Compressed Sensing (CS) Workshop: Basic Elements of Compressed Sensing

Mariya Doneva
Philips Research Europe

Almir Mutapcic
Sarajevo School of Science and Technology

Miki Lustig
EECS Department
UC Berkeley, CA, USA

Ohrid, June 17, 2011

Outline

Part I: Sparse Signals and Denoising

Part II: Sparsity of Medical Imaging

Part III: Compressed Sensing MRI

Part I: Sparse Signals and Denoising

Overview:

- sparsity
- incoherency
- sparsity based reconstruction

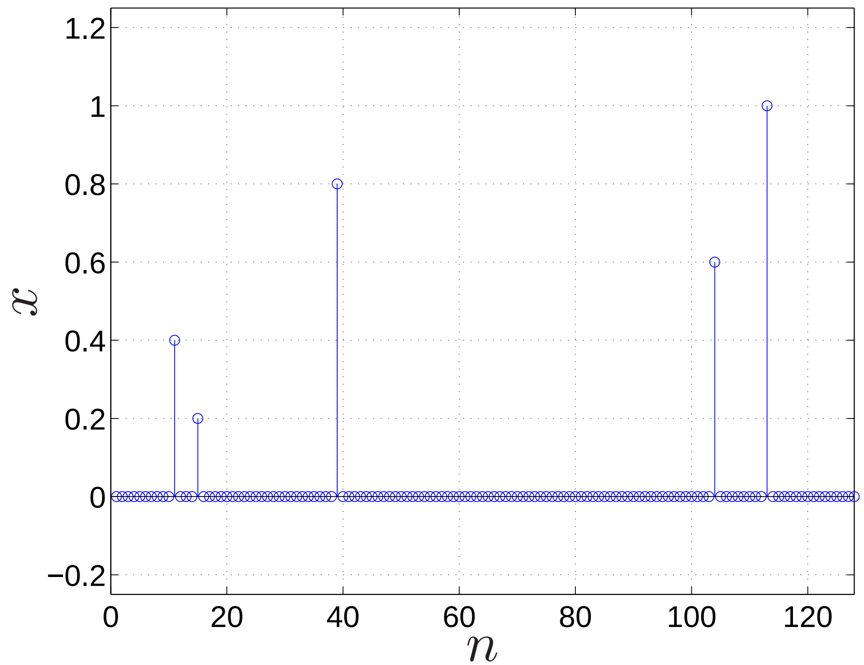
Sparse Signals and Denoising in 1D

- strong connection between CS and sparse signal denoising
- the sparsity of signal $x \in \mathbf{R}^n$, is the number of zero components of x
- similarly, the cardinality of x, $\mathbf{card}(x)$, is the number of nonzeros, we often use $||x||_0$ to denote cardinality

Sparse signal example

Generate $x \in \mathbf{R}^{128}$ with 5 nonzero coefficients (randomly permuted)

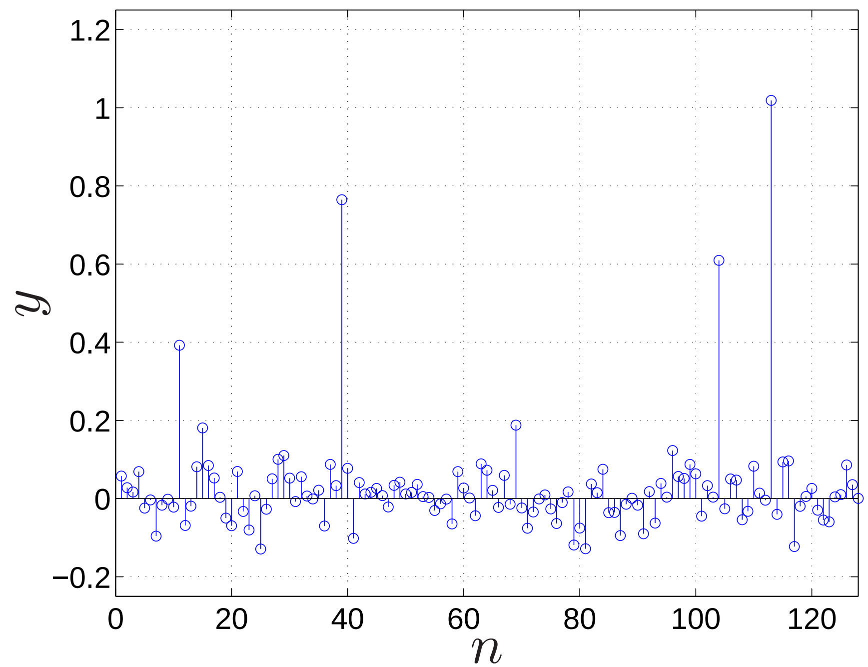
```
>> x = [[1:5]/5 zeros(1,128-5)];
>> x = x(randperm(128));
```



Corrupted sparse signal

Corrupt sparse signal with random Gaussian noise $\sigma = 0.05$ $(y = x + n)$

```
>> y = x + 0.05*randn(1,128);
```



Denoising

Many approaches for denoising (or regularization), *i.e.*, estimation of the signal from noisy data:

- ℓ_2-norm or Tychonov penalty
- ℓ_∞-norm or minimax
- ℓ_1-norm penalty (more on this soon)

ℓ_2-norm denoising

This optimization trades the norm of the solution with data consistency.

