2([1; -2; 1],
imageMatrix));

end
```

The script:

```
close all; clear all;

addpath /Users/swrangsarbasumatary/Desktop/
imageProcessingProject2/

load sim_8ch_data

disp('data loaded');

inputImage = anatomy_orig;

numberOfSpokes = 8;

% generating the data vector


theta = 0:numberOfSpokes-1;
theta = theta .* (180/numberOfSpokes);
dataMatrix = radon(inputImage, theta);
dataMatrix = fft(dataMatrix, [], 1);
imageMatrix = ifft(dataMatrix, [], 1);
imageMatrix = iradon(imageMatrix, theta, 'linear', 'Ram-Lak', 1,
size(inputImage, 1));


%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Reconstruction Parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

param.TVWeight = 0.0001;    % Weight for TV penalty
param.FOVWeight = 1;
param.POSWeight = 5;
param.LaplacianWeight = 0.23;



res = imageMatrix;  %Initial degraded image supplied to fnlcg
function
figure(300), imshow(abs(res), []);
title(['Initial image estimate using ', num2str(numberOfSpokes),
' spokes']);

% do iterations
```

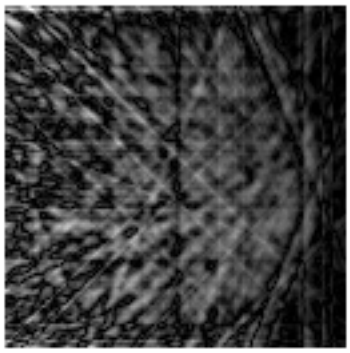
```
tic
for n=1:5
    [res, repetitionCounter] =
fnlCg(res,numberOfSpokes,dataMatrix, param);  %initialize fnlcg
    im_res = res;
    figure(100), imshow(abs(im_res),[]), drawnow;
    title(['Image estimate using ', num2str(numberOfSpokes), '
spokes']);


