Table of Contents

COMPGV15: Project MRI Case Study 1
Define MRI Parameters 1
Define Phantom to be Reconstructed 1
Simulation of MRI 1
Prepare Data for Recon 3
Conjugate Gradient Method no Regularisation 5
Preconditoned Conjugate Gradient Method 1st order Tikhonov Regularisation white Reference Image 7
Preconditioned Conjugate Gradient Method 1st order Tikhonov Regularisation with Prior 10
Total Variation Method 12

COMPGV15: Project MRI Case Study

Using BIG http://bigwww.epfl.ch/algorithms/mri-reconstruction/ toolbox MRI reconstruction is done with two main optmisation techniques.

```
addpath(genpath('BIG'));
simu = 'rasterized';
```

Define MRI Parameters

Most of this setup is based on example code from the BIG toolbox

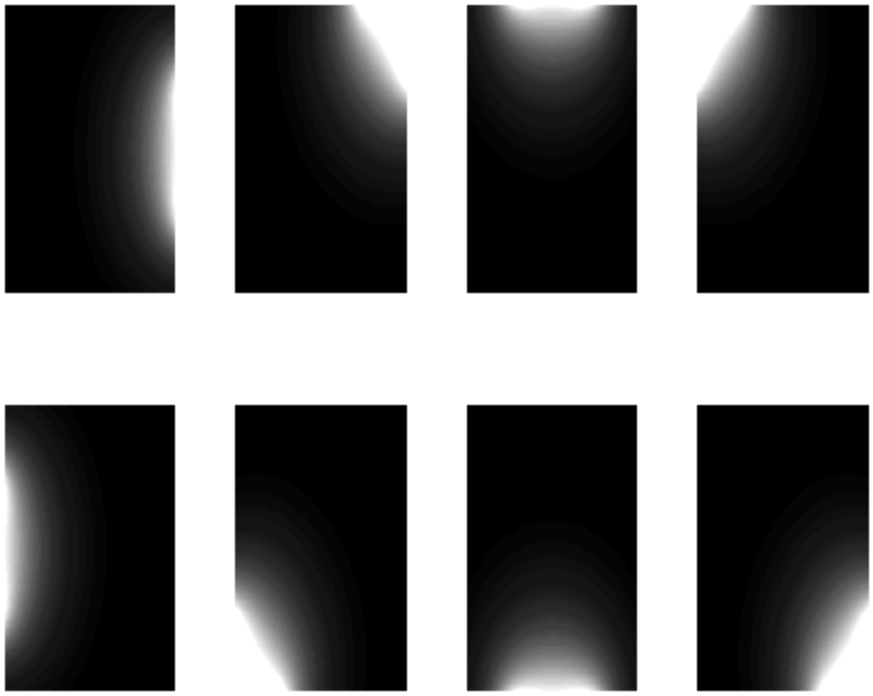
```
mxsize = 256*[1,1];
R = 4;
f_sampling = 1/1.3;
traj = 'cartesian';
snr_hf = .7;
Ncoils = 8;
FOV = 0.28;
pixelsize = FOV./mxsize;
```

Define Phantom to be Reconstructed

```
DefineBrain;
Brain.FOV = FOV*[1, 1];
ref = RasterizePhantom(Brain,mxsize);
support = (ref>5e-3*max(ref(:)));
support = imdilate(support,strel('disk',5));
```

Simulation of MRI

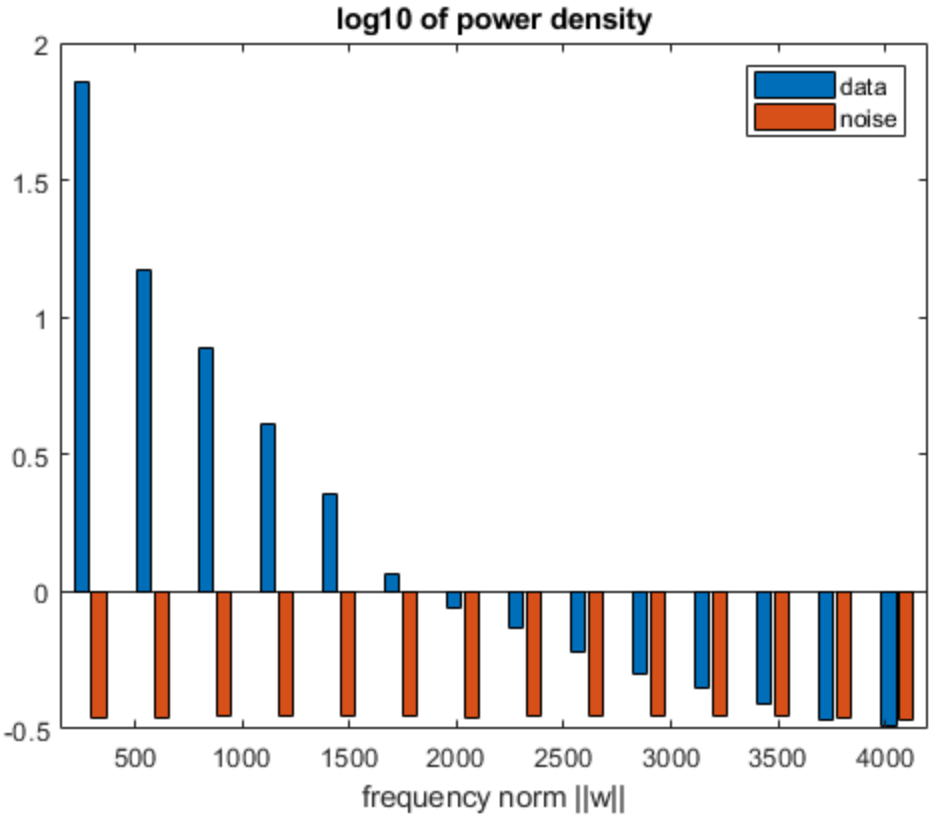
```
sensitivity = GenerateSensitivityMap( FOV, pixelsize,
 Ncoils, .07, .17);
for i=1:size(sensitivity,3)
   subplot(2,4,i);image(abs(sensitivity(:,:,i)*1e2));
 colormap(gray);axis off
end
```



