# Using Bayesian Compressive Sensing to Characterize Nonlinear Chirp Signals

Matthew Shane Goodwin

A project submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Dr. C. Shane Reese, Advisor

Department of Statistics

Brigham Young University

April 2017

ABSTRACT

Using Bayesian Compressive Sensing to Characterize Nonlinear Chirp Signals

Matthew Shane Goodwin
Department of Statistics, BYU
Master of Science

Applications in remote sensing require identifying electromagnetic signals from a limited number of samples, in the presence of noise. In this setting, we consider the problem of reconstructing a nonlinear chirp signal with multiple harmonics from a few noisy, random samples. We explore the problem under the framework of compressive sensing (CS), using a Bayesian implementation known as Bayesian compressive sensing (BCS), to find a sparse representation of the signal. This process is initiated by first decomposing the sampled signal into a basis function expansion and then using the BCS algorithm to identify the potentially sparse signal, leaving us with a characterization of the signal that is robust to noise. Different basis expansions are considered, including Fourier, B-splines, and wavelets. A comparison of basis expansions is performed based on the reconstruction accuracy with special consideration to more parsimonious representations.

Keywords: Remote sensing, signal identification, Bayesian compressive sensing, basis decomposition

ACKNOWLEDGMENTS

Above all, I would like to thank my wife for her constant support and encouragement, all while finishing her degree and being pregnant with our first child. Her incredible gift of optimism has been a steady and reliable force that has kept me going through good and bad times. She has the rare gift of knowing how to truly listen and uplift, and my life has been immeasurably blessed from my marriage to her. Thank you McKenzie!

I would also like to thank my classmates for their friendship and support. When all is said and done, the most important part of my graduate education has been the lifelong friendships I have made here at BYU. Thanks also goes to Dr. Todd Moon of Utah State for providing code that generates the signal analyzed in this project. Finally, I would like to thank my advisor Dr. Shane Reese for his guidance and patience with me. He has helped me to expand my vision, helping me to grow as a person and a professional in ways I never could on my own.

TABLE OF CONTENTS

LIST OF FIGURES

# Chapter 1
# Introduction and Background

The field of remote sensing has had and continuous to have an enormous impact on modern society. It has expanded military intelligence capabilities, facilitated the exploration of different planets, and revealed greater truths about our own planet we would not have discovered otherwise. For example, in the 1970's, NASA launched the first Landsat satellite into space with the mission of monitoring the Earth's surface, including Earth's water supply, food supply, and general vegetation growth or decline. This information has helped aid policy decisions involving topics such as agriculture and climate research (NASA, 2012). Another example is the use of radar during World War II to track and detect enemy targets, such as planes and ships (Elachi and van Zyl, 2006). Undoubtedly, this technology has saved lives and changed the tactics of competing nations. Many more examples could be given, both modern and historical, illustrating the important role remote sensing has played and continues to play in our world.

This impact will continue to be felt as more discoveries are made and more effective methods developed. The field of remote sensing is an active area of research, with various problems to explore and solve. One of these problems is identifying signals in the presence of unwanted noise. As with many real world signals, signals within the field of remote sensing are subject to measurement noise and other confounding sources. For instance, some areas of remote sensing must deal with unwanted noise coming from the atmosphere (Campbell and Wynne, 2011). Another challenge, together with the problem of noise, is that the number of samples available to identify the signal might be limited. Methods used to identify signals from some remote sensing data must therefore account for both noise and limited samples in order to be effective or realistic.

This project explores the possibility of using a recently developed technique, called compressive sensing, to develop a model that can identify certain types of signals (as discussed in section 2.1) in the presence of noise and from a limited number of samples. To explain how this model works, we first provide a very brief overview of electromagnetic signals in a remote sensing context, as well as several mathematical topics needed to develop the model. We then describe the signal used in this analysis and give a detailed summary of a Bayesian approach to compressive sensing and how we use the approach to develop the final model. Finally, we give the results and conclusions of two different experiments that test the reconstruction accuracy of the approach and how the sparsity of the model output changes with added noise. Possible future research is discussed as well.

## 1.1 Introduction to Electromagnetic Signals and Remote Sensing

There are various definitions for remote sensing, but perhaps the most basic is defined by Barrett and Curtis (as cited in Campbell and Wynne, 2011, p. 6): “Remote sensing is the observation of a target by a device separated from it by some distance”. Typically, the “observation of a target” is an electromagnetic signal or electromagnetic radiation. This type of energy is emitted by all objects or is reflected by objects after coming from other sources (Campbell and Wynne, 2011).

The type of electromagnetic radiation we are most familiar with is visible light. This type of radiation is only a relatively small fraction of the electromagnetic spectrum as can be seen from Figure 1.1. Common cameras and our eyes are built to recognize this type of radiation, but other types require special sensors to detect and analyze.

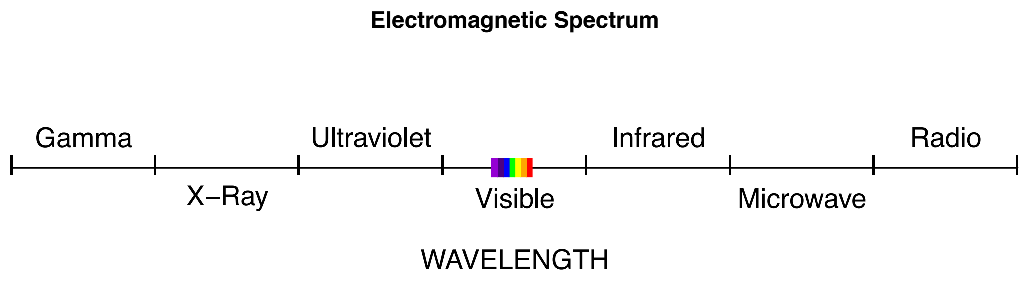


Figure 1.1: The electromagnetic spectrum. Note that this plot is only a rough representation of the electromagnetic spectrum and is not drawn to scale. The spectrum can be ordered by wavelength with wavelength increasing from left to right.

There are two related characteristics of waves that can be used to describe the ordering of the spectrum. One possible method is to use the *frequency* of a wave, or the number of cycles of a signal in unit time. Essentially, this is the number of times the wave repeats itself in unit time. Gamma waves on the far left of the spectrum have the highest frequency compared to other waves, and as one moves right along the spectrum, the frequency decreases. Another way to think of the ordering of the spectrum is to use *wavelength*. Wavelength is simply the distance from one crest of the wave to the next (see Figure 1.2). So radio waves on the far right of the spectrum have the largest wavelength, and as one moves left, the wavelength decreases.

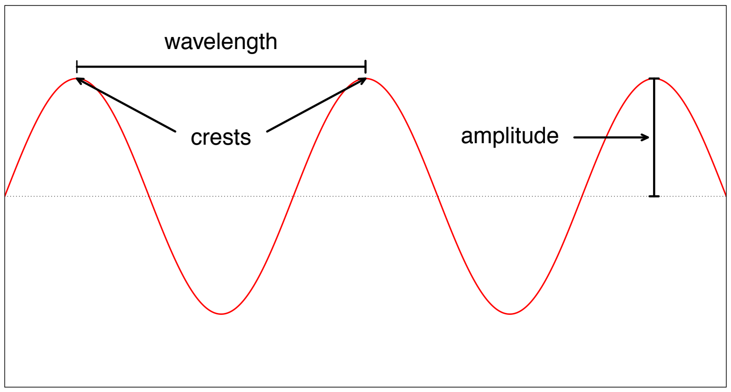


Figure 1.2: Basic anatomy of a wave.

As Campbell and Wynne discuss, one large source of electromagnetic radiation, of various wavelengths, is the sun. When these waves arrive at earth, objects on the ground can either reflect or absorb the waves. Many reflected waves are in the visible light range and as was mentioned, can be measured using a common camera. The waves that are absorbed however, are reradiated as thermal energy, a type of energy that is best measured in the infrared regions of the spectrum. To measure waves in this region requires special sensors beyond the typical camera.

The measurement of both reflected waves from the sun and emitted electromagnetic waves fall under an area of remote sensing termed *passive* remote sensing. Another area is termed *active* remote sensing. This type of remote sensing includes sensors that send out their own electromagnetic signal and then measure the reflection from the object (Campbell and Wynne, 2011). An example of this is radar detection, which sends out radio or microwaves and observes the returning signal in order to calculate distance or characteristics of the object.

## 1.2 Basis Function Expansion

From elementary linear algebra, a basis of a vector space is defined to be a set of vectors that are linear independent and span the entire space. This implies that any vector in the vector space can be written as a linear combination of the basis. This concept applies to functions as well. Given a basis of a function space, any function in the function space can be written as a linear combination of the basis functions. More specifically, for a function $f$ at a given input $t$, we can write the linear combination of $N$ basis functions

$$f(t) = c_1\phi_1(t) + c_2\phi_2(t) + ... + c_N\phi_N(t) \tag{1.1}$$

where $\phi_i$ are the different basis functions and $c_i$ are scalar coefficients, with $i = 1, 2, \ldots, N$. For multiple inputs, this can be written in matrix form

$$\begin{bmatrix} f(t_1) \\ f(t_2) \\ \vdots \\ f(t_M) \end{bmatrix} = \begin{bmatrix} \phi_1(t_1) & \phi_2(t_1) & \phi_3(t_1) & \ldots & \phi_N(t_1) \\ \phi_1(t_2) & \phi_2(t_2) & \phi_3(t_2) & \ldots & \phi_N(t_2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \phi_1(t_M) & \phi_2(t_M) & \phi_3(t_M) & \ldots & \phi_N(t_M) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_N \end{bmatrix} \tag{1.2}$$

where $M$ in our analysis is typically less than $N$. We can write this more compactly as

$$\mathbf{f} = \mathbf{\Phi c}. \tag{1.3}$$

This process is known as a basis expansion or decomposition of the function $f$. A visual example of this concept is illustrated in Figures 1.3 and 1.4. Figure 1.3 shows five basis functions from the B-spline basis. When these basis functions are multiplied by the coefficient vector $[3, 2, 14, 4, 7]'$ and added together, then the curve in Figure 1.4 is obtained.

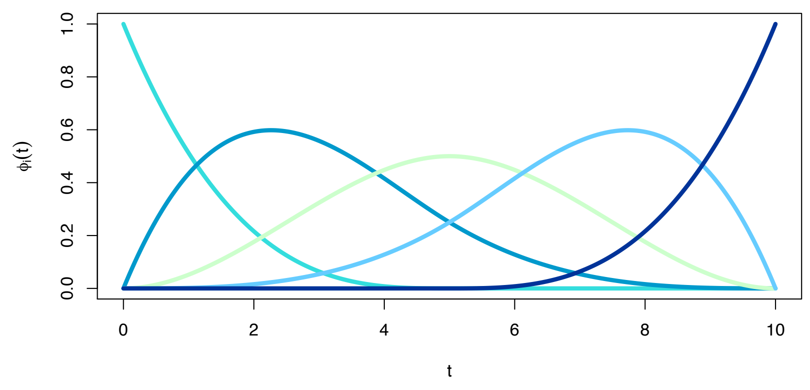


Figure 1.3: Plot of an example of a few different B-Spline basis functions. A linear combination of these basis functions equals the function shown in Figure 1.4. The basis functions here are generated using the `R` package `fda`.

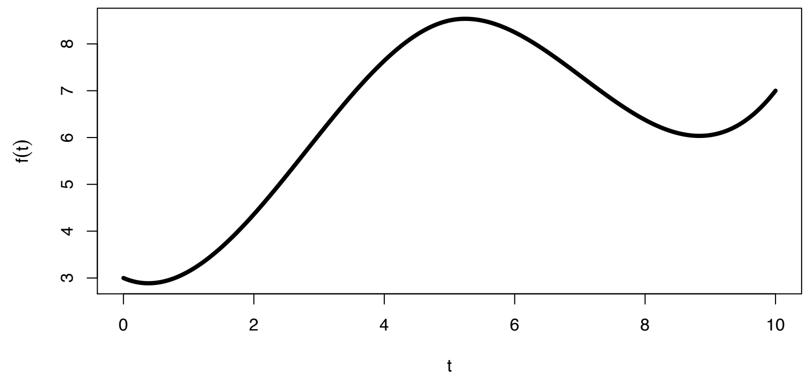


Figure 1.4: Example of a function that can be decomposed into a linear combination of B-spline basis functions.