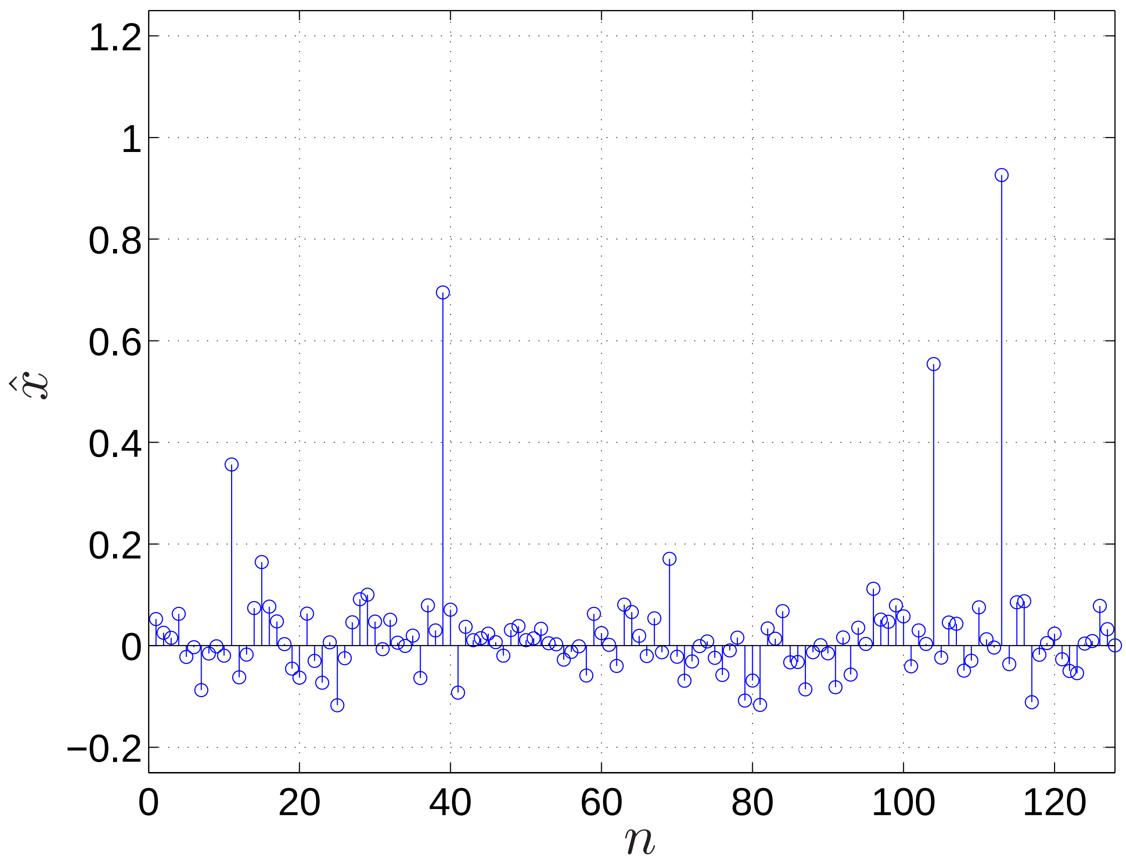
$$\text{argmin}\ \frac{1}{2}||\hat{x} - y||_2^2 + \lambda\frac{1}{2}||\hat{x}||_2^2$$

The solution for this problem is

$$\hat{x} = \frac{1}{1+\lambda}y$$

Sample solutions

Observe what happens when plot result for $\lambda = 0.1$



Is the solution sparse?

Sparse signals and the ℓ_1-norm

Now we will penalize the ℓ_1-norm, $i.e.$,

$$||x||_1 = \sum |x_i|$$

Specifically we will solve:

$$\text{argmin } \frac{1}{2}||\hat{x} - y||_2^2 + \lambda||\hat{x}||_1$$

Solution

Variables $\hat{x}_i$'s are independent, so minimize each seperately by solving

$$\text{argmin } \frac{1}{2}|\hat{x}_i - y_i|^2 + \lambda|\hat{x}_i|$$

The solution to each $\hat{x}_i$ has a closed form. The solution is

$$\hat{x} = \begin{cases} y + \lambda & \text{if} \quad y < -\lambda \\ 0 & \text{if} \quad |y| < \lambda \\ y - \lambda & \text{if} \quad y > \lambda \end{cases}$$

(This is called soft-thresholding or shrinkage).

- Show Movie

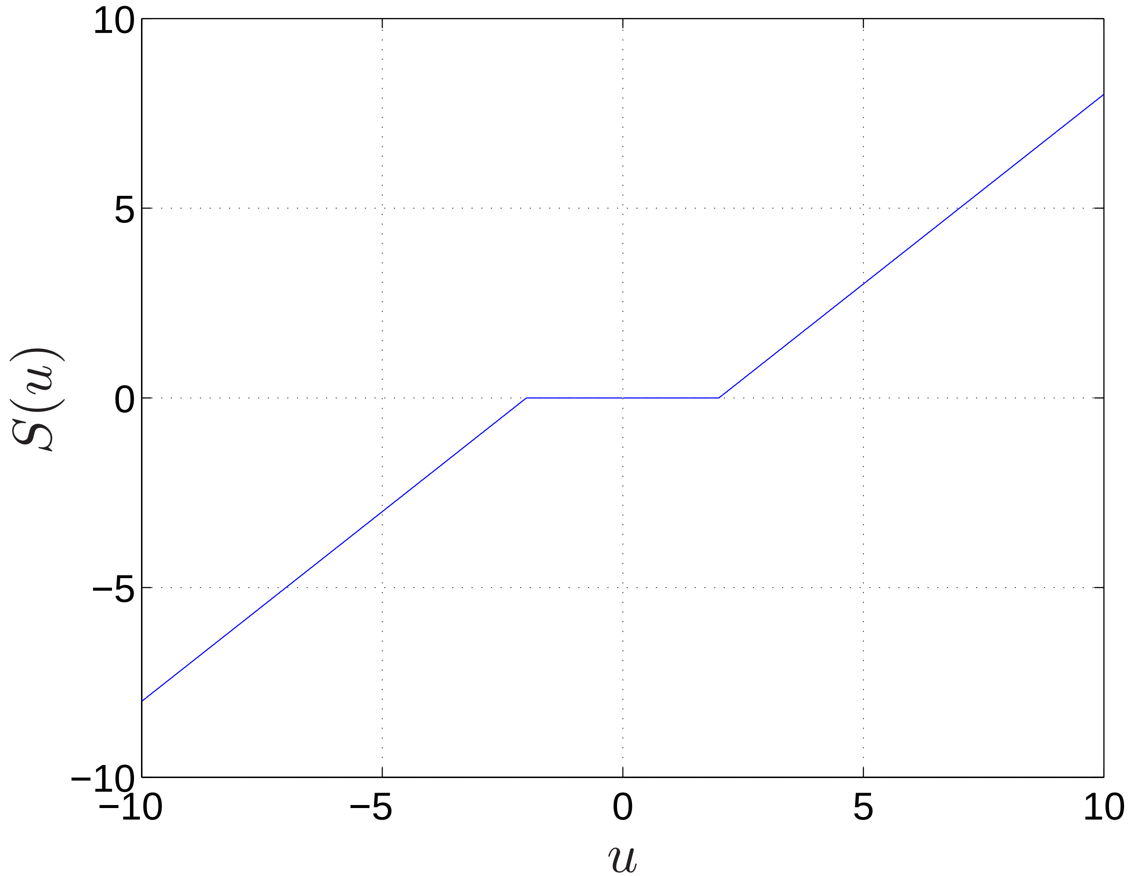
Soft thresholding or shrinkage function

`SoftThresh` (complex input case) function:

$$S(u,\lambda) = \begin{cases} 0 & \text{if } |u| \leq \lambda \\ \frac{(|u|-\lambda)}{|u|}u & \text{if } |u| > \lambda \end{cases}$$