This figure shows the different sensitivites for each of the coils. There are 8 coils distributed evenly in the radial direction

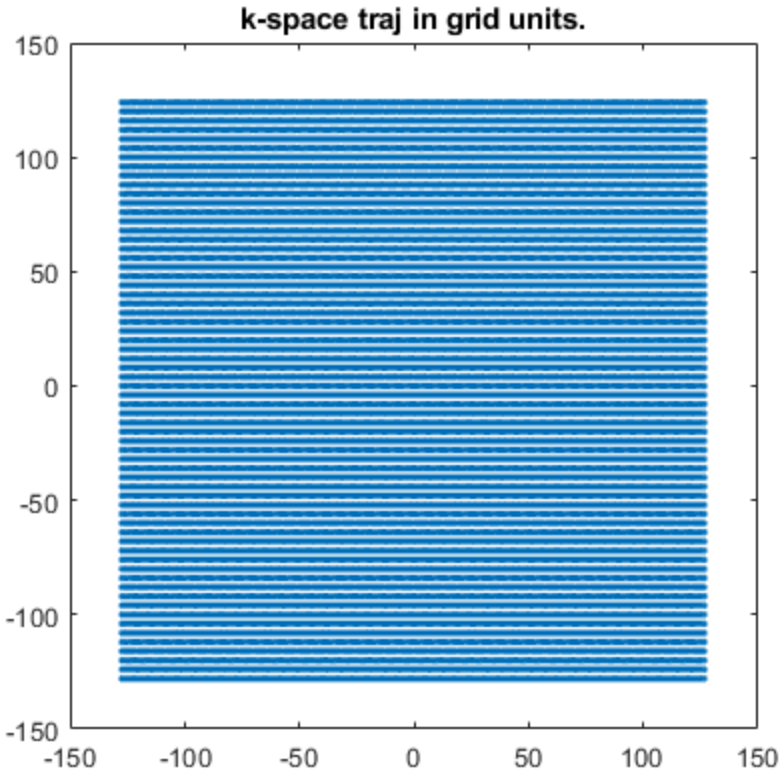
```
w = GenerateCartesianTraj(FOV, pixelsize, f_sampling, R);

m = zeros(size(w,1),Ncoils);
switch simu
    case 'analytical'
        sens=cell(1,Ncoils);
        for indCoil = 1:Ncoils
            sens{indCoil} =
 SensFitting(sensitivity(:,:,indCoil),'sinusoidal',6,support);
            m(:,indCoil) = MRDataAnalytical(Brain, sens{indCoil}, w);
            [~,sensitivity(:,:,indCoil)] =
 RasterizePhantom(Brain,mxsize,sens{indCoil});
        end
        m = prod(mxsize)*m;
    case 'rasterized'
        refinement = 3;
        im_rast = RasterizePhantom(Brain,refinement*mxsize);
        sens = GenerateSensitivityMap( FOV, FOV./mxsize/refinement,
 Ncoils, .07, .17);
        for indCoil = 1:Ncoils
            m(:,indCoil) =
 MRDataRasterized(sens(:,:,indCoil).*im_rast, w, FOV)/
refinement^ndims(im_rast);
```

```
        end
end
n = SimulateNoise(m,w,snr_hf,true);
m = m + n;
```



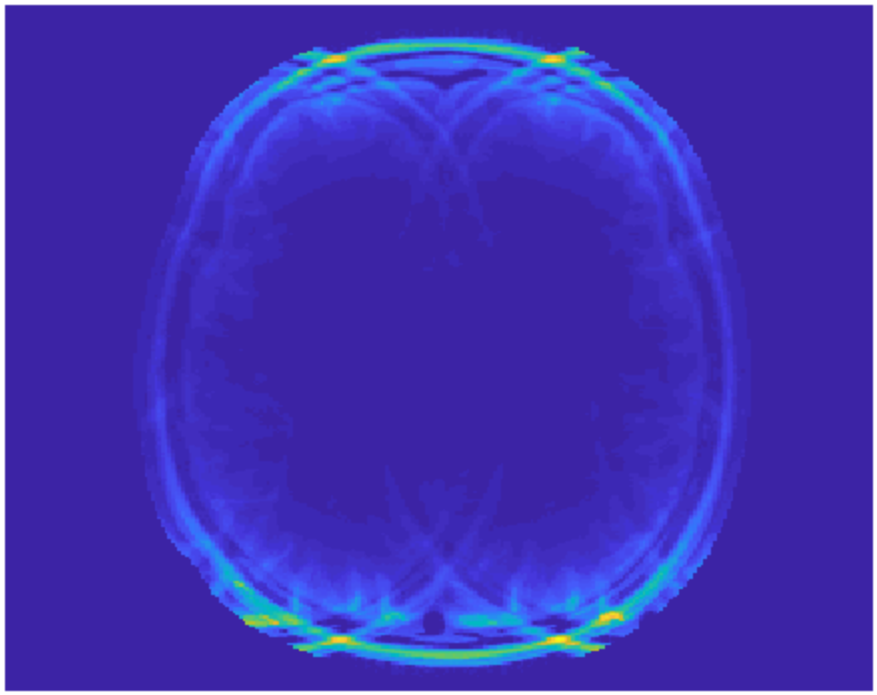
Prepare Data for Recon

```
k = TrajInGridUnits(w, FOV, mxsize);
figure;plot(k(:,1),k(:,2),'.');axis square;title('k-space traj in grid
 units.')
```



