EEE4114F - Compressive Sensing

STNJOS005

April 2019

1 Introduction

Compressive sensing (CS) is a sensing modality, used to obtain compressed signal information at the time of sensing [1]. The idea stems from the fact that many signals we capture have an underlying sparsity. Concretely, if we consider a signal as an $N \times 1$ column vector, then for K elements the signal is 0, where $K << N$. For example, sinusoidal waves in the time domain become spikes in the frequency domain, images become sparse in the discrete cosine transform (DCT) and wavelet domains, etc. This sparseness is often used in compression. For example, after capturing a large raw image, we often compress to JPEG (using the DCT transform) or JPEG-2000 (using the wavelet transform), throwing away $> 90\%$ of the image while maintaining satisfactory quality. The question therefore arises whether it is possible to capture this compressed signal *at the time of sensing*. That is, instead of capturing a rich signal and then compressing it, can we capture a compressed signal and reconstruct it as if we had captured a rich signal? This has become increasingly important as we move further into an age where we generate far more data than we can store or process efficiently [2]. Is it possible to collect less data, but still obtain satisfactory results? Traditional signal processing, which follows the Nyquist-Shannon sampling theorem [3], states that this is not possible [4]. However, as has been demonstrated with CS, it *is* possible to sample at sub-Nyquist rates and still obtain excellent results.

CS was introduced in 2004 by Donoho, Candes, Romberg and Tao [5][6][7]. Despite its young age, the field has seen an explosion in research, acquisition techniques, reconstruction techniques and applications. CS has found widespread use in areas where sensing can be very expensive, areas where a limited number of samples can be taken and/or where sensing takes significant time. A good resource containing much of the literature can be found at [8]. In this investigation I aim to use some of these techniques to investigate how CS can be applied for the purpose of image reconstruction.

2 Aim

This report has three aims:

1. To understand compressive sensing, and how it is possible to sample at sub-Nyquist rates.
2. To present a brief literature review of CS reconstruction algorithms.
3. To use CS acquisition and reconstruction strategies to sample and reconstruct 2D images.

3 How CS works

3.1 Acquisition [9] [4] [1]

Consider a signal $\mathbf{x}$, which we consider as a $N \times 1$ column vector in $\mathbb{R}^N$ with elements $x[n], n = 1, 2, ..., N$. Any signal can be represented by a linear combination of its basis vectors - in this case, an orthonormal basis Ψ. Then, we can express $\mathbf{x}$ as

$$\mathbf{x} = \sum_{i=1}^{N} s_i \psi_i \quad \text{or} \qquad \mathbf{x} = \Psi \mathbf{s} \tag{1}$$

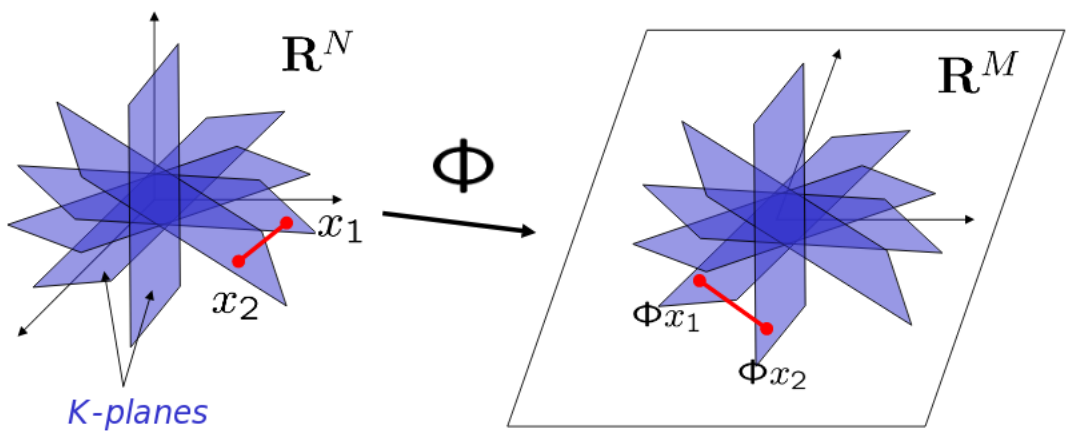

Figure 1: RIP keeping Euclidean distances the same after sensing.

where $\mathbf{s}$ is an $N \times 1$ column vector of weighting coefficients. In other words, $\mathbf{x}$ is a signal in the time domain, and $\mathbf{s}$ is the *same* vector in the Ψ domain. One of the keys to CS is to understand that in this domain $\mathbf{s}$ is sparse. In compression, we acquire $\mathbf{x}$, compute the coefficients, keep K large coefficients (discarding the rest) and then encode these largest K coefficients. Instead of acquiring the full signal $\mathbf{x}$, we would like to sample a portion of the signal (M measurements, where $M < N$). If we have an $M \times N$ sampling matrix Φ, then we can construct a measurement column vector $\mathbf{y}$ such that

$$\mathbf{y} = \Phi\mathbf{x} = \Phi\Psi\mathbf{s} = \Theta s \tag{2}$$

where $\Theta = \Phi\Psi$ is an $M \times N$ matrix. In general, Φ is rank deficient ($M < N$). Consequently, there are infinitely many $\mathbf{x}$s that can give $\mathbf{y}$. However, if we remember that $\mathbf{x}$ is sparse, then the problem becomes simplified. In this sparse case, $\mathbf{y}$ is just a linear combination of K columns of Φ whose locations correspond to the non-zero entries of $\mathbf{x}$. However, we don't know where the non-zero entries are. We are therefore faced with the challenge of designing a sensing matrix Φ which, if K columns are taken arbitrarily, are full rank. Further, we want those columns to be close to orthogonal to one another - this guarantees stable recovery. This is known as the *restricted isometry property* (RIP):

$$1 - \delta_k \leq \frac{\|\Theta x\|_2}{\|x\|_2} \leq 1 + \delta_k \tag{3}$$

where δ_k is a known constant and x is a vector having the same k-nonzero entries as $\mathbf{x}$. $\|\cdot\|_2$ represents the l_2 norm, where the l_p norm is defined as:

$$l_p : \|x\|_p = (\sum_{i=1}^{n} \|x_i\|^p)^{\frac{1}{p}} \tag{4}$$

Concretely, the RIP property states that sampling $\mathbf{x}$ with Φ approximately preserves the Euclidean distances between elements of $\mathbf{x}$. This can be seen in figure 1 [10].

