7-2016

Compressive sensing and radar imaging

John Montalbo
The University of Texas Rio Grande Valley

COMPRESSIVE SENSING AND RADAR IMAGING

A Thesis

by

JOHN MONTALBO

Submitted to the Graduate College of
The University of Texas Rio Grande Valley
In partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

July 2016

Major Subject: Mathematics

COMPRESSIVE SENSING AND RADAR IMAGING

A Thesis
by
JOHN MONTALBO

COMMITTEE MEMBERS

Dr. Zhijun Qiao
Chair of Committee

Dr. Zhaosheng Feng
Committee Member

Dr. Jasang Yoon
Committee Member

Dr. Andras Balogh
Committee Member

July 2016

ABSTRACT

Montalbo, John, Compressive Sensing and Radar Imaging. Master of Science (MS), July, 2016, 45 pp., 1 table, 34 figures, 24 references, 10 titles.

The field of remote sensing contains many unique and practical problems. Radar imaging, in all of its many forms, lies within this field of study. One problem is the need to acquire high-resolution images and store them on-board the system acquisition vessel. For some systems this could mean storing very high amounts of data, depending on the scene in question [3]. So a natural goal is to store only what is absolutely necessary. We investigate methods to compress signals into their most important components so that other parties can recover the original data completely or nearly completely. We show that certain transformations allow for minimal data acquisition and high reconstruction rate. We also show that the field of compressive sensing offers many tools highly amenable to the problems in question.

DEDICATION

I would like to dedicate this thesis to my wife for her continuous love and support.

ACKNOWLEDGMENTS

I am fully indebted to my thesis advisor Dr. Zhijun Qiao. He has been a source of constant support and mentorship through this entire process. I would also like to thank Dr. Virgil Pierce for his encouraging words and help throughout my time at the university. I would also like to thank Dr. Josef Sifuentes for our many enlightening discussions on these matters. Finally I would like to thank Dr. Cristina Villalobos for her kind guidance throughout my Bachelors and Masters degrees.

TABLE OF CONTENTS

LIST OF FIGURES

Page

CHAPTER I

INTRODUCTION

Modern radar (**RA**dio **D**etection **A**nd **R**anging) systems utilize electromagnetic waves to achieve, usually, two goals [4]. Firstly the system is asked to detect, this means we want to be able to distinguish between noise and objects of interest in the radar's path, then we ask how far away the object detected in question is [4][8]. For more robust radar systems we can acquire signals that are then formed into images of a scene through the system and then we could have someone or something decide what the objects detected by the system actually are.

Once data is acquired we are confronted with some interesting problems. Firstly, what if the system is "far" away from the operator? For example if we have a satellite system that is imaging some scene. The data acquired from the satellite from the scene could be very high [9]. Secondly, when dealing with remote sensing technology there are some very pragmatic reasons not to hold full data on-board. Most notably is the need to conserve power, but also equally troublesome is that the system would always have a finite amount of memory. For example the Mars Curiosity rover is equipped with an on-board memory that is 256 MB of DRAM and 2 GB of Flash Memory [10]. Staying with satellite imaging for the moment, even though there have been major improvements in battery technology and solar technology every piece of equipment on the system is asking for wattage and should we store full data then that is just more of a burden [4].

We investigate ways to sample signals and images from their full data, hold the most crucial components of the full data on the system, delete all but those selected values and then send off the selected data. Once received the operator may then use the selected data to reconstruct the actual scene or signal.

We achieve some results using what we call a Fourier Thresholding Selection Algorithm

(FTSA) and we show that a signal can be approximately reconstructed using only minimal Fourier data. We also show that chirp signals may also be run through the same process with similar results. As an aside we also demonstrate that a Fast-Walsh Hadamard Transformation will also produce similar results with minimal data. We conclude the paper by using ℓ_1-minimization techniques to do the reconstruction process and show that the compressive sensing scheme offers low error reconstruction with small sample in the frequency space.

CHAPTER II

INTRODUCTION TO MIMO RADAR SYSTEMS

There are many ways to both transmit and acquire electromagnetic waves for the sensing process, for example Single Input and Single Output (SISO) systems have only one output frequency that is broadcast for the detection process [2]. These are the simplest systems to study and is illustrated in the following way:

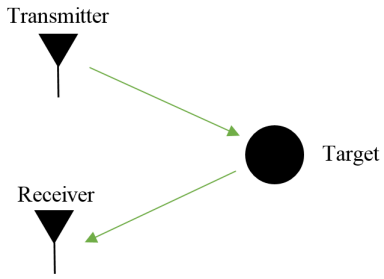


Figure 2.1: SISO System with Separate Transmitter and Receiver Arrays

The carrier signal is sent out and perhaps comes in contact with a target of interest, the sent wave is then scattered and sent back and hopefully detected by the receiver. The scenario could be that the transmitter and receiver are the same component within the radar system, but the idea still stays the same. For a multiple input-multiple output radar system (MIMO) the scheme is you have many transmitters and many receivers. These transmitters could be all broadcasting on the same frequency or could be transmitting on staggered frequencies. This can be illustrated by figure (2).

This is the scenario that we consider in this work. For our purposes we assume that the radar system is a MIMO system where the broadcast frequencies is different among all transmitters within the array. More specifically we are concerned with the data acquired at the receiver array and how to compress the data acquired at that point. For the sake of concreteness modern radar systems

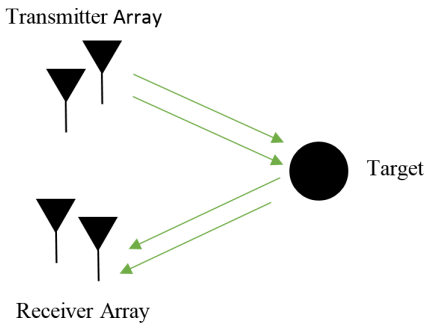


Figure 2.2: MIMO System with Separate Transmitter and Receiver Arrays

operate within a certain bandwidth of the radio and microwave spectrum usually between 3 MHz to 300 GHz [9] as illustrated as follows:

Table 2.1: Operating Frequencies

Bandwidth Designation	Frequency
HF - High Frequency	3 - 30 MHz
VHF - Very High Frequency	30 - 300 MHz
UHF - Ultra High Frequency	300 - 1000 MHz
L Band	1 - 2 GHz
S Band	2 - 4 GHz
C Band	4 - 8 GHz
X Band	8 - 12 GHz
Ku Band (under K)	12 - 18 GHz
K Band	18 - 27 GHz
Ka Band (above K)	27 - 40 GHz
MM - Wave	40 - 300 GHz

As an aside one could estimate the range of a target from a combined transmitter-receiver pair through the following; let d be the distance from the target to the system, and let T be the time from sending the signal until detection. Then since the transmitted wave velocity is close to the

speed of light c then:

$$2d = c \cdot T \quad \rightarrow \quad d = \frac{c \cdot T}{2} \tag{2.1}$$

When the continuous waveform is at the receiver there is usually an analog (continuous) to digital (discrete) converter that samples the signal either uniformly of non-uniformly to take the full continuous waveform and discretize it [6]. For example consider this sinusoid and its discrete transformation:

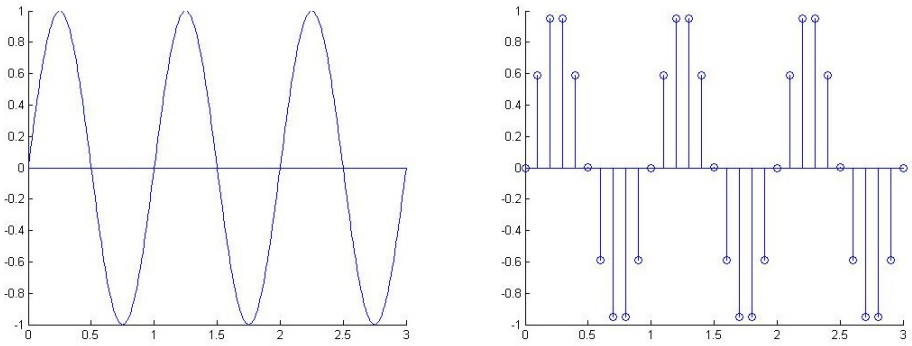


Figure 2.3: Analog to Digital Sine Wave

Sinusoids are common carrier waves for many radar systems [4] [6] [1]. So a natural question to ask is how much should we sample at the receiver stage such that we can be sure we can reconstruct the full data? To answer this we have the Shannon-Nyquist Sampling Theorem [1]

Shannon-Nyquist Sampling Theorem For a band limited signal the sampling frequency should be at or higher than twice the highest frequency in the signal. That is:

$$F_s \geq 2 \cdot \omega_{max} \tag{2.2}$$

Where ω_{max} is the largest frequency in the signal.

For example in the sinusoid below we have graphed the function $y = \sin(2\pi t)$, which means that its frequency is $1Hz$. Shannon-Nyquist says that we should at $2Hz$, which means that we should sample every $1/4$ unit of time which means taking 5 samples in total over 1 unit of time. We have

displayed above the results of sampling 10 times per unit time. Let us show this all together in figure (5).

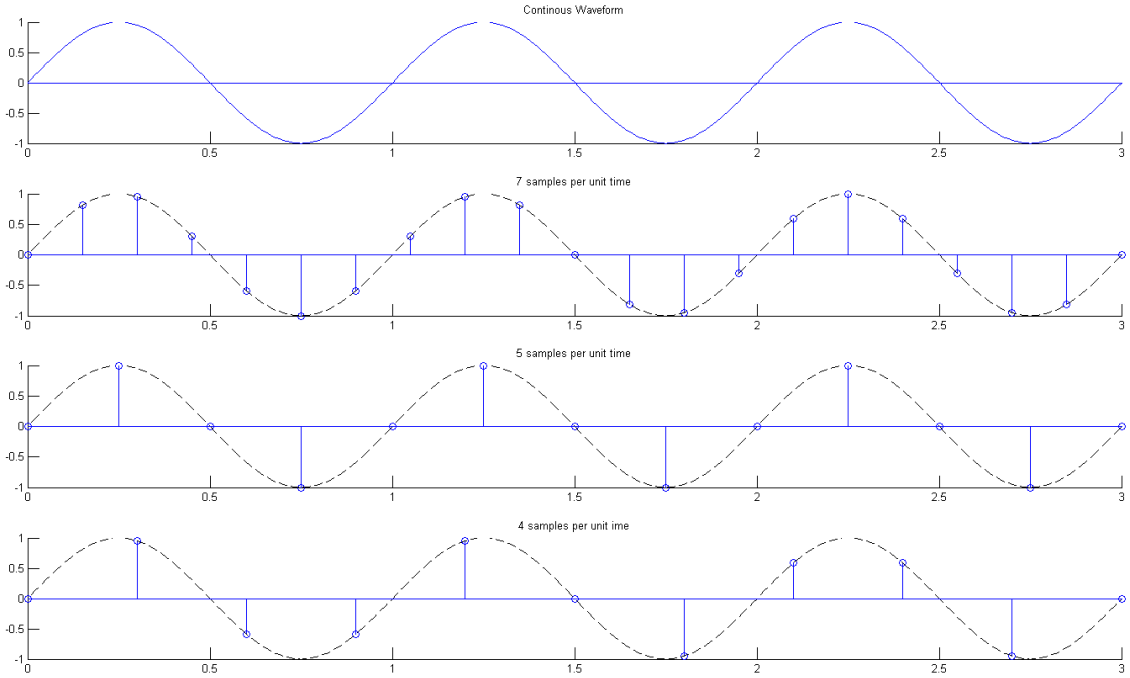


Figure 2.4: Sampling the Analog Signal at Different Rates

We can see that when we are taking 5 samples per unit time we get the highs,lows, and roots of the signal. If we increase the sample rate to 7 samples per unit time we get all the previous information plus data about the intermediate regions. On the other hand, when we dip below the sampling rate we no longer get any useful information from the samples.

So the SNST gives us a benchmark about how many pieces of data to take for a signal. Now when the frequency is small this is not necessarily a large amount of data. For example our signal $y(t) = \sin(2\pi t)$ in Matlab is stored in a 1×300 vector that was interpolated in MatLab to display the figure. Here is the stored data for the samples:

Figure	Samples	% of orginal
1	301	100 %
2	21	7 %
3	13	4.32 %
4	11	3.66 %

This illustrates just how sensitive some systems can be to under-sampling by even a minute amount. But we still have a clear goal in mind; to store information in the most succinct form possible.

CHAPTER III

MAXWELL'S EQUATIONS

3.1 THE SCALAR WAVE EQUATION

Since we are dealing with electromagnetic resonance from the transmitting source we now turn our attention to Maxwell's equations. These are the four most influential equations in all of science and their study allows for almost the entirety of modern technology [4]. These equations are:

- Faraday's Law
- Ampere's Law
- Gauss's Law
- Gauss's Law for Magnetism

These four laws when combined with the wave equation are the basis for the electromagnetic theory of light [7]. Some interesting things to note is that originally Maxwell publishes 20 equations! These are then taken later by Oliver Heavyside and Hienrich Hertz and they are condensed into the four we present now.

$$\nabla \times \mathscr{E} = -\frac{\partial \mathscr{B}}{\partial t} \tag{3.1}$$

$$\nabla \times \mathscr{H} = \mathscr{J} + \frac{\partial \mathscr{D}}{\partial t} \tag{3.2}$$

$$\nabla \cdot \mathscr{D} = \rho \tag{3.3}$$

$$\nabla \cdot \mathscr{B} = 0 \tag{3.4}$$

Where $\mathscr{E}(\mathbf{x},t)$ is the electric field, $\mathscr{B}(\mathbf{x},t)$ is the magnetic induction field, $\mathscr{D}(\mathbf{x},t)$ is the electric displacement field, $\mathscr{H}(\mathbf{x},t)$ is the magnetic field and $\rho(\mathbf{x},t)$ is the charge density and $\mathscr{J}(\mathbf{x},t)$ is the current density.