In many applications, the function or vector $\mathbf{f}$ is known and the basis $\mathbf{\Phi}$ can be selected. The unknown quantity therefore is the coefficient vector $\mathbf{c}$. In the case that $\mathbf{\Phi}$ is an orthonormal square matrix ($M = N$), we simply multiply both sides of (1.3) by the transpose of $\mathbf{\Phi}$. In this analysis however, we will find that $M \ll N$, and so other techniques will be needed to solve for $\mathbf{c}$.

In this analysis we look at three different types of bases. The first we consider is the Fourier basis, followed by the wavelet basis, and lastly the B-spline basis.

### 1.2.1 Fourier Basis

Named after Joseph Fourier, a French mathematician and physicist, Fourier analysis is used in a variety of scientific fields and has had an immeasurable impact on mathematics and different areas of science. The main idea behind Fourier analysis is that almost any periodic function can be decomposed into a linear combination of sine and cosine functions (Bachman et al., 2000). Since these basic trigonometric functions are well known and generally easier to work with, difficult processes on complicated functions and signals can be simplified.

A good starting place in an brief overview of Fourier analysis is the definition of the Fourier series, which is simply defined as an infinite sum of sines and cosines. This series can also be defined in terms of the exponential function. To see this, we first take the $N^{th}$ partial sum of sines and cosines for some periodic function $f(t)$ (Bachman et al., 2000),

$$f_N(t) = \frac{a_0}{2} + \sum_{n=1}^{N} a_n \cos(nt) + b_n \sin(nt). \tag{1.4}$$

Using Euler's Formula,

$$e^{int} = \cos(nt) + i\sin(nt), \tag{1.5}$$

we can then rewrite the sine and cosine functions (Pereyra and Ward, 2012),

$$\sin(nt) = \frac{e^{int} - e^{-int}}{2i}, \qquad \cos(nt) = \frac{e^{int} + e^{-int}}{2}. \tag{1.6}$$

Plugging these forms into (1.4) gives us

$$\begin{aligned} f_N(t) &= \frac{a_0}{2} + \sum_{n=1}^{N} a_n \frac{e^{int} + e^{-int}}{2} + b_n \frac{e^{int} - e^{-int}}{2i} \\ &= \frac{a_0}{2} + \sum_{n=1}^{N} \left( \frac{a_n}{2} + \frac{b_n}{2i} \right) e^{int} + \sum_{n=1}^{N} \left( \frac{a_n}{2} - \frac{b_n}{2i} \right) e^{-int} \\ &= \frac{a_0}{2} + \sum_{n=1}^{N} \left( \frac{a_n - ib_n}{2} \right) e^{int} + \sum_{n=1}^{N} \left( \frac{a_n + ib_n}{2} \right) e^{-int}. \end{aligned} \tag{1.7}$$

We then define $b_0 = 0$ and define new coefficients (Bachman et al., 2000)

$$c_n = \frac{a_n - ib_n}{2}, \qquad c_{-n} = \frac{a_n + ib_n}{2}. \tag{1.8}$$

Combining sums in 1.7, we now have

$$f_N(t) = \sum_{n=-N}^{N} c_n e^{int}. \tag{1.9}$$

If we let $N$ go to infinity, then the sum on the right is known as the exponential form of the Fourier series and the coefficients $c_n$ are known as the Fourier coefficients (Bachman et al., 2000).

One of the most important ideas of Fourier analysis is found in the relationship between the time domain and frequency domain of a signal. This relationship can be seen in (1.4) by noticing that at a certain time point $t$, $f(t)$ is defined by summing over the frequencies $n$ of the sine and cosine functions. The coefficient $c_n$ in (1.9) is therefore related to the amplitudes of the sines and cosines at a specific frequency. This leads us to what is known as the Fourier transform, which essentially takes a signal in the time domain and finds the frequency components of the signal. In its general form this transformation occurs with a continuous signal, but in practical applications, especially in digital signal processing, we only have a finite number of samples from the signal. This leads us to the discrete Fourier transform (DFT), which is defined as

$$\hat{f}(n) = \sum_{t=0}^{N-1} f(t) e^{\frac{-2\pi i t n}{N}}. \tag{1.10}$$

This is related to the inverse DFT which is given as

$$f(t) = \frac{1}{N} \sum_{n=0}^{N-1} \hat{f}(n) e^{\frac{2\pi i t n}{N}} \tag{1.11}$$

which is of similar form to (1.9). The proof of how these two formulas are inverses of each other is given in Bachman et al. (2000). We now have a means of going between the frequency domain and time domain of a signal.

The Fourier basis is used in our analysis, but instead of defining the basis functions as complex numbers made up of sines and cosines, the separate trig functions are used as the individual basis functions, with the first basis being a constant. The first five basis are therefore

$$1, \sin(\omega t), \cos(\omega t), \sin(2\omega t), \cos(2\omega t) \tag{1.12}$$

where $\omega = \frac{2\pi}{T}$ and $T$ is the period of each basis function (Ramsay et al., 2009). A few of these basis functions are shown in Figure 1.5.

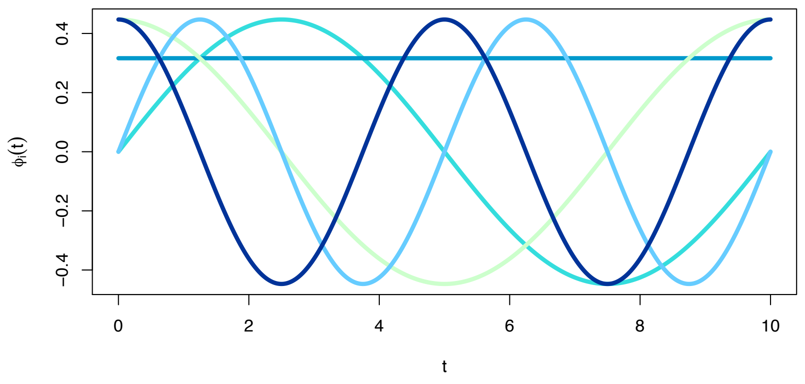


Figure 1.5: An example of a few different Fourier basis functions. The basis functions here are generated using the `R` package `fda`.

### 1.2.2 Wavelets

One of the shortcomings of the Fourier transform occurs when the given signal's frequencies change with time, a type of signal known as a chirp signal. As Najmi (2012) discusses, the Fourier transform by definition only shows the signal in the frequency or time domain, not both domains simultaneously. Therefore, it does not show when specific frequencies occur. One possible solution to work around this issue while still using the Fourier transform, as mentioned by Najmi, is to use what is called the short time Fourier transform

(STFT). The STFT divides the time domain into "windows" and finds the frequency components in each specific window. This provides information about the frequency components at different time points, and how those frequency components change with time. The change in frequency can then be shown visually by creating a plot called a spectrogram. Figure 2.1(b) in Chapter 2 gives an example of such a spectrogram.

The STFT provides a "fixed resolution method of time-frequency analysis" (Najmi, 2012). For some signals however, as Najmi discusses, we might want to vary the window or resolution size in order to better analyze the different frequency changes. For example, we might have a signal with a section of very high frequencies. If our window is too large, or equivalently, our time resolution too low, then we won't be able to determine where the frequencies occurred as accurately. On the other hand, we might have lower frequencies that only require a lower resolution to accurately analyze. Therefore, we desire a method that is able to analyze the signal at different resolutions in order to understand the different frequency ranges better, with respect to time. One such method is wavelets.

Wavelets are used in a variety of fields, and have seen increased interest in the last couple of decades (Percival and Walden, 2000). A common application of wavelets is in image compression, specifically the JPEG 2000 image and compression standard. An interesting property of almost all photographs is that they are sparse with respect to the wavelet basis (Mackenzie, 2009). This implies that photographs can be represented by a relatively few coefficients in the wavelet domain, or in other words, the photo is *compressible* in the wavelet domain. With cameras becoming more and more powerful and capturing more and more information, wavelets continue to grow in importance so that storing and sharing photos remains a reasonable process.

As Percival and Walden (2000) discuss, one way to think about wavelets is to view them as calculating the difference between windowed averages, at different scales. One example given in Percival and Walden (2000) is the performance of an atomic clock. The data shows the deviations between the atomic clock and a US Naval Observatory "master" clock, as time progresses. The data for a perfect atomic clock would therefore stay fixed at zero across time, meaning no deviations occurred. This is not the case however, and so one question of interest might be to ask how much the clock deviates from one time interval to the next, whether from day to day or from month to month. If we wanted to compare month to month for example, then we would take the average of the deviations from the first month and compare with the average of the deviations from the second month. This process is essentially equivalent to using the wavelet transform at different scales, one scale for the day to day comparison, and another scale for the month to month comparison.

On a more technical level, a wavelet $\phi$ is defined as any function that satisfies the following two equations (Percival and Walden, 2000)

$$\int_{-\infty}^{\infty} \psi(t)dt = 0 \tag{1.13}$$

$$\int_{-\infty}^{\infty} \psi(t)^2 dt = 1. \tag{1.14}$$

As Percival and Walden (2000) explain, the second equation forces the function to deviate from the zero function and excludes functions that are waves going to infinity (which would be a “large” wave instead of a “small” wave or “wavelet”). With the zero function ruled out, the first equation forces the function to be an actual wave, since for the equation to hold, equal area must be below and above zero. A third condition is sometimes included in practical applications called the *admissibility condition*. See Percival and Walden (2000) for details.

There are various types of wavelets. One of the most basic is called the Haar wavelet, and is defined as (Percival and Walden, 2000)

$$\psi(t) = \begin{cases} \frac{-1}{\sqrt{2}} & -1 < t < 0 \\ \frac{1}{\sqrt{2}} & 0 < t < 1 \\ 0 & \text{otherwise.} \end{cases} \tag{1.15}$$

Figure 1.6 gives an example of a set of Haar wavelet basis functions. Note how the basic definition above is scaled and shifted to form the other basis functions in the set.

There are an infinite number of wavelets, but in practice there are several common different wavelet families, each with different characteristics, varying between levels of

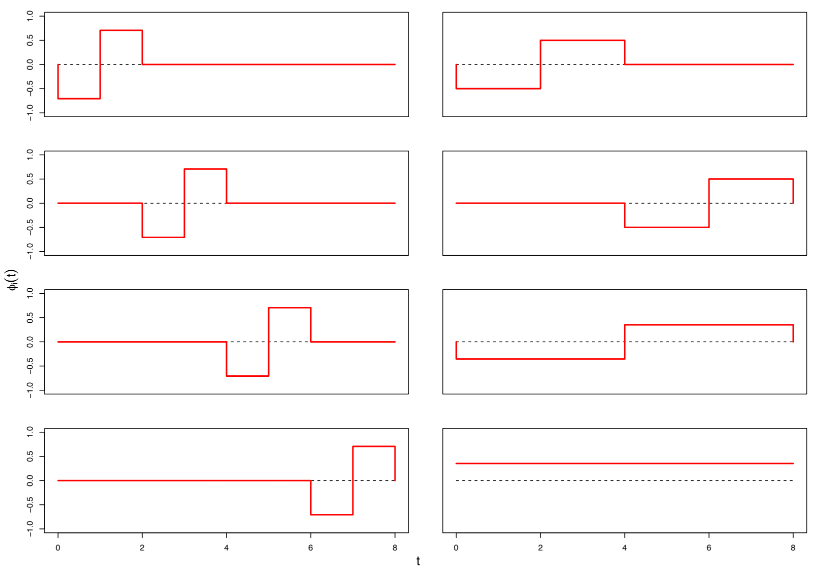
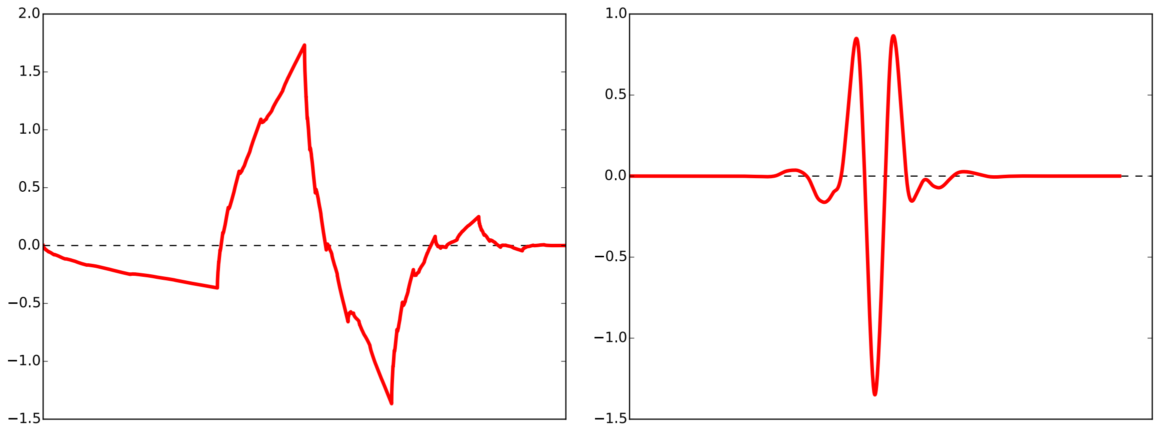


Figure 1.6: An example of a set of the Haar wavelet basis functions. The basis functions here are generated using the `R` package `wmtsa`.

smoothness and compactness (Graps, 1995). A few examples include Daubechies wavelets, Coiflets, and symlets. Within each of these families are subclasses of wavelets classified by the number of vanishing moments (Graps, 1995). An example of two of these wavelets are shown in Figures 1.7(a) and 1.7(b). Other wavelet examples include the Mexican hat wavelet, the Morlet wavelet, and the Shannon wavelet, to name a few.