This is important, because when we take compressive measurements we don't want measurements to 'overlap', or be confused with one another. That is, during the sampling of $\mathbf{x}$ with Θ, we don't want our measurements to be mapped to the same point. In other words, we want to preserve the distances between measurements to maintain their uniqueness. In practice, determining δ is difficult [1]. In the literature, δ is often determined empirically, and different values are recommended for different algorithms. An alternative approach to ensure stability is to ensure that the measurement matrix Φ and the sparsifying matrix Ψ are incoherent. Coherence is a measure of correlation among elements of matrices and is defined as

$$u(\phi, \psi) = \sqrt{n} \max_{1 \leq i,j \leq n} |\langle \phi_i, \psi_j \rangle| \tag{5}$$

where the range of coherence is $u(\phi, \psi) \in [1, \sqrt{n}]$. If Φ and Ψ contain correlated elements, the coherence is large. In general, we desire maximal incoherence (i.e. minimal coherence). For example, spikes and sinusoids are maximally incoherent. Unfortunately, designing a matrix with these properties is an NP-hard problem.

Fortunately, it turns out that many naturally occurring matrices fulfill these properties! For example, independent and identically distributed (iid) random variables (Gaussian, Bernoulli, etc.) obey the RIP property with high probability provided

$$M \geq cKlog(\frac{N}{K}) \tag{6}$$

Where c is a small constant. Further, due to the nature of iid Gaussian distributions, the matrix $\Theta = \Phi\Psi$ is also iid Gaussian (and therefore also obeys the RIP property) regardless of the sparsifying basis Ψ chosen[1]. It is for this reason that the random Gaussian matrices are thought of as universal. This leads to a surprising result: in order to get good measurements, we should acquire correlations with random wave-forms!

3.2 Reconstruction [9] [4] [11] [2]

Consider again the linear algebra problem to be solved in (2). What we see is that there are infinitely many solutions (that is, there are infinitely many **s'** that satisfy $\mathbf{y} = \Theta\mathbf{s'}$). As a reminder, this is because Θ is an $M \times N$ matrix, and in the case of CS $M < N$. Therefore, Θ is rank deficient. The challenge is therefore to find some **s'** that is the sparsest representation of **s**. This presents an optimization problem. Typically, one would minimize the l_2 norm (using least-squares) thereby choosing the **s** with the minimum energy. However, this yields poor results.

An alternative approach is to minimize the l_0 norm - that is, the sum of absolute values of **s**. This would give a perfect solution - however, it is an NP hard problem that requires exhaustive enumeration of all combinations of nonzero entries in **s**.

The key insight developed in CS is that we can solve the l_1 norm. To understand why, consider that the sparse solution lies on along the axis of some higher-dimension plane. In the 3D plane, you can image a sparse vector being (1, 0, 0), (0, 1, 0) or (0, 0, 1) - in all cases, it is easy to see that these solutions lie on one of the dimension axes. Further, consider that the problem we are solving is a plane (see figure 2 [10]). We are trying to find the solution on this plane that yields the sparest results.

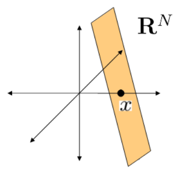

Figure 2: Plane representing all possible solutions to (2)

The l_2 norm can be visualised as a ball (see figure 3 [12]). What we see is that the solution which intersects with this l_2 norm ($\hat{\mathbf{s}}$) is significantly far away from the true sparsest solution (**s**). In contrast, the l_1 norm can be thought of as a pointy diamond. We see that the intersection between the l_1 norm and the true sparse solution is much closer to the intersection between the l_2 norm and the true sparse solution.

For accurate reconstruction, we therefore need to solve the l_1 optimization problem:

$$\hat{\mathbf{s}} = \text{argmin}\|\mathbf{s'}\|_1 \quad \text{such that} \quad \mathbf{y} = \Theta\mathbf{s'} \tag{7}$$

This method of reconstruction is known as *basis pursuit* and will be explored further in this literature review. The complexity of solving such a problem is $O(n^3)$. Solving the l_1 norm is estimated to be about "30-50 times as expensive as solving the least-squares [l_2 norm] problem" [11].