Furthermore we have ε_0 will be the permutivity of free space and μ_0 is the permeability of free space.

When we take the curl of (3) we have:

$$\nabla \times (\nabla \times \mathscr{E}) = -\nabla \times (\frac{\partial}{\partial t}\mathscr{B}) = -\frac{\partial}{\partial t}(\nabla \times \mathscr{B}) \tag{3.5}$$

$$\nabla(\nabla \cdot \mathscr{E}) - \nabla^2 \mathscr{E} = -\frac{\partial}{\partial t}(\nabla \times \mathscr{B}) \tag{3.6}$$

We make the following reductions on the left hand side of the above with the relations:

$$\mathscr{D} = \varepsilon_0 \mathscr{E} \tag{3.7}$$

$$\mathscr{B} = \mu_0 \mathscr{H} \tag{3.8}$$

Along with the free space conditions $\mathscr{J}(\mathbf{x},t) = 0$ and $\rho(\mathbf{x},t) = 0$:

$$\nabla \cdot \mathscr{E} = \nabla \cdot (\frac{1}{\varepsilon_0}\mathscr{D}) = \frac{1}{\varepsilon_0}(\nabla \cdot \mathscr{D}) = 0 \tag{3.9}$$

$$\nabla \times \mathscr{B} = \nabla \times (\mu_0 \mathscr{H}) = \mu_0(\frac{\partial \mathscr{D}}{\partial t}) = \mu_0 \left(\frac{\partial}{\partial t} \varepsilon_0 \mathscr{E} \right) \tag{3.10}$$

Returning to (8) with these substitutions we have:

$$-\nabla^2 \mathscr{E} = -\frac{\partial}{\partial t} \left(\mu_0 \varepsilon_0 \frac{\partial \mathscr{E}}{\partial t} \right) \tag{3.11}$$

$$\nabla^2 \mathscr{E} = \mu_0 \varepsilon_0 \frac{\partial^2 \mathscr{E}}{\partial t^2} \tag{3.12}$$

Simply using the fact that for our constant wave speed $c = \frac{1}{\sqrt{\mu_0 \varepsilon_0}} \approx 3.00 \times 10^8$ m/sec we then have:

$$\nabla^2 \mathscr{E} = \frac{1}{c^2} \frac{\partial^2 \mathscr{E}}{\partial t^2} \tag{3.13}$$

$$\left(\nabla^2 - \frac{1}{c^2} \frac{\partial^2}{\partial t^2} \right) \mathscr{E} = 0 \tag{3.14}$$

3.2 THE LIPPMAN-SCHWINGER INTEGRAL EQUATION

If we regard the electric field as the cumulative total of the incident wave and the scattered waves received at the transmitter, i.e.:

$$\mathscr{E}^{\text{Total}} = \mathscr{E}^{\text{In}} + \mathscr{E}^{\text{Sc}} \tag{3.15}$$

As well as regarding our speed of wave propagation as a perturbation of the speed of light and some reflectivity function $V(\mathbf{x})$:

$$\frac{1}{c^2(\mathbf{x})} = \frac{1}{c^2} - V(\mathbf{x}) \tag{3.16}$$

We arrive at:

$$\left(\nabla^2 - c^{-2}(\mathbf{x}) \partial_t^2 \right) \mathscr{E}^{\text{Tot}} = j(\mathbf{x}, t) \tag{3.17}$$

Making substitutions we have:

$$\left(\nabla^2 - (c^{-2} - V(\mathbf{x})) \partial_t^2 \right) \mathscr{E}^{\text{Tot}} = j(\mathbf{x}, t) \tag{3.18}$$

$$\left(\nabla^2 - c^{-2} \partial_t^2 + V(\mathbf{x}) \partial_t^2 \right) \mathscr{E}^{\text{Tot}} = j(\mathbf{x}, t) \tag{3.19}$$

Now we use the fact that the incident field would still satisfy (20):

$$\left(\nabla^2 - c^{-2}\partial_t^2\right)\mathscr{E}^{\mathrm{In}} = j(\mathbf{x},t) \tag{3.20}$$

Making the substitution:

$$\beta = \nabla^2 - c^{-2}\partial_t^2 \tag{3.21}$$

And substituting the $\mathscr{E}^{\mathrm{In}}$:

$$\beta(\mathscr{E}^{\mathrm{Tot}} - \mathscr{E}^{\mathrm{Sc}}) = j(\mathbf{x},t) \tag{3.22}$$

$$\beta\mathscr{E}^{\mathrm{Tot}} - \beta\mathscr{E}^{\mathrm{Sc}} = j(\mathbf{x},t) \tag{3.23}$$

Substituting β in (21) we have:

$$\beta\mathscr{E}^{\mathrm{Tot}} + V(\mathbf{x})\partial_t^2\mathscr{E}^{\mathrm{Tot}} = j(x,t) \tag{3.24}$$

Simply subtracting (25) and (26) we have:

$$-\beta\mathscr{E}^{\mathrm{Sc}} - V(\mathbf{x})\partial_t^2\mathscr{E}^{\mathrm{Tot}} = 0 \tag{3.25}$$

$$\beta\mathscr{E}^{\mathrm{Sc}} = -V(\mathbf{x})\partial_t^2\mathscr{E}^{\mathrm{Tot}} \tag{3.26}$$

$$\left(\nabla^2 - c^{-2}\partial_t^2\right)\mathscr{E}^{\mathrm{Sc}} = -V(\mathbf{x})\partial_t^2\mathscr{E}^{\mathrm{Tot}} \tag{3.27}$$

This results in the Lippmann-Schwinger integral equation with Green's function $g(t,\mathbf{x})$:

$$\mathscr{E}^{\mathrm{Sc}}(\mathbf{x},t) = \int\int g(t-\tau,\mathbf{x}-\mathbf{z})V(\mathbf{z})\partial_\tau^2\mathscr{E}^{\mathrm{Tot}}(\tau,\mathbf{z})d\tau d\mathbf{z} \tag{3.28}$$

With Green's function as:

$$g(t-\tau,\mathbf{x}-\mathbf{z}) = \frac{\delta(t-\tau-\frac{|\mathbf{x}-\mathbf{z}|}{c})}{2\pi|\mathbf{x}-\mathbf{z}|}$$

where δ is the Kronecker delta function.

CHAPTER IV

SIGNALS AND COMPRESSION

4.1 THE FOURIER THRESHOLDING SELECTION ALGORITHM (FTSA)

We turn our attention to the problem of compressing a signal into its most important components. We begin our study by considering a family of high frequency signals and their Fourier transformations. For example here we have a high frequency sine wave:

$$y = \sin(10^4 \cdot t) \tag{4.1}$$

where $t \in [0, .5]$ and each step size is .001 so as to try and avoid aliasing.

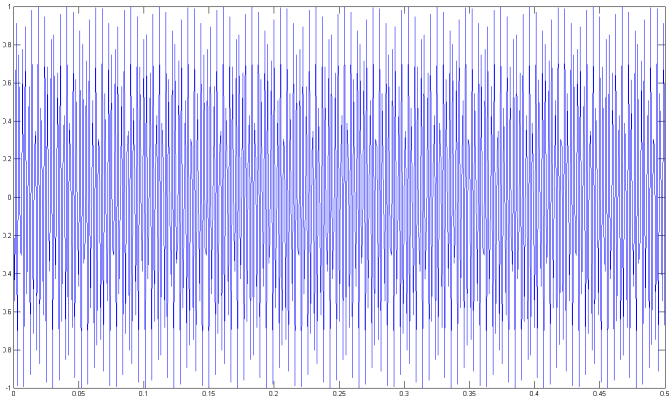


Figure 4.1: Graph of $f(t) = \sin(10^4 \cdot t)$

There is a lot of information that is being displayed, but since we have periodicity we are actually viewing more than is necessary. To illustrate this let's look at the Fourier transform of the above function, we will display this in figure (7).

Now we can see that in frequency space we have a non-erratic figure of four delta spikes. But we are only seeing one side of the picture. Because the Fourier transform could be complex we

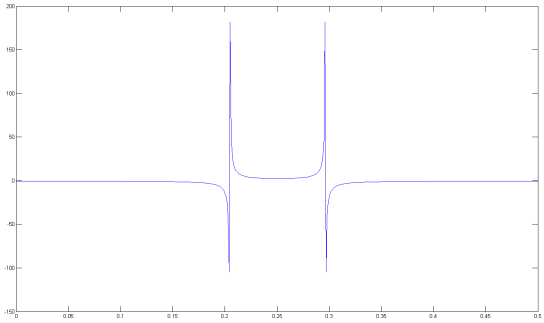


Figure 4.2: Fourier Transform of $f(t) = \sin(10^4 \cdot t)$

are currently only seeing the real part of the FFT. Here we will highlight the real part of the FFT with red and let the imaginary part be black, this will be in figure (8).

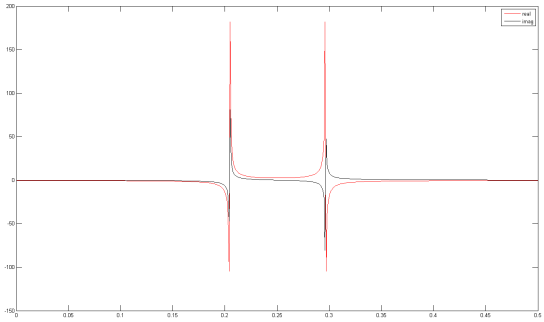


Figure 4.3: Real and Imaginary part of the Fourier Transform

This representation in the frequency domain is not quite sparse. It has many entries that are close to zero, but what if we could get a sparse representation of $f(t)$ from this data. To this end we use a Fourier Thresholding Algorithm to select the most perceptibly large Fourier coefficients and store them. Then we will do the inverse Fourier transform on those store values and then look at the difference between the original signal and the signal with limited Fourier data.

As we continue we will still ask for our signals to have high frequency and will limit ourselves to sinusoids for the moment. We give a block diagram for the algorithm in figure (9).

For the forward Fourier transform we let our continuous signal $f(t)$ for $t \in T$ where T is

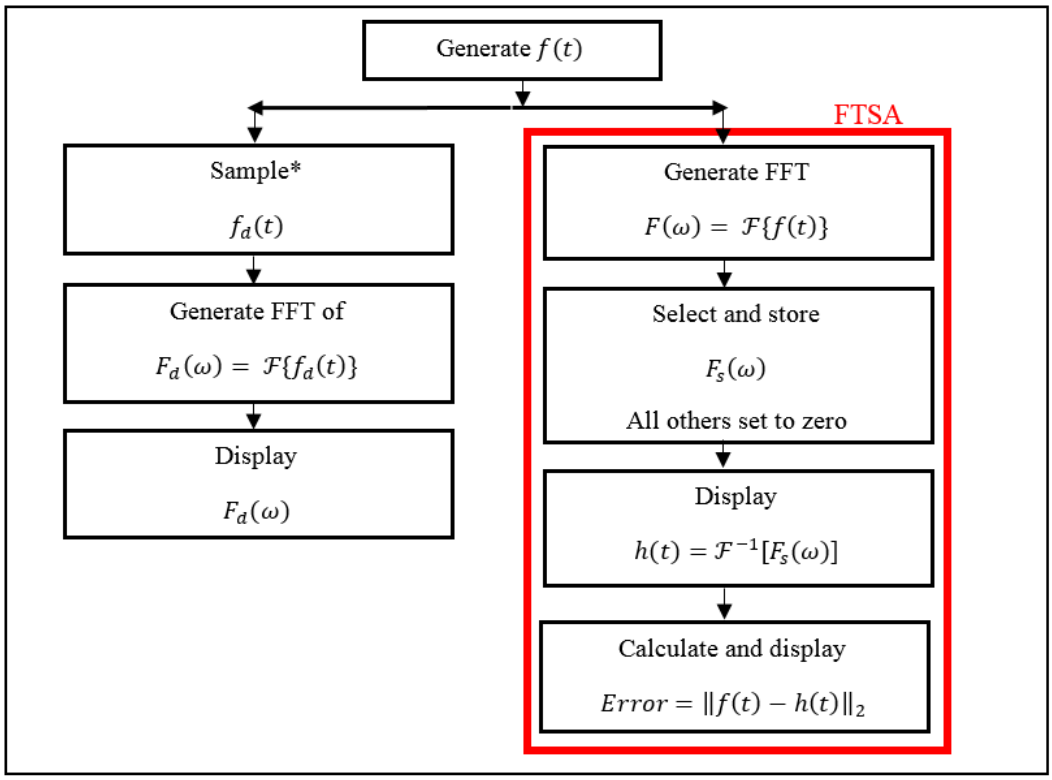


Figure 4.4: Block Diagram for the FTSA Process

some bounded domain and let:

$$\mathscr{F}\{f(t)\} = \mathscr{F}(\omega) = \int e^{i\omega t} f(t) dt \tag{4.2}$$

be the forward operation and:

$$\mathscr{F}^{-1}(\mathscr{F}\{f(t)\}) = f(t) = \int e^{-i\omega t} \mathscr{F}(\omega) d\omega \tag{4.3}$$

will be the inverse operation. In practice we will be using the discrete analog of the Fourier Transform which is the Fast Fourier Transform, which uses a separate and conquer approach to transform the original vector into the frequency domain [1].