    if repetitionCounter > 7
        break;
    end;
end
toc


rmpath /Users/swrangsarbasumatary/Desktop/
imageProcessingProject2/
```

Estimating the final image from the 8 weighted images

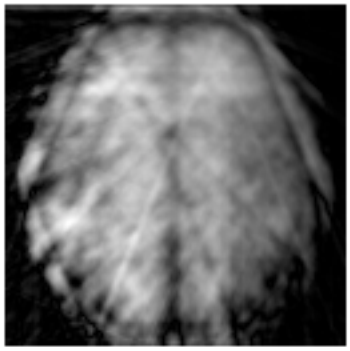
We remove the phase from each coil data by dividing it by the respective coil profile provided in the 'sim_8ch_data.mat' file. We then form a new aggregate image estimate by taking an average of all the eight coil estimates. We can now remove the imaginary parts because we have already removed the phase. The average image estimate is improved using TV constraint and other penalty functions on it to get the final estimated image.

The initial and filtered image estimate for each coil (in order from coil 1 to 8):

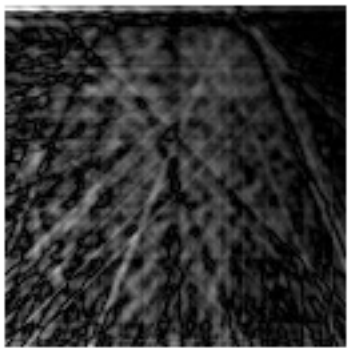
Coil data using 8 spokes

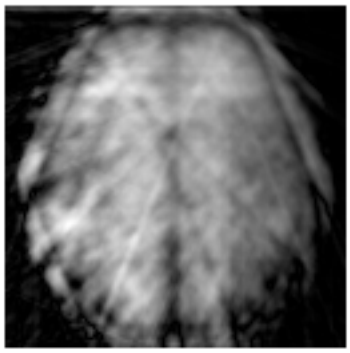


Coil estimate using 8 spokes

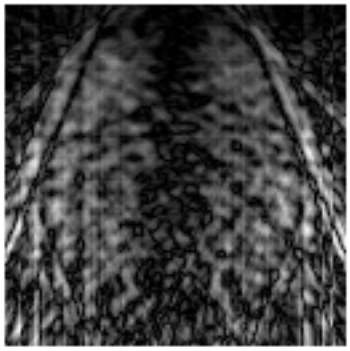


Coil data using 8 spokes

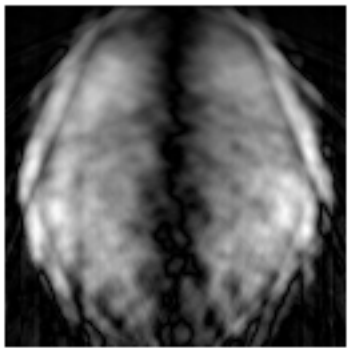




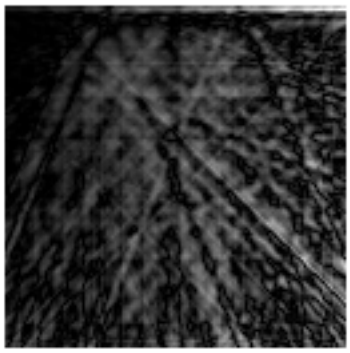
Coil data using 8 spokes



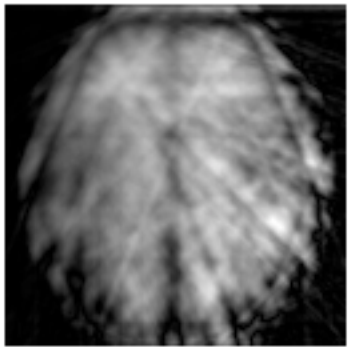
Coil estimate using 8 spokes



