Berfin Kavşut
21602459
EEE475 -1
May 12, 2020

REPORT FOR HOMEWORK #4

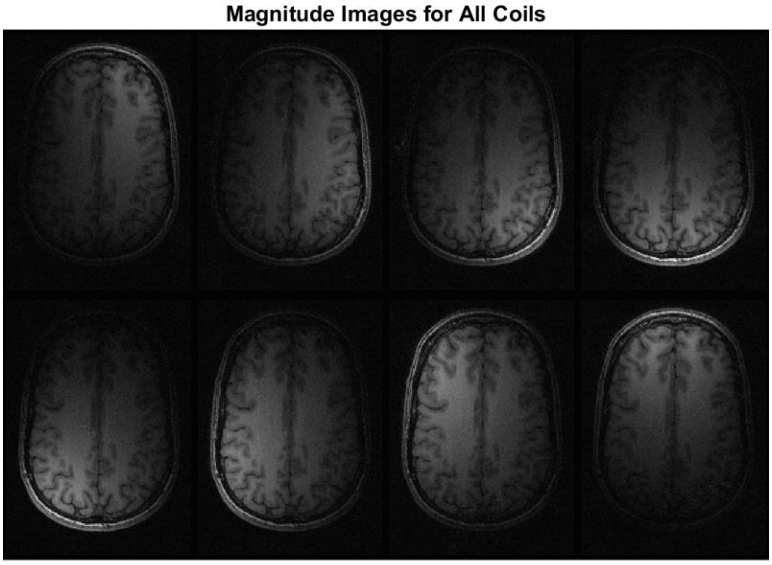
PART I – SENSE

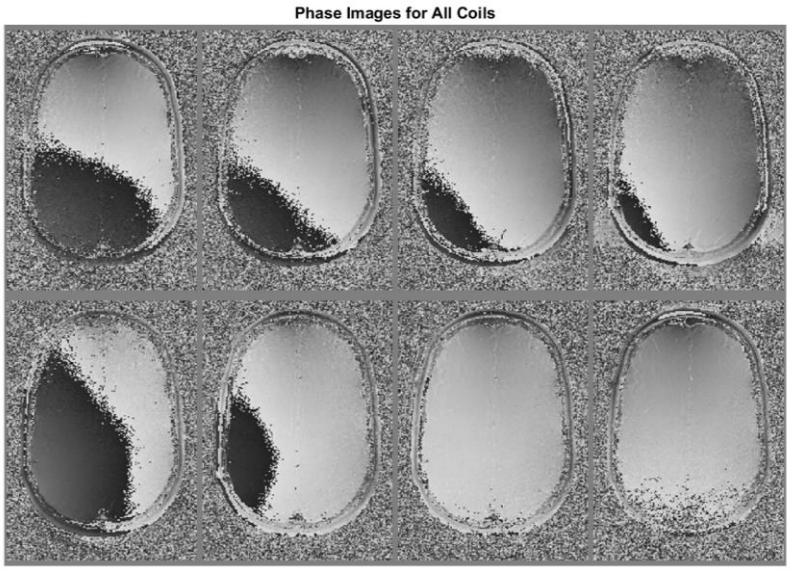
1.1) Display the Data

Magnitude of coil sensitivities for map1 correspond to the magnitude images for all coils. The phases of coil sensitivities for map1 correspond to the phases for the image from all coils. Here, we can comment that our images from all coils have the information of magnitudes and phases of coil sensitivities.

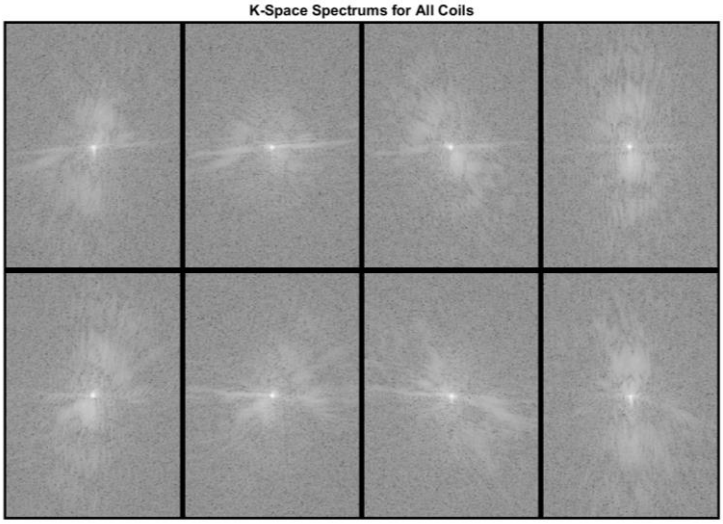
Magnitudes of coil sensitivities are more different in map1, which means map1 will give better results for SENSE reconstruction.

Images

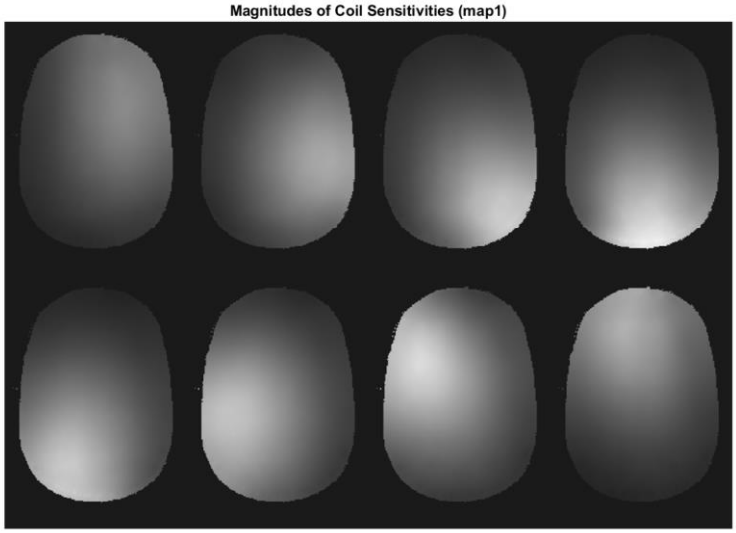




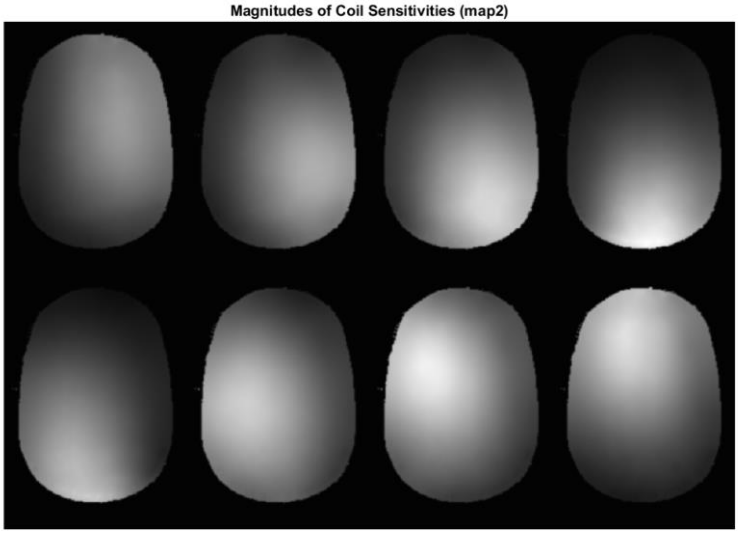
K-Space Spectrums for All Coils

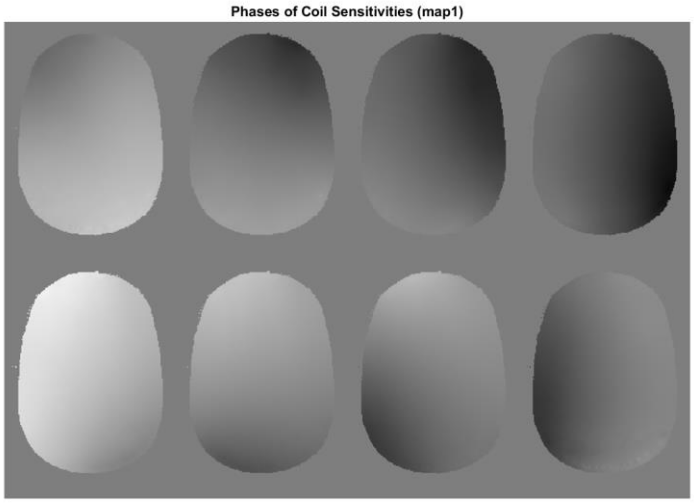


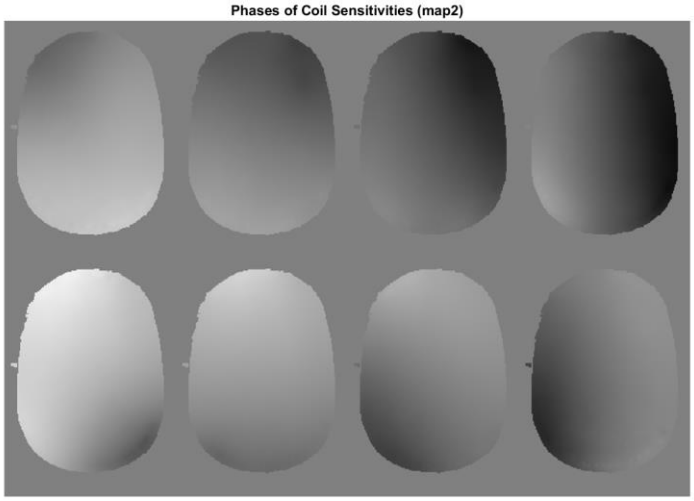
Magnitudes of Coil Sensitivities (map1)



Magnitudes of Coil Sensitivities (map2)







MATLAB Code

```
case '1.1'
disp('1.1')

disp('Display the Data');
load('multicoil-data.mat');

%take 2DFT of image
for coil_no = 1:8
    d(:,:,coil_no) = fft2c(im(:,:,coil_no));
end

%magnitude images for all coils
figure;set(gcf, 'WindowState', 'maximized');
montage(abs(im),'DisplayRange', [], 'Size', [2 4], 'BorderSize', 5);
title('Magnitude Images for All Coils');
saveas(gcf,'1.1_mag.png');
```

```
%phase images for all coils
figure;set(gcf, 'WindowState', 'maximized');
montage(angle(im),'DisplayRange', [], 'Size', [2 4], 'BorderSize',
5);
title('Phase Images for All Coils');
saveas(gcf,'1.1_phase.png');

%k-space spectrums for all coils
figure; set(gcf, 'WindowState', 'maximized');
montage(log(abs(d)+1),'DisplayRange', [], 'Size', [2 4],
'BorderSize', 5);
title('K-Space Spectrums for All Coils');
saveas(gcf,'1.1_kspace.png');

%coil sensitivities for map1
%magnitudes of coil sensitivities
figure;set(gcf, 'WindowState', 'maximized');
montage(abs(map1),'DisplayRange', [], 'Size', [2 4], 'BorderSize',
5);
title('Magnitudes of Coil Sensitivities (map1)');
saveas(gcf,'1.1_coil_mag_map1.png');

%phases of coil sensitivities
figure;set(gcf, 'WindowState', 'maximized');
montage(angle(map1),'DisplayRange', [], 'Size', [2 4], 'BorderSize',
5);
title('Phases of Coil Sensitivities (map1)');
saveas(gcf,'1.1_coil_phase_map1.png');

%coil sensitivities for map2
%magnitudes of coil sensitivities
figure;set(gcf, 'WindowState', 'maximized');
montage(abs(map2),'DisplayRange', [], 'Size', [2 4], 'BorderSize',
5);
title('Magnitudes of Coil Sensitivities (map2)');
saveas(gcf,'1.1_coil_mag_map2.png');

%phases of coil sensitivities
figure;set(gcf, 'WindowState', 'maximized');
montage(angle(map2),'DisplayRange', [], 'Size', [2 4], 'BorderSize',
5);
title('Phases of Coil Sensitivities (map2)');
saveas(gcf,'1.1_coil_phase_map2.png');
```

1.2) Multi-coil Reconstruction

Let us derive SoS and OLC solutions as derived in our lecture notes:

Sum of Squares (SoS) – Approximate Solution

$$m_{ss}(x) = \sqrt{\sum_l m_l(x) * m_l{}^*(x)}$$

$$m_{ss}(x) = |m(x)| * \sqrt{\sum_l |c_l(x)|^2}$$

$$|m(x)| = \frac{m_{ss}(x)}{\sqrt{\sum_l |c_l(x)|^2}}$$

$$\sqrt{\sum_l |c_l(x)|^2} \quad : ideally\ uniform\ over\ the\ entire\ object$$

Weighted Linear Combination (OLC) – Least Squares Solution

$$m_l(x) = c_l(x) * m(x)$$

$$m(x_p) = \frac{\sum_l c_l{}^*(x_p) * m_l(x_p)}{\sqrt{\sum_l |c_l(x_p)|^2}}$$

Phase corrected with $c_l{}^*(x_p)$

Weight corrected with $\frac{1}{\sqrt{\sum_l |c_l(x_p)|^2}}$

Comments:

PSNR and SSIM is a lot higher for SoS than for OLC map2. As it is said in Q1.1, map2 has less different or in other words orthogonal coil sensitivities. Therefore, map2 results are worse than map1 results throughout Part 1. The error is large in error image, and SSIM value is quite small.

We took OLC result as our reference, and calculated our PSNR, SSIM and error images. From SSIM value, we see SoS and OLC results are very close to each other. Also, error image shows that there is very little error visually.

Results

1.2

Multi-coil Reconstruction

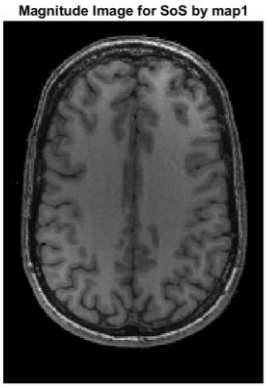
PSNR for SoS:43.1455

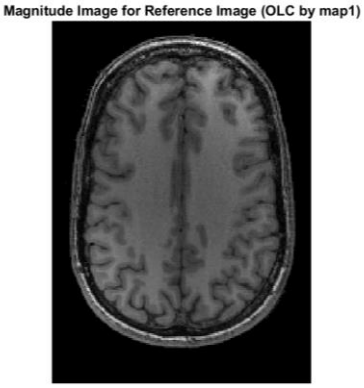
SSIM for SoS:0.99784

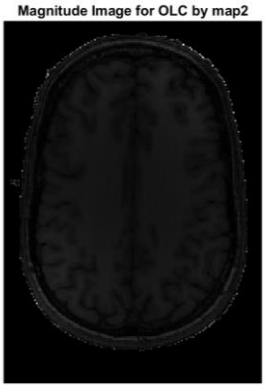
PSNR for OLC with map2:14.0153

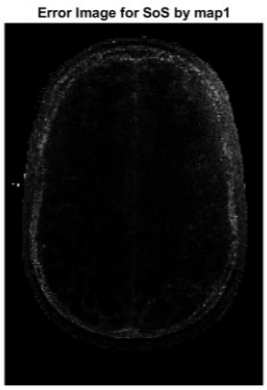
SSIM for OLC with map2:0.483

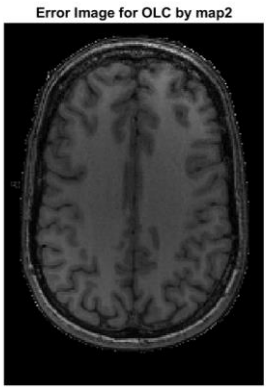
Images











MATLAB Code

```
case '1.2'
disp('1.2')
disp('Multi-coil Reconstruction');

load('multicoil-data.mat');

%SoS by map1
m_sos = sos(im,map1);

%OLC by map1
%reference image
m1_olc = olc(im,map1);
%OLC by map2
m2_olc = olc(im,map2);

%magnitude image for SoS
figure;set(gcf, 'WindowState', 'maximized');
```

```
imshow(abs(m_sos),[]);
title('Magnitude Image for SoS by map1');
saveas(gcf,'1.2_sos.png');

%magnitude image for reference image (OLC by map1)
figure;set(gcf, 'WindowState', 'maximized');
imshow(abs(m1_olc), []);
title('Magnitude Image for Reference Image (OLC by map1)');
saveas(gcf,'1.2_olc1.png');

%magnitude image for OLC by map2
figure;set(gcf, 'WindowState', 'maximized');
imshow(abs(m2_olc), []);
title('Magnitude Image for OLC by map2');
saveas(gcf,'1.2_olc2.png');

%normalize images
ref = m1_olc;
ref = abs(ref);
norm_ref = ref/max(ref(:));
save('reference.mat','norm_ref');

m_sos= abs(m_sos);
norm_m_sos = m_sos/max(m_sos(:));

m2_olc= abs(m2_olc);
norm_m2_olc = m2_olc/max(m2_olc(:));

%error image
error_sos = abs(norm_m_sos-norm_ref);
error_olc2 = abs(norm_m2_olc-norm_ref);

%error image for SoS
figure;set(gcf, 'WindowState', 'maximized');
imshow(abs(error_sos), []);
title('Error Image for SoS by map1');
saveas(gcf,'1.2_error_sos.png');

%magnitude image for OLC by map2
figure;set(gcf, 'WindowState', 'maximized');
imshow(abs(error_olc2), []);
title('Error Image for OLC by map2');
saveas(gcf,'1.2_error_olc2.png');

%IQA results
PSNR_sos = psnr(norm_m_sos,norm_ref);
SSIM_sos = ssim(norm_m_sos,norm_ref);

PSNR_olc2 = psnr(norm_m2_olc,norm_ref);
SSIM_olc2 = ssim(norm_m2_olc,norm_ref);

disp(strcat('PSNR for SoS:',num2str(PSNR_sos)));
disp(strcat('SSIM for SoS:',num2str(SSIM_sos)));

disp(strcat('PSNR for OLC with map2:',num2str(PSNR_olc2)));
disp(strcat('SSIM for OLC with map2:',num2str(SSIM_olc2)));
```

m = sos(im,map)function

```
function  m = sos(im,map)

% sum of squares (sos) function
% im: images from all coils
% map: coil sensitivities
% m: result image

%find mss from coil images
mss = sqrt( sum( im.*conj(im), 3) );
%find weights from coil sensitivities
coil_sens= sqrt( sum( abs(map).^2,3 ) );
%find image from mss and calculated weights
m = mss./coil_sens;

%correct image by setting inf and nan values to zero
m(isinf(m)) = 0;
m(isnan(m)) = 0;

end
```

m = olc(im,map) function

```
function m = olc(im,map)
% weighted linear combination (olc) function
% im: images from all coils
% map: coil sensitivities
% m: result image

%correct phase by complex conjugate of coil sensitivities
phase_corrected = sum( conj(map).*im , 3 );
%correct weight by magnitude of cail sensitivities
weight_corrected = ( 1./sum( abs(map).^2, 3 ) ).* phase_corrected;
%result image
m= weight_corrected;

%correct image by setting inf and nan values to zero
m(isinf(m)) = 0;
m(isnan(m)) = 0;

end
```