(a) Daubechies 2 wavelet (b) Symlet 8 wavelet

Figure 1.7: An example of the Daubechies 2 wavelet on the left, and the symlet 8 wavelet on the right. Plots were generated using Python's PyWavelets package.

### 1.2.3 B-spline Basis

B-splines are used in a variety of applications including computer graphics, medical imaging, numerical simulation, automated design and manufacturing, and in general approximation of functions (Höllig and Hörner, 2013). Like the basis functions mentioned previously, we use B-splines as a way to represent the given signal.

In general, splines are simply piecewise polynomials with certain properties that control how the polynomials are connected and how smooth they appear. We follow the assumption made in Ramsay et al. (2009), and assume that the polynomials all have the same degree $n$ or equivalently, order $n+1$. Ramsay et al. uses the term degree to mean the largest power found in the polynomial. For example, if we have the polynomial expression $x^3 + 2x$, the degree would be 3 and the order $3 + 1 = 4$.

The locations where the polynomials are connected are known as *knots*. More formally, knots are part of a nondecreasing sequence of real numbers, $\xi_0 \leq \xi_1 \leq \cdots \leq \xi_m$, that partition an interval of the real number line into knot intervals, $[\xi_i, \xi_{i+1})$ (Höllig and Hörner, 2013). These knot intervals can be different sizes, but in our analysis we keep them uniform. The multiplicity of a knot $\xi_i$ is the number of repetitions of $\xi_i$ in the knot sequence (Höllig and Hörner, 2013). For example, given the knot sequence $1, 3, 4, 4, 5$, the knot found at 4 would have a multiplicity of 2.

The smoothness of the spline function at the knot points depends on the multiplicity of the knots and the degree of the polynomials. For a particular knot $\xi_i$ with multiplicity $\mu$, and polynomial of degree $n$, the number of continuous derivatives at $\xi_i$ is $n - \mu$ (Höllig and Hörner, 2013). A few examples are motivated by Ramsay et al. If we have polynomials of degree 3 and a knot with multiplicity 1, the spline function will be twice continuously differentiable $(3 - 1 = 2)$ at that knot. If we want the spline function to be continuous at the knot, but not continuously differentiable, then we can let the knot multiplicity equal 3 at that location $(3 - 3 = 0)$. Finally, if we want to allow the function value to abruptly change at the knot location, we would let the knot multiplicity equal 4 $(3 - 4 = -1)$. The knot multiplicity at both ends of the interval of interest is typically equal to the order of the polynomials. This allows the spline function to go to zero outside the interval (Ramsay et al., 2009).

One way to define the B-spline basis is to use a recursive relationship discovered by de Boor, Cox, and Mansfield (as cited in Höllig and Hörner (2013)). This is given by

$$b^n_{i,\xi} = \gamma^n_{i,\xi} b^{n-1}_{i,\xi} + (1 - \gamma^n_{i+1,\xi}) b^{n-1}_{i+1,\xi}, \qquad \gamma^n_{i,\xi}(x) = \frac{x - \xi_i}{\xi_{i+n} - \xi_i} \tag{1.16}$$

where $b^n_{i,\xi}$ is the $i^{th}$ B-spline basis function of degree $n$ and with the knot sequence $\xi$. The variable $\xi_i$ is the $i^{th}$ knot. To begin using the recursive relationship define

$$b^0_{i,\xi} = \begin{cases} 1 & \text{for } \xi_i < x < \xi_{i+1} \\ 0 & \text{otherwise.} \end{cases} \tag{1.17}$$

An example of a few B-spline basis functions is shown in Figure 1.3. A linear combination of these functions gives the function in Figure 1.4.

## 1.3 Compressed Sensing

One of the classic topics of research in the field of signal processing is the problem of reconstructing a signal from a limited number of samples. In 1949, Claude Shannon published the famous sampling theorem which states that if a signal does not contain frequencies higher than $B$ hertz, then it is completely determined by taking samples at a series of points spaced $\frac{1}{2B}$ seconds apart (Shannon, 1949). Many of the advances in signal processing during the twentieth century can be traced back to this theorem, including the development of sensing systems that are more flexible and cheap to build, resulting in a "digital revolution" where more and more data are being collected (Davenport et al., 2012).

In some areas however, the deluge of data can create complications or inefficiencies. For example, as discussed in Mackenzie (2009), camera manufactures adhering to Moore's law continually increase the number of pixels a camera is able to capture. This might seem to be a desirable feature of cameras, but in reality the amount of information captured can become too large to realistically store in a file. To work around this, engineers developed algorithms to compress photos into more manageable sizes by transforming a photo with respect to some basis, such as the wavelet basis used in JPEG 2000, and keeping only the most "important" elements of the transformed signal. What is amazing is that the compressed photos are nearly indistinguishable from the original, suggesting that at least some types of signals can be characterized by only a relatively few elements.

While the sensors found in cameras are relatively cheap to produce, sensors developed for other types of signals can be costly to manufacture. One example of this, as mentioned in Mackenzie (2009), is the sensors used to detect tetrahertz radiation, which could be used to detect hidden contraband under clothing. This type of radiation is found in the infra-infrared region of the electromagnetic spectrum, so special types of sensors are needed that are more costly to produce than the common camera sensors. Mackenzie also mentions how other applications might be more constrained by the lack of space for sensors, such as the limited capacity of spacecraft.

The framework of compressive sensing (CS) is a promising solution to the above problems since it significantly reduces the number of samples needed to accurately represent a signal, when compared to the number of samples required by the sampling theorem. Instead of using information about the *frequencies* of the signal, the CS framework uses information about the *sparsity* of the signal, or how many elements of the signal are zero or close to zero. This idea allows the two steps of signal acquisition and compression, as in the camera example, to be combined into one, by taking a few, random samples of a transformed signal and finding the most important elements. Instead of having to take uniformly spaced samples, only a relatively few, random samples are needed. This allows some applications to be more feasible, reducing costs and overcoming different constraints since less sensors are needed. It has an impact in medical imaging, for example, where CS helps reduce MRI time, while still providing high resolution images (Lustig et al., 2007).

Mathematically, we can describe the CS process by first starting with some $P \times 1$ signal $\mathbf{f}$ we are interested in sampling from and possibly reconstructing. It might be that $\mathbf{f}$ itself is not sparse, but is sparse with respect to some basis $\boldsymbol{\Psi}$. This can be written as

$$\mathbf{f} = \boldsymbol{\Psi}\mathbf{w}. \tag{1.18}$$

where $\mathbf{\Psi}$ is $P \times N$ and $\mathbf{w}$ is an $N \times 1$ sparse signal. Let $\mathbf{\Phi}'$ be some measurement matrix with dimensions $M \times P$ and $M \ll N$. If we let $\mathbf{n}$ be measurement noise, we then have

$$\begin{aligned} \mathbf{y} &= \mathbf{\Phi}'\mathbf{f} + \mathbf{n} \\ &= \mathbf{\Phi}'\mathbf{\Psi}\mathbf{w} + \mathbf{n} \end{aligned} \tag{1.19}$$

where $\mathbf{y}$ is the observed sample from the original signal $\mathbf{f}$ plus measurement noise. If we let $\mathbf{\Phi} = \mathbf{\Phi}'\mathbf{\Psi}$ we then have

$$\mathbf{y} = \mathbf{\Phi}\mathbf{w} + \mathbf{n}. \tag{1.20}$$

In many applications, we typically don't know the signal $\mathbf{f}$, only the observed samples $\mathbf{y}$. To solve $\mathbf{f}$, we can first solve for $\mathbf{w}$ in (1.20) and then use (1.18) to reconstruct $\mathbf{f}$. We can also view the process of solving for $\mathbf{w}$ as a way of finding a sparse representation of the original signal $\mathbf{f}$.

The system of equations in (1.20) is highly underdetermined, meaning there are many more variables to solve for than there are equations. This implies that there are infinitely many solutions and no obvious method of solving for $\mathbf{w}$. However, as mentioned above, by assuming $\mathbf{w}$ is sparse, we can narrow down the possible solutions. This process therefore requires a way of measuring sparsity. In the CS literature, generally two norms are considered: the $\ell_0$ norm and the $\ell_1$ norm. The $\ell_0$ norm is defined to be the number of nonzero components of the signal. It is important to note that this norm is not a true norm but is considered to be a "pseudo-norm" since it fails some of the criteria of norms. However, it does still provide an efficient method of determining the sparsity of a signal, and is used in the definition of a $k$-sparse signal, which states that a signal $\mathbf{x}$ is $k$-sparse if $\|\mathbf{x}\|_0 \leq k$ (Davenport et al., 2012).

Using this measure of sparsity, we can then reframe (1.20) as a minimization problem

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} \{ \, \|\mathbf{y} - \mathbf{\Phi}\mathbf{w}\|_2^2 + \tau \|\mathbf{w}\|_0 \, \}. \tag{1.21}$$

This formulation provides a nice theoretical setup, but in practice there does not exist a practical algorithm for finding a solution (Candes and Tao, 2005). Another alternative given

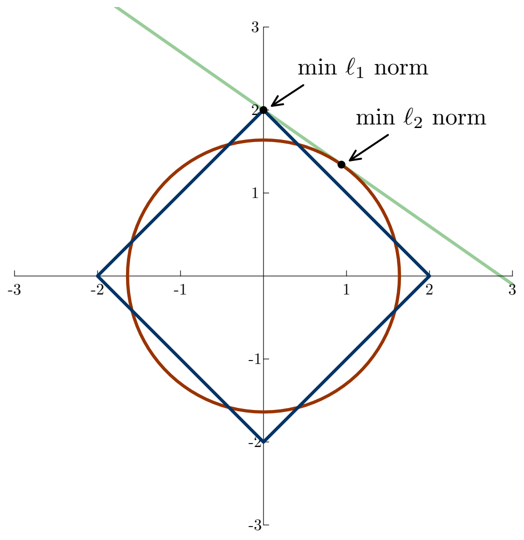


Figure 1.8: Simple example of how minimizing with respect to the $\ell_1$ norm finds a sparse solution. Figure is reconstructed from example found in Nelson et al. (2013).

in Candes et al. (2006), is to use the $\ell_1$ norm in the problem setup

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} \left\{ \|\mathbf{y} - \mathbf{\Phi}\mathbf{w}\|_2^2 + \tau \|\mathbf{w}\|_1 \right\} \tag{1.22}$$

where the $\ell_1$ norm is defined as

$$\|\mathbf{w}\|_1 = \sum_{i=1}^{N} |w_i|. \tag{1.23}$$

To see how using this norm will still find a sparse solution, consider the following simple example from Nelson et al. (2013). The equation $10y + 7x = 20$ has infinitely many solutions, which can be visualized as a line passing through the plane as shown in Figure 1.8. As can be seen from the figure, if we were to use the $\ell_2$ norm as a constraint on the unknown variables, then the solution would be where the circle (representing the $\ell_2$ norm) intersects the line. This would give us a dense solution where neither variable is zero. However, if we use the $\ell_1$ norm, then the solution will be where the diamond (representing the $\ell_1$ norm) intersects the line. This intersection happens to occur on the y-axis, implying our solution is sparse. Although this example is only in two dimensions, the principle can be generalized to much higher dimensions where most applications occur.