This plot shows the k-space trajectory generated by BIG. It matches the cartesian trajectory specified earlier.

```
[a,A,P] = Prepare4Recon(m, k, sensitivity, support);
x = a./P./P;
figure;imagesc(abs(x));axis off;
```



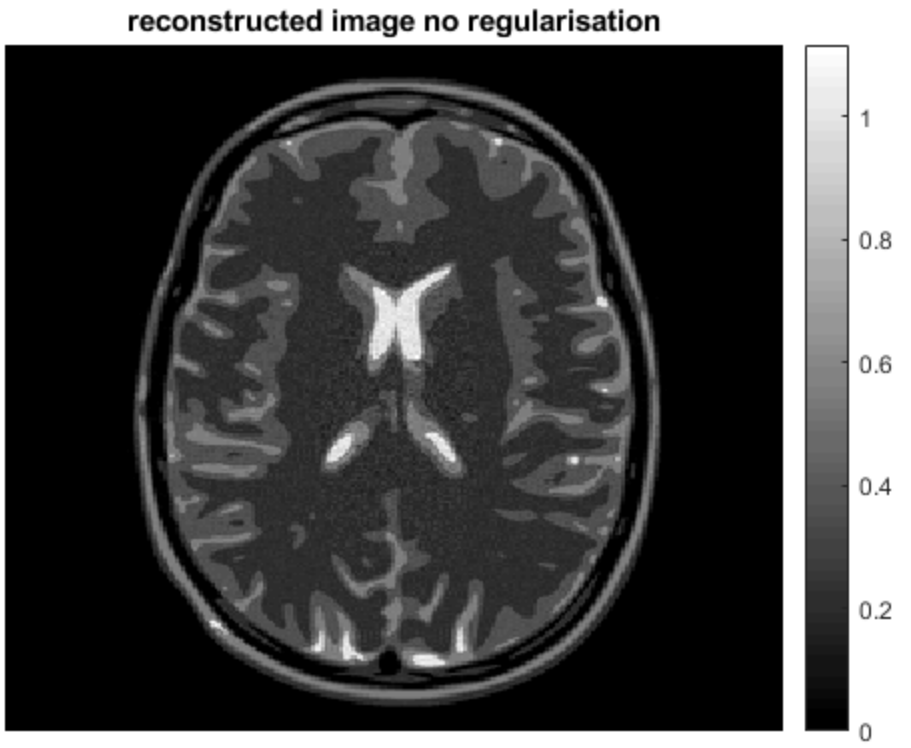
This figure shows the starting point of the optimisation. It is the simple fourier transform from k-space to image space.