One question to ask is "how should we do the thresholding so as to minimize error?" For this we choose a simple linear search for the maximum value (say Ω) in the Fourier space $\mathscr{F}\{f(t)\}$. Once this value is found we then specify a $K \in (1, \Omega)$ such that we store any points in the range

space that are above the threshold of Ω/K. Any other values that are below our thresholding value will be sent to zero.

Once the selection process is done we will have a sparse representation for $\mathscr{F}(\omega)$ through $z(\omega)$. Once we have these stored values we simply do the inverse Fourier transform on those values and record the error between the original signal and the new one.

Let us walk through a few examples. We will be using the following family of sinusoids:

$$f(t) = \sum_{i=1}^{J} a_i \sin(c_i t) \tag{4.4}$$

We typically call this family of sines a sine train. Also we will typically only take up to five terms so as to try and avoid any aliasing that might occur. We will also ask that the a_i amplitude form a monotonically increasing sequence and we will also want the frequency to be close to 300 GHz, although if we are over that limit we show that the filtering process is applicable to higher frequencies signals that might be common in thermal imaging.

Let $J = 5$ with:

$a_1 = 10^1$	$r_1 = \text{randi}([1,10],1,1)$	$c_1 = a_1 \cdot r_1^{r_1}$
$a_2 = 10^2$	$r_2 = \text{randi}([1,10],1,1)$	$c_2 = a_2 \cdot r_2^{r_2}$
$a_3 = 10^3$	$r_3 = \text{randi}([1,10],1,1)$	$c_3 = a_3 \cdot r_3^{r_3}$
$a_4 = 10^4$	$r_4 = \text{randi}([1,10],1,1)$	$c_4 = a_4 \cdot r_4^{r_4}$
$a_5 = 10^5$	$r_5 = \text{randi}([1,10],1,1)$	$c_5 = a_5 \cdot r_5^{r_5}$

We will simultaneously sample non-uniformly throughout the time domain so as to show the results. We also want these values so as to use them as a precondition for the compressive sensing algorithms to be used later. Our sampling algorithm will be:

1. Start at entry $e = \text{randi}([1,10],1,1) \in \hat{y}$ where $\dim(\hat{y}) = 1 \times c$.

2. From initial starting point move with step size $e := e + k$ with $k = \text{randi}([1,10],1,1)$.

3. Repeat this process until $e > c$

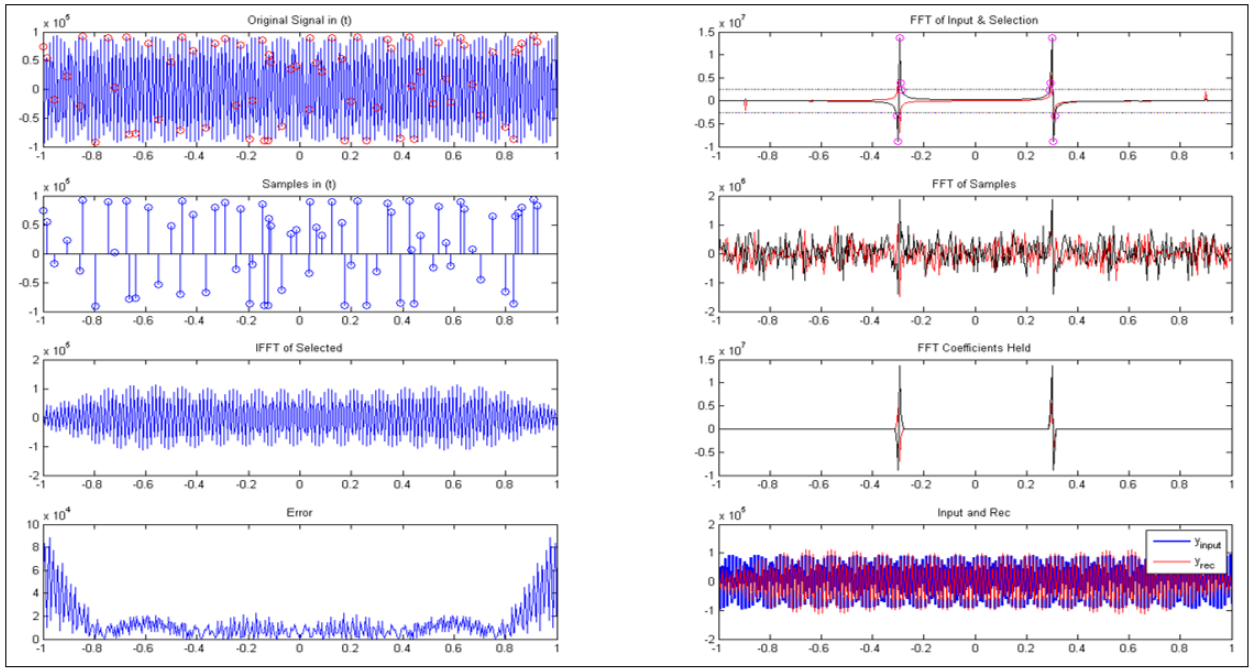
We will use the following notation:

$\lvert y \rvert$	the number of values in the time domain
$\lvert y_s \rvert$	the number of values selected in time from NUS
$\lvert F \rvert$	the number of selected Fourier coefficients
r_i	the random numbers generated

Example 1

So for example here is a plot of:

$$f(t) = 10\sin(10 \cdot 9^9 t) + 10^2 \sin(10^{12} t) + 10^3 \sin(10^3 \cdot 2^2 t) + 10^4 \sin(10^{14} t) + 10^5 \sin(10^5 \cdot 7^7 t) \tag{4.5}$$



$r_i = \{9, 10, 2, 10, 7\}$ $\quad |y| = 400$ $\quad |y_s| = 67$ $\quad Samp\% = 16.75\%$ $\quad |F| = 10$

Figure 4.5: Results for Example 1

On the top left we have the original signal, the red dots are the points that are going to be sampled using the non-uniform sampling algorithm outlined above. On the top right we have the Fourier transform of the original signal. Also we have highlighted in pink the Fourier coefficients

that have been selected with a $K = 6$. The horizontal lines in the graph show the regions that meet the criteria. Also in the top right graph we show the real part of the FFT in red and the imaginary part of the FFT in black.

In the second row from the top we have the time samples on the left along with the FFT of the time samples. Here we can stop and observe that the peaks of the FFT of the time samples and the original coincide very well. Although the smaller Fourier coefficents are lost. I am currently thinking of a way to use these values in a general case.

In the third row on the right hand side we have the held Fourier coefficients from the top. We can see that the smaller Fourier coefficients are no longer present. On the left hand side we have the inverse Fourier transform of the held Fourier coefficient. At first glance we may think that things may have gone wrong because the first plot and the third in the left columns are different. But if we look in the last row we can see that things have turned out very well.

In the last row we have the error $E(t) = \|f(t) - h(t)\|$ where y is the original signal and $h(t)$ is the inverse Fourier transform of the selected Fourier coefficients. We can see that the error is on the magnitude of the original signal. We can also take note that the error in the middle part of the two signals is very low but as we get closer to the sides we start loosing structure.

In the last row on the right hand side we have the graph of both $f(t)$ and $h(t)$ and we can see that they correspond very nicely in the middle but lose each other on the ends.

Before going on to another example I think its important to stress two things. a) The original signal was a vector of dimension 1×400 most of these values are non-zero. From these we get Fourier data with dimension 1×400, these values are again non-zero and complex. From these we do our selection process and end up with a subset of Fourier data with dimensions 1×400, however in this instance only 10 entries in that vector are not zero! With only 2.5 % of the Fourier data we are able to reconstruct the original signal. This is better than the non-uniform time data, which samples 16.75 % of the original time data.

The other thing I like to mention is that if we were to instead use the FTSA on the Fourier data from the NUS data we would have much more data because more and more coefficients would

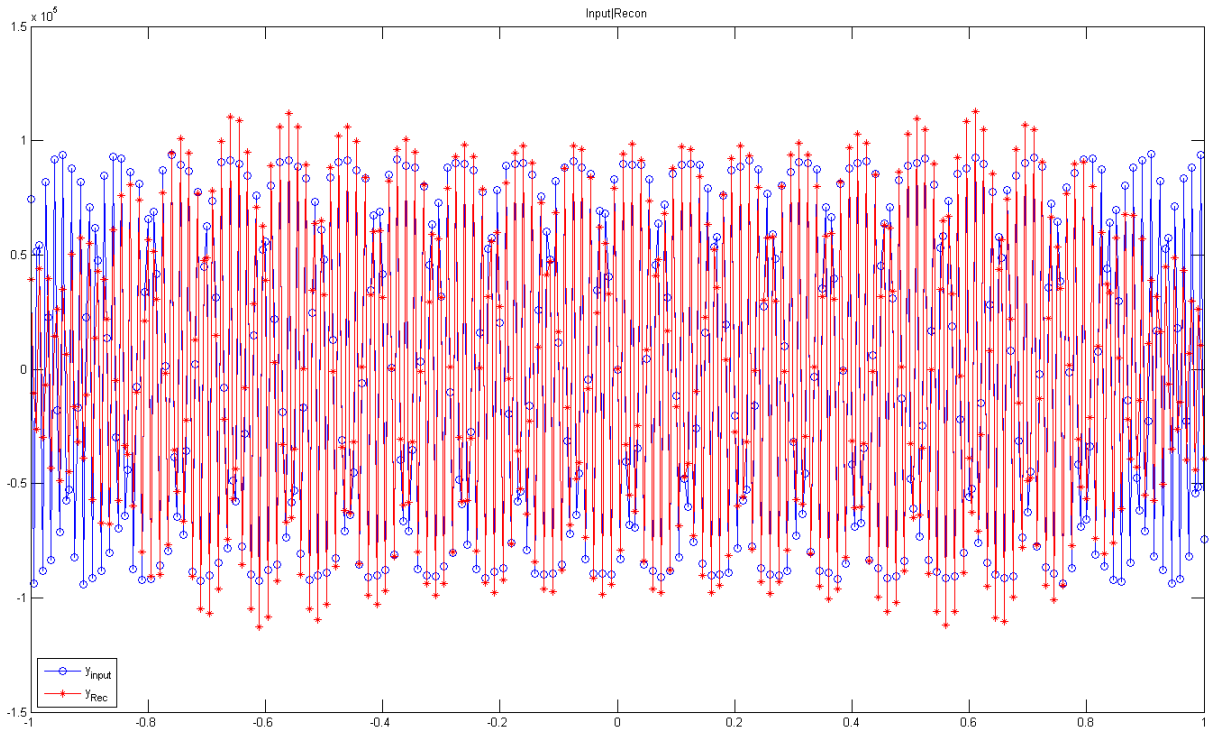


Figure 4.6: Original Signal and Reconstruction

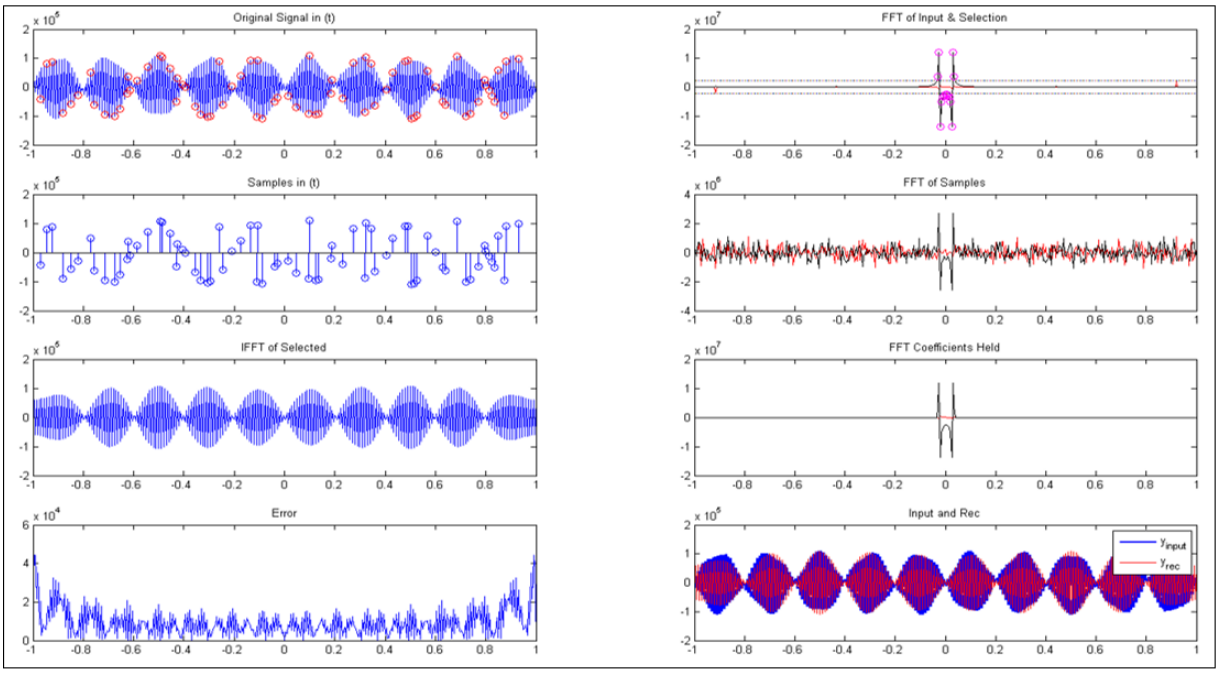
meet our criteria. I should also point out that in the MatLab code that I have written the user is also given the option to hear the original signal and the reconstructed signal one after another. Most of the time these signals sound like a dial-up modem but sometimes they have really interesting sounds.