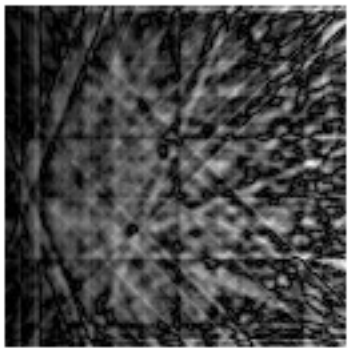
Coil data using 8 spokes



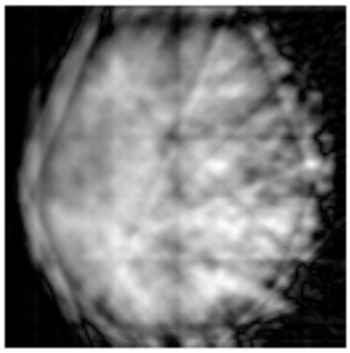
Coil estimate using 8 spokes



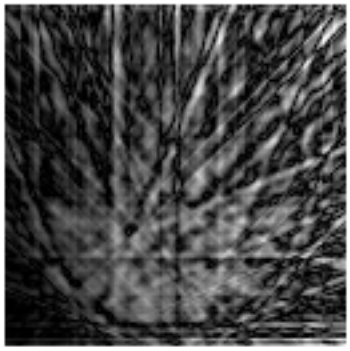
Coil data using 8 spokes



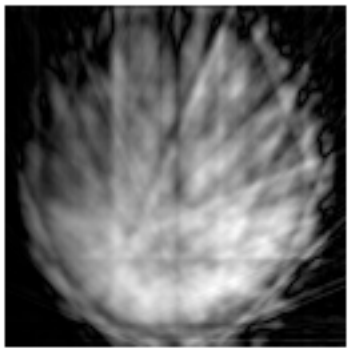
Coil estimate using 8 spokes



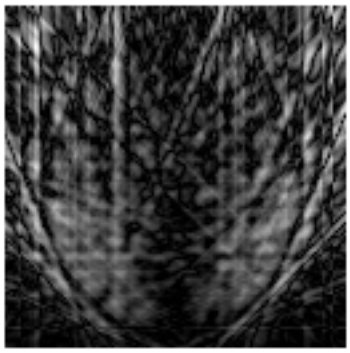
Coil data using 8 spokes



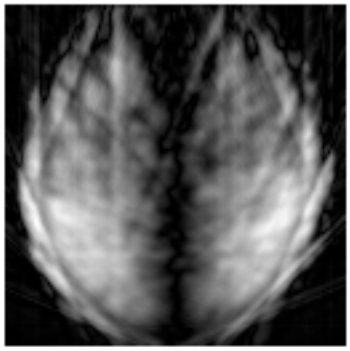
Coil estimate using 8 spokes



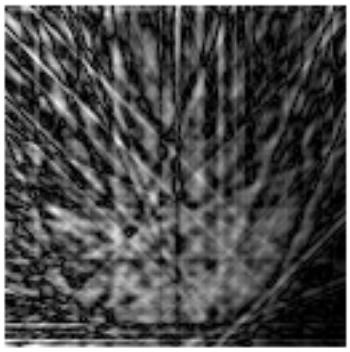
Coil data using 8 spokes

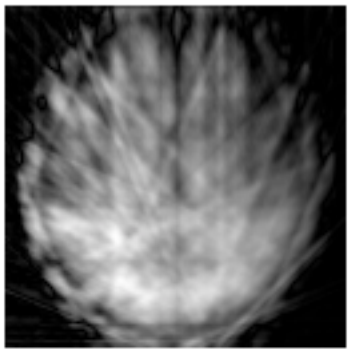


Coil estimate using 8 spokes

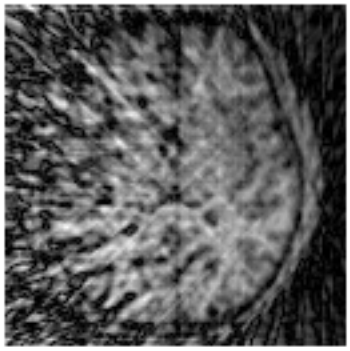


Coil data using 8 spokes

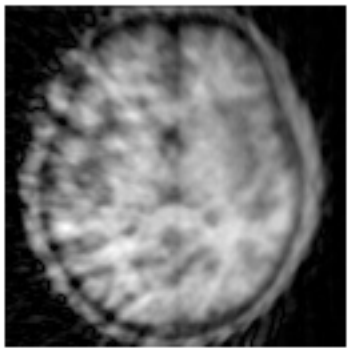




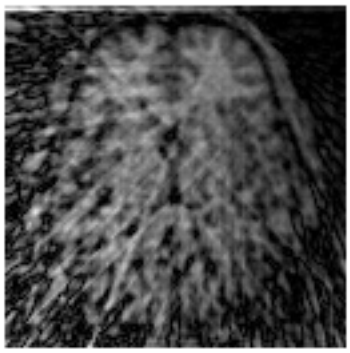
Coil data using 16 spokes



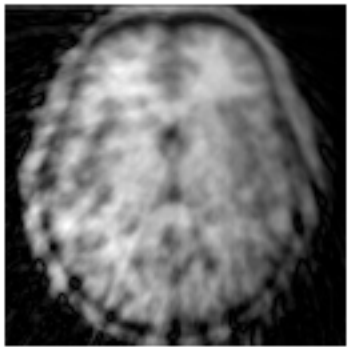
Coil estimate using 16 spokes



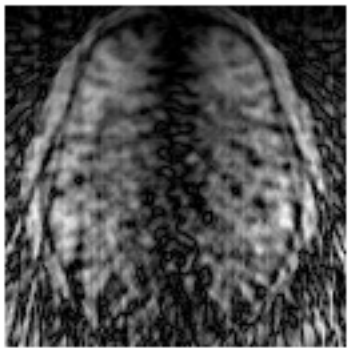
Coil data using 16 spokes



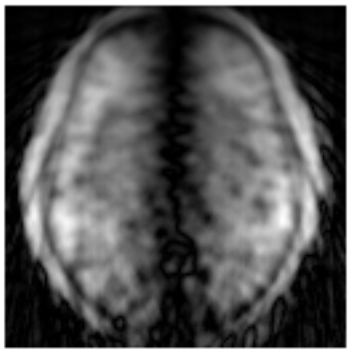
Coil estimate using 16 spokes



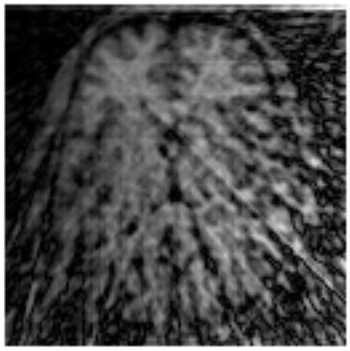
Coil data using 16 spokes



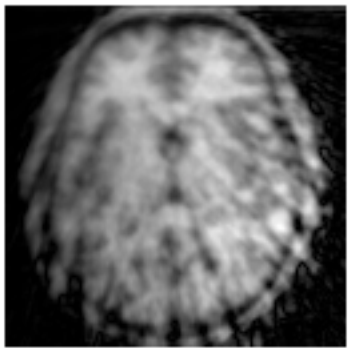
Coil estimate using 16 spokes



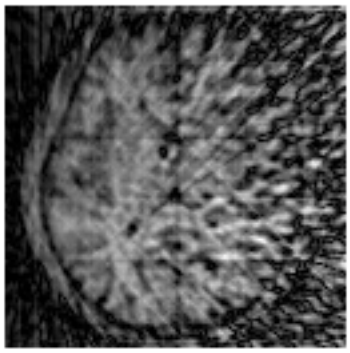
Coil data using 16 spokes



