Compressed Sensing in MRI

Kashif Hussain
ID: 8444507
Supervisor: Oliver Dorn
Course Code: MATH4000

May 10, 2016

Contents

Preface

My project is devoted to understanding the role and applications of Compressed Sensing (CS) in Magnetic Resonance Imaging (MRI). The focus is on demonstrating the application of compressed sensing in MRI through the use of a variety of algorithms. Numerical experiments will be performed with the following algorithms, namely: Basis Pursuit (BP), Orthogonal Matching Pursuit (OMP) and Compressive Sampling Matching Pursuit (CoSaMP).

Prior knowledge may be useful in the following areas:

- **Linear Algebra/Matrix Algebra**: As I will be using Matlab for my numerical experiments, knowledge of linear algebra and how matrices are defined and treated would be useful in understanding the code in the Appendix. A glance at MATLAB's refresher page would help you understand most of what was written down.

- **Signal Processing Basics**: Knowledge of Fourier Transforms (FT), Discrete Cosine Transforms (DCT) and Discrete Wavelet Transforms (DWT) would be useful. Having some background in signal processing and understanding what the Nyquist rate is (introduced almost immediately) will help in understanding the topic of compressed sensing quickly. Some of the signals we will use will be sparse in the time domain, however, MRI images are not sparse and need to be transformed for use in compressive sensing techniques. Transformations to sparse domains are at the heart of compressed sensing.

- **Compression Techniques**: This ties in heavily with signal processing. Whilst knowledge of such techniques is not necessary, they will help in the understanding of the techniques employed in my project. Familiarity with JPEG and its use of the Discrete Cosine Transform which will help point out similarities between my use and that in JPEG.

Chapter Breakdown

- **Chapter 1**: Chapter one introduces the Nyquist-Shannon sampling theorem and how it relates to compressed sensing. In addition, I will introduce the concept of compressed sensing with a brief example as well as an introduction as to how MRI scans work. The chapter ends in detailing the role of compressed sensing in MRI.

- **Chapter 2**: Chapter two is very brief and outlines an important property called the Restricted Isometric Property (RIP) which many algorithms I explore rely on for certain theorems.

- **Chapter 3**: Chapter 3 introduces some of the basic algorithmic approaches to compressed sensing. Some of the algorithms will not be explored in detail and those that are, are given a brief introduction. Model-based compressed sensing, whilst covering a large portion of this chapter, is not experimented with but was discussed in detail due to its expansiveness.

- **Chapter 4**: Chapter 4 covers 3 main algorithms: Basis Pursuit, Orthogonal Matching Pursuit and Compressive Sampling Matching Pursuit. In addition, numerical experiments are carried out on 1-D signals, noise-perturbed signals and finally 2-D MRI images. A breakdown is given on the performance of each of these algorithms with respect to PSNR, running time and the the error produced by the difference in the reconstructed image and original image.

- **Chapter 5**: Chapter 5, the final chapter, marks the end of my project and summarizes the essential information gathered. Using the criterion I developed in chapter 3, I state which algorithms satisfy each criterion.

- **Appendix**: The appendix located at the back of this report details the code use to present the results located within. Some of the code may need to be modified to output certain results. All of the code was written in MATLAB.

Table 1: Table of Notation

		Fields
$\mathbb{R}$	:=	Field of real numbers.
$\mathbb{C}$	:=	Field of complex numbers.
$\mathbb{F}$	:=	Any field.
		Vector Notation
$\mathbf{x} \in \mathbb{F}^{N \times 1}$	:=	$N \times 1$ column vector with N entries from a field $\mathbb{F}$.
$x \in \mathbb{F}$	:=	A scalar quantity in a given field $\mathbb{F}$.
$\mathbf{x}_i$	:=	The ith entry in the vector $\mathbf{x}$.
$\mathbf{x}^{(i)}$	:=	The ith iteration of $\mathbf{x}$ in an algorithm.
$supp(\mathbf{x})$	:=	The position of the non-zero values of $\mathbf{x} = \left\{ \mathrm{i} : \mathbf{x}_i \neq 0 \right\}$.
		Matrix Notation
$\mathbf{A} \in \mathbb{F}^{M \times N}$	:=	$M \times N$ matrix with entries from from a field $\mathbb{F}$.
$\mathbf{A}_{i,j}$	:=	The (i,j)th entry of $\mathbf{A}$.
$\mathbf{A}^T$	:=	The transpose of $\mathbf{A}$.
$\mathbf{A}^\dagger$	:=	The pseudo-inverse of $\mathbf{A}$.
		Norm Notation
$\lvert x \rvert$	:=	The absolute value of the scalar x.
$\lvert\lvert \mathbf{x} \rvert\rvert_1$	:=	The ℓ_1-norm: $\sum_{i=1}^{n} \lvert x_i \rvert$.
$\lvert\lvert \mathbf{x} \rvert\rvert_2$	:=	The ℓ_2-norm: $\sqrt{\sum_{i=1}^{n} \lvert x_i \rvert^2}$.
$\lvert\lvert \mathbf{x} \rvert\rvert_p$	:=	The ℓ_p-norm: $\left(\sum_{i=1}^{n} \lvert x_i \rvert^p \right)^{1/p}$.
		Other Notation
$\mathbb{P}(A)$	:=	Probability of event A

Chapter 1

Introduction to Compressed Sensing

1.1 The Nyquist-Shannon Sampling Theorem

Before introducing what compressed sensing actually is, we must first understand the importance of the Nyquist rate. The sampling theorem relies on a signal whose Fourier transform is zero outside a certain frequency range and this is known as bandlimiting. The Nyquist-Shannon sampling theorem states that perfect reconstruction of a bandlimited signal requires the sampling rate to be, at minimunum, twice the maximumum frequency found in the bandlimited signal.[27]

For example, given $x(t)$ is a function of continuous time with a Fourier transform $X(f)$, we find the following equation:

$$X(f) = \int_{\infty}^{\infty} x(t) e^{i2\pi ft} dt. \tag{1.1}$$

In mathematical terms, for the bandlimited signal $x(t)$, we have that the Fourier transform of such a signal obeys the following:

$$\exists B \in \mathbb{R} \quad \text{such that} \quad X(f) = 0 \quad \forall |f| > B. \tag{1.2}$$

A sufficient sample-rate is 2B samples/second, or anything larger.

Equivalently, for a given sample rate f_s, perfect reconstruction of the signal is guaranteed possible if $f_s > 2B$. If the bandlimit is too large or is non-existant, aliasing will be introduced after reconstruction and similarly if f_s is less than $2B$. The two thresholds $2B$ and f_s are known as the Nyquist rate and the Nyquist frequency respectively. Figure 1.1 shows the Fourier transform of the above signal $x(t)$.

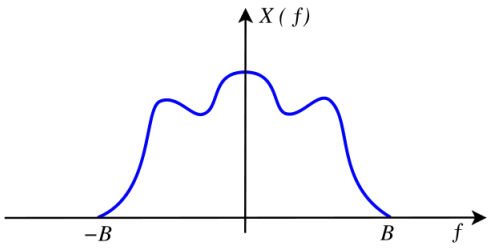


Figure 1.1: The Fourier transform of the bandlimited function $x(t)$ (amplitude against frequency)[27]

In the medical field and specifically in the MRI field, Nyquist's sampling rate is still too large. The goal is to reduce a patient's exposure to electromagnetic radiation and in order to carry out this goal, we need to take as few samples as possible whilst still reconstructing the scanned image to a high degree of accuracy. This leads to the introduction of compressed sensing which exploits the sparsity of an image to reconstruct the image at a rate far below the Nyquist rate.

1.2 What is Compressed Sensing

Throughout the science and technology industry, there are tasks which require you to retrieve data of some importance from a measurable piece of information. We can use compressed sensing to perform such a task. Compressed sensing is a signal processing technique which is used to efficiently acquire and reconstruct signals by finding solutions to underdetermined linear systems (i.e. fewer equations than unknowns)[26]. The equations in (1.3) show an underdetermined system of equations with no solutions.

$$\begin{aligned} a + b + c &= 1, \\ a + b + c &= 2. \end{aligned} \tag{1.3}$$

Compressive sensing exploits the sparsity of a signal/image to recover the signal/image at a rate far below Nyquist rate. As current technology depends heavily on the acquisition and processing of many signals/data (i.e. MRI/CT scans), the process of retrieving compressible signals and reconstructing them can be greatly improved by acquiring them in a compressed form. From this, we can produce an excellent approximation of the actual data through the use of various methods you will see inside this report. Compressible data approximates the data being compressed accurately, but has far fewer non-zero entries.

When the signal/data acquisition process is linear, the problem is reduced to solving a linear system of equations. As a result, the problem of compressed sensing is to find a k-sparse solution to an underdetermined system of linear equations. We refer to a vector as k-sparse if, at most, k of its co-ordinates are non-zero. The matrix in figure 1.2 demonstrates a k-sparse matrix when $k = 6$

$$\mathbf{X}_{4,5} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 3 & 3 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 5 \end{pmatrix}$$

Figure 1.2: 6-Sparse 4 by 5 Matrix

As an example, the process of recovering an image from sampled data is as follows:

Given M measurements of the measured data, we need to recover the vector (the signal) $\mathbf{x}$ from the observed data $\mathbf{b}$ whilst letting the matrix $\mathbf{A}$ represent the process of reading the signal. So given $\mathbf{A} \in \mathbb{R}^{M\times N}$, $\mathbf{x} \in \mathbb{R}^{N\times 1}$, $\mathbf{b} \in \mathbb{R}^{M\times 1}$ as described as well as $M < N$, such a system is can be written in the form:

$$\mathbf{Ax} = \mathbf{b}. \tag{1.4}$$

Figure 1.3 is an illustration of all the ingredients in the compressive sensing process:

$$\mathbf{A}_{m,n} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}, \mathbf{x}_{n,1} = \begin{pmatrix} x_{1,1} \\ x_{2,1} \\ \vdots \\ x_{n,1} \end{pmatrix}, \mathbf{b}_{m,1} = \begin{pmatrix} b_{1,1} \\ b_{2,1} \\ \vdots \\ b_{m,1} \end{pmatrix}.$$

Figure 1.3: The vectors and matrices for $\mathbf{Ax} = \mathbf{b}$

Constructing such a system raises a few questions:

- Can we reconstruct signal $\mathbf{x} \in \mathbb{R}^{N\times 1}$ given $\mathbf{Ax} = \mathbf{b}$?
- What matrices $\mathbf{A} \in \mathbb{R}^{M\times N}$ are appropriate?

Whilst the two questions are closely related, there are in-fact numerous recovery algorithms (discussed later) dedicated to the task of reconstructing the signal $\mathbf{x}$. However, the design of the matrix $\mathbf{A}$, the measurement matrix, is equally important and through various breakthroughs in the field, results were found to be the best through the use of *random matrices*.

Bernoulli matrices whose entries are +1 or -1 with equal probability and Gaussian matrices formed from a normal distribution are some examples of random matrices. The key point being, that a Gaussian or Bernoulli $M \times N$ matrix $\mathbf{A}$, with randomly generated coefficients, can reconstruct any K-sparse vectors $\mathbf{x}$ from $\mathbf{Ax} = \mathbf{b}$ through recovery algorithms.

To summarize, we have the following:

- $\mathbf{x}$ - $N \times 1$ vector which represents our original signal.
- $\mathbf{b}$ - $M \times 1$ vector which represents our output values.
- $\mathbf{A}$ - $M \times N$ measurement matrix (i.e. getting from input to output) and $M < N$

1.3 Simple Compressive Sensing Example

To illustrate what compressive sensing is about, I will demonstrate by showing a simple example. Let $\mathbf{x}$ be a 4×9 vector of random values. We would like to keep only 4 of these values namely the 1st, 4th, 7th and 9th.

$$\mathbf{A}_{4,9} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \mathbf{x}_{9,1} = \begin{pmatrix} 0.61 \\ 0.01 \\ 0.04 \\ 0.12 \\ 0.38 \\ 0.83 \\ 0.42 \\ 0.94 \\ 0.76 \end{pmatrix}, \mathbf{b}_{4,1} = \begin{pmatrix} 0.61 \\ 0.12 \\ 0.42 \\ 0.76 \end{pmatrix}.$$

Figure 1.4: A simple example of $\mathbf{Ax} = \mathbf{b}$

In figure 1.4, we keep only 4 samples of our original data and discard the remaining 5. However, in the context of medical imaging, our original data x is usually sparse under specific transforms and as such our measurement matrix A is inappropriate. This is because we do not know the location of the most important and useful values of the original data beforehand and as such, we may end up discarding them with our present measurement matrix. So now, suppose $\mathbf{x}$ is sparse as below.

$$\mathbf{A}_{4,9} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \mathbf{x}_{9,1} \begin{pmatrix} 0.61 \\ 0.01 \\ 0.04 \\ 0 \\ 0 \\ 0 \\ 0.42 \\ 0 \\ 0 \end{pmatrix}, \mathbf{b}_{4,1} = \begin{pmatrix} 0.61 \\ 0 \\ 0.42 \\ 0 \end{pmatrix}.$$

Figure 1.5: Showing the effect of an inappropriate choice for $\mathbf{A}$ on our output values $\mathbf{b}$ - we lose essential information

We require a new approach towards creating our measurement matrix $\mathbf{A}$. As discussed before, we can use *random matrices* for an effective measurement matrix and then use recovery algorithms on our output values $\mathbf{b}$ to get a close approximation for our original data $\mathbf{x}$. As an example, we can use a Gaussian matrix as in figure 1.6. Our output is different from the original $\mathbf{x}$, but the original values are still encoded into the entries of $\mathbf{b}$.

$$\mathbf{A}_{4,9} = \begin{pmatrix} 0.09 & 0.73 & 0.61 & -0.25 & 1.26 & 0.55 & -0.26 & 0.22 & -1.22 \\ -1.50 & -0.60 & -0.01 & 0.80 & 0.16 & -0.02 & 0.37 & -0.91 & -0.03 \\ 0.21 & -0.43 & 0.10 & 1.15 & 1.78 & -1.99 & -0.86 & -2.59 & 1.43 \\ -0.99 & -0.37 & -2.24 & 0.26 & 0.73 & -1.07 & -0.94 & 0.18 & -0.41 \end{pmatrix},$$

$$\mathbf{x}_{9,1} \begin{pmatrix} 0.61 \\ 0.01 \\ 0.04 \\ 0 \\ 0 \\ 0 \\ 0.42 \\ 0 \\ 0 \end{pmatrix}, \mathbf{b}_{4,1} = \begin{pmatrix} -0.02 \\ -0.77 \\ -0.23 \\ -1.09 \end{pmatrix}.$$

Figure 1.6: Obtaining 4 output values using Gaussian random variables for our measurement matrix $\mathbf{A}$

Figure 1.6 shows a system with some knowns, unknowns and, in particular, one assumption. With this information at our disposable, we need to recover our original vector $\mathbf{x}$. It is at this point that we make use of compressed sensing recovery algorithms to retain a close approximation to the original vector $\mathbf{x}$.

A summary of the information attained from figure 1.6:

Known:

- The 4×9 measurement matrix $\mathbf{A}$ which was randomly generated with Gaussian random variables.
- The 4×1 output vector $\mathbf{b}$.

Unknown:

- The 9×1 original vector $\mathbf{x}$.

The Assumption:

- The 9×1 original vector $\mathbf{x}$ is sparse.

To summarize:

In the context of compressively sensing MRI images, we need the measurement matrix to be able to encode the values present in the sparse domain of the image into the output vector so that no value is discarded. Following the encoding of the transform data, we make use of a recovery algorithm to approximate the values of the original sparse domain of the image from our encoded data.

The following section details what Magnetic Resonance Imaging (MRI) scans are and how the a scan is achieved as well as the importance and applications of compressed sensing in MRI

1.4 How MRI Scans work

Most of the human body is comprised of water (about 70%) and each water molecule is comprised of hydrogen and oxygen atoms. Hydrogen atoms have a strong magnetic moment, which means that in a magnetic field, the protons in the centre of these hydrogen atoms which are sensitive to magnetic fields and have a positive charge, line up in the direction of the field.

Laying down under an MRI scanner magnetizes the protons so that half point to the patient's feet and the other half point to the patient's head. As a result, most of the protons cancel each other out meaning that for each proton facing the patient's head, there is a proton facing the patient's feet so that they differ by approximately 180 degrees. However, as there millions of protons inside the human body, some are not canceled out. Following the alignment of the protons, short bursts of radio waves, specific to hydrogen atoms, are sent to designated areas of the body. These radio frequency pulses (RF) distort the alignment of those unmatched protons which absorb the energy and these protons begin to spin at a particular frequency and direction. When the burst of radio frequency pulses stop, the protons realign so that they face their original direction inside the magnetic field whilst releasing the energy absorbed from the RF pulses by sending out their own radio signals which are picked up by receivers.[20]

From this, the system now knows the location of the protons inside your body. Like the amalgamation of sound produced by the notes of a piano all point to certain signals, MRI scanners detects tissues like a musician would detect the intensity or type of note. This is because the various types of tissues in your body have protons which align at different speeds and so they all send their own distinct and identifiable signals.

Whilst, on its own, a single proton may not provide useful information, using these millions of signals from the protons in the body, you can reconstruct a a high-resolution image of the internal body like pixels on a computer screen. Figure 1.7 provides a visual illustration of how MRI scans work.

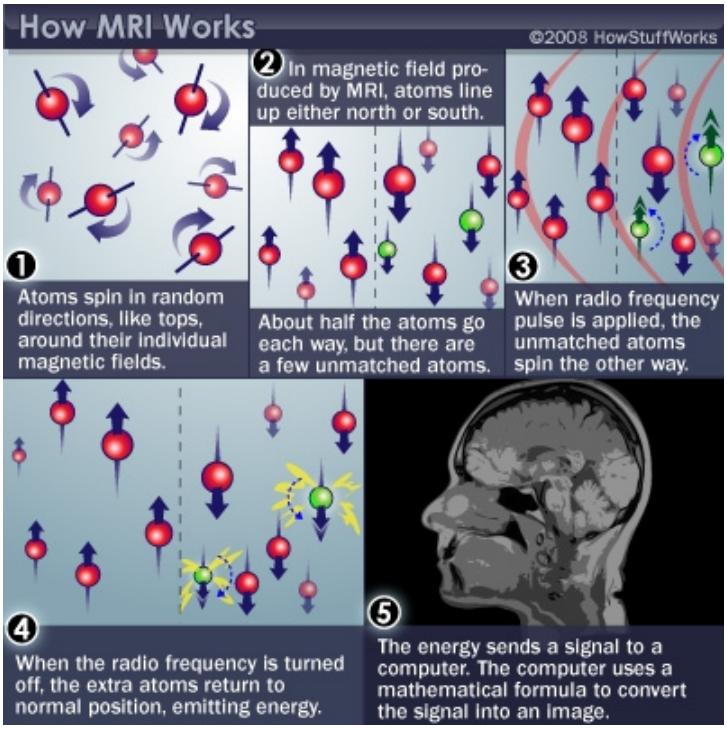


Figure 1.7: How an image is formed from an MRI scan[14]

1.5 Compressed Sensing in MRI

Image compression is an area of research which has been thoroughly studied. Take for example well-known image compression tools such as JPEG which makes use of the Discrete Cosine Transform (DCT). This transform is useful in its ability to transform the image content into a vector of sparse coefficients. The widespread success and use of image compression algorithms such as JPEG makes us question whether or not we need to acquire all the data in the first place as the data is clearly compressible. Why not just retrieve the compressed information directly? As a result, in order to reduce patient's exposure to electromagnetic radiation and reduce scan times, compressive sensing is heavily studied in the medical imaging field.

Magnetic Resonance Imaging (MRI) is a common technology in medical imaging. It is used for various tasks such as brain imaging, heart imaging, examining blood vessels and more. The time taken to produce high-resolution images can range between several minutes up to hours depending on the task. In emergency situations, the usefulness of compressive sensing is highlighted by its ability to speed up the imaging process. As a result, speeding up the scanning time has always been of interest to those researching medical imaging. In recent years, compressed sensing has been touted to be able to achieve accurate reconstruction of signals which are sparse or can be compressed from undersampled measurements (such as MRI/CT scans). As a result, through the use of compressive sensing in MRI, we can cut-down scanning time of MRI scans and produce accurate image reconstruction with the only assumptions about the MRI image used is that it is compressible or sparse in a transform domain.

High quality reference images with similar structure to that of the target image can be acquired easily in various practical MRI applications. Using such a reference image to get prior information can further decrease the number of measurements we need to take. The problems which needs to be overcome by compressive sensing in MRI is that when the K-space is undersampled, the Nyquist criterion is violated and reconstructions show artefacts. The 2D or 3D Fourier transform of an MRI image is known as the K-space and it is utilized as, in general, MRI images are not sparse and they require to be transformed into a sparse domain to apply compressive sensing techniques. Figure 1.8 shows a three-level wavelet decomposition.

An image is represented by a two-dimension array of coefficients where each coefficient corresponds to the colour or brightness of that point. Using

the fact that most natural images have smooth colour variations with details being sharper at edges, we can treat the smooth variations of colour as low frequency variations and the sharp as high frequency.

Both frequencies play an important role and can be described as follows:

- **Low Frequency**: This makes up the base of the image.
- **High Frequency**: This frequency improves upon the base provided by the low frequency and refines the image resulting in a more detailed image.

We can separate the low frequencies of an image from the high by using a wavelet transform and the one I will be using in my experiments on MRI images will be the Direct Wavelet Transform (DWT). We can order the bands in the image by importance with LL_3 being the most important and the order decreases as you move to a higher decomposition level. From these coefficients, we are able to inverse the transform and retain our original image.

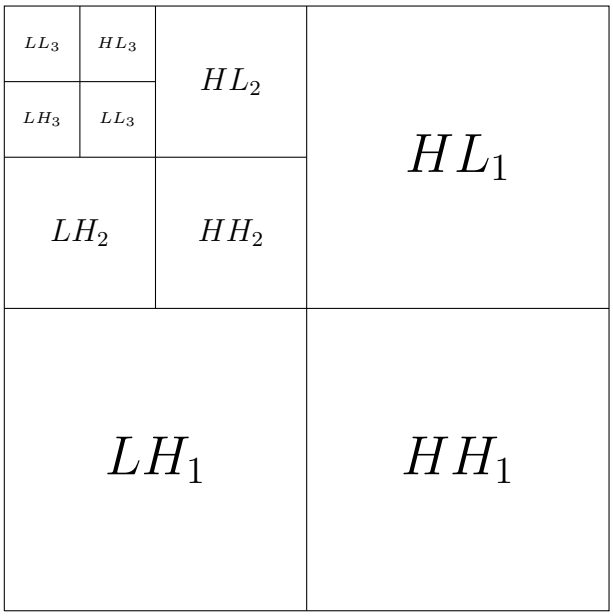


Figure 1.8: A 3 level decomposition of a "Discrete Wavelet Transform" (DWT)

As mentioned earlier, undersampling the K-space will results in artefacts. There are many ways to mitigate artefacts as a result of undersampling:

1 Use methods in which artefacts are less visually apparent to the viewer.

2 Exploit the K-space redundancy by using parallel imaging or other techniques

3 Exploit the spatial/temporal redundancy or both[16].

To get an idea of compressive sensing in MRI, we can represent its undersampling process mathematically. We can form the sampling matrix and its Fourier transform as follows:

- $\mathbf{S} \in \mathbb{C}^{M \times N}$ represent the sampling matrix with $m < n$,
- $\mathbf{F} \in \mathbb{C}^{N \times N}$ is the Fourier transform.

Further, as each image has some noise associated with it, let:

- $\boldsymbol{\eta} \in \mathbb{C}^{M \times 1}$ represent the noise,
- $\mathbf{b} \in \mathbb{C}^{M \times 1}$ being the undersampled k-sparse data.

The the original signal is represented as:

- $\mathbf{x} \in \mathbb{C}^{N \times 1}$ the vector produced via MRI.

As a result, we get the following equation:

$$\boldsymbol{SF}\mathbf{x} + \boldsymbol{\eta} = \mathbf{b}. \tag{1.5}$$

The purpose of image reconstruction then becomes our ability to recover $\mathbf{x} \in \mathbb{C}^{N \times 1}$ produced by Magnetic Resonance Imaging (MRI) from our undersampled data $\mathbf{b}$[29]. Given an idea of how MRI scans work, we must introduce an important property for which many compressed sensing algorithms rely on.

Chapter 2

Restricted Isometric Property

In order to transform an $N \times 1$, K-sparse signal vector $\mathbf{x}$ to our $M \times 1$ output sample vector $\mathbf{b}$ we must use some $M \times N$ measurement matrix $\mathbf{A}$. This measurement matrix must be able to map two different signals $\mathbf{x}_1$ and $\mathbf{x}_2$ to two different output signals $\mathbf{b}_1 = \mathbf{A}\mathbf{x}_1$ and $\mathbf{b}_2 = \mathbf{A}\mathbf{x}_2$.

In order for this to occur, a certain condition must be satisfied known as the Restricted Isometric Property (RIP) as proposed by Candes and Tao[7]:

Definition RIP
For each integer K = 1, 2, ..., the isometry constant δ_K of the measurement matrix $\boldsymbol{A}$ is the smallest number such that:

$$(1 - \delta_K)||\boldsymbol{x}||_2^2 \leq ||\boldsymbol{A}\boldsymbol{x}||_2^2 \leq (1 + \delta_K)||\boldsymbol{x}||_2^2 \tag{2.1}$$

holds for all K-sparse vectors $\boldsymbol{x}$.

In addition if $\delta_K << 1$ then our measurement matrix $\mathbf{A}$ has a large probability of stably reconstructing the $\frac{K}{2}$-sparse signal. Whilst $\delta_K \geq 1$ is not explicitly forbidden, the relevant use involves $\delta_K < 1$. Whilst $\delta_K < 1$, the inequalities presented in (2.1) hold and further implies that all subsets of the K columns from A are almost orthogonal (however, they cannot be exactly orthogonal as we have more columns than rows as $M < N$).

The link between RIP and Compressed Sensing is described as follows:

- We want to recover a K-sparse signal using the measurement matrix **A**.
- Let δ_{2K} be sufficiently less than 1.

Then all pairwise distances between K-sparse signals must be well persevered in the measurement space which implies:

$$(1 - \delta_{2K})||\mathbf{x}_1 - \mathbf{x}_2||_2^2 \leq ||\mathbf{A}\mathbf{x}_1 - \mathbf{A}\mathbf{x}_2||_2^2 \leq (1 + \delta_{2K})||\mathbf{x}_1 - \mathbf{x}_2||_2^2 \tag{2.2}$$

holds for all K-sparse signal vectors $\mathbf{x}_1$ and $\mathbf{x}_2$. Further as $\mathbf{x}_1$ and $\mathbf{x}_2$ are different signal vectors, the following is true:

$$||\mathbf{x}_1 - \mathbf{x}_2||_2^2 > 0, \tag{2.3}$$

$$||\mathbf{A}\mathbf{x}_1 - \mathbf{A}\mathbf{x}_2||_2^2 > 0, \tag{2.4}$$

which means that the measurement matrix **A** maps the two signal vectors to two different output signal vectors $\mathbf{b}_1$ and $\mathbf{b}_2$. In order to reconstruct a K-sparse signal $\mathbf{x}$ from a sample output signal $\mathbf{b}$ by means of a stable process, we require the restricted isometry constant δ_{2K} to be small. Whilst it is computationally difficult to check if a matrix satisfies 2.1, it has been shown that many matrices satisfy the restricted isometry condition with large probability and few measurements. A concrete example would be random Gaussian or Bernoulli matrices which can be shown to satisfy the restricted isometry condition with very high probability. Further, any subguassian measurement matrix can be shown to follow the isometry condition. We define a subgaussian matrix as follows:

Definition Subgaussian
An random variable X is subgaussian if:

$$\mathbb{P}(|X| > t) \leq Ce^{vt^2} \tag{2.5}$$

for all $t > 0$ and some positive constants C and v.

The proof of subguassian matrices satisfying the restricted isometry condition is not shown here but can be researched elsewhere. [28]

Following this, we see that any subguassian random matrices have fat-tail distributions which is due to Gaussian random variables. Bernoulli matrices (whose entries are either +1 or -1) are subgaussian which is defined with 2.5 by choosing $C = \frac{1}{e}$ and $v = 1$. Further, we have that the Gaussian matrices (whose entries are identically distributed random variables from a Gaussian probability density function) are subgaussian by choosing $C = v = 1$.

An $M \times N$ subguassian matrix has restricted isometry behavior with large probability if the following condition holds:

$$M > C \cdot K \cdot log(N/K) \tag{2.6}$$

where is C some constant depending on the circumstance [10][12]. Therefore, we can recover K-sparse signals $\mathbf{x}$ of length N from output signals $\mathbf{b}$ of length M ($M < N$) after taking M random Gaussian or Bernoulli measurements through a measurement matrix $\mathbf{A}$.

Throughout, I will be using a Gaussian random matrix without checking if the above conditions are satisfied as they have been proven to satisfy the restricted isometry condition given they are subgaussian.

Chapter 3

Introduction to Basic Algorithms

3.1 Introduction

This section will give a short and succinct introduction to some of the basic algorithms used in compressive sensing with a more detailed look at some of them in chapter 4. Model-based compressed sensing will be discussed in more detail due to its nature.

Let $||\mathbf{x}||_0$ represent the number of non-zero elements in some vector $\mathbf{x}$. We want to retrieve the original vector $\mathbf{x}$ from the output vector $\mathbf{b}$ after sampling it with our measurement matrix $\mathbf{A}$. Mathematically speaking, we want to:

$$\min_{\mathbf{x}\in\mathbb{R}^{N\times 1}} ||\mathbf{x}||_0 \quad \text{subject to} \quad \mathbf{b} = \mathbf{A}\mathbf{x}. \tag{3.1}$$

We try and locate the sparsest vector possible from the measurable data we have available $\mathbf{A}\mathbf{x} = \mathbf{b}$. However, ℓ_0-minimization is non-deterministic polynomial-time hard (NP-hard) and so not very applicable in MRI. This is due to how computationally intensive it is to solve given the fact we want to retrieve MRI scans faster than previously achieved. There is another algorithm called Basis Pursuit or otherwise known as, ℓ_1-minimization which was proven equivalent[6] to the ℓ_0-minimization problem and is explored in detail in chapter 4.

3.2 Optimization Methods

3.2.1 Basis Pursuit

Whilst ℓ_0-minimization is NP-hard and a non-convex optimization problem and therefore potentially very difficult to solve, there exists another algorithm known as ℓ_1-minimization or Basis Pursuit.

Like ℓ_0-minimization, Basis Pursuit requires us to:

$$\min_{\mathbf{x}\in\mathbb{R}^{N\times 1}} ||\mathbf{x}||_1 \quad \text{subject to} \quad \mathbf{b} = \mathbf{Ax}. \tag{3.2}$$

Basis Pursuit is a convex optimization problem which is easier to solve than a non-convex as there exist accurate solvers. However, it does not take into account the complexity of the problem. An extension to this algorithm provides a solution to a more general case such as the presence of some random noise $\boldsymbol{\eta}$ discussed briefly below, but is explored in chapter 4.

3.2.2 Quadratically Constrained Basis Pursuit

This algorithm provides a solution to the general case when there is some error in the image and is also known as the noise-aware ℓ_1-minimization.

The mathematical representation of the problem is represented as follows:

$$\min_{\mathbf{x}\in\mathbb{R}^{N\times 1}} ||\mathbf{x}||_1 \quad \text{subject to} \quad ||\mathbf{Ax - b}||_2 \leq E. \tag{3.3}$$

This extension of the original algorithm applies better to the general case as it is very likely that the output signal vector $\mathbf{b} \in \mathbb{R}^{M\times 1}$ is not equivalent to the the vector $\mathbf{Ax} \in \mathbb{R}^{M\times 1}$ but is of the form $\mathbf{Ax}+\boldsymbol{\eta}$ instead where $\boldsymbol{\eta} \in \mathbb{R}^{M\times 1}$ is some error which produces the noise in the output signal.

Given some $E \geq 0$ we can estimate the error produced from our measurements using the ℓ_2-norm i.e. $||\boldsymbol{\eta}||_2 \leq E$. This again is a convex problem and can be solved in numerous ways. The solution to the quadratically constrained algorithm also provides a link to the Basis Pursuit Denoising algorithm which involves solving from some parameter $\lambda \geq 0$ such that we must:[1]

$$\min_{\mathbf{x}\in\mathbb{R}^{N\times 1}} \lambda||\mathbf{x}||_1 + ||\mathbf{Ax - b}||_2^2 \leq E. \tag{3.4}$$

3.2.3 Least Squares Solution

An ℓ_2-minimization problem, also known as the least squares solution, can also be constructed via the following equation:

$$\min_{\mathbf{z} \in \mathbb{R}^{N \times 1}} ||\mathbf{z}||_2 \quad \text{subject to} \quad \mathbf{b} = \mathbf{A}\mathbf{z}, \tag{3.5}$$

where given the transpose of $\mathbf{A}$, which is $\mathbf{A}^T$, we can use the following to formulate $\mathbf{z}$:

$$||\mathbf{z}||_2 = \mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{b} \tag{3.6}$$

as the best guess for $\mathbf{x}$ in the system $\mathbf{A}\mathbf{x} = \mathbf{b}$. However, recovering $\mathbf{x}$ exactly only occurs under special circumstances.

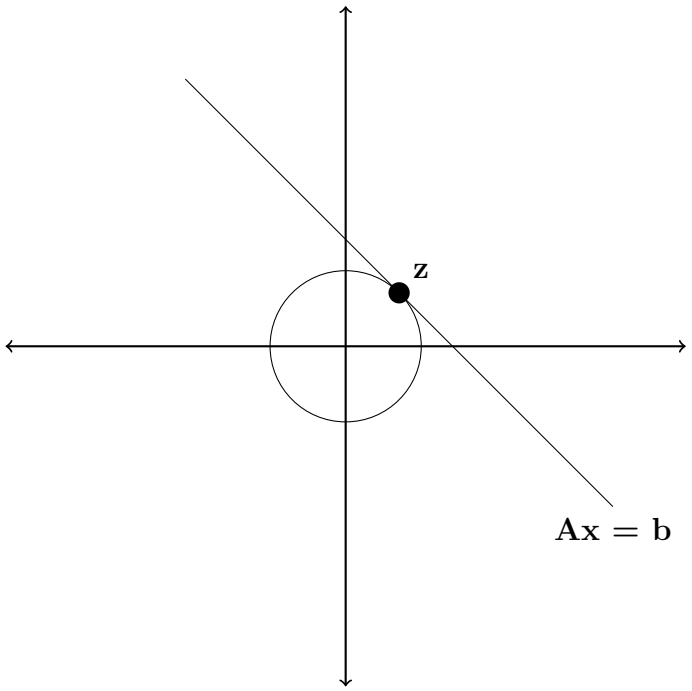


Figure 3.1: An example of a least squares guess

Suppose $\mathbf{A}\mathbf{x} = \mathbf{b}$ then if $\mathbf{x}$ is in the Range($\mathbf{A}^T$) (i.e. in the column space of the measurement matrix $\mathbf{A}$), we can recover $\mathbf{x}$ exactly.