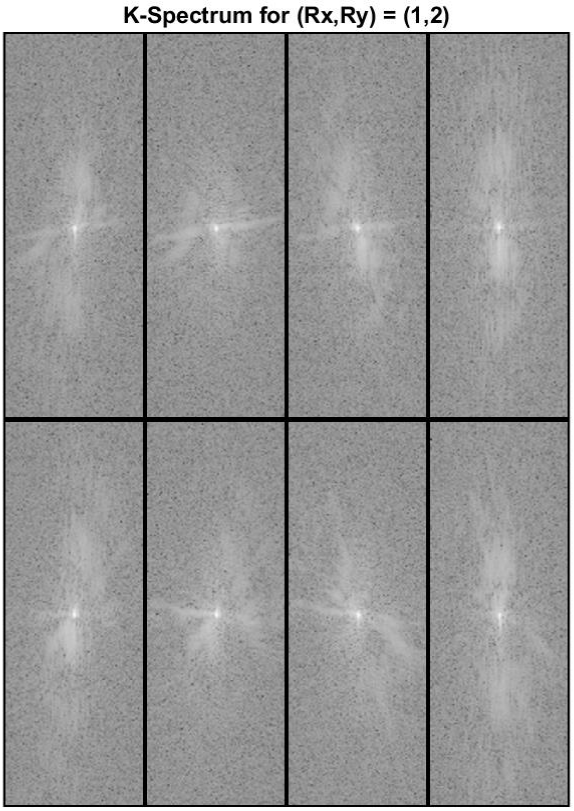
1.3) Generate Undersampled Images

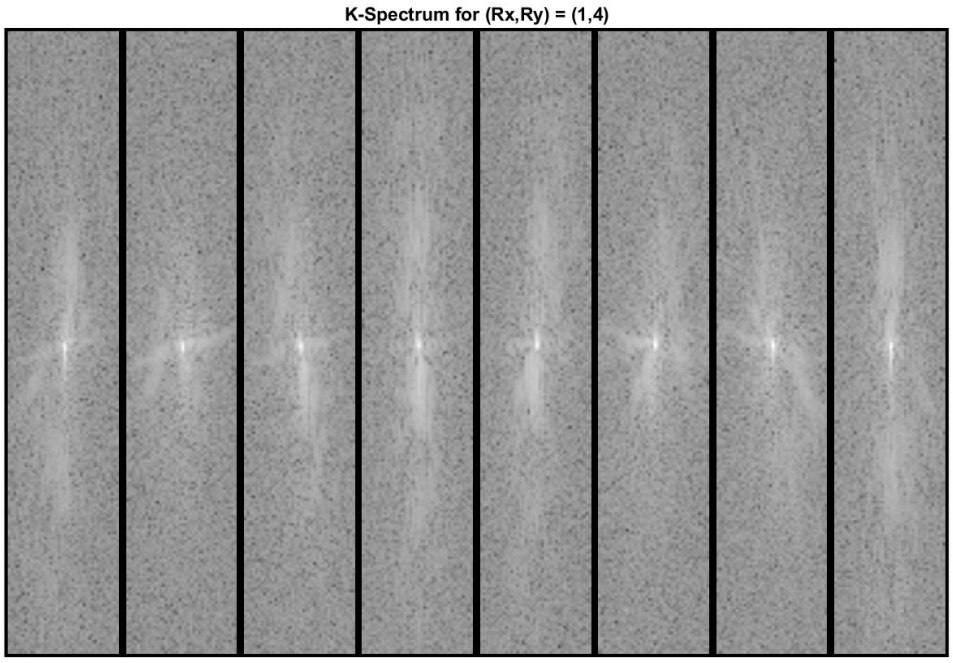
Rx causes reduction of k-space data in kx direction, Ry causes reduction in k-space data in ky-direction. Undersampling in kx-direction causes aliasing in x-direction. Similarly, undersampling in ky-direction causes aliasing in y-direction. Horizontal axis is kx and vertical axis is ky for k-space spectrums.

From k-space spectrum images, the reduction in k-space data can be seen as described. When acceleration rate increases, the artifacts gets closer to each other due to undersampling and this aliasing effect can be seen in magnitude images. For each aliasing, we obtain the central part of the

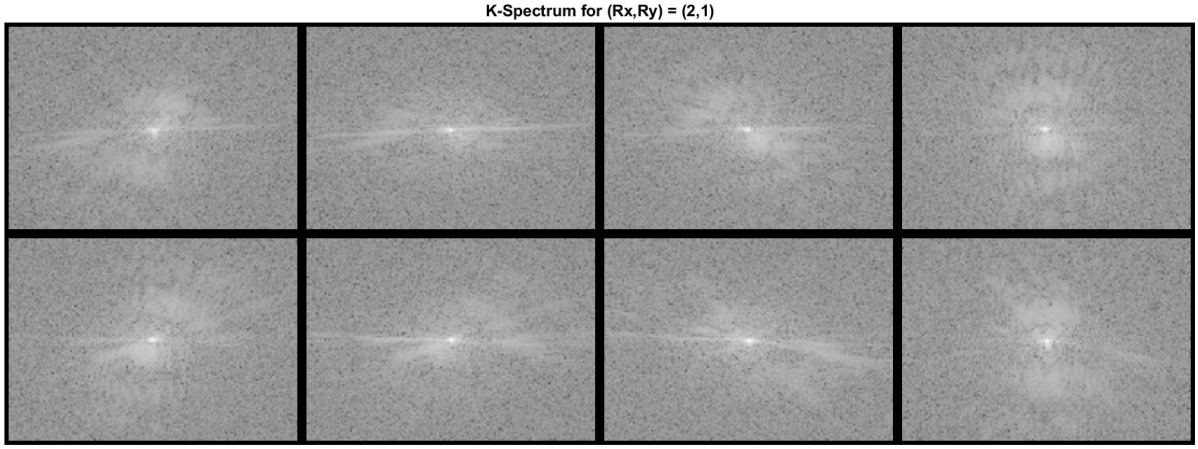
real image and artifacts on top of the real image. The size is smaller than original image due to discarding k-space data. If we zeroed out for undersampling, but we discarded the k-space data, we would see images in original size with aliasing.

Images

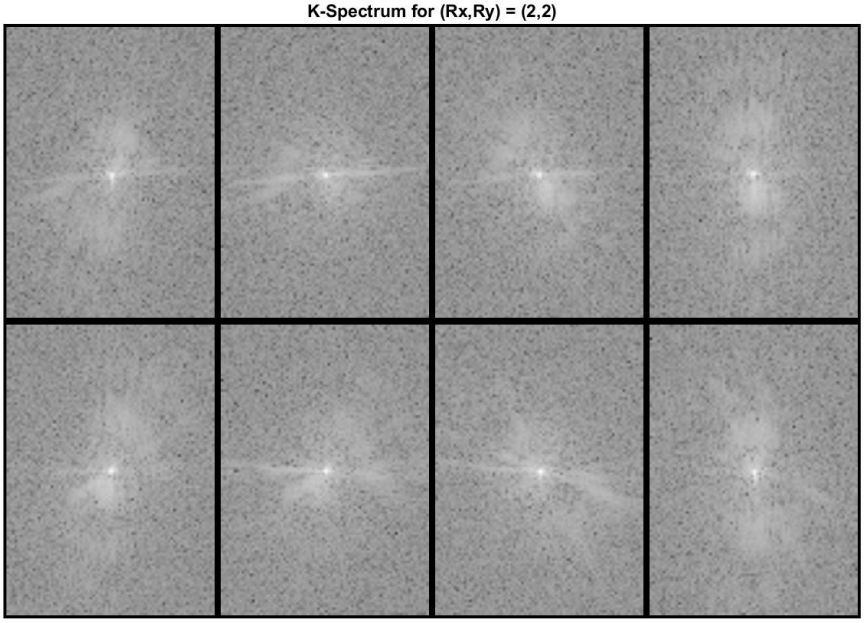




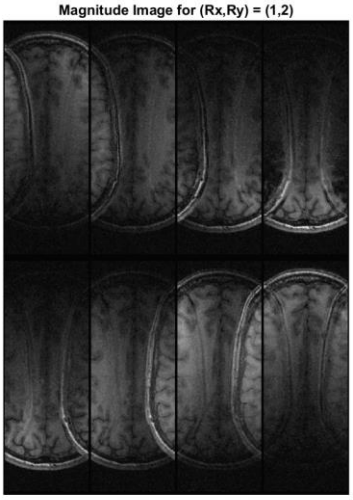
K-Spectrum for (Rx,Ry) = (2,1)



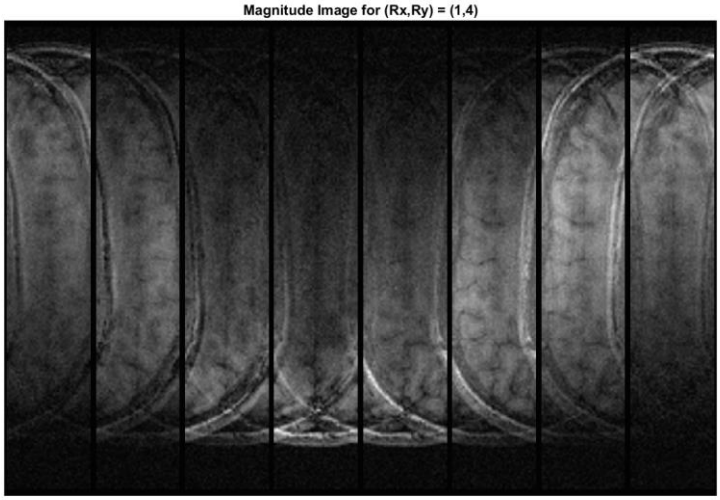
K-Spectrum for (Rx,Ry) = (2,2)



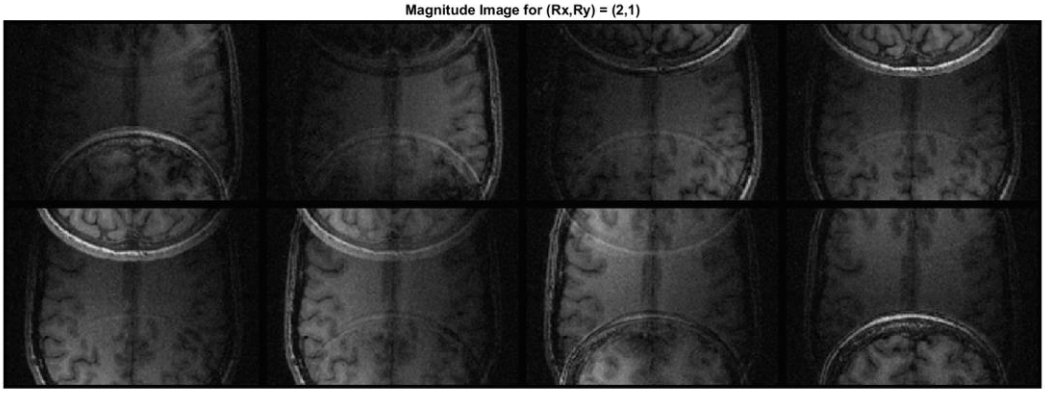
Magnitude Image for (Rx,Ry) = (1,2)



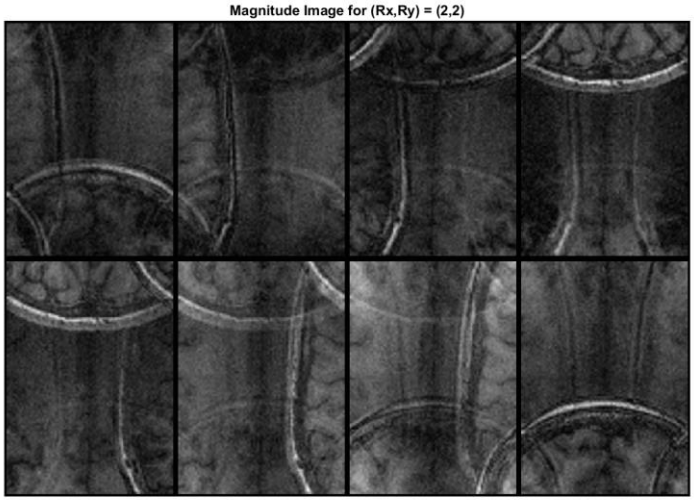
Magnitude Image for (Rx,Ry) = (1,4)

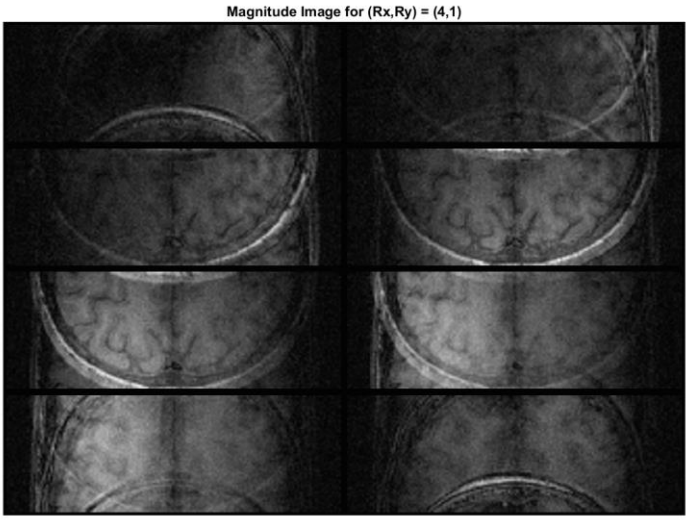


Magnitude Image for (Rx,Ry) = (2,1)



Magnitude Image for (Rx,Ry) = (2,2)





MATLAB Code

```
case '1.3'
disp('1.3')

disp('Generate Undersampled Images');

load('multicoil-data.mat');
load('reference.mat','norm_ref');

%(Rx,Ry) = (1,2)
Rx = 1; Ry = 2;
[imu, Mu] = undersample(im, Rx, Ry);

%aliased images for all coils
figure; set(gcf, 'WindowState', 'maximized');
montage(abs(imu),'displayRange',[], 'Size', [2 4], 'BorderSize', 5);
title('Magnitude Image for (Rx,Ry) = (1,2)');
saveas(gcf,'1.3_mag(1,2).png');

%undersampled k-space spectrums for all coils
figure; set(gcf, 'WindowState', 'maximized');
montage(log(abs(Mu)+1),'displayRange',[], 'Size', [2 4],
'BorderSize', 5);
title('K-Spectrum for (Rx,Ry) = (1,2)');
saveas(gcf,'1.3_kspace(1,2).png');

%(Rx,Ry) = (2,1)
Rx = 2; Ry = 1;
[imu, Mu] = undersample(im, Rx, Ry);

%aliased images for all coils
figure; set(gcf, 'WindowState', 'maximized');
montage(abs(imu),'displayRange',[], 'Size', [2 4], 'BorderSize', 5);
title('Magnitude Image for (Rx,Ry) = (2,1)');
saveas(gcf,'1.3_mag(2,1).png');
```

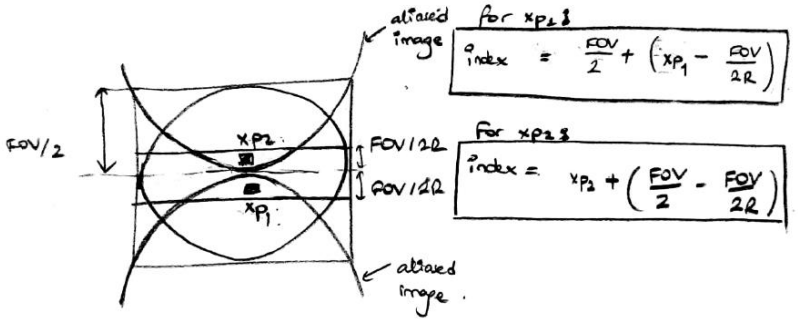
```
%undersampled k-space spectrums for all coils
figure; set(gcf, 'WindowState', 'maximized');
montage(log(abs(Mu)+1),'displayRange',[], 'Size', [2 4],
'BorderSize', 5);
title('K-Spectrum for (Rx,Ry) = (2,1)');
saveas(gcf,'1.3_kspace(2,1).png');

%(Rx,Ry) = (2,2)
Rx = 2; Ry = 2;
[imu, Mu] = undersample(im, Rx, Ry);

%aliased images for all coils
figure; set(gcf, 'WindowState', 'maximized');
montage(abs(imu),'displayRange',[], 'Size', [2 4], 'BorderSize', 5);
title('Magnitude Image for (Rx,Ry) = (2,2)');
saveas(gcf,'1.3_mag(2,2).png');

%undersampled k-space spectrums for all coils
figure; set(gcf, 'WindowState', 'maximized');
montage(log(abs(Mu)+1),'displayRange',[], 'Size', [2 4],
'BorderSize', 5);
title('K-Spectrum for (Rx,Ry) = (2,2)');
saveas(gcf,'1.3_kspace(2,2).png');

%(Rx,Ry) = (1,4)
Rx = 1; Ry = 4;
[imu, Mu] = undersample(im, Rx, Ry);

%aliased images for all coils
figure; set(gcf, 'WindowState', 'maximized');
montage(abs(imu),'displayRange',[], 'Size', [1 8], 'BorderSize', 5);
title('Magnitude Image for (Rx,Ry) = (1,4)');
saveas(gcf,'1.3_mag(1,4).png');

%undersampled k-space spectrums for all coils
figure; set(gcf, 'WindowState', 'maximized');
montage(log(abs(Mu)+1),'displayRange',[], 'Size', [1 8],
'BorderSize', 5);
title('K-Spectrum for (Rx,Ry) = (1,4)');
saveas(gcf,'1.3_kspace(1,4).png');

%(Rx,Ry) = (4,1)
Rx = 4; Ry = 1;
[imu, Mu] = undersample(im, Rx, Ry);

%aliased images for all coils
figure; set(gcf, 'WindowState', 'maximized');
montage(abs(imu),'displayRange',[], 'Size', [4 2], 'BorderSize', 5);
title('Magnitude Image for (Rx,Ry) = (4,1)');
saveas(gcf,'1.3_mag(4,1).png');

%undersampled k-space spectrums for all coils
figure; set(gcf, 'WindowState', 'maximized');
montage(log(abs(Mu)+1),'displayRange',[], 'Size', [4 2],
'BorderSize', 5);
title('K-Spectrum for (Rx,Ry) = (4,1)');
```

```
saveas(gcf,'1.3_kspace(4,1).png');
```

[imu, Mu] = undersample(im, Rx, Ry) function

```
function [imu, Mu] = undersample(im, Rx, Ry)

% inputs
% im : images from all coils
% Rx : acceleration rate on x-direction
% Ry : acceleration rate on y-direction

% outputs
% Mu : undersampled k-space data for all coils
% imu : aliased images by undersampled k-space data for all coils

for coil_no = 1:8

    %take 2D FFT
    d = fft2c(im(:,:,coil_no));
    [Nx Ny] = size(d);

    %generate positions to remain after undersampling
    pos1 = 1:Rx:Nx; %rows (x-direction)
    pos2 = 1:Ry:Ny; %columns (y-direction)

    %discard k-space data with respect to Rx and Ry
    Mu(:,:,coil_no) = d(pos1,pos2); %size: (Nx/Rx) x (Ny/Ry) x Nc
    %take inverse 2D FFT
    imu(:,:,coil_no) = ifft2c(Mu(:,:,coil_no)); %size: (Nx/Rx) x
(Ny/Ry) x Nc

end

end
```

1.4) SENSE Reconstruction



The indices of source voxels are calculated with the logic shown above. For each source voxel, the index is increased by step size, and then its mod with respect to FOVx/FOVy is calculated in *l2sense* function.