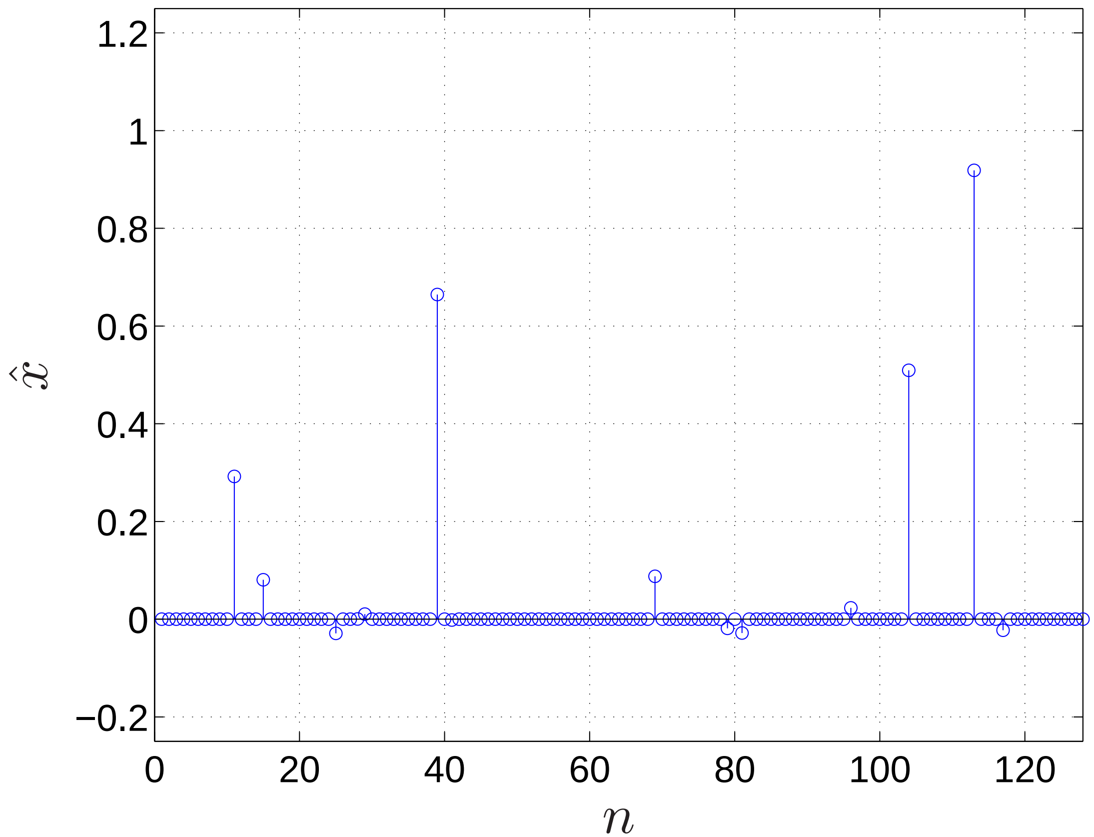
Matlab implementation

Write a function `SoftThresh` that accepts u and λ and returns $S(u)$.
Plot for $u \in [-10, 10]$ and $\lambda = 2$.



Back to our example

Apply `SoftThresh` to the noisy signal with $\lambda = 0.1$.



Is the solution sparse?

Random Frequency Domain Sampling and Aliasing

- a strong connection between compressed sensing and denoising
- explore this connection and the importance of incoherent sampling
- in compressed sensing, we undersample the measurements
- measure subset of k-space, $X_u = F_u x$ where F_u is a Fourier transform evaluated only at a subset of frequency domain samples.

Example: Uniform vs random undersampling

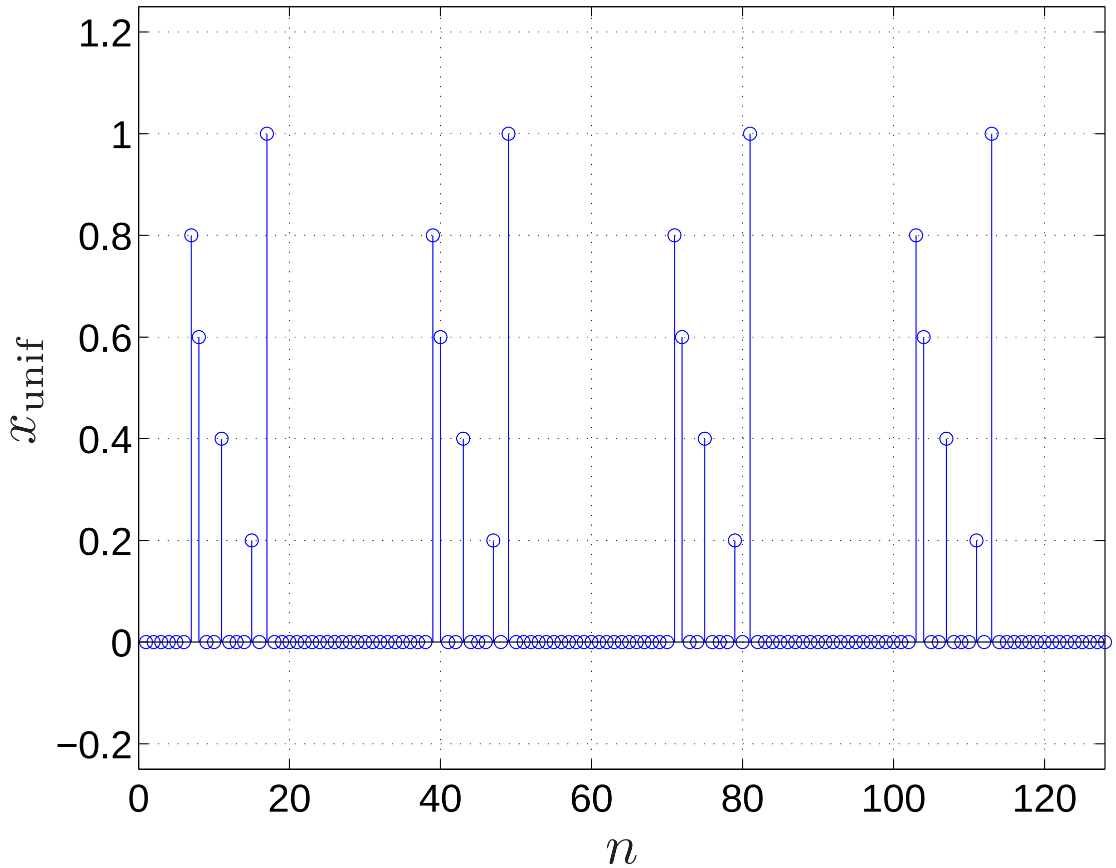
- start with the Fourier transform of a sparse signal
- undersample k-space by taking 32 equispaced samples
- compute the inverse Fourier transform, filling the missing data with zeroes
- multiply by 4 to correct for the fact that we have only 1/4 the samples

```
>> X = fftc(x);
>> Xu = zeros(1,128);
>> Xu(1:4:128) = X(1:4:128);
>> xu = ifftc(Xu)*4;
```

this is uniform sampling and minimum ℓ_2 norm solution (why?).