As a result, if $\mathbf{x} = \mathbf{A}^T\mathbf{z}$ for some vector $\mathbf{z} \in \mathbb{R}^{M \times 1}$, then:

$$\mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{z} = \mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{A}\mathbf{x} \tag{3.7}$$
$$= \mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{A}^T\mathbf{z} \tag{3.8}$$
$$= \mathbf{A}^T\mathbf{z} \tag{3.9}$$
$$= \mathbf{x}. \tag{3.10}$$

To conclude, if $\mathbf{x}$ lies in some M-dimensional subspace of $\mathbb{R}^{N\times 1}$, it can be recovered from observations through the $M \times N$ measurement matrix $\mathbf{A}$. However, in many cases, the results produced by the least squares solution is not satisfactory and almost never gives a K-sparse solution and as such, I will not be investigating an ℓ_2-minimization. A low level illustration of a least squares solution is shown in figure 3.1

3.3 Greedy Methods

Whilst convex optimization methods are some of the most powerful methods for obtaining sparse representations of matrices, there are some iterative or greedy methods for solving similar problems.

Through the use of iterative approximations of the signal coefficients and support, greedy algorithms repetitively identify the support of a signal until some criteria for the convergence is met or it obtains a better approximation of the sparse signal at each iteration accounting for data which doesn't match.

3.3.1 Orthogonal Matching Pursuit

Orthogonal Matching Pursuit (OMP) is a greedy algorithm I will be investigating in more detail in chapter 4. A thorough analysis on the algorithm is available in a paper by Gilbert and Tropp[24].

Through the use of subgaussian measurement matrices, OMP is able to reconstruct sparse signals. Given a K-sparse signal $\mathbf{x} \in \mathbb{R}^{N\times 1}$ and a subguassian matrix $\mathbf{A} \in \mathbb{R}^{M\times N}$, the signal $\mathbf{x}$ can be represented by a signal $\mathbf{b} = \mathbf{Ax}$. Denoting the columns of $\mathbf{A}$ as $\mathbf{A}_1$, $\mathbf{A}_2$, $\mathbf{A}_3$, ..., $\mathbf{A}_N$, $\mathbf{b}$ can be thought of as a linear combination of the columns of $\mathbf{A}$ as $\mathbf{x}$ has only K non-zero coefficients. OMP works by greedily picking columns of $\mathbf{A}$ to find which participate in the construction of the output signal $\mathbf{b}$. This is done by iteratively locating the largest correlation between the residual of $\mathbf{b}$ and $\mathbf{A}$. At each iteration, one coefficient of the support vector is calculated and as a result, we could stop after K iterations for a K-sparse solution. The stopping criteria, however, could be defined as one or more of the following:

- **Residue norm:** If the norm of the residue is less than some $E \geq 0$ then we should stop.
- **Reaching target sparsity:** Given the sparsity value K, stop after K iterations.
- **Max number of iterations:** Stop OMP algorithm if iteration count reaches a certain number.

3.3.2 Stagewise Orthogonal Matching Pursuit

Stagewise Orthogonal Matching (StOMP) was introduced by Donoho and his collaborators[8]. StOMP, similarly to OMP, utilizes the signal $\mathbf{x} = \mathbf{A}^T\mathbf{b}$ where $\mathbf{b} = \mathbf{Ax}$ is the usual output signal. It improves on OMP and allows multiple coefficients to be added to the support in one iteration as opposed to OMP only allowing one coefficient to be added per iteration. This is possible due to the introduction of thresholding which allows all coefficients above a certain threshold to be added to the support. Following this, the least-squares problem is solved and updates the residual before iterating.

In order to describe thresholding strategies below, we need the following definitions:

Definition: False Alarm
A false alarm is the presence of a coefficient in the estimate of the support $\boldsymbol{I}^{(K)}$ *but not in the actual support* $\boldsymbol{I}$ *of the original signal* $\boldsymbol{x}$*.*

Definition: False Discovery
Given that the coefficients in our estimation of the support $\boldsymbol{I}^{(K)}$ *are called discoveries, false discoveries are all the coefficients in* $\boldsymbol{I}^{(K)}$ *but not in the actual support* $\boldsymbol{I}$ *of the original signal* $\boldsymbol{x}$*,*

Choosing the thresholding parameter involves choosing between two main strategies:

1 **False Alarm Control**: The threshold governed by this strategy focuses on limiting the false alarm rate so that it does not exceed a specific number per iteration.

2 **False Discovery Control**: This threshold is chosen to limit the fraction of incorrectly selected coefficients in our estimation for the support.

The algorithm for StOMP is defined as follows:

Algorithm 1 StOMP Algorithm

Inputs

1: Measurement matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$
2: Output signal vector $\mathbf{b} \in \mathbb{R}^{M \times 1}$
3: Threshold value: TV
4: The number of iterations (K) to perform (K = Sparsity value)

Outputs

1: A K-sparse Signal $\mathbf{x} \in \mathbb{R}^{N \times 1}$ which satisfies $\mathbf{b} = \mathbf{A}\mathbf{x}$

Steps

1: **procedure** StOMP(A, b, TV, K)
2: $\quad \mathbf{z}^{(0)} \leftarrow 0$
3: $\quad \mathbf{I}^{(0)} \leftarrow \emptyset$
4: $\quad$ **for** $i = 1, \ldots, K$ **do**
5: $\quad\quad \mathbf{r}^{(i)} \leftarrow \mathbf{b} - \mathbf{A}\mathbf{z}^{(i-1)}$ ▷ Get the Residual
6: $\quad\quad \mathbf{v}^{(i)} \leftarrow \mathbf{A}^T\mathbf{r}^{(i)}$ ▷ The observation vector
7: $\quad\quad \mathbf{C}^{(i)} \leftarrow \{j : |v_j^{(i)}| > TV\}$ ▷ Select coefficients using threshold
8: $\quad\quad \mathbf{I}^{(i)} \leftarrow \mathbf{I}^{(i-1)} \cup \mathbf{C}^{(i)}$
9: $\quad\quad \mathbf{z}^{(i)} \leftarrow \mathbf{A}^{\dagger}_{\mathbf{I}^{(i)}}\mathbf{b}$ ▷ Estimation (Note $\mathbf{A}^{\dagger} = (\mathbf{A}_T\mathbf{A})^{-1}\mathbf{A}_T\mathbf{b}$)
10: $\quad$ **end for**
11: $\quad$ **return** $\mathbf{x} \leftarrow \mathbf{z}^{(K)}$
12: **end procedure**

I will not be exploring this algorithm due to its similarity with OMP and CoSaMP both of which I will be investigating.

3.3.3 Compressive Sampling Matching Pursuit

Compressive Sampling Matching Pursuit (CoSaMP) is an extension to OMP developed by Needell and Tropp[9]. CoSaMP is similar to another algorithm introduce by the same authors known as Regularized Orthogonal Matching Pursuit (ROMP)[19]. Whilst ROMP is not discussed here, it suffices to note that CoSaMP was constructed to overcome the shortcomings of ROMP. A key difference between the two is that ROMP was designed to select more than one coefficient to be in the support set, which meant ROMP was able to make mistakes in the support set, but was still able to accurately reconstruct the original signal. Further, the uniform guarantees provided by the ROMP algorithms are persevered by CoSaMP without introducing a logarithmic term to the restricted isometric property.

The goal of CoSaMP is to identify the K largest coefficients in the source signal $\mathbf{x}$. Given that the measurement matrix $\mathbf{A}$ satisfies the restricted isometric property discussed earlier and given that the output signal is $\mathbf{b} = \mathbf{Ax}$, every K entries of the signal proxy $\mathbf{v} = \mathbf{A}^T\mathbf{Ax} = \mathbf{A}^T\mathbf{b}$ have ℓ_2-norms which are close to the ℓ_2-norms of the K corresponding entries in the source signal $\mathbf{x}$. In order to identify the K largest coefficients in the source signal $\mathbf{x}$, it is sufficient to identify the K largest coefficients in the proxy signal $\mathbf{v}$. At each iteration, the largest 2K coefficients of the signal proxy $\mathbf{v}$ are found and then by performing a least squares step, we are able to construct an estimate $\mathbf{z}$ for the signal $\mathbf{x}$ after which the signal is pruned so that it remains K-sparse.

3.4 Threshold Methods

Thresholding algorithms are algorithms which perform a thersholding function in each iteration. Let $\mathbb{T}$ be a thresholding function in an iterative method with the iteration counter being i, then an example of one a thresholding function would be as follows:

$$z^{(i)} = \mathbb{T}\big(f(x^{(i-1)})\big), \tag{3.11}$$

where f is a function which acts on the previous iteration of the variable $\mathbf{x}$.

There are two types of thresholding namely Hard Thresholding (HT) and Soft Thresholding (ST). We only discuss the former here but soft thresholding differs in its crucial updating step in each iteration.

Its iteration step is know as the *Landweber Iteration* which is as follows on the (i+1)th iterative step:

$$\mathbf{x}^{(i+1)} = \mathbb{S}\big(\mathbf{x}^{(i)} + \mathbf{A}^T\mathbf{b} - \mathbf{A}^T\mathbf{A}\mathbf{x}^{(i)}\big) \tag{3.12}$$

where $\mathbf{b} \in \mathbb{R}^{M\times 1}$, $\mathbf{x} \in \mathbb{R}^{N\times 1}$, $\mathbf{A} \in \mathbb{R}^{M\times N}$.

3.4.1 Hard Thresholding Algorithm

A hard thresholding algorithm denoted $\mathbb{H}_K$ sets every value, except the K-largest coefficient of absolute value in a vector $\mathbf{x} \in \mathbb{R}^{N\times 1}$, to 0. If this coefficient is not unique, the tie can be broken randomly or through orderings such as the lexicographical orderings. The algorithm is defined as follows:

Algorithm 2 Iterative Hard Thresholding

Inputs

1: Measurement matrix $\mathbf{A} \in \mathbb{R}^{M\times N}$
2: Output signal vector $\mathbf{b} \in \mathbb{R}^{M\times 1}$
3: The sparsity K of the original signal $\mathbf{x}$

Outputs

1: A K-sparse signal $\mathbf{x} \in \mathbb{R}^{N\times 1}$ which satisfies $\mathbf{b} = \mathbf{A}\mathbf{x}$

Steps

1: **procedure** IHT(A, b, K)
2: $\quad \mathbf{z}^{(0)} \leftarrow 0$
3: $\quad$ **for** $i = 1, \ldots$ **do**
4: $\quad\quad \mathbf{z}^{(i)} \leftarrow \mathbb{H}\big(\mathbf{z}^{(i-1)} + \mathbf{A}^T\big(\mathbf{b} - \mathbf{A}\mathbf{z}^{(i-1)}\big)\big)$
5: $\quad$ **end for**
6: $\quad$ **return** $\mathbf{x} \leftarrow \mathbf{z}^{(i)}$
7: **end procedure**

The following theorems developed by Thomas Blumensath and Mike E. Davies[3] show that the IHT algorithm will find an approximation that comes close to the true value of the signal $\mathbf{x}$. However, there is a limit to the accuracy of the solution and we find that for sparse source signals $\mathbf{x}$ with noisy observations $\mathbf{b}$ the success of the algorithm is limited by the size of the error. The number of iterations required to achieve a vector to a desired degree of accuracy depends on the logarithmic value of $\frac{||\mathbf{x}^K||_2}{\varepsilon_K}$ which can be thought of as a signal-to-noise ratio for sparse signal estimates for $\mathbf{x}$. It is large when our observation noise $\boldsymbol{\eta}$ and the signal $\mathbf{x}$ is approximated to a good degree of accuracy by a K-sparse vector. The proof of both the Theorem and Corollary can be seen in the original paper [3]

Theorem
Given a noisy observation $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{\eta}$ *where,* $\boldsymbol{b} \in \mathbb{R}^{M\times 1}$, $\boldsymbol{x}, \boldsymbol{\eta} \in \mathbb{R}^{N\times 1}$, $\boldsymbol{A} \in \mathbb{R}^{M\times N}$ *let* $\boldsymbol{x}^K$ *be a K-sparse approximation to* $\boldsymbol{x}$ *such that* $||\boldsymbol{x} - \boldsymbol{x}^K||_2$ *is minimal. If* $\boldsymbol{A}$ *obeys RIP with* $\delta_{3K} < 1/8$*, then at iteration r, IHT will recover an approximation* $\boldsymbol{x}^{(r)}$ *satisfying:*

$$||\boldsymbol{x} - \boldsymbol{x}^{(r)}||_2 \leq 2^{-r}||\boldsymbol{x}^K||_2 + 5\varepsilon_K, \tag{3.13}$$

given that

$$\varepsilon_K = ||\boldsymbol{x} - \boldsymbol{x}^K||_2 + \frac{1}{\sqrt{K}}||||\boldsymbol{x} - \boldsymbol{x}^K||_1 + ||\boldsymbol{\eta}||_2. \tag{3.14}$$

Following from this, IHT estimates $\boldsymbol{x}$ *with accuracy:*

$$||\boldsymbol{x} - \boldsymbol{x}^{(r*)}||_2 \leq 6\left[||\boldsymbol{x} - \boldsymbol{x}^K||_2 + \frac{1}{\sqrt{K}}||||\boldsymbol{x} - \boldsymbol{x}^K||_1 + ||\boldsymbol{\eta}||_2\right], \tag{3.15}$$

after a max of

$$r* = \left\lceil log_2\left(\frac{||\boldsymbol{x}^K||_2}{\varepsilon_K}\right)\right\rceil, \tag{3.16}$$

iterations.

For exact sparse signals, the following corollary is given:

Corollary
Suppose we are given a noisy observation $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{x}^K + \boldsymbol{\eta}$ *where,* $\boldsymbol{b} \in \mathbb{R}^{M\times 1}$ $\boldsymbol{x}, \boldsymbol{\eta} \in \mathbb{R}^{N\times 1}$, $\boldsymbol{A} \in \mathbb{R}^{M\times N}$ *where* $\boldsymbol{x}^K$ *is K-sparse. If* $\boldsymbol{A}$ *obeys RIP with* $\delta_{3K} < 1/8$*), then at iteration r, IHT will recover an approximation* $\boldsymbol{x}^{(r)}$ *satisfying:*

$$||\boldsymbol{x} - \boldsymbol{x}^{(r)}||_2 \leq 2^{-r}||\boldsymbol{x}^K||_2 + 4||\boldsymbol{\eta}||_2. \tag{3.17}$$

Following from this, IHT estimates $\boldsymbol{x}$ *with accuracy:*

$$||\boldsymbol{x} - \boldsymbol{x}^{(r*)}||_2 \leq 5||\boldsymbol{\eta}||_2, \tag{3.18}$$

after a a max of

$$r* = \left\lceil log_2\left(\frac{||\boldsymbol{x}^K||_2}{||\boldsymbol{\eta}||_2}\right)\right\rceil, \tag{3.19}$$

iterations.

3.5 Model-based Compressed Sensing

Whilst the some of the algorithms we have discussed to this point can robustly recover K-sparse and compressible signals $\mathbf{x}$ from just M = $\mathcal{O}(K \cdot log(N/K))$ noisy measurements through greedy methods or linear programs, it can still be done better through Model-based compressed sensing techniques.

Model based compressed sensing introduces the assumption that the sparse coefficients of a signal follow some structure. An example would be the modern wavelet image coders which exploit the fact that wavelet coefficients in natural images are small. A particular structure is followed by the values and locations of the large wavelet coefficients present in natural images. The main idea being that reconstruction algorithms which account for this structure can improve the reconstruction process[22]. Nothing previously discussed has required an assumption to be made about the position of the K non-zero and/or large coefficients of the signal $\mathbf{x}$, but we can look at real world examples where there is a solid structure present in the data which we can make use of to improve the reconstruction of the signals.

An example of a structure would be a structure which accounts for the clustering of large wavelet coefficients. Another is a connected tree structure

that have the most significant coefficients lie on the rooted subtree[2] an example of which is shown in figure 3.2.

By making considerations for the interdependence structure among the coefficients, the process of reconstruction requires fewer measurements from which robust recovery is possible. This is the foundation of model-based compressive sensing.

To understand the theorems from [22] we need the definitions provided below all of which are from the work of Baraniuk and his collaborators. Let $\mathbf{x}|_\Omega$ represent the entries of $\mathbf{x}$ corresponding to the set of indices $\Omega \subseteq \{1, \ldots, N\}$ where Ω^C denote the complement of the set Ω.

Definition Structured Signal Model
A structured signal model $\mathcal{M}_K$ is defined as the union of m_K canonical K-dimensional subspaces

$$\mathcal{M}_K = \bigcup_{m=1}^{m_K} \mathcal{X}_m \quad \textit{such that} \quad \mathcal{X}_m = \left\{ \boldsymbol{x} \in \mathbb{R}^N : \boldsymbol{x}|_{\Omega_m} \in \mathbb{R}^K, \boldsymbol{x}|_{\Omega_m^C} = 0 \right\}, \tag{3.20}$$

where each subspace $\mathcal{X}_m$ contains all signals $\boldsymbol{x}$ with supp$(\boldsymbol{x}) \in \Omega_m$ and so the model $\mathcal{M}_K$ is defined by the set of possible supports $\{\Omega_1, \ldots, \Omega_{mK}\}$. Signals from $\mathcal{M}_K$ are called K-model sparse.

Given that the source signal $\mathbf{x}$ is K-model sparse, the RIP constraint on the measurement matrix $\mathbf{A}$ can be relaxed and still achieve stable recovery from $\mathbf{b} = \mathbf{Ax}$

Definition $\mathcal{M}_K$-Restricted Isometric Property
The $\mathcal{M}_K$-restricted isometry property ($\mathcal{M}_K$-RIP) is followed bv an $M \times N$ matrix $\boldsymbol{A}$ if, with constant $\delta_{\mathcal{M}_K}$, for all $\boldsymbol{x} \in \mathcal{M}_K$, we have the following:

$$(1 - \delta_{\mathcal{M}_K})||\boldsymbol{x}||_2^2 \leq ||\boldsymbol{A}\boldsymbol{x}||_2^2 \leq (1 + \delta_{\mathcal{M}_K})||\boldsymbol{x}||_2^2. \tag{3.21}$$

The B-*Minkowski* defines an enlarged union of subspaces which includes the summation of all the elements in the model which is necessary for good performance of the model-based recovery of K-model sparse signals from the noise perturbed output signal.

Definition: B-Minkowski
The B-Minkowski sum for a set $\mathcal{M}_K$, for some integer $B > 1$ is defined as:

$$\mathcal{M}_K^B = \left\{ \boldsymbol{x} = \sum_{n=1}^{B} \boldsymbol{x}^{(n)}, \quad with \quad \boldsymbol{x}^{(n)} \in \mathcal{M}_K \right\}. \tag{3.22}$$

We can write $\mathcal{M}_K$ in place of $\mathcal{M}_K^1$ as you can easily show that $\mathcal{M}_K = \mathcal{M}_K^1$. In addition, $\mathcal{M}_K^B \subset \mathcal{M}_{BK}$. We can make use of the B-*Minkowski* sum to from a sparse approximation algorithm defined below.

Definition: Sparse Approximation Algorithm
We define $\mathbb{M}_B(\boldsymbol{x}, K)$ as the algorithm that obtains the best sparse approximation of $\boldsymbol{x}$ given the enlarged union of subspaces $\mathcal{M}_K^B$

$$\mathbb{M}_B(\boldsymbol{x}, K) = arg \min_{\hat{\boldsymbol{x}} \in \mathcal{M}_K^B} ||\boldsymbol{x} - \hat{\boldsymbol{x}}||_2 \tag{3.23}$$

where $\mathcal{M}_K^B$ is the B-Minkowski sum. Further, $\mathbb{M}(\boldsymbol{x}, K) = \mathbb{M}_1(\boldsymbol{x}, K)$ when $B = 1$.

From this, we can write $\mathcal{M}(\mathbf{x}, K)$ in place of $\mathcal{M}_1(\mathbf{x}, K)$ and given that generally, $\mathcal{M}_K^B \subset \mathcal{M}_{BK}$ we can get a better approximation for $\mathcal{M}_B(\mathbf{x}, K)$ from $\mathcal{M}(\mathbf{x}, BK)$

Leading from this, the number of measurements M necessary for a random compressed sensing matrix to have the $\mathcal{M}_K$-Restricted Isometric Property($\mathcal{M}_K$-RIP) was quantified by Blumensath and Davies for a given probability[4]

Theorem: Lower Bound on Measurements
Let $\mathcal{M}_K$ be the union of m_K subspaces of K-dimensions in $\mathbb{R}^{N \times 1}$. Then, for any $t > 0$ and any:

$$M \geq \frac{2}{c\delta_{M_K}^2} \big(ln(2mk) + Kln\frac{12}{\delta_{M_K}} + t \big), \tag{3.24}$$

an $M \times N$ identically distributed subgaussian random matrix $\boldsymbol{A}$ has the $\mathcal{M}_K$-RIP with the constant δ_{M_K} with probability at least $1 - e^{-t}$.

Substituting $m_K = \binom{N}{K} \approx (Ne/K)^K$, 3.24 can be used for the bound of the standard Restriced Isometric Property.

Model compressible signals are almost K-model sparse and live close to the restricted union of subspaces $\mathcal{M}_K$ which is similar to how almost K-sparse compressible signals live close to the union of the subspaces $\sum_K$ in $\mathbb{R}^{N\times 1}$

We define $\sigma_{M_K}(\mathbf{x})$ as ℓ_2 error produced by approximating the source signal $\mathbf{x} \in \mathbb{R}^{N\times 1}$ by the best model-based approximation $\hat{\mathbf{x}} \in \mathcal{M}_K$ as:

$$\sigma_{M_K}(\mathbf{x}) = \inf_{\hat{\mathbf{x}}\in\mathcal{M}_K} ||\mathbf{x} - \hat{\mathbf{x}}||_2, \tag{3.25}$$

then

$$\sigma_{M_K}(\mathbf{x}) = ||\mathbf{x} - \mathbb{M}(\mathbf{x}, K)||_2. \tag{3.26}$$

From this, given the approximation error decays according to the power law, we can define the set of s-model-compressible signals.

Definition: Decay According to Power Law
Let $\boldsymbol{x}$ be a signal whose coefficients are stored in decreasing order depending on their absolute value. They decay according to the power law:

$$|\boldsymbol{x}_{\mathcal{I}(i)}| \leq Si^{-1/r} \quad for \quad i = \{1, \ldots, N\}, \tag{3.27}$$

where $\mathcal{I}$ is an index for the sorted

Let $\mathbf{x}_K$ be the best K-sparse signal approximation for $\mathbf{x}$ containing the first K absolute largest coefficients.

The ℓ_p norm is given by the following:

$$\sigma_K(\mathbf{x})_p = arg \quad \min_{\hat{\mathbf{x}}} ||\mathbf{x} - \mathbf{x}_K||_p. \tag{3.28}$$

Then we have that:

$$\sigma_K(\mathbf{x})_p \leq (rs)^{-1/p} SK^{-s} \tag{3.29}$$

where s = $\dfrac{1}{r} - \dfrac{1}{p}$

Definition: S-Model-Compressible Signals
The set of s-model-compressible signals is defined to be:

$$\mathfrak{M}_s = \{\boldsymbol{x} \in \mathbb{R}^{N\times 1} : \sigma_{M_K}(\boldsymbol{x}) \leq SK^{-1/s}, 1 \leq K \leq N, S < \infty\} \tag{3.30}$$

Any $\boldsymbol{x} \in \mathfrak{M}_s$ is an s-model-compressible signal under the structured sparsity model $\mathcal{M}_K$.

Figure 3.2 shows a binary wavelet tree which is a possible structure for a model based problem. Without delving any deeper into the realm of Model-Based compressive sensing, we can write down the process for the CoSaMP variant of the model-based signal reconstruction algorithm.

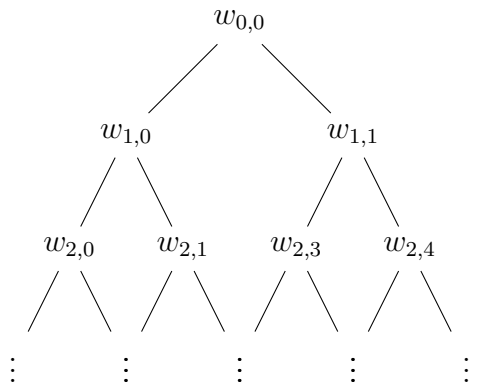


Figure 3.2: Binary wavelet tree for a one-dimensional signal. The nodes represent the large wavelet coefficients that arise from the discontinuities in smooth piecewise signals. The support of the large coefficients forms a rooted, connected tree.

3.5.1 Model-Based Algorithm (CoSaMP)

As the title suggests, the model-based signal recovery algorithm is based on the CoSaMP algorithm. We can construct the model-based algorithm by replacing the best K-sparse approximation and 2K-sparse approximation step in CoSaMP with a best (K-Model)-sparse approximation instead. This way, we search for m_K subspaces of $\mathcal{M}_K$ which is notably fewer than $\binom{n}{k}$ subspaces of $\sum_K$ in the CoSaMP recovery algorithm.

The following theorem sets an upper bound for the error when experimenting with model sparse signals:

Theorem
Let $\boldsymbol{x} \in \mathcal{M}_K$ *and let* $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{\eta}$ *where* $\boldsymbol{x}, \boldsymbol{\eta}, \in \mathbb{R}^{N\times 1}$, $\boldsymbol{b} \in \mathbb{R}^{M\times 1}$ *and* $\boldsymbol{A} \in \mathbb{R}^{M\times N}$ *be a set of noisy compressed sensing measurements. Then if,* $\boldsymbol{A}$ *has an* $\mathcal{M}_K^4$*-RIP constant of* $\delta_{\mathcal{M}_K^4} \leq 0.1$*, then the signal estimate* $\hat{\boldsymbol{x}}$ *obtained at iteration* i *of the model based CoSaMP algorithm satisfies:*

$$||\boldsymbol{x} - \hat{\boldsymbol{x}}||_2 \leq 2^{-i}||\boldsymbol{x}||_2 + 15||\boldsymbol{\eta}||_2. \tag{3.31}$$

Figure 3.3 shows the model-based signal reconstruction algorithm in action.

Figure 3.3: An example of a Model-Based reconstruction and the usual CoSaMP reconstruction. The left-most image is the original image with $n = 128 \times 128$ pixels. The images to the right were recovered taking M = 5000 measurements with the picture in the middle representing the usual CoSaMP reconstruction and the right-most image is the reconstruction attempt using Model-Based recovery(Images taken from: [22])

The algorithm for the CoSaMP variant of a Model-Based reconstruction algorithm is shown below.

Algorithm 3 Model Based CoSaMP

Inputs

1: Measurement matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$
2: Output signal vector $\mathbf{b} \in \mathbb{R}^{M \times 1}$
3: K the sparsity of the original signal $\mathbf{x}$
4: A model $\mathcal{M}_K$

Outputs

1: Signal $\hat{\mathbf{x}} \in \mathbb{R}^{N \times 1}$ where $\hat{\mathbf{x}}$ is K-sparse and is attuned to the model $\mathcal{M}_K$ given $\mathbf{b} = \mathbf{A}\hat{\hat{\mathbf{x}}}$

Steps

procedure MBCoSaMP(A, b, K, M)
  $\mathbf{x}^{(0)} \leftarrow 0$
  $\mathbf{v} \leftarrow \mathbf{b}$
  $i \leftarrow 0$
  while Halting Criterion False **do**
    $i \leftarrow i + 1$
    $z \leftarrow \mathbf{A}^T v$ ▷ Signal Proxy
    $\Omega \leftarrow supp(\mathbb{M}_2(\mathbf{z}, K))$ ▷ Support of residual approximation
    $I \leftarrow \Omega \cup supp(\mathbf{x}^{(i-1)})$ ▷ Augment Index Set
    $\mathbf{u}|_I \leftarrow \mathbf{A}_I^{\dagger}\mathbf{b}; \quad \mathbf{u}|_{I^c} \leftarrow 0$ ▷ Solve least squares
    $\mathbf{x}^{(i)} \leftarrow \mathbb{M}(\mathbf{u}, K)$ ▷ Prune according to Model
    $\mathbf{v} \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}^{(i)}$ ▷ Update Residual
  end while
  return $\hat{\mathbf{x}} \leftarrow \mathbf{x}^{(i)}$
end procedure

3.6 The Criterion for our Algorithms

Overall, to select the best algorithm, they must all be measured against certain criteria which I will define as follows:

1 The number of samples M needed for a given sparsity K to achieve accurate reconstruction.

2 The speed of the algorithm.

3 The algorithms accuracy and recovery rate including those perturbed by noise.

4 The applicability of the algorithm to all K-sparse signals.

The first criterion is relatively easy to test for by comparing the recovery rates with fixed values for the sparsity of each algorithm.

The second criterion similarly is accomplished through numerical tests and comparing algorithm run-times. However, a division of the results could be made so that you could identify algorithms which are fastest for datasets of a certain size. You could then determine which algorithm is the most reliable given the first criterion and comparing their logarithmic running time.

A third criterion to compare each algorithm would be its ability to reconstruct the vector with the lowest amount of error possible given the measured data. This allows you to see, if in the general case of an MRI scan with noise, which algorithm is able to reconstruct the MRI image with a large degree of likeness (or the least amount of discernible differences).

The fourth criteria requires testing to be done on many K-sparse signals and analysing the performance on each K-sparse signal as the number of measurements increase.

To conclude, there are many criterion for which you can compare algorithms and I will be using some of the above in my analysis. Each algorithm may be better suited to each task based on its properties whether its speed of reconstruction, accuracy or applicability for any level of sparsity or measurements. In my conclusion, I will measure the algorithms against this set criteria.

Chapter 4

Reconstruction Algorithms

4.1 Goals and Approaches

As a quick recap, we want to reconstruct a K-sparse signal $\mathbf{x} \in \mathbb{R}^{N\times 1}$ from our sample output signal $\mathbf{b} = \mathbf{A}\mathbf{x} \in \mathbb{R}^{M\times 1}$. Given $\mathbf{A} \in \mathbb{R}_{M\times N}$ and $M < N$, the system is undetermined as we have more unknowns than equations. We may have infinite solutions $\mathbf{z}$ for the original input signal $\mathbf{x}$, however, as our assumption is that $\mathbf{x}$ is K-sparse, we can find a sufficient approximation to $\mathbf{x}$ from $\mathbf{b}$. One of the most simple ways to recover such a vector from $\mathbf{b} = \mathbf{A}\mathbf{x}$ is by finding the sparsest vector (the one that has the most zero coefficients).

We have that a vector $\mathbf{x}$ is K-sparse if:

$$\mathbf{x} \in \mathbb{R}^{N\times 1}, \quad ||\mathbf{x}||_0 = |supp(\mathbf{x})| \leq K << N, \tag{4.1}$$

where $||\cdot||_0$ is a vector norm. We can define other norms, formally known as p-norms as $||\cdot||_p$ with $1 \leq p < \infty$

$$\mathbf{x} \in \mathbb{R}^{N\times 1}, \quad ||\mathbf{x}||_p = (\sum_{i=1}^{N} |x_i|^p)^{1/p}, \tag{4.2}$$

as well as

$$\mathbf{x} \in \mathbb{R}^{N\times 1}, \quad ||\mathbf{x}||_\infty = \max |x_i|. \tag{4.3}$$

The ℓ_0 norm counts the number of non-zero coefficients and, so we try to find the smallest possible norm of the our approximation to the input signal $\mathbf{x}$ that satisfies this. the ℓ_0-minimization problem as follows:

$$\min_{\mathbf{z}\in\mathbb{R}^{N\times 1}} ||\mathbf{z}||_0 \quad \text{subject to} \quad \mathbf{b} = \mathbf{A}\mathbf{z}. \tag{4.4}$$

Therefore, if the minimizer of 4.4 is $\mathbf{z}$, then we must have that $\mathbf{z} = \mathbf{x}$. The ℓ_0-minimization problem in 4.4, which works perfectly well in theory, is non-deterministic polynomial-time hard (NP-Hard) in general[17]. The problem is too computationally intensive for any measurement matrix $\mathbf{A}$ and output signal $\mathbf{b}$. However there are other ways to solve a similar problem to 4.4 through lineary programming or greedy pursuits[24], some of which are listed below:

1 Basis Pursuit (BP)[23].

2 Orthogonal Matching Pursuit (OMP)[25].

3 Regularized Orthogonal Matching Pursuit (ROMP) [19][18].

4 Compressive Sampling Matching Pursuit (CoSaMP).[9].

5 Others such as StOMP[8] or Threshold methods[3].

Whether we solve the optimization problem through linear programming or utilize greedy algorithms, both types of algorithms have their advantages and disadvantages which will be discussed next after our initial numerical experiments on MRI images.

The first algorithm which I will discuss is called Basis Pursuit or otherwise known as, ℓ_1-minimization which utilizes linear programming to solve a similar problem to 4.4.

For practical purposes, algorithms used in compressive sensing must be relatively fast in their reconstruction of a signal. As previously established, the time taken to produce high-resolution images can range between several minutes to multiple hours. As a result, the speed of a reconstruction algorithm is one of the biggest reasons why research in this area has been growing at a rapid pace and so is an important criterion for which the algorithms are test on.

4.2 Basis Pursuit

As established earlier, the ℓ_0 problem in equation 4.4 is an NP-Hard problem to recover the signal $\mathbf{x}$. This NP-hard problem, however, for certain matrices[11] $\mathbf{A} \in \mathbb{R}^{M \times N}$ we can perfectly recover K-sparse and noise free signals stably and approximate the compressible signal with large probability given $M = \mathcal{O}(KLog(N/K))$ identically distributed Gaussian measurements. Further the problem becomes equivalent to:

$$\min_{\mathbf{x} \in \mathbb{R}^{N \times 1}} ||\mathbf{x}||_1 \quad \text{subject to} \quad \mathbf{b} = \mathbf{A}\mathbf{x}. \tag{4.5}$$

The programs 4.4 and 4.5, are equivalent as proven by Candes and Tao[6]. An illustration of ℓ_1 minimization is show in figure 4.1.

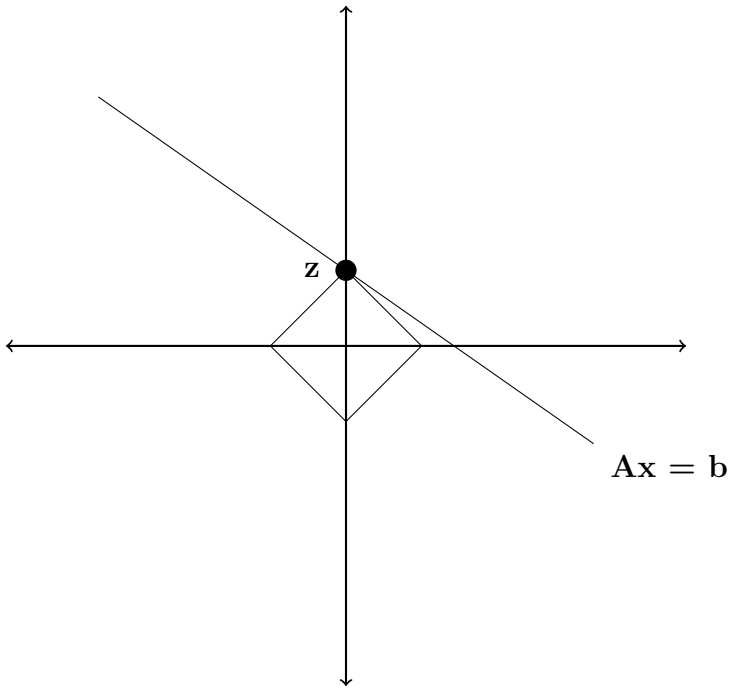


Figure 4.1: Here $\mathbf{x}$ is recovered exactly from $\mathbf{Ax} = \mathbf{b}$ via ℓ_1-minimization and $\mathbf{z} = \mathbf{x}$

For measurement matrices that satisfy the restricted isometry property as detailed in chapter 2, Basis Pursuit is able to achieve exact recover all sparse signals as showned by Candes and Tao[6]. The following theorems define the conditions for exact sparse recovery along side theorems describing the stability of BP.