SENSE reconstruction gives the same result with OLC. We can verify this fact by SSIM being equal to 1. The maximum value that SSIM can take is one, which means the compared images are identical. Also, it can be seen that PSNR value is very high, which means the images should be very close to each other numerically.

Results

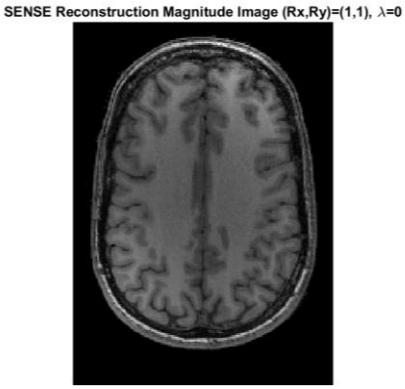
1.4

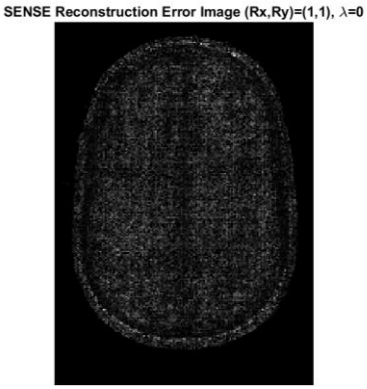
SENSE Reconstruction

PSNR for SENSE Reconstruction (Rx,Ry)=(1,1), lambda=0:321.864

SSIM for SENSE Reconstruction (Rx,Ry)=(1,1), lambda=0:1

Images





MATLAB Code

```
case '1.4'
disp('1.4')
disp('SENSE Reconstruction');

load('multicoil-data.mat');
load('reference.mat');

%test inputs
Rx = 1; Ry = 1;
lambda = 0;

[imu, Mu] = undersample(im, Rx, Ry);
im_sense = l2sense(imu, map1, Rx, Ry, lambda);

%normalize image
im_sense = abs(im_sense);
im_sense = im_sense/max(im_sense(:));
```

```
%magnitude image for l2-regularized SENSE reconstruction
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense),[]);
title('SENSE Reconstruction Magnitude Image (Rx,Ry)=(1,1),
{\lambda}=0');
saveas(gcf,'1.4_mag.png');

%error image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense-ref),[]);
title('SENSE Reconstruction Error Image (Rx,Ry)=(1,1),
{\lambda}=0');
saveas(gcf,'1.4_error.png');

%IQA results
PSNR= psnr(im_sense,ref);
SSIM= ssim(im_sense,ref);

disp(strcat('PSNR for SENSE Reconstruction (Rx,Ry)=(1,1),
lambda=0:',num2str(PSNR)));
disp(strcat('SSIM for SENSE Reconstruction (Rx,Ry)=(1,1),
lambda=0:',num2str(SSIM)));
```

im_sense = l2sense(imu, map, Rx, Ry, lambda) function

```
function im_sense = l2sense(imu, map, Rx, Ry, lambda)

% inputs
% imu : aliased images from all coils
% map : coil sensitivities from all coils
% Rx : acceleration rate on x-direction
% Ry : acceleration rate on y-direction
% lambda : regularization parameter

% outputs
% im_sense : l2-regularized SENSE reconstruction image

%take sizes of aliased image
[row_no col_no coil_no] = size(imu);
%take sizes for real image
[Nx Ny coil_no] = size(map);

%result image
im_sense = zeros(Nx,Ny); %size : (FOVx)x(FOVy)

%when finding source voxels for the alised voxel,
%increase aliased voxel (xp) index by step size FOVx/Rx
step_x = row_no;

%when finding source voxels for the alised voxel,
%increase aliased voxel (xp) index by step size FOVy/Ry
step_y = col_no;

%compute image with all aliased points (xp)
```

```
%index of xp is (indx,indy)
for indx = 1:row_no
    for indy = 1:col_no

        %ms is from all coils at xp
        ms = imu(indx,indy,:);
        ms = ms(:);

        %C is coil sensitivities for each xp
        C = [];

        %go through all source voxels for the aliased voxel xp
        for x = 0:Rx-1
            for y = 0:Ry-1

                %source voxel in x-direction
                indx_real = indx + Nx/2 - row_no/2 + (x * step_x);
                if (mod(indx_real, Nx)~=0)
                    indx_real = mod(indx_real, Nx);
                end

                %source voxel in y-direction
                indy_real = indy + Ny/2 - col_no/2 + (y * step_y);
                if (mod(indy_real, Ny)~=0)
                    indy_real = mod(indy_real, Ny);
                end

                %take coil sensitivities for one source voxel
                temp = map(indx_real,indy_real,:);
                temp = temp(:);

                % for one source voxel
                % C : (coil #) x (source voxel #)
                C = [C temp];
            end
        end

        %l2-regularized SENSE reconstruction

        %m is source voxels for each xp,
        %and we will solve our linear system of equations for m
        m = inv(C'*C + lambda*eye(Rx*Ry))*C'*ms;

        m(isnan(m)) = 0;
      m(isinf(m)) = 0;

        %put back calculated source voxels to their positions
        %in result image im_sense in the order of positions where
        %we took source voxel sensitivities from coil sensitivities
        for x = 0:Rx-1
            for y = 0:Ry-1

                 %source voxel in x-direction
                indx_real = indx + Nx/2 - row_no/2 + (x * step_x);
                if (mod(indx_real, Nx)~=0)
                    indx_real = mod(indx_real, Nx);
```

```
                end

                %source voxel in y-direction
                indy_real = indy + Ny/2 - col_no/2 + (y * step_y);
                if (mod(indy_real, Ny)~=0)
                    indy_real = mod(indy_real, Ny);
                end
                im_sense(indx_real,indy_real) = m((x*Ry)+y+1);

            end
        end

    end
end

im_sense(isnan(m)) = 0;
im_sense(isinf(m)) = 0;

end
```

1.5) SENSE Reconstruction without l2 Regularization

PSNR and SSIM values are really low in each acceleration case. The worst case is for (Rx,Ry)=(1,4). The reason is for using inv function in our solutions. We set Inf and NaN values to zero in our case. Therefore, we lost information with the pattern of aliasing in our images. If we used pinv function which creates pseudo-inverse of matrix and has solution for all matrices, we would not have this problem. Actually, I tried pinv function and the results were not seen as if they were cut with the pattern of aliasing.

We will remedy this problem by regularization parameter in next question.

Results

1.5

No Regularization

PSNR for SENSE Reconstruction (Rx,Ry)=(1,2):16.3678

SSIM for SENSE Reconstruction (Rx,Ry)=(1,2):0.71295

PSNR for SENSE Reconstruction (Rx,Ry)=(2,1):17.1841

SSIM for SENSE Reconstruction (Rx,Ry)=(2,1):0.73345

PSNR for SENSE Reconstruction (Rx,Ry)=(2,2):13.1258

SSIM for SENSE Reconstruction (Rx,Ry)=(2,2):0.45627

PSNR for SENSE Reconstruction (Rx,Ry)=(1,4):12.5817

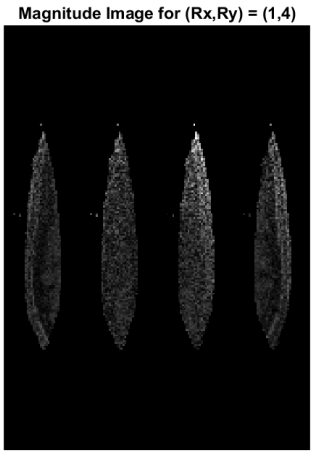
SSIM for SENSE Reconstruction (Rx,Ry)=(1,4):0.38781

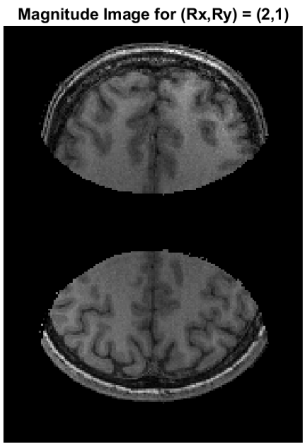
PSNR for SENSE Reconstruction (Rx,Ry)=(4,1):13.0495

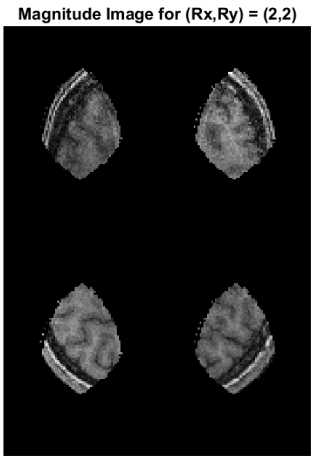
SSIM for SENSE Reconstruction (Rx,Ry)=(4,1):0.42337

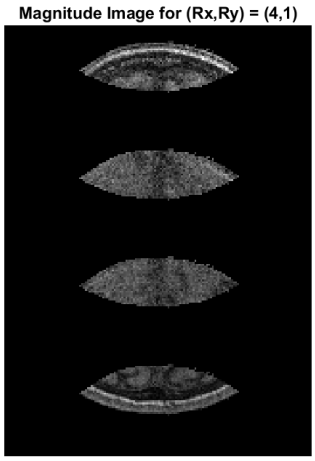
Images

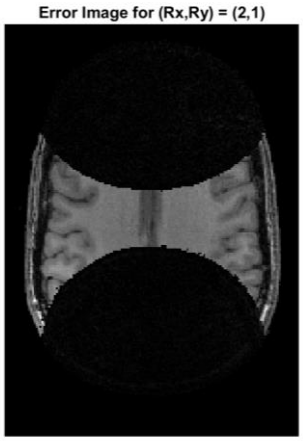
- The images are obtained with inv function.

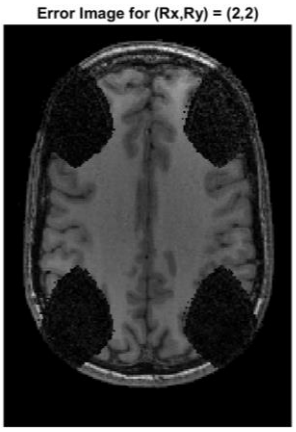


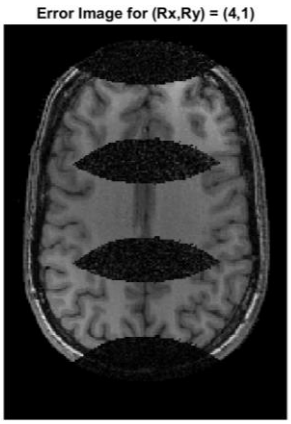












MATLAB Code

```
case '1.5'
disp('1.5');
disp('No Regularization');

load('multicoil-data.mat');
load('reference.mat');

%set regularization parameter to zero
lambda = 0;

%(Rx,Ry) = (1,2)
Rx = 1; Ry = 2;
[imu, Mu] = undersample(im, Rx, Ry);
im_sense = l2sense(imu, map1, Rx, Ry, lambda);

%normalize image
im_sense = abs(im_sense);
im_sense = im_sense/max(im_sense(:));

%magnitude image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense),[]);
```

```
title('Magnitude Image for (Rx,Ry) = (1,2)');
saveas(gcf,'1.5_mag(1,2).png');

%error image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense-ref),[]);
title('Error Image for (Rx,Ry) = (1,2)');
saveas(gcf,'1.5_error(1,2).png');

%IQA results
PSNR= psnr(im_sense,ref);
SSIM= ssim(im_sense,ref);
disp(strcat('PSNR for SENSE Reconstruction
(Rx,Ry)=(1,2):',num2str(PSNR)));
disp(strcat('SSIM for SENSE Reconstruction
(Rx,Ry)=(1,2):',num2str(SSIM)));

%(Rx,Ry) = (2,1)
Rx = 2; Ry = 1;
[imu, Mu] = undersample(im, Rx, Ry);
im_sense = l2sense(imu, map1, Rx, Ry, lambda);

%normalize image
im_sense = abs(im_sense);
im_sense = im_sense/max(im_sense(:));

%magnitude image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense),[]);
title('Magnitude Image for (Rx,Ry) = (2,1)');
saveas(gcf,'1.5_mag(2,1).png');

%error image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense-ref),[]);
title('Error Image for (Rx,Ry) = (2,1)');
saveas(gcf,'1.5_error(2,1).png');

%IQA results
PSNR= psnr(im_sense,ref);
SSIM= ssim(im_sense,ref);
disp(strcat('PSNR for SENSE Reconstruction
(Rx,Ry)=(2,1):',num2str(PSNR)));
disp(strcat('SSIM for SENSE Reconstruction
(Rx,Ry)=(2,1):',num2str(SSIM)));

%(Rx,Ry) = (2,2)
Rx = 2; Ry = 2;
[imu, Mu] = undersample(im, Rx, Ry);
im_sense = l2sense(imu, map1, Rx, Ry, lambda);

%normalize image
im_sense = abs(im_sense);
im_sense = im_sense/max(im_sense(:));

%magnitude image
```

```
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense),[]);
title('Magnitude Image for (Rx,Ry) = (2,2)');
saveas(gcf,'1.5_mag(2,2).png');

%error image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense-ref),[]);
title('Error Image for (Rx,Ry) = (2,2)');
saveas(gcf,'1.5_error(2,2).png');

%IQA results
PSNR= psnr(im_sense,ref);
SSIM= ssim(im_sense,ref);
disp(strcat('PSNR for SENSE Reconstruction
(Rx,Ry)=(2,2):',num2str(PSNR)));
disp(strcat('SSIM for SENSE Reconstruction
(Rx,Ry)=(2,2):',num2str(SSIM)));

%(Rx,Ry) = (1,4)
Rx = 1; Ry = 4;
[imu, Mu] = undersample(im, Rx, Ry);
im_sense = l2sense(imu, map1, Rx, Ry, lambda);

%normalize image
im_sense = abs(im_sense);
im_sense = im_sense/max(im_sense(:));

%magnitude image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense),[]);
title('Magnitude Image for (Rx,Ry) = (1,4)');
saveas(gcf,'1.5_mag(1,4).png');

%error image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense-ref),[]);
title('Error Image for (Rx,Ry) = (1,4)');
saveas(gcf,'1.5_error(1,4).png');

%IQA results
PSNR= psnr(im_sense,ref);
SSIM= ssim(im_sense,ref);
disp(strcat('PSNR for SENSE Reconstruction
(Rx,Ry)=(1,4):',num2str(PSNR)));
disp(strcat('SSIM for SENSE Reconstruction
(Rx,Ry)=(1,4):',num2str(SSIM)));


%(Rx,Ry) = (4,1)
Rx = 4; Ry = 1;
[imu, Mu] = undersample(im, Rx, Ry);
im_sense = l2sense(imu, map1, Rx, Ry, lambda);

%normalize image
im_sense = abs(im_sense);
```

```
im_sense = im_sense/max(im_sense(:));

%magnitude image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense),[]);
title('Magnitude Image for (Rx,Ry) = (4,1)');
saveas(gcf,'1.5_mag(4,1).png');

%error image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense-ref),[]);
title('Error Image for (Rx,Ry) = (4,1)');
saveas(gcf,'1.5_error(4,1).png');

%IQA results
PSNR= psnr(im_sense,ref);
SSIM= ssim(im_sense,ref);
disp(strcat('PSNR for SENSE Reconstruction
(Rx,Ry)=(4,1):',num2str(PSNR)));
disp(strcat('SSIM for SENSE Reconstruction
(Rx,Ry)=(4,1):',num2str(SSIM)));
```

1.6) SENSE Reconstruction with l2 Regularization

By trial and error, I chose regularization parameter to be 1. We used only map1 for this question.

When acceleration rate increases, we have lower PSNR and SSIM values and the artifacts become more visible. Similar to previous results, the worst SSIM is for (Rx,Ry)=(1,4), and then (Rx,Ry)=(4,1).

The aliasing patterns can also be seen in error images. Intersection of aliased artifacts end in more error.