Let's take a look at another example.

Example 2

$$f(t) = 10\sin(10 \cdot 7^7 t) + 10^2 \sin(10^2 \cdot 4^4 t) + 10^3 \sin(10^3 \cdot 4^4 t) + 10^4 \sin(10^4 t) + 10^5 \sin(10^5 \cdot 6^6 t) \tag{4.6}$$

I have to admit that this is probably my favorite from this family. Again in the first row first column we have the original time signal with the non-uniform sampled entires highlighted in red. In total 75 pieces of time data are sampled. In the first row second column we have the FFT of the time signal along with the threshold and the values selected for storage. In total 14 Fourier coefficients have been held.



$r_i = \{7, 4, 4, 1, 6\}$ $|y| = 400$ $|y_s| = 75$ $Samp\% = 18.75\%$ $|F| = 14$

Figure 4.7: Results for Example 2

In the second row, first column we have the time sampled points and their FFT to their immediate right. And in the third row, first column we have the reconstructed time signal from the 14 sampled Fourier coefficients. And in the last row and first column we again see that the error is again small in the interior of the signal and as we get closer to the boundaries we start to lost accuracy. And in the last row, last column we have the two signals pictured on top of one another.

Here is a close up of the final plot:

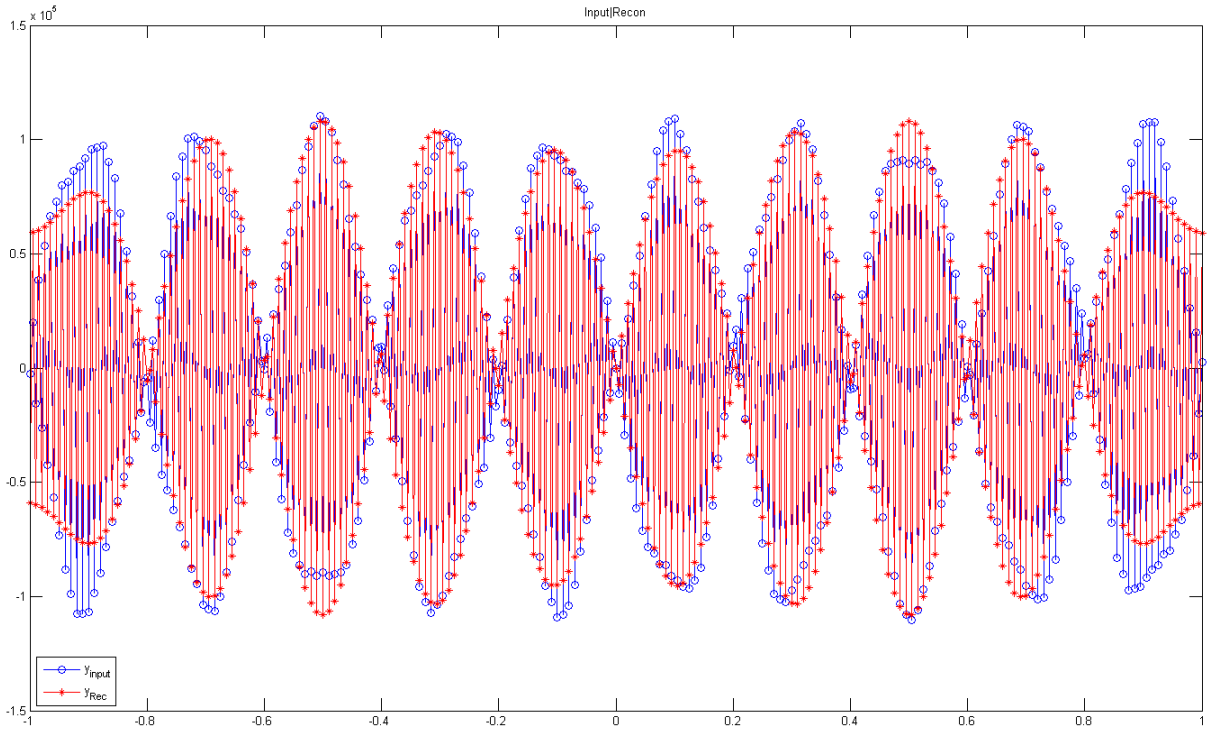


Figure 4.8: Original Signal and Reconstruction

4.2 INTRODUCING NOISE AND LIMITATIONS

We will now investigate the limitations of this algorithm and show that in the presence of noise more and more terms meet our threshold and we begin to have more and more non-zero terms in our compressed sparse representation of the original signal. We will be looking at the same family as before:

$$f(t) = \sum_{i=1}^{J} a_i \sin(c_i t) + \eta(t) \tag{4.7}$$

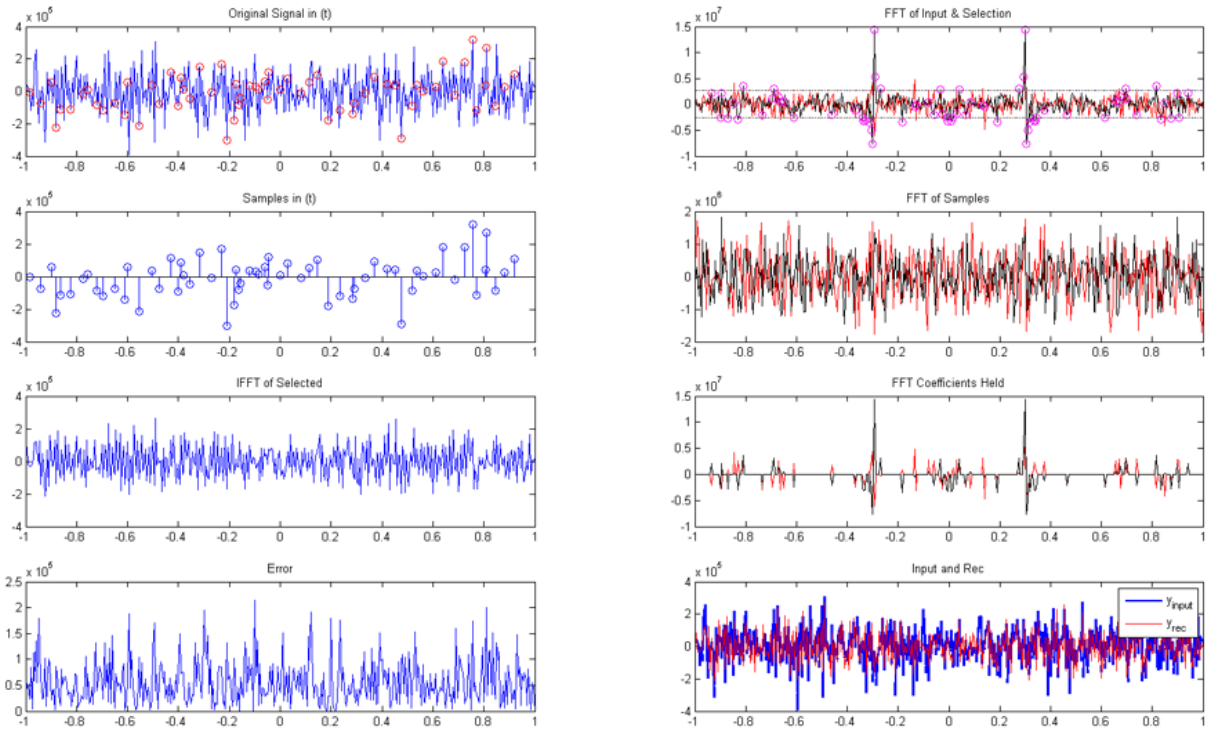
Where $\eta(t)$ is white Gaussian noise.

Example 1 (Noise)

We re-run the same simulation with a new noise term and we arrive at:

$$f(t) = 10\sin(10\cdot 9^9 t) + 10^2\sin(10^{12}t) + 10^3\sin(10^3\cdot 2^2 t) + 10^4\sin(10^{14}t) + 10^5\sin(10^5\cdot 7^7 t) + \eta(t) \tag{4.8}$$

Turning our attention to the first row second column we can see that more Fourier coefficients are now meeting our threshold criteria. Before when we had the noiseless signal we were only



$r_i = \{9, 10, 2, 10, 7\}$ $|y| = 400$ $|y_s| = 63$ $Samp\% = 15.7\%$ $|F| = 68$

Figure 4.9: Graph of $f(t)$ with Noise Term

selecting 10 entries, now we are selecting 68. Which is not too bad considering it's a compression rate of 17%. Plus we are doing much better than the non-uniform time samples that are sampling at a slightly smaller rate of 15.7 %.

If we zoom in and compare the original signal and the reconstructed we see that we are not terribly off and our threshold is able to coincide frequently with the noisy signal. However when we plot 1) the original signal 2) the noisy signal 3) and the decompressed signal we see that we haven't done anything in terms of de-noising the noisy signal with this method.

The take away is that in the presence of white Gaussian noise we are able to still compress the original signal down to a fraction of itself. But we would need other methods for the de-noising process.

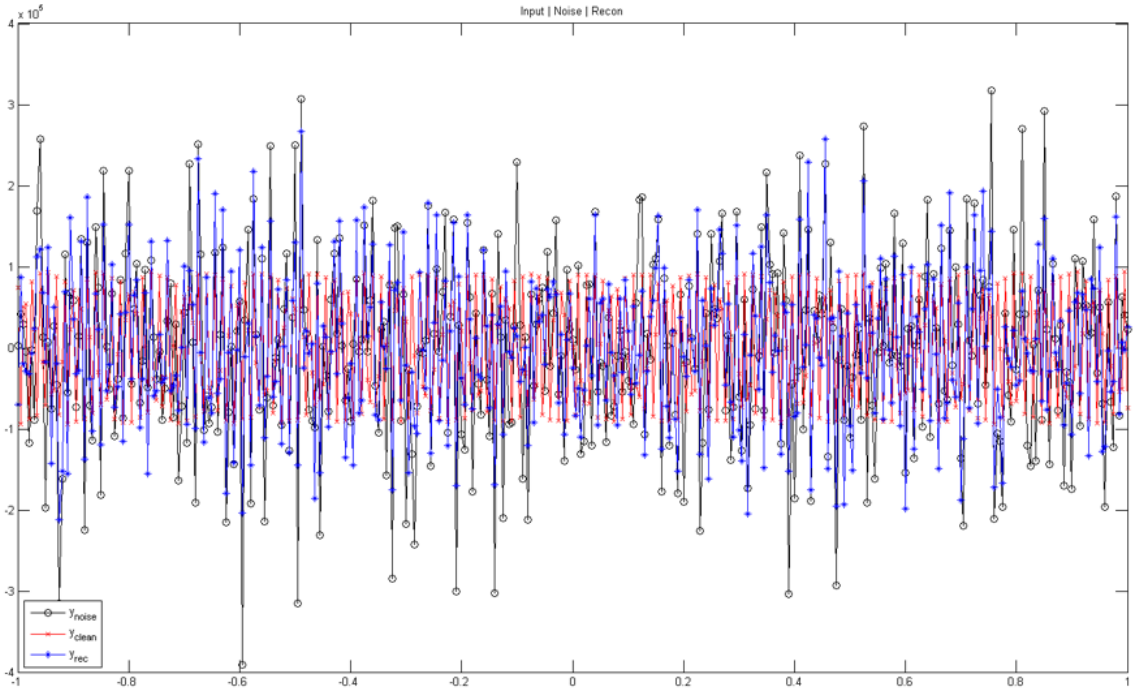


Figure 4.10: Graph of $f(t)$ W/O Noise Term, $f(t)$ W Noise Term, and the Reconstucted $f(t)$

4.3 WALSH-HADAMARD SELECTION

We now ask if it is possible to use another transform, besides Fourier, and obtain similar results. To this end we will employ a Fast Walsh-Hadamard transform to our original signals and see if we can get similar results. The Walsh-Hadamard Transform of a signal $\hat{f}(t) = \hat{y}$ of size $N = 2^n$ is the product of the vectored signal with the appropriately sized Hadamard matrix. That is:

$$\mathrm{WHT}_N = \bigotimes_{i=1}^{n} H_2 = \underbrace{H_2 \otimes \cdots \otimes H_2}_{n-\text{times}} \tag{4.9}$$

Where $H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$ and $\otimes$ is the tensor product, sometimes called the Kronecker product. For the tensor product we simply allow the first matrix to "step into" the second and multiply inside. For example:

$$\mathrm{WHT}_4 = H_2 \otimes H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{4.10}$$

$$= \begin{bmatrix} 1 \cdot \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} & 1 \cdot \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \\ 1 \cdot \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} & -1 \cdot \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \tag{4.11}$$

$$= \begin{bmatrix} H_2 & H_2 \\ H_2 & -H_2 \end{bmatrix} \tag{4.12}$$

So if we have a short signal $\hat{y}$ with dimension 1×4 then the Walsh-Hadamard transformation of $\hat{y}$ will be:

$$\mathrm{WHT}_4(\hat{y}^T) = (H_2 \otimes H_2)(\hat{y}^T) = H_4(\hat{y}^T) \tag{4.13}$$

There are a few interesting things about this transformation; these matrices were first studied by Sylvester (1867) [5] and later further studied by Hadamard (1893)[5]. Hadamard matrices also have a link to Walsh functions which can be regarded as the discrete version of a sine wave. Also interesting is the fact that NASA used WFT transformations to do image compression for the lunar images. The reason for using the WHT instead of a FFT was because this was a time before the FFT had the first F [1]. The FFT was made fast later by engineers at Bell Labs.