Theorem: Exact Sparse Recovery under RIP
Assume that a measurement matrix $\boldsymbol{A}$ *satisfies the restricted isometry property. Then given that* $\boldsymbol{A}$*'s isometry constants obey:*

$$\delta_K + \delta_{2K} + \delta_{3K} < 1, \tag{4.6}$$

then any K-sparse signal $\boldsymbol{x}$ *can be recovered exactly given its unique solution from 4.5.*

Furthermore, these guarantees are uniform. Once a measurement matrix satisfies the restricted isometric property, Basis Pursuit is able to correctly recover all sparse signals.

A more realistic scenario is where our measurements are noisy and the signal is not exactly sparse. Both the signals and the measurements are usually noisy and so we seek a solution that permits these conditions. The program in 4.5 is not an appropriate meathod as the equality is not likely to be attainable. However, it has been shown that Basis pursuit is able to recover an approximation to noisy signals[13]. We are able to modify the method to be able to account for some unknown perturbations. We can denote this perturbation as a signal vector $\boldsymbol{\eta}$ which is bounded by some amount E i.e. $||\boldsymbol{\eta}||_2 \leq E$ and we can denote our noisy observation as $\mathbf{b} = \mathbf{A}\mathbf{x} + \boldsymbol{\eta}$. Further, small changes our observation of a signal, due to noise, should result in a small change in the recovery of the signal meaning that the method should be stable. We can denote the method as follows:

$$\min_{\mathbf{x} \in \mathbb{R}^{N \times 1}} ||\mathbf{x}||_1 \quad \text{subject to} \quad ||\mathbf{A}\mathbf{x} - \mathbf{b}||_2 \leq E. \tag{4.7}$$

An illustration of this program is given by figure 4.2

The following theorems show that the program is able to recover a sparse signal with an error that's proportional to the noise level[13].

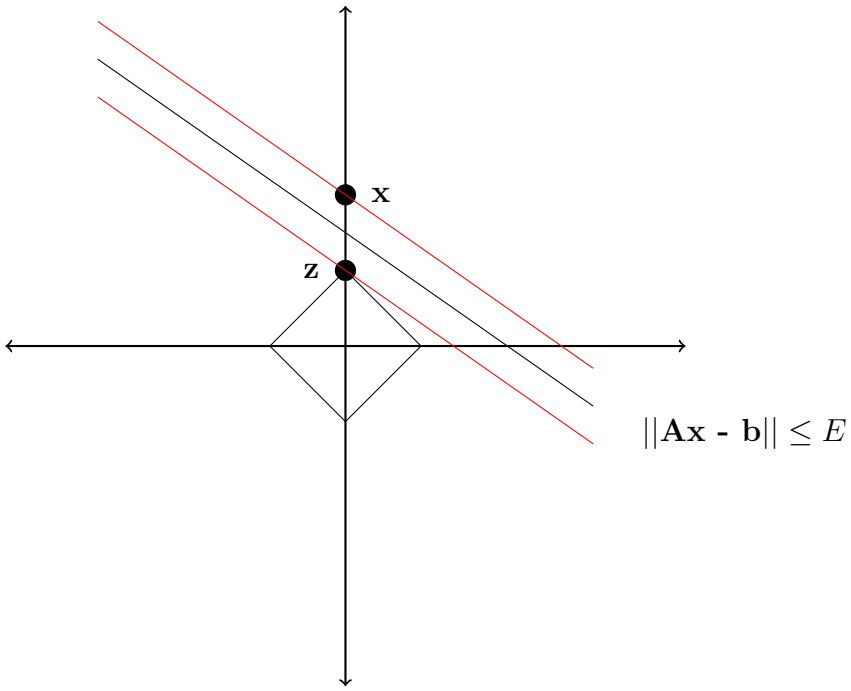


Figure 4.2: An illustration of the reconstruction of a sparse signal $\mathbf{x}$ approximately from noisy data ($\mathbf{b} = \mathbf{Ax} + \boldsymbol{\eta}$) where $\boldsymbol{\eta}$ is assumed to have a norm less than some error E.

Theorem: Noisy Signal Recovery Bound
Assume that a measurement matrix $\boldsymbol{A}$ *satisfies the restricted isometry property. Further, given K, assume the isometry constants obey* $\delta_{3K} + 3\delta_{4K} < 2$. *Then for any K-sparse signal* $\boldsymbol{x}$ *with perturbation* $\boldsymbol{\eta}$ *with* $||\boldsymbol{\eta}||_2 \leq E$, *there is a solution* $\boldsymbol{z}$ *such that:*

$$||\boldsymbol{z} - \boldsymbol{x}||_2 \leq C_K \cdot E, \tag{4.8}$$

where C_K *can only depend on* δ_{4k}.

This theorem is consistent with the noiseless case given in 4.6. Further, the majority of the singular values of the measurement matrix $\mathbf{A}$ is 0. This is because $\mathbf{A}$ is a rectangular matrix with fewer rows than columns. As a result, even though the problem is ill-posed, the error is controlled. In addition, the condition in equation 4.6 is optimal for perturbations of size E. To show this, assume that the knowledge of the *supp*($\mathbf{x}$) prior to computation. Then, the most optimal way to reconstruct the signal $\mathbf{x}$ would be to make use of

the least-squares method. Let $\mathbf{z}$ be the approximation to the original signal, then:

$$\mathbf{z} = \begin{cases} A_K^{\dagger} b, & \text{on supp}(\mathbf{x}). \\ 0, & \text{elsewhere.} \end{cases} \tag{4.9}$$

It is obvious that the error of $||\mathbf{z} - \mathbf{x}||_2 = 0$ on the complement of the $supp(\mathbf{x})$ where as the error on the support is given by: $||\boldsymbol{\eta}||_2 \leq E$. As a result, the best case scenario for the reconstruction of the signal is for it to have an error of which the size is proportional to the noise level.

The following theorem shows stable recovery of all sparse vectors is possible[13].

Theorem: Stability of Basis Pursuit
Assume that a measurement matrix $\boldsymbol{A}$ satisfies the restricted isometry property. Let $\boldsymbol{x}_K$ denote the vector of the largest coefficients of $\boldsymbol{x}$. Then for any K-sparse noisy signal $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{\eta}$ with perturbation with $||\boldsymbol{\eta}||_2 \leq E$, the solution $\boldsymbol{z}$ to 4.7 satisfies:

$$||\boldsymbol{z} - \boldsymbol{x}||_2 \leq C_K \cdot E + C_K^T \cdot \frac{||x - x_K||_1}{\sqrt{K}}. \tag{4.10}$$

The theorem states that Basis Prusuit stabely recovers the K-largest entries of some signal $\mathbf{x}$.

To conclude, we now know that Basis Pursuit can exactly recover noiseless sparse signals and recovery of noisy signals is stable.

4.2.1 Algorithm

The algorithm is given as follows:

Algorithm 4 Basis Pursuit via Dual Point Algorithm

Inputs

1: Measurement matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$
2: Output signal vector $\mathbf{b} \in \mathbb{R}^{M \times 1}$

Outputs

1: A K-sparse Signal $\mathbf{x} \in \mathbb{R}^{N \times 1}$ which satisfies $\mathbf{b} = \mathbf{A}\mathbf{x}$

Steps

1: **procedure** BP(A, b)
2: $\quad \mathbf{z} \leftarrow 0$
3: $\quad \mathbf{v} \leftarrow \mathbf{A}^T\mathbf{b}$ ▷ Initial Guess
4: $\quad \mathbf{z} \leftarrow \mathcal{DP}(\mathbf{v}, \mathbf{A}, \mathbf{b})$ ▷ Dual Point Method (L1 Magic)
5: $\quad$ **return** $\mathbf{x} \leftarrow \mathbf{z}$
6: **end procedure**

The key to this algorithm is clearly the function $\mathcal{DP}$ and a detailed breakdown of the function is available online[5].

4.2.2 1D Signal - Basis Pursuit

The following example makes used of ℓ_1-Magic[5] for the implementation of Basis Pursuit. We begin by creating a sparse-time signal. The signal shown in figure 4.3.

The code is available in the appendix and the following graphs were generated with the following parameters:

- Signal Length (N) = 256.
- Measurements (M) = 64.
- Sparsity/Peaks (K) = 20.
- rng(10) (A random number seed to replicate results)

We measure the quality of the reconstruction via PSNR which is the Peak-Signal-to-Noise Ratio. PSNR is defined as follows:

$$PSNR = 10 \cdot log_{10}\left(\frac{MAX_I^2}{MSE}\right), \tag{4.11}$$

where I is an $M \times N$ matrix. MAX is the maximum component of the signal and the Mean Squared Error (MSE) is given by:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{z} - \mathbf{x})^2, \tag{4.12}$$

where $\mathbf{z}$ and $\mathbf{x}$ vectors of length N. As the value of the PSNR grows larger, the better the recovery performance. A low PSNR value indicates a low-quality reconstruction. We also display the ℓ_2-norm between the output and input signal on each graph.

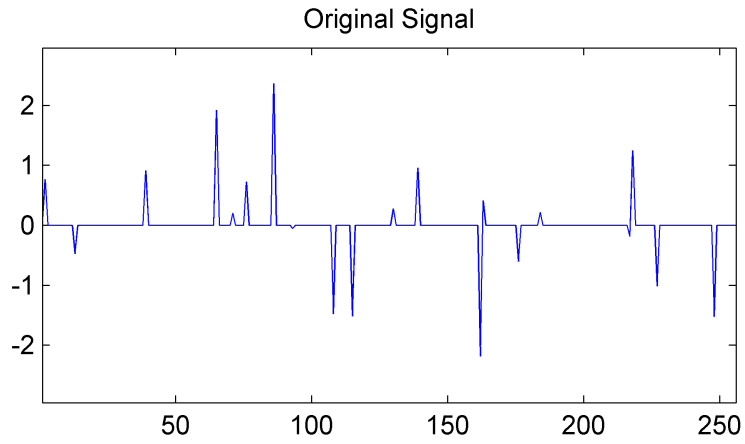


Figure 4.3: The plot of the (N = 256) original signal $\mathbf{x}$ with 20 peaks

We then obtain 64 samples of the original signal as shown in the below figure:

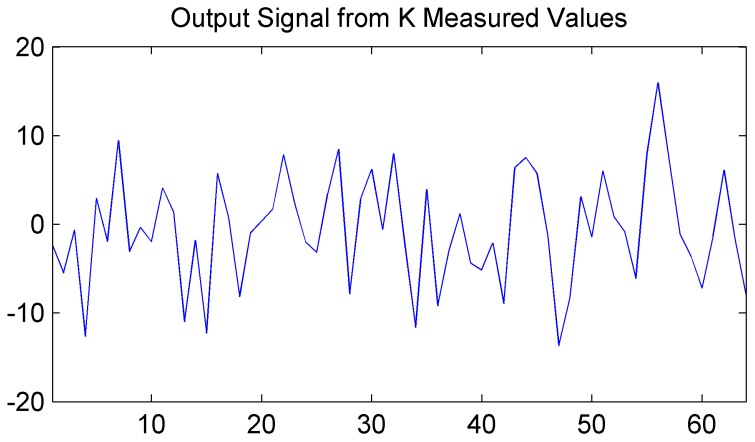


Figure 4.4: The plot of the (M = 64) signal generated by $\mathbf{Ax}$

We perform the Basis Pursuit algorithm and print the reconstructed signal along with its PSNR and 2-norm error:

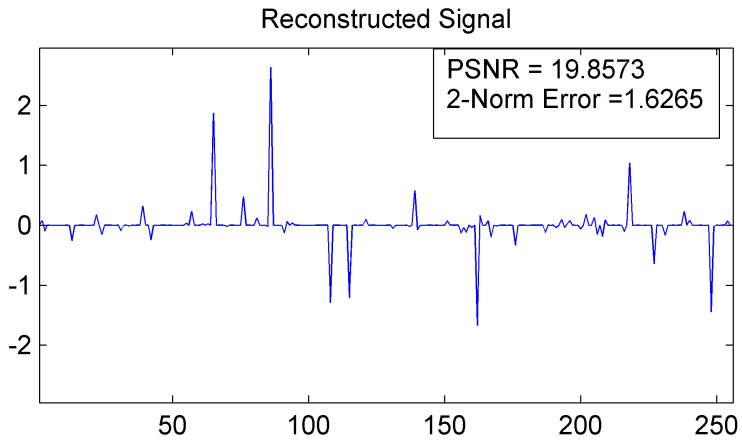


Figure 4.5: The plot of the (N = 256) signal reconstructed via ℓ_1-minimization

Using only 64 samples taken using a measurement matrix with Guassian random variables as its entries, we recovered the 20-sparse original signal with a PSNR of 19.86. Taking a larger amount of samples of the same signal would results in a large PSNR i.e. a more accurate reconstruction of the original signal. Keeping the amount of samples constant and increasing the sparsity, meaning more zero coefficients in the signal, would have also increased the PSNR of the reconstruction when compared to the original.

4.2.3 1D Signal - Basis Pursuit and Noisy Signals

This section details the effect of noise and how the BP algorithm copes. As such, the equations which the signal follows is listed here:

$$\min_{\mathbf{x}\in\mathbb{R}^{N\times 1}} ||\mathbf{x}||_1 \quad \text{subject to} \quad ||\mathbf{Ax}-\mathbf{b}||_2 \leq E. \tag{4.13}$$

Figures 4.6 to 4.10 show an example of a noise perturbed signal. The code of the for the below graphs can be located in the appendix. The quadratically constrained version of BP is needed to reconstruct a signal from a noisy observation. We again make use of ℓ_1-Magic to compute the solution.

Figure 4.6 shows the original signal constructed. Notice how this signal is not sparse and as such, needs to be transformed into a sparse domain before we can compute its reconstruction using BP.

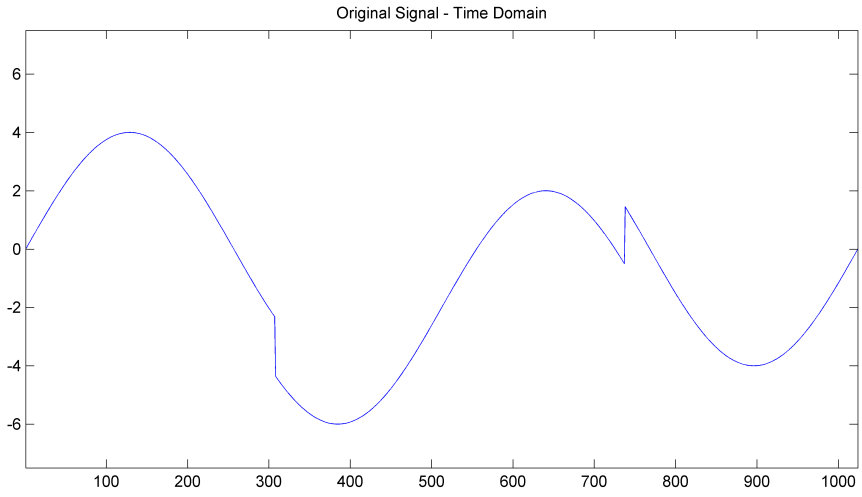


Figure 4.6: The plot of the (N = 1024) original signal $\mathbf{x}$

We can transform the signal into to the Discrete Cosine Transform (DCT).

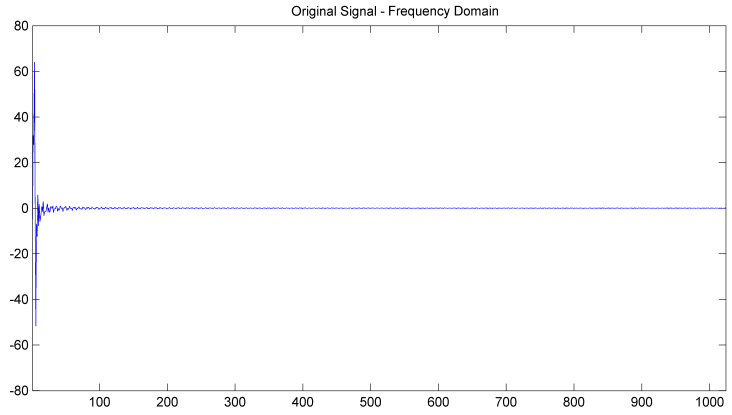


Figure 4.7: The plot of the Discrete Cosine Transform (DCT) coefficients for the (N = 1024) original signal. Notice how the majority of the coefficients are approximately 0.

Given that our signal under DCT is sparse, we are able to apply the BP algorithm to the DCT coefficients of the signal. We can easily find the coefficients of corresponding sinusoid components and these coefficients (in the frequency domain) are shown in figure 4.7. Figure 4.8 shows the 200 samples taken of the signal in the time domain as opposed to the frequency domain where we can find the DCT coefficients.

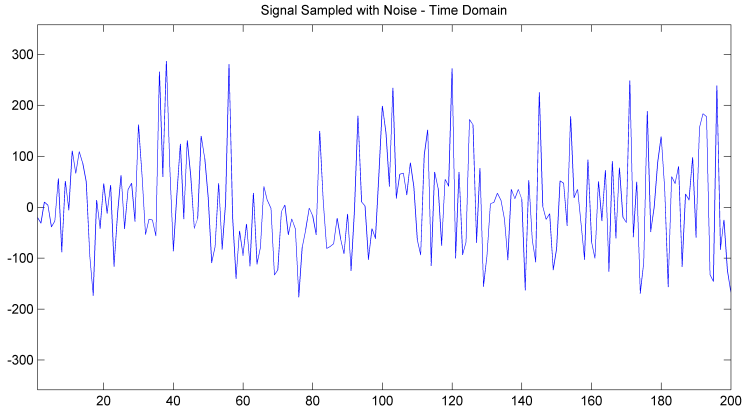


Figure 4.8: The plot of the (M = 200) signal generated by $\mathbf{A}^*\mathbf{x} + \boldsymbol{\eta}$ where $\boldsymbol{\eta}$ is some error vector.

Figure 4.9 shows the reconstruction of the original signal from 200 samples. Figure 4.10 shows the coefficients of the DCT of the reconstructed signal.

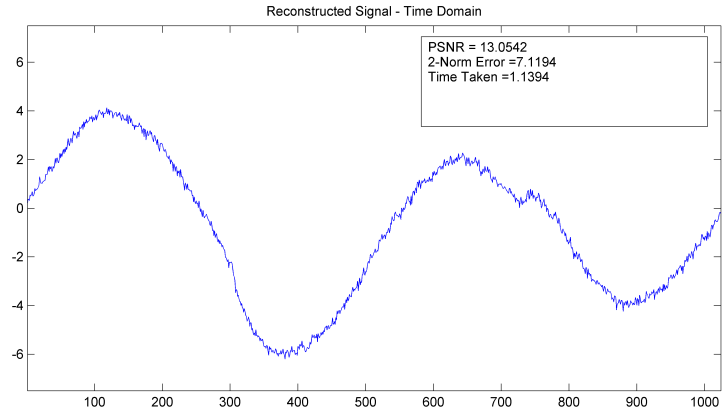


Figure 4.9: The reconstructed signal (N = 1024) from (M = 200) the output vector $\mathbf{Ax} + \boldsymbol{\eta}$ where $\boldsymbol{\eta}$ is some error vector

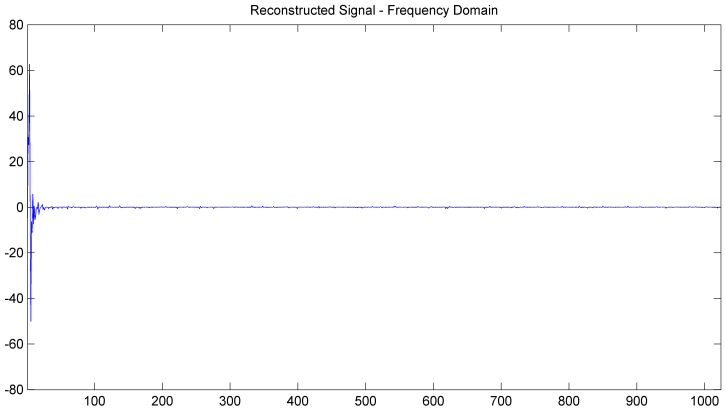


Figure 4.10: The plot of the Discrete Cosine Transform (DCT) coefficients for the (N = 1024) reconstructed signal.

We can see why the PSNR is low visually and similarly for the error. From this, I generate 4 more figures 4.12 to 4.15 to demonstrate the effect of an increasing number of measurements on the time taken to reconstruct the signal, the PSNR of the reconstruction and the 2-Norm error

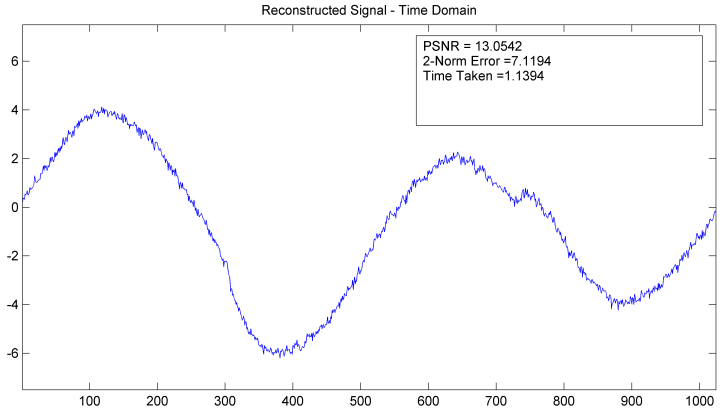


Figure 4.11: The reconstruction of the original signal (N = 1024) using 200 measurements

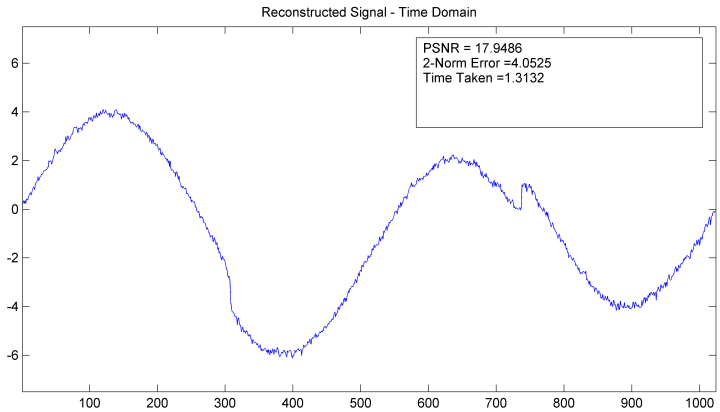


Figure 4.12: The reconstruction of the original signal (N = 1024) using 400 measurements

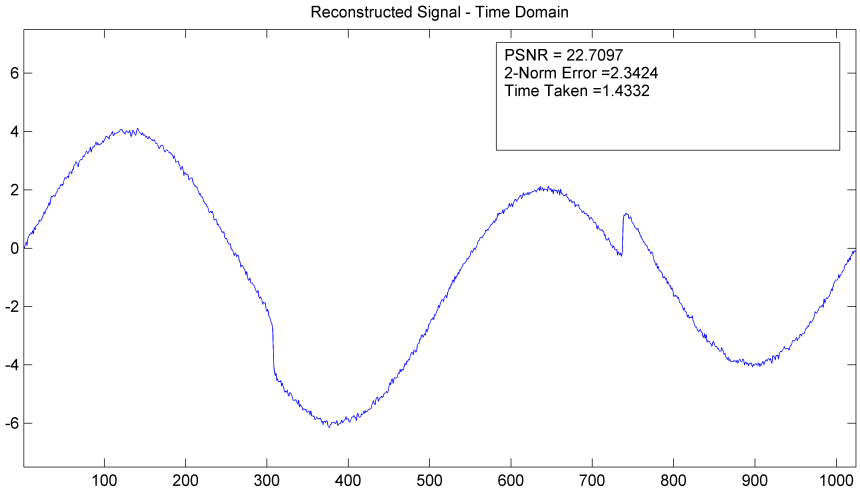


Figure 4.13: The reconstruction of the original signal (N = 1024) using 600 measurements

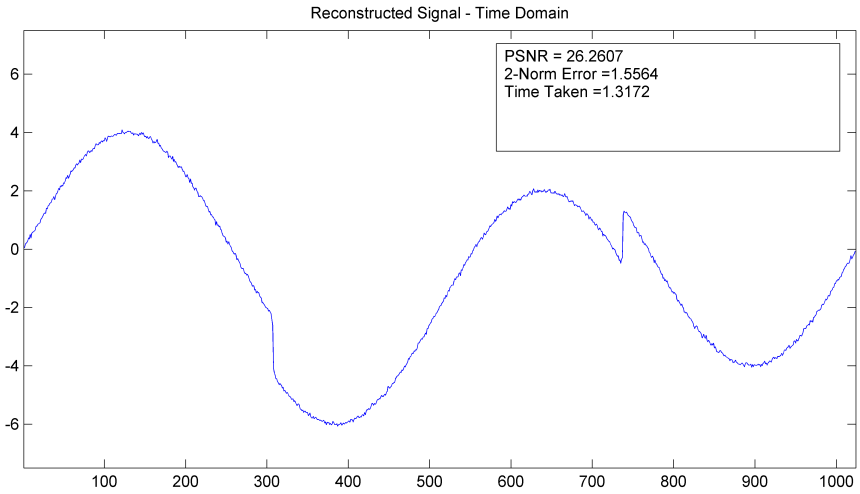


Figure 4.14: The reconstruction of the original signal (N = 1024) using 800 measurements

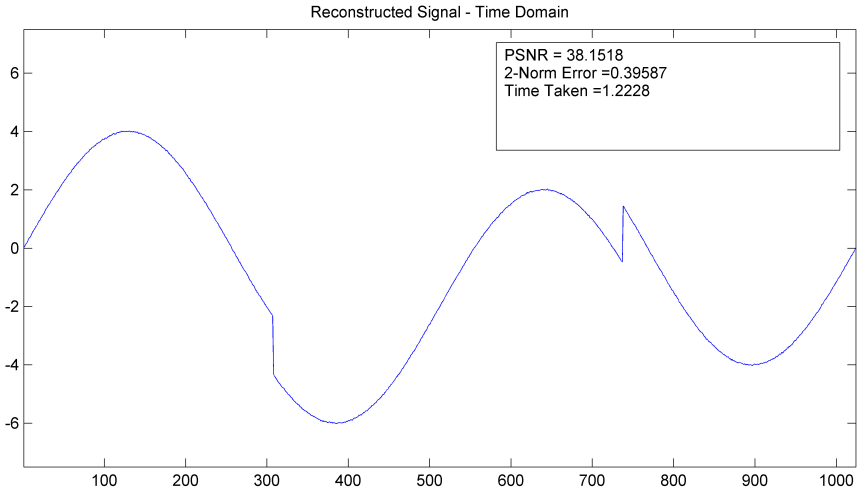


Figure 4.15: The reconstruction of the original signal (N = 1024) using 1000 measurements

The above figures clearly show that as we take more measurements and execute the BP algorithm, the PSNR of the reconstructed signal increases and continually becomes more similar to the original signal presented in figure 4.6. The time taken for the execution of the algorithm fluctuates between 1.1 and 1.4 seconds and the 2-Norm also continually decreases as we take more measurements.

To conclude, Basis Pursuit is able to reconstruct signals from noisy observations to a high accuracy. As shown earlier, however, the resulting signal 2-norm error is bounded by the 2-norm of the error vector.

4.2.4 1D Signal - Basis Pursuit Numerical Results

The code for the graphs in this section are found in the appendix. In every case, the measurement matrix used was a Gaussian matrix and the graphs were all produced using MATLAB. Outside functions used include: ℓ_1-Magic[5].

This first graph was generated by running my code for the given parameters:

1 Sample measurements ranging from 1 to 100.

2 Sparsity levels ranging from 5 to 30.

For each point on the graph, 100 values were taken and averaged to represent the data point. Figure 4.16 shows the relationship between the measurements M and the sparsity K and their effect on the PSNR of the reconstructed signal. We can clearly see that as the number of measurements increase, so does the PSNR of the reconstructed signal. Further, as the sparsity of signal decreases, the lower the PSNR of the reconstructed image. We can see that, for example, at 50 measurements the PSNR for the reconstruction of a 15-sparse signal is approximately 40dB. As the signal gains more non-zero values, the PSNR decreases as shown for K = 30 where the PSNR is approximately 10dB. If the sparsity of the signal was to increase to say, K = 10, we can see that the PSNR of the reconstructed signal jumps to approximately 90dB. For our given measurements, the graphs seem to approach a limit at a point below 140dB. From this we can conclude that there exists a point where the computational cost of reconstructing an image from a large number of measurements is too large for the increase in PSNR achieved by doing so unless there exists another large "jump" as seen for every sparsity level measured.

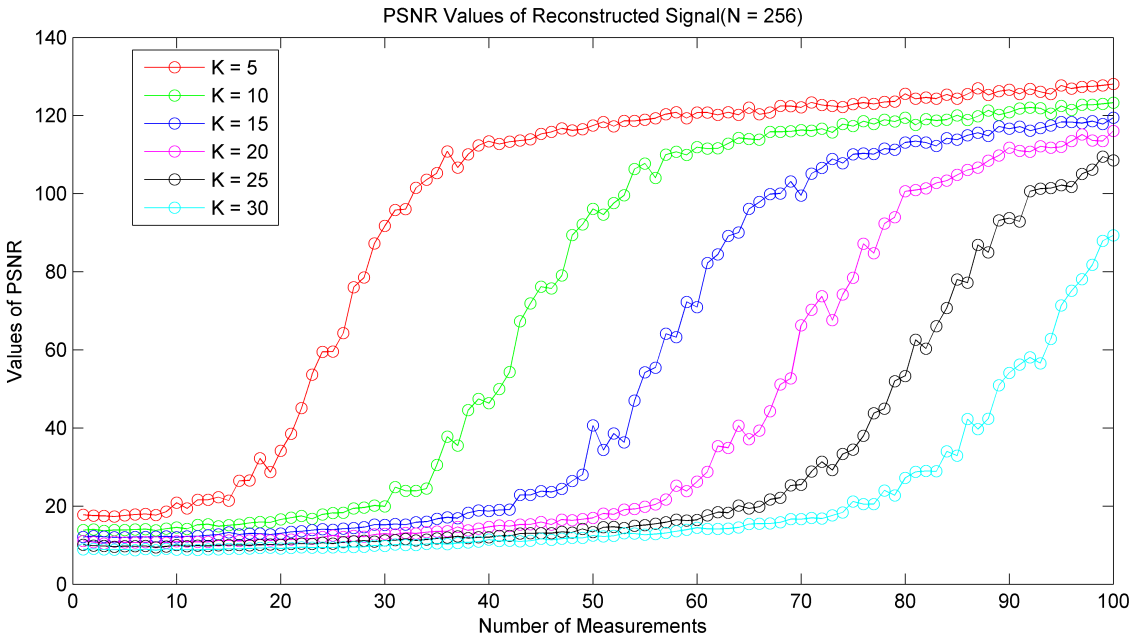


Figure 4.16: Plot of how PSNR values change for a N = 256 dimensional signal with increasing samples (1 to 100 in steps of 1) and 6 levels of sparsity (5 to 30 in steps of 5). The output is the average of 100 trials that were run for each combination of the number of samples and sparsity.

Figure 4.17 shows the relationship between the measurements M and the sparsity K and their effect on the 2-Norm error of the reconstructed signal. Again, the effect is obvious as we can see that an increase in measurements coincides with the deprecation of the 2-Norm error value. The sparser the signal, the smaller the initial 2-Norm error and the faster that error decays as the number of measurements increases. We can approximate the gradient of the curves by finding the gradient of the line of best fit. For K = 5, we see that the error becomes approximately 0 at M = 35. We can approximate that at M =1, the error is 2.5 and as such, the gradient is approximately 0.071. For K = 15, the error is approximately 0 at 70 and we can assume that a line of best fit would coincide with error value 4.5. This gives a gradient of 0.064. From our results, we can conclude that the denser the signal is, the slower 2-norm error decays to 0.

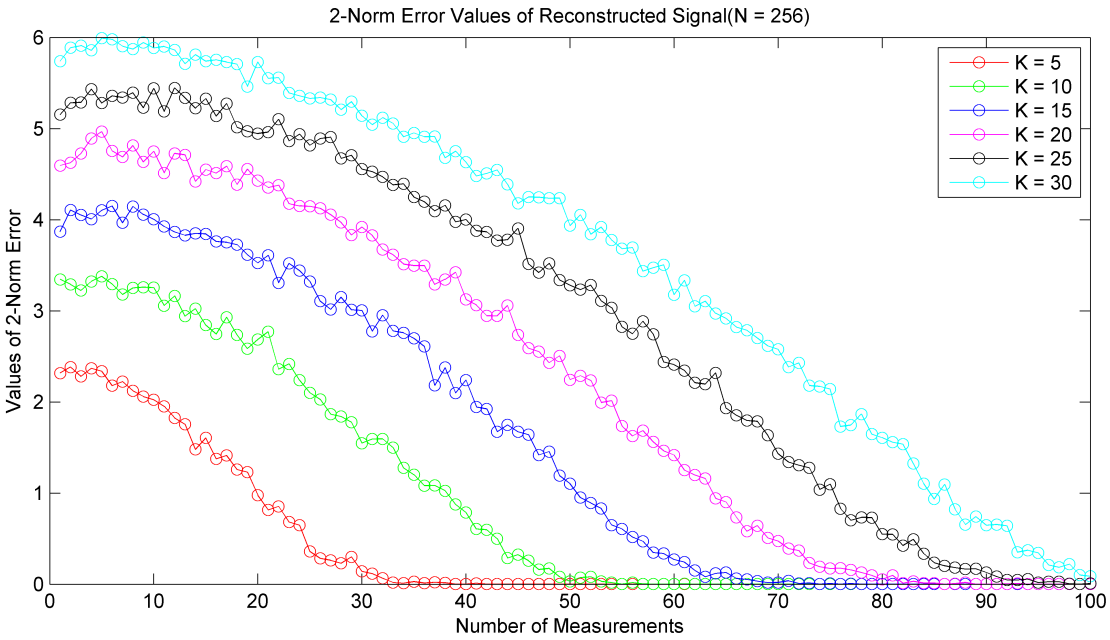


Figure 4.17: Plot of how the 2-norm error values change for a N = 256 dimensional signal with increasing samples (1 to 100 in steps of 1) and 6 levels of sparsity (5 to 30 in steps of 5). The output is the average of 100 trials that were run for each combination of the number of samples and sparsity. Code for graph generation is available in Appendix H

The graph in Figure 4.18 displays how many signals BP is able to correctly construct given the threshold that the 2-norm must be less than 0.01. For very sparse signals we can clearly see that the measurements required to achieve 100% successful recovery is small i.e. K = 5 requires approximately 35 measurements. For the measurements and sparsity levels displayed, we notice that for an additional 5 non-zero values present in a signal, the amount of measurements required to achieve 100% correct reconstruction increases by 20 i.e for K = 10, M = 55 which is 20 more than for K = 5. Our displayed results show that 100% successful reconstruction is possible for $K \geq 5$ given that we take $M \geq 20$ measurements more than those that were needed for a signal with 5 less non-zero values. Due to this constant, the ratio of measurements and sparsity for 100% signal reconstruction decreases as the signal becomes more dense.