Results

1.6

SENSE Reconstruction With Regularization Parameter

PSNR for SENSE Reconstruction (Rx,Ry)=(1,2), lambda=1:34.8899

SSIM for SENSE Reconstruction (Rx,Ry)=(1,2), lambda=1:0.95579

PSNR for SENSE Reconstruction (Rx,Ry)=(2,1), lambda=1:37.5305

SSIM for SENSE Reconstruction (Rx,Ry)=(2,1), lambda=1:0.97012

PSNR for SENSE Reconstruction (Rx,Ry)=(2,2), lambda=1:30.9409

SSIM for SENSE Reconstruction (Rx,Ry)=(2,2), lambda=1:0.90062

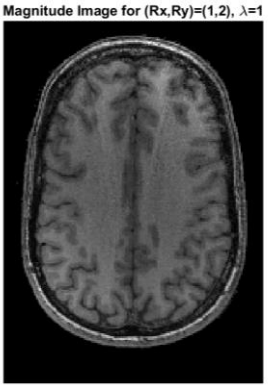
PSNR for SENSE Reconstruction (Rx,Ry)=(1,4), lambda=1:25.5364

SSIM for SENSE Reconstruction (Rx,Ry)=(1,4), lambda=1:0.77075

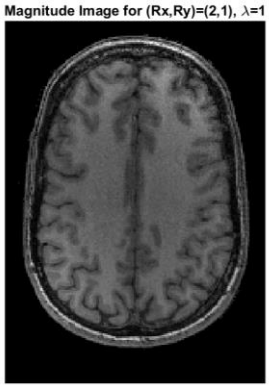
PSNR for SENSE Reconstruction (Rx,Ry)=(4,1), lambda=1:24.1885

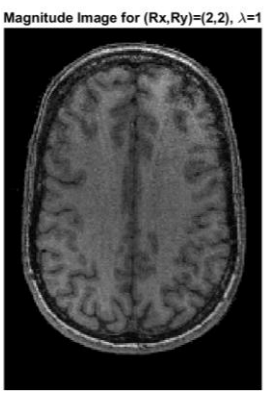
SSIM for SENSE Reconstruction (Rx,Ry)=(4,1), lambda=1:0.7829

Images


Magnitude Image for (Rx,Ry)=(1,2), λ=1


Magnitude Image for (Rx,Ry)=(1,4), λ=1

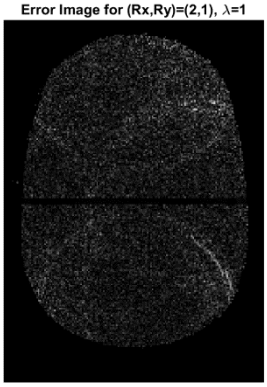

Magnitude Image for (Rx,Ry)=(2,1), λ=1

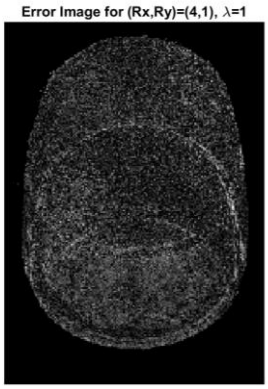

Magnitude Image for (Rx,Ry)=(2,2), λ=1


Magnitude Image for (Rx,Ry)=(4,1), λ=1


Error Image for (Rx,Ry)=(1,2), λ=1


Error Image for (Rx,Ry)=(1,4), λ=1


Error Image for (Rx,Ry)=(2,1), λ=1



MATLAB Code

```
case '1.6'

disp('1.6');
disp('SENSE Reconstruction With Regularization Parameter');

load('multicoil-data.mat');
load('reference.mat');

%set regularization parameter to 1
lambda = 1;

%(Rx,Ry) = (1,2)
Rx = 1; Ry = 2;
[imu, Mu] = undersample(im, Rx, Ry);
im_sense = l2sense(imu, map1, Rx, Ry, lambda);

%normalize image
im_sense = abs(im_sense);
im_sense = im_sense/max(im_sense(:));

%magnitude image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense),[]);
title('Magnitude Image for (Rx,Ry)=(1,2), {\lambda}=1');
saveas(gcf,'1.6_mag(1,2).png');

%error image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense-ref),[]);
title('Error Image for (Rx,Ry)=(1,2), {\lambda}=1');
saveas(gcf,'1.6_error(1,2).png');

%IQA results
PSNR= psnr(im_sense,ref);
SSIM= ssim(im_sense,ref);
```

```
disp(strcat('PSNR for SENSE Reconstruction (Rx,Ry)=(1,2),
lambda=1:',num2str(PSNR)));
disp(strcat('SSIM for SENSE Reconstruction (Rx,Ry)=(1,2),
lambda=1:',num2str(SSIM)));

%(Rx,Ry) = (2,1)
Rx = 2; Ry = 1;
[imu, Mu] = undersample(im, Rx, Ry);
im_sense = l2sense(imu, map1, Rx, Ry, lambda);

%normalize image
im_sense = abs(im_sense);
im_sense = im_sense/max(im_sense(:));

%magnitude image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense),[]);
title('Magnitude Image for (Rx,Ry)=(2,1), {\lambda}=1');
saveas(gcf,'1.6_mag(2,1).png');

%error image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense-ref),[]);
title('Error Image for (Rx,Ry)=(2,1), {\lambda}=1');
saveas(gcf,'1.6_error(2,1).png');

%IQA results
PSNR= psnr(im_sense,ref);
SSIM= ssim(im_sense,ref);
disp(strcat('PSNR for SENSE Reconstruction (Rx,Ry)=(2,1),
lambda=1:',num2str(PSNR)));
disp(strcat('SSIM for SENSE Reconstruction (Rx,Ry)=(2,1),
lambda=1:',num2str(SSIM)));

%(Rx,Ry) = (2,2)
Rx = 2; Ry = 2;
[imu, Mu] = undersample(im, Rx, Ry);
im_sense = l2sense(imu, map1, Rx, Ry, lambda);

%normalize image
im_sense = abs(im_sense);
im_sense = im_sense/max(im_sense(:));

%magnitude image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense),[]);
title('Magnitude Image for (Rx,Ry)=(2,2), {\lambda}=1');
saveas(gcf,'1.6_mag(2,2).png');

%error image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense-ref),[]);
title('Error Image for (Rx,Ry)=(2,2), {\lambda}=1');
saveas(gcf,'1.6_error(2,2).png');

%IQA results
```

```
PSNR= psnr(im_sense,ref);
SSIM= ssim(im_sense,ref);
disp(strcat('PSNR for SENSE Reconstruction (Rx,Ry)=(2,2),
lambda=1:',num2str(PSNR)));
disp(strcat('SSIM for SENSE Reconstruction (Rx,Ry)=(2,2),
lambda=1:',num2str(SSIM)));

%(Rx,Ry) = (1,4)
Rx = 1; Ry = 4;
[imu, Mu] = undersample(im, Rx, Ry);
im_sense = l2sense(imu, map1, Rx, Ry, lambda);

%normalize image
im_sense = abs(im_sense);
im_sense = im_sense/max(im_sense(:));

%magnitude image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense),[]);
title('Magnitude Image for (Rx,Ry)=(1,4), {\lambda}=1');
saveas(gcf,'1.6_mag(1,4).png');

%error image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense-ref),[]);
title('Error Image for (Rx,Ry)=(1,4), {\lambda}=1');
saveas(gcf,'1.6_error(1,4).png');

%IQA results
PSNR= psnr(im_sense,ref);
SSIM= ssim(im_sense,ref);
disp(strcat('PSNR for SENSE Reconstruction (Rx,Ry)=(1,4),
lambda=1:',num2str(PSNR)));
disp(strcat('SSIM for SENSE Reconstruction (Rx,Ry)=(1,4),
lambda=1:',num2str(SSIM)));

%(Rx,Ry) = (4,1)
Rx = 4; Ry = 1;
[imu, Mu] = undersample(im, Rx, Ry);
im_sense = l2sense(imu, map1, Rx, Ry, lambda);

%normalize image
im_sense = abs(im_sense);
im_sense = im_sense/max(im_sense(:));

%magnitude image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense),[]);
title('Magnitude Image for (Rx,Ry)=(4,1), {\lambda}=1');
saveas(gcf,'1.6_mag(4,1).png');

%error image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense-ref),[]);
title('Error Image for (Rx,Ry)=(4,1), {\lambda}=1');
saveas(gcf,'1.6_error(4,1).png');
```

```
%IQA results
PSNR= psnr(im_sense,ref);
SSIM= ssim(im_sense,ref);
disp(strcat('PSNR for SENSE Reconstruction (Rx,Ry)=(4,1),
lambda=1:',num2str(PSNR)));
disp(strcat('SSIM for SENSE Reconstruction (Rx,Ry)=(4,1),
lambda=1:',num2str(SSIM)));
```

1.7) Importance of Accurate Coil Sensitivities

In Q1.1, we stated that the coil sensitivities for map2 are close to each other compared to map1. This caused problems for SoS and OLC reconstructions. Here, we see the same problem here. Visually, the artifacts for (Rx,Ry)=(2,2), (Rx,Ry)=(1,4), and (Rx,Ry)=(4,1) are not acceptable. The PSNR and SSIM values are also lower compared to map1 results. We can see that SENSE reconstruction is strongly dependent on coil sensitivities.

Results

1.7

Importance of Accurate Coil Sensitivities

PSNR for SENSE Reconstruction (Rx,Ry)=(1,2), lambda=1:28.4905

SSIM for SENSE Reconstruction (Rx,Ry)=(1,2), lambda=1:0.92201

PSNR for SENSE Reconstruction (Rx,Ry)=(2,1), lambda=1:27.125

SSIM for SENSE Reconstruction (Rx,Ry)=(2,1), lambda=1:0.92276

PSNR for SENSE Reconstruction (Rx,Ry)=(2,2), lambda=1:24.3211

SSIM for SENSE Reconstruction (Rx,Ry)=(2,2), lambda=1:0.8543

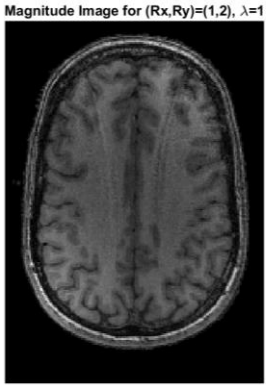
PSNR for SENSE Reconstruction (Rx,Ry)=(1,4), lambda=1:18.6429

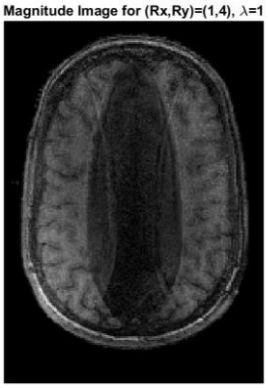
SSIM for SENSE Reconstruction (Rx,Ry)=(1,4), lambda=1:0.70048

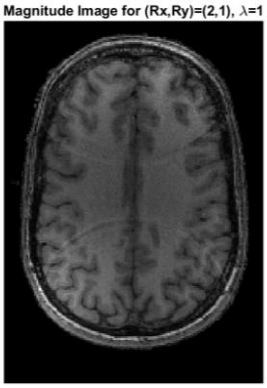
PSNR for SENSE Reconstruction (Rx,Ry)=(4,1), lambda=1:22.6616

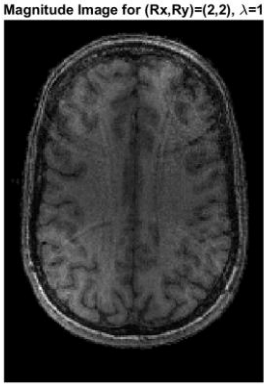
SSIM for SENSE Reconstruction (Rx,Ry)=(4,1), lambda=1:0.75711

Images


Magnitude Image for (Rx,Ry)=(1,2), λ=1


Magnitude Image for (Rx,Ry)=(1,4), λ=1

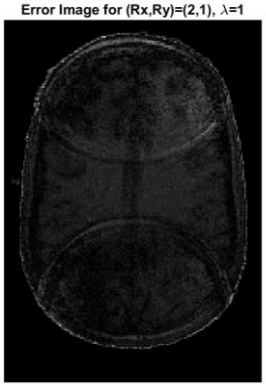

Magnitude Image for (Rx,Ry)=(2,1), λ=1

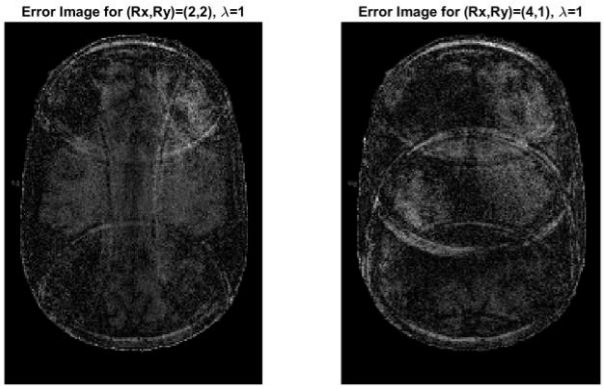

Magnitude Image for (Rx,Ry)=(2,2), λ=1


Magnitude Image for (Rx,Ry)=(4,1), λ=1


Error Image for (Rx,Ry)=(1,2), λ=1


Error Image for (Rx,Ry)=(1,4), λ=1


Error Image for (Rx,Ry)=(2,1), λ=1



MATLAB Code

```
case '1.7'

disp('1.7');
disp('Importance of Accurate Coil Sensitivities');

load('multicoil-data.mat');
load('reference.mat');

%set regularization parameter to 1
lambda = 1;

%(Rx,Ry) = (1,2)
Rx = 1; Ry = 2;
[imu, Mu] = undersample(im, Rx, Ry);
im_sense = l2sense(imu, map2, Rx, Ry, lambda);

%normalize image
im_sense = abs(im_sense);
im_sense = im_sense/max(im_sense(:));

%magnitude image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense),[]);
title('Magnitude Image for (Rx,Ry)=(1,2), {\lambda}=1');
saveas(gcf,'1.7_mag(1,2).png');

%error image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense-ref),[]);
title('Error Image for (Rx,Ry)=(1,2), {\lambda}=1');
saveas(gcf,'1.7_error(1,2).png');

%IQA results
PSNR= psnr(im_sense,ref);
SSIM= ssim(im_sense,ref);
```

```
disp(strcat('PSNR for SENSE Reconstruction (Rx,Ry)=(1,2),
lambda=1:',num2str(PSNR)));
disp(strcat('SSIM for SENSE Reconstruction (Rx,Ry)=(1,2),
lambda=1:',num2str(SSIM)));

%(Rx,Ry) = (2,1)
Rx = 2; Ry = 1;
[imu, Mu] = undersample(im, Rx, Ry);
im_sense = l2sense(imu, map2, Rx, Ry, lambda);

%normalize image
im_sense = abs(im_sense);
im_sense = im_sense/max(im_sense(:));

%magnitude image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense),[]);
title('Magnitude Image for (Rx,Ry)=(2,1), {\lambda}=1');
saveas(gcf,'1.7_mag(2,1).png');

%error image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense-ref),[]);
title('Error Image for (Rx,Ry)=(2,1), {\lambda}=1');
saveas(gcf,'1.7_error(2,1).png');

%IQA results
PSNR= psnr(im_sense,ref);
SSIM= ssim(im_sense,ref);
disp(strcat('PSNR for SENSE Reconstruction (Rx,Ry)=(2,1),
lambda=1:',num2str(PSNR)));
disp(strcat('SSIM for SENSE Reconstruction (Rx,Ry)=(2,1),
lambda=1:',num2str(SSIM)));

%(Rx,Ry) = (2,2)
Rx = 2; Ry = 2;
[imu, Mu] = undersample(im, Rx, Ry);
im_sense = l2sense(imu, map2, Rx, Ry, lambda);

%normalize image
im_sense = abs(im_sense);
im_sense = im_sense/max(im_sense(:));

%magnitude image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense),[]);
title('Magnitude Image for (Rx,Ry)=(2,2), {\lambda}=1');
saveas(gcf,'1.7_mag(2,2).png');

%error image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense-ref),[]);
title('Error Image for (Rx,Ry)=(2,2), {\lambda}=1');
saveas(gcf,'1.7_error(2,2).png');

%IQA results
```

```
PSNR= psnr(im_sense,ref);
SSIM= ssim(im_sense,ref);
disp(strcat('PSNR for SENSE Reconstruction (Rx,Ry)=(2,2),
lambda=1:',num2str(PSNR)));
disp(strcat('SSIM for SENSE Reconstruction (Rx,Ry)=(2,2),
lambda=1:',num2str(SSIM)));

%(Rx,Ry) = (1,4)
Rx = 1; Ry = 4;
[imu, Mu] = undersample(im, Rx, Ry);
im_sense = l2sense(imu, map2, Rx, Ry, lambda);

%normalize image
im_sense = abs(im_sense);
im_sense = im_sense/max(im_sense(:));

%magnitude image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense),[]);
title('Magnitude Image for (Rx,Ry)=(1,4), {\lambda}=1');
saveas(gcf,'1.7_mag(1,4).png');

%error image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense-ref),[]);
title('Error Image for (Rx,Ry)=(1,4), {\lambda}=1');
saveas(gcf,'1.7_error(1,4).png');

%IQA results
PSNR= psnr(im_sense,ref);
SSIM= ssim(im_sense,ref);
disp(strcat('PSNR for SENSE Reconstruction (Rx,Ry)=(1,4),
lambda=1:',num2str(PSNR)));
disp(strcat('SSIM for SENSE Reconstruction (Rx,Ry)=(1,4),
lambda=1:',num2str(SSIM)));

%(Rx,Ry) = (4,1)
Rx = 4; Ry = 1;
[imu, Mu] = undersample(im, Rx, Ry);
im_sense = l2sense(imu, map2, Rx, Ry, lambda);

%normalize image
im_sense = abs(im_sense);
im_sense = im_sense/max(im_sense(:));

%magnitude image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense),[]);
title('Magnitude Image for (Rx,Ry)=(4,1), {\lambda}=1');
saveas(gcf,'1.7_mag(4,1).png');

%error image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(im_sense-ref),[]);
title('Error Image for (Rx,Ry)=(4,1), {\lambda}=1');
saveas(gcf,'1.7_error(4,1).png');
```

```
%IQA results
PSNR= psnr(im_sense,ref);
SSIM= ssim(im_sense,ref);
disp(strcat('PSNR for SENSE Reconstruction (Rx,Ry)=(4,1),
lambda=1:',num2str(PSNR)));
disp(strcat('SSIM for SENSE Reconstruction (Rx,Ry)=(4,1),
lambda=1:',num2str(SSIM)));
```

1.8) *g*-Factor for *l*2-regularized SENSE

Derivation for the g-factor for l2-regularized SENSE:

$$g = \sqrt{[C^H C]_{i,i}[AA^H]_{i,i}}$$

$$A = (C^H C + \lambda I)^{-1} C^H$$

$$A^H = ((C^H C + \lambda I)^{-1} C^H)^H$$

$$= ((C^H C + \lambda I)^{-1} C^H)^H$$

$$= C((C^H C + \lambda I)^{-1})^H$$

Hermitian of inverse of matrix is equal to inverse of Hermitian of the matrix:

$$A^H = C((C^H C + \lambda I)^H)^{-1}$$

Hermitian of diagonal matrix λI is equal to itself:

$$A^H = C(C^H C + \lambda I)^{-1}$$

Lastly, the covariance for sense reconstrion is as follows:

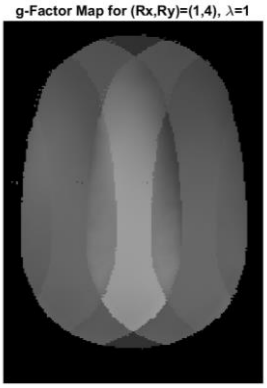
$$Cov(m) = AA^H = (C^H C + \lambda I)^{-1}(C^H C)(C^H C + \lambda I)^{-1}$$

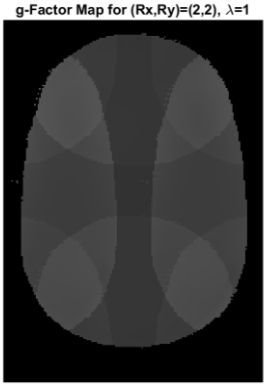
1.9) *g*-Factor Map

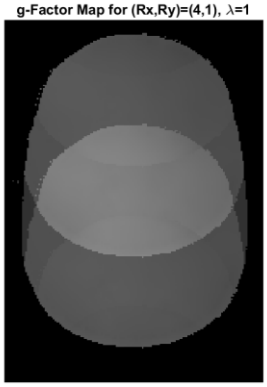
We used our derivation in Q1.8 in gfactor function. The images for g-factor maps are displayed in range of [0 5].

The artifacts seen in Q1.6 are corresponding to the g-factor maps for map1. This can be seen from magnitude images from Q1.6, but it is seen better in error images of Q1.6.