As mentioned before, one of the key elements that allows CS to work, is assuming the signal is sparse with respect to some basis. This extra information allows us to solve an underdetermined system of equations. Another important element of CS however, is to ensure the matrix $\mathbf{\Phi}$ has certain properties. One of these properties is called the restricted isometry property (RIP). This property was introduced by Candes and Tao (2005) and says that a matrix satisfies the RIP of order $k$ if there exists a $\delta_k \in (0,1)$ such that

$$(1-\delta_k)\left\|\mathbf{x}\right\|_2^2 \leq \left\|\mathbf{\Phi x}\right\|_2^2 \leq (1+\delta_k)\left\|\mathbf{x}\right\|_2^2. \tag{1.24}$$

This property is important because it guarantees the existence of efficient algorithms that can differentiate between different sparse vectors, allowing us to recover the original signal (Candes and Wakin, 2008). To understand this concept better, we first look at the null space of $\mathbf{\Phi}$ defined as

$$\mathcal{N}(\mathbf{\Phi}) = \big\{\mathbf{x} : \mathbf{\Phi x} = 0\big\}. \tag{1.25}$$

As explained by Davenport et al., a potential undesirable property of the system of equations in (1.20) is if $\mathbf{y} = \mathbf{\Phi w} = \mathbf{\Phi w}'$ for two distinct $k$-sparse signals $\mathbf{w}$ and $\mathbf{w}'$. If this were the case, given our sample $\mathbf{y}$ it would be impossible to distinguish between $\mathbf{w}$ and $\mathbf{w}'$, and therefore we would not be able to recover all possible signals. This is related to the null space of $\mathbf{\Phi}$ by observing that if $\mathbf{\Phi w} = \mathbf{\Phi w}'$ then $\mathbf{\Phi}(\mathbf{w}-\mathbf{w}') = 0$ where the $2k$-sparse signal $\mathbf{w}-\mathbf{w}'$ is in the null space of $\mathbf{\Phi}$. The property we then desire of $\mathbf{\Phi}$ is that its null space does not contain any $2k$-sparse signals, which then implies that $\mathbf{\Phi w} \neq \mathbf{\Phi w}'$, further implying that $\mathbf{\Phi}$ "uniquely represents" all $k$-sparse signals. The matrix $\mathbf{\Phi}$ will satisfy this property if it has the RIP.

In practice, it can be hard to verify if a matrix has the RIP. Fortunately, there are other, more easily calculated properties that still allow for recovery guarantees (Davenport et al., 2012). One of these properties is coherence. The coherence of a matrix $\mathbf{\Phi}$ is the largest absolute inner product between any two columns of $\mathbf{\Phi}$ (Davenport et al., 2012)

$$\mu(\mathbf{\Phi}) = \max_{1\leq i \leq j \leq N} \frac{|\langle \phi_i, \phi_j \rangle|}{\left\|\phi_i\right\|_2 \left\|\phi_j\right\|_2}. \tag{1.26}$$

This value is bounded between $\left[\sqrt{\frac{N-M}{M(N-1)}}, 1\right]$, where $M$ is the number of rows in $\mathbf{\Phi}$ and $N$ is the number of columns. This is a general definition of coherence. Candes and Wakin discuss the case where $\mathbf{\Phi} = \mathbf{\Phi}'\mathbf{\Psi}$ and $\mathbf{\Phi}', \mathbf{\Psi}$ are orthonormal bases, and define the coherence as

$$\mu(\mathbf{\Phi}', \mathbf{\Psi}) = \sqrt{N} \cdot \max_{1 \leq i,j \leq N} |\langle \phi_i', \psi_j \rangle| \tag{1.27}$$

where $\mu$ is bounded by $\left[1, \sqrt{N}\right]$. The interpretation of coherence in this format is the measure of the largest correlation between any two columns of $\mathbf{\Phi}'$ and $\mathbf{\Psi}$. The smaller this measure, the more incoherent the matrices are, and the better the reconstruction of the signal is.

Intuitively, as discussed in Compressed Sensing (n.d.), if the sparse basis and sensing basis are incoherent, then the sensing basis does a good job of "spreading out" the information found in the sparse signal, throughout the sensing domain. This is desirable because each of the under-sampled measurements we take in the sensing domain will then contain different parts of the sparse signal, and reconstructing the signal will be possible.

There are various pairs of sensing and representation matrices that provide maximal incoherence. One combination, as suggested by Candes and Wakin (2008), is to let $\mathbf{\Psi}$ be a fixed orthonormal basis, and $\mathbf{\Phi}'$ a $M \times N$ measurement matrix where each column is drawn from a uniform distribution and then normalized. When this is the case, with overwhelming probability, the matrix $\mathbf{\Phi} = \mathbf{\Phi}'\mathbf{\Psi}$ will have the RIP. In our analysis, we generate the same type of measurement matrix $\mathbf{\Phi}'$ as above, but our representation matrix is not necessarily orthonormal. We will still make the assumption however, that the matrix $\mathbf{\Phi}$ has the desired properties.

### 1.4 Bayesian Statistics

Bayesian statistics is based on Bayes formula, originally discovered by Thomas Bayes in the 1700's. The formula is easily derived using basic probability laws. From Casella and Berger (2002), first let $A_1, A_2, \ldots$ be a partition of the sample space, and $B$ be any set.

Using conditional probability and letting $i = 1, 2, \ldots$, we can write

$$P(A_i|B) = \frac{P(A_i \cap B)}{P(B)}. \tag{1.28}$$

The probability $P(A_i \cap B)$ can be written as $P(B \cap A_i)$ which can then be rewritten using conditional probability again. We also can use the law of total probability to rewrite $P(B)$ giving us

$$P(A_i|B) = \frac{P(B|A_i)P(A_i)}{\sum_{j=1}^{\infty} P(B \cap A_j)} = \frac{P(B|A_i)P(A_i)}{\sum_{j=1}^{\infty} P(B|A_j)P(A_j)}. \tag{1.29}$$

To understand better how this formula is used, a simple example is given from Casella and Berger (2002). In Morse code there are two main symbols used: dots and dashes. Say that the probability that a dot is sent is $\frac{3}{7}$ and the probability that a dash is sent is $\frac{4}{7}$. Also, say that sometimes, due to interference, a dot is mistakenly interpreted as a dash and a dash as a dot. Say the probability for either of these events is $\frac{1}{8}$. Since we know that there is a chance that we might get the wrong symbol, we may want to quantify, for example, the probability that a dot was actually sent, given a dot was received. Using Bayes formula we have

$$P(\text{dot was sent}|\text{dot was received}) = \frac{P(\text{dot was received}|\text{dot was sent})P(\text{dot was sent})}{P(\text{dot was received})}. \tag{1.30}$$

We know from the law of total probability that

$$\begin{aligned} P(\text{dot was received}) &= P(\text{dot was received}|\text{dot was sent})P(\text{dot was sent}) \\ &\quad + P(\text{dot was received}|\text{dash was sent})P(\text{dash was sent}). \end{aligned} \tag{1.31}$$

Plugging in the appropriate values we get

$$P(\text{dot was sent}|\text{dot was received}) = \frac{\frac{7}{8} \times \frac{3}{7}}{\frac{7}{8} \times \frac{3}{7} + \frac{1}{8} \times \frac{4}{7}} = \frac{21}{25}. \tag{1.32}$$

We can therefore feel fairly confident that when we receive a dot it was actually sent as a dot.

Bayes rule generalizes beyond simple discrete events. The formula works with probability distributions as well. One of the key differences between classical and Bayesian statistics is that a parameter in classical statistics is considered an unknown, but fixed quantity while in Bayesian statistics the parameter of interest is considered an unknown, random variable with a distribution. With this idea in mind, we can write Bayes formula from Casella and Berger (2002) again as

$$\pi(\theta|\mathbf{x}) = \frac{f(\mathbf{x}|\theta)\pi(\theta)}{m(\mathbf{x})} \tag{1.33}$$

where $\pi(\theta|\mathbf{x})$ is known as the posterior distribution over the parameter of interest $\theta$, $\pi(\theta)$ is the prior distribution, and

$$m(\mathbf{x}) = \int f(\mathbf{x}|\theta)\pi(\theta)d\theta \tag{1.34}$$

or the marginal distribution of $\mathbf{x}$. The term $f(\mathbf{x}|\theta)$ refers to the sampling distribution, or the distribution the data are drawn from. However, because the unknown quantity in this term is not $\mathbf{x}$ but really $\theta$, it does not have the properties of a true density function, and so it is sometimes referred to as the likelihood function (Lee, 2012).

Since $\theta$ is integrated out of the term $m(\mathbf{x})$, we can consider $m(\mathbf{x})$ a normalizing constant with respect to the posterior density $\pi(\theta|\mathbf{x})$. This implies that if we drop this term the posterior will no longer be a true density function. However, dropping the term does not affect our final analysis since point and interval estimates will be the same. Therefore, we can simplify Bayes formula to

$$\pi(\theta|\mathbf{x}) \propto f(\mathbf{x}|\theta)\pi(\theta). \tag{1.35}$$

### 1.4.1 Conjugacy

In some cases, the posterior and prior distributions are found to be from the same family of distributions. This occurs when the sampling distribution and the prior distribution are conjugate pairs. A simple example of a conjugate pair is given in Nelson et al. (2013). The likelihood in this case is the normal with mean $\mu$ and variance $\sigma^2$, and the prior is an inverse gamma on $\sigma^2$ with parameters $\alpha$, $\beta$. When we combine these two distributions together, we get an inverse gamma posterior with parameters $\alpha_{\text{post}} = \alpha + \frac{n}{2}$ and $\beta_{\text{post}} = \beta + \frac{\sum_{i=1}^{n}(y_i-\mu)^2}{2}$, where the $y_i$ come from a random sample of size $n$.

If we are able to use distributions that are conjugate pairs, we are able to simplify our analysis and find an analytical form for the posterior parameters. Fortunately, this is the case in our analysis. However, many applications in the real world require using distributions that are not conjugate pairs. If this occurs, computing techniques such as Markov chain Monte Carlo (MCMC) methods are required.

# Chapter 2
# Methods and Model

In this chapter we first discuss the form of the simulated signal used throughout the analysis and then discuss Bayesian compressive sensing (BCS), a Bayesian approach to solving the compressive sensing problem. Our model uses BCS by first taking the given signal and decomposing it into basis functions. As was mentioned earlier, we focus on the Fourier, wavelet, and B-spline bases, but other bases could be used. Instead of truncating the basis decomposition, as is sometimes done, we instead consider essentially all possible basis functions and then use BCS to select the most important or relevant coefficients. This gives us a sparse representation of the signal that is robust to noise, providing a method for identifying signals.

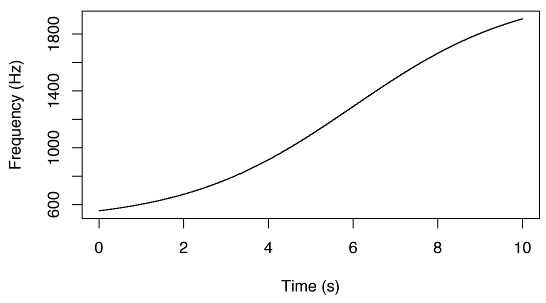
## 2.1 Description of Signal

The signal used throughout this analysis was provided by Todd Moon, a professor of electrical engineering at Utah State University, and is similar to the signal described in the paper Moon et al. (2015). As mentioned in this paper, the signal aims to simulate the output of a physical nonlinear device. Its form can be written as

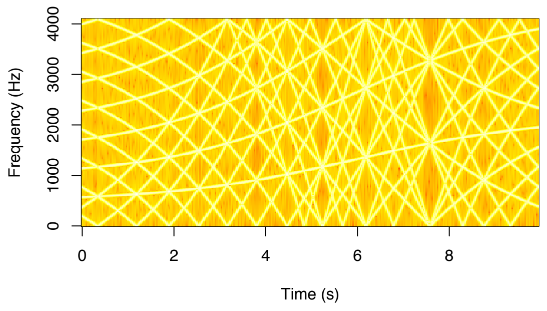
$$s(t) = \sum_{\eta=1}^{N_\eta} a_\eta[\cos(2\pi\eta(\int_0^t f(\tau)d\tau)]. \tag{2.1}$$

The function $f$ represents the *fundamental frequency* (Moon et al., 2015). Frequency, as defined in Chapter 1, is the number of cycles of a signal in unit time, and is the reciprocal of the *period*, or the duration of one cycle. For a signal with multiple frequencies, the fundamental frequency then is defined to be the smallest frequency of the signal. Integer multiples of this value are called *harmonics*, which are represented by $\eta$ in (2.1) (Moon et al., 2015). The final signal sums over fifteen harmonics. The variable $a_\eta$ represents the harmonic dependent amplitudes (Moon et al., 2015), but can be ignored since all amplitudes are set to one in the actual signal used for simulations.