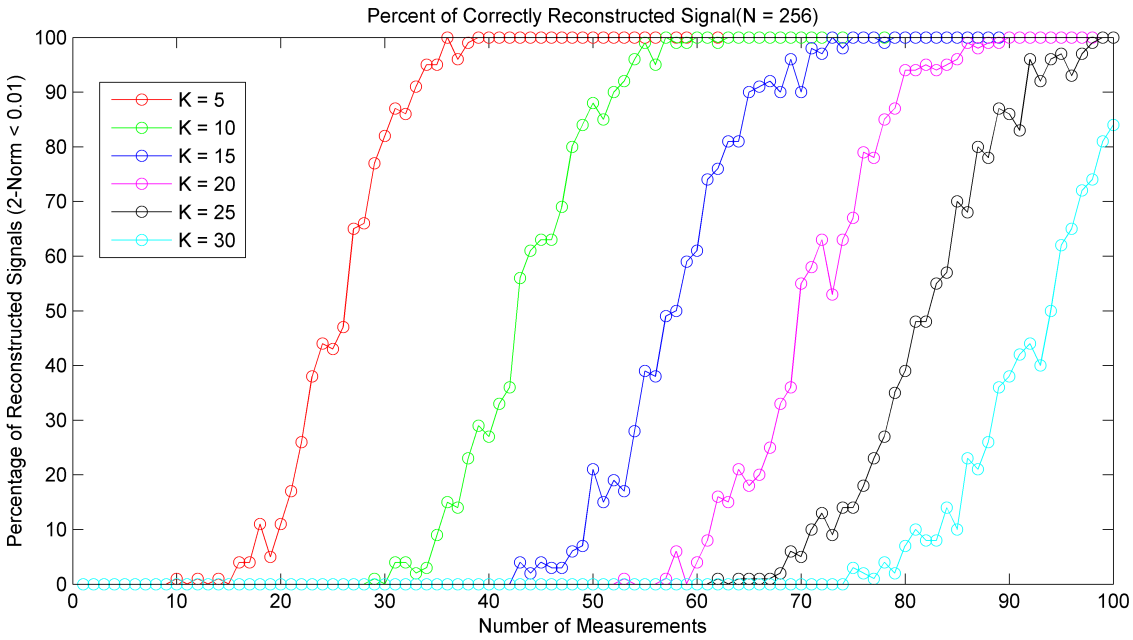


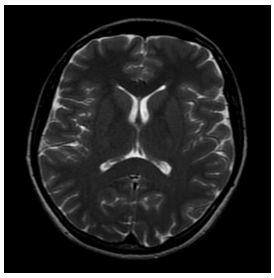
Figure 4.18: Plot of how the percentage of signals correctly recovered change for a N = 256 dimensional signal with increasing samples (1 to 100 in steps of 1) and 6 levels of sparsity (5 to 30 in steps of 5). The output is the average of 100 trials that were run for each combination of the number of samples and sparsity. Code for graph generation is available in Appendix H

4.3 MRI Images - Experiments

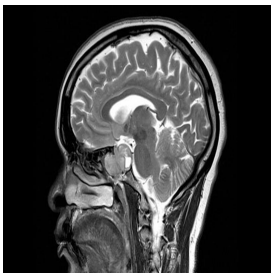
I will perform my numerical experiments on the two MRI images and they will remain the same for BP, OMP and CoSaMP. However, due to data processing ability of my computer, every image will be of size $N = 80 \times 80$ in my experiments carried out. As a result the images are small and I've had to magnify them, however, the change is obvious to see in the majority of the cases I'll explore. These images need to be transformed into a sparse domain and to that end I threshold the largest 10% of the values if the followings domains:

- Discrete Cosine Transform (Denoted: DCT)
- Fourier Transform (Denoted: FFT)
- Discrete Wavelet Transform (Denoted: DWT)

The original MRI images are given below but the K-sampled images will be given for each of the transforms we perform.



(a) MRI - Brain 1 : T2-weighted MR image of a normal brain[21].



(b) MRI - Brain 2: 3T MRI image of a human head with a pituitary tumour[15].

Figure 4.19: I will be using the following MRI scans to test the CS algorithms.

4.4 Basis Pursuit - Experiments

4.4.1 Brain 1 - Image Reconstruction

I begin my numerical experiments on the MRI image 4.19a. Each column will represent the image produced from each transform which from left to right is: DCT, FFT, DWT.

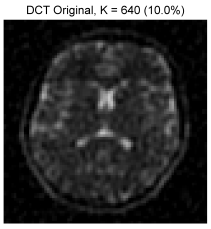


Figure 4.20: DCT: Image K = 10%

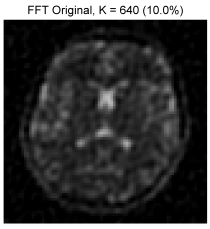


Figure 4.21: FFT: Image K = 10%

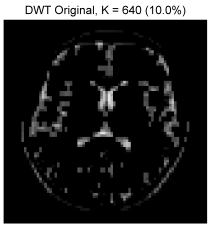


Figure 4.22: DWT: Image K = 10%

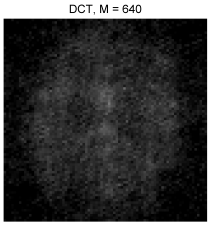


Figure 4.23: DCT: Image M = K = 640

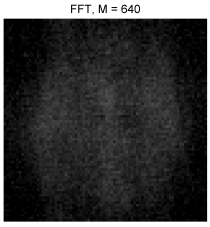


Figure 4.24: FFT: Image M = K = 640

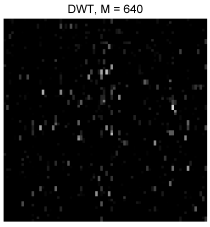


Figure 4.25: DWT: Image M = K = 640

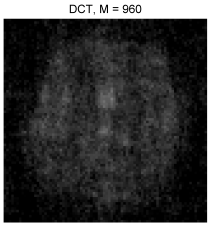


Figure 4.26: DCT: Image M = 1.5K = 960

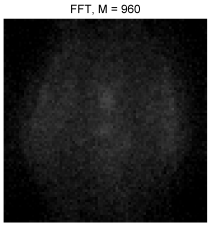


Figure 4.27: FFT: Image M = 1.5K = 960

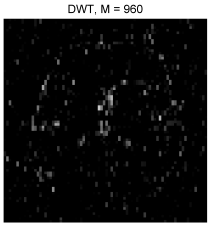


Figure 4.28: DWT: Image M = 1.5K = 960

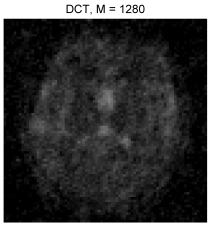


Figure 4.29: DCT: Image M = 2K = 1280

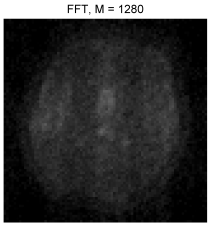


Figure 4.30: FFT: Image M = 2K = 1280

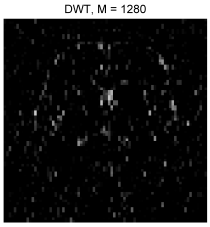


Figure 4.31: DWT: Image M = 2K = 1280

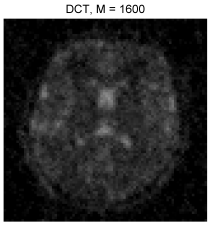


Figure 4.32: DCT: Image M = 2.5K = 1600

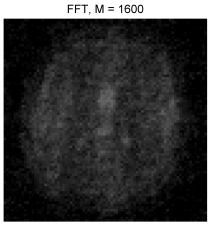


Figure 4.33: FFT: Image M = 2.5K = 1600

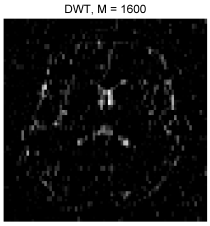


Figure 4.34: DWT: Image M = 2.K = 1600

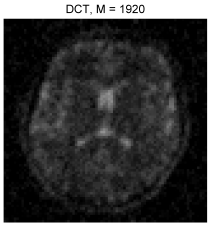


Figure 4.35: DCT: Image M = 3K = 1920

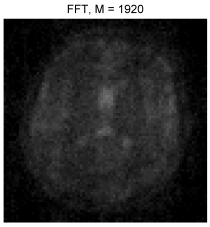


Figure 4.36: FFT: Image M = 3K = 1920

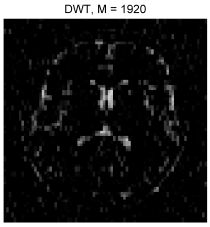


Figure 4.37: DWT: Image M = 3K = 1920

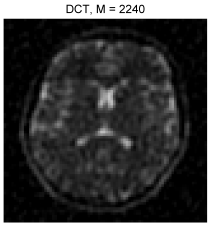


Figure 4.38: DCT: Image M = 3.5K = 2240

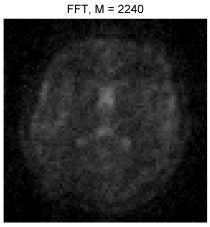


Figure 4.39: FFT: Image M = 3.5K = 2240

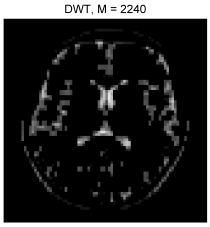


Figure 4.40: DWT: Image M = 3.5K = 2240

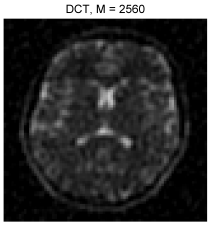


Figure 4.41: DCT: Image M = 4K = 2240

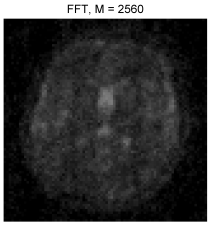


Figure 4.42: FFT: Image M = 4K = 2560

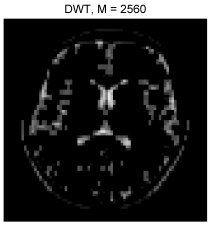


Figure 4.43: DWT: Image M = 4K = 2560

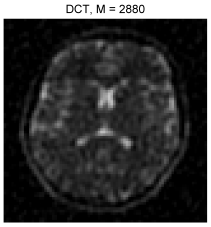


Figure 4.44: DCT: Image M = 4.5K = 2880

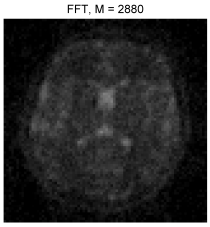


Figure 4.45: FFT: Image M = 4.5K = 2880

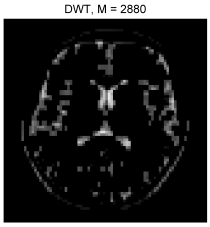


Figure 4.46: DWT: Image M = 4.5K = 2880

4.4.2 Brain 1 Image - Analysis

Following the results, I generated some graphs showing how the time, error and PSNR changed as I increased the number of measurements and these graphs are shown below. The x-axis is labelled so that the measurements in each image is displayed as a multiple of the sparsity K of the original image.

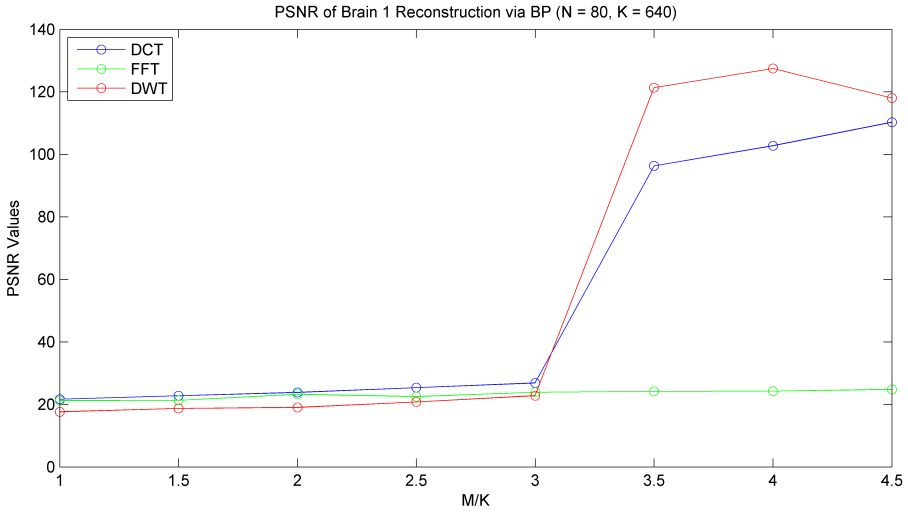


Figure 4.47: Plot of how the PSNR values change for an increasing number of measurements for Brain 1.

We immediately note that there is a problem with the FFT reconstruction of the image. This is due to the algorithm in ℓ_1-Magic where the programs gets stuck backtracking in its line search and returns to its last successful iterations. As a result, all the images look similar and we aren't able to read too much into the results of FFT reconstructions. More informations is available about this in the relevant documentation for ℓ_1-Magic.

As expected, however, the PSNR does increase, no matter how small as the number of measurements increases. The largest jump is see at M = 3.5K after which the signals shows a slow rise until stabilization to the maximum PSNR. The error in reconstruction behaves similar to that of the graph of the PSNR values with the biggest change in error to almost 0 is at M = 3.5K. The time taken to perform the algorithm on each transform is shown below with the wavelet transform being the fastest. In general, we see a slow rise in the time taken as M/K increases which is to be expected.

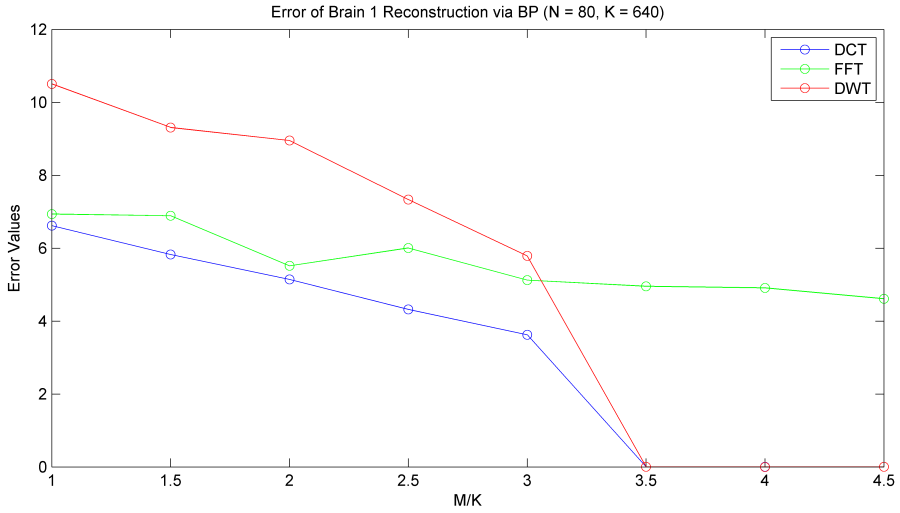


Figure 4.48: Plot of how the error values change for an increasing number of measurements for Brain 1.

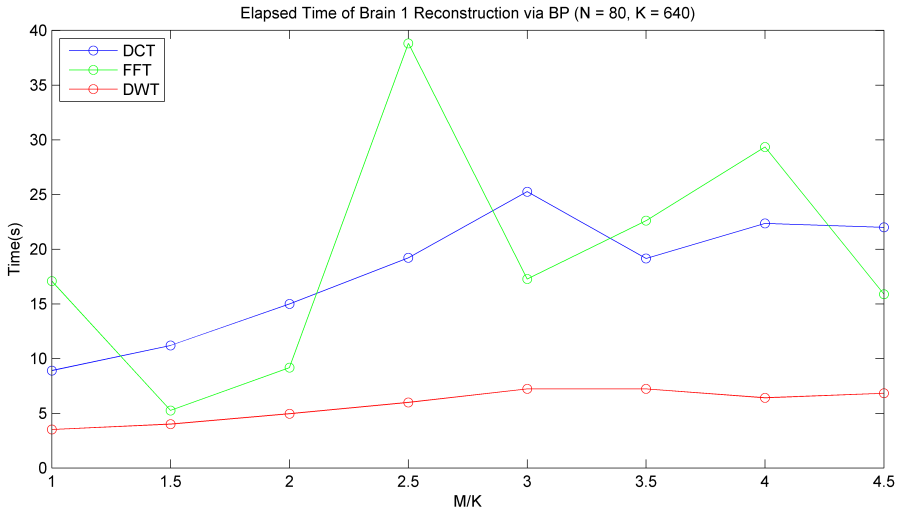


Figure 4.49: Plot of how the time to generate the results change for an increasing number of measurements for Brain 1.

4.4.3 Brain 2 - Image Reconstruction

I now run tests on image 4.19b. As before each column will represent the image produced from each transform which from left to right is: DCT, FFT, DWT.

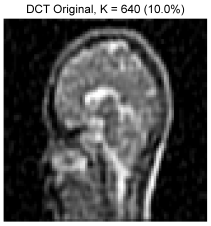


Figure 4.50: DCT: Image K = 10%

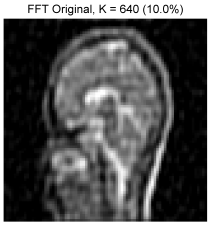


Figure 4.51: FFT: Image K = 10%

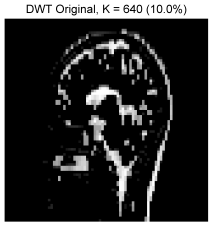


Figure 4.52: DWT: Image K = 10%

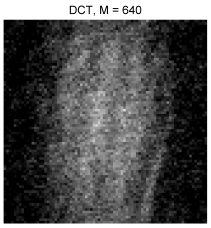


Figure 4.53: DCT: Image M = K = 640

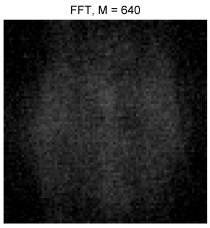


Figure 4.54: FFT: Image M = K = 640

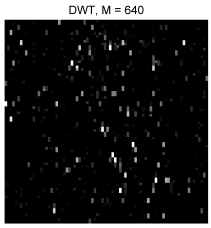


Figure 4.55: DWT: Image M = K = 640

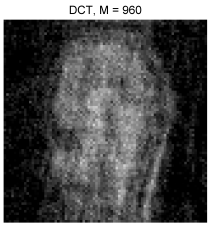


Figure 4.56: DCT: Image M = 1.5K = 960

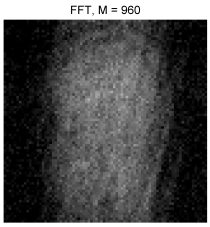


Figure 4.57: FFT: Image M = 1.5K = 960

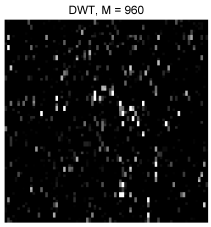


Figure 4.58: DWT: Image M = 1.5K = 960

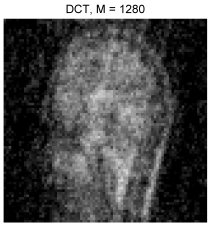


Figure 4.59: DCT: Image M = 2K = 1280

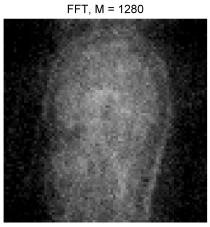


Figure 4.60: FFT: Image M = 2K = 1280

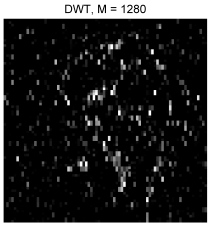


Figure 4.61: DWT: Image M = 2K = 1280

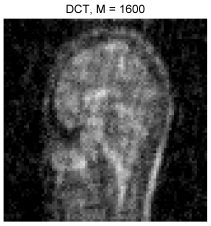


Figure 4.62: DCT: Image M = 2.5K = 1600

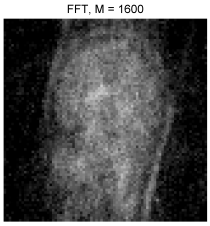


Figure 4.63: FFT: Image M = 2.5K = 1600

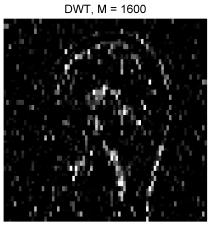


Figure 4.64: DWT: Image M = 2.K = 1600

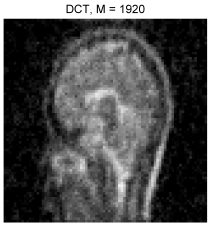


Figure 4.65: DCT: Image M = 3K = 1920

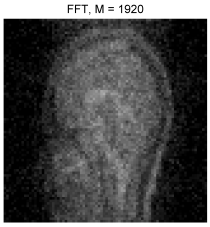


Figure 4.66: FFT: Image M = 3K = 1920

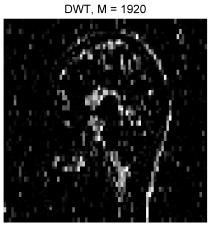


Figure 4.67: DWT: Image M = 3K = 1920

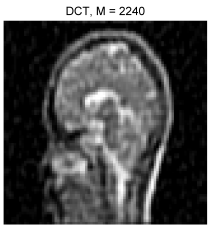


Figure 4.68: DCT: Image M = 3.5K = 2240

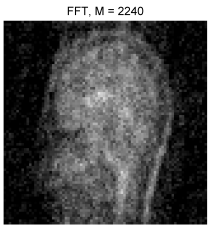


Figure 4.69: FFT: Image M = 3.5K = 2240

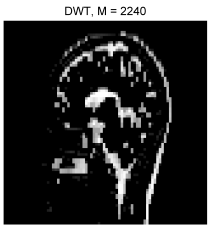


Figure 4.70: DWT: Image M = 3.5K = 2240

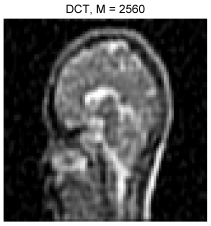


Figure 4.71: DCT: Image M = 4K = 2240

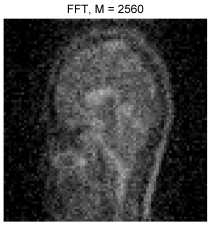


Figure 4.72: FFT: Image M = 4K = 2560

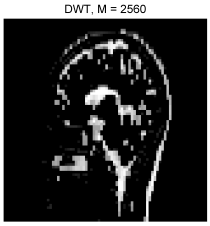


Figure 4.73: DWT: Image M = 4K = 2560

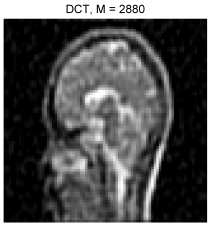


Figure 4.74: DCT: Image M = 4.5K = 2880

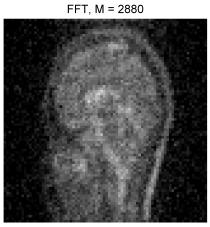


Figure 4.75: FFT: Image M = 4.5K = 2880

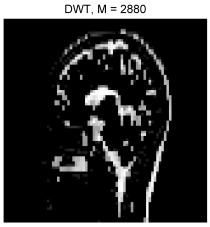


Figure 4.76: DWT: Image M = 4.5K = 2880

4.4.4 Brain 2 Image - Analysis

The graphs below show how the time, error and PSNR changed as I increased the number of measurements.

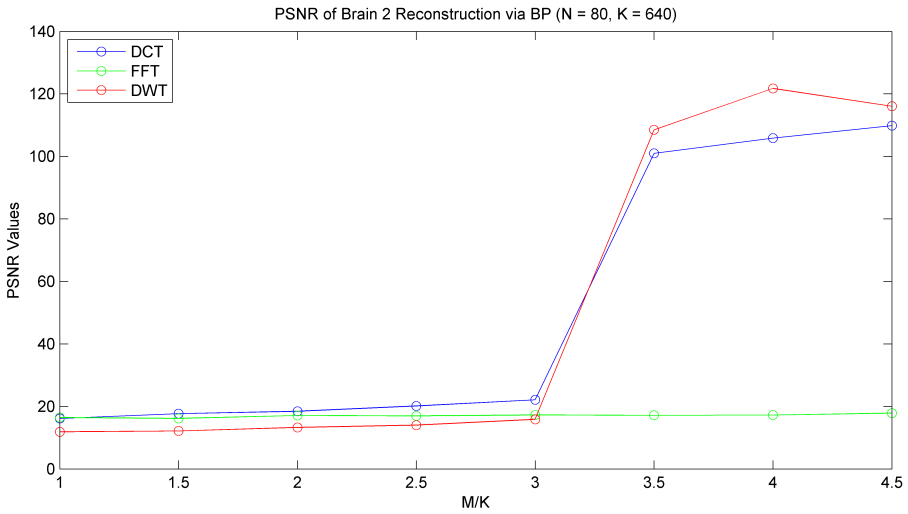


Figure 4.77: Plot of how the PSNR values change for an increasing number of measurements for Brain 2.

The second images suffers from the same problem in regards to the FFT reconstruction. However, we do note that the the image does visually become more clear as the ratio of M/K increases. Unsurprisingly, the results show the same trend as discussed in the analysis of the first brain. There may be small variance in the actual points in the graphs, but the overall trend remains the same and there is no large discernible difference.

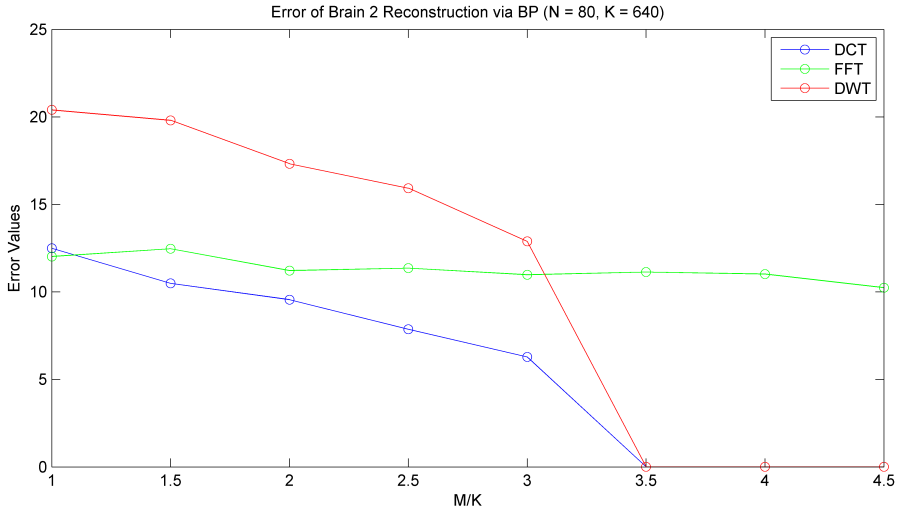


Figure 4.78: Plot of how the error values change for an increasing number of measurements for Brain 2.

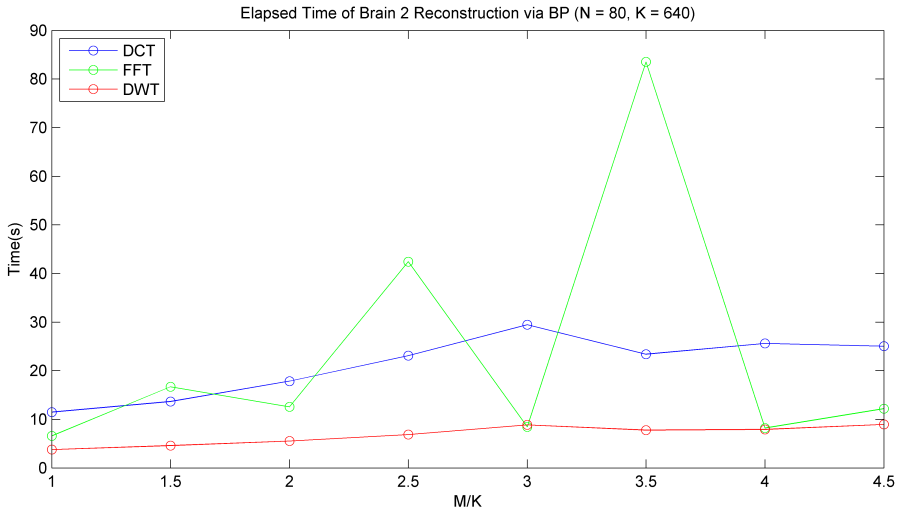


Figure 4.79: Plot of how the time to generate the results change for an increasing number of measurements for Brain 2.

4.5 Conclusion

From our analysis and the theorems described earlier, it is obvious that ℓ_1-minimization method, Basis Pursuit, has many advantages. Exact recovery of a sparse noiseless signal is possible when the condition specified in equation 4.6 is satisfied. The method is also stable meaning given that a measurement matrix which satisfies the restricted isometric property, Basis Pursuit is able to reconstruct all K-sparse signals $\mathbf{x}$,noisy or noise-less, as demonstrated by equation 4.10. As a result of this, the guarantees provided are uniform meaning there are no failures in reconstructing any K-sparse signal. As described in the introduction in section 4.2, given $M = \mathcal{O}(KLog(N/K))$ identically distributed Gaussian measurements, we can approximate a compressible signal with large probability.

Whilst Basis Pursuit provides uniform guarantees and is stable, it is based on linear programming which is known to have algorithms of polynomial run time and therefore, is slower than some of the greedier methods. However, the ability to reconstruct all sparse signals even those perturbed by noise means that the algorithm can be used as a method to solve many problems because of its applicability.

To summarize:

Advantages

- Provides uniform guarantees.
- Is a stable algorithm.

Disadvantages

- Has slow, polynomial run time.

4.6 Orthogonal Matching Pursuit

A brief introduction to how OMP works was given in section 3.3.1. Following this, we can introduce a theorem developed by Tropp and Gilbert[25] which shows that OMP can accurately reconstruct a K-sparse signal $\mathbf{x}$ from an M measurements using a $M \times N$ subgaussian measurement matrix .

Theorem: Sparse Recovery via OMP[25]
Fix $\delta \in (0, 0.36)$ *and choose* $M \geq ZKlog\left(\frac{N}{\delta}\right)$. *Suppose that* $\boldsymbol{x} \in \mathbb{R}^{N\times 1}$ *is a K-sparse signal and let* $\boldsymbol{A} \in \mathbb{R}^{M\times N}$ *be a measurement matrix with entries consisting of independent and identically distributed variables from the standard Gaussian distribution. Let* $\boldsymbol{b} \in \mathbb{R}^{M\times 1}$, *then given* $\boldsymbol{b} = \boldsymbol{Ax}$, *the OMP algorithm is able to correctly reconstruct the K-sparse signal* $\boldsymbol{x}$ *with greater than* $1 - 2\delta$ *probability. The constant Z must satisfy* $Z \leq 20$ *and for large values of K,* $Z \approx 4$.

Note, however, that this guarantee is weak as it is only for a fixed K-sparse signal $\mathbf{x}$. As a result, OMP may fail for some K-sparse signals.

The iterative steps of the algorithm can be explained as follows:

1 **Identify:** Find the largest coefficient of the residual $\mathbf{r}$ where initially, $(\mathbf{r} = \mathbf{b} = \mathbf{Ax})$, and retrieve its index and store it as λ.

2 **Augment:** Augment the initially empty index set $\boldsymbol{\mathcal{I}}$ with the newly identified index λ. Also augment the initially empty matrix $\mathbf{A}$ with the column $\mathbf{A}_\lambda$.

3 **Estimate:** Retrieve an estimate for the original signal $\mathbf{x}$ by using least squares. We want to find the projection of $\mathbf{b}$ on the range of the columns of $\mathbf{A}$.

4 **Update:** Update the residual with its new value using the estimated signal $\mathbf{x}$.

4.6.1 Algorithm

The algorithm for OMP is given as follows:

Algorithm 5 OMP Algorithm

Inputs

1: Measurement matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$.
2: Output signal vector $\mathbf{b} \in \mathbb{R}^{M \times 1}$.
3: The number of iterations (K) to perform (K = Sparsity value of original signal $\mathbf{x} \in \mathbb{R}^{N \times 1}$).

Outputs

1: A K-sparse Signal $\mathbf{x} \in \mathbb{R}^{N \times 1}$ which satisfies $\mathbf{b} = \mathbf{A}\mathbf{x}$.
2: An index set $\boldsymbol{\mathcal{I}}$ which holds the index of the non-zero values of $\mathbf{x}$.
3: An estimation of the observation vector $\mathbf{b}$ via $\mathbf{y}$.
4: The residual $\mathbf{r}$.

Steps

1: **procedure** OMP(A, b, K)
2: $\quad \mathbf{r}^{(0)} \leftarrow \mathbf{b}$
3: $\quad i \leftarrow 0$
4: $\quad \boldsymbol{\mathcal{I}}^{(0)} \leftarrow \emptyset$
5: $\quad$ **while** $i \leq K$ **do**
6: $\quad\quad i \leftarrow i + 1$
7: $\quad\quad \lambda^{(i)} \leftarrow arg \max_{j=1,\ldots,N} | < \mathbf{r}^{(i-1)}, \mathbf{A}_j > |$ ▷ Get index of largest entry
8: $\quad\quad \boldsymbol{\mathcal{I}}^{(i)} \leftarrow \boldsymbol{\mathcal{I}}^{(i-1)} \cup \lambda^{(i)}$ ▷ Augment index set
9: $\quad\quad \mathbf{A}^{(i)} \leftarrow [\mathbf{A}^{(i-1)}, \mathbf{A}_{\lambda^{(i)}}]$ ▷ Augment the measurement matrix
10: $\quad\quad \mathbf{x}^{(i)} \leftarrow arg \min_{\mathbf{x} \in \mathbb{R}^{N \times 1}} ||\mathbf{A}^{(i)}\mathbf{x} - b||_2$ ▷ Get estimate by least squares
11: $\quad\quad \mathbf{y}^{(i)} \leftarrow \mathbf{A}^{(i)}\mathbf{x}^{(i)}$ ▷ Update approx for output signal
12: $\quad\quad \mathbf{r}^{(i)} \leftarrow \mathbf{b} - \mathbf{y}^{(i)}$ ▷ Update Residual
13: $\quad$ **end while**
14: $\quad \mathbf{x} \leftarrow \mathbf{x}^{(K)}$
15: $\quad$ **return** $\mathbf{x}, \boldsymbol{\mathcal{I}}^K, \mathbf{y}^{(K)}, \mathbf{r}^{(K)}$
16: **end procedure**

4.6.2 1D Signal - OMP

The following example use my code for the implementation of Orthogonal Matching Pursuit. We begin, as before, by creating a sparse-time signal. The signal shown in figure 4.80.