Images







MATLAB Code

```
case '1.9'
disp('1.9');

disp('g-Factor Map');

load('multicoil-data.mat');

%set regularization parameter to 1
lambda = 1;

%map1 is used for g-factor

%(Rx,Ry) = (1,2)
Rx = 1; Ry = 2;
g = gfactor(map1, Rx, Ry, lambda);

%g-factor map
```

```
figure; set(gcf, 'WindowState', 'maximized');
imshow(real(g),'DisplayRange',[0 5]);
title('g-Factor Map for (Rx,Ry)=(1,2), {\lambda}=1');
saveas(gcf,'1.9_mag(1,2).png');

%(Rx,Ry) = (2,1)
Rx = 2; Ry = 1;
g = gfactor(map1, Rx, Ry, lambda);

%g-factor map
figure; set(gcf, 'WindowState', 'maximized');
imshow(real(g),'DisplayRange',[0 5]);
title('g-Factor Map for (Rx,Ry)=(2,1), {\lambda}=1');
saveas(gcf,'1.9_mag(2,1).png');

%(Rx,Ry) = (2,2)
Rx = 2; Ry = 2;
g = gfactor(map1, Rx, Ry, lambda);

%g-factor map
figure; set(gcf, 'WindowState', 'maximized');
imshow(real(g),'DisplayRange',[0 5]);
title('g-Factor Map for (Rx,Ry)=(2,2), {\lambda}=1');
saveas(gcf,'1.9_mag(2,2).png');

%(Rx,Ry) = (1,4)
Rx = 1; Ry = 4;
g = gfactor(map1, Rx, Ry, lambda);

%g-factor map
figure; set(gcf, 'WindowState', 'maximized');
imshow(real(g),'DisplayRange',[0 5]);
title('g-Factor Map for (Rx,Ry)=(1,4), {\lambda}=1');
saveas(gcf,'1.9_mag(1,4).png');

%(Rx,Ry) = (4,1)
Rx = 4; Ry = 1;
g = gfactor(map1, Rx, Ry, lambda);

%g-factor map
figure; set(gcf, 'WindowState', 'maximized');
imshow(real(g),'DisplayRange',[0 5]);
title('g-Factor Map for (Rx,Ry)=(4,1), {\lambda}=1');
saveas(gcf,'1.9_mag(4,1).png');
```

g = gfactor(map, Rx, Ry, lambda) function

```
function g = gfactor(map, Rx, Ry, lambda)

% inputs
% map : coil sensitivities from all coils
% Rx : acceleration rate on x-direction
% Ry : acceleration rate on y-direction
% lambda : regularization parameter
```

```
% outputs
% g : g-factor

%take sizes of coil sensitivities
[Nx Ny coil_no] = size(map);

%take sizes for images to be aliased due to undersampling
row_no = Nx/Rx;
col_no = Ny/Ry;

step_x = row_no; %increase source voxel indices as much as FOVx/Rx
step_y = col_no; %increase source voxel indices as much as FOVy/Ry

%go through all aliased points (xp)
for indx = 1:row_no
    for indy = 1:col_no

        %coil sensitivities for each xp
        C = [];

        for x = 0:Rx-1
            for y = 0:Ry-1

                %x-index of source voxel for xp
                indx_real = indx + Nx/2 - row_no/2 + (x * step_x);
                if (mod(indx_real, Nx)~=0)
                    indx_real = mod(indx_real, Nx);
                end

                %y-index of source voxel for xp
                indy_real = indy + Ny/2 - col_no/2 + (y * step_y);
                if (mod(indy_real, Ny)~=0)
                    indy_real = mod(indy_real, Ny);
                end

                %take coil sensitivities at index of source voxel
                temp = map(indx_real,indy_real,:);
                temp = temp(:);

                %collect coil sensitivities for all source voxels in
C
                C = [C temp];
            end
        end

        index = 1;
        for x = 0:Rx-1
            for y = 0:Ry-1

                %x-index of source voxel for xp
                indx_real = indx + Nx/2 - row_no/2 + (x * step_x);
                if (mod(indx_real, Nx)~=0)
                    indx_real = mod(indx_real, Nx);
                end
```

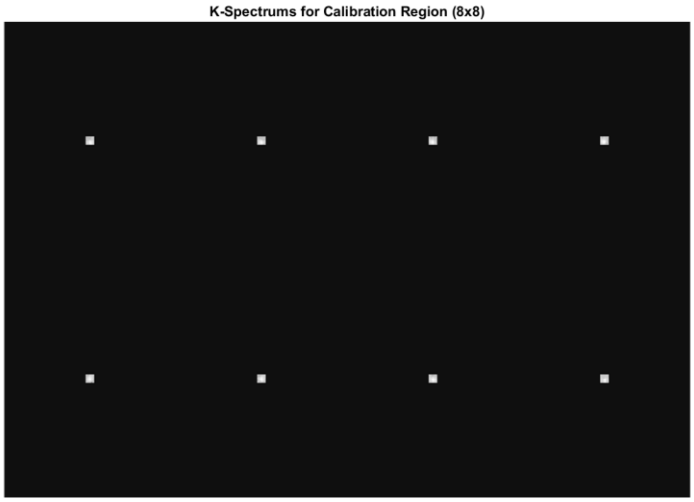
```
                %y-index of source voxel for xp
                indy_real = indy + Ny/2 - col_no/2 + (y * step_y);
                if (mod(indy_real, Ny)~=0)
                    indy_real = mod(indy_real, Ny);
                end

                %take coil sensitivities corresponding to real image
voxel
                Cx = map(indx_real,indy_real,:);
                Cx = Cx(:);
                %find covariance from olc reconstruction
                cov_olc = Cx'*Cx;

                %C is coil sensitivities for all source voxels for
xp
                %find covariance from sense reconstruction
                cov_temp = inv(C'*C + lambda*eye(Rx*Ry));
                cov_temp(isnan(cov_temp)) = 0;
                cov_temp(isinf(cov_temp)) = 0;


                cov_sense = cov_temp * C' * C * cov_temp;

                %if all of the original sensitivity maps have a
value zero at a voxel,
                %set g-factor to 0 for that voxel
                if (sum(Cx(:))==0)
                    g_temp = 0;
                else
                    g_temp = sqrt( cov_olc * cov_sense(index,index)
);
                end

                %put g-factor value to the index for the real image
voxel
                g(indx_real,indy_real) = g_temp;
                %increment index in order to to go through all
source voxels for xp
                index = index + 1;
            end
        end
    end
end

g(isnan(g)) = 0;
g(isinf(g)) = 0;

end
```

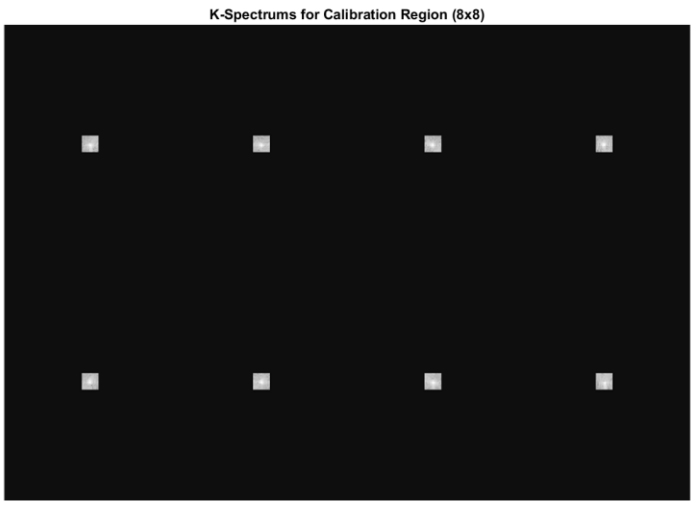
PART II – GRAPPA

2.1) Calibration Image

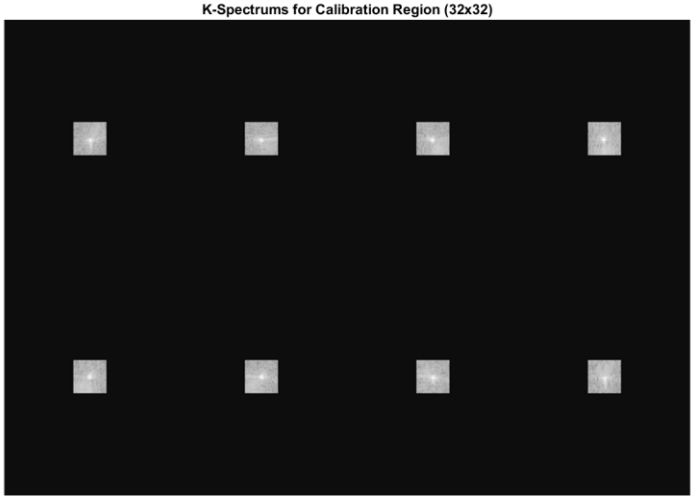
We zeroed out the k-space data out of calibration region. This reduction in k-space data caused loss of resolution in magnitude images. The smaller calibration region is, the smoother the magnitude image becomes. This is due to loss of high frequency components of k-space data.

Images

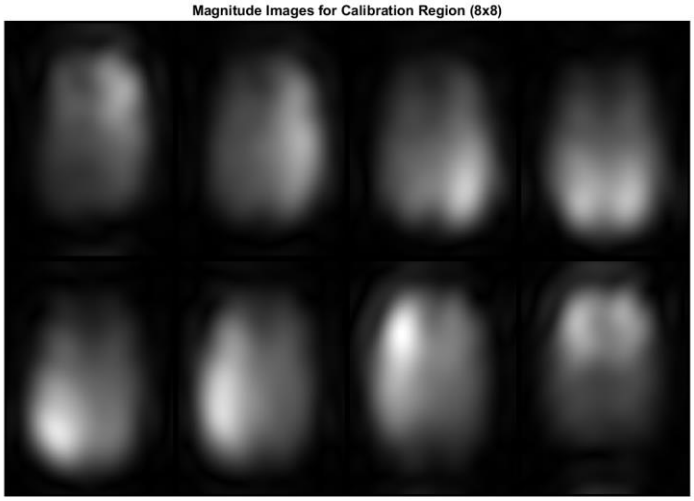




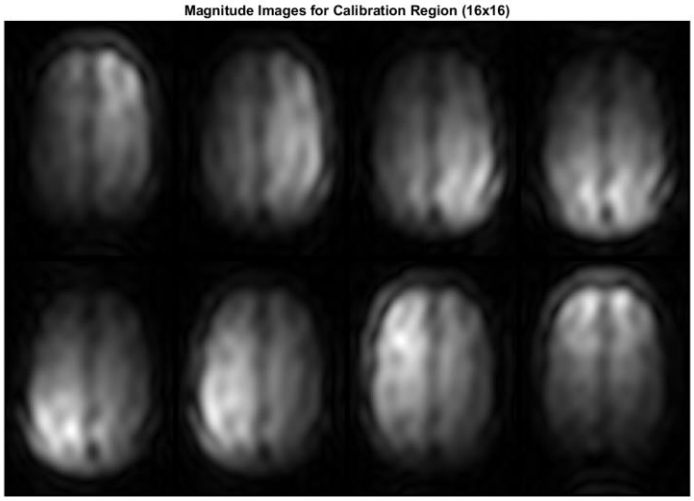
K-Spectrums for Calibration Region (32x32)

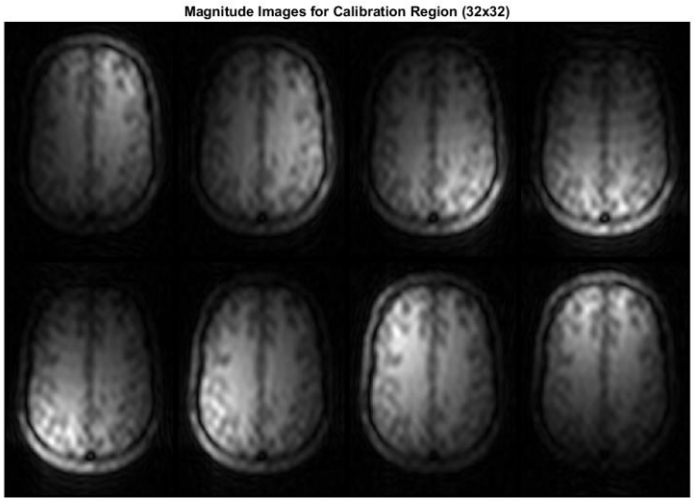


Magnitude Images for Calibration Region (8x8)



Magnitude Images for Calibration Region (16x16)





MATLAB Code

```
case '2.1'
disp('2.1');
disp('Calibration Images');

load('multicoil-data.mat');

%(calibx, caliby) = (8,8)
calibx = 8; caliby = 8;
[imc, Mc] = imcalib(im, calibx, caliby);

figure; set(gcf, 'WindowState', 'maximized');
montage(abs(imc),'displayRange',[], 'Size', [2 4], 'BorderSize', 5);
title('Magnitude Images for Calibration Region (8x8)');
saveas(gcf,'2.1_mag(8,8).png');

figure; set(gcf, 'WindowState', 'maximized');
montage(log(abs(Mc)+1),'displayRange',[], 'Size', [2 4],
'BorderSize', 5);
title('K-Spectrums for Calibration Region (8x8)');
saveas(gcf,'2.1_kspace(8,8).png');

%(calibx, caliby) = (16,16)
calibx = 16; caliby = 16;
[imc, Mc] = imcalib(im, calibx, caliby)

figure; set(gcf, 'WindowState', 'maximized');
montage(abs(imc),'displayRange',[], 'Size', [2 4], 'BorderSize', 5);
title('Magnitude Images for Calibration Region (16x16)');
saveas(gcf,'2.1_mag(16,16).png');

figure; set(gcf, 'WindowState', 'maximized');
montage(log(abs(Mc)+1),'displayRange',[], 'Size', [2 4],
'BorderSize', 5);
title('K-Spectrums for Calibration Region (8x8)');
saveas(gcf,'2.1_kspace(16,16).png');
```

```
%(calibx, caliby) = (32,32)
calibx = 32; caliby = 32;
[imc, Mc] = imcalib(im, calibx, caliby)

figure; set(gcf, 'WindowState', 'maximized');
montage(abs(imc),'displayRange',[], 'Size', [2 4], 'BorderSize', 5);
title('Magnitude Images for Calibration Region (32x32)');
saveas(gcf,'2.1_mag(32,32).png');

figure; set(gcf, 'WindowState', 'maximized');
montage(log(abs(Mc)+1),'displayRange',[], 'Size', [2 4],
'BorderSize', 5);
title('K-Spectrums for Calibration Region (32x32)');
saveas(gcf,'2.1_kspace(32,32).png');
```

[imc, Mc] = imcalib(im, calibx, caliby) function

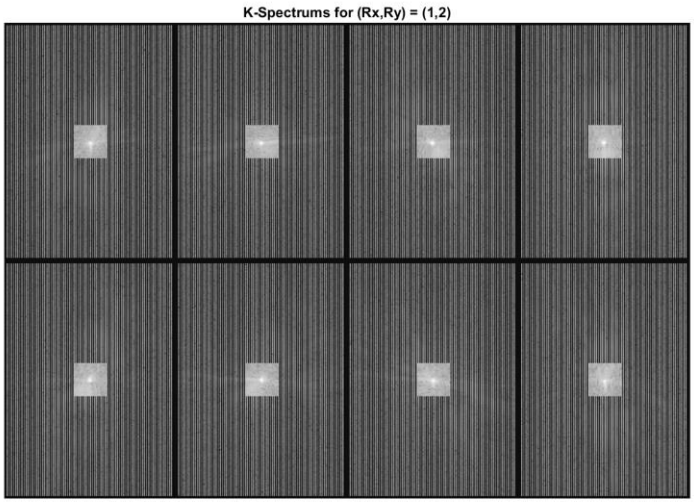
```
function [imc, Mc] = imcalib(im, calibx, caliby)

% inputs
% im : images from all coils
% calibx : size of calibration region in x-direction
% caliby : size of calibration ragion in y-direction

% outputs
% Mc : k-space data of calibration region
% imc : image computed with Mc

for coil_no = 1:8

    %2D FFT
    d = fft2c(im(:,:,coil_no));

    %k-space data will be zeroed out everywhere
    %except central region with size of (calibx x caliby)
    d(1:(end/2 - calibx/2),:) = 0;
    d((end/2 + calibx/2+1):end,:) = 0;
    d(:,1:(end/2 - caliby/2)) = 0;
    d(:,(end/2 + caliby/2+1):end) = 0;

    Mc(:,:,coil_no) = d;
    imc(:,:,coil_no) = ifft2c(Mc(:,:,coil_no));

end

end
```

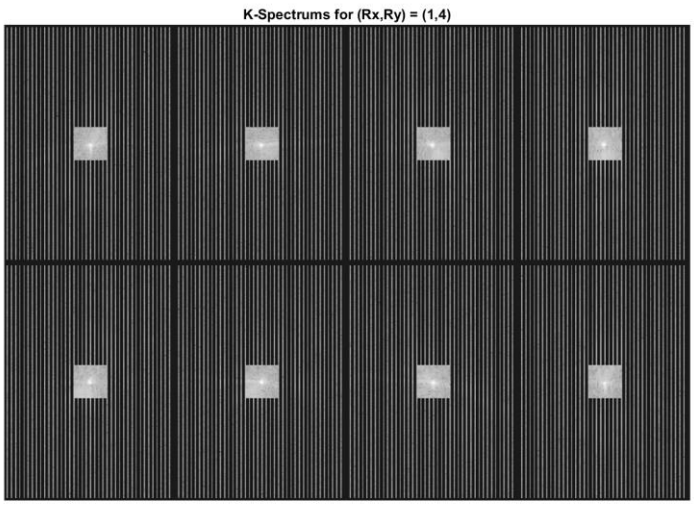
2.2) Undersampled Images with Calibration Region

We created undersampled k-space data here. Unlike Part 1, the undersampled parts are not discarded but zeroed out. The central part of k-space data comes from calibration regions. For Rx, we have loss of k-space data in kx-direction and this causes aliasing in x-direction. Similarly, the same happens for Ry and y-direction.

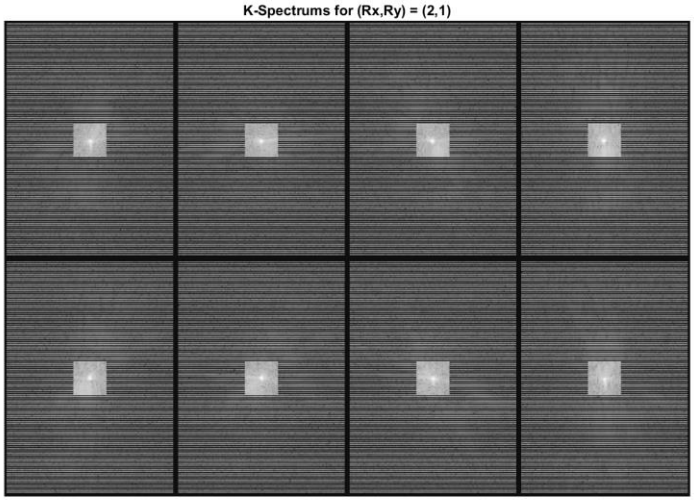
We have the calibration region in our k-space data, which gives most of the information about our magnitude images. We have undersampling in high frequencies, therefore aliasing happens for high frequency component in our image. In other words, aliasing artifacts are coming from edges of the image.