[1]random matrices are also largely incoherent with any fixed basis Ψ

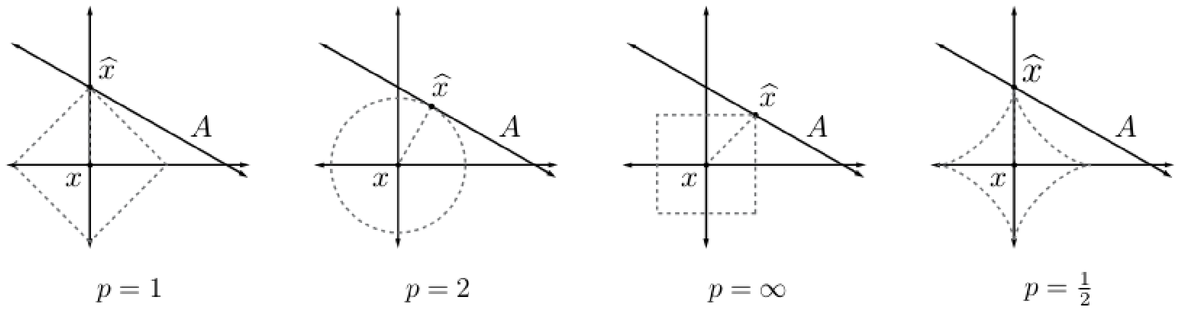

Figure 3: Shape of different norms in 2 dimensions. The line represents the possible solutions we are searching for. The true sparse solution is on the vertical axis. We see the intersection between the solution space and the l_1 norm is much closer compared to the solution space and the l_2 norm. Using $0 < p < 1$ is also appropriate, but requires non-convex solvers to find the solution.

4 Literature Review

Broadly speaking, recovery algorithms can be put into 5 major categories: convex approaches, greedy algorithms, bayesian frameworks, non-convex approaches and brute force methods [13].

4.1 Convex approach

Basis pursuit [14] is convex optimization problem which searches for the minimum l_1 norm, subject to (7). Basis pursuit depends on having noise free measurements. More robust recovery schemes that incorporate noisy measurements include basis pursuit denoising (BPDN) [14], the Dantzig selector [15] and total variation (TV) denoising [16].

BPDN searches for a solution having minimum l_1 norm subject to:

$$\hat{\mathbf{s}} = \text{argmin}\|\mathbf{s'}\|_1 \quad \text{such that} \quad \frac{1}{2}\|\mathbf{y} - \Theta\mathbf{s'}\|_2^2 \leq \epsilon \tag{8}$$

The Dantzig selector searches for a solution:

$$\hat{\mathbf{s}} = \text{argmin}\|\mathbf{s'}\|_1 \quad \text{such that} \quad \frac{1}{2}\|\mathbf{y} - \Theta\mathbf{s'}\|_\infty^2 \leq \epsilon \tag{9}$$

Total variation denoising searches for a solution:

$$\hat{\mathbf{s}} = \text{argmin}\|\mathbf{s'}\|_{TV} \quad \text{such that} \quad \frac{1}{2}\|\mathbf{y} - \Theta\mathbf{s'}\|_2^2 \leq \epsilon \tag{10}$$

where

$$\|x\|_{TV} = \sum_{i,j} \sqrt{|x(i+1,j) - x(i,j)|^2 + |x(i,j+1) - x(i,j)|^2}$$

4.2 Greedy algorithms

Greedy algorithms work by iteratively refining a solution. Each iteration, the solution is updated by finding a solution that is more correlated with the measurements than before. The iterations end either when an iteration limit is reached or when the generated solution approximates the real solution within some constant error. Some well known greedy algorithms are matching pursuit (MP) [17], orthogonal matching pursuit (OMP) [18], gradient pursuit (GP) [19] and COmpressive Sampling MP (CoSaMP) [20].

Matching pursuit algorithms work on the basis of being able to decompose a function, f, into the sum of finitely many weighted functions (s_i) [21].

$$f \approx \hat{f} := \sum_{i=1}^{N} \alpha_i s_{\gamma_i} + r$$

where α_i is some weight, s_{γ_i} is a sparse vector found from the measuremnt matrix Θ and r is some residual.

The basis of these algorithms is [1]:

- Initialize a residual vector, r, to the measurement vector y. Initialize s, a solution set and γ, an index set to be null vectors. A counter is initialized to 1.
- Search for a column in the measurement matrix (Θ) which is maximally correlated with the residual vector. The index of the column is updated in γ.
- Update s with the respective column chosen from Θ.
- Update the residual (by subtracting the product $\Theta_{\gamma_i} s_i$).
- Increment the counter, and continue looping until an iteration limit is reached or a desired value of the residual is reached.

The difference between these algorithms is how s is updated. In MP, the correlation vector between the residual and the measurement matrix is searched for the value with maximum correlation. A unit vector is then constructed, which is 1 at this location and 0 everywhere else. This unit vector is then added to the current sparse solution [1]. In such a way one coefficient weight is changed every iteration.

In OMP, the sparse solution is found by optimizing the correlation column [13]:

$$s_i = arg\min \|r - \Theta_{\gamma_i} y_i\|_2$$

In OMP, all the coefficients are updated every iteration (instead of just one, as in MP) [21]. This requires more computation.

GP works by updating the solution set in a gradient direction. The direction chosen is that which minimizes the same optimization problem in OMP [22].

CoSaMP is a parallel algorithm that works similarly to the above, but selects multiple columns of Θ each iteration. These multiple columns are compared to the columns from the previous iteration, keeping only the best columns after least squares optimization [1].

Pseudo-code for all of these algorithms can be found in [22].

4.2.1 Thresholding

Thresholding algorithms are another type of greedy algorithm by iteratively applying a thresholding operator to multiple columns of Θ every iteration.

Iterative hard thresholding (IHT) [23] works according to:

$$s = n_k(s + \lambda \Theta^T (y - \Theta s)$$

where $n_k(\cdot)$ is a thresholding operator and λ is some step size. The n_k operator works by setting all but the largest elements in a vector to 0 (leaving the remaining components unaffected) [22]. IHT does not always converge.