The code is available in the appendix and the following graph was generated with the following parameters:

- Signal Length (N) = 256.
- Measurements (M) = 50 to 200.
- Sparsity/Peaks (K) = 20.
- rng(5) (A random number seed to replicate results)

We make use of PSNR and the 2-Norm error to measure the quality of reconstruction as before. The signal generated is given below:

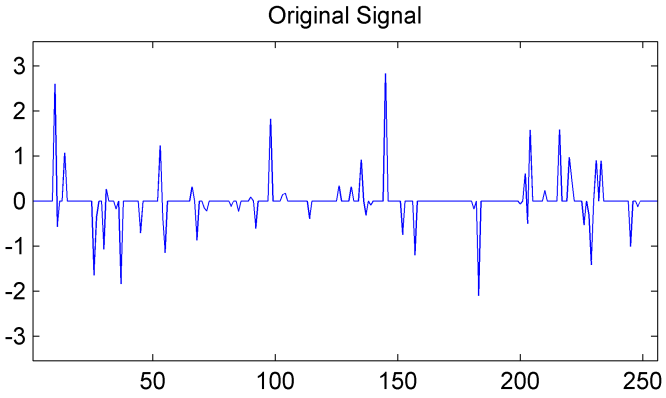


Figure 4.80: The original (N = 256) symbol generated

We then take 50 samples of this signal which appears as follows:

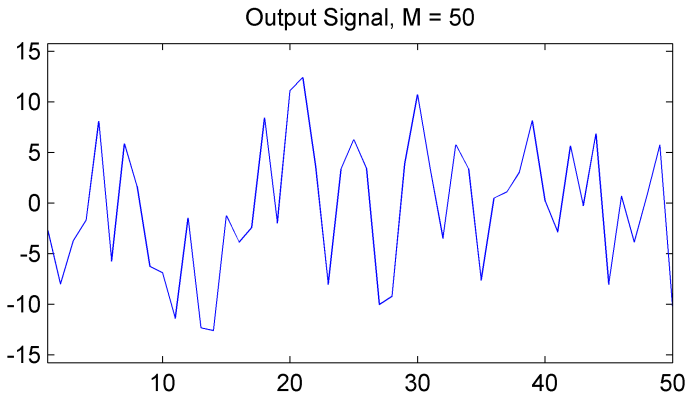


Figure 4.81: The output generated from the original signal when sampled at 20 points using a Gaussian measurement matrix $\mathbf{A}$

The reconstruction is then performed. We see a large error value and a low PSNR value. The goal is to get the PSNR above 20.

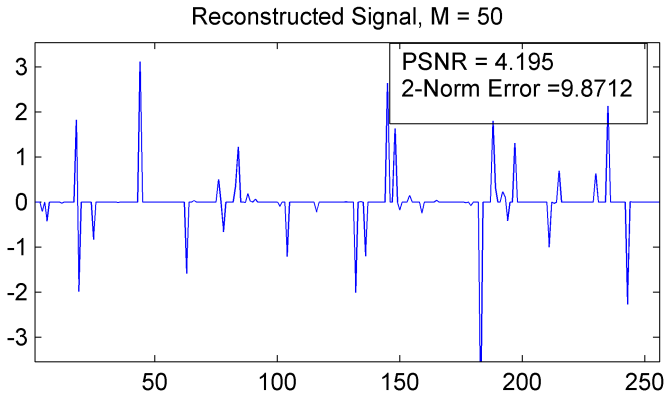


Figure 4.82: The reconstruction of the original signal $\mathbf{x}$ using 50 samples

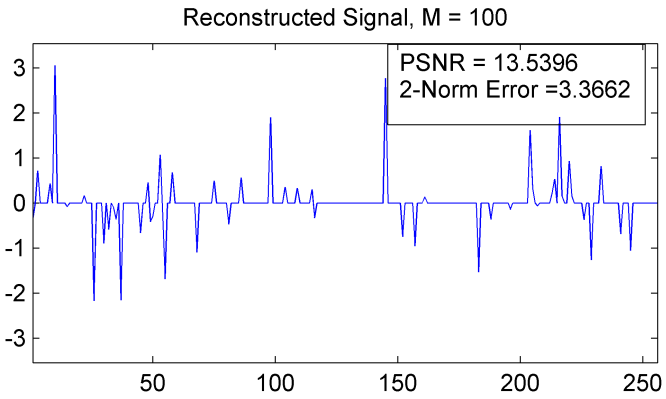


Figure 4.83: The reconstruction of the original signal $\mathbf{x}$ using 100 samples

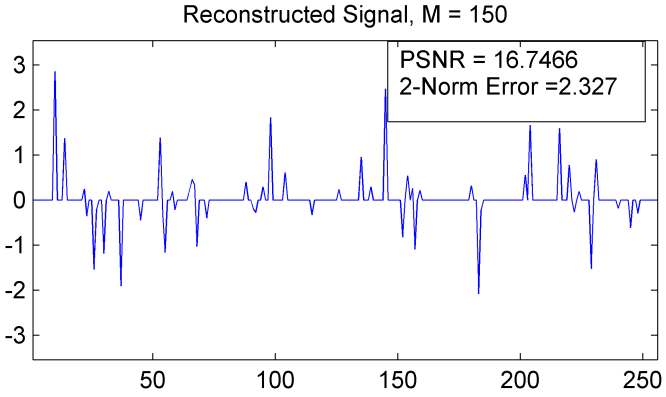


Figure 4.84: The reconstruction of the original signal $\mathbf{x}$ using 150 samples

After 150 samples, our reconstructed signal is approximately the same as the original but still has discernible differences.

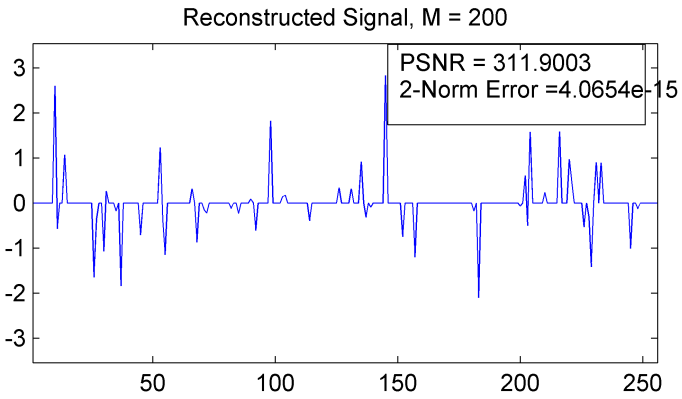


Figure 4.85: The reconstruction of the original signal $\mathbf{x}$ using 200 samples

At 200 samples however, the reconstruction is near perfect with tiny error. As we can see, the more measurements we took, the greater the accuracy of the reconstruction became. We can run this experiment many times and collect the results. I've done so and displayed them in the following section.

4.6.3 1D Signal - OMP Numerical Results

The results are displayed and calculated in the same way as shown in the section regarding the numerical results for Basis Pursuit.

Again, as the number of measurements increase so does the PSNR of the reconstructed signal. However, the PSNR of the reconstructed signal is about 2.5 times larger than those reconstructed via BP. As the sparsity of signal decreases, the PSNR of the reconstructed image also decreases but does more sharply in comparison to Basis Pursuit. At 60 samples for a sparsity of K = 15, we can see that the PSNR for the reconstruction of a is approximately 300dB. However, at 40 samples its at 100dB which is a drop of 200. The graphs show that the PSNR of the reconstruction stabilizes in a region between 300dB and 350dB. Observe also that the gradient of the PSNR values for each sparsity decreases as the signal becomes more dense (larger K value).

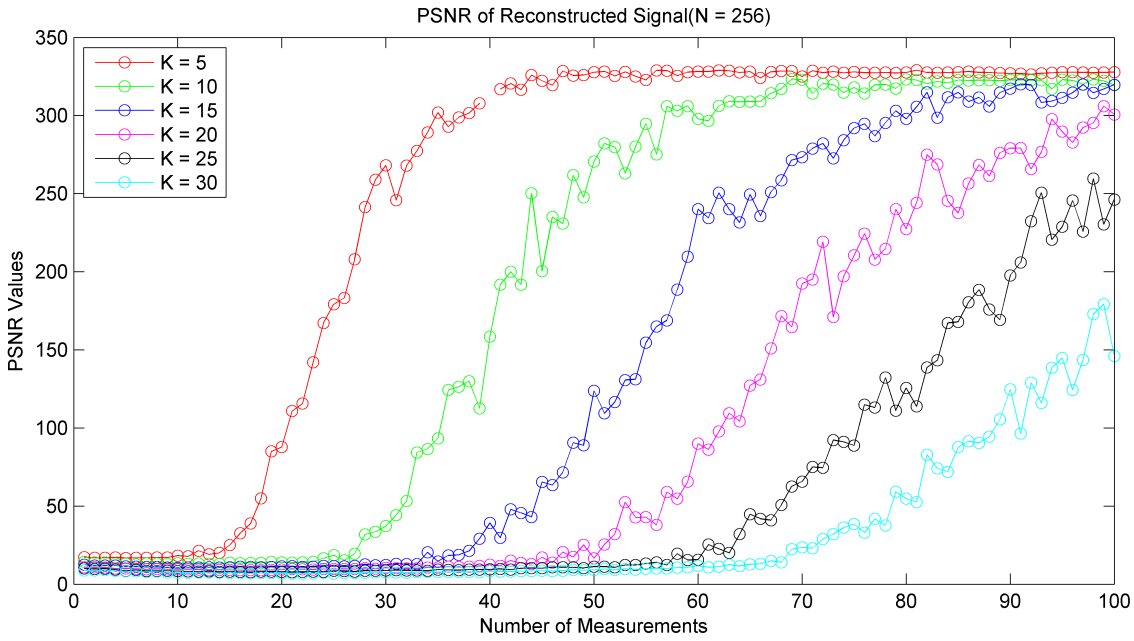


Figure 4.86: A graph detailing the change in PSNR for signals obtained from OMP with respect to sample size and sparsity

Figure 4.87 shows the relationship between the measurements M and

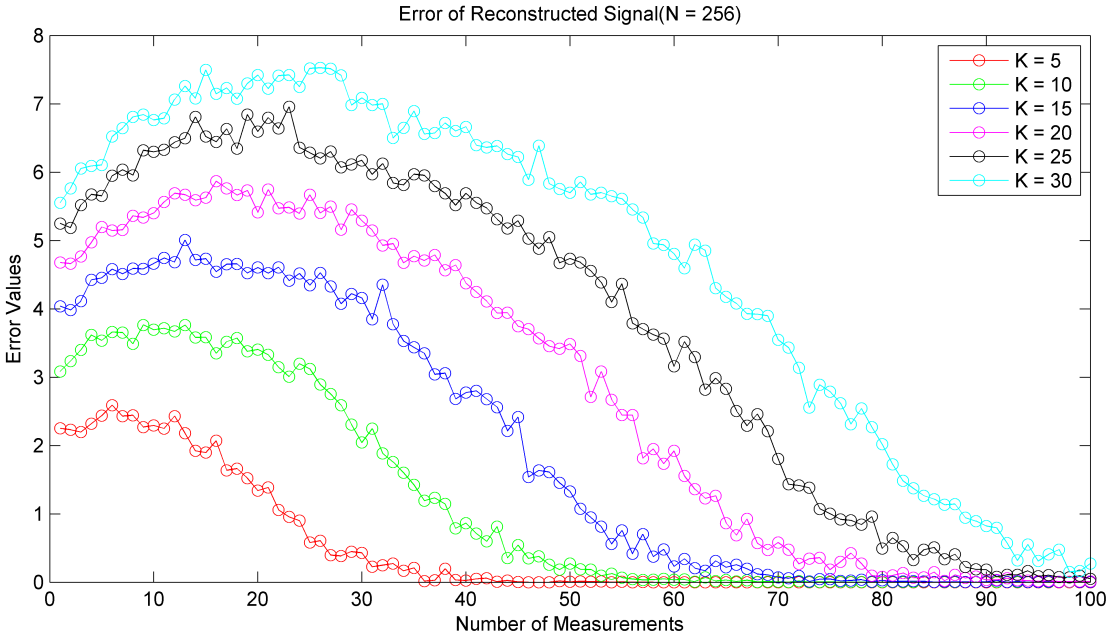


Figure 4.87: A graph detailing the change in 2-Norm error for signals obtained from OMP with respect to sample size and sparsity

the sparsity K and their effect on the 2-Norm error of the reconstructed signal. In similar fashion to BP, the error reaches a peak from which it begins decreasing. However, we also note that the error is larger in general when compared to BP and more measurements need to be taken for OMP to each a 2-Norm error of close to 0. The same trend follows: An increase in measurements coincides with the deprecation of the 2-Norm error value. The error begins to decay after a certain threshold of measurements is reached. It is also clear that the sparser the signal, the smaller the initial 2-Norm error and the faster that error decays as the number of measurements increases. As before, we can conclude that the denser the signal is, the slower 2-norm error decays to 0.

The graph in Figure 4.88 displays how many signals BP is able to correctly construct given the threshold that the support of the the reconstructed signal is the same as the support of the original signal. As before, for very sparse signals we can clearly see that the measurements required to achieve 100% successful recovery is for i.e. $K = 5$ approximately 50 measurements. We note

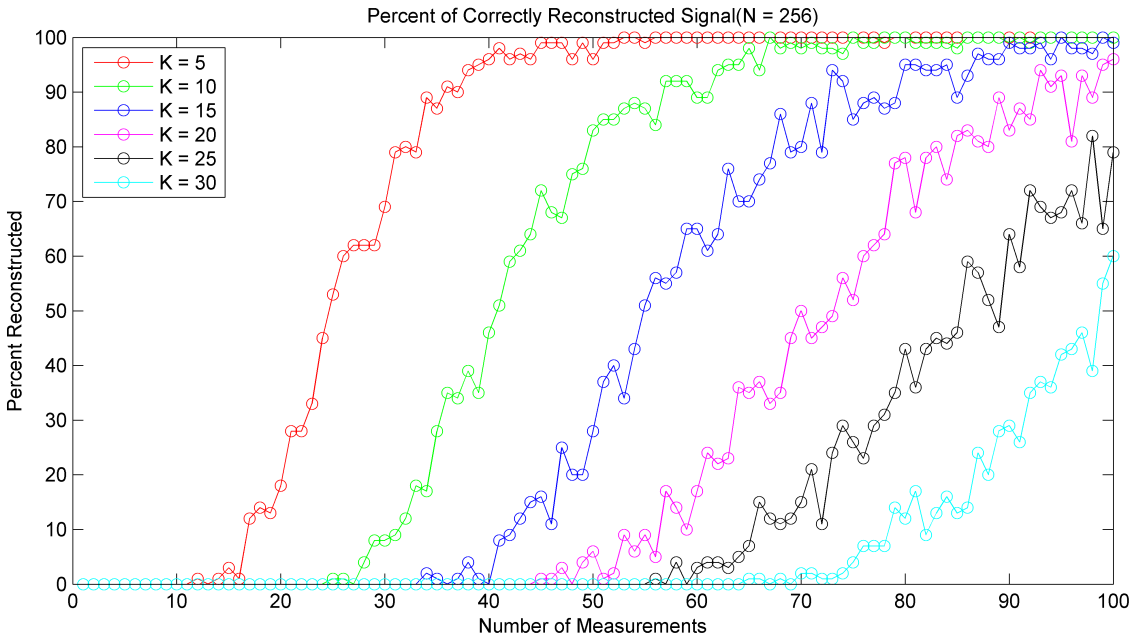


Figure 4.88: A graph detailing the percent of signals correctly reconstructed for signals obtained from OMP with respect to sample size and sparsity

that this is larger than in the case for BP. There is a larger difference when the signals become more dense. Here, the gradient of the results become less steep as the signal becomes more dense. In BP, the gradient of the results remained close to the same value regardless of the increase in non-zero values in the signal. As a result, the ratio of measurements and sparsity for 100% signal reconstruction decreases as the signal becomes more dense. Further, whilst we do not see that 100% correct reconstruction is possible for larger values of K, the graph does suggest that it is possible if the measurements were larger.

4.7 OMP - Experiments

4.7.1 Brain 1 - Image Reconstruction

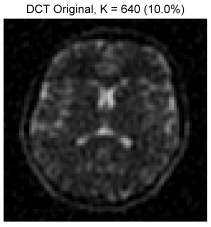


Figure 4.89: DCT: Image K = 10%

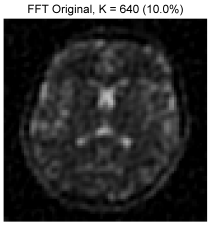


Figure 4.90: FFT: Image K = 10%

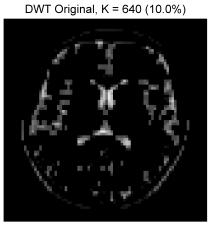


Figure 4.91: DWT: Image K = 10%

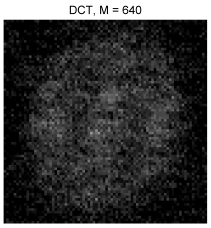


Figure 4.92: DCT: Image M = K = 640

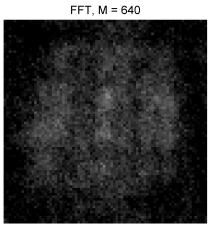


Figure 4.93: FFT: Image M = K = 640

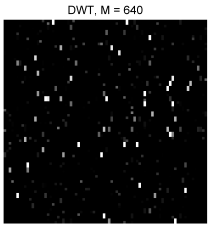


Figure 4.94: DWT: Image M = K = 640

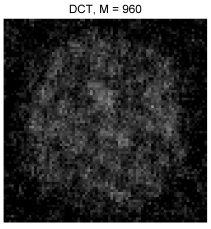


Figure 4.95: DCT: Image M = 1.5K = 960

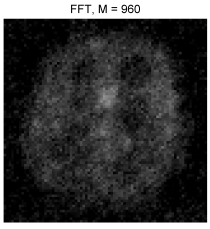


Figure 4.96: FFT: Image M = 1.5K = 960

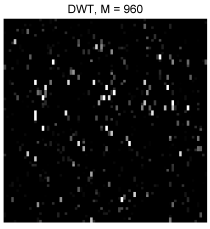


Figure 4.97: DWT: Image M = 1.5K = 960

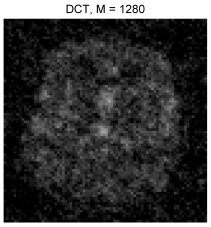


Figure 4.98: DCT: Image M = 2K = 1280

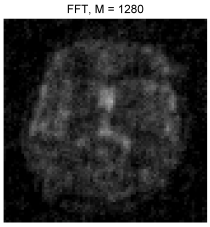


Figure 4.99: FFT: Image M = 2K = 1280

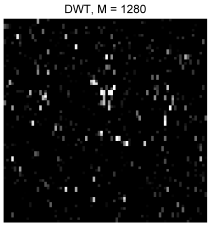


Figure 4.100: DWT: Image M = 2K = 1280

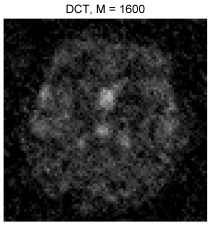


Figure 4.101: DCT: Image M = 2.5K = 1600

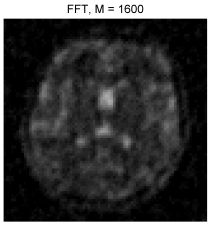


Figure 4.102: FFT: Image M = 2.5K = 1600

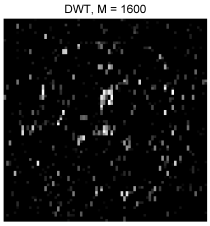


Figure 4.103: DWT: Image M = 2.K = 1600

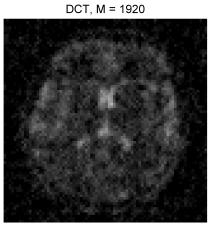


Figure 4.104: DCT: Image M = 3K = 1920

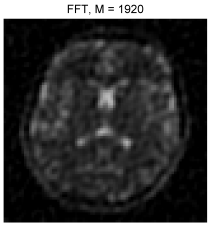


Figure 4.105: FFT: Image M = 3K = 1920

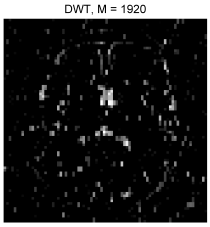


Figure 4.106: DWT: Image M = 3K = 1920

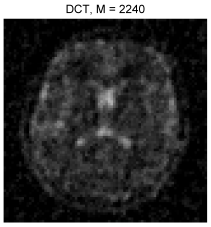


Figure 4.107: DCT: Image M = 3.5K = 2240

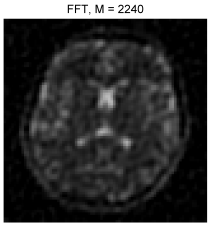


Figure 4.108: FFT: Image M = 3.5K = 2240

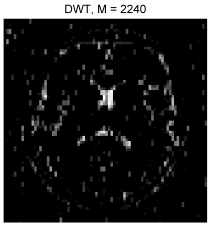


Figure 4.109: DWT: Image M = 3.5K = 2240

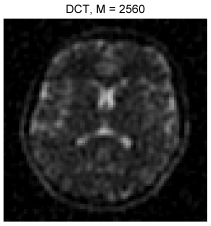


Figure 4.110: DCT: Image M = 4K = 2240

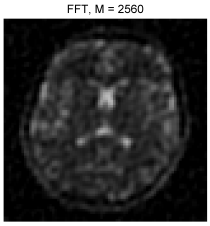


Figure 4.111: FFT: Image M = 4K = 2560

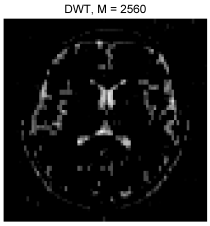


Figure 4.112: DWT: Image M = 4K = 2560

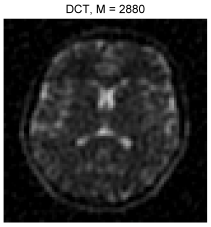


Figure 4.113: DCT: Image M = 4.5K = 2880

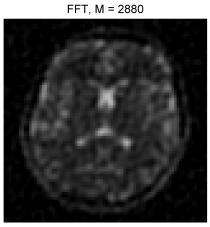


Figure 4.114: FFT: Image M = 4.5K = 2880

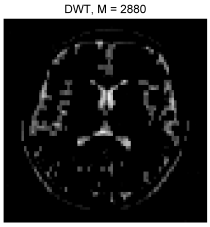


Figure 4.115: DWT: Image M = 4.5K = 2880

4.7.2 Brain 1 Image - Analysis

Following the results, I generated some graphs showing how the time, error and PSNR changed as I increased the number of measurements. These are shown below. The x-axis is labelled so that the measurements in each image is displayed as a multiple of the sparsity K of the original image.

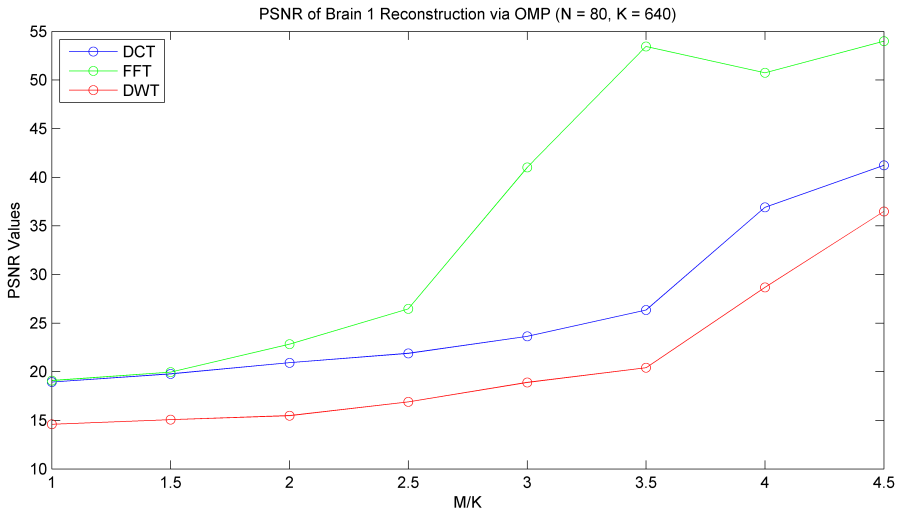


Figure 4.116: Plot of how the PSNR values change for an increasing number of measurements for Brain 1.

With OMP we are able to analyse the FFT results. Comparatively to the BP results, there the "jump" in PSNR is not as large when M/K = 3.5. We also observe a steady rise in the PSNR of the reconstruction prior to this M/K value similar to that of BP.

The error in reconstruction behaves similar to that of the graph produced via BP. The DWT transform begins with the largest error however, FFT and DCT begin with the same error. The FFT transform seems to reduce the error at a faster rate comparative to DCT.

The time taken to perform the algorithm on each transform is shown below with the wavelet transform being the fastest again. My implementation of OMP recovers the DCT and DWT reconstruction slower than BP. As expected the time taken increases as M/K increases. Working out the gradients of the speed of reconstruction, we see that the reconstruction via DCT takes almost 5 seconds per M = 0.5K.

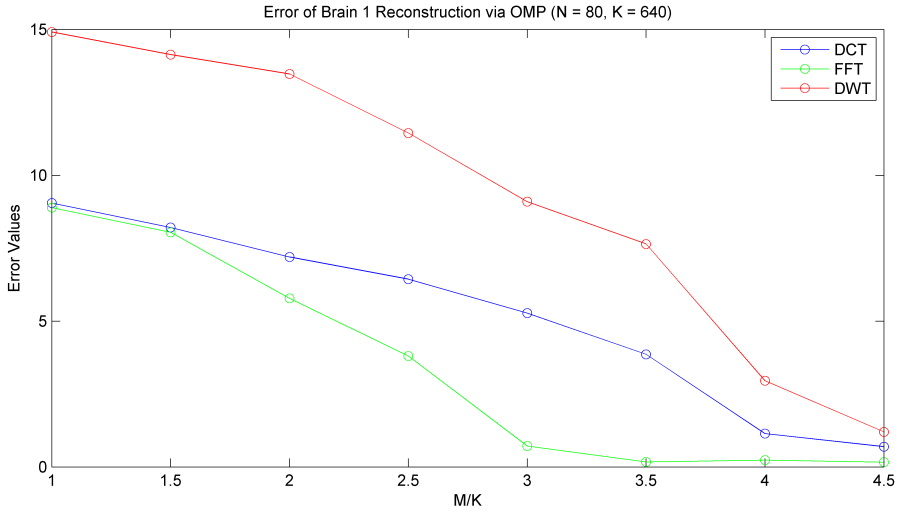


Figure 4.117: Plot of how the error valueschange for an increasing number of measurements for Brain 1.

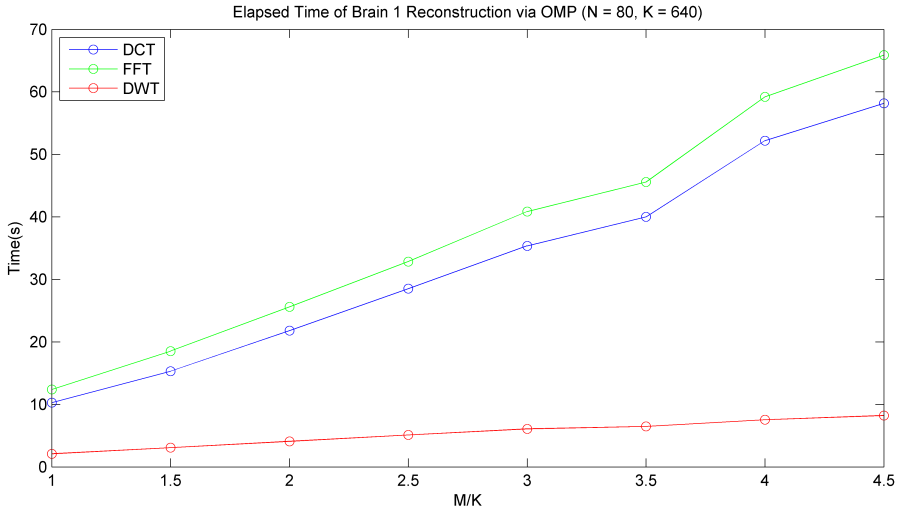


Figure 4.118: Plot of how the time to generate the results change for an increasing number of measurements for Brain 1.

4.7.3 Brain 2 - Image Reconstruction

I now run tests on image 4.19b. As before each column will represent the image produced from each transform which from left to right is: DCT, FFT, DWT.

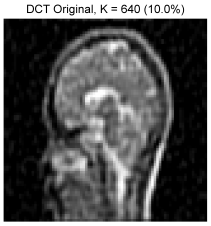


Figure 4.119: DCT: Image K = 10%

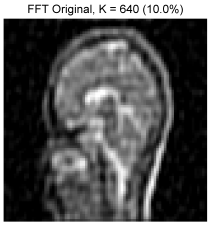


Figure 4.120: FFT: Image K = 10%

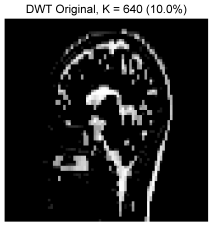


Figure 4.121: DWT: Image K = 10%

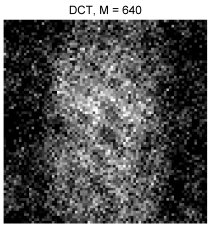


Figure 4.122: DCT: Image M = K = 640

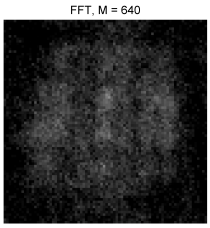


Figure 4.123: FFT: Image M = K = 640

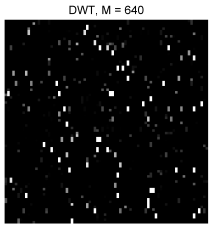


Figure 4.124: DWT: Image M = K = 640

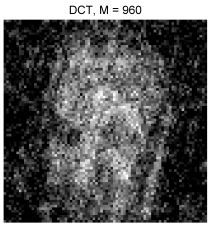


Figure 4.125: DCT: Image M = 1.5K = 960

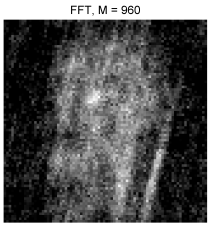


Figure 4.126: FFT: Image M = 1.5K = 960

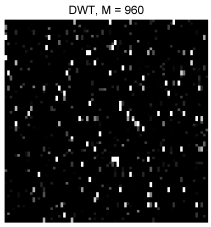


Figure 4.127: DWT: Image M = 1.5K = 960

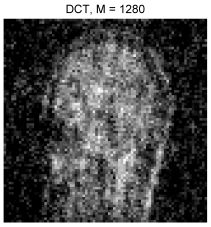


Figure 4.128: DCT: Image M = 2K = 1280

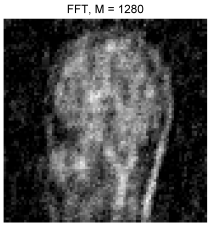


Figure 4.129: FFT: Image M = 2K = 1280

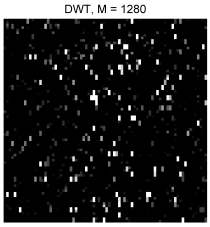


Figure 4.130: DWT: Image M = 2K = 1280

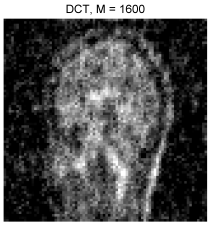


Figure 4.131: DCT: Image M = 2.5K = 1600

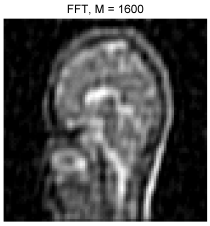


Figure 4.132: FFT: Image M = 2.5K = 1600

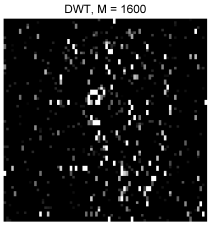


Figure 4.133: DWT: Image M = 2.K = 1600

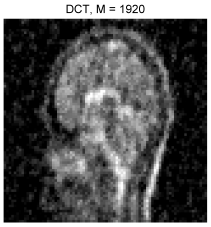


Figure 4.134: DCT: Image M = 3K = 1920

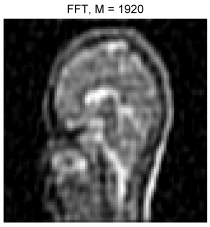


Figure 4.135: FFT: Image M = 3K = 1920

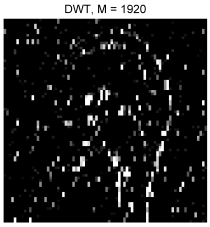


Figure 4.136: DWT: Image M = 3K = 1920

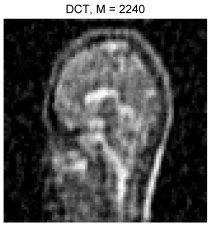


Figure 4.137: DCT: Image M = 3.5K = 2240

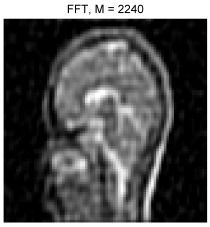


Figure 4.138: FFT: Image M = 3.5K = 2240

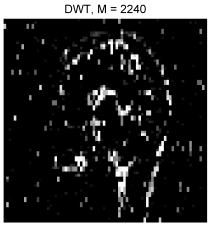


Figure 4.139: DWT: Image M = 3.5K = 2240

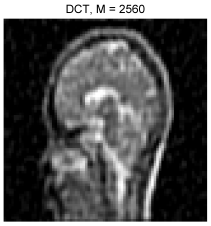


Figure 4.140: DCT: Image M = 4K = 2240

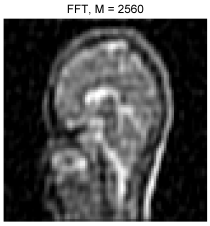


Figure 4.141: FFT: Image M = 4K = 2560

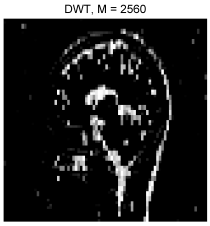


Figure 4.142: DWT: Image M = 4K = 2560

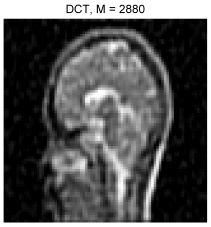


Figure 4.143: DCT: Image M = 4.5K = 2880

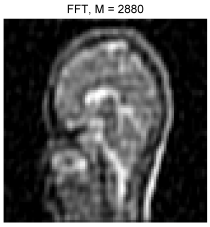


Figure 4.144: FFT: Image M = 4.5K = 2880

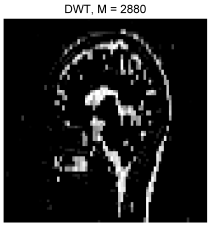


Figure 4.145: DWT: Image M = 4.5K = 2880

4.7.4 Brain 2 Image - Analysis

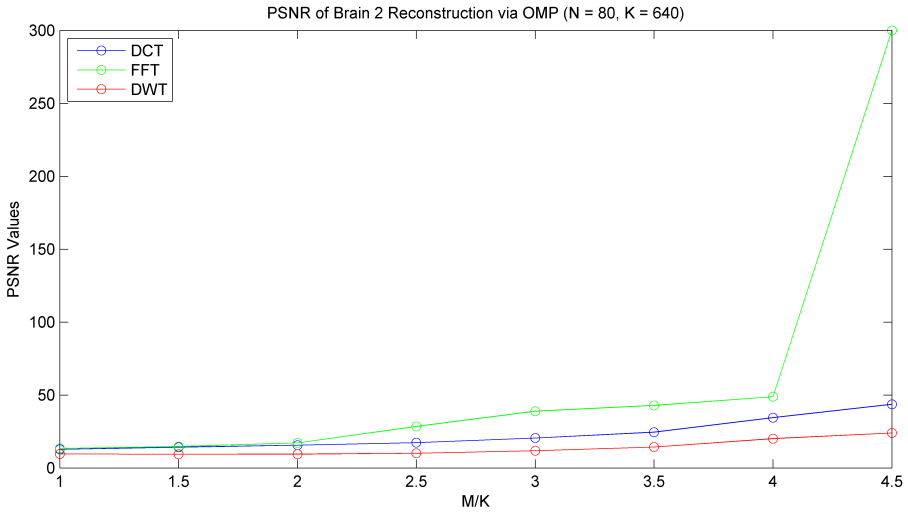


Figure 4.146: Plot of how the PSNR values change for an increasing number of measurements for Brain 2.