Result in signal domain

Plot of the absolute value of the result. Describe what you see.



Will we be able to reconstruct the original signal from the result?

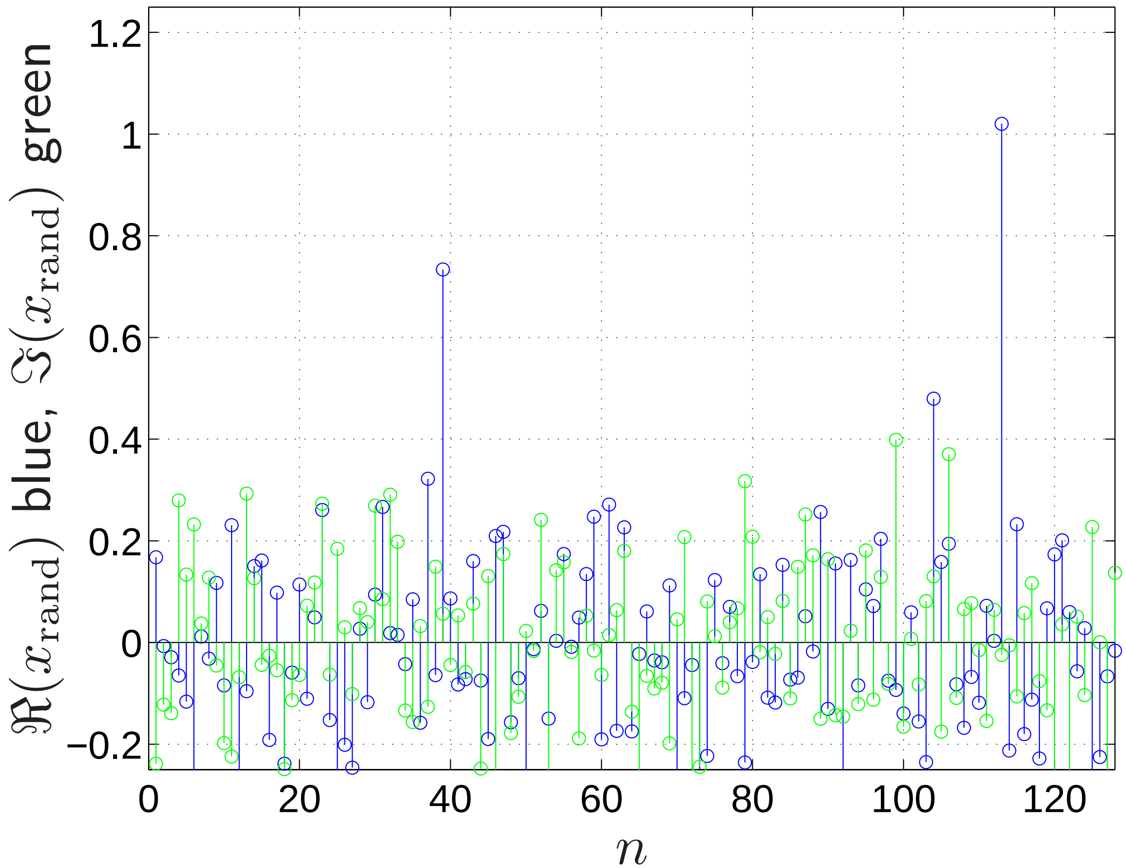
Random sampling

Now, undersample k-space by taking 32 samples at random.

```
>> X = fftc(x);
>> Xr = zeros(1,128);
>> prm = randperm(128);
>> Xr(prm(1:32)) = X(prm(1:32));
>> xr = ifftc(Xr)*4;
```

Results

Plot the real and imaginary value, and describe the result.



Reconstruct the original signal?

- Will we be able to reconstruct the signal from the result?
- How does this resemble the denoising problem?

This is the important part, so say it out loud:

By random undersampling, we've turned the ill-conditioned problem into a sparse signal denoising problem.

Reconstruction from Randomly Sampled Frequency Domain Data

Inspired by the denoising example, we will add an ℓ_1 penalty and solve,

$$\text{argmin } \frac{1}{2}||F_u\hat{x} - Y||_2^2 + \lambda|\hat{x}|_1$$

- $\hat{x}$ is the estimated signal
- $F_u\hat{x}$ is the undersampled Fourier transform of the estimate
- Y are the samples of the Fourier transform that we have acquired

variables are coupled through FT, no closed-form solution

Iterative solution algorithm

Projection Over Convex Sets (POCS) type algorithm, iterate between soft-thresholding and constraining data consistency

Let $\hat{X} = F\hat{x}$. Initially set $\hat{X}_0 = Y$.

1. Compute inverse FT to get signal estimate $\hat{x}_i = F^* \hat{X}_i$
2. Apply SoftThresh $\hat{x}_i = S(\hat{x}_i, \lambda)$ in the signal domain
3. Compute the FT $\hat{X}_i = F\hat{x}_i$
4. Enforce data consistency in the frequency domain

$$\hat{X}_{i+1}[j] = \begin{cases} \hat{X}_i[j] & \text{if } Y[j] = 0 \\ Y[j] & \text{otherwise} \end{cases}$$