Soft thresholding [24] works similarly to hard thresholding, except that the thresholding operator is adaptive based on the values in s. Defining n_θ as the soft thresholding operator, the form follows the same as IHT, except n_θ updates as [22]:

$$n_\theta(s) = \begin{cases} s - \theta & \text{if } s > \theta \\ 0 & \text{if } |x| \leq \theta \\ s + \theta & \text{if } s < -\theta \end{cases}$$

4.3 Non-convex approaches

Non-convex approaches work by trying to minimize the l_p norm, where $0 < p < 1$. As can be seen in figure 3, l_p norms where p is in this given range still gives a close approximation to the true sparse solution. This approach requires fewer measurements as compared to minimizing the l_1 norm [25][1].

4.4 Bayesian approach

Bayesian approaches assume that the signal belongs to some known probability distribution. The reconstruction problem therefore becomes a Bayesian inference problem, which can be solved using maximum likelihood estimation (MLE) or maximum a posterio (MAP) estimation [1].

4.5 Brute force

Brute force methods work by searching through all possible solutions in an attempt to find the sparsest vector s [13]. This may be appropriate for smaller data sets.

5 Method and $l1$ minimization

In order to reconstruct an image using l_1 norm minimization, compressive samples first needed to be obtained. As outlined in section 3, one way to do this is to simply use random measurements.

I chose to work in Python. I therefore used numpy's [26] random choice generator to select K measurements (without replacement). Because I was working with images, I used the DCT as my sparsifying basis (the wavelet basis is also an appropriate choice). My images were obtained from [27].

One way to generate a 2D-DCT matrix (as required for the minimization problem) is to use the Kronecker product [28].

Unfortunately, generating a Kronecker product is too computationally intense to run on larger images - it would require far too much memory. I therefore broke down the test image into 16x16 blocks - I could generate one 2D-DCT matrix that could then be applied to each of these blocks. For each block I randomly sampled and solved the minimization problem ($min\|s\|_1 \quad \text{such that} \quad y = \Theta s$). This was done using CVXPY [29], although other solvers do exist - a list of solvers is available at [30]. For consistency with later reconstructions I set an iteration limit of 200. I then reconstructed my image based on each of these smaller 16x16 blocks. This is a similar to approach JPEG compression, which finds the DCT matrix for an 8x8 block, and then iteratively applies that matrix (and compression) to a larger image [31] [32]. This process is embarrassingly parallel, as each block can be processed independently of one another.

In my research, I came across [28], where the author had used a variation of the L-BFGS algorithm[33], known as OWL-QN, for reconstruction. This is a dedicated method for fitting l_1 regularized models that exploits the sparsity of those models [33]. A Python wrapper for the algorithm can be found at [34]. I used this wrapper for reconstruction.

I then investigated some of the other reconstruction algorithms using l_1 magic [35], a set of Matlab scripts released by some of the pioneers in CS alongside their first papers. I investigated minimizing TV with equality constraints ($\min \mathrm{TV}(s) \quad \text{such that} \quad y = \Theta s$), minimizing TV with quadratic constraints ($\min \mathrm{TV}(s) \quad \text{such that} \quad \|(y - \Theta s)\|_2 \leq \epsilon$) and minimizing TV with the Dantzig selector ($\min \mathrm{TV}(s) \quad \text{such that} \quad \|(\Theta^*(\Theta s - y)\|_\infty \leq \epsilon$) [36]. For each of these. I chose an ϵ of 0.005, or an iteration limit of 200.