Example 1 (Revisited)

We again take a look at the sine waves under a WHT. We still use the same selections for our random integers as before in the first example. However one thing that needed to be changed was the length of the original signal vector. We originally used a vector with dimensions 1×400 but we are now in need of a power of 2 for the signal length. We choose a signal length of 256 rather than 512, but the ideas remain the same regardless.

As before we still allow the non-uniform sampling to attempt to capture the original signal. One important thing to note about this transform is that all of the coefficients are real, so even though

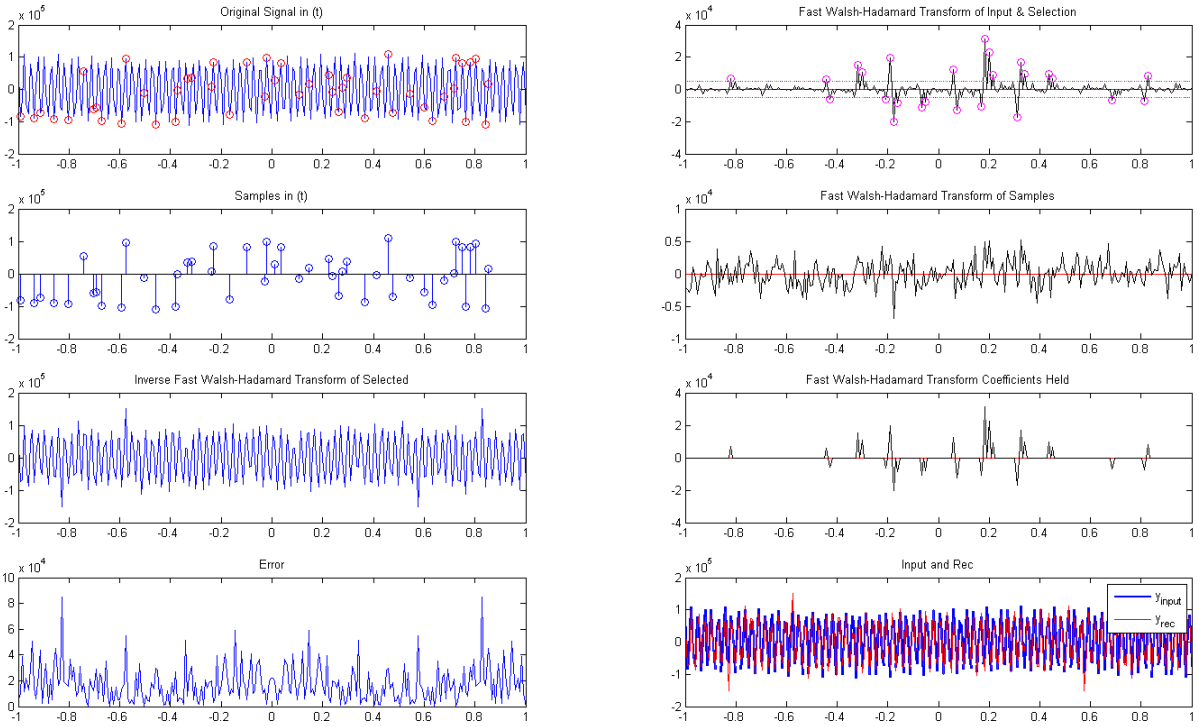


Figure 4.11: Example 1 with Walsh-Hadamard Thresholding Selection

we display the real and imaginary parts together we only need the real parts. In the Hadamard selection we have 25 pieces of data that meet our threshold, still keeping our $K = 6$. This amounts to $(25/256) \cdot 100 = 9.76\%$ of the available data. This is somewhat worse than the results we were getting from the FFT selection. We are taking more of these Hadamard coefficients and getting a higher relative error as illustrated in the last plot of the first column.

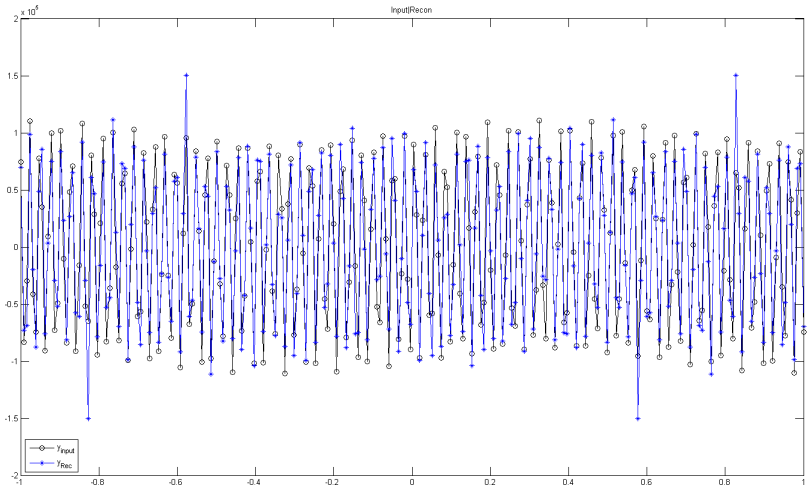


Figure 4.12: Example 1 Original Signal and Reconstructed Signal

Example 2 (Revisited)

Taking another look at example 2 from the first section we see that we are still getting a very nice representation of the original signal from this thresholding. Here we have the original signal with dimensions 1×256 and 42 time samples from the NUS. We are also only selecting 8 Hadamard coefficients. This accounts for only 3.25% of the available data in the transformed space. We don't seem to be getting the same small error like we were when using the Fourier data. But when we look at the two signals on top of one another we see that they indeed match up very frequently.

As before we can see that the thresholding selection is doing well. One thing to also note is that even though the Walsh-Hadamard transform may have higher error than the FFT it does out preform the FFT around the edges. With the FFT we would have really good results in the interior of the signal but would lose the edges. Here we can see that the edges using the WHT are preserved.

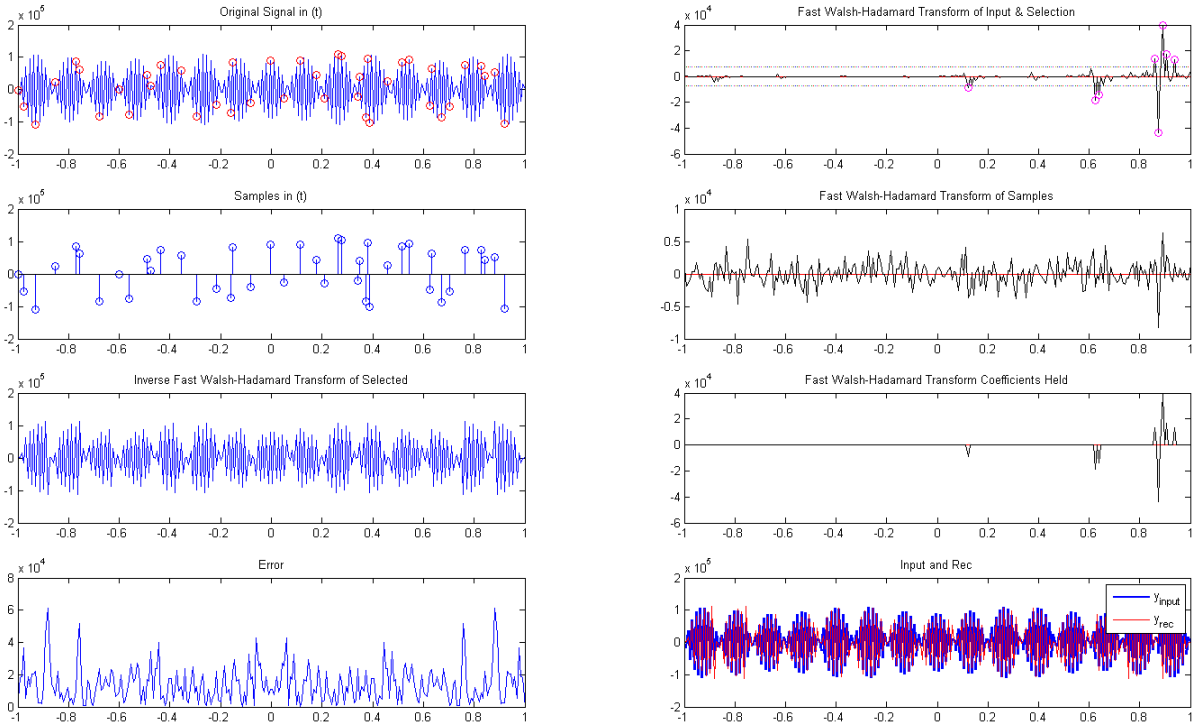


Figure 4.13: Example 2 with Walsh-Hadamard Thresholding Selection

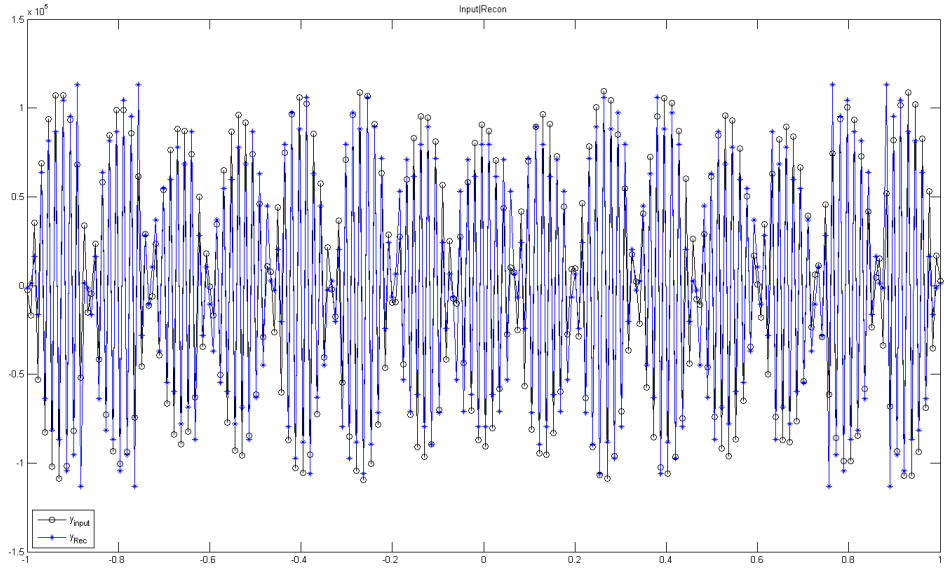


Figure 4.14: Example 2 Original and Reconstructed Signal

Example 3

One of my favorite waves that just so happened to pop-up was when we have coefficients $\{r_i\} = \{10, 5, 2, 8, 8\}$. I really like this wave because I think it shows very astutely what the WHT attempts to accomplish with its transform.

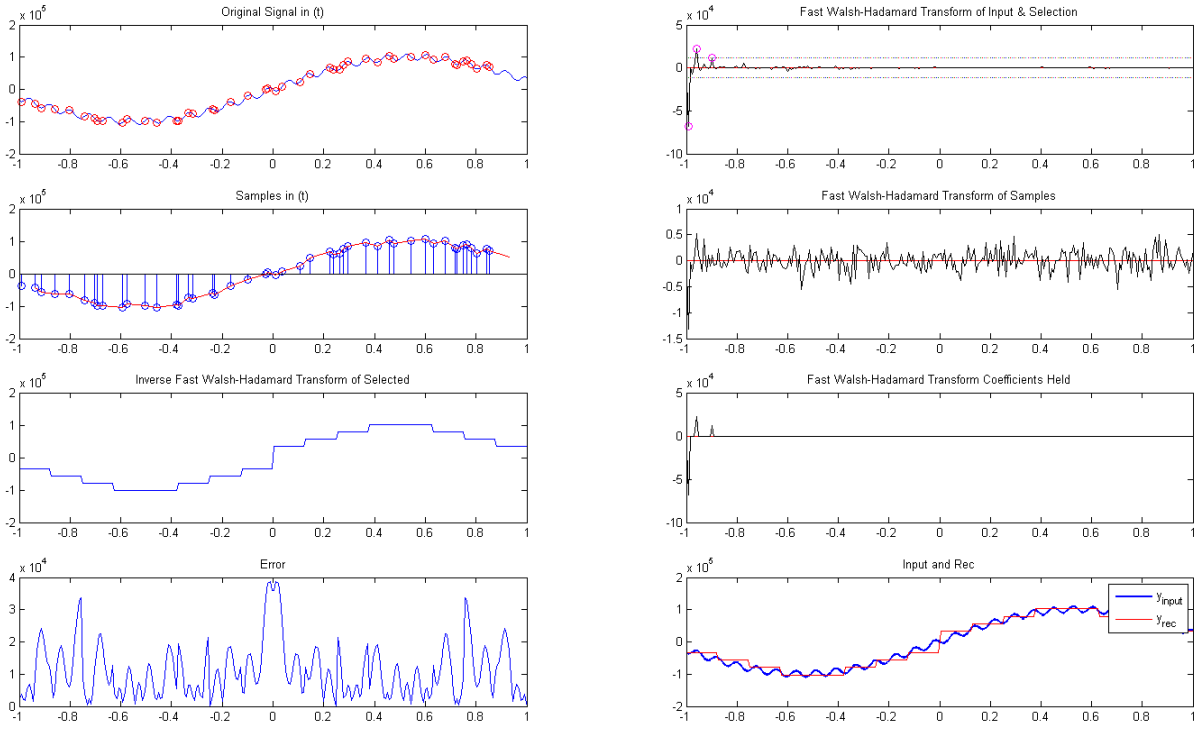


Figure 4.15: Example 3 Original and Reconstructed Signal

Here we are selecting only 3 values in the transformed space to be stored. We manage to get a really great outline of what the input signal is. However we lose almost all of the details (i.e. the smaller bumps) with this process. But if we only ask for a general outline of a signal then we have accomplished that task. Another thing to note is that the non-uniform sampling is actually doing a great job getting the outline of the input signal as well.