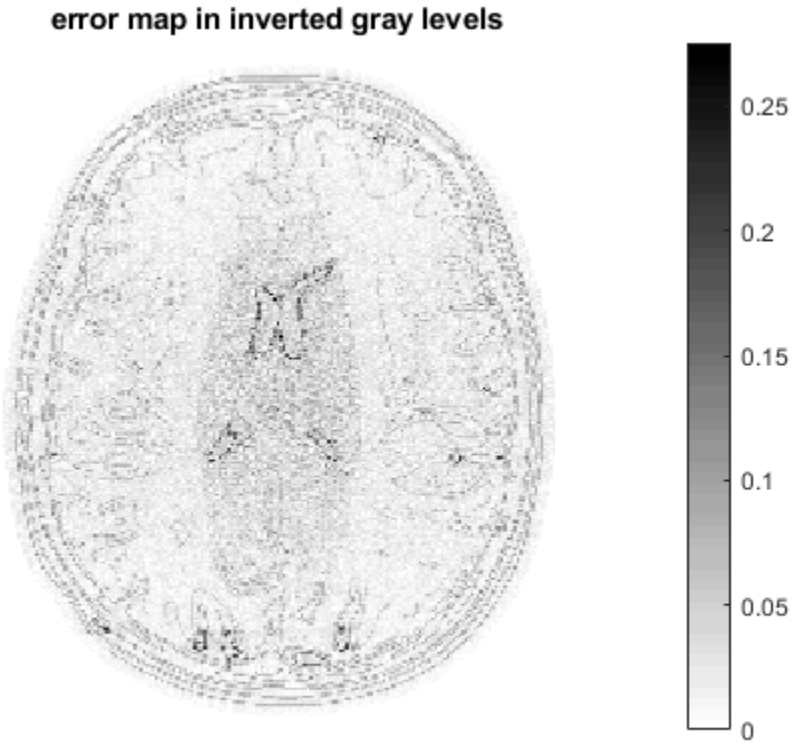
```
clear k m n w sens sensitivity;
```

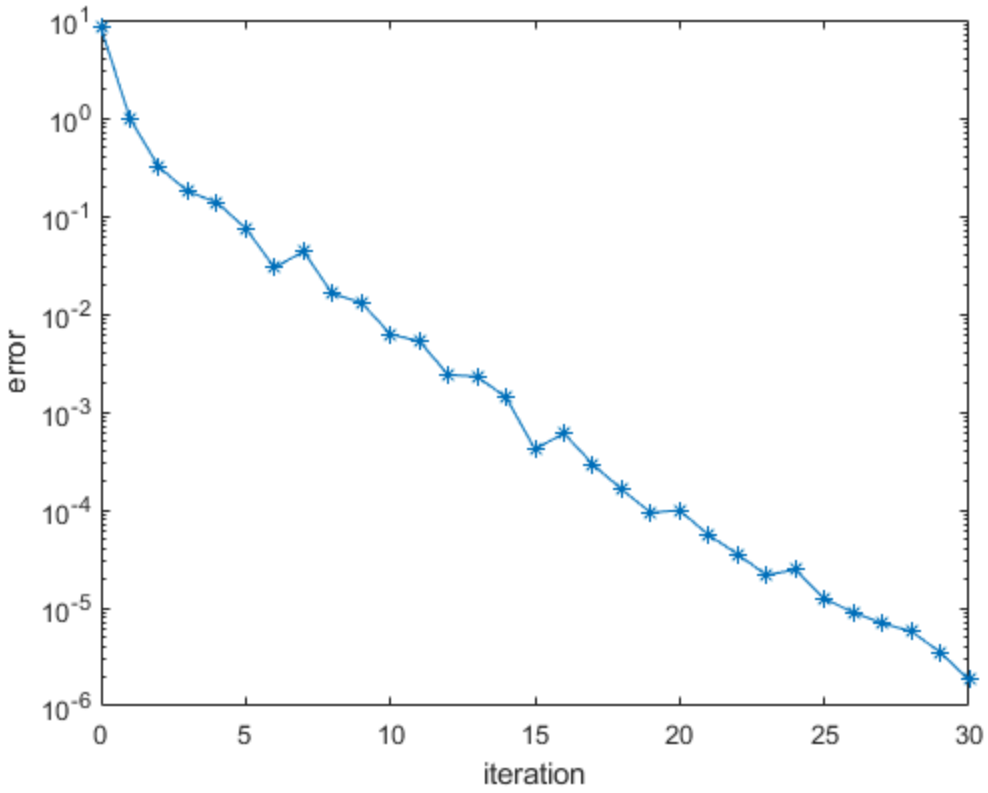
Conjugate Gradient Method no Regularisation

```
[x, t, d, ~] = conjugateGradient(@(x) A(x), a, 1e-9, 30, P, a);
disp(['PCG no regularisation took ' num2str(t) ' iterations and has an
 error of ' num2str(d(end)) '.']);
err = x-ref;
figure;imagesc(abs(x));colormap gray;axis off;colorbar;title('reconstructed
 image no regularisation');
figure;imagesc(abs(err));colormap(1-
gray);axis off;colorbar;title('error map in inverted gray levels');
figure;semilogy(0:1:t,d,'*-');xlabel('iteration');ylabel('error');
```

```
PCG no regularisation took 30 iterations and has an error of
 1.8426e-06.
```


reconstructed image no regularisation
1
0.8
0.6
0.4
0.2
0


error map in inverted gray levels
0.25
0.2
0.15
0.1
0.05
0



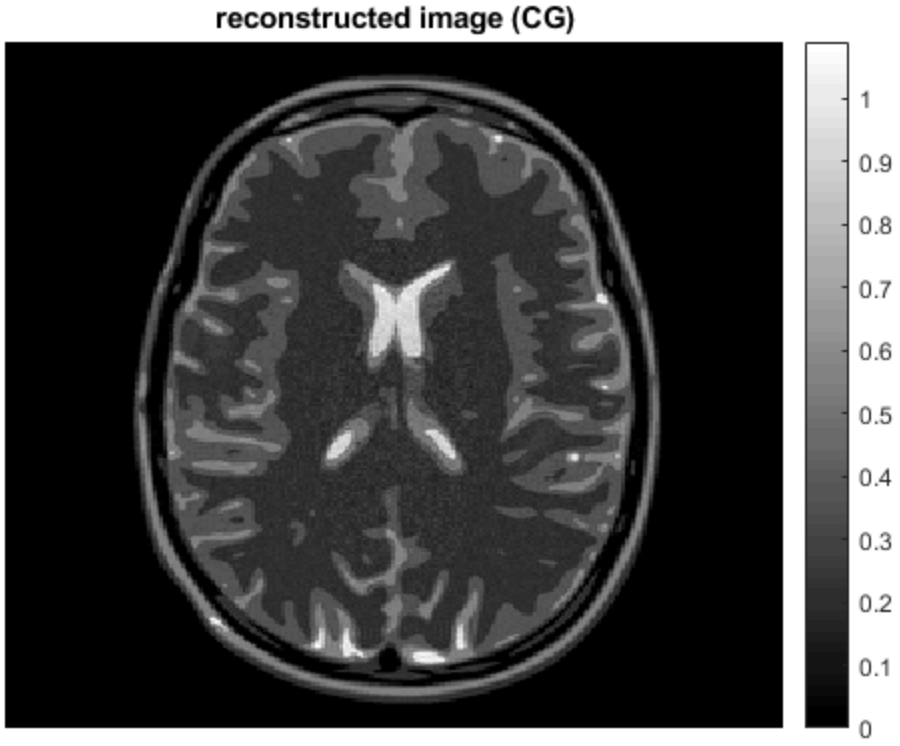
These plots show the results for cg method with no regularisation to demostrate the effect of regularisation. A preconditoning prior is used. The reconstructed image, error and error over time are shown.