6 Results

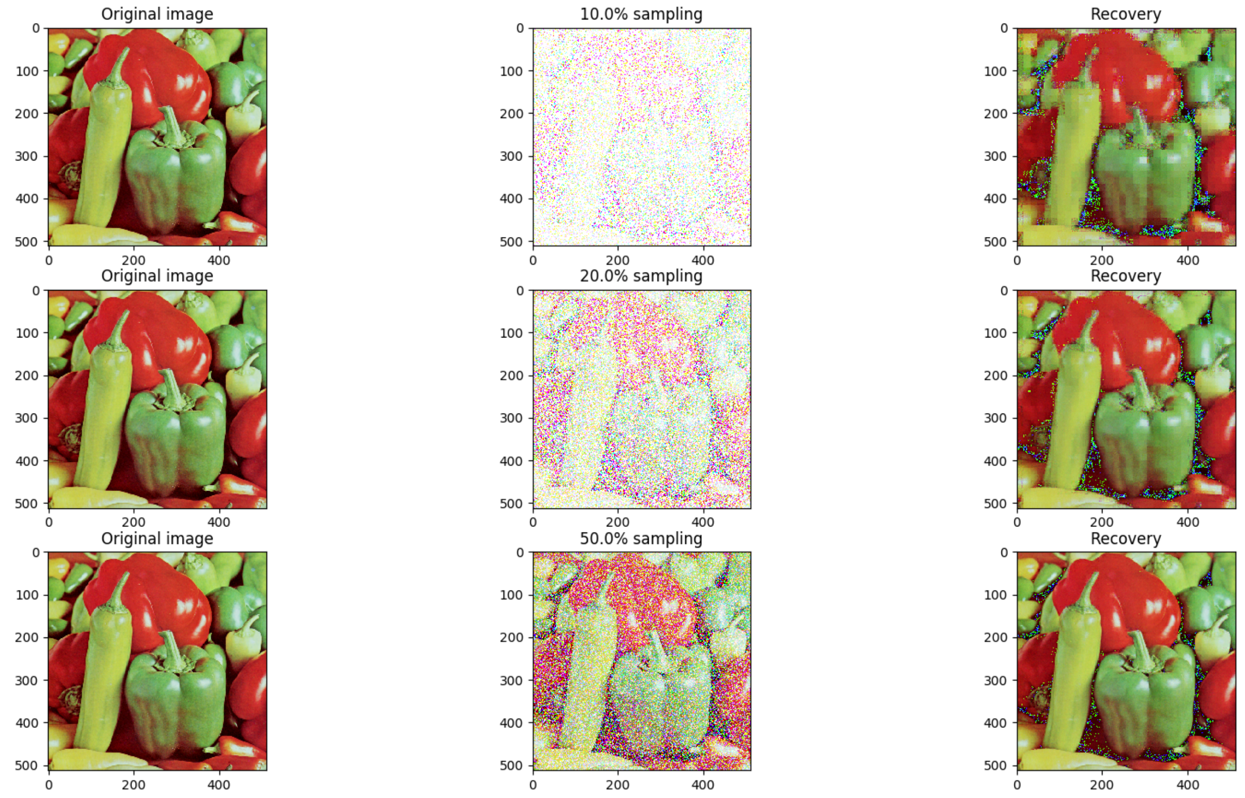

Figure 4: Reconstruction using basis pursuit.

Table 1: Speeds of BP and OWL-QN reconstruction

Technique	**Sample percentage**	**Time taken (s)**
BP	10%	28.57
BP	20%	50.20
BP	50%	184.57
OWL-QN	10%	10.67
OWL-QN	20%	8.51
OWL-QN	50%	5.94

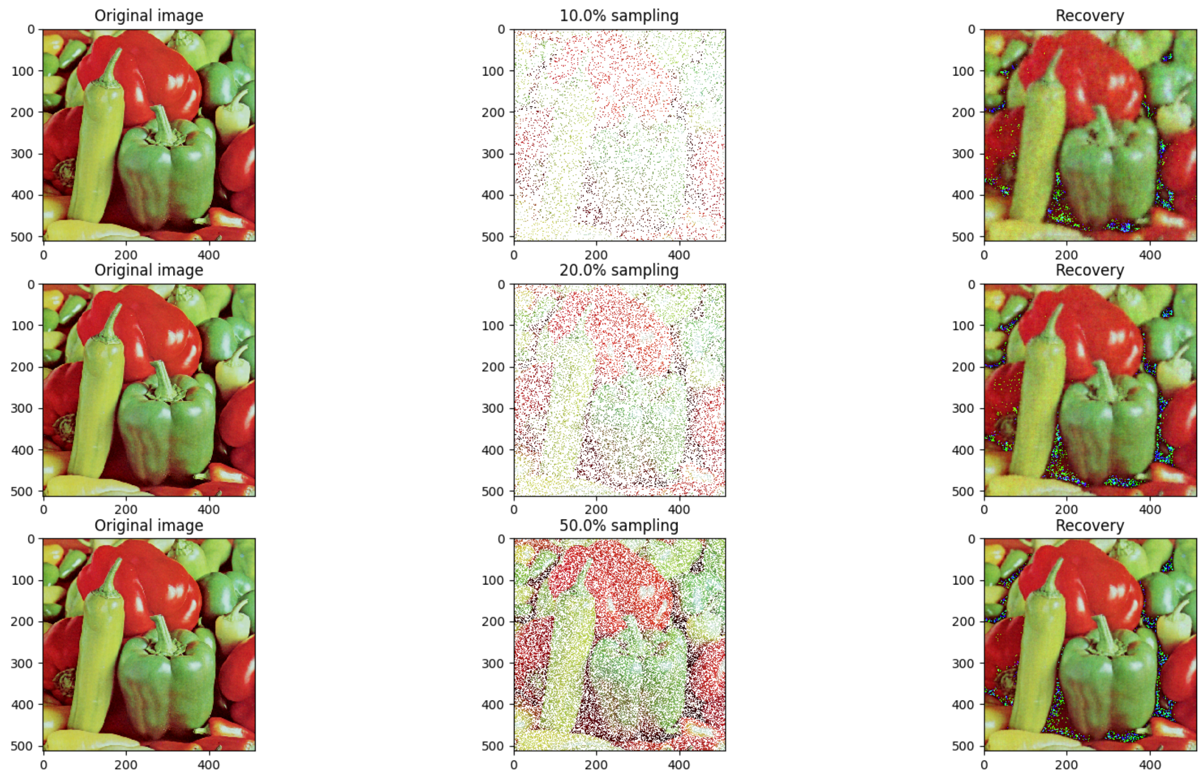

Figure 5: Reconstruction using OWL-QN.