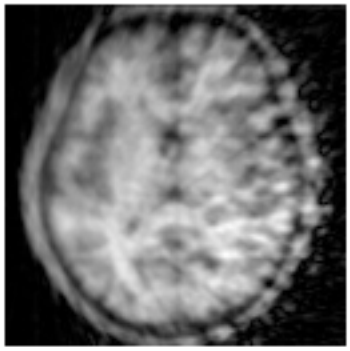
Coil estimate using 16 spokes



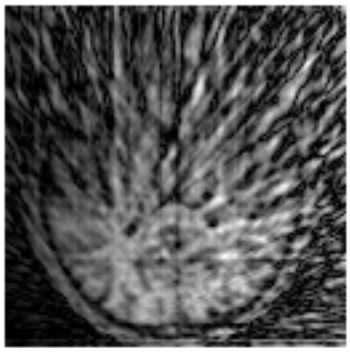
Coil data using 16 spokes



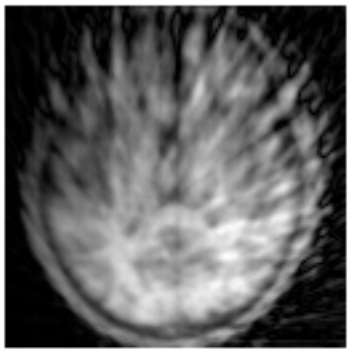
Coil estimate using 16 spokes



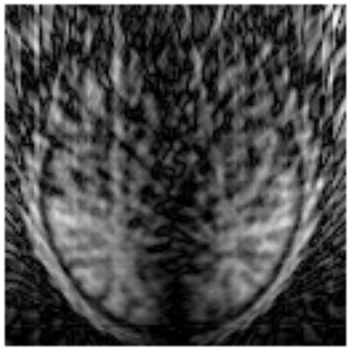
Coil data using 16 spokes



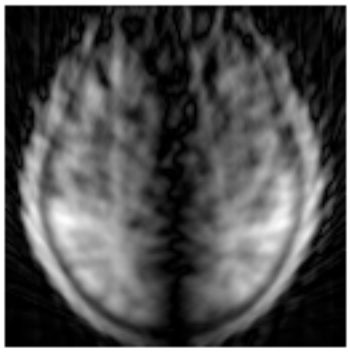
Coil estimate using 16 spokes



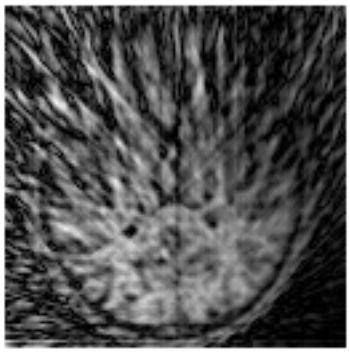
Coil data using 16 spokes



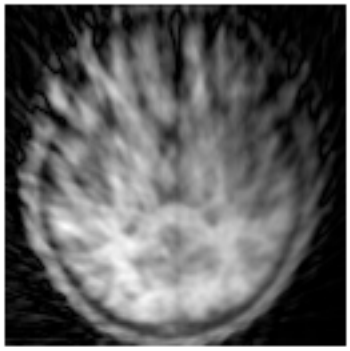
Coil estimate using 16 spokes



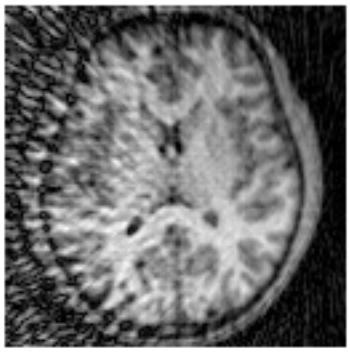
Coil data using 16 spokes



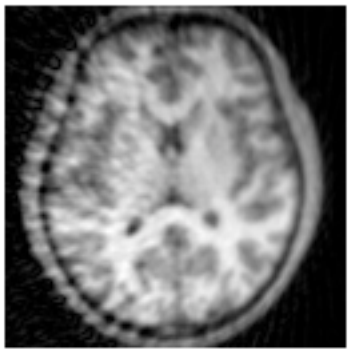
Coil estimate using 16 spokes



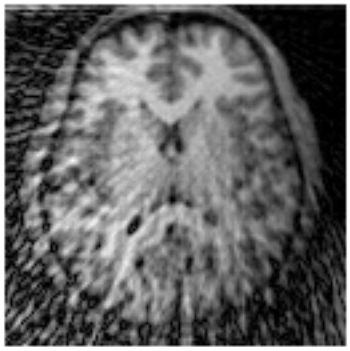
Coil data using 32 spokes



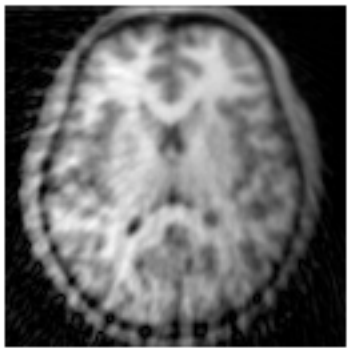
Coil estimate using 32 spokes



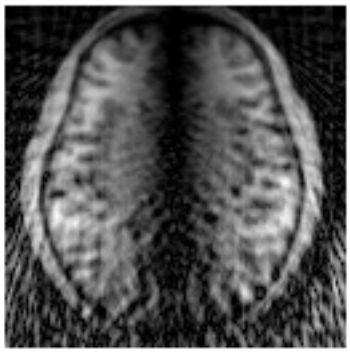
Coil data using 32 spokes



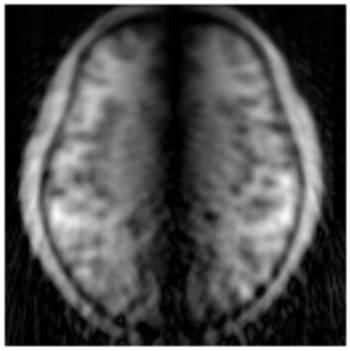
Coil estimate using 32 spokes



Coil data using 32 spokes

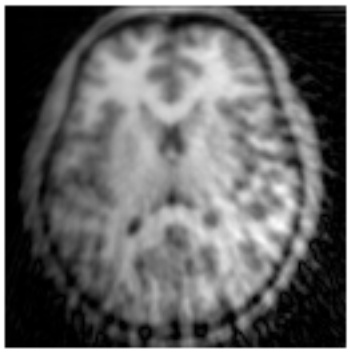


Coil estimate using 32 spokes

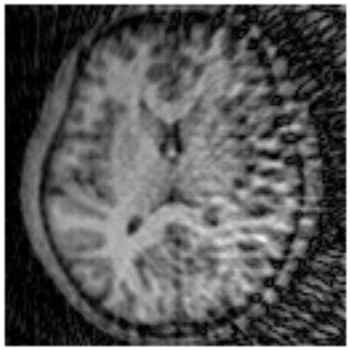


Coil data using 32 spokes

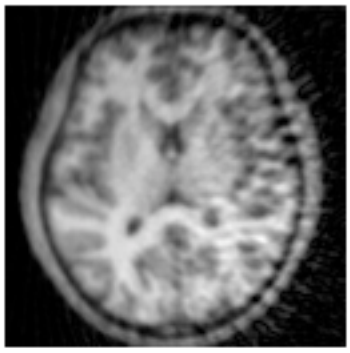