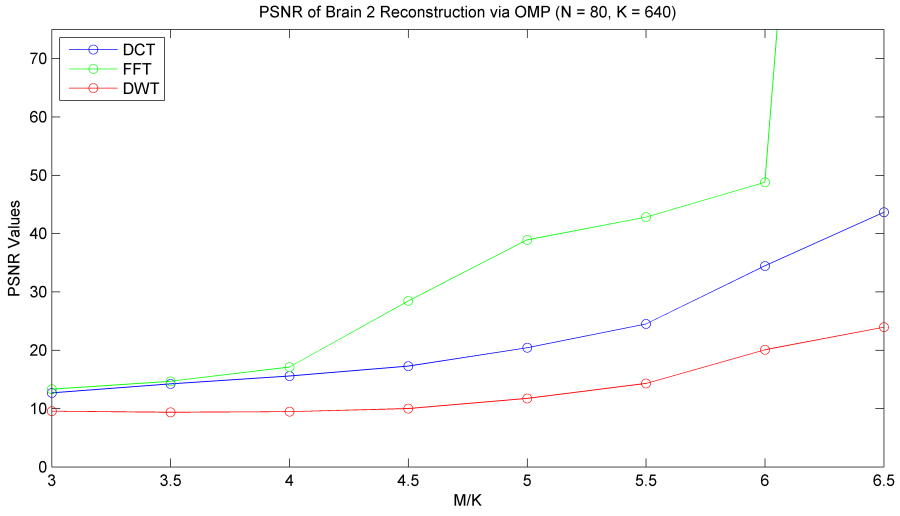


Figure 4.147: A closer look at the above plot

Again, the results show the same trend as discussed in the analysis of the first brain. There may be small variance in the actual points in the graphs, but the overall trend remains the same. There is one large discernible difference which is the sharp rise in PSNR for FFT reconstruction at the M/K value of 4.5. For this reason, I've included a second PSNR graph with a limit set on the y-axis.

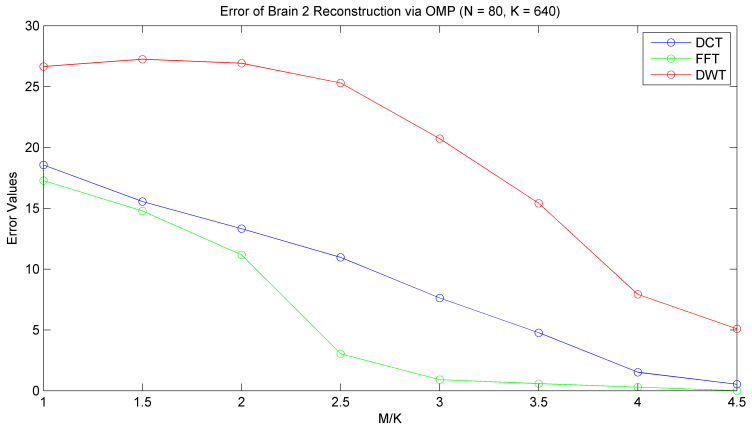


Figure 4.148: Plot of how the error values change for an increasing number of measurements for Brain 2.

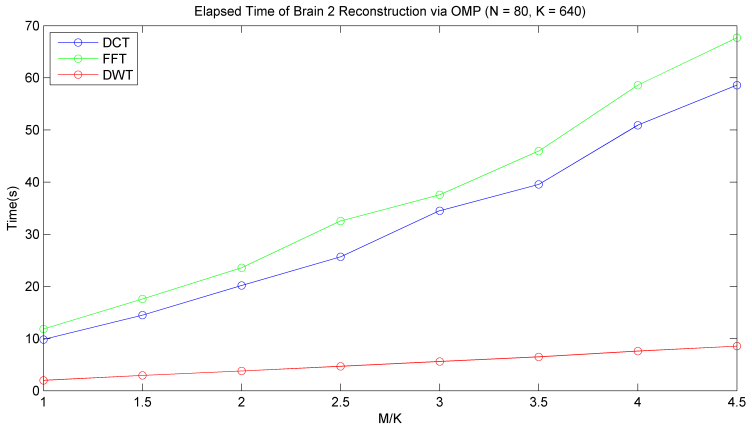


Figure 4.149: Plot of how the time to generate the results change for an increasing number of measurements for Brain 2.

4.8 Conclusion

In our numerical experiments, we observed that Orthogonal Matching Pursuit was slow in its reconstruction of our MRI images. The reason for this is that my implementation required K iterations which for my 80×80 2-D signal reshaped to a 6400×1 1-D signal had a value of 640. As a result, my algorithm iterates 640 times! However, in general, OMP is a fast algorithm which has strongly polynomial run time and is faster than Basis Pursuit. However, OMP is more restrictive on the set of measurement matrices $\mathbf{A}$ which are admissible namely subguassian matrices as defined by Tropp and Gilbert [25]. This is less than those utilized by CoSaMP (introduced next) which are the matrices which satisfy the restricted isometric property.

Further, the guarantees of OMP are non-uniform. Theorem 4.6 states that OMP reconstructs a signal correctly with large probability for a *fixed* sparsity and therefore OMP fails for certain sparse signals. Another benefit of OMP is its ease of implementation. Whilst I used ℓ_1-Magic for my Basis Pursuit algorithm, I was able to easily code the algorithm presented in my project.

To summarize:

Advantages

- Faster than Basis Pursuit
- Easy to implement

Disadvantages

- Has non-uniform guarantees
- More restrictive on acceptable measurement matrices
- Not known to be stable

4.9 Compressive Sampling Matching Pursuit

In section 3.3.3, we introduced the basic premise of how CoSaMP works and how it differs from ROMP. This section will produce an algorithm for CoSaMP as well as numerical experiments involving the algorithm. Needell and Tropp developed the following theorem which shows that accurate reconstruction of sparse signals is guranteed as is an approximation of a noisy signal[9].

Theorem: Sparse Recovery via CoSaMP
Let $\boldsymbol{x} \in \mathbb{R}^{N\times 1}$ be the signal we wish to reconstruct give the noisy observation $\boldsymbol{b} \in \mathbb{R}^{M\times 1}$ where $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{\eta}$ and $\boldsymbol{\eta} \in \mathbb{R}^{M\times 1}$. Let $\boldsymbol{A} \in \mathbb{R}^{M\times N}$ be a measurement matrix which satisfies the restricted isometry property and has constant $\delta_{2K} \leq c$. The the algorithm approximates a K-sparse solution $\boldsymbol{z}$ satisfying:

$$||\boldsymbol{x} - \boldsymbol{z}||_2 \leq C \cdot \max\left\{\rho, \frac{1}{\sqrt{K}}||\boldsymbol{x} - \boldsymbol{x}_{K/2}||_1 + ||\boldsymbol{\eta}||_2\right\}. \tag{4.14}$$

where ρ is some given precision and C is some positive constant. Also note that $\boldsymbol{x}_{K/2}$ is a $(K/2)$-sparse approximation to $\boldsymbol{x}$

As a result, a signal without any additive noise, is guaranteed to be recovered with high accuracy. The algorithm requires halting criteria which can be found in the appendix in a paper by Needell and Topp[9]

The iterative steps of the algorithm can be explained as follows:

1 **Identify:** Form a proxy signal from the current sample's residual and find its largest 2K components.

2 **Augment:** Augment the initially empty index set $\boldsymbol{\mathcal{I}}$ with the newly identified index of the proxy's largest components Ω.

3 **Estimate:** Retrieve an estimate for the original signal $\mathbf{x}$ by using least squares on the augmented set.

5 **Prune:** Keep only the K largest coefficients of the least squares solution.

5 **Update:** Update the residual with its new value using the estimated signal $\mathbf{x}$.

4.9.1 Algorithm

Algorithm 6 CoSaMP Algorithm

Inputs

1: Measurement matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$.
2: Output signal vector $\mathbf{b} \in \mathbb{R}^{M \times 1}$.
3: The number of iterations (K) to perform (K = Sparsity value of original signal $\mathbf{x} \in \mathbb{R}^{N \times 1}$).

Outputs

1: A K-sparse Signal $\mathbf{x} \in \mathbb{R}^{N \times 1}$ which satisfies $\mathbf{b} = \mathbf{A}\mathbf{x}$.
2: An index set $\boldsymbol{\mathcal{I}}$ which holds the index of the non-zero values of $\mathbf{x}$.
3: The residual $\mathbf{r}$.

Steps

1: **procedure** CoSaMP(A, b, K)
2: $\quad \mathbf{x}^{(0)} \leftarrow 0$
3: $\quad \mathbf{r}^{(0)} \leftarrow \mathbf{b}$
4: $\quad i \leftarrow 0$
5: $\quad \boldsymbol{\mathcal{I}}^{(0)} \leftarrow \emptyset$
6: $\quad$ **while** Halting Criterion False **do**
7: $\quad\quad i \leftarrow i + 1$
8: $\quad\quad \mathbf{v} \leftarrow \mathbf{A}^T\mathbf{r}$ ▷ Get Proxy
9: $\quad\quad \boldsymbol{\Omega} \leftarrow supp(\mathbf{v}^{2K})$ ▷ Get 2K largest coefficients
10: $\quad\quad \boldsymbol{\mathcal{I}}^{(i)} \leftarrow \boldsymbol{\Omega} \cup supp(\mathbf{x}^{(i-1)})$ ▷ Augment index set
11: $\quad\quad \mathbf{s}|_{\boldsymbol{\mathcal{I}}} \leftarrow \mathbf{A}_{\boldsymbol{\mathcal{I}}}^{\dagger}\mathbf{b}; \quad \mathbf{s}|_{\boldsymbol{\mathcal{I}}^C} \leftarrow \mathbf{0}$ ▷ Get estimate by least squares
12: $\quad\quad \mathbf{x}^{(i)} \leftarrow \mathbf{s}^{(K)}$ ▷ Prune for K Coefficients
13: $\quad\quad \mathbf{r} \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}^{(i)}$ ▷ Update Residual
14: $\quad$ **end while**
15: $\quad \mathbf{x} \leftarrow \mathbf{x}^{(K)}$
16: $\quad$ **return** $\mathbf{x}, \boldsymbol{\mathcal{I}}^K, \mathbf{r}^{(K)}$
17: **end procedure**

4.9.2 1D Signal - CoSaMP

The following example use my code for the implementation of CoSaMP. We begin by creating a sparse time signal as we have done before. The signal is shown in figure 4.150.

The code is available in the appendix and the following graphs were generated with the following parameters:

- Signal Length (N) = 256.
- Measurements (M) = 50 to 200.
- Sparsity/Peaks (K) = 20.
- Max Iterations = 50.
- rng(5) (A random number seed to replicate results)

We make use of PSNR and the 2-Norm error to measure the quality of reconstruction as before. The signal generated is given below:

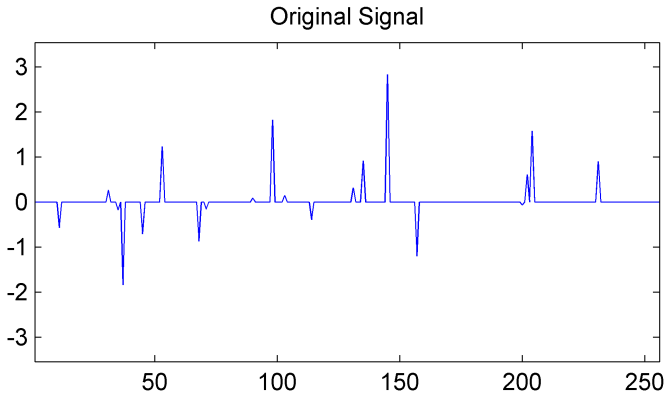


Figure 4.150: The original (N = 256) symbol generated

We then take 50 samples of this signal which appears as follows:

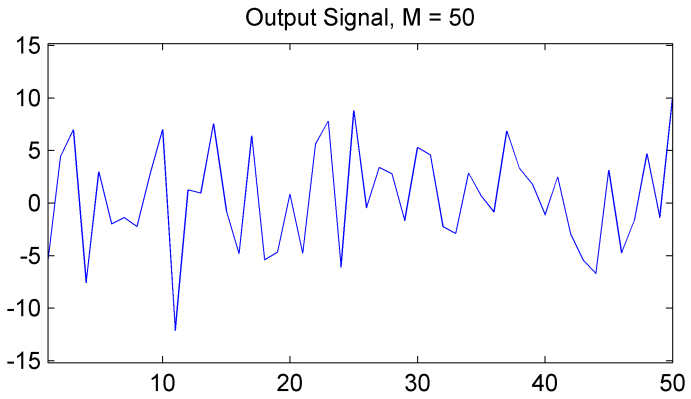


Figure 4.151: The output generated from the original signal when sampled at 50 points using a Gaussian measurement matrix $\mathbf{A}$

The reconstruction is then performed and we see that the error is large for small measurements and the PSNR very small. Visually, the signal looks very different to the original.

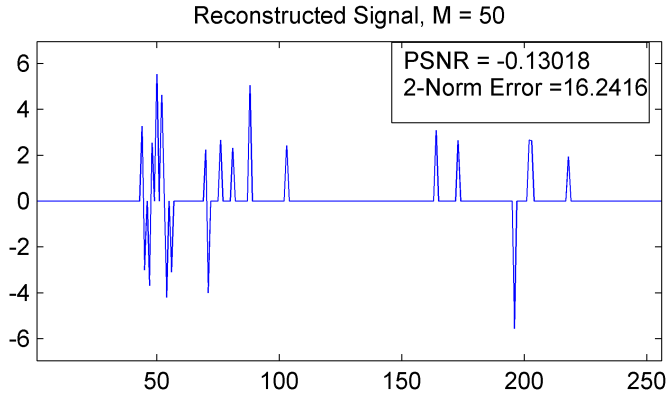


Figure 4.152: The reconstruction of the original signal $\mathbf{x}$ using 50 samples

After 100 measurements, the signal has similar peaks but their amplitudes are too large.

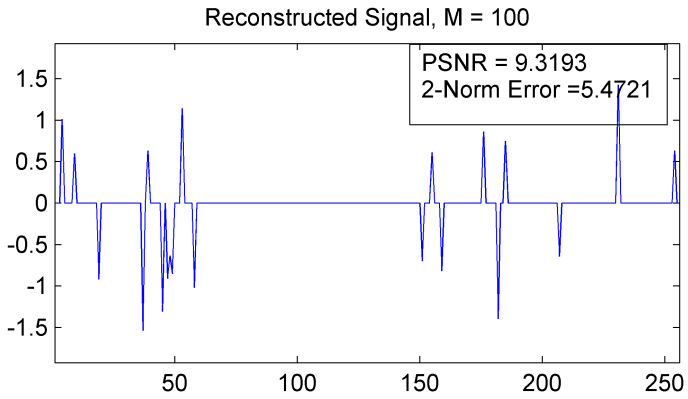


Figure 4.153: The reconstruction of the original signal $\mathbf{x}$ using 100 samples

After 150 measurements the signal looks like a better approximation to the original, but there are still clear visual differences.

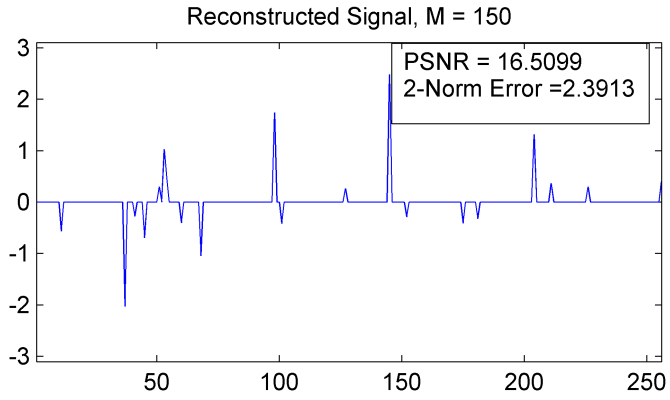


Figure 4.154: The reconstruction of the original signal $\mathbf{x}$ using 150 samples

At 200 measurements, we have a reconstruction of the original signal with a PSNR above 20. The signal still has discernible differences, but it is a better approximation to the signal when compared to the previous at M = 100.

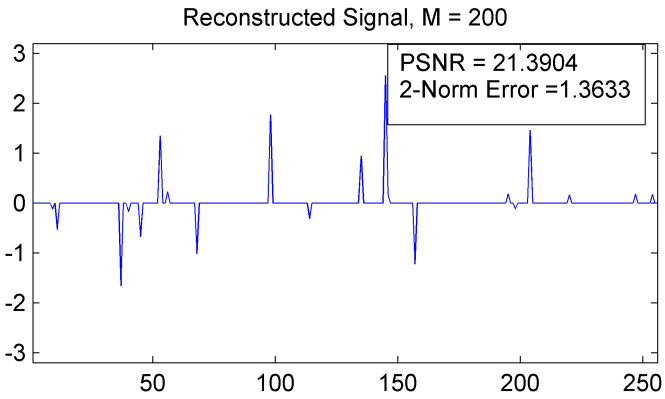


Figure 4.155: The reconstruction of the original signal $\mathbf{x}$ using 200 samples q

Again, improving the measurements increases the accuracy. In comparison to BP and OMP however, CoSaMP required more measurements for a PSNR value of above 10.

4.9.3 1D Signal - CoSaMP Numerical Results

The results are displayed and calculated in the same way as demonstrated in the numerical results for Basis Pursuit and OMP.

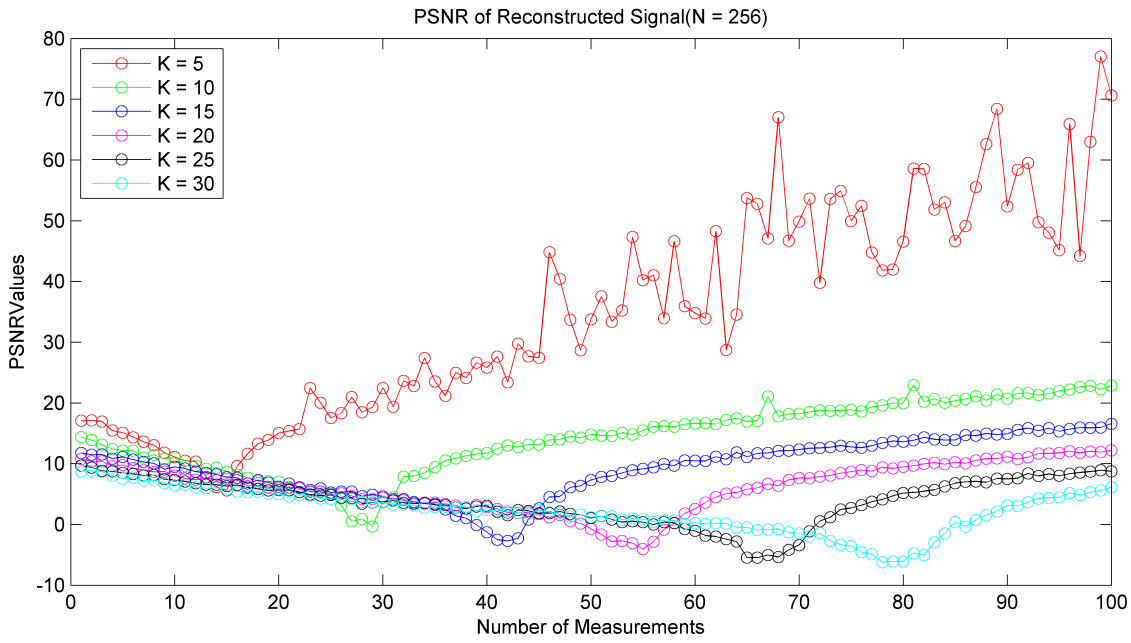


Figure 4.156: A graph detailing the change in PSNR for signals obtained from CoSaMP with respect to sample size and sparsity

The first thing we notice is there exists a measurement value for each sparsity K that contains the minimum PSNR value for that sparsity. This minimum is defined at a measurement point less than M = 3K. After this, the PSNR of the reconstructed signals begins to increase, however, in comparison to the previous results shown by OMP and BP, the PSNR for each sparsity value measured and each measurement, is smaller in magnitude. Whilst it is clear that after M = 3K the PSNR of the signal increases, the rate at which the PSNR increases given increasing number of measurements is small. As the sparsity of signal decreases, the PSNR of the reconstructed image also decreases. As observed in the BP's case, the gradient of the gradient of the PSNR values for each sparsity remains the same almost as the signal becomes more dense (large K value). This shows growth rate of the PSNR

value remains the same regardless of sparsity as long as the condition of the number of measurements are above M = 3K.

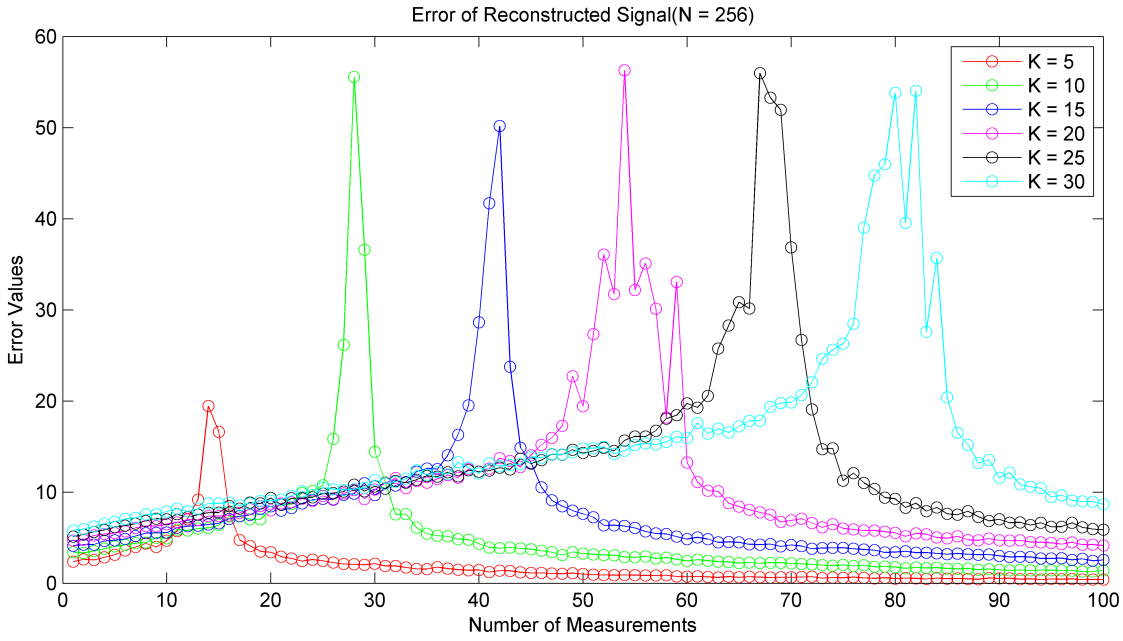


Figure 4.157: A graph detailing the percent of signals correctly reconstructed for signals obtained from CoSaMP with respect to sample size and sparsity

Figure 4.157 shows the relationship between the measurements M and the sparsity K and their effect on the 2-Norm error of the reconstructed signal. Again, we can clearly see the that beyond a measurement point under M = 3K, the error decreases. We can see that the error for measurements below M = 3K are large and it is only past this point where the errors decrease in a similar fashion to the results displayed when analysing OMP and BP. We see that for K = 20 and M = 80, the error slowly decays from approximately 14 to 5 in 20 additional measurements (i.e. taking 10% more samples). Along these points, we see that the gradient of the curve becomes smaller as we increase the measurements meaning each extra measurement means a smaller decrease in error than the last. We only see the sparsity get close to 0 at K = 5 and after over approximately 70 measurements.

The graph in Figure 4.158 displays how many signals are correctly reconstructed using CoSaMP. Measuring correct reconstruction as having an error norm of less than 0.01 would display mostly zeros on the graph and so i decided to add an "OR" condition which states that the PSNR value must be over 10dB. Unsurprisingly, at M = 3K, the % of "Correct signals" reconstructed increases. Prior to this, the "OR" condition means that any signal constructed priori to this point has a PSNR value of above 10dB as we can see that error clearly was not below 0.01 using figure 4.157. For this reason, we can regard these points as having 0% recovery rate even though it is not displayed as such. After the special point of M = 3K, we see that the curves of the graph are similar to those found when exploring the percent of correctly reconstructed signals of OMP and BP. The same observations can be made here where sparse signals require less measurements to achieve a PSNR of above 10dB. However, the gradients of the sparsity curves decrease sharply in comparison to OMP and BP and we see that for K = 30, the percent correctly reconstructed is still close to 0 if not 0. However, the graph suggests that an increasing number of measurements will surely increase the percentage chance of correct reconstruction.

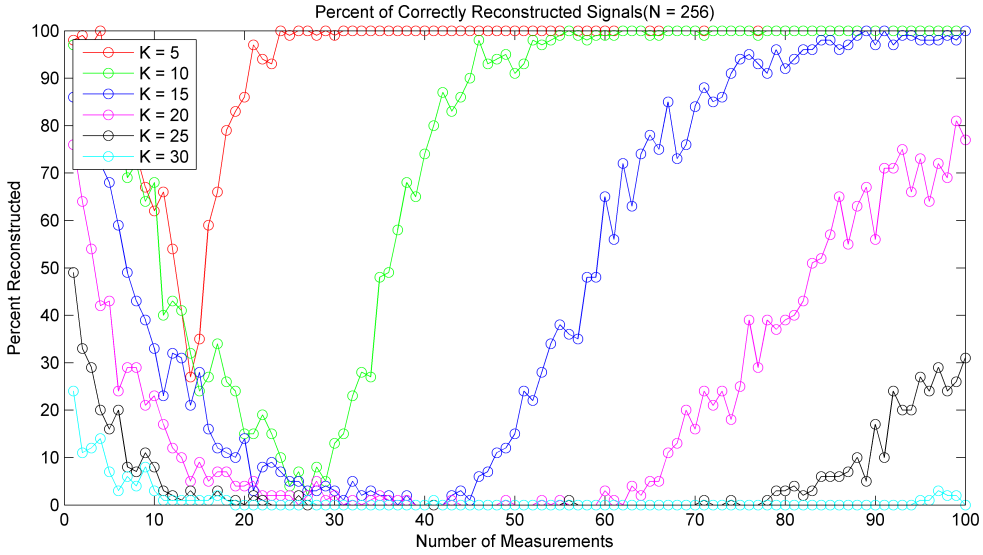


Figure 4.158: A graph detailing the percent of signals correctly reconstructed for signals obtained from CoSaMP with respect to sample size and sparsity

4.10 CoSaMP - Experiments

The images displayed here differ to the case of OMP and BP as we start with M = 3K measurements to produce good quality reconstruction.

4.10.1 Brain 1 - Image Reconstruction

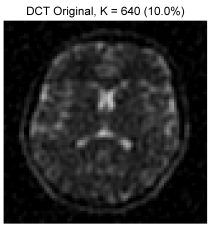


Figure 4.159: DCT: Image K = 10%

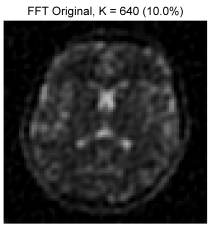


Figure 4.160: FFT: Image K = 10%

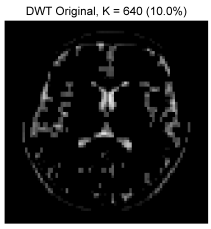


Figure 4.161: DWT: Image K = 10%

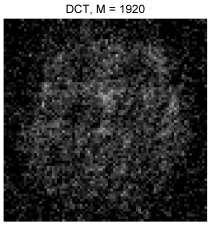


Figure 4.162: DCT: Image M = 3K = 1920

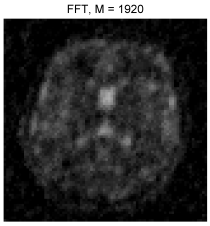


Figure 4.163: FFT: Image M = 3K = 1920

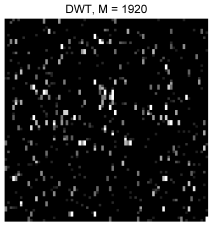


Figure 4.164: DWT: Image M = 3K = 1920

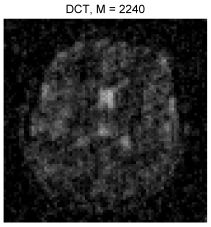


Figure 4.165: DCT: Image M = 3.5K = 2240

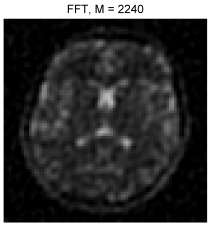


Figure 4.166: FFT: Image M = 3.5K = 2240

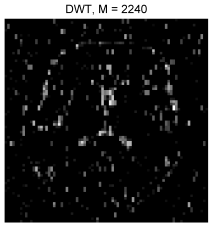


Figure 4.167: DWT: Image M = 3.5K = 2240

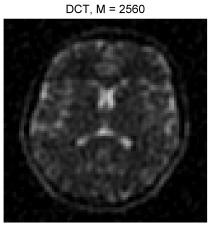


Figure 4.168: DCT: Image M = 4K = 2240

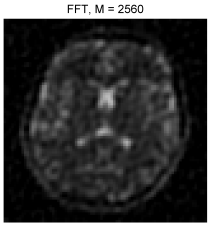


Figure 4.169: FFT: Image M = 4K = 2560

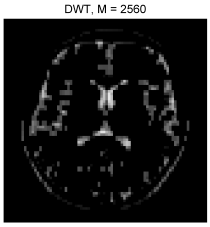


Figure 4.170: DWT: Image M = 4K = 2560

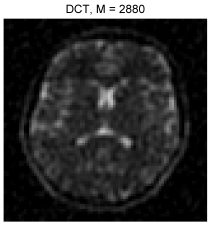


Figure 4.171: DCT: Image M = 4.5K = 2880

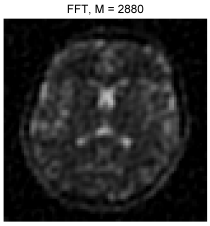


Figure 4.172: DCT: Image M = 4.5K = 2880

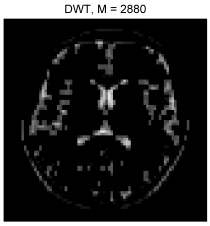


Figure 4.173: DCT: Image M = 4.5K = 2880

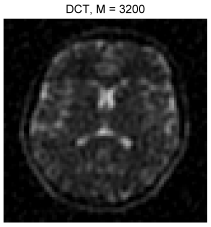


Figure 4.174: DCT: Image M = 5K = 3200

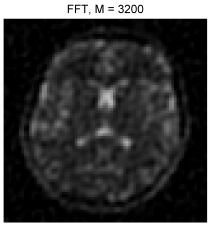


Figure 4.175: DCT: Image M = 5K = 3200

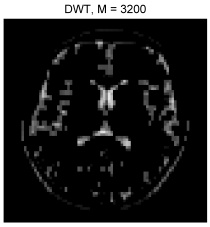


Figure 4.176: DCT: Image M = 5K = 3200

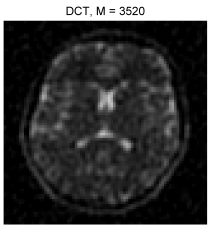


Figure 4.177: DCT: Image M = 5.5K = 3520

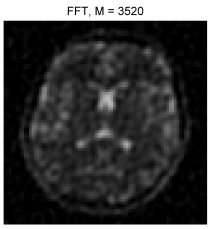


Figure 4.178: DCT: Image M = 5.5K = 3520

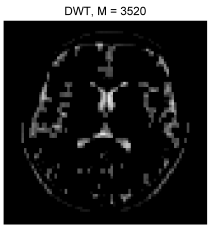


Figure 4.179: DCT: Image M = 5.5K = 3520

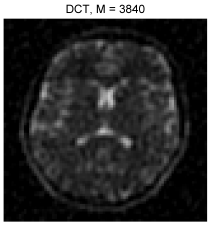


Figure 4.180: DCT: Image M = 6K = 3840

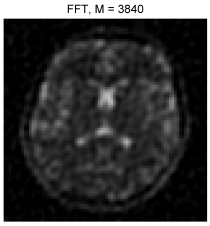


Figure 4.181: FFT: Image M = 6K = 3840

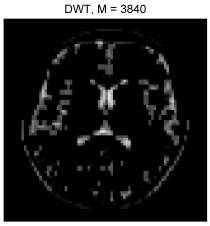


Figure 4.182: DWT: Image M = 6K = 3840

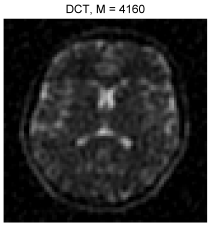


Figure 4.183: DCT: Image M = 6.5K = 4160

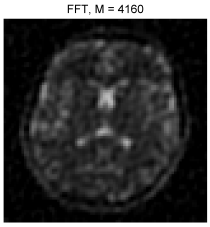


Figure 4.184: FFT: Image M = 6.5K = 4160

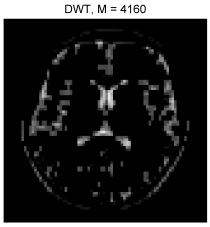


Figure 4.185: DWT: Image M = 6.5K = 4160

4.10.2 Brain 1 Image - Analysis

The analysis follows a similar structure to that of BP and OMP. However, I have adapted my results so that the measurements begin at M = 3K which is a key threshold for the algorithm to display a reconstruction to a decent degree of accuracy as explored in the numerical results. Therefore, the x-axis begins at M/K = 3.

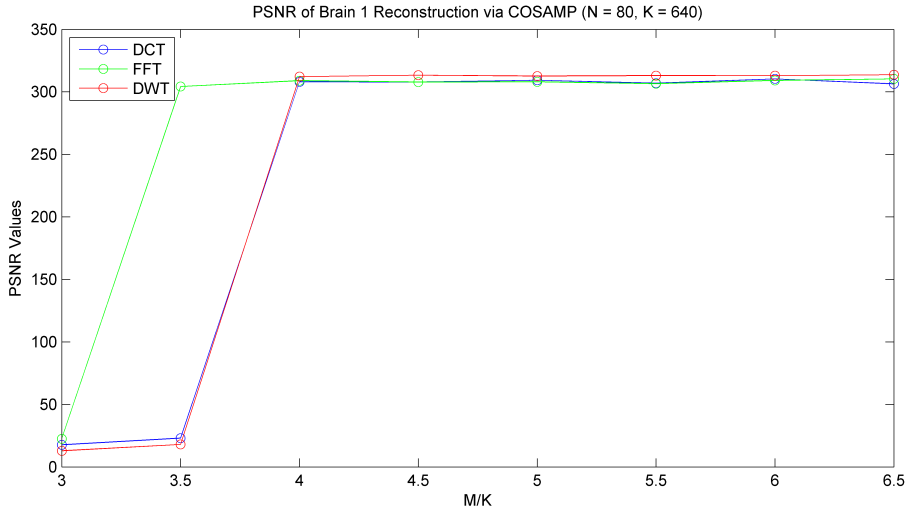


Figure 4.186: Plot of how the PSNR values change for an increasing number of measurements for Brain 1.