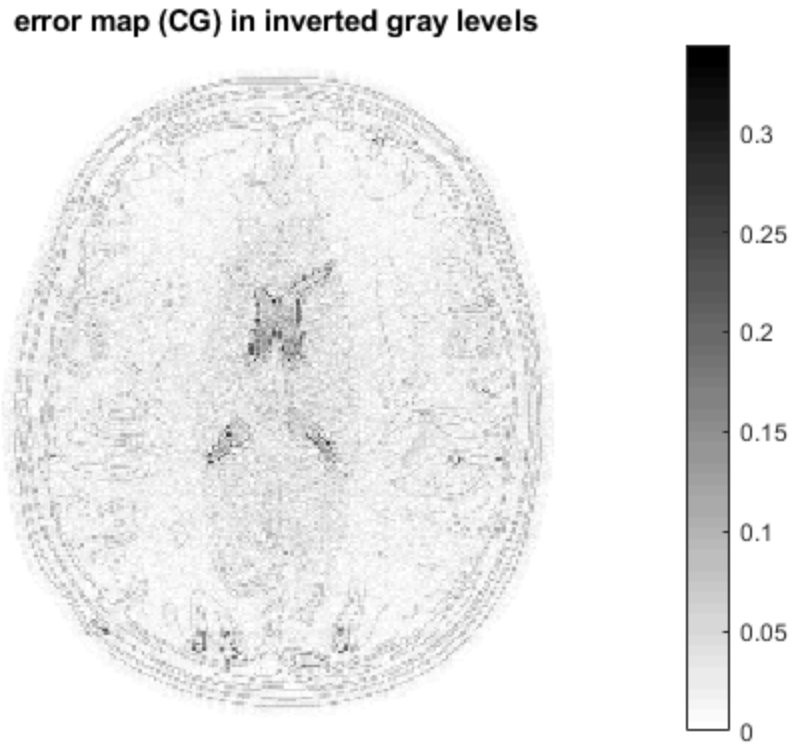
Preconditoned Conjugate Gradient Method 1st order Tikhonov Regularisation white Reference Image

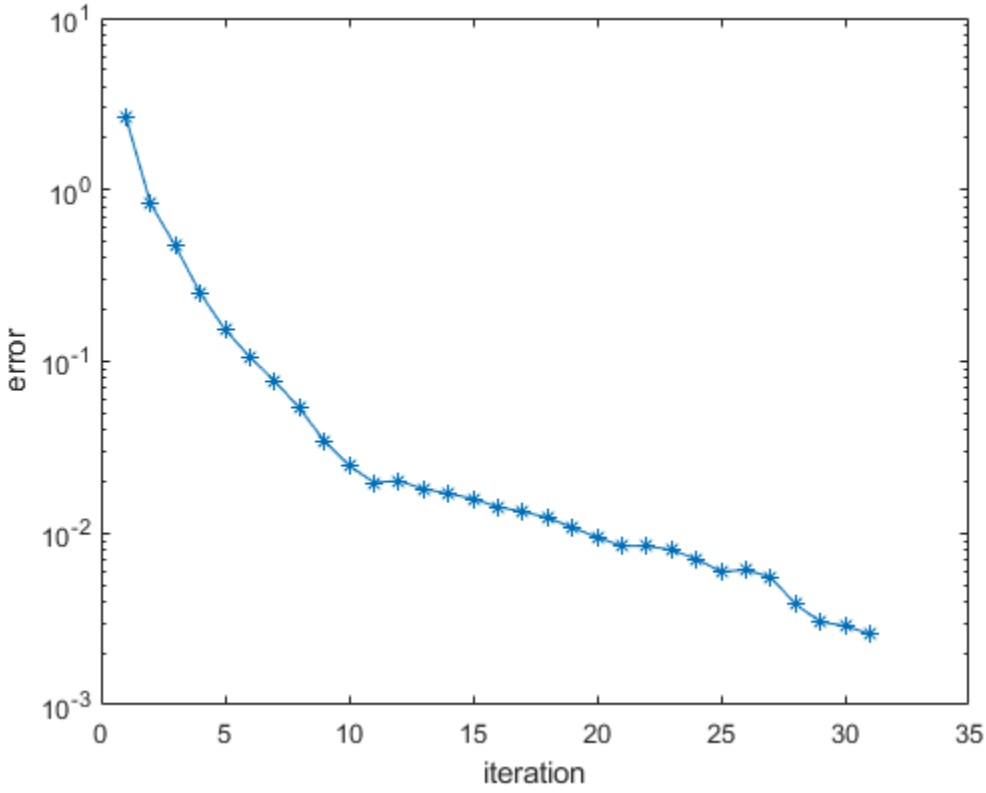
$\lambda = 1e-3 * abs(a))$ is from the BIG toolbox example code. This parameter can be explicitly found from Discrepancy Princple or Miller Criterion etc.

```
lambda = 1e-3*max(abs(a(:)));
[x, t, d, ~] = conjugateGradient(@(x) A(x) + lambda*x, a, 1e-9, 30,
 ones(size(P)), a);
disp(['CG with no preconditioning took ' num2str(t) ' iterations and
 has an error of ' num2str(d(end)) '.']);
err = x-ref;
figure;imagesc(abs(x));colormap gray;axis off;colorbar;title('reconstructed
 image (CG)');
figure;imagesc(abs(err));colormap(1-
gray);axis off;colorbar;title('error map (CG) in inverted gray
 levels');
figure;semilogy(d,'*-');xlabel('iteration');ylabel('error');
```

CG with no preconditioning took 30 iterations and has an error of 0.002583.