5. Repeat until $||\hat{x}_{i+1} - \hat{x}_i|| < \epsilon$

Matlab implementation

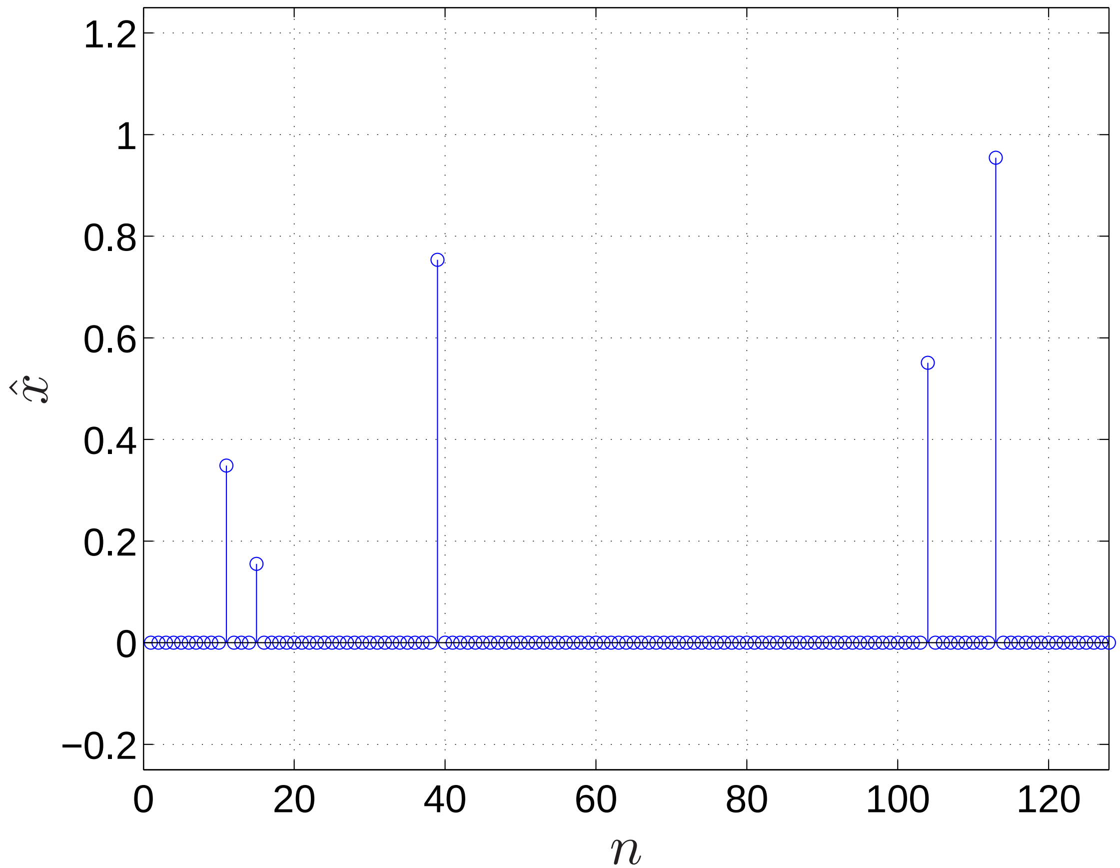
- `Y` is randomly sampled Fourier data with zeros for non-acquired data
- Initialize estimate of Fourier transform of the signal as `X = Y`

The core of the iteration can then be written as

```
>> x = ifftc(X);
>> xst = SofthThresh(x,lambda);
>> X = fftc(xst);
>> X = X.*(Y==0) + Y;
```

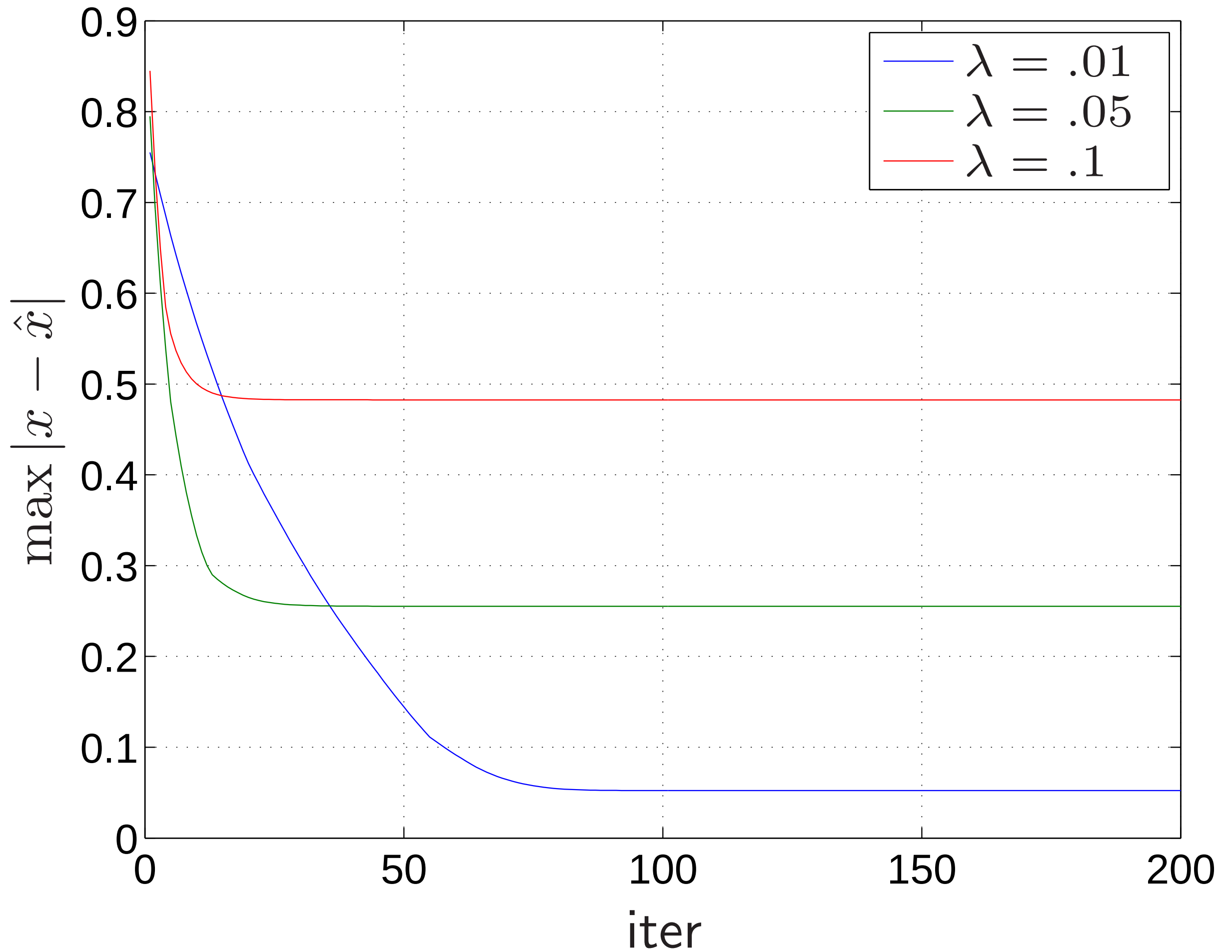
Results

Apply the algorithm (at least 300 iterations) to the undersampled signal with $\lambda = \{0.01, 0.05, 0.1\}$ and plot the results.