Coil estimate using 32 spokes



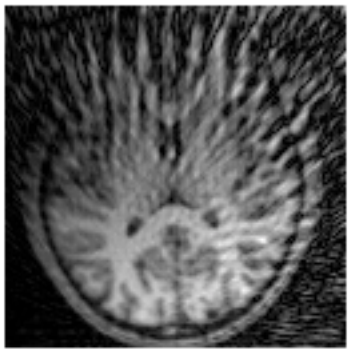
Coil data using 32 spokes

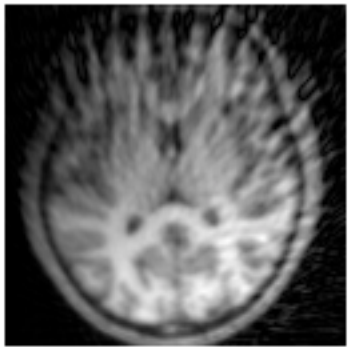


Coil estimate using 32 spokes

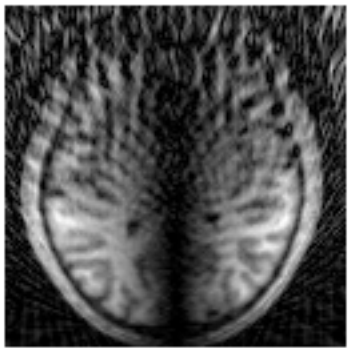


Coil data using 32 spokes

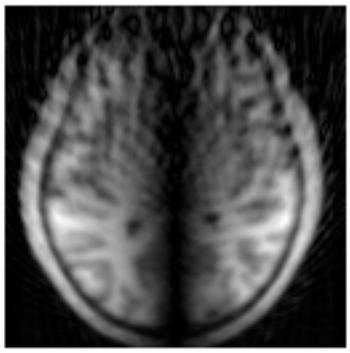




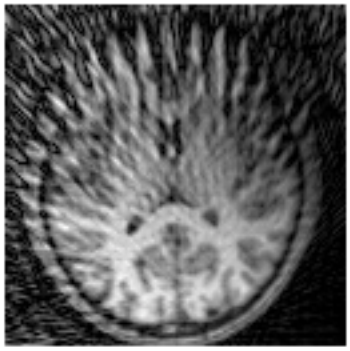
Coil data using 32 spokes



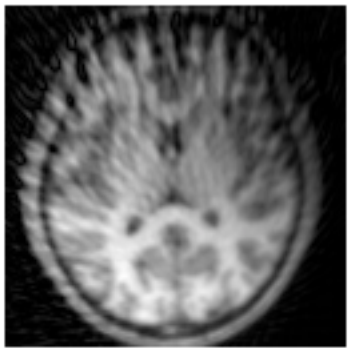
Coil estimate using 32 spokes



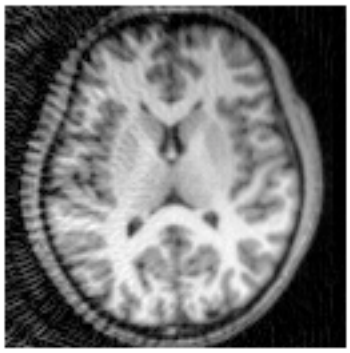
Coil data using 32 spokes



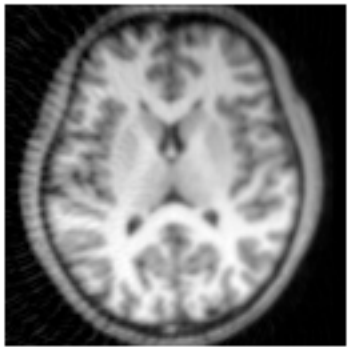
Coil estimate using 32 spokes



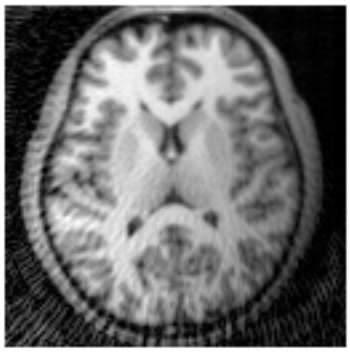
Coil data using 64 spokes



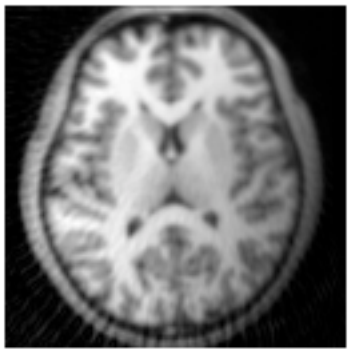
Coil estimate using 64 spokes



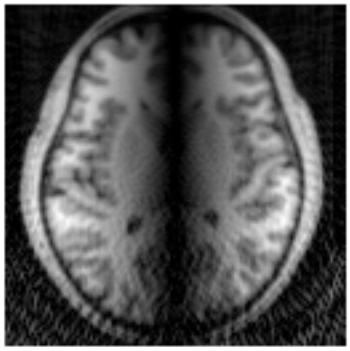
Coil data using 64 spokes



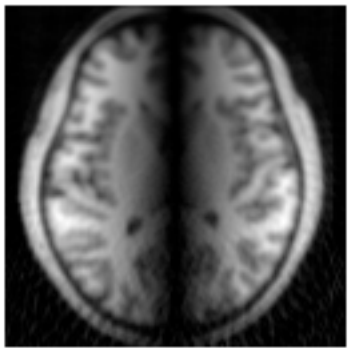
Coil estimate using 64 spokes



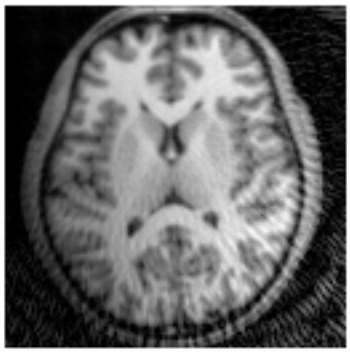
Coil data using 64 spokes



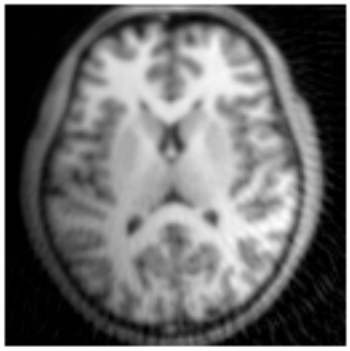
Coil estimate using 64 spokes



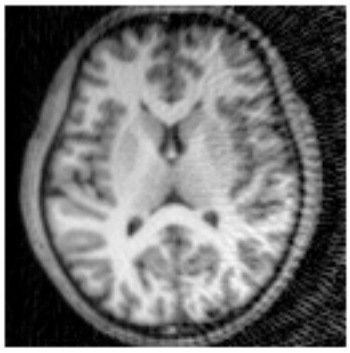
Coil data using 64 spokes



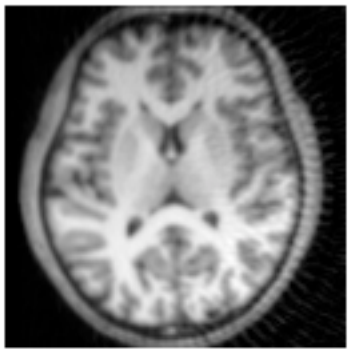
Coil estimate using 64 spokes