Like OMP, we can analyse the results from the FFT transform. Like BP, there is a large "jump" in the PSNR which changes depending on the transform the algorithm is applied to. As seen in the case of FFT, the "jump" occurs earlier. However, the magnitude of the increase is much larger than that which is present in OMP or BP as in just K/2 extra measurements past a certain measurement threshold, we see a rise of 275dB in the PSNR. This then remains constant almost constant about 300dB no matter how many more measurements are taken. This means that for a very high quality reconstruction of the sparse data, we need no more measurements than 4K or 3.5K depending on the transform.

We see a similar pattern with the error. at M = 3.5K and M = 4K for the FFT, DCT and DWT respectively, the error becomes almost 0. Taking more measurements results in a very small error decrease for the computation effort

involved. For OMP, the error at these points is also close to 0 for the FFT transform, however, for DWT and DCT, the error is still larger than 1 and we need more measurements for the error to become close to 0. The error of BP is similar to the case of CoSaMP where the error becomes approximately 0 at M/K = 3.5.

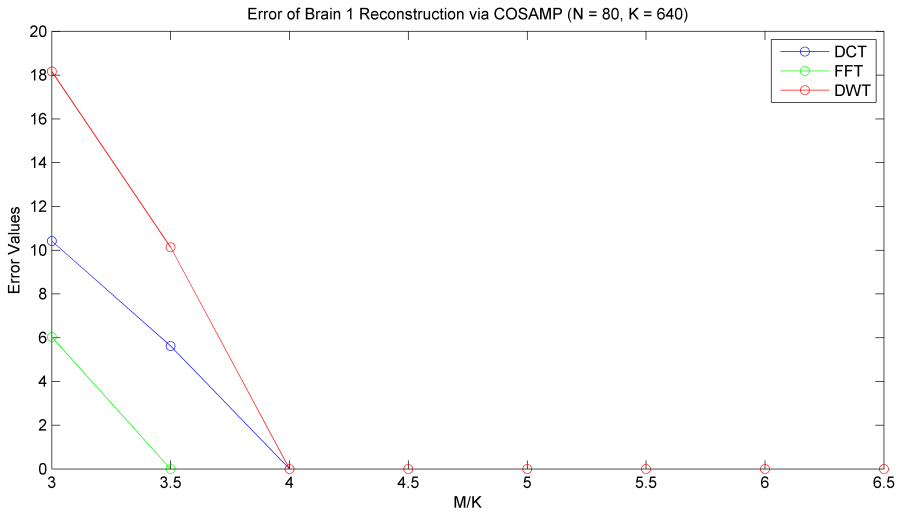


Figure 4.187: Plot of how the error values change for an increasing number of measurements for Brain 1.

The time taken to perform the algorithm on each transform is shown below with the wavelet transform being the fastest as it has been in the prior tests with OMP and BP. The speed of the algorithm is also dictated by the halting criterion. As I supplied a max iteration count of 20, it is likely that measurement less than M = 3K took the entire 20 iterations where as those over M = 3K satisfied the other condition that the residual must be small enough. notice that the time taken for DCT and FFT stabilizes to almost 15 seconds and the DWT transform requires < 5 seconds no matter the ratio of M/K. Comparatively, OMP takes more time as the amount of measurements increase and the same could be said for BP.

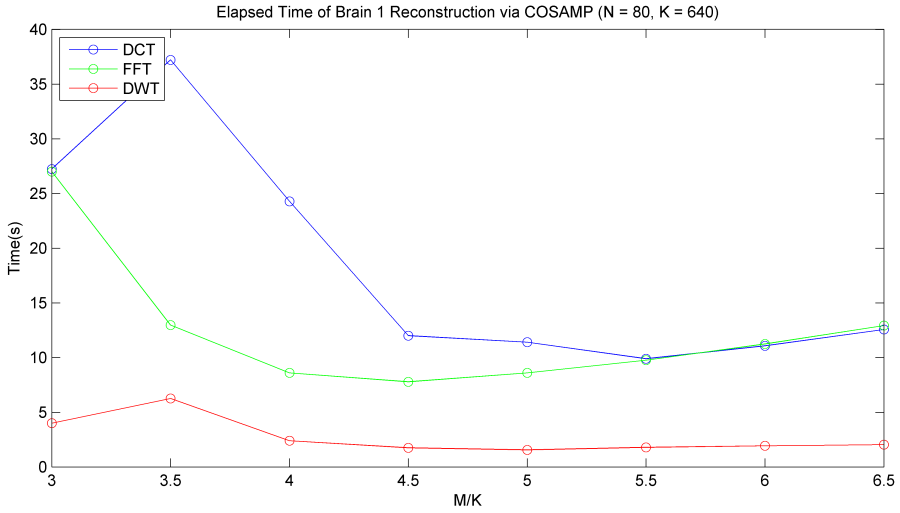


Figure 4.188: Plot of how the time to generate the results change for an increasing number of measurements for Brain 1.

4.10.3 Brain 2 - Image Reconstruction

I now run tests on image 4.19b as before.

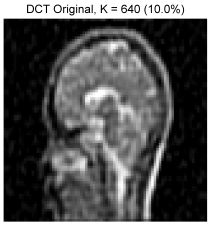


Figure 4.189: DCT: Image K = 10%

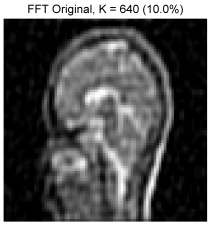


Figure 4.190: FFT: Image K = 10%

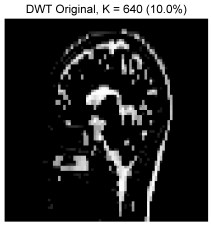


Figure 4.191: DWT: Image K = 10%

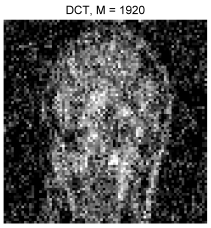


Figure 4.192: DCT: Image M = 3K = 1920

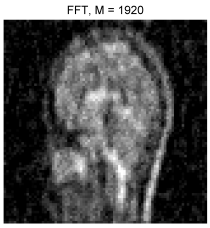


Figure 4.193: FFT: Image M = 3K = 1920

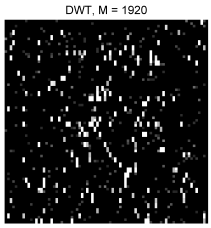


Figure 4.194: DWT: Image M = 3K = 1920

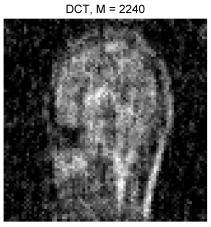


Figure 4.195: DCT: Image M = 3.5K = 2240

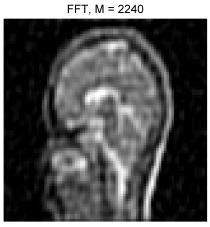


Figure 4.196: FFT: Image M = 3.5K = 2240

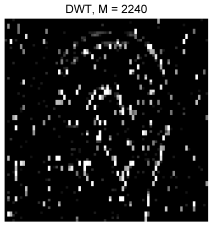


Figure 4.197: DWT: Image M = 3.5K = 2240

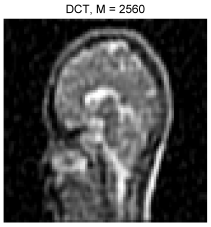


Figure 4.198: DCT: Image M = 4K = 2240

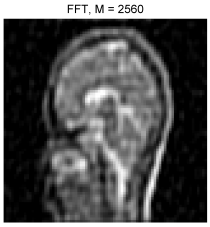


Figure 4.199: FFT: Image M = 4K = 2560

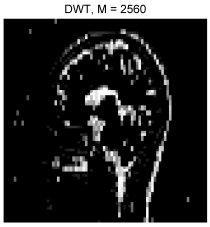


Figure 4.200: DWT: Image M = 4K = 2560

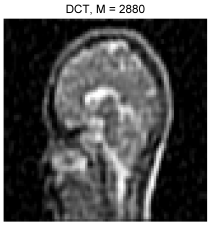


Figure 4.201: DCT: Image M = 4.5K = 2880

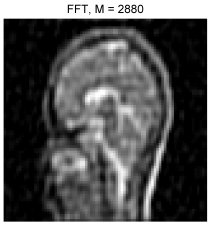


Figure 4.202: DCT: Image M = 4.5K = 2880

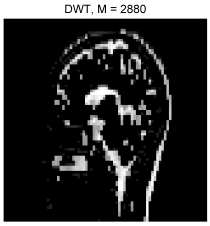


Figure 4.203: DCT: Image M = 4.5K = 2880

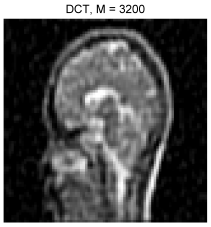


Figure 4.204: DCT: Image M = 5K = 3200

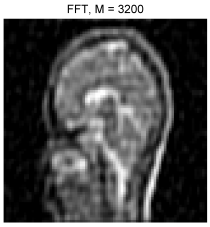


Figure 4.205: DCT: Image M = 5K = 3200

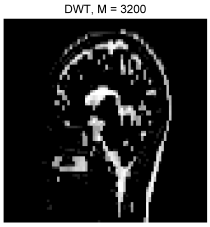


Figure 4.206: DCT: Image M = 5K = 3200

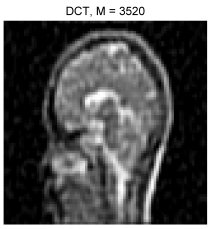


Figure 4.207: DCT: Image M = 5.5K = 3520

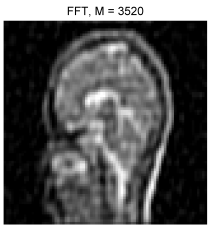


Figure 4.208: DCT: Image M = 5.5K = 3520

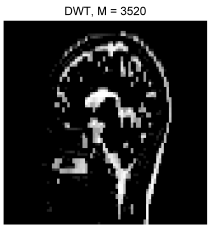


Figure 4.209: DCT: Image M = 5.5K = 3520

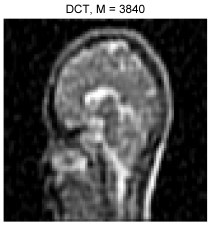


Figure 4.210: DCT: Image M = 6K = 3840

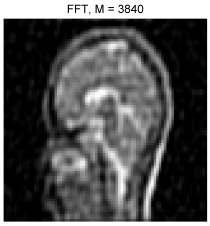


Figure 4.211: FFT: Image M = 6K = 3840

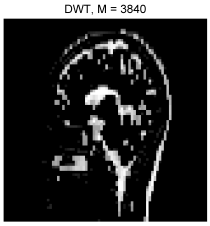


Figure 4.212: DWT: Image M = 6K = 3840

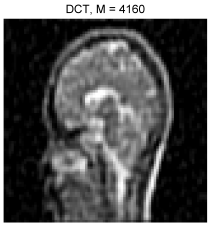


Figure 4.213: DCT: Image M = 6.5K = 4160

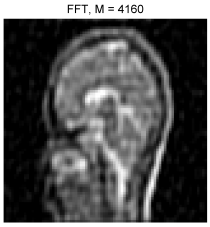


Figure 4.214: FFT: Image M = 6.5K = 4160

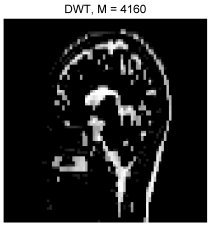


Figure 4.215: DWT: Image M = 6.5K = 4160

4.10.4 Brain 2 Image - Analysis

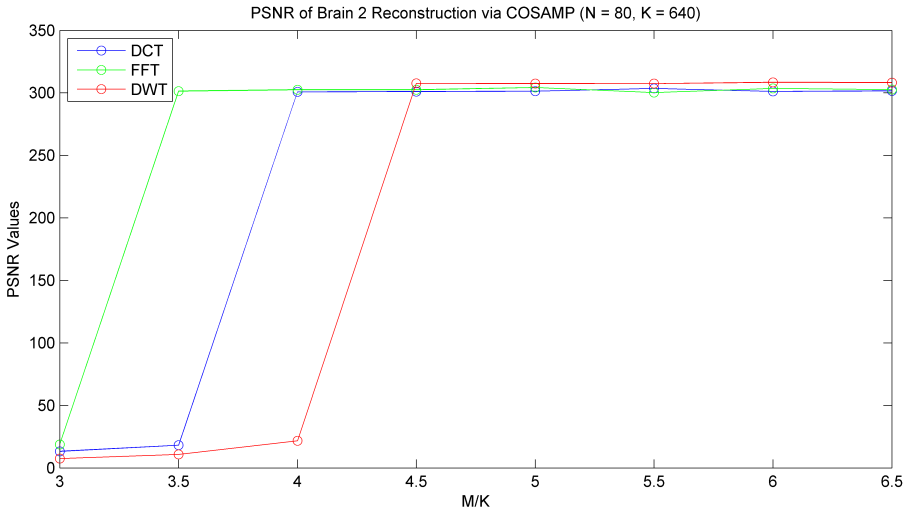


Figure 4.216: Plot of how the PSNR values change for an increasing number of measurements for Brain 2.

Again, the results show the same obvious trend as discussed in the analysis of the first brain. However, we notice that the threshold for the PSNR of the DWT transform has changed from the previous M/K = 3.5 to 4. A similar result is observed for the error where the DWT error now is approximately 0 at M/K = 4.5 instead of 4. However, the time taken follows the same pattern as seen when we analysed the results from brain 1. The points on the graph may differ slightly from those of brain 1 as they are different images, but the trend and overall shape remains the same.

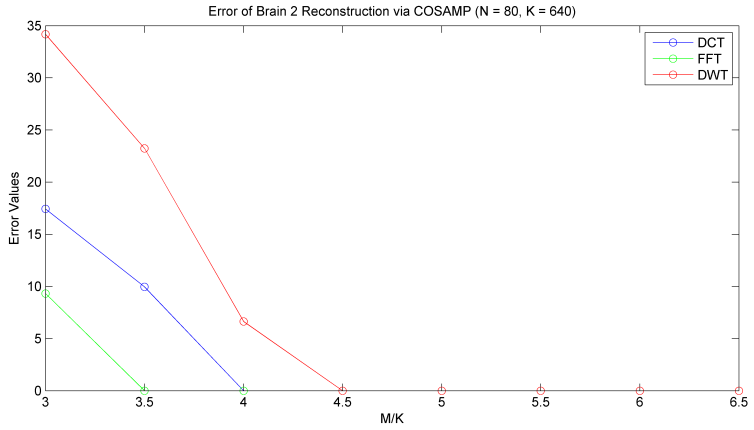


Figure 4.217: Plot of how the error values change for an increasing number of measurements for Brain 2.

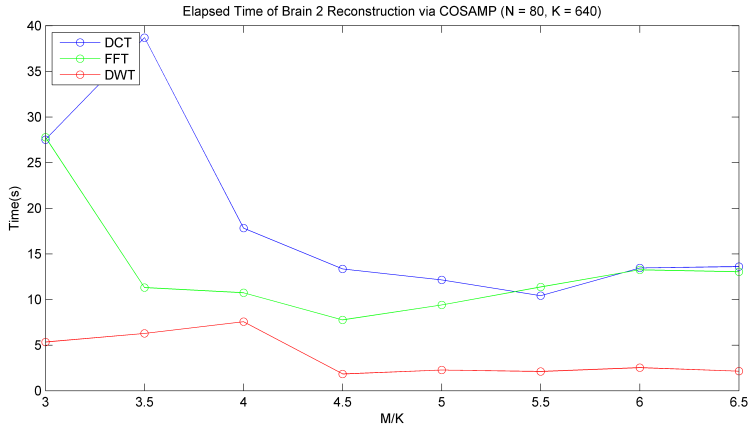


Figure 4.218: Plot of how the time to generate the results change for an increasing number of measurements for Brain 2.

4.11 Conclusion

From our experiments we see that CoSaMP requires a lot of measurements in order to produce accurate results. However, the results produced often have a have a high degree of accuracy and the algorithm performs faster than both OMP and BP as demonstrated for larger ratio's of M/K. Further, it was shown in Theorem 4.9 that accurate reconstructions of sparse signals are guranteed as are approximations to noisy signals[9].

Following this, we see that CoSaMP is an improvement on OMP. It mimics the performance of Basis Pursuit whilst retaining the speed meaning it is able to provide uniform guarantees and is also a stable algorithm.

To summarize:

Advantages

- Provides uniform guarantees.
- Is a stable algorithm.
- Fast run time

Disadvantages

- Requires a large amount of measurements

Chapter 5

Conclusion

Compressed sensing improves on the traditional process of image compression where the image is first acquired and then compressed which results in discarding most of the data. Instead of this, the two steps are performed at the same time which is an improvement on the traditional methods. Through compressed sensing, we are able to reconstruct images at a rate far below the Nyquist rate by exploiting the sparsity of an image. Many signals and images are usually dense in information such as MRI images however, they can be made sparse under some transforms such as DCT, DWT and FFT. Once the images are transformed into a sparse domain, we sample the data in this domain and reconstruct the original sparse data by utilizing the algorithms presented in this document and inverting the transformation.

For the three algorithms that were experimented with, we discussed their differences, advantages and disadvantages. It was found that whilst Basis Pursuit was generally slow, it was applicable to all sparse or noise perturbed signals and provided uniform guarantees. Orthogonal Matching Pursuit however, was quite fast but provided non-uniform guarantees and required extra conditions on the measurement matrix besides the restricted isometric property. Compressive Sampling Matching Pursuit was then tested and through the acclimation of the tests and outside resource, it was found that CoSaMP provided the stability and uniform guarantees of Basis Pursuit whilst its run time was faster and appeared as an overall improvement on OMP. Further, no extra condition other than the Restricted Isometric Property was required for admissible measurement matrices. However, our tests did demonstrate that a large number of measurements were required for accurate reconstruction.

Following our experiments on MRI images and the information I gleaned from various resources, we are able to use the criterion defined in section 3.6 to sort the algorithms in terms of their use.

General Applicability:

For signals with noise or any sparse signals in general, CoSaMP and BP are best as both provide uniform guarantees and are stable.

Speed:

For speed, CoSaMP and OMP are best as they have strong polynomial running time. No algorithms have yet been found that allow strongly polynomial-time performance in linear programming.

Reconstruction from Small Samples:

If the goal was to take the least amount of measurements for successful signal reconstruction then my results indicate that OMP and BP are best with OMP providing a better PSNR at smaller measurements than BP.

In conclusion, my project focused on researching three main algorithms and analysing their effects on 1-D signals and 2-D MRI images under sparse transforms. The goal was to introduce what compressed sensing is, how it worked and its use in MRI coupled with experiments which provided empirical evidence of the use of compressed sensing. The field is still young and there are many more reconstruction algorithms some with better applicability in reconstructing 2-D MRI images from sparse domains than the ones discussed.

Appendices

Appendix A

BP - Algorithm

```
function [correct,normError, peakSNR] = BasisPursuit1D(N,M,K)
%[correct, normError, peakSNR] = BasisPursuit1D(N,M,K)
%Function Description:
%Performs Basis Pursuit on a Nx1 randomly
%generated symbol x with K peaks
%and sparsity taking M samples by using a
%MxN gaussian measurement matrix.
%If no peak value is given, it generates a
%zero signal with 20 peaks
%
%Uses: l1eq_pd.m from L1Magic
(http://users.ece.gatech.edu/justin/l1magic/)
%Parameters:
%N - Generate signal lengh N
%M - Number of samples to take
%K - Number of peaks in signal in the randomly generated signal
%
%Outputs:
%correct - If the signal was correctly recovered according
%to some criterion
%normError - The 2-Norm error between original signal
%and reconstruction
%peakSNR - PSNR of reconstructed signal compared to original
```

```
%Adding the path
%addpath(genpath((strcat(pwd,'/l1magic'))));

if (nargin < 3)
    %Number of peaks in the signal
    K = 20;
end
if (nargin < 2)
    %Number of samples to take
    M = 64;
end
if (nargin < 1)
    %Length of Signal
    N = 256;
end

%Keep Seed
%rng(10)

%Number of peaks
peakCount = K;

%Store Signal
x = zeros(N,1);

%Peak Locations
peakLoc = randperm(N);
peakLoc = peakLoc(1:peakCount);

%Add peaks to zero signal
x(peakLoc) = randn(1, peakCount);

%Set Plot Limits
yLimit = 1.25*max(abs(x));

%Plot Original Signal
figure;
plot(x);
```

```
title('Original Signal'); xlim([1 N]); ylim([-yLimit yLimit]);

%Width and Height of Plots
width = 30; % cm
height = 15; % cm
set(gcf, 'PaperPosition', [0, 0, width / 2.54, height / 2.54])
print -dpng -r500 OSignal

% Generate Measurement Matrix for K samples
A = randn(M, N);

% Generate Output Vector
y = A*x;

%Plot K Value
figure;
plot(y);
title('Output Signal from K Measured Values');
xlim([1 M]);
set(gcf, 'PaperPosition', [0, 0, width / 2.54, height / 2.54])
print -dpng -r500 KVals

%Do CS Recovery via l1 minimization.
xInitial = A.'*y;

tic
xp = l1eq_pd(xInitial, A, [], y);
elapsedTime = toc;

%Plot the output
figure;
plot(xp);
title('Reconstructed Signal');
xlim([1 N]);
ylim([-yLimit yLimit]);

%Get Norm Error, PSNR and Correct values
normError = norm(x-real(xp));
```

```
[peakSNR, ~] = psnr(x, real(xp));
correct = 0;
if normError <  1e-2
    correct = correct + 1;
end

%Add String Annotation to Plot
str = {['PSNR = ' num2str(peakSNR)],['2-Norm Error  '...
    '=' num2str(normError)]};
annotation('textbox',[0.57  0.70  0.32  0.20],'String',str);
%Print
set(gcf, 'PaperPosition', [0, 0, width / 2.54, height / 2.54])
print -dpng -r500 RSignal
```

Appendix B

BP - Noisy

```
function [correct, normError, peakSNR] = BasisPursuit1D(N,M)
%[normError, peakSNR] = BasisPursuit1D(N,M)
%Function Description:
%The image is reconstructed via IDCT.
%Performs Basis Pursuit on a Nx1 signal retrieved
%using ex4mwden (2nd signal). The DCT coefficients are
%calculated and M samples are taken by using
%a MxN gaussian measurement matrix.
%Reconstructed via IDCT.
%If no value for N or M are given, defaults are
%N = 1024, M = 256 (25% sample of original)
%
%Uses: l1eq_pd.m from L1Magic
%(http://users.ece.gatech.edu/justin/l1magic/)
%Parameters:
%N - Generate signal lengh N
%M - Number of samples to take
%
%Outputs:
%normError - The 2-Norm error between original signal
%and reconstuction
%peakSNR - PSNR of reconstructed signal compared
%to original
```

```
load ex4mwden

%Generate Signal
x = x_orig(:,2);

if (nargin < 2)
    %Number of samples to take
    M = 256;
end
if (nargin < 1)
    %Length of Signal
    N = 1024;
end

if (N ~= 1024)
    temp = x_orig(:,2);
    temp = imresize(temp, [256,1]);
    x = temp;
end

%Signal not sparse in time domain, but DCT coefficients
%are sparse
dctCoeff = dct(x);

%Set Plot Limits
yLimit = 1.25*max(abs(x));

%Plot Original Signal in Time Domain
figure;
plot(x);
title('Original Signal - Time Domain');
xlim([1 N]); ylim([-yLimit yLimit]);

%Width and Height of Plots
width = 60; % cm
height = 30; % cm
set(gcf, 'PaperPosition', ...
[0, 0, width / 2.54, height / 2.54])
```

```
print -dpng -r500 OSignalNoise

yLimitF = 1.25*max(abs(dctCoeff));

%Plot Original Frequency in Frequency Domain
figure;
plot(dctCoeff)
title('Original Signal - Time Domain');
xlim([1 N]); ylim([-yLimitF yLimitF]);
set(gcf, 'PaperPosition', ...
[0, 0, width / 2.54, height / 2.54])
print -dpng -r500 OSignalNoiseF

% Generate Measurement Matrix for K samples
A = randn(M, N);
% noisy observations
sigma = 0.005;
e = sigma*randn(M,1);
y = A*dctCoeff + e;

%Plot Noisy Signal in Time Domain
figure;
plot(A*x + e);
title('Signal Sampled with Noise - Time Domain');
yLimitE = 1.25*max(abs(A*x + e));
xlim([1 M]); ylim([-yLimitE yLimitE]);
set(gcf, 'PaperPosition', ...
[0, 0, width / 2.54, height / 2.54])
print -dpng -r500 KValsNoise

%Error constraint for ||Ax - b||_1 <= Error
%(from L1Magic)
Error =  sigma*sqrt(M)*sqrt(1 + 2*sqrt(2)/sqrt(M));


% Do CS Recovery via l1 minimization.
%Initial guess via least squares
xInitial = A.'*y;
```

```
tic
xp = l1qc_logbarrier(xInitial, A, [], y, Error, 1e-3);
elapsedTime = toc;

%Plot Reconstructed Signal in Time Domain
figure;
plot(idct(xp));
title('Reconstructed Signal - Time Domain');
xlim([1 N]);
ylim([-yLimit yLimit]);


%Get Norm Error, PSNR and Correct values
normError = norm(x-real(xp));
[peakSNR, ~] = psnr(x, real(xp));
correct = 0;
if normError <  1e-2
    correct = correct + 1;
end

%Add String Annotation to Plot
str = {['PSNR = ' num2str(peakSNR)],['2-Norm Error  '...
    '=' num2str(normError)], ['Time Taken' ...
    ' =' num2str(elapsedTime)]};
annotation('textbox',[0.57 0.70 0.32 0.20], ...
'String',str);
%Print
set(gcf, 'PaperPosition',...
 [0, 0, width / 2.54, height / 2.54])
print -dpng -r500 RSignalNoise

%Plot Reconstructed Signal in Time Domain
figure;
plot(xp);
title('Reconstructed Signal - Frequency Domain');
xlim([1 N]);
ylim([-yLimitF yLimitF]);
set(gcf, 'PaperPosition', ...
```

```
 [0, 0, width / 2.54, height / 2.54])
print -dpng -r500 RSignalNoiseF
```

Appendix C

BP - MRI Images

```
function [normError, peakSNR] = BPImage(x,N,M,K,type,brainNumber)
%[normError, peakSNR] = BPImage(x,N,M,K,type,brainNumber)
%Function Description:
%Performs Basis Pursuit on a NxN image. The image
%is resized to N*N x 1 (taking the gray values).
%The DCT/FFT/DWT coefficients are calculated and
%the largest K values are considered.
%M samples are taken via MxN Gaussian measurement
%matrix. The image is then reconstructed
%
%
%Uses: l1eq_pd.m from L1Magic
%(http://users.ece.gatech.edu/justin/l1magic/)
%Parameters:
%x - An image read into matlab via imread()
%K - Sparsity of image required
%N - Generate signal length N
%M - Number of samples to take
%brainNumber - Brain used in algorithm
%1 - T2 Axial of Brain
%2 - Side view of Brain
%
%Outputs:
%normError - The 2-Norm error between original image
```

```
%and reconstruction
%peakSNR - PSNR of reconstructed image compared
%to original

%Set up matrix
x = rgb2gray(x);
x = im2double(x);
x = imresize(x, [N N]);
x = reshape(x,N*N,1);

%Get brain name
if brainNumber == 1
    brainName = sprintf('Brain1');
elseif brainNumber == 2
    brainName = sprintf('Brain2');
end

%Default type = 1; K = 1280; M = K; N = 80 (K = 640 for Wave);

%Get coefficients in a transform domain
if type == 1
    coeff = dct(x);
    str=sprintf('DCT Original, K = %d (%2.1f%%)', ...
        K,(K/length(x))*100);
elseif type == 2
    %Always gets stuck backtracking in linesearch
    %Not a good example
    %See Section 4 of L1Magic notes
    coeff = fft(x);
    str=sprintf('FFT Original, K = %d (%2.1f%%)', ...
        K,(K/length(x))*100);
elseif type == 3
    %Remember, we are taking K of the details AND
    %coefficients therefore choose K = K/2.
    %Using db1 wavelet.
    [coeffA,coeffD] = dwt(x,'db1');
    str=sprintf('DWT Original, K = %d (%2.1f%%)', ...
        K*2,(K/length(x))*100*2);
```

```
    coeff = [coeffA;coeffD];
end

%Find K largest coefficients, others 0
if type == 3
    [vals, index] = sort(abs(coeffA), 'descend');
    coeff(index(K+1:N*N/2)) = 0;
    [vals, index] = sort(abs(coeffD), 'descend');
    coeff(index(K+1:N*N/2)+(N*N)/2) = 0;
else
    [vals, index] = sort(abs(coeff), 'descend');
    coeff(index(K+1:length(coeff))) = 0;
end


%Plot Original Image

figure;
if type == 1
    imshow(reshape(idct(coeff),N,N),'InitialMagnification','fit');
     saveAs = sprintf('BP-%s-Original-DCT',brainName);
elseif type == 2
    imshow(reshape(ifft(coeff),N,N),'InitialMagnification','fit');
     saveAs = sprintf('BP-%s-Original-FFT',brainName);
elseif type == 3
    imshow(reshape(idwt(coeff(1:N*N/2),coeff(N*N/2 + 1: N*N),'db1'),N,N
    saveAs = sprintf('BP-%s-Original-DWT',brainName);
end
title(str, 'FontSize', 20);
print( '-dpng', '-r500', saveAs);


% Generate Measurement Matrix
if type == 3
    A = randn(M,N*N/2);
else
    A = randn(M, N*N);
end
```

```
%Width and Height of Plots
width = 30; % cm
height = 15; % cm

% large scale
%Use to save space for larger matrix but takes
%very long!
%Afun = @(z) A*z;
%Atfun = @(z) A.'*z;
%y = Afun(coeff);
%xInitial = Atfun(y);
%tic
%xp = l1eq_pd(xInitial, Afun, Atfun, y);
%elapsedTime = toc

% Do CS Recovery via l1 minimization.
%Initial guess via least squares
if type == 1 || type == 2
        %Generate output signal
        y = A*coeff;

        xInitial = A.'*y;
        tic
        xp = l1eq_pd(xInitial, A, [], y);
        elapsedTime = toc;
elseif type == 3
    y_1 = A*coeff(1:N*N/2);
    y_2 = A*coeff(N*N/2 + 1:N*N);

    xInitial_1 = A.'*y_1;
    xInitial_2 = A.'*y_2;
    tic
    xp_1 = l1eq_pd(xInitial_1, A, [], y_1);
    xp_2 = l1eq_pd(xInitial_2, A, [], y_2);
    elapsedTime = toc;
end
```

```
%Show reconstructed image
figure;
if type == 1
    recImage = idct(xp);
    imshow(reshape(recImage,N,N),'InitialMagnification','fit');
    str_2=sprintf('DCT, M = %d',M);
    measured = sprintf('BP-%s-%d-DCT',brainName,M);
elseif type == 2
    recImage = ifft(xp);
    imshow(reshape(recImage,N,N),'InitialMagnification','fit');
    str_2=sprintf('FFT, M = %d',M);
    measured = sprintf('BP-%s-%d-FFT',brainName,M);
elseif type == 3
    recImage = idwt(xp_1(1:N*N/2),xp_2(1:N*N/2),'db1');
    imshow(reshape(recImage,N,N),'InitialMagnification','fit');
    str_2=sprintf('DWT, M = %d',M*2);
    measured = sprintf('BP-%s-%d-DWT',brainName,M*2);
end
title(str_2, 'FontSize', 20);
print('-dpng','-r500',measured);

%Get Norm Error & PSNR
if type == 3
    oImage = idwt(coeff(1:N*N/2),coeff(N*N/2 + 1: N*N),'db1');
elseif type == 2
    oImage = ifft(coeff);
else
    oImage = idct(coeff);
end

normError = norm(recImage - oImage);
[peakSNR, ~] = psnr(recImage, oImage);

%Set file name
if type == 1
    fileName = sprintf('BPDCT%sResults%d.txt',brainName,M);
elseif type == 2
    fileName = sprintf('BPFFT%sResults%d.txt',brainName,M);
```

```
elseif type ==3
    fileName = sprintf('BPDWT%sResults%d.txt',brainName,M*2);
end

%Write results to file
fileID = fopen(fileName,'w');
fprintf(fileID,'%6s %6s %6s\r\n','PSNR','Error','Time');
fprintf(fileID,'%6.4f %6.4f %6.4f\r\n',[peakSNR; normError; elapsedTime
fclose(fileID);
```

Appendix D

OMP - Algorithm

```
function [numCorrect, normError, peakSNR] = OMP1D(N,M,K)
%[numCorrect, normError, peakSNR] = OMP1D(N,M,K)
%Function Description:
%Performs OMP on a Nx1 randomly generated signal x with
%K peaks and sparsity taking M samples by using
%a MxN gaussian measurement matrix.
%If no peak value is given, it generates a zero
%signal with 20 peaks
%
%Parameters:
%N - Generate signal lengh N
%M - Number of samples to take
%K - Number of peaks in signal in the
%randomly generated signal (and iteration limit)
%
%Outputs:
%numCorrect - Increments by 1 if the support and index set
%match
%normError - The 2-Norm error between original signal
%and reconstuction
%peakSNR - PSNR of reconstructed signal compared to
%original
```

```
if (nargin < 3)
    %Number of peaks in the signal
    K = 20;
end
if (nargin < 2)
    %Number of samples to take
    M = 64;
end
if (nargin < 1)
    %Length of Signal
    N = 256;
end

%Sparse non-zero vals
peakCount = K;

%Store Signal
x = zeros(N,1);

%Peak Locations
peakLoc = randperm(N);
peakLoc = peakLoc(1:peakCount);
%Add peaks to zero signal
x(peakLoc) = randn(1, peakCount);

% Generate Measurement Matrix for K-Sparse x
A = randn(M, N);

%Index Set
I = zeros(1,1);

%Output Signal
b = A * x;

%same Seed
%rng(5)

%Set Plot Limits
```

```
yLimit = 1.25*max(abs(x));

%Width and Height of Plots
width = 30; % cm
height = 15; % cm

%plot the original signal
figure;
plot(x);
title('Original Signal');
xlim([1 N]);
ylim([-yLimit yLimit]);
set(gcf, 'PaperPosition', [0, 0, width / 2.54, height / 2.54])
print -dpng -r500 OMPOSignal50


%Set Plot Limits
yLim = 1.25*max(abs(b));

%Plot Output Signal
figure;
plot(b);
title('Output Signal, M = 50');
xlim([1 M]);
ylim([-yLim yLim]);
set(gcf, 'PaperPosition', [0, 0, width / 2.54, height / 2.54])
print -dpng -r500 OMPKVals50

%Residual
r = b;

%holds new values of MxN measurement matrix
ANew = [];

%Count iterations
iter = 0;

%Run OMP
```

```
while iter < K
    iter = iter + 1;
    v = A'*r;
    absoluteV = abs(v);
    [~, bIndex ] = sort(absoluteV, 'descend');
    bestInd = bIndex(1);
    %Update I
    I(length(I)+1) = bestInd;
    A_I = A(:, I(iter + 1));
    %Augment the matrix
    ANew = [ANew, A_I];
    %Solve least squares
    xNew = ANew\b;
    y = ANew * xNew;
    %Update the residual
    r = b - y;
end

%reconstruct signal
rec = zeros(N,1);
Z = I(:,2:K+1);
rec(Z) = xNew;

%Plot Reconstructed signal
figure;
plot(rec);
title('Reconstructed Signal, M = 200');
xlim([1 N]);
ylim([-yLimit yLimit]);

%Get Norm Error & PSNR
normError = norm(x - rec);
[peakSNR, ~] = psnr(x, rec);
numCorrect = 0;

%test if signals have same support with
%prior information
sortP = sort(abs(peakLoc),'descend');
```

```
sortI = sort(abs(I), 'descend');

if ismember(sortP, sortI)
    numCorrect = numCorrect + 1;
end

%Add String Annotation to Plot
str = {['PSNR = ' num2str(peakSNR)],['2-Norm Error  '...
    '=' num2str(normError)]};
annotation('textbox',[0.57 0.70 0.32 0.20],'String',str);
%Print
set(gcf, 'PaperPosition', [0, 0, width / 2.54, height / 2.54])
print -dpng -r500 OMPRSignal200
```