error map (CG) in inverted gray levels
0.3
0.25
0.2
0.15
0.1
0.05
0

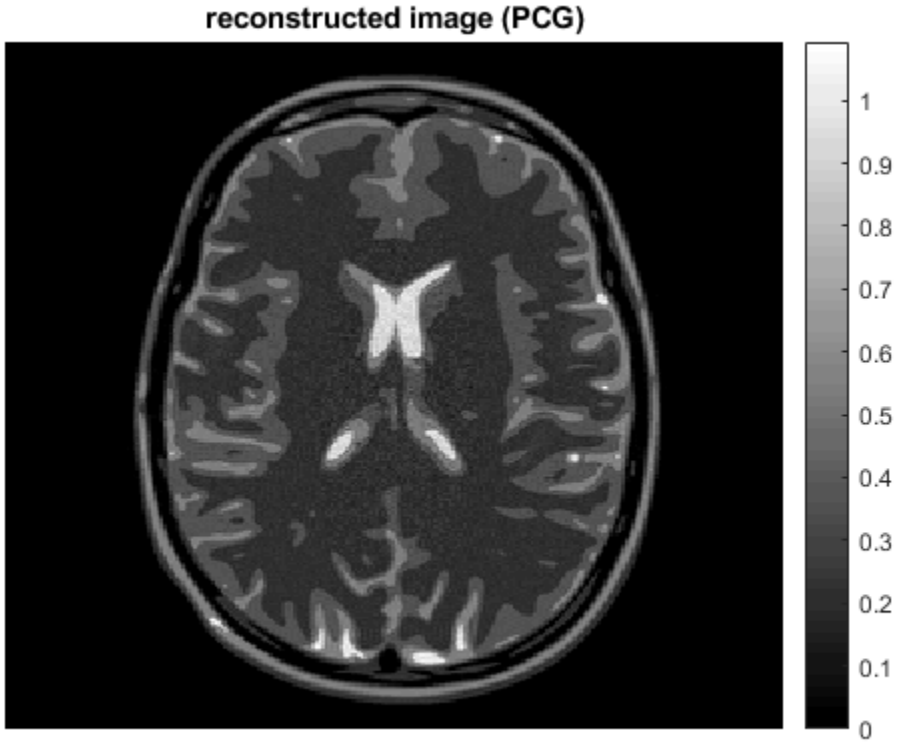

error
iteration
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
0
5
10
15
20
25
30
35

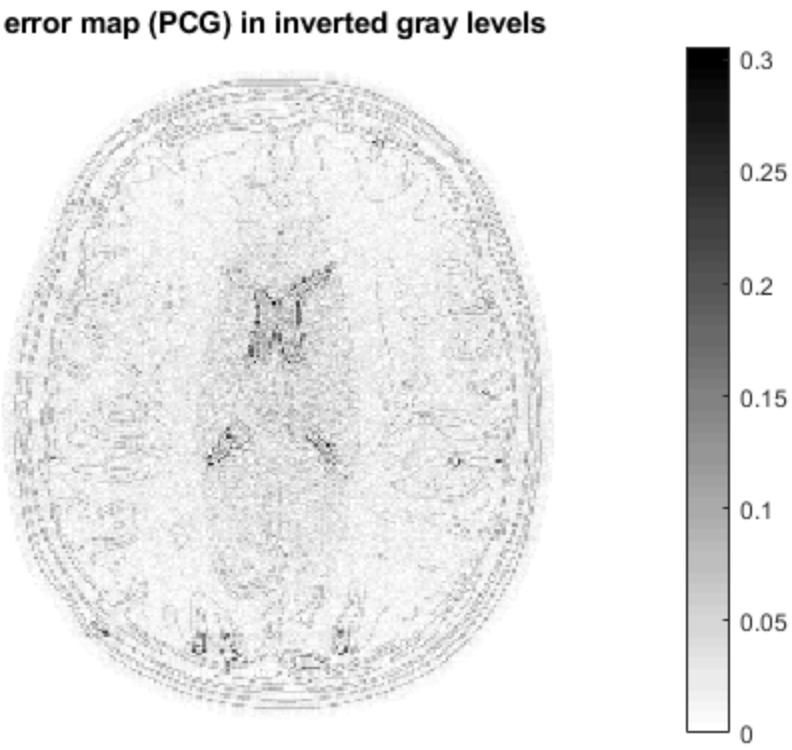
These plots show the results for cg method with no preconditioning to demostrate the effect of preconditioning.The reconstructed image, error and error over time are shown.