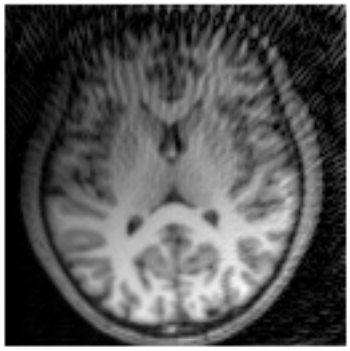
Coil data using 64 spokes



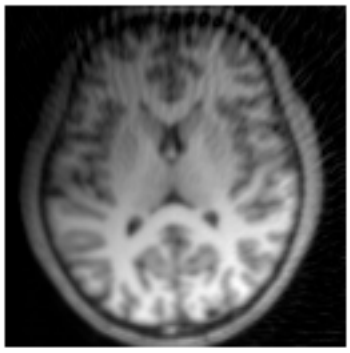
Coil estimate using 64 spokes



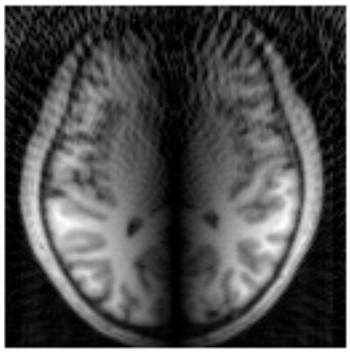
Coil data using 64 spokes



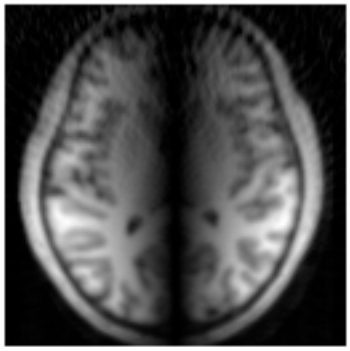
Coil estimate using 64 spokes



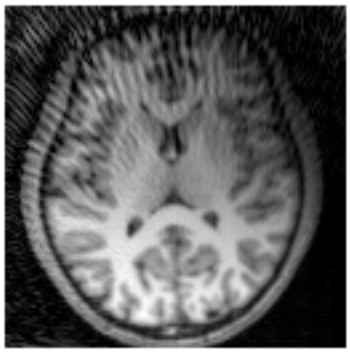
Coil data using 64 spokes



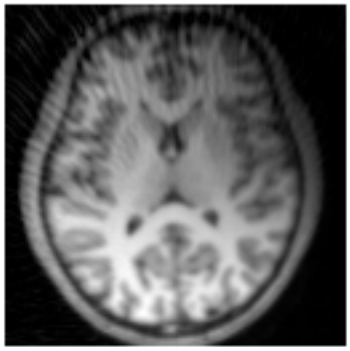
Coil estimate using 64 spokes



Coil data using 64 spokes

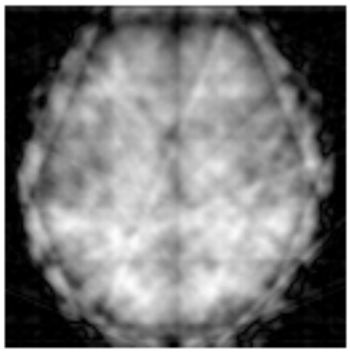


Coil estimate using 64 spokes

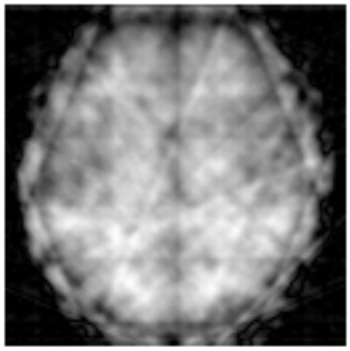


The aggregate average image estimates and their filtered versions:

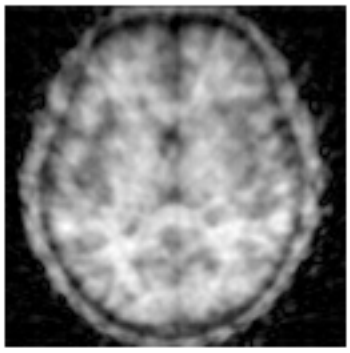
Avg coil estimate w/ 8 spokes



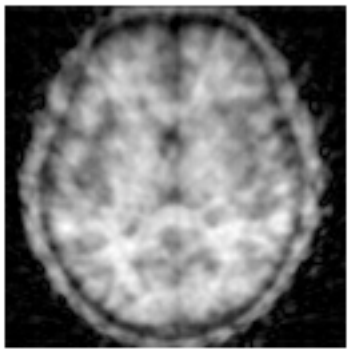
Final coil estimate w/8 spokes



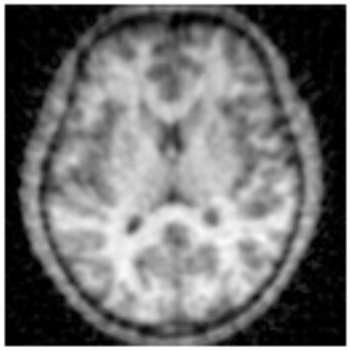
Avg coil estimate w/ 16 spokes



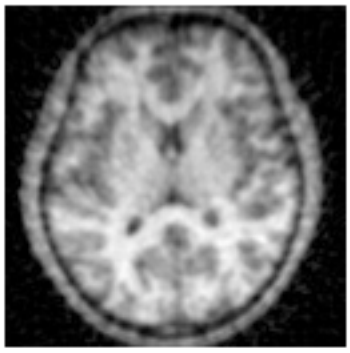
Final coil estimate w/ 16 spokes



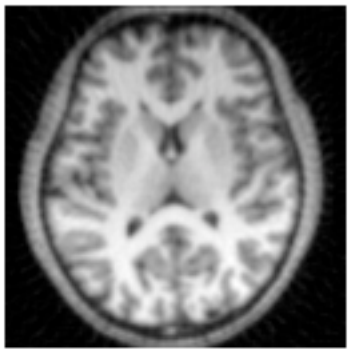
Avg coil estimate w/ 32 spokes



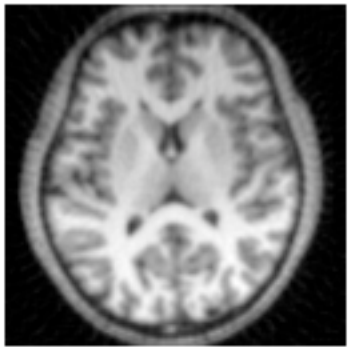
Final coil estimate w/ 32 spokes



Avg coil estimate w/ 64 spokes



Final coil estimate w/ 64 spokes



The functions:

```
function [x, repetitionCounter] =
fnlCgCoilEstimate(x0,numberOfSpokes,data, param)
%-------------------------------------------------------------
--------
%-------------------------------------------------------------
----------
% the nonlinear conjugate gradient method
disp('running fnlcgCoilEstimate');

x = x0;

% line search parameters - Dont touch..leave alone
maxlsiter = 150;
```

```
gradToll = 1.0000e-030;
alpha = 0.0100;
beta = 0.6000;
t0 = 1;
Itnlim = 16;

%%%%%%%%%%
repetitionCounter = 0;
previousObjective = 0;
%%%%%%%%%%


k = 0;

% copmute g0  = grad(Phi(x))
g0 = wGradient(x,numberOfSpokes,data, param);

dx = -g0;

% iterations
while(1)

% backtracking line-search

    % pre-calculate values, such that it would be cheap to
compute the objective
    % many times for efficient line-search
    f0 = objective(x,dx, 0, numberOfSpokes,data, param);
    t = t0;

        [f1]  =  objective(x,dx, t,numberOfSpokes,data, param);

    lsiter = 0;

    while (f1 > f0 - alpha*t*abs(g0(:)'*dx(:)))^2 &
(lsiter<maxlsiter)
        lsiter = lsiter + 1;
        t = t * beta;
        [f1]  =  objective(x,dx, t,numberOfSpokes,data, param);
    end

    if lsiter == maxlsiter
        disp('Reached max line search,.... not so good... might
have a bug in operators. exiting... ');
        return;
    end
```

```
    % control the number of line searches by adapting the
initial step search
    if lsiter > 2
        t0 = t0 * beta;
    end

    if lsiter<1
        t0 = t0 / beta;
    end

    x = (x + t*dx);

    %--------- uncomment for debug purposes
------------------------
    disp(sprintf('%d   , obj: %f ', k,f1));

%-----------------------------------------------------------------

    %conjugate gradient calculation- Dont touch

    g1 = wGradient(x,numberOfSpokes,data, param);
    bk = g1(:)'*g1(:)/(g0(:)'*g0(:)+eps);
    g0 = g1;
    dx =  - g1 + bk* dx;
    k = k + 1;

    %TODO: need to "think" of a "better" stopping criteria ;-)

    if f1 == previousObjective
        repetitionCounter = repetitionCounter + 1;
    else
        repetitionCounter = 0;
    end
    previousObjective = f1;

    if (k > Itnlim) | (norm(dx(:)) < gradToll)
        break;
    end
    if repetitionCounter > 5
        break;
    end

end

repetitionCounter
```

```
return;

end


%% the objective function

function [res] = objective(x,dx,t,numberOfSpokes,data, param)
%DEFINE obj
x = x + (t * dx);
b = data;
Ax = getDataMatrix(x, numberOfSpokes, param);
obj = (Ax - b);
res= (obj(:)'*obj(:)) + (param.TVWeight * getTotalVariation(x))
+ ...
    (param.FOVWeight * fov(x)) + ...
    (param.LaplacianWeight * getLaplacianResidual(x));

end


%% the grad function

function grad = wGradient(x,numberOfSpokes,data, param)

%Define this function
gradObj=gOBJ(x,numberOfSpokes,data, param);
grad = (gradObj) + (param.TVWeight * gradTotalVariation(x))
+ ...
    (param.FOVWeight * gradFOV(x)) ...
    + (param.LaplacianWeight * getLaplacianGradient(x));

end

%% calculating the gradient of the Objective

function gradObj = gOBJ(x,numberOfSpokes,data, param)

% computes the gradient of the data consistency
b = data;
inputSize = size(x, 1);
Ax = getDataMatrix(x, numberOfSpokes, param);
AhAx = getImageMatrix(Ax, numberOfSpokes, inputSize, param);
Ahb = getImageMatrix(b, numberOfSpokes, inputSize, param);
gradObj = 2 * (AhAx - Ahb);
```

```
end

%% the function implementing system matrix A

function dataMatrix = getDataMatrix(imageMatrix, numberOfSpokes,
param)

theta = 0:numberOfSpokes-1;
theta = theta .* (180/numberOfSpokes);
imageMatrix = imageMatrix .* param.CoilProfile;
dataMatrix = radon(imageMatrix, theta);
dataMatrix = fft(dataMatrix, [], 1); % column fft of the matrix

end

%% function implementing the adjoint system matrix A*

function imageMatrix = getImageMatrix(dataMatrix,
numberOfSpokes, inputSize, param)

theta = 0:numberOfSpokes-1;
theta = theta .* (180/numberOfSpokes);
imageMatrix = ifft(dataMatrix, [], 1);
imageMatrix = iradon(imageMatrix, theta, 'linear', 'Ram-Lak', 1,
inputSize);
imageMatrix = imageMatrix .* param.InverseCoilProfile;


end


%% the total variation penalty function

function totalVariation = getTotalVariation(imageMatrix)

variationX = filter2([1 -1 0], imageMatrix);
variationY = filter2([1; -1; 0], imageMatrix);
mag = sqrt( (abs(variationX) .^ 2) + abs((variationY) .^ 2));
totalVariation = sum(mag(:));

end


%% gradient of the total variation
```

```
function gradTV = gradTotalVariation(imageMatrix)

gradTV=filter2([0 -1 1],filter2([1 -1 0], imageMatrix))
+filter2([0;-1;1],filter2([1; -1; 0], imageMatrix));

end


%% the FOV Mask function

function fovMask = fovMask(inputMatrix)