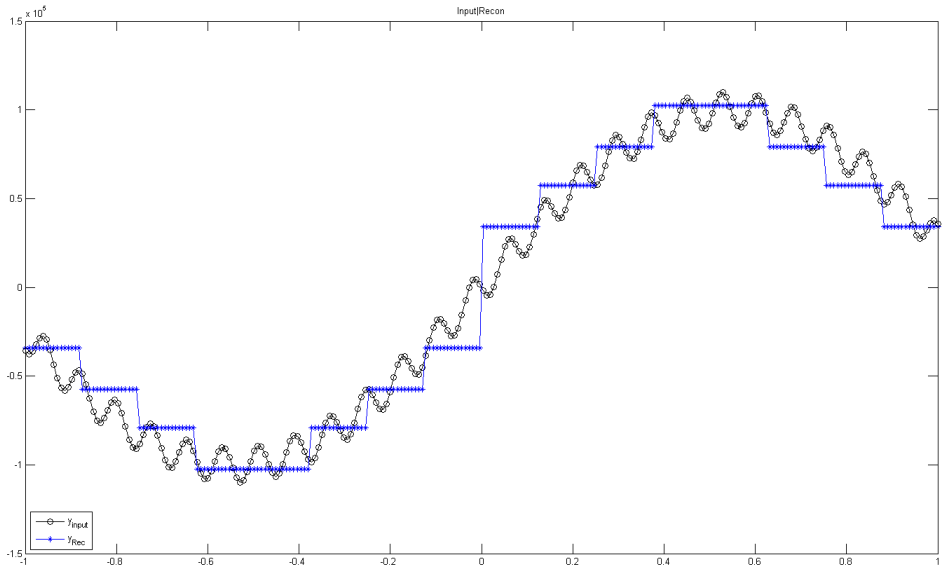


Figure 4.16: Example 3 Original and Reconstructed Signal

4.4 FFT VS. WHT

Now we want to know which transformation produces the smaller error if we feed the same signal into both. We choose the same random numbers that we generated for Example 2 where $\{r_i\} = \{7,4,4,1,6\}$:

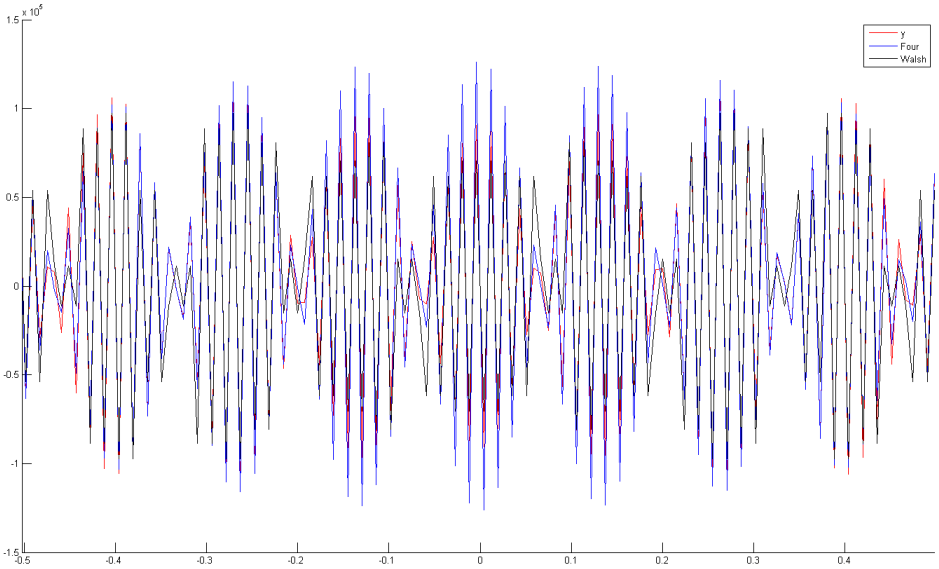


Figure 4.17: The FFT, FWHT, and Original Signal Together

Taking a look at the overall error between the two different transformations we see that both transformations have limitations in their reconstruction. Although the Fourier transform is out preforming the Walsh-Hadamard transformation on the sides, the Walsh-Hadamard transformation out preforms Fourier in the middle.

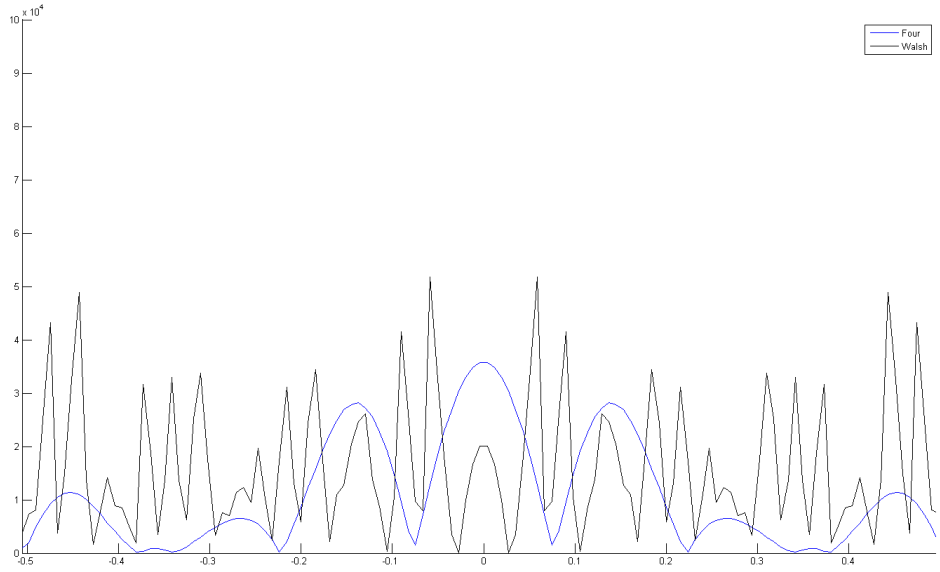


Figure 4.18: Error for FFT and FWHT

4.5 LINEAR CHIRP SIGNALS

We complete this section with some experiments we have using linear chirp signals. A linear chirp signal can be modeled using:

$$f_{\text{chirp}}(t) = \sin\left(\phi_0 + 2\pi\left(f_0 t + \frac{k_i}{2}t^2\right)\right) \tag{4.14}$$

Where:

$$k_i = \frac{(f_{\text{final}})_i - f_0}{T}$$

Here f_{final} and f_0 are the final and initial frequencies respectively and T is the time from

f_{final} to f_0. We consider a train as before:

$$y(t) = \sum_{i=1}^{J} f_{\text{chirp}}(t; k_i) \tag{4.15}$$

For us we will have no phase shift so $\phi_0 = 0$ and our initial frequency will be set to $f_0 = 0$. Our final frequencies will be of the form $(f_{\text{final}})_i = 10 \cdot r_i$.

Example 1

Here we have the results of using $K = 2$ and the FTSA for a sum of 5 chirp signals. We have the following:

$$\{r_i\} = \{9, 10, 2, 10, 7\}$$
$$y(t) = f_{\text{chirp}}(t; 10 \cdot 9) + f_{\text{chirp}}(t; 10^2) + f_{\text{chirp}}(t; 10 \cdot 2) + f_{\text{chirp}}(t; 10^2) + f_{\text{chirp}}(t; 10 \cdot 7) \tag{4.16}$$

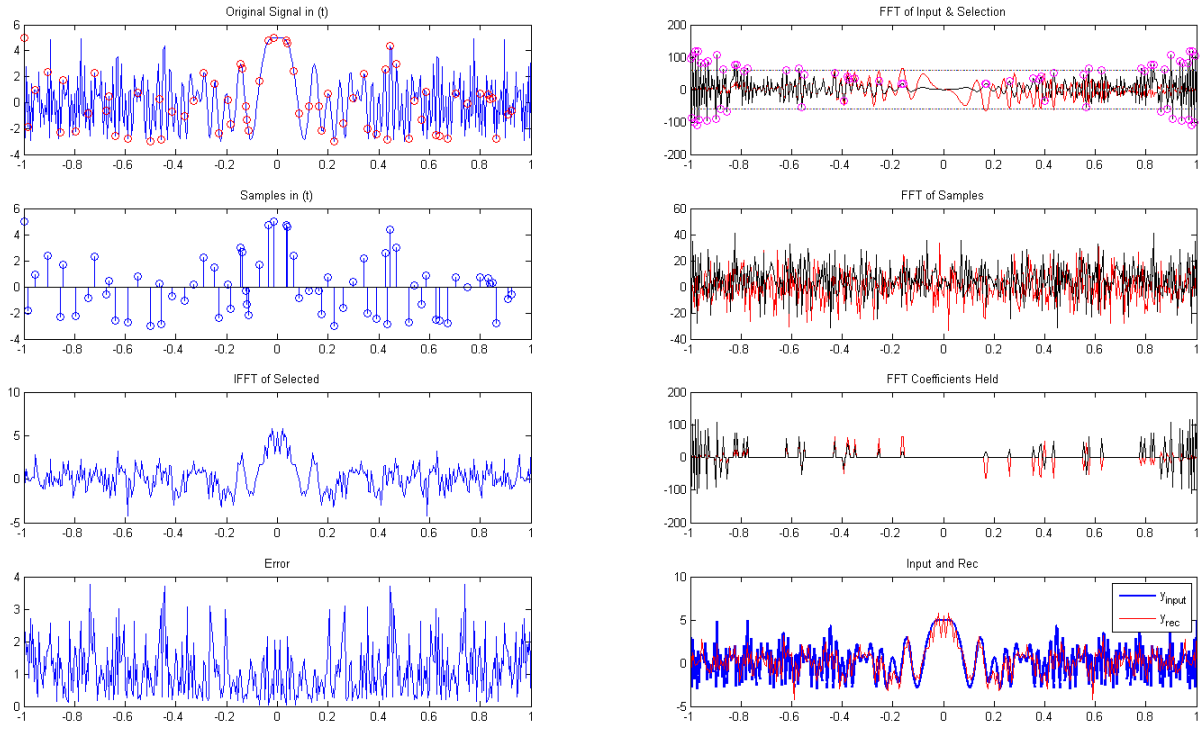


Figure 4.19: Linear Chirp Signals

As anticipated we start selecting more coefficients with different signals. Even with $K = 2$ we are selecting 64 values in the frequency space constituting for about 16% of the available data. Looking at the error plot we see that it is sort of all over the place. But when we look at the two

signals together we see that we are indeed getting a nice representation. When we zoom in on the last plot we can see that we are getting tolerable results.

CHAPTER V

IMAGES AND COMPRESSION

5.1 LOSSY COMPRESSION ALGORITHMS

We are close to our original goal. We want to show how we can use the FTSA for images, but first we will show what we can accomplish with a lossy compression.