One aspect that makes this signal unique is that $f$ is a function that varies with time, meaning the frequency changes with time, and therefore can be classified as a *chirp* signal. In the signal provided, $f$ is derived from the tanh function, with a lower frequency bound at 480 Hz and upper frequency bound at 2100 Hz. A plot of the fundamental frequency is shown in Figure 2.1(a), together with the spectrogram of the signal showing all frequencies in Figure 2.1(b).



(a) Fundamental frequency



(b) Spectrogram of signal

Figure 2.1: Fundamental frequency of the signal on the left, with the spectrogram showing the multiple harmonics on the right.

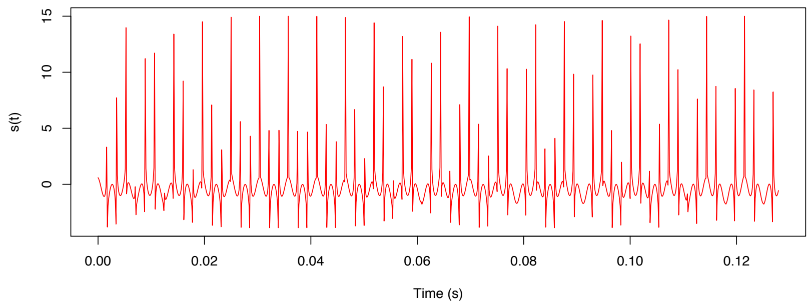


Figure 2.2: The first 1024 samples from the simulated signal.

The sampling frequency used to create the digital signal is set at 8000 Hz, or in other words, samples are taken uniformly every $\frac{1}{8000}$ second. This gives us 80001 samples over a ten second period starting at time zero and including time ten. In our simulations however, only the first 1024 values of the signal are used, as shown in Figure 2.2. The code used to create the signal is found in Appendix B.1.

As mentioned by Moon et al. (2015), one of the added challenges introduced by the sampling mechanism described above is aliasing. Aliasing occurs when the frequencies found in the signal are greater than the Nyquist frequency as described by Shannon's theorem. This implies that the sampling rate is not frequent enough to capture all aspects of the signal and a false reconstruction from the samples can occur. In our case, some of the harmonics found in the signal do surpass the Nyquist rate, so any reconstruction of the signal must take this into account.

## 2.2 Bayesian Compressive Sensing

Bayesian compressive sensing (BCS) is a Bayesian approach to solving the minimization problem (1.22) mentioned in Chapter 1. Using Bayesian statistics to solve this minimization problem provides a few significant advantages over more traditional methods (Nelson et al., 2013). First of all, the Bayesian approach eliminates the need for a tunable parameter $\tau$, allowing the model to be developed automatically. Secondly, the approach quantifies the error in the estimated coefficients and provides error bounds around these estimates. Thirdly, the approach allows for a more efficient algorithm, while still providing accurate estimates.

This section outlines the theory behind BCS, followed by an explanation of the algorithm used for the final model. Note that the majority of this section and the next comes from Babacan et al. (2010) and Nelson et al. (2013).

As mentioned in Chapter 1, Bayesian statistics requires determining a likelihood describing the distribution the data come from, as well as requiring prior distributions on the parameters of interest. Since our data $\mathbf{y} = \Phi\mathbf{w} + \mathbf{n}$ and if we assume $\mathbf{n} \sim \mathcal{N}(0, \sigma^2)$, then we can assume our data come from a normal as well and define our likelihood as a

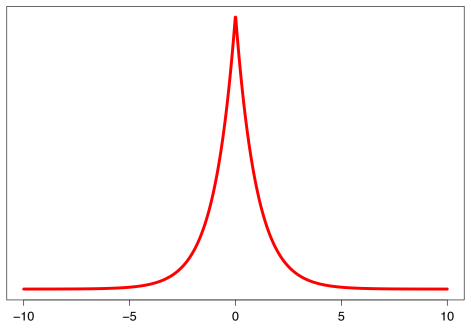


Figure 2.3: Example plot of the Laplace distribution.

multivariate normal

$$p(\mathbf{y}|\Phi\mathbf{w}, \sigma^2) = \mathcal{N}(\mathbf{y}|\Phi\mathbf{w}, \sigma^2 I). \tag{2.2}$$

Prior distributions must then be chosen on the parameters $\mathbf{w}$ and $\sigma^2$. For $\sigma^2$, the gamma distribution is used on the inverse variance or precision $\left(\frac{1}{\sigma^2}\right)$ in order to preserve conjugacy. The parametrization of the gamma distribution used follows the shape and rate parametrization

$$p\left(\frac{1}{\sigma^2}|\alpha, \beta\right) = \Gamma\left(\frac{1}{\sigma^2}|\alpha, \beta\right) \tag{2.3}$$

where $\alpha$ is the shape and $\beta$ is the rate.

When selecting a prior for $\mathbf{w}$, we must pay special attention to the assumption that $\mathbf{w}$ is sparse. One option that promotes sparsity is the Laplace distribution. As can be seen from Figure 2.3, the Laplace distribution contains most of its mass around zero, reflecting the assumption that the majority of coefficients $\mathbf{w}$ will be zero or close to zero.

The combination of the normal likelihood in (2.2), the Laplace prior distribution, and the gamma prior on $\frac{1}{\sigma^2}$, gives the posterior distribution of $\mathbf{w}$. Finding the *maximum a posterior* (MAP) of this posterior distribution is equivalent to solving the minimization problem in (1.22) (See Appendix A.2). However, the Laplace is not a conjugate prior to the normal likelihood and so in order to find the MAP, methods such as MCMC would need to be utilized, and would therefore decrease the computational efficiency of the model. In order to avoid this, a combination of priors are used equaling the Laplace distribution and therefore still enforcing sparsity, but also providing analytical expressions for the posterior parameters.

This hierarchal setup is accomplished by first using the normal distribution for the coefficients $\mathbf{w}$

$$p(\mathbf{w}|\gamma) = \prod_{i=1}^{N} \mathcal{N}(w_i|0, \gamma_i) \tag{2.4}$$

and then using the gamma distribution for the hyperpriors $\gamma_i$

$$p(\gamma|\lambda) = \prod_{i=1}^{N} \Gamma(\gamma_i|1, \frac{\lambda}{2}). \tag{2.5}$$

We then use the gamma distribution again for the hyperprior $\lambda$

$$p(\lambda|\nu) = \Gamma(\lambda|\frac{\nu}{2}, \frac{\nu}{2}). \tag{2.6}$$

Using Bayes formula and putting the likelihood and all priors together gives us

$$\begin{aligned} p(\mathbf{w}, \gamma, \lambda, \sigma^2|\mathbf{y}) &\propto p(\mathbf{y}|\mathbf{w}, \gamma, \lambda, \sigma^2)p(\mathbf{w}, \gamma, \lambda, \sigma^2) \\ &= p(\mathbf{y}|\mathbf{w}, \sigma^2)p(\mathbf{w}, \gamma, \lambda)p(\sigma^2) \\ &= p(\mathbf{y}|\mathbf{w}, \sigma^2)p(\mathbf{w}|\gamma, \lambda)p(\gamma, \lambda)p(\sigma^2) \\ &= p(\mathbf{y}|\mathbf{w}, \sigma^2)p(\mathbf{w}|\gamma)p(\gamma|\lambda)p(\lambda)p(\sigma^2). \end{aligned}$$

Substituting in (2.2), (2.3), (2.4), (2.5), and (2.6) we have for our final model

$$p(\mathbf{w}, \sigma^2, \gamma, \lambda|\mathbf{y}) = \mathcal{N}(\mathbf{y}|\Phi\mathbf{w}, \sigma^2)\left[\prod_{i=1}^{N} \mathcal{N}(w_i|0, \gamma_i)\right]\left[\prod_{i=1}^{N} \Gamma\Big(\gamma_i|1, \frac{\lambda}{2}\Big)\right]\Gamma\Big(\lambda|\frac{\nu}{2}, \frac{\nu}{2}\Big)\Gamma\Big(\frac{1}{\sigma^2}|\alpha, \beta\Big). \tag{2.7}$$

In order to increase computational speed and provide a convenient distribution for the coefficients $\mathbf{w}$ (Nelson et al., 2013), we decompose the above full posterior distribution into two separate distributions

$$p(\mathbf{w}, \sigma^2, \gamma, \lambda|\mathbf{y}) = p(\mathbf{w}|\sigma^2, \gamma, \lambda, \mathbf{y})p(\sigma^2, \gamma, \lambda|\mathbf{y}). \tag{2.8}$$

The probability $p(\mathbf{w}|\sigma^2, \gamma, \lambda, \mathbf{y})$ can be shown to be $\mathcal{N}(\mathbf{w}|\mu, \mathbf{\Sigma})$ (See Appendix A.3) where

$$\mu = \frac{1}{\sigma^2}\mathbf{\Sigma}\mathbf{\Phi}'\mathbf{y} \tag{2.9}$$

$$\mathbf{\Sigma} = \left[\frac{1}{\sigma^2}\mathbf{\Phi}'\mathbf{\Phi} + \mathbf{\Lambda}\right]^{-1} \tag{2.10}$$

$$\mathbf{\Lambda} = \text{diag}\left(\frac{1}{\gamma_i}\right). \tag{2.11}$$

We now have a distribution on just the coefficients $\mathbf{w}$ that we can use to find the most probable coefficients, and also provide error intervals around. Notice however, that the distribution depends on the nuisance parameters $\sigma^2, \gamma$, and $\lambda$. To find values for these parameters, we can use the distribution $p(\sigma^2, \gamma, \lambda|\mathbf{y})$. In order to find this distribution, we first use Bayes' rule to get

$$p(\sigma^2, \gamma, \lambda|\mathbf{y}) \propto p(\mathbf{y}|\sigma^2, \gamma, \lambda)p(\gamma|\lambda)p(\lambda)p(\sigma^2). \tag{2.12}$$

The three priors in this equation are already known from before and $p(\mathbf{y}|\sigma^2, \gamma, \lambda)$, known as the marginal likelihood, can be shown to be $\mathcal{N}(\mathbf{y}|0, C)$, where $C = \frac{1}{\sigma^2}I_M + \mathbf{\Phi}\mathbf{\Lambda}^{-1}\mathbf{\Phi}'$ (See Appendix A.3).

Once we have the distribution $p(\sigma^2, \gamma, \lambda|\mathbf{y})$, we then find the optimal values for $\sigma^2$, $\gamma$, and $\lambda$, by taking derivatives of the log of the density with respect to each parameter and setting equal to zero. This gives us separate expressions for the optimal values of each parameter. These expressions are dependent on the other parameters however, and so an iterative algorithm is used to update each parameter based on the updated values of the other parameters. The parameters $\mu$ and $\mathbf{\Sigma}$ from the distribution $p(\mathbf{w}|\sigma^2, \gamma, \lambda, \mathbf{y})$ are also updated at each iteration, which allows us to eventually find a final distribution on the coefficients $\mathbf{w}$ along with error bounds.