[rows, cols] = size(inputMatrix);
xRadius = ceil(rows/2);
yRadius = ceil(cols/2);
[X, Y] = meshgrid(-(xRadius):(rows-(xRadius+1)), -yRadius:(cols-
(yRadius + 1)));
a = 4;
b = 5;
term1 = (X./a).^2;
term2 = (Y./b).^2;
radius = sqrt(term1 + term2);
normalizer = sqrt((xRadius/a)^2 + (0/b)^2);
normalizedRadius = radius/normalizer;
mask = normalizedRadius < 0.93;

inputMatrix(mask) = 0;
fovMask = inputMatrix;

end


%% the FOV penalty function

function fov = fov(x)

x = fovMask(x);
mag = abs(x) .^ 2;
fov = sum(mag(:));

end


%% the gradient of the FOV function

function gradFOV = gradFOV(x)
```

```
x = fovMask(x);
gradFOV = 2 .* x;

end


%% the laplacian function

function laplacianResidual = getLaplacianResidual(imageMatrix)

laplacianX = filter2([1 -2 1], imageMatrix);
laplacianY = filter2([1; -2; 1], imageMatrix);
mag = sqrt( (abs(laplacianX) .^ 2) + abs((laplacianY) .^ 2));
laplacianResidual = sum(mag(:));

end

%% the gradient of the laplacian

function laplacianGradient = getLaplacianGradient(imageMatrix)

laplacianGradient = filter2([0 -1 1],filter2([1 -2 1],
imageMatrix))+filter2([0;-1;1],filter2([1; -2; 1],
imageMatrix));

end



function coilEstimate = getCoilEstimate(inputImage, coilProfile,
numberOfSpokes)


inverseCoilProfile = 1 ./ coilProfile;


% generating the data vector


theta = 0:numberOfSpokes-1;
theta = theta .* (180/numberOfSpokes);
dataMatrix = radon(inputImage, theta);
dataMatrix = fft(dataMatrix, [], 1);
imageMatrix = ifft(dataMatrix, [], 1);
```

```
imageMatrix = iradon(imageMatrix, theta, 'linear', 'Ram-Lak', 1,
size(inputImage, 1));
imageMatrix = imageMatrix .* inverseCoilProfile;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Reconstruction Parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

param.TVWeight = 0.77;
param.FOVWeight = 1;
param.LaplacianWeight = 0.23;

param.CoilProfile = coilProfile;
param.InverseCoilProfile = inverseCoilProfile;

res = imageMatrix;
% figure, imshow(abs(res), []);
% title(['Coil data using ', num2str(numberOfSpokes), '
spokes']);


tic
% figure;
for n=1:5
    [res, repetitionCounter] =
fnlCgCoilEstimate(res,numberOfSpokes,dataMatrix, param);
%initialize fnlcg
    im_res = res;
%   imshow(abs(im_res),[]), drawnow;
%     title(['Coil estimate using ', num2str(numberOfSpokes), '
spokes']);
    if repetitionCounter > 5
        break;
    end;
end
toc

coilEstimate = im_res;

end
```

The script:

```
close all; clear all;
```

```
addpath /Users/swrangsarbasumatary/Desktop/
imageProcessingProject2/

load sim_8ch_data;
disp('sim_8ch_data loaded successfully!');


inputImage = data;
coilProfile = b1;

numberOfSpokes = 64;
numberOfCoils = 8;

coilEstimate = cell(numberOfCoils, 1);

for k = 1:numberOfCoils
    coilEstimate{k} = getCoilEstimate(data(:, :, k), b1(:, :,
k), numberOfSpokes);
%     figure(k*100), imshow(abs(coilEstimate{k}), []);
end

avgCoilEstimate = zeros(size(coilEstimate{1}));
for k = 1:numberOfCoils
    avgCoilEstimate = avgCoilEstimate + coilEstimate{k};
end
avgCoilEstimate = avgCoilEstimate ./ numberOfCoils;

% since we removed the phases we can now take just the real part
of the avg
% estimate image

finalCoilEstimate = real(avgCoilEstimate);


% figure(900); clf; imshow(abs(avgCoilEstimate), []);

%% next we improve the average coil estimate

% generating the data vector

theta = 0:numberOfSpokes-1;
theta = theta .* (180/numberOfSpokes);
dataMatrix = radon(finalCoilEstimate, theta);
dataMatrix = fft(dataMatrix, [], 1);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Reconstruction Parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%

param.TVWeight = 0.0001;     % Weight for TV penalty
param.FOVWeight = 10;
param.POSWeight = 5;
param.LaplacianWeight = 0.23;



res = finalCoilEstimate;  %Initial degraded image supplied to
fnlcg function
figure(300), imshow(abs(res), []);
title(['Avg coil estimate w/ ', num2str(numberOfSpokes), '
spokes']);

% do iterations
tic
for n=1:5
    [res, repetitionCounter] =
fnlCg(res,numberOfSpokes,dataMatrix, param);  %initialize fnlcg
    im_res = res;
    figure(100), imshow(abs(im_res),[]), drawnow;
    title(['Final coil estimate w/ ', num2str(numberOfSpokes), '
spokes']);

    if repetitionCounter > 7
        break;
    end;
end
toc

rmpath /Users/swrangsarbasumatary/Desktop/
imageProcessingProject2/
```