We first build an Average Deletion Window (ADW) that have dimension 3×3 and will "move" through an image in a snake like pattern and average the pixels in a ball $\mathbf{B}$ around the center pixel. Here is a diagram of what will happen:

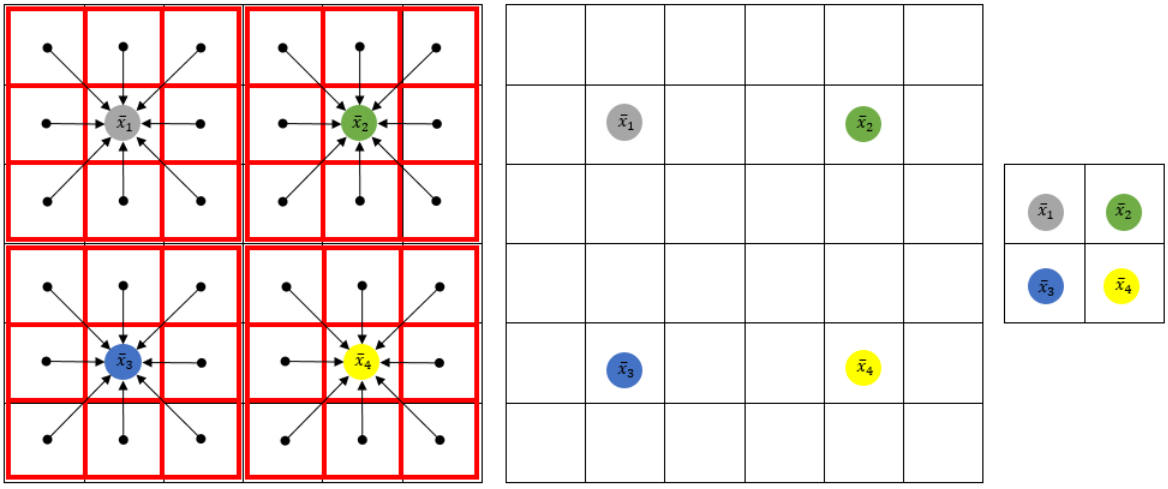


Figure 5.1: Example of ADW Process

So in our above diagram we would have:

$$\bar{x}_1 = \frac{1}{9}\sum_{x\in\mathbf{B}_1} x \qquad \bar{x}_2 = \frac{1}{9}\sum_{x\in\mathbf{B}_2} x \qquad \bar{x}_3 = \frac{1}{9}\sum_{x\in\mathbf{B}_3} x \qquad \bar{x}_4 = \frac{1}{9}\sum_{x\in\mathbf{B}_4} x \tag{5.1}$$

When we send a test image through the process we are still able to get a good represntation of the original image back. However the dimensions have now been changed, and we have lost all of the pixels that were turned to white in the process. By this I mean if we wanted to do an inverse process on the final image there doesn't seem to be a straightforward way to know what the missing 8 pixels in the ball would be. So we call this a lossy compression.

Figure 5.2: Cameraman Image, ADW Result and Compressed Image

Our original image was of size 23.3 KB and we have have compressed it down to a 5.49 KB image, so we would say that we have around a 23% compression rate. Where as before we have:

$$\text{Compression Rate} = \frac{\text{Size of New Image in KB}}{\text{Size of Original Image in KB}} \cdot 100$$

For another example we choose the "Lena" image and run it through the same process outlined above:

Here we get a much better compression rate because the original image has higher resolution. The original image is of size 257 KB and the new image is of size 5.84 KB, then our ADW algorithm results in a compression rate of 2.27 %.

Figure 5.3: Lena Image, ADW Result and Compressed Image

5.2 IMAGES AND FTSA

When we originally were discussing how we should pick the threshold we decided that each column would send its largest value (through the norm) as a representative into a binned array. Then the rest of the columns would be tested against a $1/K$ scaling of the max value in the binned array. When we did this process we got the following results:

These are not ideal numbers. Mostly due to the fact that even with $K = 16$ our file size was starting to become larger after that. But, we would argue that it is still faster to send $K = 16$ image because the compression rate of the resulting sparse representation of the image is okay. Moreover we wouldn't have so send out all of the information from the Fourier space, but rather a mostly zero matrix with only $\approx 2\%$ of its entries that are non-zero. Here are the NNZ (Number of Non-Zero entries) rates for the pictures above where:

$$\text{NNZ Rate} = \frac{\text{Number Of Non-Zeros In New Image}}{\text{Number of Non-Zeros In Original Image}} \cdot 100 \tag{5.2}$$

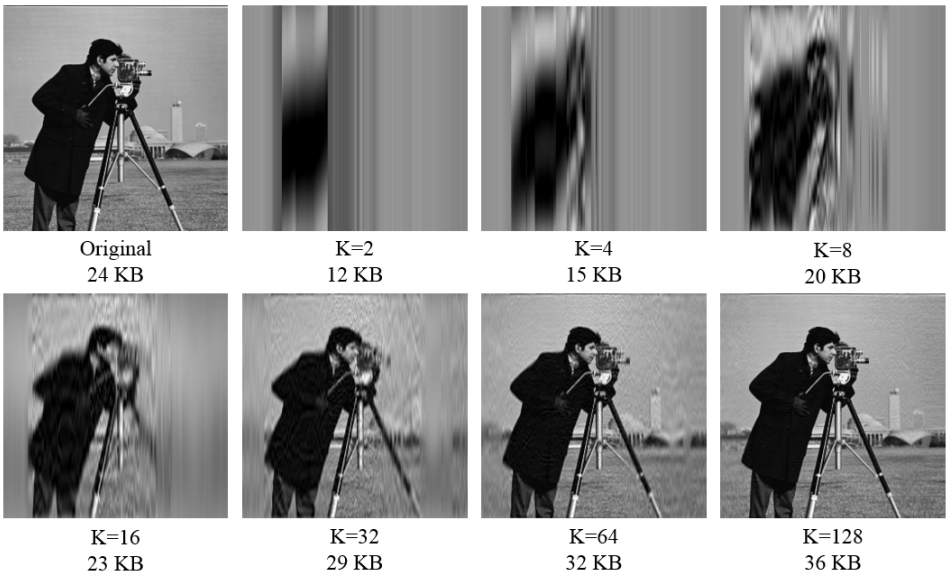


Figure 5.4: Results for FTSA along Columns

K	NNZ Rate%
2	.2747%
4	.3983%
8	.7835%
16	1.9409%
32	4.4556%
64	10.1106%
128	23.5045%

So even though we are getting the same file size of around 23 KB we are getting it through a sparse matrix with only $\approx 2\%$ of the matrix that is non-zero.

The next test we ran was to change the way we where selecting the maximum value for the threshold from the column to the row we had the following results:

Again we have at least a tolerable amount for our sparse representation of our original image. At least in the case when $K = 2$, but our file size is again a bit odd. Perhaps it is a trade-off between

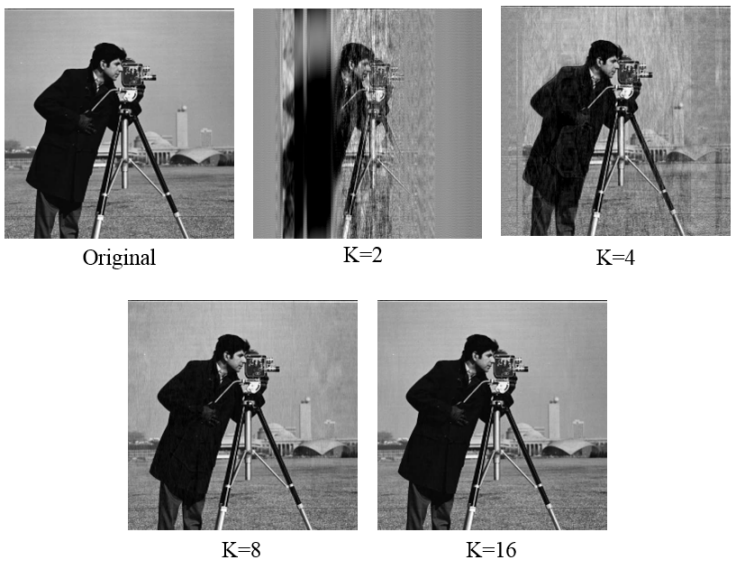


Figure 5.5: Results for FTSA along Rows

wanting a sparse representation and small file size.

K	NNZ Rate%
2	11.4217%
4	52.8906%
8	85.3815%
16	96.1204%

To demonstrate the sparseness of these matrices we will choose a small rectangle of values to display. On the left hand side we have the full FFT values for the image, and on the right hand side we have the thresholded values that have been kept. We also highlight the values in both matrices that have been kept when $K = 2$.

We choose another test image that has much more detail to see how much of our detail is lost when we run the FTSA. We are using a testpat.1k.tiff file that has an original file size of 1,052 KB. When we run the FTSA on this image we have the following results:

-396.349	-389.005	-302.824	-244.076	-128.388	-42.8488	66.6029	-51.6607	-216.877
-159.898	-114.449	-21.8488	13.7896	154.396	219.094	155.832	84.2375	-11.0299
-365.415	-310.064	-268.943	-233.913	-136.495	-76.4086	-107.336	-204.896	-367.572
-324.563	-186.785	-123.786	-90.3774	-46.5679	-148.776	-241.361	-352.276	-451.598
-201.919	-180.503	-194.259	-182.682	-51.5136	-146.822	-99.8681	-143.519	-206.418
-298.516	-306.527	-374.175	-342.104	-210.724	-235.129	-288.445	-415.305	-564.426
-263.31	-214.117	-278.607	-246.473	-116.533	-148.708	-30.3019	-156.155	-238.899
-179.561	-137.585	-223.642	-265.24	-156.892	-188.991	-33.8223	-99.3448	-163.811
-262.332	-246.872	-295.565	-332.398	-357.359	-400.049	-263.072	-407.191	-535.758
-240.109	-175.4	-179.675	-204.356	-305.802	-438.303	-54.3437	-156.855	-256.775
-189.835	-112.146	-69.0196	-154.503	-277.284	-459.414	-325.678	-323.261	-314.992
-470.066	-499.532	-434.647	-413.515	-441.122	-503.637	-385.137	-372.008	-338.6
-241.471	-307.957	-366.571	-358.587	-312.622	-331.875	-133.698	-72.7269	-60.5339
-147.02	-160.319	-209.872	-282.765	-212.56	-258.333	-198.804	-167.382	-143.634
-301.857	-344.547	-414.797	-418.147	-280.109	-206.918	-188.635	-181.495	-228.759
-228.317	-216.433	-218.308	-129.181	20.4292	65.0041	-40.9938	-88.1902	-170.089
-239.01	-188.538	-52.1171	35.8003	98.1334	42.9428	-159.106	-222.112	-254.474
-602.415	-615.023	-401.54	-230.009	-188.343	-202.727	-447.035	-462.959	-524.275
-595.016	-543.294	-343.748	-184.346	-202.113	-257.99	-422.274	-502.676	-561.482
-359.712	-290.124	-93.1069	-26.1431	-98.4468	-199.988	-324.047	-347.982	-347.306
-457.521	-450.885	-323.04	-227.671	-208.26	-188.443	-373.967	-362.514	-401.749
-267.879	-251.225	-214.288	-126.088	-54.8561	15.0426	-225.113	-319.482	-408.237
-86.9518	-43.7294	2.78337	17.2475	83.7241	94.8799	2.84825	-2.92797	-118.661
-315.485	-311.352	-283.137	-275.857	-203.297	-207.426	-303.685	-339.603	-464.495
-200.194	-226.316	-242.593	-226.588	-172.867	-156.302	-26.3824	-134.435	-271.589
-74.9741	-72.0924	-46.6483	-76.0038	-57.6713	-67.9446	0.69989	-0.77468	-103.333
-39.854	-100.229	-150.05	-245.881	-273.141	-272.622	-184.095	-219.484	-246.53
77.8768	13.9932	-92.8883	-170.669	-228.008	-185.103	2.09089	-41.4076	-124.154
-73.8833	-63.9224	-79.9954	-176.187	-281.313	-248.374	9.9053	-39.4839	-151.515
-281.429	-278.295	-234.36	-303.571	-389.87	-411.808	-222.52	-263.481	-291.426
-357.299	-368.726	-320.75	-303.356	-346.525	-371.822	-113.569	-97.4339	-108.309
-476.878	-461.554	-353.996	-336.641	-408.142	-474.007	-227.099	-215.935	-232.38
-448.519	-398.674	-261.611	-232.56	-246.487	-330.313	-245.136	-240.263	-289.424
-257.893	-230.256	-145.591	-101.354	-96.2022	-115.325	43.6919	27.8921	-106.153

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	-564.426
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	-499.532	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
-602.415	-615.023	-401.54	0	0	0	-447.035	-462.959	-524.275
-595.016	-543.294	0	0	0	0	-422.274	-502.676	-561.482
0	0	0	0	0	0	0	0	0
-457.521	-450.885	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
-281.429	-278.295	0	0	0	-411.808	0	0	0
-357.299	-368.726	-320.75	0	0	-371.822	0	0	0
-476.878	-461.554	0	0	-408.142	-474.007	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Figure 5.6: Sub-Rectangle of Matrices for $K = 2$

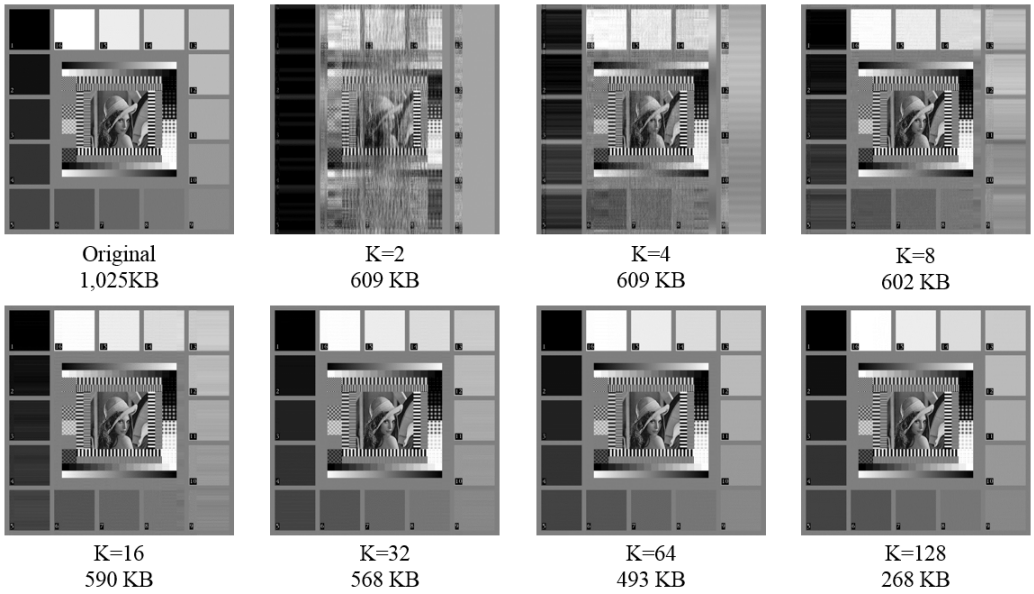


Figure 5.7: Test Image with Row FTSA

K	NNZ Rate%
2	7.4046%
4	23.6442%
8	44.0380%
16	63.6485%

Finally we demonstrate the algorithms results with thermal imaging.