## 2.3 A Practical Algorithm

One issue with updating the entire matrix $\mathbf{\Sigma}$, as described above, is the heavy computational burden of performing a matrix inversion, especially when $N$ is a relatively large number (Nelson et al., 2013). Doing so would slow the algorithm down and make it less practical for real world problems. A way around this is described by Babacan et al. (2010). Instead of updating entire vectors at each iteration, such as the vector $\gamma$, we need only update a single $\gamma_i$ at a time. This allows us to update $\mu$ and $\mathbf{\Sigma}$ in an efficient way and avoid a full matrix inversion. An important observation to make that allows this efficient updating to take place, is that if $\gamma_i$ is zero, then $w_i$ is zero, and the corresponding coefficient in the final model is zero (Nelson et al., 2013). We began with the assumption that most of the coefficients in our final model will be zero, so therefore most of the $\gamma_i$ will be zero, implying that $\mu$ and $\mathbf{\Sigma}$ can be represented in much fewer dimensions. We therefore start the iterative algorithm with all $\gamma_i = 0$, and then update a single $\gamma_i$ at a time, followed by the appropriate updates on the other parameters.

For us to decide which $\gamma_i$ to update, we need to know which $\gamma_i$ results in the largest increase in the log of the density function found in (2.12), which requires calculating the optimal value of each $\gamma_i$ at each iteration. To find the expression for a given $\gamma_i$, first take the log of the density $p(\sigma^2, \gamma, \lambda|\mathbf{y})$

$$\begin{aligned}\mathcal{L} = {} & \frac{N}{2}\log\frac{1}{2\pi} - \frac{1}{2}\log|C| - \frac{1}{2}\mathbf{y}'C^{-1}\mathbf{y} + N\log\frac{\lambda}{2} - \frac{\lambda}{2}\sum_{i=1}^{N}\gamma_i \\ & + \frac{\nu}{2}\log\frac{\nu}{2} - \log\Gamma\left(\frac{\nu}{2}\right) + \left(\frac{\nu}{2} - 1\right)\log\lambda - \frac{\nu}{2}\lambda \\ & + \alpha\log\beta - \log\Gamma(\alpha) + (\alpha - 1)\log\frac{1}{\sigma^2} - \frac{\beta}{\sigma^2}.\end{aligned} \tag{2.13}$$

In order to find optimal values for each $\gamma_i$, we must take the derivative with respect to each $\gamma_i$. This requires us then to find a way to rewrite $\mathcal{L}$ in terms of a particular $\gamma_i$ separated from all of the other $\gamma_i$, and to be able to do this for each $\gamma_i$. To rewrite $\mathcal{L}$, we first rewrite

$C$ by separating a particular $\gamma_i$ from the other $\gamma_i$ (Babacan et al., 2010)

$$\begin{aligned} C &= \sigma^2 I_M + \sum_{i=1}^{N} \gamma_i \phi_i \phi_i' \\ &= \sigma^2 I_M + \sum_{\substack{j=1 \\ j \neq i}}^{N} \gamma_j \phi_j \phi_j' + \gamma_i \phi_i \phi_i' \\ &= C_{-i} + \gamma_i \phi_i \phi_i'. \end{aligned} \tag{2.14}$$

We can rewrite $C^{-1}$ as well using (2.14) as the definition of $C$. Applying the Woodbury identity referenced in Appendix A.1 and letting $D = C_{-i}$, $U = \phi_i$, $E = \gamma_i$, and $V = \phi_i'$, we have

$$(C_{-i} + \gamma_i \phi_i \phi_i')^{-1} = C_{-i}^{-1} - C_{-i}^{-1} \phi_i \left( \frac{1}{\gamma_i} + \phi_i' C_{-i}^{-1} \phi_i \right)^{-1} \phi_i' C_{-i}^{-1}. \tag{2.15}$$

Using the same definition for $C$ (2.14) we can also rewrite $|C|$ by first factoring out $C_{-i}$

$$|C| = |C_{-i}||I_M + C_{-i}^{-1} \gamma_i \phi_i \phi_i'|.$$

Looking at the second determinant, we apply the Sylvester determinant identity referenced in Appendix A.1. If we let $A = C_{-i}^{-1}\phi_i$ and $B = \phi_i'$ then we have

$$|C_{-i}||I_M + C_{-i}^{-1} \gamma_i \phi_i \phi_i'| = |C_{-i}||1 + \gamma_i \phi_i' C_{-i}^{-1} \phi_i|. \tag{2.16}$$

Note that the last determinant is now a scalar so we can drop the determinant signs.

We can now rewrite $\mathcal{L}$ by plugging in the alternative definitions defined above and dropping the terms that do not include $\gamma$, since these evaluate to zero in the derivative of $\gamma_i$

$$\mathcal{L} = -\frac{1}{2} \log \Big[ |C_{-i}|(1 + \gamma_i \phi_i' C_{-i}^{-1} \phi_i) \Big] - \frac{1}{2} \mathbf{y}' \left[ C_{-i}^{-1} - C_{-i}^{-1} \phi_i \left( \frac{1}{\gamma_i} + \phi_i' C_{-i}^{-1} \phi_i \right)^{-1} \phi_i' C_{-i}^{-1} \right] \mathbf{y} - \frac{\lambda}{2} \sum_{i=1}^{N} \gamma_i.$$

Focusing on the second term, and simplifying we have

$$-\frac{1}{2} \mathbf{y}' C_{-i}^{-1} \mathbf{y} + \frac{1}{2} \mathbf{y}' \frac{\gamma_i C_{-i}^{-1} \phi_i \phi_i' C_{-i}^{-1}}{1 + \gamma_i \phi_i' C_{-i}^{-1} \phi_i} \mathbf{y}. \tag{2.17}$$

If we then introduce the terms

$$\begin{aligned} s_i &= \phi_i' C_{-i}^{-1} \phi_i \\ q_i &= \phi_i' C_{-i}^{-1} \mathbf{y} \end{aligned} \tag{2.18}$$

we can simplify $\mathcal{L}$ further

$$\mathcal{L} = -\frac{1}{2}\log|C_{-i}| - \frac{1}{2}\log(1+\gamma_i s_i) - \frac{1}{2}\mathbf{y}' C_{-i}^{-1}\mathbf{y} + \frac{1}{2}\frac{\gamma_i q_i^2}{(1+\gamma_i s_i)} - \frac{\lambda}{2}\sum_{\substack{j=1\\ j\neq i}}^{N}\gamma_j - \frac{\lambda}{2}\gamma_i.$$

We now have a form of $\mathcal{L}$ that can be separated into terms that are independent of $\gamma_i$ and terms that depend on $\gamma_i$. To see these more clearly we can write $\mathcal{L} = \mathcal{L}(\gamma_{-i}) + l(\gamma_i)$ where

$$\begin{aligned} \mathcal{L}(\gamma_{-i}) &= -\frac{1}{2}\log|C_{-i}| - \frac{1}{2}\mathbf{y}' C_{-i}^{-1}\mathbf{y} - \frac{\lambda}{2}\sum_{\substack{j=1\\ j\neq i}}^{N}\gamma_j \\ l(\gamma_i) &= -\frac{1}{2}\log(1+\gamma_i s_i) + \frac{1}{2}\frac{\gamma_i q_i^2}{(1+\gamma_i s_i)} - \frac{\lambda}{2}\gamma_i. \end{aligned} \tag{2.19}$$

where $\mathcal{L}(\gamma_{-i})$ is independent of $\gamma_i$ and $l(\gamma_i)$ is dependent on $\gamma_i$.

We can then take the derivative of $\mathcal{L}$ with respect to $\gamma_i$

$$\begin{aligned} \frac{d\mathcal{L}}{d\gamma_i} = \frac{dl(\gamma_i)}{d\gamma_i} &= -\frac{s_i}{2(1+\gamma_i s_i)} + \frac{q_i^2}{2(1+\gamma_i s_i)^2} - \frac{\lambda}{2} \\ &= \frac{-s_i - \gamma_i s_i^2 + q_i^2 - \lambda - 2\lambda\gamma_i s_i - \lambda\gamma_i^2 s_i^2}{2(1+\gamma_i s_i)^2} \\ &= -\frac{1}{2}\frac{\gamma_i^2 s_i^2\lambda + \gamma_i(s_i^2 + 2\lambda s_i) - q_i^2 + \lambda + s_i}{(1+\gamma_i s_i)^2}. \end{aligned} \tag{2.20}$$

Setting this entire equation to zero is equivalent to setting the numerator to zero, and so using the quadratic formula on the numerator we can find the optimal values of $\gamma_i$

$$\begin{aligned} \gamma_i &= \frac{-(s_i^2 + 2\lambda s_i) \pm \sqrt{(s_i^2 + \lambda 2 s_i)^2 - 4 s_i^2\lambda(-q_i^2 + \lambda + s_i)}}{2 s_i^2 \lambda} \\ &= \frac{-s_i(s_i + 2\lambda) \pm s_i\sqrt{(s_i + 2\lambda)^2 - 4\lambda(-q_i^2 + \lambda + s_i)}}{2 s_i^2\lambda}. \end{aligned} \tag{2.21}$$

To simplify, let $\Delta = (s_i + 2\lambda)^2 - 4\lambda(-q_i^2 + \lambda + s_i)$.

Notice that the signs of the two critical values depend on $4\lambda(-q_i^2 + \lambda + s_i)$. If this term is found to be zero, which happens when -$q_i^2 = \lambda + s_i$, then the radical $\sqrt{\Delta}$ simplifies to $(s_i + 2\lambda)$, and we are left with a negative value and zero for the critical points. If $4\lambda(-q_i^2 + \lambda + s_i)$ is positive, which occurs when $q_i^2 - s_i < \lambda$, then $s_i\sqrt{\Delta}$ will be a smaller value than $s_i(s_i + 2\lambda)$, and we will have two negative values for the critical points. Lastly, if $q_i^2 - s_i > \lambda$ then $4\lambda(-q_i^2 + \lambda + s_i)$ is negative, and $s_i\sqrt{\Delta}$ will be a larger number than $s_i(s_i + 2\lambda)$, implying that we will have a positive and a negative value for the critical values.

In each of these cases, recall that $\gamma_i \geq 0$ by definition. When evaluating the case where both critical points are negative ($q_i^2 - s_i < \lambda$), we first evaluate the derivative (2.20) at $\gamma_i = 0$, and observe that the slope is negative. Since the only two critical points are both negative, we can assume that the log function will continue to slope downward from $\gamma_i = 0$, for all positive $\gamma_i$. Therefore, over the support of $\gamma_i$, the optimal value is zero. For the case where $q_i^2 - s_i > \lambda$, we only consider the positive critical point and test to see if it is truly a max by evaluating the derivative at zero and infinity. Since the values from these tests are positive and negative respectively, then we know that the positive critical point for $\gamma_i$ is indeed the max. Following this same reasoning, the final case where $q_i^2 = \lambda + s$, implies that zero is the optimal $\gamma_i$.

In summary, we can write the optimal $\gamma_i$ in the following, concise form

$$\gamma_i = \begin{cases} \frac{-s_i(s_i+2\lambda)+s_i\sqrt{(s_i+2\lambda)^2-4\lambda(-q_i^2+\lambda+s_i)}}{2s_i^2\lambda} & \text{if } q_i^2 - s_i > \lambda \\ 0 & \text{otherwise.} \end{cases} \tag{2.22}$$

The optimal hyperparameter $\lambda$ is also found by taking the derivative, this time with respect to $\lambda$ of the log function in the form of (2.13). This gives us

$$\lambda = \frac{N - 1 + \frac{\nu}{2}}{\frac{\sum_{i=1}^{N}\gamma_i}{2} + \frac{\nu}{2}}. \tag{2.23}$$

The remaining hyperparameters $\nu$ and $\sigma^2$ are given fixed values at the beginning of the algorithm, instead of using the expressions found from taking derivatives. As discussed by Babacan et al. (2010), for the parameter $\nu$ experiments show that setting the parameter $\nu = 0$ provides slightly better performance than using the expression found by the derivative. For the parameter $\sigma^2$, a fixed value is also given at the beginning, specifically $\frac{1}{\sigma^2} = 0.01 \|\mathbf{y}\|_2^2$ (Babacan et al., 2010). Fixing $\sigma^2$ is needed because the algorithm is constructive and so initial estimates of $\sigma^2$ are unreliable. If the updated estimates become too far off, then the reconstruction accuracy will be off as well.