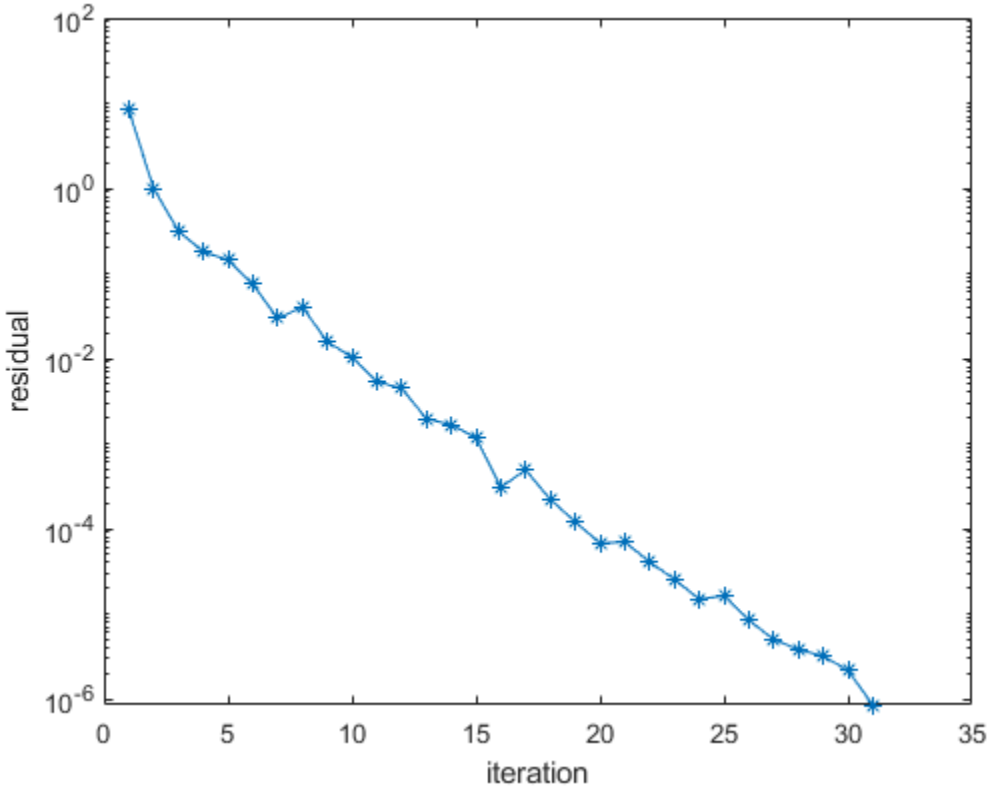
Preconditioned Conjugate Gradient Method 1st order Tikhonov Regularisation with Prior

```
lambda = 1e-3*max(abs(a(:)));
[x, t, d, ~] = conjugateGradient(@(x) A(x) + lambda*x, a, 1e-9, 30, P,
 a);
disp(['PCG with preconditioning and regulariser took ' num2str(t) '
 iterations and has an error of ' num2str(d(end)) '.']);
err = x-ref;
figure;imagesc(abs(x));colormap gray;axis off;colorbar;title('reconstructed
 image (PCG)');
figure;imagesc(abs(err));colormap(1-
gray);axis off;colorbar;title('error map (PCG) in inverted gray
 levels');
figure;semilogy(d,'*-');xlabel('iteration');ylabel('residual');
```

PCG with preconditioning and regulariser took 30 iterations and has an error of 8.5261e-07.