Plots

Make a plot of error between the true x and $\hat{x}_i$ as a function of the iteration number, plotting the result for each of the λs.



Part II: Sparsity of Medical Imaging

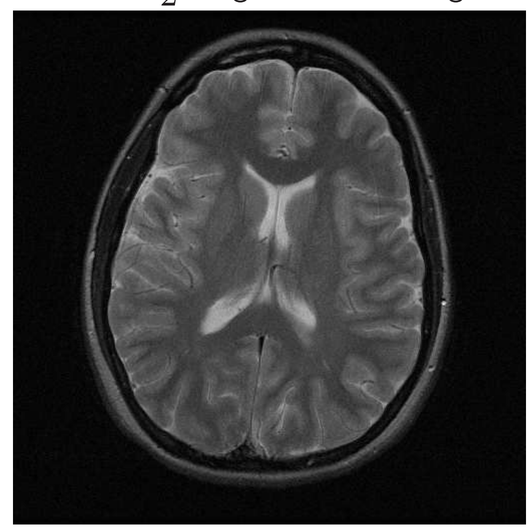
- Medical Images are generally not sparse.
- Images have a sparser representation in a transform domain
- The transform depends on the type of signal

Sparsity of Brain Scans

The file `brain.mat` contains a very pretty axial T_2-weighted FSE image of a brain stored in the matrix `im`. Load the file and display the magnitude image

```
>> load brain.mat
>> figure, imshow(abs(im),[])
```

Axial T_2-weighted Brain image



Is the brain image sparse?

The Wavelet Transform

The Wavelet transform is known to sparsify natural images.

- Orthogonal transformation (Here)
- Wavelet coefficients are band-pass filters
- Coefficients hold both position and frequency information
- There are many kinds of wavelets (Haar, Daubechies, Symmlets,...)
- Fast to compute

Matlab Implementation

- Original code from Wavelab (David Donoho)
  `http://www-stat.stanford.edu/~wavelab/`

- The Matlab class @Wavelet implements the Wavelet transform

- Usage:

```
>> W = Wavelet; % Daubechies-4 wavelet operator
>> im_W = W*im; % Forward Wavelet transform
>> im_rec = W'*im_W; % Inverse Wavelet transform
```

- The function imshowWav.m conveniently displays wavelet coefficients.

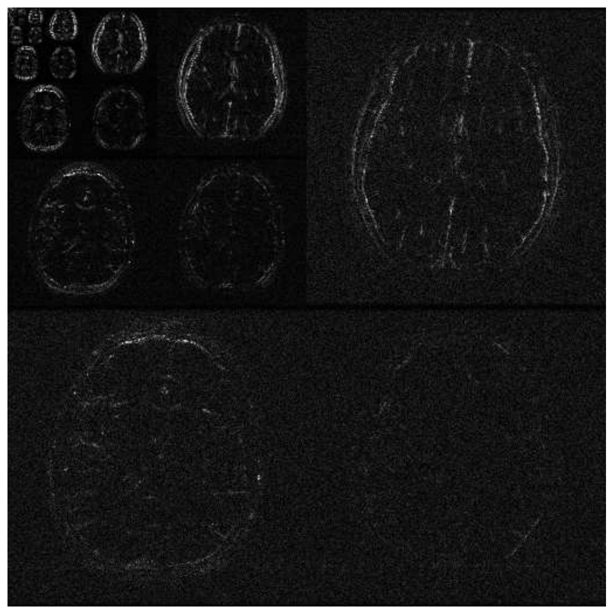
```
>> Figure, imshowWAV(im_W)
```

Wavelet Transform of a Brain Scan

Compute the Wavelet transform of the brain images and display the coefficients.

```
>> W = Wavelet; % Daubechies-4 wavelet operator
>> im_W = W*im; % forward Wavelet transform
>> figure, imshowWAV(im_W)
```

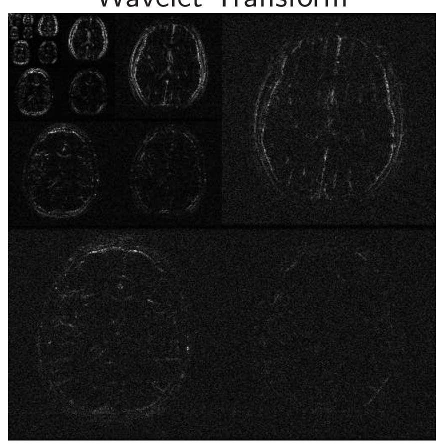
Wavelet Transform



Sparsity in The Wavelet Domain

- Each band of wavelet coefficients represent a scale (frequency band) of the image.
- The location of the wavelet coefficient within the band represent its location in space.
- What you see are edges of the image at different resolutions and directions.

Wavelet Transform



Is the signal sparse?

Wavelet Thresholding

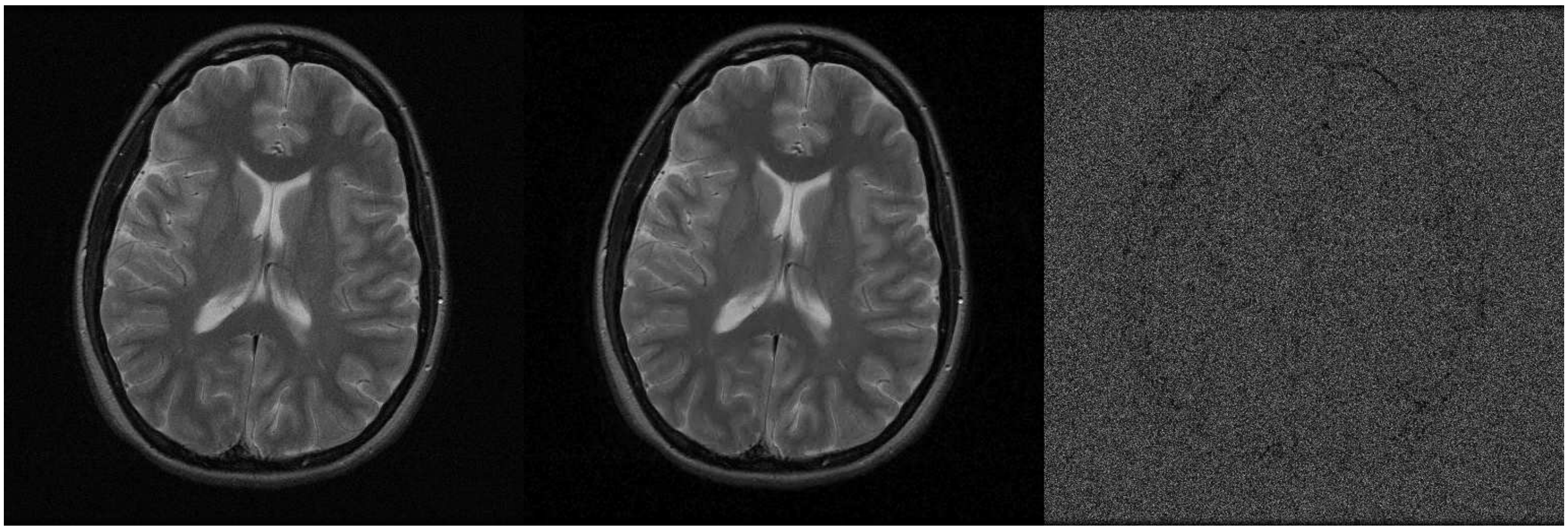
Threshold the wavelet coefficients retaining only the largest 10% of the coefficients. Plot the reconstructed image. (Take a note of the threshold for later)