Appendix E

OMP - MRI Images

```
function [normError, peakSNR] = OMPImage(x,N,M,K,type,brainNumber)
%[normError, peakSNR] = OMPImage(x,N,M,K,type,brainNumber)
%Function Description:
%Performs OMP on a NxN image. The image
%is resized to N*N x 1 (taking the gray values).
%The DCT/FFT/DWT coefficients are calculated and
%the largest K values are considered.
%M samples are taken via MxN gaussian measurement
%matrix. The image is then reconstructed
%
%Parameters:
%x - An image read into matlab via imread()
%K - Sparsity of image required
%N - Generate signal lengh N
%M - Number of samples to take
%Type - Type of transform on image
%brainNumber - Brain used in algorithm
%1 - T2 Axial of Brain
%2 - Side view of Brain
%
%Outputs:
%normError - The 2-Norm error between original image
%and reconstruction
```

```
%peakSNR - PSNR of reconstructed image compared
%to original

%Set up matrix
x = rgb2gray(x);
x = im2double(x);
x = imresize(x, [N N]);
x = reshape(x,N*N,1);

if brainNumber == 1
    brainName = sprintf('Brain1');
elseif brainNumber == 2
    brainName = sprintf('Brain2');
end


%Default type = 1; K = 640; M = K; N = 80;

%Get coefficients in a transform domain
if type == 1
    coeff = dct(x);
    str=sprintf('DCT Original, K = %d (%2.1f%%)', ...
        K,(K/length(x))*100);
elseif type == 2
    coeff = fft(x);
    str=sprintf('FFT Original, K = %d (%2.1f%%)', ...
        K,(K/length(x))*100);

elseif type == 3
    %Remember, we are taking K of the details AND
    %coefficients therefore choose K = K/2.
    %Using db1 wavelet.
    [coeffA,coeffD] = dwt(x,'db1');
    str=sprintf('DWT Original, K = %d (%2.1f%%)', ...
        K*2,(K/length(x))*100*2);
    coeff = [coeffA;coeffD];
end
```

```
%Find K largest coefficients, others 0
if type == 3
    [vals, index] = sort(abs(coeffA), 'descend');
    coeff(index(K+1:N*N/2)) = 0;
    [vals, index] = sort(abs(coeffD), 'descend');
    coeff(index(K+1:N*N/2)+(N*N)/2) = 0;
else
    [vals, index] = sort(abs(coeff), 'descend');
    coeff(index(K+1:length(coeff))) = 0;
end

%Plot Original Image
figure;
if type == 1
    imshow(reshape(idct(coeff),N,N),'InitialMagnification','fit');
    saveAs = sprintf('OMP-%s-Original-DCT',brainName);
elseif type == 2
    imshow(reshape(ifft(coeff),N,N),'InitialMagnification','fit');
    saveAs = sprintf('OMP-%s-Original-FFT',brainName);
elseif type == 3
    imshow(reshape(idwt(coeff(1:N*N/2),coeff(N*N/2 + 1: N*N),'db1')
    ,N,N),'InitialMagnification','fit');
    saveAs = sprintf('OMP-%s-Original-DWT',brainName);
end
title(str, 'FontSize', 20);
print( '-dpng', '-r500', saveAs);



% Generate Measurement Matrix
if type == 3
    A = randn(M,N*N/2);
else
    A = randn(M, N*N);
end

%Width and Height of Plots
```

```
width = 30; % cm
height = 15; % cm

%Run OMP
if type == 3
    tic
    for i = 1:2
        %Calculate output signal
        if i == 1
            b = A*coeff(1:N*N/2);
        else
            b = A*coeff(N*N/2 + 1:N*N);
        end
        %Residual
        r = b;
        %Iteration count
        iter = 0;
        %Index Set
        I = zeros(1,1);
        %Columns in range
        ANew = [];
        while iter < K
            iter = iter + 1;
            v = A'*r;
            absoluteV = abs(v);
            [~, bIndex ] = sort(absoluteV, 'descend');
            bestInd = bIndex(1);
            %Update I
            I(length(I)+1) = bestInd;
            A_I = A(:, I(iter + 1));
            %Augment the matrix
            ANew = [ANew, A_I];
            %Solve least squares
            xNew = ANew\b;
            y = ANew * xNew;
            %Update the residual
            r = b - y;
        end
```

```
        Z = I(:,2:K+1);
        %reconstruct signal
        if i == 1
            recA = zeros(N*N,1);
            recA(Z) = xNew;
        else
            recB = zeros(N*N,1);
            recB(Z) = xNew;
        end
    end
    elapsedTime = toc
else
    %Calculate output signal
    b = A*coeff;
    %Index Set
    I = zeros(1,1);
    %Residual
    r = b;
    %holds new values of MxN measurement matrix
    ANew = [] ;
    %Count iter
    iter = 0;
    tic
    while iter < K
        iter = iter + 1;
        v = A'*r;
        absoluteV = abs(v);
        [~, bIndex ] = sort(absoluteV, 'descend');
        bestInd = bIndex(1);
        %Update I
        I(length(I)+1) = bestInd;
        A_I = A(:, I(iter + 1));
        %Augment the matrix
        ANew = [ANew, A_I];
        %Solve least squares
        xNew = ANew\b;
        y = ANew * xNew;
        %Update the residual
```

```
        r = b - y;
    end
end
elapsedTime = toc;

%reconstruct signal
if (type == 1) || (type == 2)
    rec = zeros(N*N,1);
Z = I(:,2:K+1);
rec(Z) = xNew;
end

%Show reconstructed image
figure;
if type == 1
    recImage = idct(rec);
    imshow(reshape(recImage,N,N),'InitialMagnification','fit');
    str_2=sprintf('DCT, M = %d',M);
    measured = sprintf('OMP-%s-%d-DCT',brainName,M);
elseif type == 2
    recImage = ifft(rec);
    imshow(reshape(recImage,N,N),'InitialMagnification','fit');
    str_2=sprintf('FFT, M = %d',M);
    measured = sprintf('OMP-%s-%d-FFT',brainName,M);
elseif type == 3
    recImage = idwt(recA(1:N*N/2),recB(1:N*N/2),'db1');
    imshow(reshape(recImage,N,N),'InitialMagnification','fit');
    str_2=sprintf('DWT, M = %d',M*2);
    measured = sprintf('OMP-%s-%d-DWT',brainName,M*2);
end
title(str_2, 'FontSize', 20);
%print('-dpng','-r500',measured);

%Get Norm Error & PSNR
if type == 3
    oImage = idwt(coeff(1:N*N/2),coeff(N*N/2 + 1: N*N),'db1');
elseif type == 2
    oImage = ifft(coeff);
```

```
else
    oImage = idct(coeff);
end
normError = norm(recImage - oImage);
[peakSNR, ~] = psnr(recImage, oImage);


%Get File Name
if type == 1
    fileName = sprintf('OMPDCT%sResults%d.txt',brainName,M);
elseif type == 2
    fileName = sprintf('OMPFFT%sResults%d.txt',brainName,M);
elseif type ==3
    fileName = sprintf('OMPDWT%sResults%d.txt',brainName,M*2);
end

%Write results to file
fileID = fopen(fileName,'w');
fprintf(fileID,'%6s %6s %6s\r\n','PSNR','Error','Time');
fprintf(fileID,'%6.4f %6.4f %6.4f\r\n',[peakSNR; normError;...
 elapsedTime]);
fclose(fileID);
```

Appendix F

CoSaMP - Algorithm

```
function [numCorrect, normError, peakSNR] = CoSaMP1D(N,M,K, maxIter)
%[numCorrect, normError, peakSNR] = CoSaMP1D(N,M,K, maxIter)
%Function Description:
%Performs CoSaMP on a Nx1 randomly generated signal x with
%K peaks and sparsity taking M samples by using
%a MxN gaussian measurement matrix.
%If no peak value is given, it generates a zero
%signal with 20 peaks
%
%Parameters:
%N - Generate signal lengh N
%M - Number of samples to take
%K - Number of peaks in signal in the
%randomly generated signal (and iteration limit)
%maxIter - Max number of iterations
%
%Outputs:
%numCorrect - Increments by 1 if the support and index set
%match
%normError - The 2-Norm error between original signal
%and reconstruction
%peakSNR - PSNR of reconstructed signal compared to
%original
```

```
if (nargin < 3)
    %Number of peaks in the signal
    K = 20;
end
if (nargin < 2)
    %Number of samples to take
    M = 64;
end
if (nargin < 1)
    %Length of Signal
    N = 256;
end

%Sparse non-zero vals
peakCount = K;

%Store Signal
x = zeros(N,1);

%Peak Locations
peakLoc = randperm(N);
peakLoc = peakLoc(1:peakCount);
%Add peaks to zero signal
x(peakLoc) = randn(1, peakCount);

% Generate Measurement Matrix for K-Sparse x
A = randn(M, N);

%Index Set
I = zeros(N,1);

%Output Signal
b = A * x;

%same Seed
%rng(5)

%Set Plot Limits
```

```
yLimit = 1.25*max(abs(x));

%Width and Height of Plots
width = 30; % cm
height = 15; % cm

%Plot original image
figure;
plot(x);
title('Original Signal');
xlim([1 N]);
ylim([-yLimit yLimit]);

set(gcf, 'PaperPosition', [0, 0, width / 2.54, height / 2.54])
saveAs = sprintf('COSAMPOSignal%d',M);
print('-dpng', '-r500', saveAs);


%Set Plot Limits
%yLim = 1.25*max(abs(b));

%Plot Output Signal
figure;
plot(b);
outTitle = sprintf('Output Signal, M = %d',M);
title(outTitle);
xlim([1 M]);
ylim([-yLim yLim]);
set(gcf, 'PaperPosition', [0, 0, width / 2.54, height / 2.54])
saveAs = sprintf('COSAMPKVals%d',M);
print('-dpng', '-r500', saveAs);


%Residual
r = b;

%our estimate for the original signal
z = zeros(N,1);
```

```
%count iterations
iter = 0;

%halting condition
goalReached = 0;

%Run CoSaMP
while ~goalReached

    %Signal Proxy
    v = A'*r;
    %Find 2K largest coefficients
    [~, index] = sort(abs(v), 'descend');
    Omega = index(1:2*K);
    [~, index] = sort(abs(I), 'descend');
    %get the union
    I = union(Omega, index(1:K));
    %Signal Estimation
    y = zeros(N, 1);
    y(I) = A(:, I) \ b;
    %Prune the signal
    [~, index] = sort(abs(y), 'descend');
    %Get K largest coefficients
    z = zeros(N, 1);
    z(index(1:K), 1) = y(index(1:K), 1);
    %Update residual
    r = b - A*z;
    %Iteration counter
    iter = iter + 1;
    %Check Halting Condition
    if iter > maxIter || norm(z-x) <= 1e-2
        goalReached = 1;
    end
end

%Plot Reconstructed signal
figure;
```

```
plot(z);
yLimit = 1.25*max(abs(z));
sizeM = sprintf('Reconstructed Signal, M = %d',M);
title(sizeM);
xlim([1 N]);
ylim([-yLimit yLimit]);

%Get Norm Error & PSNR
normError = norm(z-x);
[peakSNR, ~] = psnr(z, x);
numCorrect = 0;

if normError < 1e-2 || peakSNR >= 10
    numCorrect = numCorrect + 1;
end

%Add String Annotation to Plot
str = {['PSNR = ' num2str(peakSNR)],['2-Norm Error  '...
    '=' num2str(normError)]};
annotation('textbox',[0.57 0.70 0.32 0.20],'String',str);
%Print
set(gcf, 'PaperPosition', [0, 0, width / 2.54, height / 2.54])
str_2 = sprintf('COSAMPRSignal%d',M);
print('-dpng', '-r500', str_2);
```

Appendix G

CoSaMP - MRI Images

```
function [normError, peakSNR] =
 COSAMPImage(x,N,M,K,type,brainNumber,maxIter)
%[normError, peakSNR] = COSAMP(x,N,M,K,type)
%Function Description:
%Performs CoSaMP on a NxN image. The image
%is resized to N*N x 1 (taking the gray values).
%The DCT/FFT/DWT coefficients are calculated and
%the largest K values are considered.
%M samples are taken via MxN Gaussian measurement
%matrix. The Image is then reconstructed
%
%Parameters:
%x - An image read into matlab via imread()
%K - Sparsity of image required
%N - Generate signal lengh N
%M - Number of samples to take
%maxIter - The maximum number of iterations
%Type - Type of transform on image
%brainNumber - Brain used in algorithm
%1 - T2 Axial of Brain
%2 - Side view of Brain
%
%Outputs:
%normError - The 2-Norm error between original signal
```

```
%and reconstuction
%peakSNR - PSNR of reconstructed signal compared
%to origianl

%Set up matrix
x = rgb2gray(x);
x = im2double(x);
x = imresize(x, [N N]);
x = reshape(x,N*N,1);

%Get brain name
if brainNumber == 1
    brainName = sprintf('Brain1');
elseif brainNumber == 2
    brainName = sprintf('Brain2');
end


%Default type = 1; K = 640; M = K; N = 80;

%Get coefficients in a transform domain
if type == 1
    coeff = dct(x);
    str=sprintf('DCT Original, K = %d (%2.1f%%)', ...
        K,(K/length(x))*100);
elseif type == 2
    coeff = fft(x);
    str=sprintf('FFT Original, K = %d (%2.1f%%)', ...
        K,(K/length(x))*100);

elseif type == 3
    %Remember, we are taking K of the details AND
    %coefficients therefore choose K = K/2.
    %Using db1 wavelet.
    [coeffA,coeffD] = dwt(x,'db1');
    str=sprintf('DWT Original, K = %d (%2.1f%%)', ...
        K*2,(K/length(x))*100*2);
    coeff = [coeffA;coeffD];
```

```
end


%Find K largest coefficients, others 0
if type == 3
    [vals, index] = sort(abs(coeffA), 'descend');
    coeff(index(K+1:N*N/2)) = 0;
    [vals, index] = sort(abs(coeffD), 'descend');
    coeff(index(K+1:N*N/2)+(N*N)/2) = 0;
else
    [vals, index] = sort(abs(coeff), 'descend');
    coeff(index(K+1:length(coeff))) = 0;
end

%Plot Original Image

figure;
if type == 1
    imshow(reshape(idct(coeff),N,N),'InitialMagnification','fit');
    saveAs = sprintf('COSAMP-%s-Original-DCT',brainName);
elseif type == 2
    imshow(reshape(ifft(coeff),N,N),'InitialMagnification','fit');
    saveAs = sprintf('COSAMP-%s-Original-FFT',brainName);
elseif type == 3
    imshow(reshape(idwt(coeff(1:N*N/2),coeff(N*N/2 + 1: N*N),'db1'),N,N
    saveAs = sprintf('COSAMP-%s-Original-DWT',brainName);
end
title(str, 'FontSize', 20);
print('-dpng', '-r500', saveAs);


% Generate Measurement Matrix
if type == 3
    A = randn(M,N*N/2);
else
    A = randn(M, N*N);
end
```

```
%Index Set
I = zeros(N*N,1);

%construct output signal
if type == 1 || type == 2
    b = A*coeff;
    %Residual
    r = b;
end

%Width and Height of Plots
width = 30; % cm
height = 15; % cm

%our estimate for final value
z = zeros(N*N,1);

%count iterations
iter = 0;

%halting condition
goalReached = 0;

%Run CoSaMP
if type == 1 || type == 2
    tic
    while ~goalReached
       %Signal Proxy
       v = A'*r;
       %Find 2K largest coefficients
       [~, index] = sort(abs(v), 'descend');
       Omega = index(1:2*K);
       %get support of current estimate
       [~, index] = sort(abs(z), 'descend');
       %get the union
       I = union(Omega, index(1:K));
       %Signal Estimation
       y = zeros(N*N, 1);
```

```
        y(I) = A(:, I) \ b;
        %Prune the signal
        [~, index] = sort(abs(y), 'descend');
        %Get K largest coefficients
        z = zeros(N*N, 1);
        z(index(1:K), 1) = y(index(1:K), 1);
        %Update residual
        r = b - A*z;
        %Iteration counter
        iter = iter + 1;
        %Check Halting Condition
        if iter > maxIter || norm(r) <= 1e-2
            goalReached = 1;
        end
    end
    elapsedTime = toc;
end
if type == 3
    tic
     for i = 1:2
        %Run for both filters
        goalReached = 0;
        iter = 0;
        z = zeros(N*N/2,1);
        if i == 1
            b = A*coeff(1:N*N/2);
            r = b;
        else
            b = A*coeff(N*N/2 + 1:N*N);
            r = b;
        end
        while ~goalReached
            %Signal Proxy
            v = A'*r;
            %Find 2K largest coefficients
            [~, index] = sort(abs(v), 'descend');
            Omega = index(1:2*K);
            [~, index] = sort(abs(z), 'descend');
```

```
            %get the union
            I = union(Omega, index(1:K));
            %Signal Estimation
            y = zeros(N*N/2, 1);
            y(I) = A(:, I) \ b;
            %Prune the signal
            [~, index] = sort(abs(y), 'descend');
            %Get K largest coefficients
            z = zeros(N*N/2, 1);
            z(index(1:K), 1) = y(index(1:K), 1);
            %Update residual
            r = b - A*z;
            %Iteration counter
            iter = iter + 1;
            %Check Halting Condition
            if i == 1
                z_1 = z;
            else
                %disp(sprintf('i val %d' ,i));
                z_2 = z;
            end
            if iter > maxIter || norm(r) <= 1e-2
                goalReached = 1;
            end
        end
    end
    elapsedTime = toc;
end

%Show reconstructed image
figure;
if type == 1
    recImage = idct(z);
    imshow(reshape(recImage,N,N),'InitialMagnification','fit');
    str_2=sprintf('DCT, M = %d',M);
    measured = sprintf('COSAMP-%s-%d-DCT',brainName,M);
elseif type == 2
    recImage = ifft(z);
```

```
    imshow(reshape(recImage,N,N),'InitialMagnification','fit');
    str_2=sprintf('FFT, M = %d',M);
    measured = sprintf('COSAMP_%s_%d_FFT',brainName,M);
elseif type == 3
    recImage = idwt(z_1,z_2,'db1');
    imshow(reshape(recImage,N,N),'InitialMagnification','fit');
    str_2=sprintf('DWT, M = %d',M*2);
    measured = sprintf('COSAMP_%s_%d_DWT',brainName,M*2);
end
title(str_2, 'FontSize', 20);
print('-dpng','-r500',measured);

%Get Norm Error & PSNR
if type == 3
    oImage = idwt(coeff(1:N*N/2),coeff(N*N/2 + 1: N*N),'db1');
elseif type == 2
    oImage = ifft(coeff);
else
    oImage = idct(coeff);
end
normError = norm(recImage - oImage);
[peakSNR, ~] = psnr(recImage, oImage);

%Get file name
if type == 1
    fileName = sprintf('COSAMPDCT%sResults%d.txt',brainName,M);
elseif type == 2
    fileName = sprintf('COSAMPFFT%sResults%d.txt',brainName,M);
elseif type ==3
    fileName = sprintf('COSAMPDWT%sResults%d.txt',brainName,M*2);
end

%Write results to file
fileID = fopen(fileName,'w');
fprintf(fileID,'%6s %6s %6s\r\n','PSNR','Error','Time');
fprintf(fileID,'%6.4f %6.4f %6.4f\r\n',[peakSNR; normError;...
elapsedTime]);
fclose(fileID);
```

Appendix H

Graphs for 1D Signals

```
function [resultsSNR, resultsError, resultsCorrect] =
Generate1DGraph(alg,measurement)
%function [resultsSNR, resultsError, resultsCorrect]
%= Generate1DGraph(alg,measurement)
%Function Description:
%Generates graphs for Correct Signal Reconstruction %, PSNR,
%and the 2-Norm Error for 1 to 100 samples of 6 levels of
%sparsity (K = 5 to 30 in steps of 5) with 100 trials
%at each combination of samples and sparsity level
%
%Parameters:
%alg = number of algorithm you want to run
%1 = BP
%2 = OMP
%3 = COSAMP
%measurement = what we want to measure
%1 = PSNR
%2 = Error
%3 = % Correctly reconstructed
%
%Outputs: (100 samples, 6 Sparsity Levels)
%resultsSNR - The PSNR Matrix values
%resultsError - The 2-Norm matrix values
%resutltsCorrect - The number of signals correctly reconstructed
```

```
%Dimensions
N = 256;
%Samples
M = [1:1:100];
%Sparsity
K = [5:5:30];
%Number of Trials
trialCount = 100;

mSize = length(M);
kSize = length(K);

resultsSNR = zeros(mSize,kSize);
resultsError = zeros(mSize,kSize);
resultsCorrect = zeros(mSize,kSize);
snrVals = 0;
errorVals = 0;
%Collect Results
for i = 1:mSize
    for j = 1:kSize
        for h = 1:trialCount
            if alg == 1
                [count, normError, peakSNR] =
                BasisPursuit1D(N,M(i),K(j));
            elseif alg == 2
                [count, normError, peakSNR] =
                OMP1D(N,M(i),K(j));
            elseif alg == 3
                [count, normError, peakSNR] =
                 CoSaMP1D(N,M(i),K(j),20);
            end
            resultsCorrect(i,j) = resultsCorrect(i,j) + count;
            snrVals = snrVals + peakSNR;
            errorVals = errorVals + normError;
        end
        resultsSNR(i,j) = snrVals/trialCount;
        resultsError(i,j) = errorVals/trialCount;
```

```
        snrVals = 0;
        errorVals = 0;
    end
    disp(sprintf('Finished M = %d', i));
end
figure

%Colour code array
C = ['r-o','g-o','b-o','m-o','k-o','c-o'];

%Swap resultsSNR for resultsError or resultsCorrect
%Generate plots
for i = 1:kSize
    if measurement == 1
        plot(M,resultsSNR(:,i), C(1,1+3*(i-1):i*3));
    elseif measurement == 2
        plot(M,resultsError(:,i), C(1,1+3*(i-1):i*3));
    elseif measurement == 3
        plot(M,resultsCorrect(:,i),C(1,1+3*(i-1):i*3));
    end
    hold on;
end

% %Label Graph
if measurement == 1
    title('PSNR of Reconstructed Signals(N = 256)');
    ylabel('PSNR Values');
    legend('K = 5', 'K = 10', 'K = 15', 'K = 20', 'K = 25', ...
    'K = 30', 'Location', 'NorthWest');
elseif measurement == 2
    title('Error of Reconstructed Signals(N = 256)');
    ylabel('Error Values');
    legend('K = 5', 'K = 10', 'K = 15', 'K = 20', 'K = 25',...
     'K = 30', 'Location', 'NorthEast');
elseif measurement == 3
    title('Percent of Correctly Reconstructed Signals(N = 256)');
    ylabel('Percent Reconstructed');
    legend('K = 5', 'K = 10', 'K = 15', 'K = 20', 'K = 25',...
```

```
      'K = 30', 'Location', 'NorthWest');
end
xlabel('Number of Measurements');
%Print Graph
width = 60; % cm
height = 30; % cm
set(gcf, 'PaperPosition', [0, 0, width / 2.54, height / 2.54])

if alg == 1
    name = sprintf('BP');
elseif alg == 2
    name = sprintf('OMP');
elseif alg == 3
    name = sprintf('COSAMP');
end

if measurement == 1
    measured = sprintf('%spsnr',name);
    print('-dpng','-r500',measured);
elseif measurement == 2
     measured = sprintf('%serror',name);
    print('-dpng','-r500',measured);
elseif measurement ==3
    measured = sprintf('%scorrect',name);
    print('-dpng','-r500',measured);
end
```

Appendix I

Graphs for MRI Images

```
function [B] = GenBPImageGraphs(alg, brainNumber, N, K)
%function [B] = GenBPImageGraphs(alg, brainNumber, N, K)
%
%Parameters:
%brainNumber - The number for the brain we want to perform
%the reconstruction algorithms on
%brainNumber 1 = T2-Axial of brain
%brainNumber 2 = Side view of brain
%N - NxN quality of image we want to work on
%K - The sparsity of image

%Max number of iterations for CoSaMP
iter = 20;

%Set algorithm name
if alg == 1
    algName = sprintf('BP');
elseif alg == 2
    algName = sprintf('OMP');
elseif alg == 3
    algName = sprintf('COSAMP');
end

%Read the 3-D matrix for each image
```

```
%image must be in area!
if brainNumber == 1
    brain = imread('Brain-T2-axial.png');
    brainFileLabel = sprintf('Brain1');
elseif brainNumber == 2
    brain = imread('BrainImage.jpg');
    brainFileLabel = sprintf('Brain2');
end

%3-D Matrix
%Stores results at 8 different M/K ratios
%Stores results for PSNR, Error and Time Taken
%Stores results for each transform DCT, FFT, DWT
B = zeros(8,3,3);

%Generate graph for 8 M/K points.
for i = 1:8
    factor = i/2 + 0.5;
    if alg == 1
        BPImage(brain,N,K*factor,K,1,brainNumber);
        str = sprintf('%sDCT%sResults%d.txt',algName,...
        brainFileLabel,K*factor);
        B(i,:,1) = dlmread(str,' ', 1,0);
        BPImage(brain,N,K*factor,K,2,brainNumber);
        str = sprintf('%sFFT%sResults%d.txt',algName,...
        brainFileLabel,K*factor);
        B(i,:,2) = dlmread(str,' ', 1,0);
        BPImage(brain,N,K*factor*0.5,K*0.5,3,brainNumber);
        str = sprintf('%sDWT%sResults%d.txt',algName,...
        brainFileLabel,K*factor);
        B(i,:,3) = dlmread(str,' ', 1,0);
    end

    if alg == 2
        OMPImage(brain,N,K*factor,K,1,brainNumber);
        str = sprintf('%sDCT%sResults%d.txt',algName,...
        brainFileLabel,K*factor);
        B(i,:,1) = dlmread(str,' ', 1,0);
```

```
        OMPImage(brain,N,K*factor,K,2,brainNumber);
        str = sprintf('%sFFT%sResults%d.txt',algName,...
        brainFileLabel,K*factor);
        B(i,:,2) = dlmread(str,' ', 1,0);
        OMPImage(brain,N,K*factor*0.5,K*0.5,3,brainNumber);
        str = sprintf('%sDWT%sResults%d.txt',algName,...
        brainFileLabel,K*factor);
        B(i,:,3) = dlmread(str,' ', 1,0);
    end

    if alg == 3
        COSAMPIImage(brain,N,K*factor + 1280,K,1,brainNumber,iter);
        str = sprintf('%sDCT%sResults%d.txt',algName,...
        brainFileLabel,K*factor+K*2);
        B(i,:,1) = dlmread(str,' ', 1,0);
        COSAMPIImage(brain,N,K*factor + 1280,K,2,brainNumber,iter);
        str = sprintf('%sFFT%sResults%d.txt',algName,...
        brainFileLabel,K*factor+K*2);
        B(i,:,2) = dlmread(str,' ', 1,0);
        COSAMPIImage(brain,N,K*factor*0.5 + 1280/2,K*0.5,...
        3,brainNumber,iter);
        str = sprintf('%sDWT%sResults%d.txt',algName,...
        brainFileLabel,K*factor+K*2);
        B(i,:,3) = dlmread(str,' ', 1,0);
    end
end

M_K = [3:0.5:6.5];
width = 60; % cm
height = 30; % cm
%Print Graph

%Methods
if brainNumber == 1
    brainLabel = sprintf('Brain 1');
elseif brainNumber == 2
    brainLabel = sprintf('Brain 2');
end
```

```
%Measurements
for k = 1:3
    figure;
    plot(M_K,B(:,k,1),'b-o')
    hold on;
    plot(M_K,B(:,k,2),'g-o')
    hold on;
    plot(M_K,B(:,k,3),'r-o')
    %Label Graph
    if k == 1
        measureLabel = sprintf('PSNR');
        yAxesLabel = sprintf('PSNR Values');
        nameLabel = sprintf('PSNR');
        locationLabel = sprintf('NorthWest');
    elseif k == 2
        measureLabel = sprintf('Error');
        yAxesLabel = sprintf('Error Values');
        nameLabel = sprintf('Error');
        locationLabel = sprintf('NorthEast');
    elseif k == 3
        measureLabel = sprintf('Elapsed Time');
        yAxesLabel = sprintf('Time(s)');
        nameLabel = sprintf('Time');
        locationLabel = sprintf('NorthWest');
    end
    titleLabel = sprintf('%s of %s Reconstruction via %s
    (N = %d, K = %d)', measureLabel,brainLabel,algName,N,K);
    title(titleLabel);
    ylabel(yAxesLabel);
    legend('DCT','FFT','DWT','Location', locationLabel);
    xlabel('M/K');
    set(gcf, 'PaperPosition', [0, 0, width / 2.54, height / 2.54])
    graphName = sprintf('%s-MRI-%s-%s',algName,...
    nameLabel, brainFileLabel);
    print( '-dpng', '-r500', graphName)
end
```

Bibliography

[1] *A Mathematical Introduction to Compressive Sensing*. Birkhauser, 2013.

[2] Richard Baraniuk. Model-based compressive sensing. `http://people.ee.duke.edu/~lcarin/baraniuk.pdf`, visited on 2016-03-12.

[3] T. Blumensath and M. E. Davies. Iterative hard thresholding for compressed sensing. 2008.

[4] T. Blumensath and M. E. Davies. Sampling theorems for signals from the union of finite-dimensional linear subspaces. *IEEE Transactions on Information Theory (Volume:55 , Issue: 4)*, 2009.

[5] E. Candes and J. Romberg. 1-magic. `http://statweb.stanford.edu/~candes/l1magic/`, visited on 2015-11-21.

[6] E. J. Candes and T. Tao. Decoding by linear programming. *IEEE Transactions on Information Theory (Volume:51 , Issue: 12)*, 2005.

[7] Emmanuel J. Candes and Michael B. Wakin. An introduction to compressive sampling. *IEEE Signal Processing Magazine [21]*, 2008.

[8] Iddo Drori Jean-Luc Starck David L. Donoho, Yaakov Tsaig. Sparse solution of underdetermined linear equations by stagewise orthogonal matching pursuit (stomp). *IEEE Transactions on Information Theory (Volume:58 , Issue: 2), year = 2012,*.

[9] Iddo Drori Jean-Luc Starck David L. Donoho, Yaakov Tsaig. Cosamp: Iterative signal recovery from incomplete and inaccurate samples. 2009.

[10] D. Donoho. Compressed sensing. *IEEE Transactions on Information Theory (Volume:52 , Issue: 4)*, 2006.

[11] D. L. Donoho and P. B. Stark. Uncertainty principles and signal recovery. *SIAM (Volume:49 , Issue: 3)*, 1989.

[12] J. Romberg E. Candes and T. Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on Information Theory (Volume:52 , Issue: 2)*, 2006.

[13] Justin Romberg Emmanuel Candes and Terence Tao. Stable signal recovery from incomplete and inaccurate measurements. *Numerical Analysis, volume = arXiv:math/0503066 [math.NA]*, 2005.

[14] howstuffworks.com. Mri scan steps. `http://science.howstuffworks.com/mri3.htm`, visited on 2016-01-17.

[15] Nuada Medical/Wellcome Images. 3t mri image of a human head with a pituitary tumour. `http://wellcomeimages.org/`, visited on 2015-11-23.

[16] John M. Pauly Michael Lustig, David Donoho. Sparse mri: The application of compressed sensning for rapid mr imaging. *Magnetic Resonance in Medicine*, 2007.

[17] S. Muthukrishnan. *Data Streams: Algorithms and Applications*. Now publishers, 2005.

[18] D. Needell and R. Vershynin. Signal recovery from incomplete and inaccurate measurements via regularized orthogonal matching pursuit. 2007.

[19] D. Needell and R. Vershynin. Uniform uncertainty principle and signal recovery via regularized orthogonal matching pursuit. 2007.

[20] NHS. Mri scan. `http://www.nhs.uk/conditions/MRI-scan/Pages/Introduction.aspx`, visited on 2016-01-02.

[21] Sean Novak. T2-weighted mr image of the brain. `https://commons.wikimedia.org/w/index.php?curid=39617865`, visited on 2015-11-23.

[22] Marco F. Duarte Richard G. Baraniuk, Volkan Cevher and Chinmay Hegde. Model-based compressive sensing. *Information Theory, volume = arXiv:0808.3572v5 [cs.IT]*, 2009.

[23] D. L. Donoho S. S. Chen and M. A. Saunders. Atomic decomposition by basis pursuit. *SIAM Review, 43*, 2001.

[24] J. Tropp and A.C. Gilbert. Signal recovery from partial information via orthogonal matching pursuit. *IEEE Transactions on Information Theory (Volume:53 , Issue: 12)*, 2007.

[25] J. Tropp and A.C. Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Transactions on Information Theory (Volume:53 , Issue: 12)*, 2007.

[26] Wikipedia. Compressed sensing. `https://en.wikipedia.org/wiki/Compressed_sensing`, visited on 2016-01-02.

[27] Wikipedia. Nyquistshannon sampling theorem. `https://en.wikipedia.org/wiki/Nyquist-Shannon_sampling_theorem`, visited on 2016-03-20.

[28] Gitta Kutyniok Yonina C. Eldar. *Compressed Sensing Theory and Applications*, volume ISBN: 9781107005587. 2012.

[29] Jian-Feng Cai Di Guo Zhong Chen Xiaobo Qu Yunsong Liu, Zhifang Zhan. Projected iterative soft-thresholding algorithm for tight frames in compressed sensing magnetic resonance imaging. *Medical Physics*, arXiv:1504.07786 [physics.med-ph], 2015.