Images

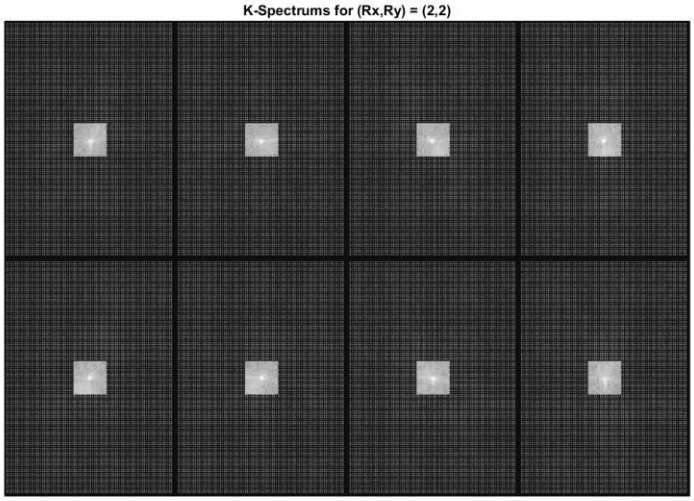




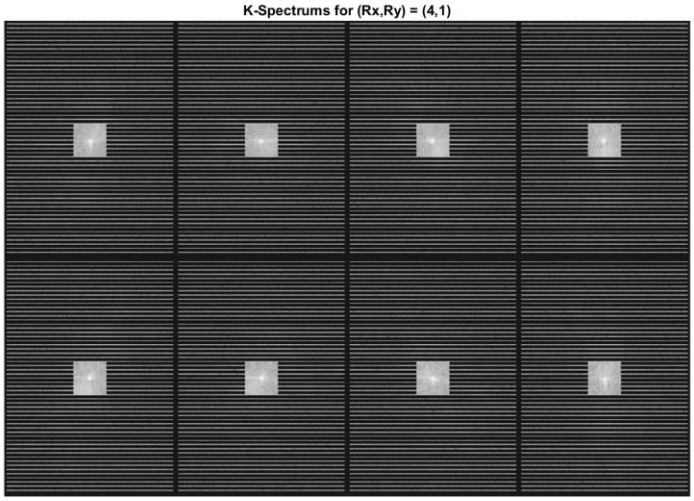
K-Spectrums for (Rx,Ry) = (2,1)



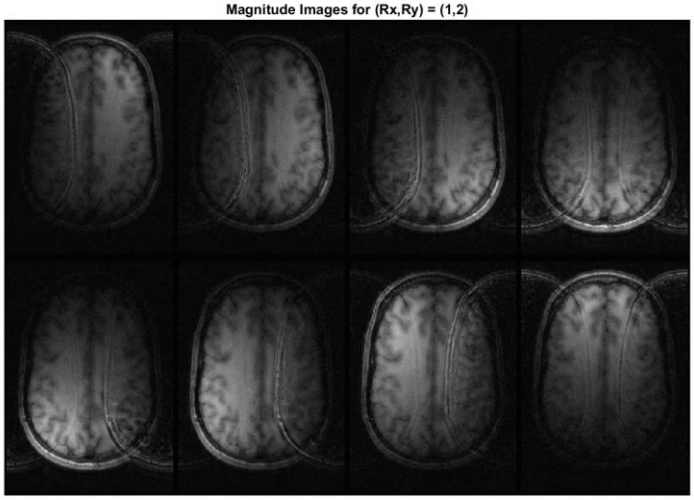
K-Spectrums for (Rx,Ry) = (2,2)



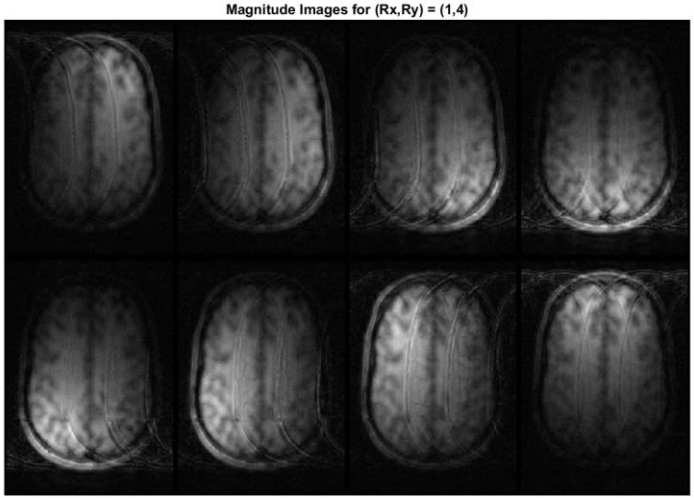
K-Spectrums for (Rx,Ry) = (4,1)



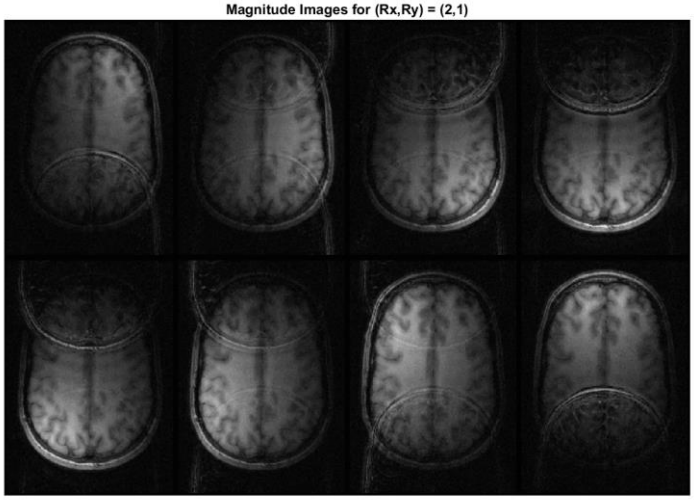
Magnitude Images for (Rx,Ry) = (1,2)

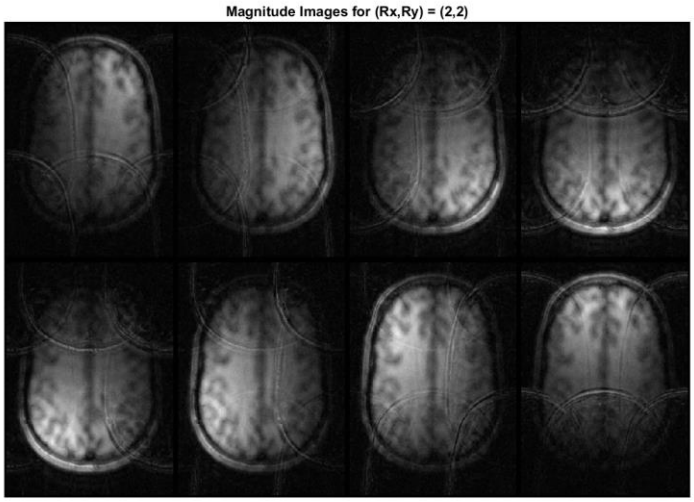


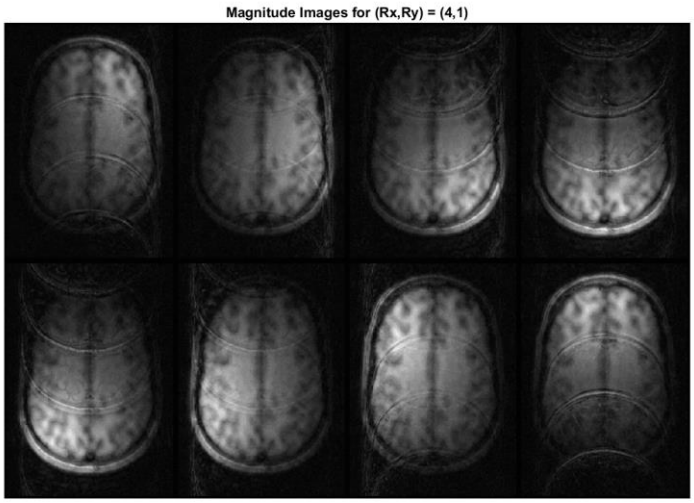
Magnitude Images for (Rx,Ry) = (1,4)



Magnitude Images for (Rx,Ry) = (2,1)







MATLAB Code

```
case '2.2'
disp('2.2');
disp('Undersampled Images with Calibration Region');

load('multicoil-data.mat');

%(calibx,caliby) = (32,32)
calibx = 32; caliby = 32;

%(Rx, Ry) = (1,2)
Rx = 1; Ry = 2;
[imu, Mu] = undersamplecalib(im, Rx, Ry, calibx, caliby);

figure; set(gcf, 'WindowState', 'maximized');
montage(abs(imu),'displayRange',[], 'Size', [2 4], 'BorderSize', 5);
title('Magnitude Images for (Rx,Ry) = (1,2)');
saveas(gcf,'2.2_mag(1,2).png');
```

```
figure; set(gcf, 'WindowState', 'maximized');
montage(log(abs(Mu)+1),'displayRange',[], 'Size', [2 4],
'BorderSize', 5);
title('K-Spectrums for (Rx,Ry) = (1,2)');
saveas(gcf,'2.2_kspace(1,2).png');

%(Rx, Ry) = (2,1)
Rx = 2; Ry = 1;
[imu, Mu] = undersamplecalib(im, Rx, Ry, calibx, caliby);
figure; set(gcf, 'WindowState', 'maximized');
montage(abs(imu),'displayRange',[], 'Size', [2 4], 'BorderSize', 5);
title('Magnitude Images for (Rx,Ry) = (2,1)');
saveas(gcf,'2.2_mag(2,1).png');

figure; set(gcf, 'WindowState', 'maximized');
montage(log(abs(Mu)+1),'displayRange',[], 'Size', [2 4],
'BorderSize', 5);
title('K-Spectrums for (Rx,Ry) = (2,1)');
saveas(gcf,'2.2_kspace(2,1).png');

%(Rx, Ry) = (2,2)
Rx = 2; Ry = 2;
[imu, Mu] = undersamplecalib(im, Rx, Ry, calibx, caliby);
figure; set(gcf, 'WindowState', 'maximized');
montage(abs(imu),'displayRange',[], 'Size', [2 4], 'BorderSize', 5);
title('Magnitude Images for (Rx,Ry) = (2,2)');
saveas(gcf,'2.2_mag(2,2).png');

figure; set(gcf, 'WindowState', 'maximized');
montage(log(abs(Mu)+1),'displayRange',[], 'Size', [2 4],
'BorderSize', 5);
title('K-Spectrums for (Rx,Ry) = (2,2)');
saveas(gcf,'2.2_kspace(2,2).png');

%(Rx, Ry) = (1,4)
Rx = 1; Ry = 4;
[imu, Mu] = undersamplecalib(im, Rx, Ry, calibx, caliby);
figure; set(gcf, 'WindowState', 'maximized');
montage(abs(imu),'displayRange',[], 'Size', [2 4], 'BorderSize', 5);
title('Magnitude Images for (Rx,Ry) = (1,4)');
saveas(gcf,'2.2_mag(1,4).png');

figure; set(gcf, 'WindowState', 'maximized');
montage(log(abs(Mu)+1),'displayRange',[], 'Size', [2 4],
'BorderSize', 5);
title('K-Spectrums for (Rx,Ry) = (1,4)');
saveas(gcf,'2.2_kspace(1,4).png');

%(Rx, Ry) = (4,1)
Rx = 4; Ry = 1;
[imu, Mu] = undersamplecalib(im, Rx, Ry, calibx, caliby);

figure; set(gcf, 'WindowState', 'maximized');
montage(abs(imu),'displayRange',[], 'Size', [2 4], 'BorderSize', 5);
title('Magnitude Images for (Rx,Ry) = (4,1)');
saveas(gcf,'2.2_mag(4,1).png');
```

```
figure; set(gcf, 'WindowState', 'maximized');
montage(log(abs(Mu)+1),'displayRange',[], 'Size', [2 4],
'BorderSize', 5);
title('K-Spectrums for (Rx,Ry) = (4,1)');
saveas(gcf,'2.2_kspace(4,1).png');
```

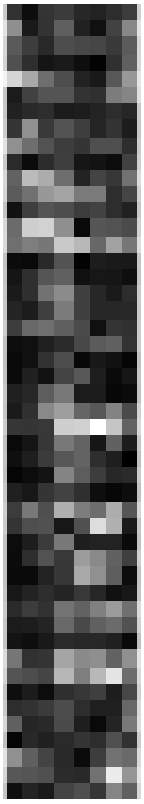
[imu, Mu] = undersamplecalib(im, Rx, Ry, calibx, caliby) function

```
function [imu, Mu] = undersamplecalib(im, Rx, Ry, calibx, caliby)

% inputs
% im : images from all coils
% Rx : acceleration rate on x-direction
% Ry : acceleration rate on y-direction
% calibx : size of calibration region in x-direction
% caliby : size of calibration ragion in y-direction

% outputs
% Mu : undersampled k-space data with calibration region
% imu : undersampled image with calibration region

%find calibration region
[imc, Mc] = imcalib(im, calibx, caliby);

%set undersampled k-space data to undersampled k-space data
Mu = Mc;

%take sizes of k-space data
[Nkx Nky coil_no] = size(Mu);

%find positions to be set to Mu from full k-space data
posx = 1:Rx:Nkx; % skip Rx-1 rows
posy = 1:Ry:Nky; % skip Ry-1 columns

for i = 1:coil_no

    %take 2D FFT of image
    d = fft2c(im(:,:,i));

    %find undersampled k-space data with calibration region
    %set Mu from full k-space data at desired skipped positions
    Mu(posx,posy,i) = d(posx,posy);
    %find undersampled image with calibration region
    imu(:,:,i) = ifft2c(Mu(:,:,i));

end

end
```

2.3) GRAPPA Kernel Weights (Ry = 2)

In images part, kernel is displayed as an image. We started creating our kernel from left-top data point. We went through each column separately from top to bottom. The coefficients in one full column are coming from all coils. Each column in kernel image is computed for one coil.

Images



MATLAB Code

```
case '2.3'
disp('2.3');
disp('GRAPPA Kernel Weights');

load('multicoil-data.mat');

%(Rx, Ry) = (1,2), (calibx,caliby) = (32,32), lambda=0

Rx = 1; Ry = 2;
lambda = 0;
calibx = 32; caliby = 32;

[imc, Mc] = imcalib(im, calibx, caliby);
kernel = calibrate(Mc,Ry,lambda); % size : ( ( 6*Nc ) x Nc ) = ( 48
x 8 )

%display magnitude of kernel as an image
figure; set(gcf, 'WindowState', 'maximized');
```

```
imshow(abs(kernel),[]);
%title('Kernel for (Rx,Ry) = (1,2)');
saveas(gcf,'2.3_kernel(1,2).png');
```

kernel = calibrate(Mc,Ry,lambda) function

```
function kernel = calibrate(Mc,Ry,lambda)

% calibrate function
% inputs
% Mc : k-space data with calibration region
% Ry : acceleration rate on y-direction, geometry depends on Ry
% lambda - regularization parameter
%
% outputs
% kernel : size of (6xNc), Nc = coil #
%
% pre-condition : Rx = 1
% pre-condition : (calibx,caliby) = (32,32)

Rx = 1;
calibx = 32; caliby = 32;

%take sizes of k-space data
[Nx Ny Nc] = size(Mc);

%kernel for Ry = 2
if (Ry == 2)

    %Ma will be constant for all coils
    Ma = [];
    %go through calibration region
    %skip sample points for the points on first and last
rows/columns
    for col = (Ny/2 - caliby/2 + 1 + 1) : 1 : (Ny/2 + caliby/2 - 1)
        for row = (Nx/2 - calibx/2 + 1 + 1) : 1 : (Nx/2 + calibx/2 -
1)

            %there are 6 samples in neighbourhood of our point
            %for each coil
            temp = zeros(1,6);
            %Ma_row is for 6 samples in neighbourhood of our point
            %Ma_row has samples from all coils
            Ma_row = [];
            for coil = 1:Nc
                ind = 1;  %for going through all sample points
                for j = [-1 1]
                    for i = [-1 0 1]
                        %temp is for one point and one coil
                        temp(ind) = Mc(row+i,col+j,coil);
                        ind = ind + 1;
                    end
                end
                %Ma_row is for one point and all coils
                Ma_row = [Ma_row temp];
```

```
            end
            %Ma is for all points and all coils
            Ma = [Ma; Ma_row];
        end
    end

    %for each coil, we will find different kernels
    %our result for kernel will contain coefficients to be used with
all coils
    for coil = 1:Nc

        Mkc = [];
        %take all points in calibration region
        for col = (Ny/2 - caliby/2 + 1 + 1) : 1 : (Ny/2 + caliby/2 -
1)
            for row = (Nx/2 - calibx/2 + 1 + 1) : 1 : (Nx/2 +
calibx/2 - 1)
            temp = Mc(row,col,coil);
            %collect all data points inside Mkc
            Mkc = [ Mkc; temp(:) ] ;
            end
        end

        %for each coil, solve the linear system of equations
        %for kernel coefficients
        pseudo = Ma'*Ma;
        ak_temp= pinv(pseudo + lambda*eye(size(pseudo))) * Ma' *
Mkc;
        %each columns has coefficients for each coil
        ak(:,coil) = ak_temp(:);
    end
    kernel = ak;
end

%kernel for Ry = 4
if (Ry == 4)

    %Ma is constant for all coils
    %Ma is a 3d-matrix with Ma1, Ma2, and Ma3
    Ma1 = [];
    Ma2 = [];
    Ma3 = [];

    %go through calibration region
    %skip sample points for the points on first two and last two
rows/columns
    for col = (Ny/2 - caliby/2 + 1 + 2) : 1 : (Ny/2 + caliby/2 - 2)
        for row = (Nx/2 - calibx/2 + 1 + 2) : 1 : (Nx/2 + calibx/2 -
2)

            %there are 10 samples in neighbourhood of our point
            %for each coil
            temp = zeros(1,10);

            %Ma_row1/Ma_row2/Ma_row3 is for 10 samples in
neighbourhood of our point
```