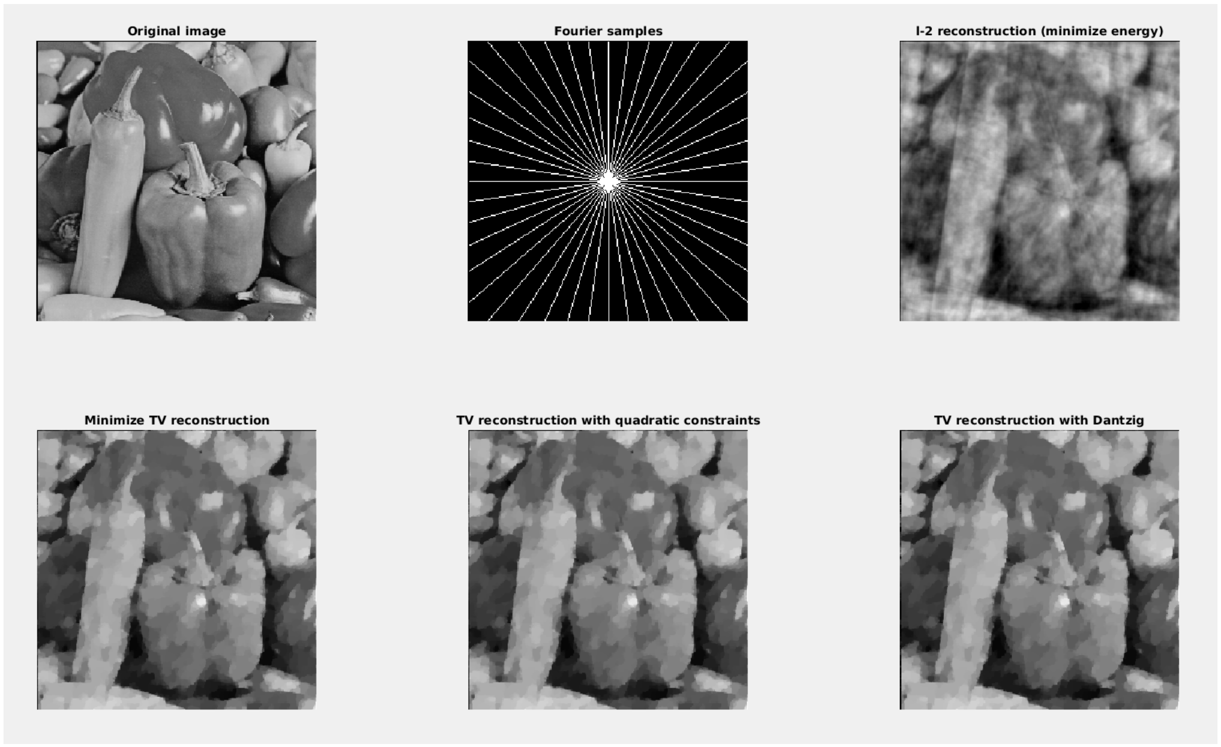
Original image
Fourier samples
l-2 reconstruction (minimize energy)
Minimize TV reconstruction
TV reconstruction with quadratic constraints
TV reconstruction with Dantzig

Figure 6: Different reconstructions using variations on minimizing the TV.

Table 2: Speeds of TV minimization reconstruction

TV technique	**Time taken (s)**
Minimization with equality	46.76
Minimization with quadratic constraints	53.43
Minimization with Dantzig selector	72.55

7 Discussion

7.1 Basis pursuit

Firstly, one notices that even with very low sampling of original images very good recovery is achieved. The broad shapes and colours are recognizable. There are noticeable artifacts, which could be removed using a filter. In addition, the effect of sampling in 16x16 blocks is clear - the recovered images clearly look 'segmented', in the sense that there are obvious boundaries between blocks.

One solution to this is to apply a blurring filter to smooth some of these edges. Another would be to use a different reconstruction technique that is more robust, and does not have to break the image down into blocks.

As expected, the quality of reconstruction improves with more samples.

7.2 OWL-QN

These recoveries are much smoother as compared to figure 4. This is because the image was not processed in block segments. Once again, there are artifacts present. However, the recovery is still very good. This method of reconstruction was faster than basis pursuit.

7.3 TV

Total variation yielded the poorest results. The recovered images are significantly blurred, although still much better than classical reconstruction techniques that minimize the l_2 norm. These reconstruction technique was also slower than other techniques used.

Although the outputs look very similar, they are subtly different (when comparing element-by-element the images have differences that cannot necessarily be seen with the naked eye).

8 Conclusion

I successfully acquired compressive samples of images by using random measurements. I was then able to apply different CS reconstruction algorithms to recover the images from these compressive samples. I investigated Basis Pursuit and OWL-QN using Python, and investigated TV minimization reconstruction techniques using the l_1 magic toolbox available for Matlab. In all cases the original image was successfully recovered, although the degree of recovery ranged based on techniques and the number of random samples acquired.

Given more time I would like to investigate the more robust reconstruction techniques (to deal with noise). I would also like to investigate the outputs of parallelizing some of the algorithms, or implementing some of the inherently parallel algorithms (such as CoSaMP). In addition, I would be interested investigating different optimization solvers. Further, I would like to investigate some of the other reconstruction algorithms to compare them against one another. Lastly, I would like to understand more of the maths behind CS - the original papers are beyond my current abilities, but given more time I would like to grasp them fully.