- Show Movie

```
>> m = sort(abs(im_W(:)),'descend');
>> ndx = floor(length(m)*10/100);
>> thresh = m(ndx);
>> im_W_th  = im_W .* (abs(im_W) > thresh);
>> im_denoise = W'*im_W_th;
>> figure, imshow(abs(cat(2,im,im_denoise, ...
(im-im_denoise)*10)),[0,1]);
```

Wavelet Denoising

Original | Thresholded 15% | Difference (x10)

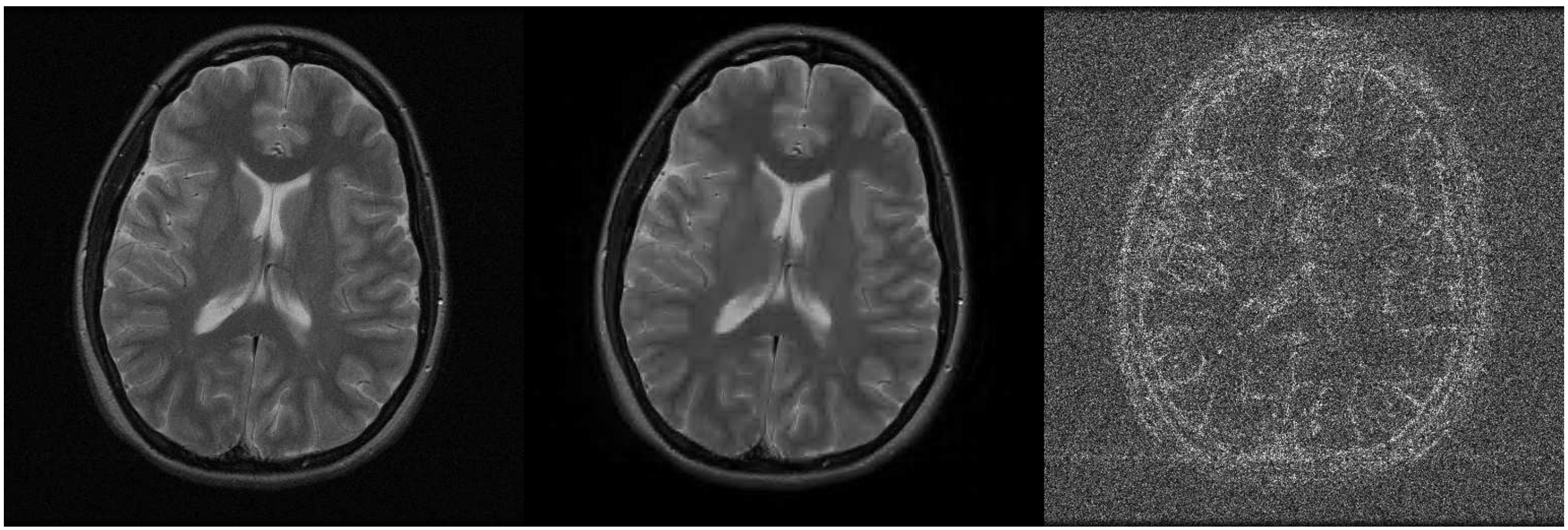


Q) What has been thresholded?
A) The wavelet transform sparsifies the brain image, and concentrates the "important" image energy into a subset of the coefficients. This helps us denoise the image by thresholding the coefficients which contain mostly noise!

Wavelet Over Denoising

Repeat the experiment with a threshold of 2.5%

Original | Thresholded 2.5% | Difference (x10)



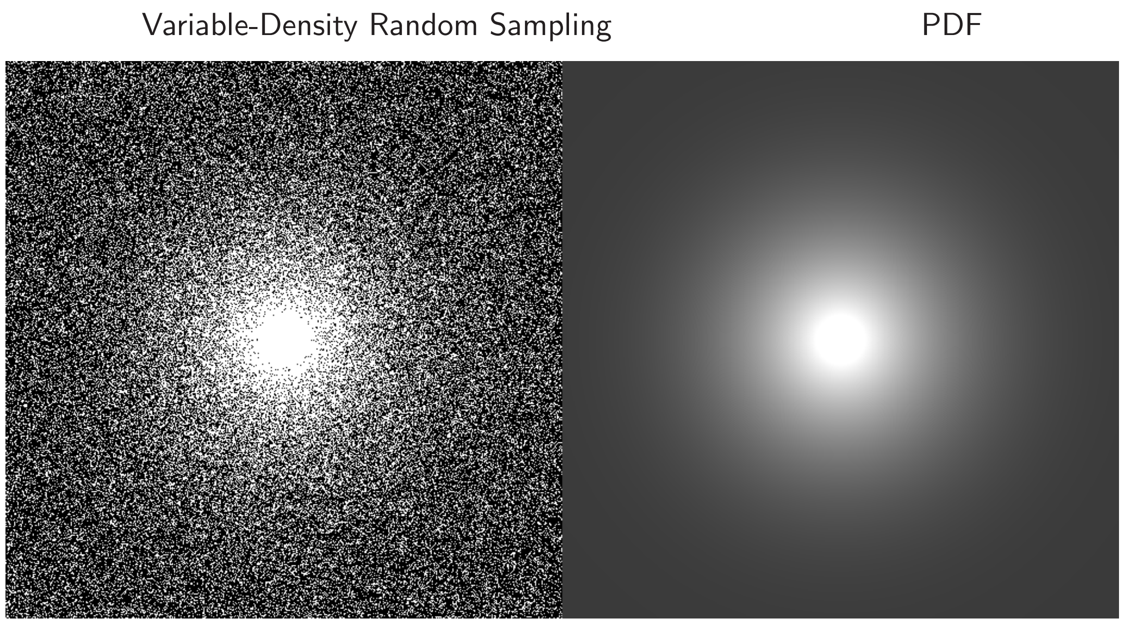
What have been thresholded?
What's the approximate sparsity of the image?