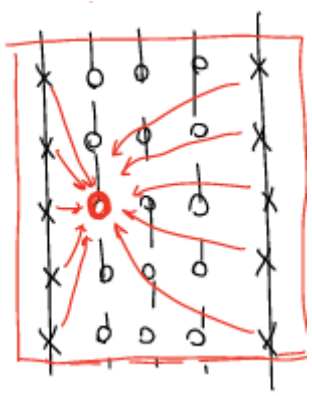
```
        %Ma_row1_Ma_row2_Ma_row3 has samples from all coils
        Ma_row1 = [];
        Ma_row2 = [];
        Ma_row3 = [];

        for coil = 1:Nc
            ind = 1; %for going through all sample points
            for j = [-1 3]
                for i = [-2 -1 0 1 2]
                    %temp is for one point and one coil
                    temp(ind) = Mc(row+i,col+j,coil);
                    ind = ind + 1;
                end
            end
            %Ma_row1 is for one point and all coils
            Ma_row1 = [Ma_row1 temp];
        end
        %Ma1 is for all points and all coils
        Ma1 = [Ma1; Ma_row1];

        for coil = 1:Nc
            ind = 1;  %for going through all sample points
            for j = [-2 2]
                for i = [-2 -1 0 1 2]
                    %temp is for one point and one coil
                    temp(ind) = Mc(row+i,col+j,coil);
                    ind = ind + 1;
                end
            end
            %Ma_row2 is for one point and all coils
            Ma_row2 = [Ma_row2 temp];
        end
        %Ma2 is for all points and all coils
        Ma2 = [Ma2;Ma_row2];

        for coil = 1:Nc
            ind = 1;  %for going through all sample points
            for j = [-3 1]
                for i = [-2 -1 0 1 2]
                    %temp is for one point and one coil
                    temp(ind) = Mc(row+i,col+j,coil);
                    ind = ind + 1;
                end
            end
            %Ma_row2 is for one point and all coils
            Ma_row3 = [Ma_row3 temp];
        end
        %Ma3 is for all points and all coils
        Ma3 = [Ma3;Ma_row3];

    end
end

%construct Ma
%Ma is a 3d-matrix because each lines sees a different geometry
Ma(:,:,1) =Ma1;
```

```
    Ma(:,:,2) =Ma2;
    Ma(:,:,3) =Ma3;

    %for each line, we have different geometry,
    %and we will solve them seperately
    for l = 1:Ry-1

        %for each coil, we will find different kernels
        %our result for kernel will contain coefficients
        %to be used with all coils
        for coil = 1:Nc

            Mkc = [];
            %take all points in calibration region
            for col = (Ny/2 - caliby/2 + 1 + 2) : 1 : (Ny/2 +
caliby/2 - 2)
                for row = (Nx/2 - calibx/2 + 1 + 2) : 1 : (Nx/2 +
calibx/2 - 2) %
                    temp = Mc(row,col,coil);
                    %collect all data points inside Mkc
                    Mkc = [ Mkc; temp(:) ] ;
                end
            end

            %for each coil, solve the linear system of equations
            %for kernel coefficients
            pseudo = Ma(:,:,l)'*Ma(:,:,l);
            ak_temp= pinv(pseudo + lambda*eye(size(pseudo))) *
Ma(:,:,l)' * Mkc;
            %each columns has coefficients for each coil
            ak(:,coil,l) = ak_temp(:);
        end

    end
    kernel = ak;
end


end
```

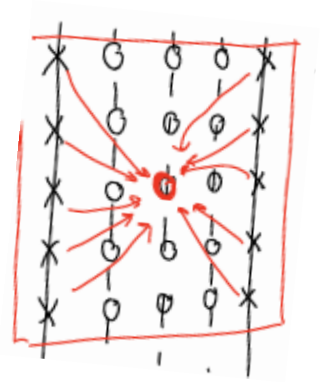
2.4) GRAPPA Kernel Weights (Ry = 4)

In images part, kernels are displayed as separate images. We started creating our kernel from left-top data point. We went through each column separately from top to bottom. There are three lines because three lines are seeing different geometry, therefore calculated kernels are different. The coefficients in one full column are coming from all coils. Each column in kernel images is computed for one coil.

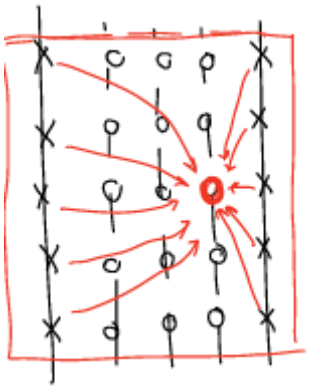
The lines are shown below. The drawings are taken from lecture notes directly.



First Line

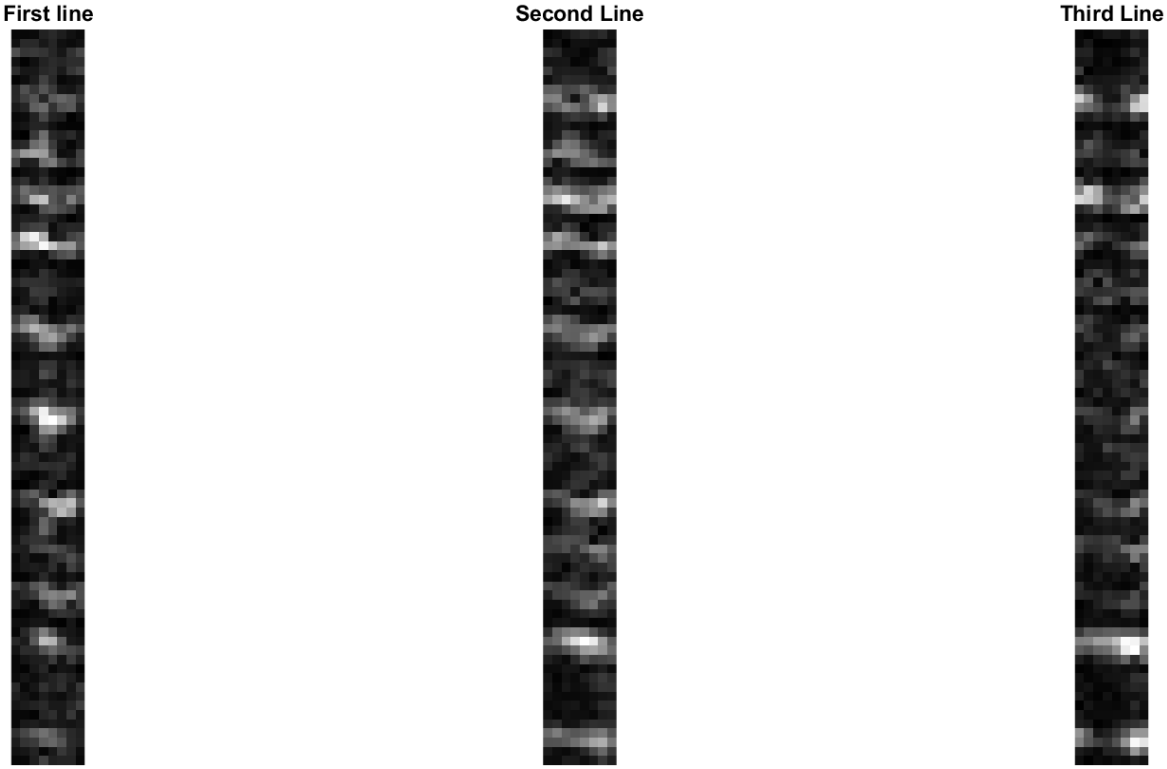


Second Line



Third Line

Images



MATLAB Code

```
case '2.4'
disp('2.4');
disp('GRAPPA Kernel Weights for (Rx,Ry) = (1,4)');

load('multicoil-data.mat');

%(Rx, Ry) = (1,4), (calibx,caliby) = (32,32), lambda=0
Rx = 1; Ry = 4;
```

```
lambda = 0;
calibx = 32; caliby = 32;

[imc, Mc] = imcalib(im, calibx, caliby);
% kernel size : ( ( 10*Nc ) x Nc x (Ry-1) ) = ( 80 x 8 x 3 )
kernel = calibrate(Mc,Ry,lambda);

%display magnitude of kernel as an image
%for first line
figure; set(gcf, 'WindowState', 'maximized');
subplot(1,3,1); imshow(abs(kernel(:,:,1)),[]);
title('First line');

%display magnitude of kernel as an image
%for second line
subplot(1,3,2); imshow(abs(kernel(:,:,2)),[]);
title('Second Line');

%display magnitude of kernel as an image
%for third line
subplot(1,3,3); imshow(abs(kernel(:,:,3)),[]);
title('Third Line');
saveas(gcf,'2.4.png');
```

2.5) GRAPPA Reconstruction (Ry = 2)

By trial-and-error, the regularization parameter is chosen to be 1e12. Visual results are very close to each other when regularization parameter is zero and 1e12. The difference can be seen in IQA values. Still, PSNR and SSIM are very close in two cases.

Results

2.5

GRAPPA Reconstruction

PSNR for GRAPPA Reconstruction (Rx,Ry)=(1,2), lambda=0:24.4922

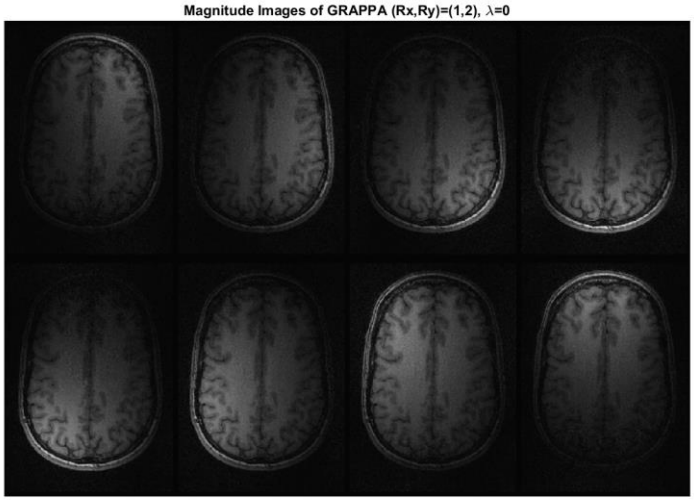
SSIM for GRAPPA Reconstruction (Rx,Ry)=(1,2), lambda=0:0.60705

PSNR for GRAPPA Reconstruction (Rx,Ry)=(1,2), lambda=1e12:24.5407

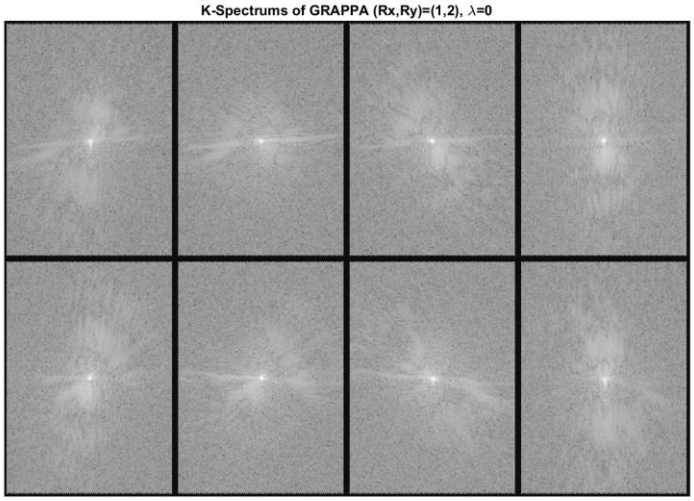
SSIM for GRAPPA Reconstruction (Rx,Ry)=(1,2), lambda=1e12:0.61937

Images

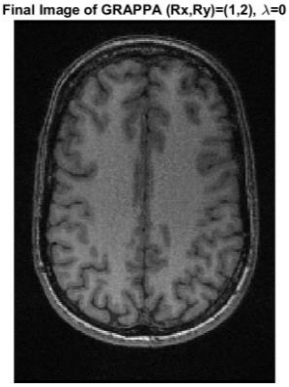
Magnitude Images of GRAPPA (Rx,Ry)=(1,2), λ=0



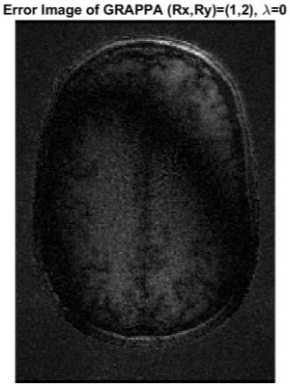
K-Spectrums of GRAPPA (Rx,Ry)=(1,2), λ=0



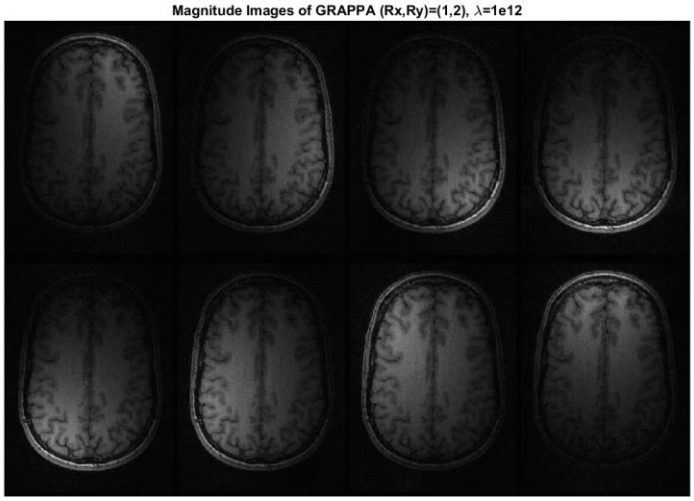
Final Image of GRAPPA (Rx,Ry)=(1,2), λ=0



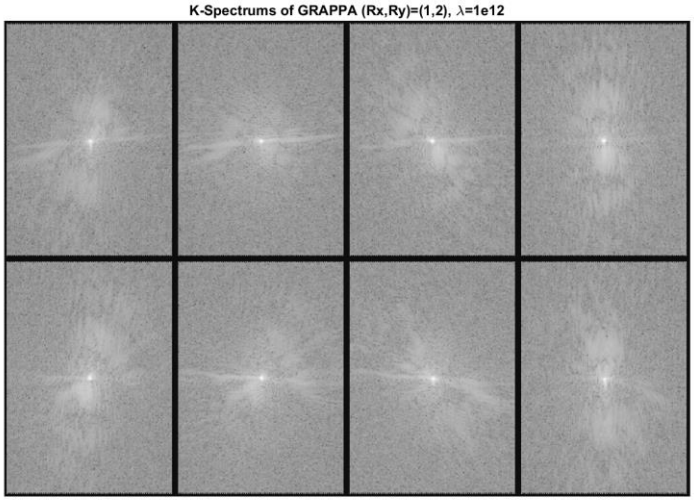
Error Image of GRAPPA (Rx,Ry)=(1,2), λ=0



Magnitude Images of GRAPPA (Rx,Ry)=(1,2), λ=1e12

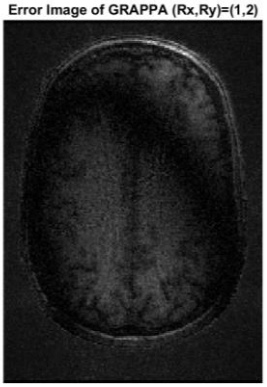


K-Spectrums of GRAPPA (Rx,Ry)=(1,2), λ=1e12



Final Image of GRAPPA (Rx,Ry)=(1,2), λ=1e12

Error Image of GRAPPA (Rx,Ry)=(1,2)



MATLAB Code

```
case '2.5'
disp('2.5');
disp('GRAPPA Reconstruction');

load('multicoil-data.mat');

%use SoS by map1 from Q1.2 as reference image
ref = sos(im,map1);
ref = abs(ref);
ref = ref/max(ref(:));

%(Rx, Ry) = (1,2), (calibx,caliby) = (32,32), lambda=0
Ry = 2; Rx = 1;
calibx = 32; caliby = 32;
lambda = 0;

[imu, Mu] = undersamplecalib(im, Rx, Ry, calibx, caliby);
[imc, Mc] = imcalib(im, calibx, caliby);
[imr, Mr] = grappa(Mu, Mc, Ry, lambda);

%magnitude images
figure; set(gcf, 'WindowState', 'maximized');
montage(abs(imr),'displayRange',[], 'Size', [2 4], 'BorderSize', 5);
title('Magnitude Images of GRAPPA (Rx,Ry)=(1,2), {\lambda}=0');
saveas(gcf,'2.5_mag.png');

%k-space images
figure; set(gcf, 'WindowState', 'maximized');
montage(log(abs(Mr)+1),'displayRange',[], 'Size', [2 4],
'BorderSize', 5);
title('K-Spectrums of GRAPPA (Rx,Ry)=(1,2), {\lambda}=0');
saveas(gcf,'2.5_kspace.png');

%final image of GRAPPA by SoS
map = ones(size(imr));
m = sos(imr,map);

%final image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(m),[]);
title('Final Image of GRAPPA (Rx,Ry)=(1,2), {\lambda}=0');
saveas(gcf,'2.5_final.png');

%normalize final image
m = abs(m);
m = m/max(m(:));

%error image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(m-ref),[]);
title('Error Image of GRAPPA (Rx,Ry)=(1,2), {\lambda}=0');
saveas(gcf,'2.5_error.png');
```

```
%IQA results
PSNR= psnr(m,ref);
SSIM= ssim(m,ref);
disp(strcat('PSNR for GRAPPA Reconstruction (Rx,Ry)=(1,2),
lambda=0:',num2str(PSNR)));
disp(strcat('SSIM for GRAPPA Reconstruction (Rx,Ry)=(1,2),
lambda=0:',num2str(SSIM)));

%select lambda as regularization parameter
lambda = 1e12;

[imu, Mu] = undersamplecalib(im, Rx, Ry, calibx, caliby);
[imc, Mc] = imcalib(im, calibx, caliby);
[imr, Mr] = grappa(Mu, Mc, Ry, lambda);

%magnitude images
figure; set(gcf, 'WindowState', 'maximized');
montage(abs(imr),'displayRange',[], 'Size', [2 4], 'BorderSize', 5);
title('Magnitude Images of GRAPPA (Rx,Ry)=(1,2), {\lambda}=1e12');
saveas(gcf,'2.5_mag_lambda.png');

%k-space images
figure; set(gcf, 'WindowState', 'maximized');
montage(log(abs(Mr)+1),'displayRange',[], 'Size', [2 4],
'BorderSize', 5);
title('K-Spectrums of GRAPPA (Rx,Ry)=(1,2), {\lambda}=1e12');
saveas(gcf,'2.5_kspace_lambda.png');

%final image of GRAPPA by SoS
map = ones(size(imr));
m = sos(imr,map);

%final image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(m),[]);
title('Final Image of GRAPPA (Rx,Ry)=(1,2), {\lambda}=1e12');
saveas(gcf,'2.5_final_lambda.png');

%normalize final image
m = abs(m);
m = m/max(m(:));

%error image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(m-ref),[]);
title('Error Image of GRAPPA (Rx,Ry)=(1,2)');
saveas(gcf,'2.5_error_lambda.png');

%IQA results
PSNR= psnr(m,ref);
SSIM= ssim(m,ref);
disp(strcat('PSNR for GRAPPA Reconstruction (Rx,Ry)=(1,2),
lambda=1e12:',num2str(PSNR)));
disp(strcat('SSIM for GRAPPA Reconstruction (Rx,Ry)=(1,2),
lambda=1e12:',num2str(SSIM)));
```