Appendix - Code

8.1 Basis Pursuit

```python
import imageio as im
import numpy as np
import matplotlib.pyplot as plt
import scipy.fftpack as spfft
import scipy.ndimage as spimg
import scipy as sp
import cvxpy as cvx
from timeit import default_timer as timer

def dct2(x):
    return spfft.dct(spfft.dct(x.T, norm='ortho', axis=0).T, norm='ortho',
            axis=0)

def idct2(x):
    return spfft.idct(spfft.idct(x.T, norm='ortho', axis=0).T, norm='ortho',
            axis=0)

def dctmtx(N):
    return spfft.idct(np.eye(N), norm='ortho', axis=0)

def dctmtx2(N):
    return np.kron(dctmtx(N), dctmtx(N))

def recover_patch(patch, sample_matrix, psi, percentage_samples):
    size_y, size_x  = patch.shape
    N = size_y*size_x
    K = round(N*percentage_samples)
    random_samples = np.random.choice(N, K, replace=False)

    # take samples
    sample_matrix.T.flat[random_samples] = patch.T.flat[random_samples]

    # measurements
    y = patch.T.flat[random_samples]

    # theta = phi*psi, where psi is DCT matrix and phi is measurement matrix
    theta = psi[random_samples, :]

    s = cvx.Variable(N)
    objective = cvx.Minimize(cvx.norm(s, 1))
    constraints = [theta*s == y]
    prob = cvx.Problem(objective, constraints)
    result = prob.solve(solver=cvx.ECOS, max_iters = 200)
    return np.array(s.value).squeeze()

orig_image = im.imread('../Images/pepper.ppm')
dim_y, dim_x, channels = orig_image.shape

patch_length = 16
psi = dctmtx2(patch_length)

# sample_matrix is phi
sample_matrix = 255*np.ones(orig_image.shape, dtype="uint8")
recovered_image = 255*np.ones(orig_image.shape, dtype="uint8")

percentage_samples = (0.1, 0.2, 0.5)

plt_count = 1
for sample in percentage_samples:
    start = timer()
    for colour in range(channels):
        #sample_matrix[:,:,colour].T.flat[random_indeces] = orig_image[:,:,colour].T.flat[
    random_indeces]
        for i in range(0, dim_y, patch_length):
            for j in range(0, dim_y, patch_length):
                result = recover_patch(orig_image[i:i+patch_length, j:j+patch_length, colour
    ],
                                        sample_matrix[i:i+patch_length, j:j+patch_length,
    colour],
                                            psi, sample
                                            )
                 result = result.reshape(patch_length, patch_length).T
                 result = idct2(result)
                 recovered_image[i:i+patch_length, j:j+patch_length, colour] = result

    end = timer()
    print(end-start)

    plt.subplot(3,3,plt_count)
    plt.imshow(orig_image)
    plt.title("Original image")
    plt_count += 1
    plt.subplot(3,3,plt_count)
    plt.imshow(sample_matrix)
    plt.title("{}% sampling".format(sample*100))
    plt_count += 1
    plt.subplot(3,3,plt_count)
    plt.imshow(recovered_image)
    plt.title("Recovery")
    plt_count += 1

plt.show()
```

```
% Written by: Justin Romberg, Caltech
% Email: jrom@acm.caltech.edu
% Created: October 2005
%
% Modified by Josh Stein


path(path, './l1magic/Optimization');
path(path, './l1magic/Measurements');
path(path, './l1magic/Data');

load peppers_gray;

n = 256;
N = n*n;
X = im;
x = X(:);

% number of radial lines in the Fourier domain
L = 22;

% Fourier samples we are given
[M,Mh,mh,mhi] = LineMask(L,n);
OMEGA = mhi;
A = @(z) A_fhp(z, OMEGA);
At = @(z) At_fhp(z, OMEGA, n);

% measurements
y = A(x);

% min l2 reconstruction (backprojection)
xbp = At(y);
Xbp = reshape(xbp,n,n);

epsilon = 5e-3;

% Dantzig reconstruction
tic
xp_dantzig =  tvdantzig_logbarrier(xbp, A, At, y, epsilon, 1e-3, 5, 1e-8, 200);
Xtv_dant = reshape(xp, n, n);
toc

% Quadratic reconstruction
```

```
tic
xp_qc =  tvqc_logbarrier(xbp, A, At, y, epsilon, 1e-3, 5, 1e-8, 200);
Xtv_qc = reshape(xp, n, n);
toc

% Equality reconstruction
tic
tvI = sum(sum(sqrt([diff(X,1,2) zeros(n,1)].^2 + [diff(X,1,1); zeros(1,n)].^2 )));
xp = tveq_logbarrier(xbp, A, At, y, 1e-1, 5, 1e-8, 200);
Xtv = reshape(xp, n, n);
toc


subplot(2,3,1), imshow(X, [0 255]);
title("Original image");
subplot(2,3,2), imshow(fftshift(M));
title("Fourier samples");
subplot(2,3,3), imshow(Xbp, [0 255]);
title("l-2 reconstruction (minimize energy)");

subplot(2,3,4), imshow(Xtv, [0 255]);
title("Minimize TV reconstruction");

subplot(2,3,5), imshow(Xtv_qc, [0 255]);
title("TV reconstruction with quadratic constraints");

subplot(2,3,6), imshow(Xtv_dant, [0 255]);
title("TV reconstruction with Dantzig");
```

References

[1] M. Rani, S. B. Dhok, and R. B. Deshmukh, "A systematic review of compressive sensing: Concepts, implementations and applications," *IEEE Access*, vol. 6, pp. 4875–4894, 2018.

[2] U. of Delaware, "Richard Baraniuk, "Compressive Sensing," ECE Lecturer Series," 2012. [Online]. Available: https://www.youtube.com/watch?v=RvMgVv-xZhQ

[3] Wikipedia contributors, "Nyquist–Shannon sampling theorem," 2019. [Online]. Available: https://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem

[4] E. J. Candès and M. B. Wakin, "An introduction to compressive sampling [a sensing/sampling paradigm that goes against the common knowledge in data acquisition]," *IEEE signal processing magazine*, vol. 25, no. 2, pp. 21–30, 2008.

[5] E. Candes, J. Romberg, and T. Tao, "Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information," *arXiv preprint math/0409186*, 2004.

[6] E. Candes and T. Tao, "Near optimal signal recovery from random projections: Universal encoding strategies?" *arXiv preprint math/0410542*, 2004.

[7] D. L. Donoho *et al.*, "Compressed sensing," *IEEE Transactions on information theory*, vol. 52, no. 4, pp. 1289–1306, 2006.

[8] Rice University, "Compressive Ssensing Resources," 2017. [Online]. Available: http://dsp.rice.edu/cs/#talks

[9] R. G. Baraniuk, "Compressive sensing," *IEEE signal processing magazine*, vol. 24, no. 4, 2007.

[10] J. Romberg and M. Wakin, "Compressed Sensing: A Tutorial," 2007. [Online]. Available: http://web.yonsei.ac.kr/nipi/lectureNote/Compressed%20Sensing%20by%20Romberg%20and%20Wakin.pdf

[11] J. K. Romberg, "Imaging via compressive sampling," *IEEE Signal Processing Magazine*, vol. 25, pp. 14–20, 2008.