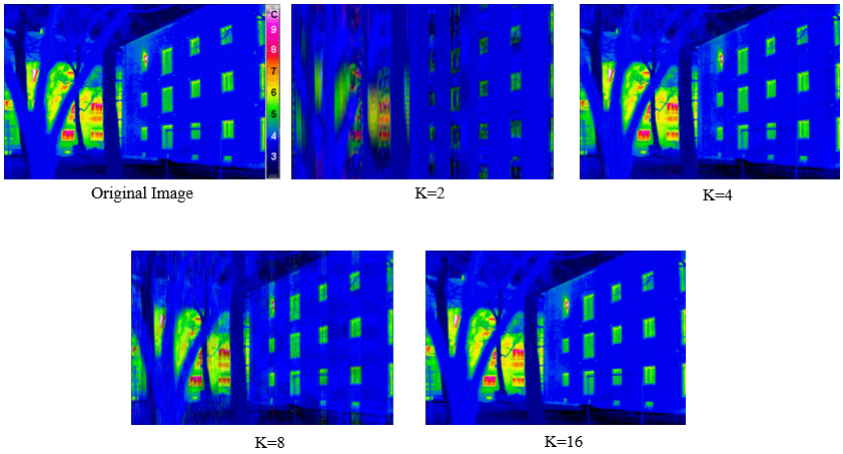


Figure 5.8: Thermal Images with FTSA

5.3 CONTOURS AND EDGE DETECTION

To close out this section we discuss how its possible to get a simple edge detection process. If we view our image as a matrix $\mathbf{A}$ and think a point $p \in \mathbf{A}$ with coordinates $p = (r, c, i)$, where r is the row and c is the column where p resides. Then think of the color hues (either on a gray scale or in color) as an intensity i then we could think of the matrix as a mesh grid of points. Then we could look for regions of points where there is small variation and delete them and where we find significant deviation we store those values we could come up with a contour plot that would contain only the edges of the image.

Here is an example of this process in action:

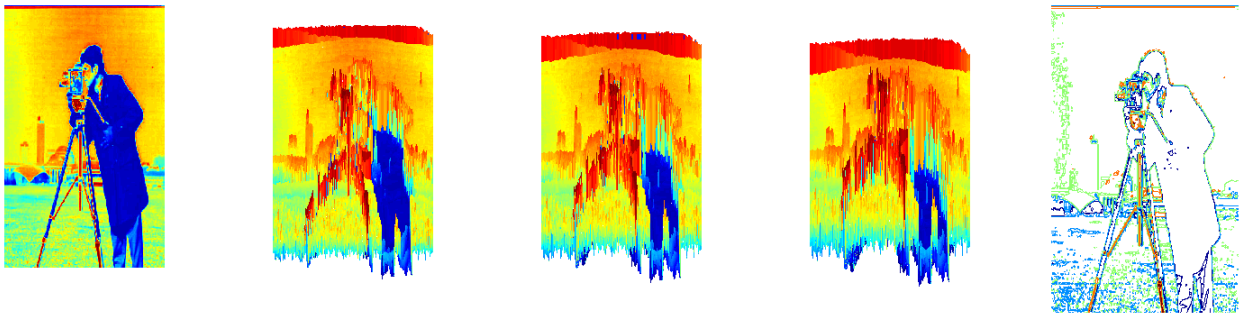


Figure 5.9: Example of Edge Detection through Contours

CHAPTER VI

COMPRESSIVE SENSING

6.1 COMPRESSIVE SENSING FOR SIGNALS

We note at this time that the Fourier transform of the samples in the time domain have correctly coincided with the Fourier data of the original signal with regards to the largest Fourier coefficients in the frequency domain. For example in Example 1 in the first chapter the second row /second column is the FFT of the time samples and it coincides in two correct pikes with the true Fourier data in the first row/second column. But the smaller coefficients are lost to "pseudo noise". But we have a sparse sample of the original signal and wish to use those to recover the original signal. This is where the compressive sensing paradigm comes in to play. Essentially we will use the non-uniformed sampled time data as a preconditioner for the compressive sensing algorithms. We wish to solve the minimization problem:

$$x^* = \min_{x_0} \|x_0\|_1 \qquad \text{subject to} \qquad f(t) = Dx_0 \tag{6.1}$$

Finding x^* corresponds to finding the sparsest solution to an underdetermined system [3]. Fortunately there are many "off the shelf" packages and algorithms to solve these types of problems [3] [8] [1]. We will use the ℓ_1-Magic packaged that was developed by Emmanuel Candes and Justin Romberg at Caltech in 2005. Specifically we use the l1eq_ pd $(\cdot)$. We will also choose as our dictionary a permuted Discrete Fourier matrix.

For those that are curious about how we utilize our time samples I would direct you to the last section where we have the codes available. Mostly just pay attention to the point where the ℓ_1-minimization process takes place. Essentially the process works by a process called orthogonal matching pursuit (OMP).

Example 1 - Using CS

We go back to the first example we used in the begining section as will now use the package above to reconstruct our $f(t)$.

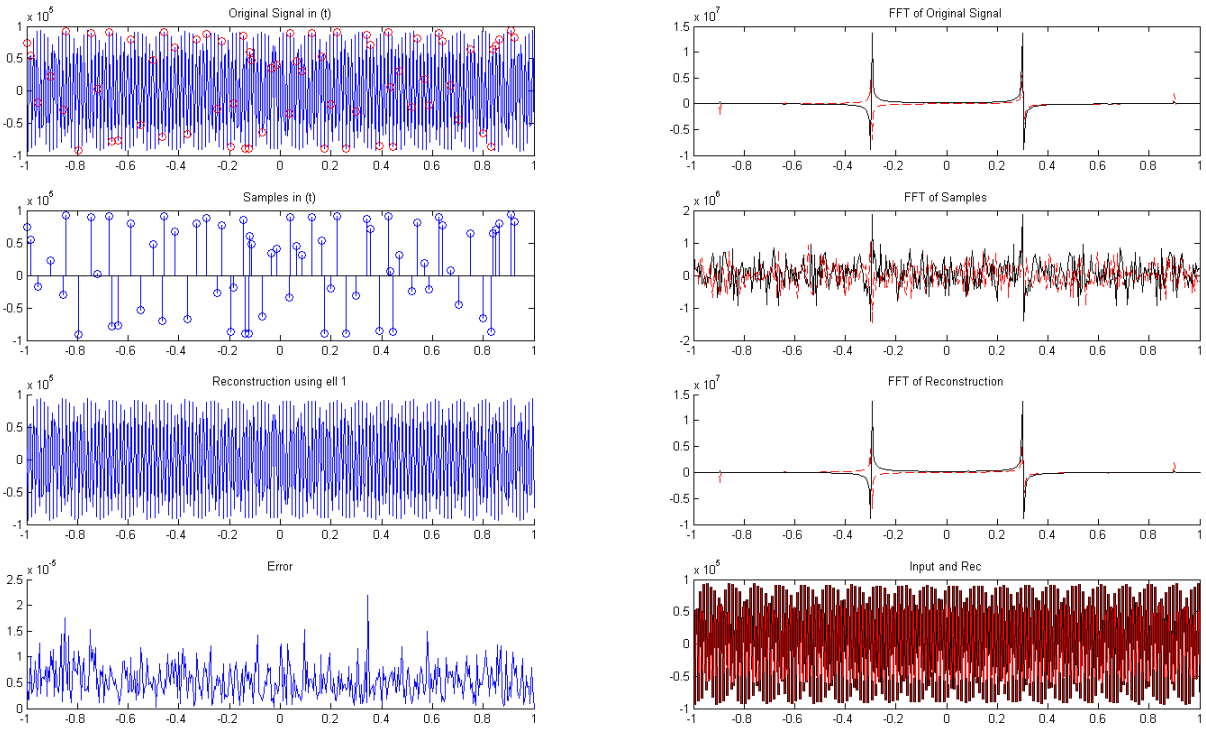


Figure 6.1: Example 1 with Compressive Sensing Algorithm

Just as before the layout of the data is exactly as it was in the previous examples. However we now use the time sampled data to get an very nice representation of the original signal. Essentially we are now using the NUS time data as an initial guess of the original signal. When we do this our error is now on the magnitude of $\approx 2.5 \times 10^{-5}$, whereas when we were using the FTSA we have an error on the magnitude of $\approx 10^5$.

Some things to note is that we are using 67 time samples which accounts for around 16.7% of the available time data. This is slightly higher than what we would ideally want it to be, but for this example it's ok. Another thing to note is that in order for the algorithm to work it needs to have the original signal in order to to the reconstruction.

This is somewhat less than ideal since our problem is that the person on the receiving end of

the signals transmission wouldn't have access to that information a priori.

6.2 COMPRESSIVE SENSING FOR IMAGES

Finally we use the techniques of minimization of the total variation to reconstruct our image. We will use samples taken along 40 radial lines in Fourier space as a precondition for the total variation algorithm. We will essential define the total variation to be:

$$\mathrm{TV}(x) = \sqrt{(x_{i+1,j} - x_{i,j})^2 + (x_{i,j+1} - x_{i,j})^2} \tag{6.2}$$

and we solve iterativly the problem through a second order cone problem:

$$\min \mathrm{TV}(x) \qquad \text{subject to} \qquad y = Dx \tag{6.3}$$

where D is our dictionary matrix and y is the set of measurements. This is solved using interior point methods, and we have the following results:

(a) (c) (b)

Figure 6.2: Results using Total Variation and 40 Sampled Radial Lines

CHAPTER VII

CONCLUSION

We give an overview of signal compression and show results using the FTSA when applied to multiple signals and images. We also show that certain classes of signals have sparse representations that can allow for low error reconstructions.

We also show that compressive sensing when applied to signals and images allow for low data acquisition while allowing for high probability of reconstruction. The next phases in this research is to learn more about interior point algorithms so as to try and develop our own compressive sensing algorithm that can be utilized within the radar system process for low cost high return signal and image acquisition.

BIBLIOGRAPHY

[1] Moeness Amin. *Compressive sensing for urban radar*. CRC Press, 2014.

[2] Charles L Byrne. *Signal Processing: a mathematical approach*. CRC Press, 2014.

[3] Emmanuel Candes and Justin Romberg. l1-magic: Recovery of sparse signals via convex programming. *URL: www. acm. caltech. edu/l1magic/downloads/l1magic. pdf*, 4:14, 2005.

[4] Margaret Cheney and Brett Borden. *Fundamentals of radar imaging*, volume 79. Siam, 2009.

[5] H Evangelaras, Ch Koukouvinos, and J Seberry. Applications of hadamard matrices. *Journal of Telecommunications and Information Technology*, pages 3–10, 2003.

[6] Krzysztof Kulpa. *Signal processing in noise waveform radar*. Artech House, 2013.

[7] Jaime Lopez. The scalar wave model for radar imaging. Working paper, 2012.

[8] Caner Ozdemir. *Inverse synthetic aperture radar imaging with MATLAB algorithms*, volume 210. John Wiley & Sons, 2012.

[9] Fawwaz Tayssir Ulaby, David Gardner Long, William J Blackwell, Charles Elachi, Adrian K Fung, Chris Ruf, Kamal Sarabandi, Howard A Zebker, and Jakob Van Zyl. *Microwave radar and radiometric remote sensing*. University of Michigan Press Ann Arbor, 2014.

[10] Wei Zhu and Jun Tang. Robust design of transmit waveform and receive filter for colocated mimo radar. *IEEE Signal Processing Letters*, 22(11):2112–2116, 2015.

BIOGRAPHICAL SKETCH

John Montalbo was born in San Antonio, TX in 1986. He attended high school at Earl Warren High School in San Antonio, TX. He received a Bachelor's of Science Degree from the University of Texas - Pan American in Mathematics with a concentration in Pure Mathematics in 2014. He continued working towards his Master's of Science in Mathematical Sciences which he earned in 2016. He has been a member of Dr. Zhijun Qiao's Radar and PDE seminar group since 2014 and has been an active graduate student throughout his time with the department. He will continue to seek a PH.D. from the University of Texas - Arlington, which he will start in Fall 2016.

He currently lives in Edinburg, TX with his wife Suzy and their two cats and can be reached via john.montalbo01@utrgv.edu or j.montalb1@gmail.com.