error map (PCG) in inverted gray levels
0.3
0.25
0.2
0.15
0.1
0.05
0

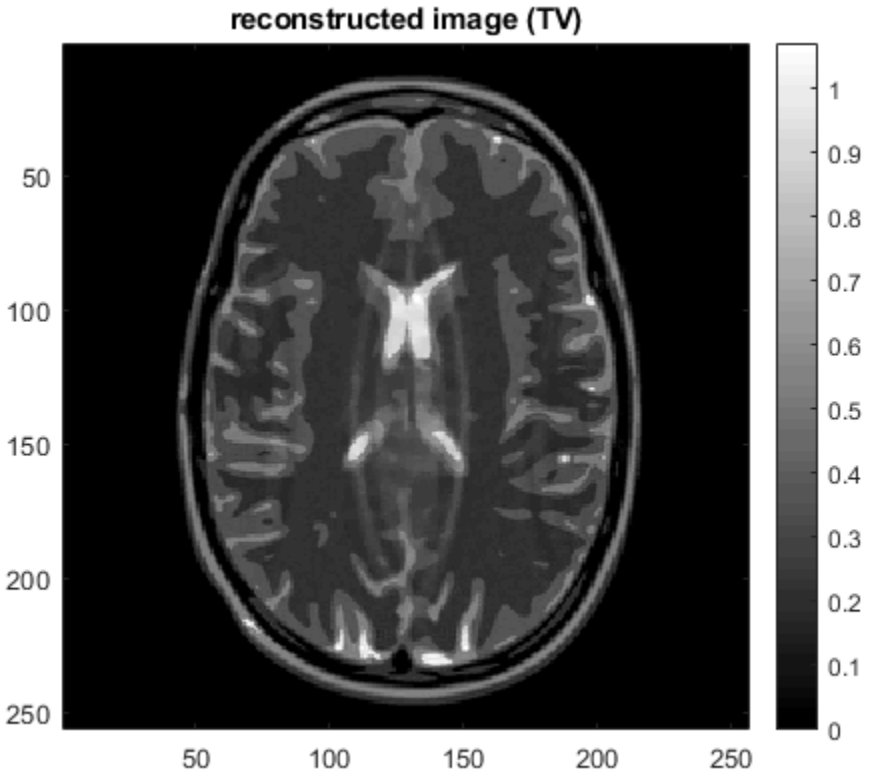

residual
iteration
10^{2}
10^{0}
10^{-2}
10^{-4}
10^{-6}
0
5
10
15
20
25
30
35

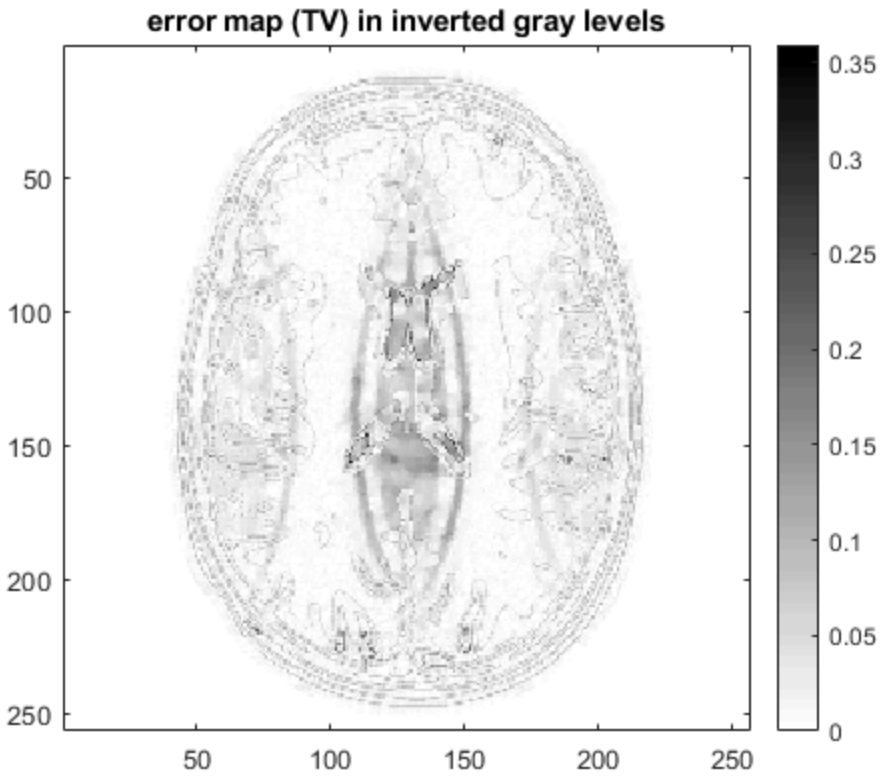
These plots show the results for cg method with preconditioning and regularisation. The reconstructed image, error and error over time are shown.

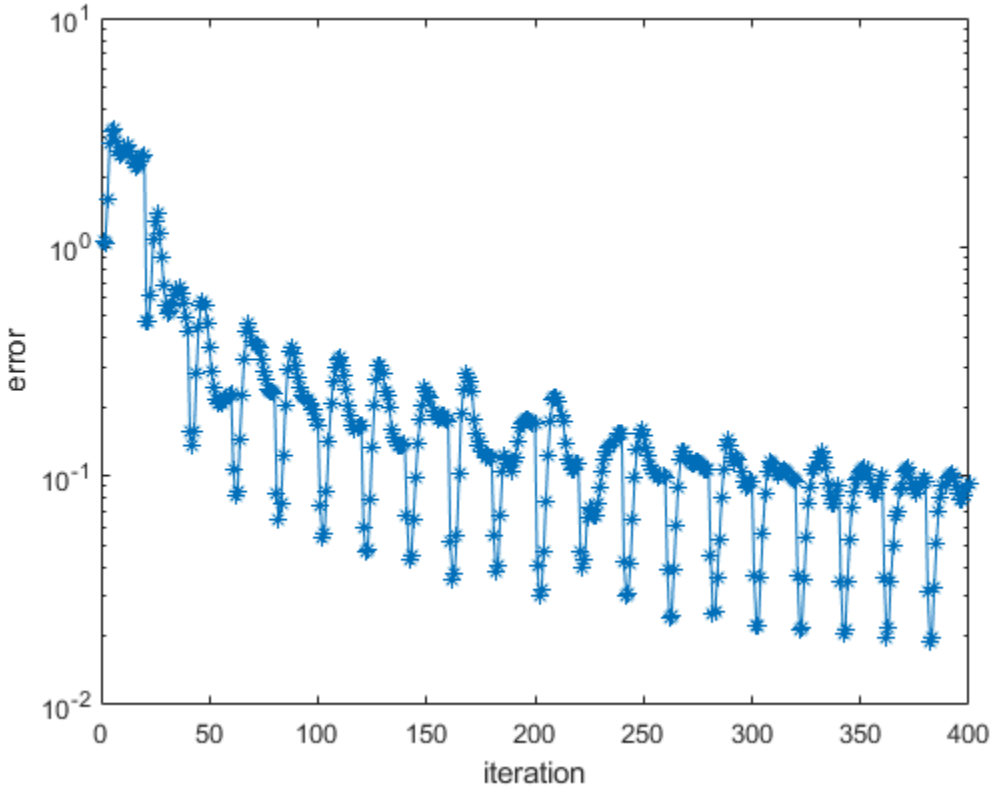
Total Variation Method

```
[x, t, d, info] = totalVariation(A, a, lambda, 1e-3, 20, P, a);
disp(['TV with preconditioning took ' num2str(t(end)) ' iterations and
 has an error of ' num2str(d(end)) '.']);
err = x-ref;
figure;imagesc(abs(x));colormap gray;axis image;colorbar;title('reconstructed
 image (TV)');
figure;imagesc(abs(err));colormap(1-
gray);axis image;colorbar;title('error map (TV) in inverted gray
 levels');
figure;semilogy(d,'*-');xlabel('iteration');ylabel('error');
```

TV with preconditioning took 58 iterations and has an error of 0.092971.




error map (TV) in inverted gray levels
50
100
150
200
250
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0


error
iteration
10^{1}
10^{0}
10^{-1}
10^{-2}
0
50
100
150
200
250
300
350
400

These plots show the results for the total variation iteratively re-weighted weights. The reconstructed image, error and error over time are shown.

The results show the PCG with regularisation is the best performing algorithm. From the results, the prior preconditioning matrix has a large affect on the solution. The total variation method does perform very poorly which is probably due to implementation error.

Published with MATLAB® R2018a