**Algorithm 1** Fast Laplace

```
 1: Initialize all $\gamma_i = 0$ and $\lambda = 0$.
 2: while not converged do
 3:     Calculate all optimal $\gamma_i$
 4:     Choose $\gamma_i$ that results in the greatest increase of (2.19)
 5:     if $q_i^2 - s_i > \lambda$ and $\gamma_i = 0$ then
 6:         Add $\gamma_i$ to model
 7:     else if $q_i^2 - s_i > \lambda$ and $\gamma_i > 0$ then
 8:         Re-estimate $\gamma_i$
 9:     else
10:         Delete basis by setting $\gamma_i = 0$
11:     Update parameters $\mathbf{\Sigma}, \mu, q_i, s_i, \lambda$
```

The summary of the Fast Laplace algorithm above is based off the summary from Babacan et al. (2010). The convergence criteria we use follows code also associated with Babacan et al. (2010) (see Appendix B.2). This criteria ends the algorithm when the absolute difference between the max log-likelihood (2.19) from the current iteration and the max log-likelihood from the previous iteration is less than a given threshold. Note that the update formula for $\mathbf{\Sigma}$, $\mu$, $q_i$ and $s_i$ are given in the appendix of Tipping and Faul (2003).

In summary, we now have a process that finds a distribution on the coefficients $\mathbf{w}$. This was accomplished by assuming a normal likelihood and a combination of priors that both enforce sparsity and avoid computationally intensive sampling methods. Once these distributions were chosen, we then decomposed the posterior into two convenient distributions - one distribution on just the coefficients $\mathbf{w}$ and another on the hyperparameters. Since the distribution on $\mathbf{w}$ is dependent on the hyperparameters, we take the log of the distribution on the hyperparameters and find optimal values for each parameter by taking the appropriate derivatives. An iterative procedure is then used to update the parameters and the distribution on $\mathbf{w}$. Once the final distribution on $\mathbf{w}$ is found, we use the mean vector $\mu$ as an estimate of the coefficients and the diagonal of $\mathbf{\Sigma}$ as error bounds on the coefficients.

# Chapter 3

# Results

One of the main goals of this analysis was to compare the performance of different basis function expansions with respect to the accuracy of the reconstructed signal, while paying particular attention to the sparsity of the different basis representations. There are a variety of different bases that could have been chosen, but this analysis specifically focuses on the Fourier, B-spline and wavelet bases. These bases were chosen mostly due to ease of implementation and popularity, especially the Fourier and wavelet bases, which are found in many signal processing applications.

The same signal was used throughout both experiments. Recall that the original signal has 80001 time points between zero and ten seconds, but only the first 1024 time points were used in our analysis (see Figure 2.2). This was done primarily to reduce the computational requirement of the experiments and more quickly analyze results. For each simulation and for each basis, the signal was decomposed into 8192 basis functions. Note that 8192 is a power of two ($2^{13} = 8192$) which makes the wavelet decomposition easier. Also, choosing a relatively large number of basis functions aligns with compressive sensing theory, forming a highly underdetermined matrix. The other two bases, Fourier and B-splines, were decomposed into the same number of basis functions in order to make a fair comparison across simulations, even though they can be easily decomposed into any number of basis functions.

## 3.1 Reconstruction Accuracy Results

One of the main results of compressive sensing is that only a few, random samples are needed in order to accurately reconstruct the original signal. In order to see how well this result holds for the given signal, a simulation was performed analyzing the reconstruction accuracy of each basis function decomposition as a function of sample size. Details of the simulation are given below, with code shown in Appendix B.3.

For each iteration of the simulation, a random $M \times P$ ($P = 1024$) sensing matrix $\mathbf{\Phi}'$ was generated, using code associated with Babacan et al. (2010) (see Appendix B.3), where each column is a normalized draw from the uniform distribution. This matrix is then multiplied by the basis representation matrix $\mathbf{\Psi}$ ($1024 \times 8192$), giving us the matrix $\mathbf{\Phi} = \mathbf{\Phi}'\mathbf{\Psi}$. Note that $\mathbf{\Psi}$ is the basis representation matrix for either the Fourier, B-spline or wavelet basis. The reconstruction accuracy is then evaluated at different sample sizes, or equivalently, at different values of $M = 100, 200, 300, \ldots, 1000$. This process is then repeated 49 more times and averaged over the 50 total iterations. Figure 3.1 shows the results of the simulation.

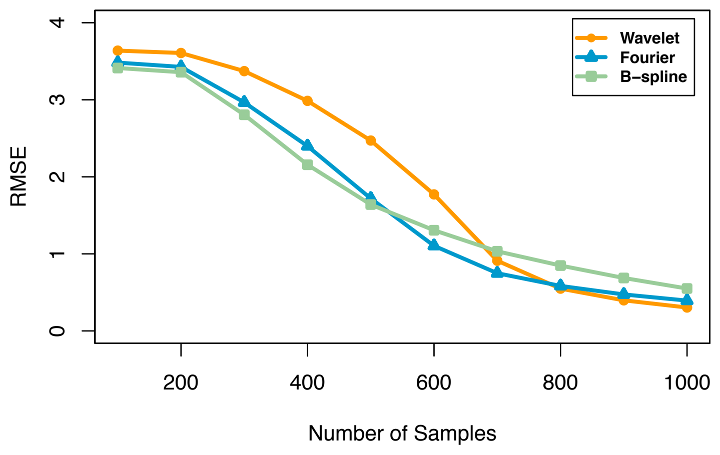


Figure 3.1: RMSE across number of samples. Notice that the wavelet basis tends to lag behind the Fourier and B-Spline basis for smaller samples, but converges with larger sample sizes.

In Figure 3.1 the y-axis is in terms of root-mean-square error (RMSE). This measure of error is given by

$$\text{RMSE}(\mathbf{y} - \hat{\mathbf{y}}) = \sqrt{\text{MSE}(\mathbf{y} - \hat{\mathbf{y}})} = \sqrt{E[(\mathbf{y} - \hat{\mathbf{y}})^2]}. \tag{3.1}$$

where $E$ is the expected value, $\mathbf{y}$ is the original signal, and $\hat{\mathbf{y}}$ is the reconstructed signal. This essentially gives the average distance between the original signal and reconstructed signal.

As can be seen from Figure 3.1, the performance of the three different bases is similar. The wavelet basis seems to perform more poorly than the Fourier and B-spline bases with smaller samples, but all bases essentially converge as the number of samples increases.

## 3.2 Sparsity Across Noise Results

The second part of our analysis compares the bases with regards to sparsity. A key assumption of compressive sensing is that the original signal is sparse in the representation basis. This is an assumption we make for our given signal using the bases we have chosen. Making this assumption allows us to use BCS as a modeling tool for finding the key components of the signal or a "sparse signature" of the signal, which we hope is robust to noise. We explore these concepts in this section and discuss the results of an experiment designed to partially assess these assumptions. Code is shown in Appendix B.3.

As mentioned in Chapter 1, the $\ell_0$ norm and $\ell_1$ norm are two different ways of measuring the sparsity of a signal. The $\ell_1$ norm, defined in (1.23), is the sum of the absolute values of the elements of the signal. The $\ell_0$ norm is simply the number of nonzero elements found in the signal. Using these two measurements of sparsity, we examine how the sparse representation found by BCS is affected as noise increases. We specifically looked at two types of noise: regular white Gaussian noise, and noise proportional to the mean. White Gaussian noise simply implies that the noise vector $\mathbf{n}$ in (1.20) is distributed as the normal distribution

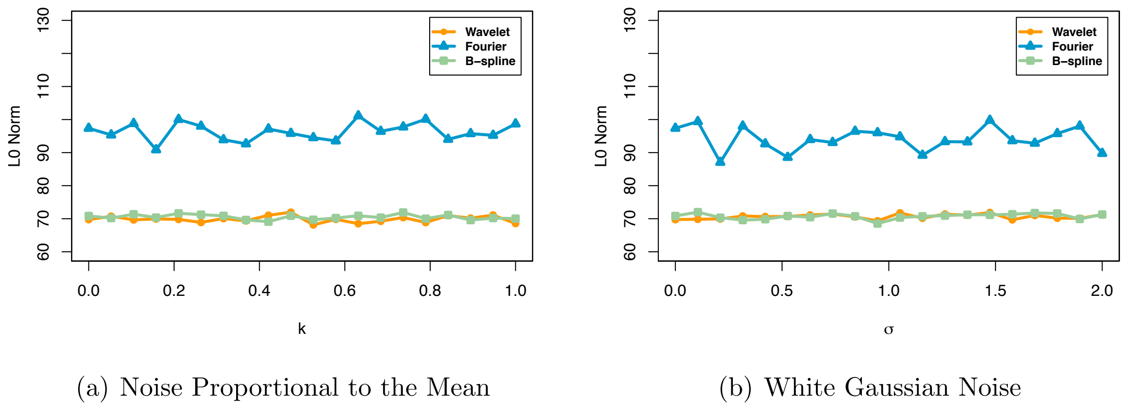
$$\mathbf{n} \sim \mathcal{N}(0, \sigma^2). \tag{3.2}$$

The term *white noise* is used to refer to the type of random noise that doesn't have a pattern, but is present in many environments. Noise proportional to the mean can be defined as

$$\mathbf{n} \sim \mathcal{N}(0, k\,|\mathbf{y}|) \tag{3.3}$$

where $k \in [0,1]$. This type of noise is dependent on the signal itself. If the absolute value of a point in the signal is higher, then the variance of the noise distribution will be higher at that point as well. Therefore, the noise is different across the signal, with higher and lower variances depending on the absolute value of the signal.

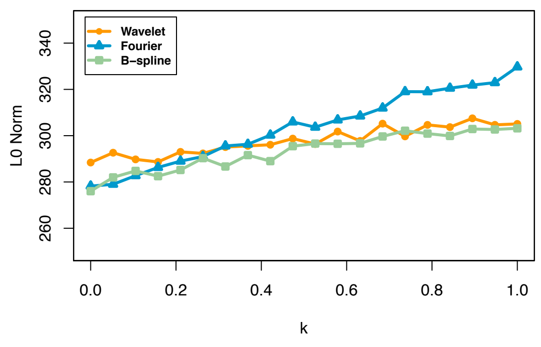
Figures 3.2(a) and 3.2(b) show how the $\ell_0$ norm varies as the different types of noise vary. The results here are found using the same signal as before, but setting the number of samples at 100 for all iterations and noise levels. In particular, for each iteration, a different random $100 \times 1024$ sensing matrix $\mathbf{\Phi}'$ is generated, which is then used to evaluate the sparsity of the model output at different noise levels, for the different bases. Results are averaged over 50 such iterations.



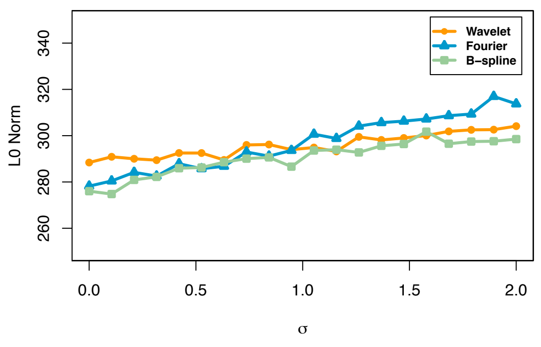
(a) Noise Proportional to the Mean (b) White Gaussian Noise

Figure 3.2: The $\ell_0$ norm remains essentially the same as noise increases, for both types of noise and for the different bases. Sample size here is 100.

Notice that both types of noise produce similar plots, with the sparsity of the Fourier basis decomposition more variable than both the B-spline and wavelet bases. The $\ell_0$ norm is essentially unchanged for all basis decompositions as noise is added to the signal. However, if we keep the same setup, but change the number of samples to 400, we get the plots shown in Figures 3.3(a) and 3.3(b). Instead of staying flat, the $\ell_0$ norm steadily increases for both types of noise. One interesting observation is that the sparsity of the Fourier basis decreases more quickly than the other two bases. This shows that added noise has a different effect for different bases.