Part III: Compressed Sensing MRI

- In MRI $\#$ of measurements $\propto$ scan time
- Reduce samples to reduce time
- Extrapolate missing samples by enforcing sparsity in transform

Variable-Density Random Sampling

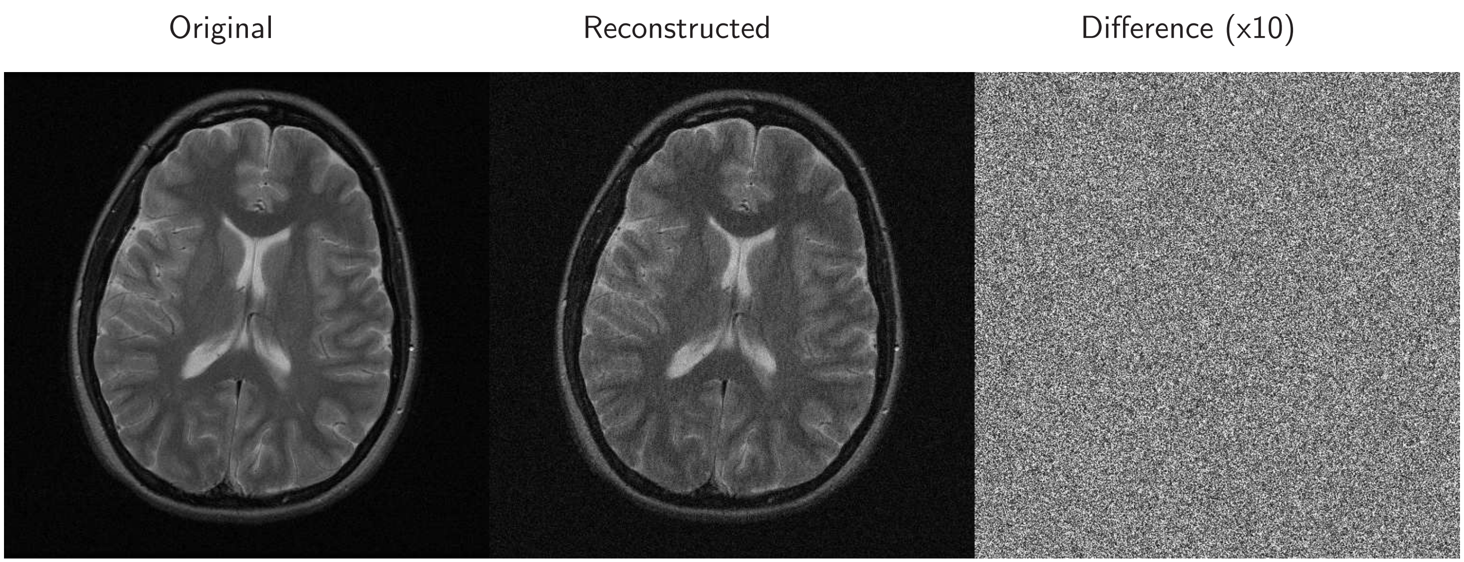
The variable `mask_vardens` is a $\times 3$-fold subsampled, variable-density random mask, drawn from a probability distribution given by `pdf_vardens`.



Linear Reconstruction

Compute the 2D Fourier transform of the image. Multiply with the mask, divide by the PDF. Compute the inverse Fourier transform and display the result.

```
>> M = fft2c(im);
>> M_us = (M.*mask_vardens)./pdf_vardens;
>> im_us = ifft2c(M_us);
>> figure, imshow(abs(cat(2,im,im_us, (im_us-im)*10)),[0,1])
```



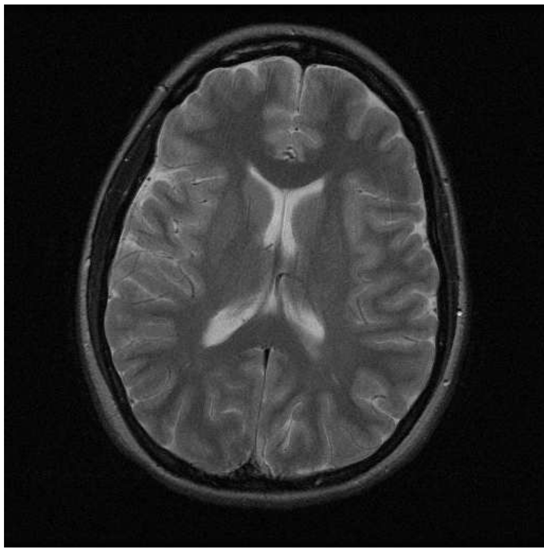
Compressed Sensing MRI Reconstuction

Implement the POCS algorithm for 2D images. Use lambda value from the thresholding experiment. Use 20 iterations.

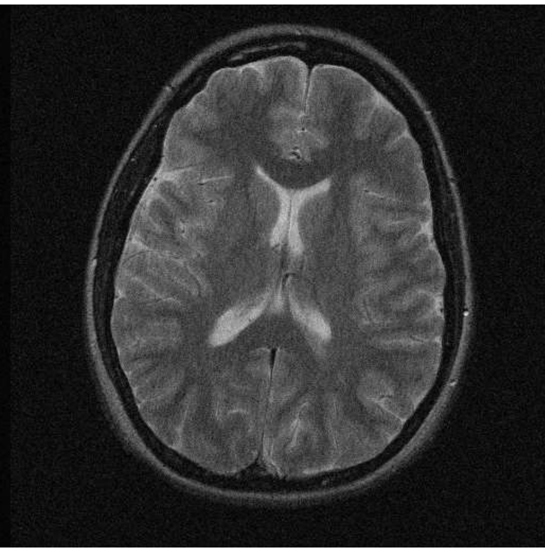
```
>> DATA = fft2c(im).*mask_vardens;
>> im_cs = ifft2c(DATA./pdf_vardens); % initial value
>> figure;
>> for iter=1:20
>>im_cs = W'*(SoftThresh(W*im_cs,0.025));
>>im_cs = ifft2c(fft2c(im_cs).*(1-mask_vardens) + DATA);
>>imshow(abs(im_cs),[]), drawnow;
>> end
```

Results

Original



Linear



Compressed Sensing