[12] Y. C. Eldar and G. Kutyniok, *Compressed sensing: theory and applications.* Cambridge University Press, 2012.

[13] J. A. Tropp and S. J. Wright, "Computational methods for sparse solution of linear inverse problems," *Proceedings of the IEEE*, vol. 98, no. 6, pp. 948–958, 2010.

[14] S. S. Chen, D. L. Donoho, and M. A. Saunders, "Atomic decomposition by basis pursuit," *SIAM review*, vol. 43, no. 1, pp. 129–159, 2001.

[15] E. Candes, T. Tao *et al.*, "The dantzig selector: Statistical estimation when p is much larger than n," *The annals of Statistics*, vol. 35, no. 6, pp. 2313–2351, 2007.

[16] L. I. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D: nonlinear phenomena*, vol. 60, no. 1-4, pp. 259–268, 1992.

[17] S. G. Mallat and Z. Zhang, "Matching pursuits with time-frequency dictionaries," *IEEE Transactions on signal processing*, vol. 41, no. 12, pp. 3397–3415, 1993.

[18] Y. C. Pati, R. Rezaiifar, and P. S. Krishnaprasad, "Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition," in *Proceedings of 27th Asilomar conference on signals, systems and computers.* IEEE, 1993, pp. 40–44.

[19] T. Blumensath and M. E. Davies, "Gradient pursuits," *IEEE Transactions on Signal Processing*, vol. 56, no. 6, pp. 2370–2382, 2008.

[20] D. Needell and J. A. Tropp, "Cosamp: Iterative signal recovery from incomplete and inaccurate samples," *Applied and computational harmonic analysis*, vol. 26, no. 3, pp. 301–321, 2009.

[21] Wikipedia contributors, "Matching Pursuit," 2019. [Online]. Available: https://en.wikipedia.org/wiki/Matching_pursuit

[22] G. Pope, "Compressive sensing: A summary of reconstruction algorithms," Master's thesis, ETH, Swiss Federal Institute of Technology Zurich, Department of Computer . . . , 2009.

[23] T. Blumensath and M. E. Davies, "Iterative hard thresholding for compressed sensing," *Applied and computational harmonic analysis*, vol. 27, no. 3, pp. 265–274, 2009.

[24] I. Daubechies, M. Defrise, and C. De Mol, "An iterative thresholding algorithm for linear inverse problems with a sparsity constraint," *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, vol. 57, no. 11, pp. 1413–1457, 2004.

[25] S. R. Becker, "Practical compressed sensing: modern data acquisition and signal processing," Ph.D. dissertation, California Institute of Technology, 2011.

[26] T. Oliphant, "NumPy: A guide to NumPy," USA: Trelgol Publishing, 2006–. [Online]. Available: http://www.numpy.org/

[27] H. Levkin, "Test images," 2019. [Online]. Available: https://www.hlevkin.com/06testimages.htm

[28] R. Taylor, "Compressed sensing in python," 2016. [Online]. Available: http://www.pyrunner.com/weblog/2016/05/26/compressed-sensing-python/

[29] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.

[30] I. Carron, "Compressive Sensing: the Big Picture," 2013. [Online]. Available: https://sites.google.com/site/igorcarron2/cs#reconstruction

[31] MathWorks, "dctmtx," 2019. [Online]. Available: https://www.mathworks.com/help/images/ref/dctmtx.html

[32] Wikipedia contributors, “JPEG,” 2019. [Online]. Available: https://en.wikipedia.org/wiki/JPEG#Encoding

[33] ——, “Limited-memory BFGS — Wikipedia, the free encyclopedia,” 2019. [Online]. Available: https://en.wikipedia.org/wiki/Limited-memory_BFGS

[34] R. Taylor, “PyLBFGS,” 2018. [Online]. Available: https://bitbucket.org/rtaylor/pylbfgs/overview

[35] E. Candes and J. Romberg, “$l1$-magic,” 2005. [Online]. Available: http://statweb.stanford.edu/~candes/l1magic/

[36] ——, “l1-magic: Recovery of sparse signals via convex programming,” *URL: www. acm. caltech. edu/l1magic/downloads/l1magic. pdf*, vol. 4, p. 14, 2005.

[37] N. University, “Compressed Sensing by Terence Tao.” [Online]. Available: https://www.youtube.com/playlist?list=PLC94A02A1218B24DF

[38] ICM, “Gauss Prize Lecture: Compressed sensing — from blackboard to bedside — David Donoho — ICM2018.” [Online]. Available: https://www.youtube.com/watch?v=mr-oT5gMboM

[39] A. Xia, “MIT 6.854 Spring 2016 Lecture 22: Compressed Sensing.” [Online]. Available: https://www.youtube.com/watch?v=G3WLsZAoTuo

[40] M. Cleve Moler, “”Magic” reconstruction: Compressed sensing,” 2010. [Online]. Available: https://www.mathworks.com/company/newsletters/articles/magic-reconstruction-compressed-sensing.html

[41] Miliarde, “Compressed Sensing Intro & Tutorial w/ Matlab,” 2016. [Online]. Available: https://www.codeproject.com/Articles/852910/Compressed-Sensing-Intro-Tutorial-w-Matlab

[42] S. Brunton, “A Compressed Overview of Sparsity,” 2016. [Online]. Available: https://www.youtube.com/watch?v=aHCyHbRIz44

[43] Wikipedia contributors, “Broyden–fletcher–goldfarb–shanno algorithm — Wikipedia, the free encyclopedia,” 2019. [Online]. Available: https://en.wikipedia.org/wiki/Broyden%E2%80%93Fletcher%E2%80%93Goldfarb%E2%80%93Shanno_algorithm