[imr, Mr] = grappa(Mu, Mc, Ry, lambda) function

```
function [imr, Mr] = grappa(Mu, Mc, Ry, lambda)

% grappa function
% inputs
% Mu : undersampled k-space data with calibration region
% Mc : k-space data with calibration region
% Ry : acceleration rate on y-direction, geometry depends on Ry
% lambda - regularization parameter
%
% outputs
% imr : size of (Nx x Ny x Nc)
% Mr : size of (Nx x Ny x Nc)
%
% pre-condition : Rx = 1
% pre-condition : (calibx,caliby) = (32,32)

Rx = 1;
calibx = 32; caliby = 32;

%take sizes of k-space data
[Nx Ny Nc] = size(Mc);

%find kernel to be used in grappa reconstruction
kernel = calibrate(Mc,Ry,lambda);

%set Mr to undersampled k-psace data with calibration region
Mr = Mu;

if (Ry == 2)
%find grappa reconstruction for all coils
for k = 1:Nc
    for col = [2:Ry:Ny-1] %skip one line in each iteration
        for row = [2:Rx:Nx-1] %Rx = 1

            %take kernel for one coil
            kernel_temp = kernel(:,k);

            sum = 0;
            index = 1;
            %take coefficients calculated for all coils
            for coil = 1:Nc
                for j = [-1 1]
                    for i = [-1 0 1]
                        %take coefficient for the sample in
neighbourhood
                        coefficient = kernel_temp(index);
                        %sum up the multiplications of each
                        %coefficient and each sample
                        sum = sum + coefficient *
Mu(row+i,col+j,coil);
                        %index for going through all coefficients
                        index = index + 1;
                    end
                end
```

```
            end
            %put found value to missing point
            Mr(row,col,k) = sum;
        end
    end
end
end

if (Ry == 4)

for k = 1:Nc
    for col = 5:Ry:Ny-5
        for row =  3:1:Nx-2

        sum = 0;
        index = 1;
        kernel_temp = kernel(:,k,1);
        for coil = 1:Nc
            ind = 1;
            for j = [-1 3]
                for i = [-2 -1 0 1 2]
                    coefficient = kernel_temp(index);
                    sum = sum + coefficient *
Mu(row+i,col+1+j,coil);
                    index = index + 1;
                end
            end
        end
        Mr(row,col+1,k) = sum;

        sum = 0;
        index = 1;
        kernel_temp = kernel(:,k,2);
        for coil = 1:Nc
            for j = [-2 2]
                for i = [-2 -1 0 1 2]
                    coefficient = kernel_temp(index);
                    sum = sum + coefficient *
Mu(row+i,col+2+j,coil);
                    index = index + 1;
                end
            end
        end
        Mr(row,col+2,k) = sum;

        sum=0;
        index = 1;
        kernel_temp = kernel(:,k,3);
        for coil = 1:Nc
           for j = [-3 1]
                for i = [-2 -1 0 1 2]
                    coefficient = kernel_temp(index);
                    sum = sum + coefficient *
Mu(row+i,col+3+j,coil);
                    index = index + 1;
                end
```

```
            end
        end
        Mr(row,col+3,k) = sum;

        end
    end
end
end

%correct calibration region
for i = 1:Nc

    %positions for calibration region
    pos_calibx = (Nx/2 - calibx/2 +1): (Nx/2 + calibx/2);
    pos_caliby = (Ny/2 - caliby/2 +1): (Ny/2 + caliby/2);

    %set calibration region to original calibration measurement
    Mr(pos_calibx,pos_caliby,i) = Mu(pos_calibx,pos_caliby,i);

    %find grappa reconstructed images for all coils
    imr(:,:,i) = ifft2c(Mr(:,:,i));
end

end
```

2.6) GRAPPA Reconstruction (Ry = 4)

By trial-and-error, regularization parameter is chosen to be 1e14. When lambda is high, the magnitude images gets smoother due to low pass filtering effect of regularization parameter. When lambda is lower, the central noise cannot be discarded. The magnitude images are the almost best results that can be reconstructed with GRAPPA for Ry = 4.

The difference for having regularization parameter can be seen in increase in SSIM values. Visually, the central noise is discarded relatively. Still, there is the aliased artifact in magnitude image of lambda = 1e14. Also, the result image is smoother than actual image.

Results

2.6

GRAPPA Reconstruction

PSNR for GRAPPA Reconstruction (Rx,Ry)=(1,4), lambda=0:19.4442

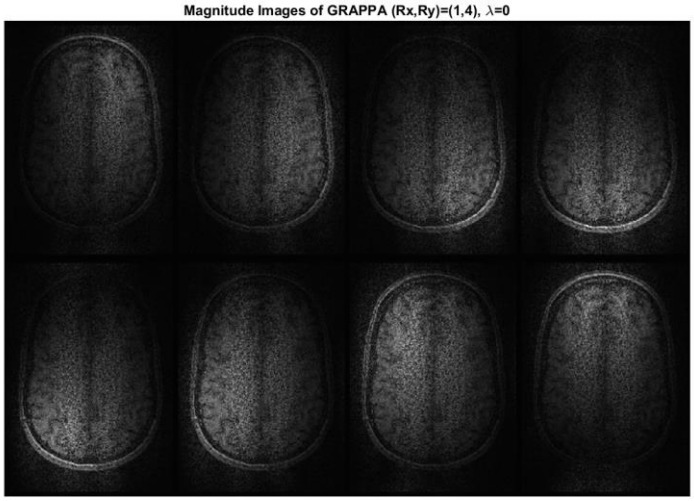
SSIM for GRAPPA Reconstruction (Rx,Ry)=(1,4), lambda=0:0.29287

PSNR for GRAPPA Reconstruction (Rx,Ry)=(1,4), lambda=1e14:19.2455

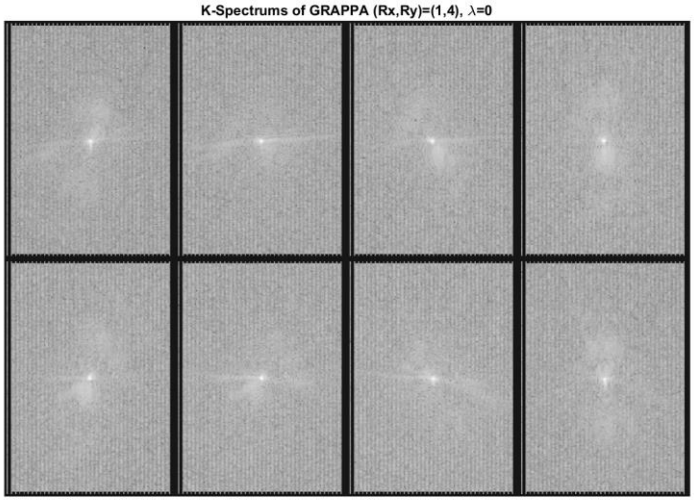
SSIM for GRAPPA Reconstruction (Rx,Ry)=(1,4), lambda=1e14:0.46264

Images

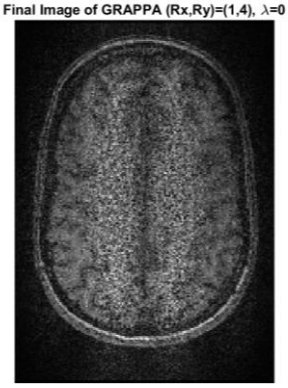
Magnitude Images of GRAPPA (Rx,Ry)=(1,4), λ=0



K-Spectrums of GRAPPA (Rx,Ry)=(1,4), λ=0

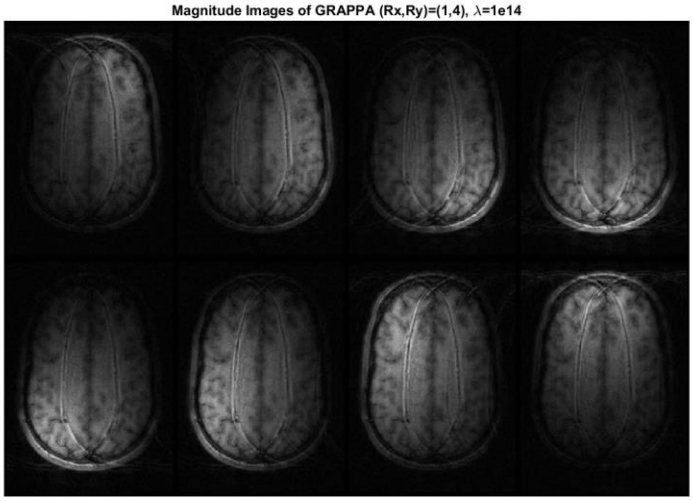


Final Image of GRAPPA (Rx,Ry)=(1,4), λ=0

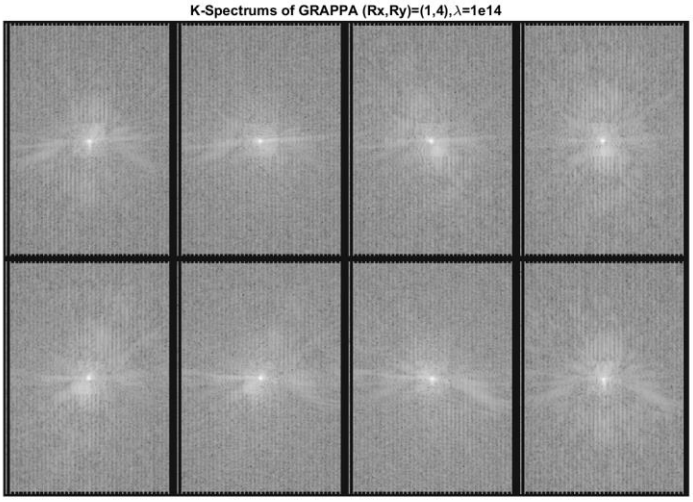


Error Image of GRAPPA (Rx,Ry)=(1,4), λ=0

Magnitude Images of GRAPPA (Rx,Ry)=(1,4), λ=1e14

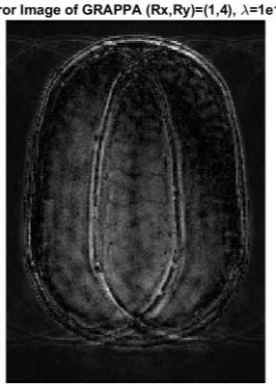


K-Spectrums of GRAPPA (Rx,Ry)=(1,4),λ=1e14



Final Image of GRAPPA (Rx,Ry)=(1,4), λ=1e14

Error Image of GRAPPA (Rx,Ry)=(1,4), λ=1e14



MATLAB Code

```
case '2.6'

disp('2.6');
disp('GRAPPA Reconstruction');

load('multicoil-data.mat');

%use SoS by map1 from Q1.2 as reference image
ref = sos(im,map1);
ref = abs(ref);
ref = ref/max(ref(:));

%(Rx, Ry) = (1,4), (calibx,caliby) = (32,32), lambda=0
Ry = 4; Rx = 1;
calibx = 32; caliby = 32;
lambda = 0;

[imu, Mu] = undersamplecalib(im, Rx, Ry, calibx, caliby);
[imc, Mc] = imcalib(im, calibx, caliby);
[imr, Mr] = grappa(Mu, Mc, Ry, lambda);

%magnitude images
figure; set(gcf, 'WindowState', 'maximized');
montage(abs(imr),'displayRange',[], 'Size', [2 4], 'BorderSize', 5);
title('Magnitude Images of GRAPPA (Rx,Ry)=(1,4), {\lambda}=0');
saveas(gcf,'2.6_mag.png');

%k-space images
figure; set(gcf, 'WindowState', 'maximized');
montage(log(abs(Mr)+1),'displayRange',[], 'Size', [2 4],
'BorderSize', 5);
title('K-Spectrums of GRAPPA (Rx,Ry)=(1,4), {\lambda}=0');
saveas(gcf,'2.6_kspace.png');

%final image of GRAPPA by SoS
map = ones(size(imr));
m = sos(imr,map);

%final image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(m),[]);
title('Final Image of GRAPPA (Rx,Ry)=(1,4), {\lambda}=0');
saveas(gcf,'2.6_final.png');

%normalize final image
m = abs(m);
m = m/max(m(:));

%error image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(m-ref),[]);
title('Error Image of GRAPPA (Rx,Ry)=(1,4), {\lambda}=0');
```

```
saveas(gcf,'2.6_error.png');

%IQA results
PSNR= psnr(m,ref);
SSIM= ssim(m,ref);
disp(strcat('PSNR for GRAPPA Reconstruction (Rx,Ry)=(1,4),
lambda=0:',num2str(PSNR)));
disp(strcat('SSIM for GRAPPA Reconstruction (Rx,Ry)=(1,4),
lambda=0:',num2str(SSIM)));

%select lambda as regularization parameter
lambda = 1e14;

[imu, Mu] = undersamplecalib(im, Rx, Ry, calibx, caliby);
[imc, Mc] = imcalib(im, calibx, caliby);
[imr, Mr] = grappa(Mu, Mc, Ry, lambda);

%magnitude images
figure; set(gcf, 'WindowState', 'maximized');
montage(abs(imr),'displayRange',[], 'Size', [2 4], 'BorderSize', 5);
title('Magnitude Images of GRAPPA (Rx,Ry)=(1,4), {\lambda}=1e14');
saveas(gcf,'2.6_mag_lambda.png');

%k-space images
figure; set(gcf, 'WindowState', 'maximized');
montage(log(abs(Mr)+1),'displayRange',[], 'Size', [2 4],
'BorderSize', 5);
title('K-Spectrums of GRAPPA (Rx,Ry)=(1,4),{\lambda}=1e14');
saveas(gcf,'2.6_kspace_lambda.png');

%final image of GRAPPA by SoS
map = ones(size(imr));
m = sos(imr,map);

%final image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(m),[]);
title('Final Image of GRAPPA (Rx,Ry)=(1,4), {\lambda}=1e14');
saveas(gcf,'2.6_final_lambda.png');

%normalize final image
m = abs(m);
m = m/max(m(:));

%error image
figure; set(gcf, 'WindowState', 'maximized');
imshow(abs(m-ref),[]);
title('Error Image of GRAPPA (Rx,Ry)=(1,4), {\lambda}=1e14');
saveas(gcf,'2.6_error_lambda.png');

%IQA results
PSNR= psnr(m,ref);
SSIM= ssim(m,ref);
disp(strcat('PSNR for GRAPPA Reconstruction (Rx,Ry)=(1,4),
lambda=1e14:',num2str(PSNR)));
```

```
disp(strcat('SSIM for GRAPPA Reconstruction (Rx,Ry)=(1,4),
lambda=1e14:',num2str(SSIM)));
```

[imr, Mr] = grappa(Mu, Mc, Ry, lambda) function

```
function [imr, Mr] = grappa(Mu, Mc, Ry, lambda)

% grappa function
% inputs
% Mu : undersampled k-space data with calibration region
% Mc : k-space data with calibration region
% Ry : acceleration rate on y-direction, geometry depends on Ry
% lambda - regularization parameter
%
% outputs
% imr : size of (Nx x Ny x Nc)
% Mr : size of (Nx x Ny x Nc)
%
% pre-condition : Rx = 1
% pre-condition : (calibx,caliby) = (32,32)

Rx = 1;
calibx = 32; caliby = 32;

%take sizes of k-space data
[Nx Ny Nc] = size(Mc);

%find kernel to be used in grappa reconstruction
kernel = calibrate(Mc,Ry,lambda);

%set Mr to undersampled k-psace data with calibration region
Mr = Mu;

if (Ry == 2)
%find grappa reconstruction for all coils
for k = 1:Nc
    for col = [2:Ry:Ny-1] %skip one line in each iteration
        for row = [2:Rx:Nx-1] %Rx = 1

            %take kernel for one coil
            kernel_temp = kernel(:,k);

            sum = 0;
            index = 1;
            %take coefficients calculated for all coils
            for coil = 1:Nc
                for j = [-1 1]
                    for i = [-1 0 1]
                        %take coefficient for the sample in
neighbourhood
                        coefficient = kernel_temp(index);
                        %sum up the multiplications of each
                        %coefficient and each sample
                        sum = sum + coefficient *
Mu(row+i,col+j,coil);
                        %index for going through all coefficients
```

```
                        index = index + 1;
                    end
                end
            end
            %put found value to missing point
            Mr(row,col,k) = sum;
        end
    end
end
end

if (Ry == 4)

for k = 1:Nc
    for col = 5:Ry:Ny-5
        for row =  3:1:Nx-2

        sum = 0;
        index = 1;
        kernel_temp = kernel(:,k,1);
        for coil = 1:Nc
            ind = 1;
            for j = [-1 3]
                for i = [-2 -1 0 1 2]
                    coefficient = kernel_temp(index);
                    sum = sum + coefficient *
Mu(row+i,col+1+j,coil);
                    index = index + 1;
                end
            end
        end
        Mr(row,col+1,k) = sum;

        sum = 0;
        index = 1;
        kernel_temp = kernel(:,k,2);
        for coil = 1:Nc
            for j = [-2 2]
                for i = [-2 -1 0 1 2]
                    coefficient = kernel_temp(index);
                    sum = sum + coefficient *
Mu(row+i,col+2+j,coil);
                    index = index + 1;
                end
            end
        end
        Mr(row,col+2,k) = sum;

        sum=0;
        index = 1;
        kernel_temp = kernel(:,k,3);
        for coil = 1:Nc
           for j = [-3 1]
                for i = [-2 -1 0 1 2]
                    coefficient = kernel_temp(index);
```

```
                    sum = sum + coefficient *
Mu(row+i,col+3+j,coil);
                    index = index + 1;
                end
            end
        end
        Mr(row,col+3,k) = sum;

        end
    end
end
end

%correct calibration region
for i = 1:Nc

    %positions for calibration region
    pos_calibx = (Nx/2 - calibx/2 +1): (Nx/2 + calibx/2);
    pos_caliby = (Ny/2 - caliby/2 +1): (Ny/2 + caliby/2);

    %set calibration region to original calibration measurement
    Mr(pos_calibx,pos_caliby,i) = Mu(pos_calibx,pos_caliby,i);

    %find grappa reconstructed images for all coils
    imr(:,:,i) = ifft2c(Mr(:,:,i));
end

end
```

2.7) Final Remarks

~~GRAPPA can reconstruct images for separate coils. Then, we combine images for each coil with SoS or OLC. SENSE does not give this information for us. SENSE directly gives the result image.~~

GRAPPA gave better visual results for higher acceleration rates.

SENSE is sensitive to coil sensitivities, and cannot be used in each case. Coil sensitivities change in each case for different patients. Therefore, the image is not guaranteed to be good at the end for different patients.

GRAPPA gave better results due to calibration region. We need to calibrate for different patients each case, which means this result is more trustable for different patients.

GRAPPA algorithm was easier in our homework but it becomes harder when acceleration rate increases. Therefore, we would prefer SENSE if we wanted accelerate more.