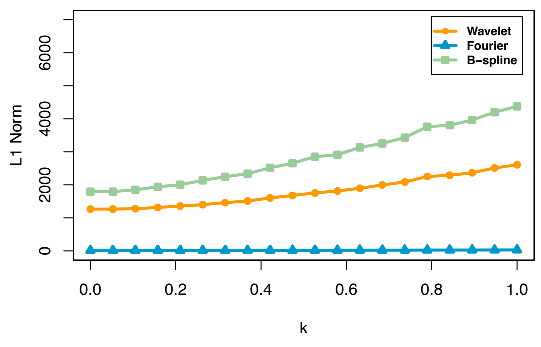
(a) Noise Proportional to the Mean



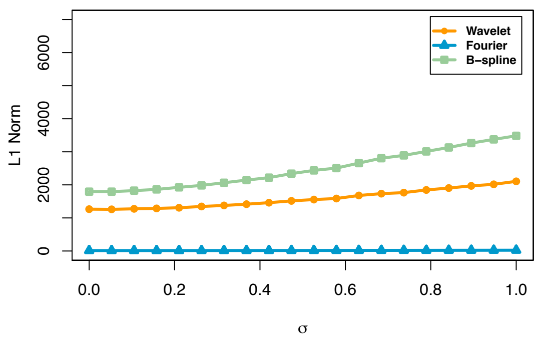
(b) White Gaussian Noise

Figure 3.3: The $\ell_0$ norm steadily increases when the sample size is 400.

If we keep the sample size at 400, but instead look at the $\ell_1$ norm as a measure of sparsity, we get the plots shown in Figures 3.4(a) and 3.4(b). One obvious difference between



(a) Noise Proportional to the Mean



(b) White Gaussian Noise

Figure 3.4: The $\ell_1$ norm steadily increases when the sample size is 400.

these plots and the $\ell_0$ norm plots, is that the bases differ greatly in terms of their measured sparsity. In the previous plots, when the sample size is 400, the $\ell_0$ norm is about the same for each basis, around 280 or 290 nonzero coefficients when there is no noise added. This leads us to conclude that the $\ell_1$ norm, in this case at least, is mainly influenced by the magnitude of the coefficients, which appears to depend on the basis. The $\ell_0$ norm therefore might be considered a more accurate and fair measure of sparsity since it is not influenced as much by the choice of basis.

In summary, if we choose to focus on the $\ell_0$ norm as the preferred method for measuring sparsity, then we can conclude from both Figures 3.2 and 3.3 that the model is robust to noise. We make this conclusion because in both figures the number of nonzero coefficients stays about the same or increases only a little, even as noise is increased. The model seems to be finding a common sparse representation that can still be found even in the presence of noise.

# Chapter 4

# Conclusions and Further Work

As mentioned previously, one of the main applications of remote sensing is to identify electromagnetic signals from a limited number of samples, in the presence of noise. These challenges led us to chose an approach involving compressive sensing, a relatively recent field in signal processing, to identify signals. The first step of our approach was to decompose the given signal into a basis function expansion: either the Fourier, B-spline, or wavelet basis functions. Then a Bayesian approach to compressive sensing (BCS) was used to find the potentially sparse representation of the signal in the chosen basis. This provided a model that essentially finds the “core elements” of the signal, or a sparse signature, and therefore provides a means of identifying signals.

We tested how well this model did in reconstructing the original signal and compared the reconstruction accuracy for the different bases. We also tested how well the model did with added noise. In both cases, we saw positive results that show a model that is robust to noise and a model with high reconstruction accuracy.

The performance of the different bases was found to be similar, in terms of reconstruction accuracy and change in sparsity across different levels of noise. This leads us to conclude that of the three bases we examined, there is not a particular one that is preferable to the other two. However, other bases might yield more satisfactory results.

These concepts show promise, but need to be explored more. One important direction to explore would be to find the sparse signatures of multiple electromagnetic signals and evaluate how effective the model is in identifying and distinguishing between the different signals, even with added noise. This would require a method that determines or measures how close one sparse signature is to another. A sparse signature library could then be created containing the sparse representation of various known signals. Given a new signal, the model described in this analysis could then find the sparse representation and match it to one of the known signals in the library.

# REFERENCES

Babacan, S. D., Molina, R., and Katsaggelos, A. K. (2010), "Bayesian Compressive Sensing Using Laplace Priors," *IEEE Transactions on Image Processing*, 19, 53–63. 25, 29, 30, 33, 36, 47, 52, 61

Bachman, G., Narici, L., and Beckenstein, E. (2000), *Fourier and Wavelet Analysis*, New York, NY: Springer. 6, 7, 8

Campbell, J. B. and Wynne, R. H. (2011), *Introduction to Remote Sensing, Fifth Edition*, New York, NY: The Guilford Press., 5th ed., Available from `http://www.ebrary.com`. 1, 2, 4

Candes, E. J., Romberg, J., and Tao, T. (2006), "Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information," *IEEE Transactions on Information Theory*, 52, 489–509. 16

Candes, E. J. and Tao, T. (2005), "Decoding by linear programming," *IEEE Transactions on Information Theory*, 51, 4203–4215. 15, 17

Candes, E. J. and Wakin, M. B. (2008), "An Introduction To Compressive Sampling," *IEEE Signal Process. Mag.*, 25, 21–30. 17, 18

Casella, G. and Berger, R. L. (2002), *Statistical Inference, Second Edition*, Pacific Grove, CA: Duxbury, 2nd ed. 18, 19, 20

Compressed Sensing (n.d.), Unpublished draft of lab manual for Brigham Young University's Applied and Computational Mathematics program. Available at `http://www.acme.byu.edu/wp-content/uploads/2014/09/Vol2Lab17CompressedSensing.pdf`. 18

Davenport, M. A., Duarte, M. F., Eldar, Y. C., and Kutyniok, G. (2012), "Introduction to compressed sensing," in *Compressed Sensing: Theory and Applications*, eds. Eldar, Y. C. and Kutyniok, G., Cambridge, UK: Cambridge University Press, pp. 1–64. 13, 15, 17

Elachi, C. and van Zyl, J. (2006), *Introduction to the Physics and Techniques of Remote Sensing, Second Edition*, Hoboken, NJ: John Wiley & Sons, Inc., 2nd ed., Available from `http://www.onlinelibrary.wiley.com`. 1

Graps, A. (1995), "An introduction to wavelets," *IEEE Computational Science and Engineering*, 2, 50–61. 11

Höllig, K. and Hörner, J. (2013), *Approximation and Modeling with B-splines*, Philadelphia, PA: Society for Industrial and Applied Mathematics. 12, 13

Lee, P. M. (2012), *Bayesian Statistics: An Introduction*, Chichester, UK: John Wiley & Sons Ltd., 4th ed., Available from `http://site.ebrary.com`. 20

Lustig, M., Donoho, D., and Pauly, J. M. (2007), "Sparse MRI: The Application of Compressed Sensing for Rapid MR Imaging," *Magnetic Resonance in Medicine*, 58, 1182–1195. 14

Mackenzie, D. (2009), "Compressed Sensing Makes Every Pixel Count," *Whats Happening in the Mathematical Sciences*, 7, 114–127. 9, 13, 14

Moon, T. K., Gunther, J. H., and Williams, G. (2015), "Extracting the fundamental frequency of a nonlinear chirp signal with modulated harmonic structure using ML, target tracking, and the Viterbi algorithm," *2015 IEEE Signal Processing and Signal Processing Education Workshop (SP/SPE)*, 347–352. 23, 24, 25

Najmi, A.-H. (2012), *Wavelets: A Concise Guide*, Baltimore, MD: Johns Hopkins University Press. 8, 9

NASA (2012), "Landsat Data Continuity Mission: Continuity in Global Land Observation," Available at `http://www.nasa.gov/pdf/716684main_LDCMFactSheet.pdf`. 1

Nelson, L. J., Ozoliņš, V., Reese, C. S., Zhou, F., and Hart, G. L. W. (2013), "Cluster expansion made easy with Bayesian compressive sensing," *Physical Review B*, 88. 16, 20, 25, 27, 29

Percival, D. B. and Walden, A. T. (2000), *Wavelet Methods for Time Series Analysis*, Cambridge, UK: Cambridge University Press. 9, 10, 61

Pereyra, M. C. and Ward, L. A. (2012), *Harmonic Analysis: From Fourier to Wavelets*, Providence, RI: American Mathematical Society. 6

Ramsay, J. O., Hooker, G., and Graves, S. (2009), *Functional Data Analysis with R and MATLAB*, New York, NY: Springer, Available from `http://link.springer.com`. 8, 12, 61

Shannon, C. E. (1949), "Communication in the Presence of Noise," *Proceedings of the IRE*, 37, 10–21. 13

Tipping, M. E. (2001), "Sparse Bayesian Learning and the Relevance Vector Machine," *Journal of Machine Learning Research*, 1, 211–244. 48

Tipping, M. E. and Faul, A. C. (2003), "Fast Marginal Likelihood Maximisation for Sparse Bayesian Models," in *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, eds. Bishop, C. M. and Frey, B. J., Key West, FL. 33

# Appendix A

## A.1 Useful matrix identities

### A.1.1 Woodbury matrix identity

$$(D + UEV)^{-1} = D^{-1} - D^{-1}U(E^{-1} + VD^{-1}U)^{-1}VD^{-1}. \tag{A.1}$$

where $D$ is $M \times M$, $U$ is $M \times N$, $E$ is $N \times N$, and $V$ is $N \times M$.

### A.1.2 Sylvester determinant identity

$$\det(I_M + AB) = \det(I_N + BA) \tag{A.2}$$

where $A$ is $M \times N$ and $B$ is $N \times M$. Note that this identity allows us to change the dimensions of the matrix within the determinant, since we go from a $M \times M$ matrix to a $N \times N$ matrix, or vice versa.

## A.2 Equivalency of $\ell_1$ minimization and finding the MAP

The Laplace distribution is given as (Babacan et al., 2010)

$$p(\mathbf{w}|\lambda) = \frac{\lambda}{2} \exp\left( - \frac{\lambda}{2} \|\mathbf{w}\|_1 \right). \tag{A.3}$$

Combining this prior distribution with the likelihood we get

$$(2\pi)^{-\frac{k}{2}} |\mathbf{\Sigma}|^{-\frac{1}{2}} \exp\left( - \frac{1}{2}(\mathbf{y} - \Phi\mathbf{w})'\mathbf{\Sigma}^{-1}(\mathbf{y} - \Phi\mathbf{w})\right) \frac{\lambda}{2} \exp\left( - \frac{\lambda}{2} \|\mathbf{w}\|_1 \right) \tag{A.4}$$

Since we are maximizing the posterior with respect to $\mathbf{w}$, we ignore the prior on $\sigma^2$ and any hyperprior on $\lambda$. Following this same reasoning, we can drop all constants with respect to $\mathbf{w}$ and combine terms to get

$$\exp\Bigg( - \frac{1}{2\sigma^2}(\mathbf{y} - \Phi\mathbf{w})'(\mathbf{y} - \Phi\mathbf{w}) - \frac{\lambda}{2}\left\|\mathbf{w}\right\|_1 \Bigg) \tag{A.5}$$

The $\frac{1}{\sigma^2}$ term appears from the main diagonal of the inverse covariance matrix $\mathbf{\Sigma}^{-1}$. Factoring out this term and $\frac{1}{2}$ gives us

$$\Bigg(\exp\Bigg( - (\mathbf{y} - \Phi\mathbf{w})'(\mathbf{y} - \Phi\mathbf{w}) - \sigma^2\lambda\left\|\mathbf{w}\right\|_1 \Bigg)\Bigg)^{\frac{1}{2\sigma^2}} \tag{A.6}$$

Taking the log of this and setting $\tau = \lambda\sigma^2$, gives us equivalent results as the $\ell_1$ minimization problem.

## A.3 Finding the distributions for $p(\mathbf{w}|\sigma^2, \gamma, \lambda, \mathbf{y})$ and $p(\mathbf{y}|\sigma^2, \gamma, \lambda)$

The approach used here is suggested by Tipping (2001). We first use Bayes rule on $p(\mathbf{w}|\sigma^2, \gamma, \lambda, \mathbf{y})$ to get:

$$p(\mathbf{w}|\sigma^2, \gamma, \lambda, \mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{w}, \sigma^2, \gamma, \lambda)p(\mathbf{w}|\sigma^2, \gamma, \lambda)}{p(\mathbf{y}|\sigma^2, \gamma, \lambda)}. \tag{A.7}$$

If we multiply the denominator in (A.7) to the other side and focus on just the right side of the equation, we have

$$\begin{aligned} p(\mathbf{y}|\mathbf{w}, \sigma^2, \gamma, \lambda)p(\mathbf{w}|\sigma^2, \gamma, \lambda) &= p(\mathbf{y}|\mathbf{w}, \sigma^2)p(\mathbf{w}|\gamma) \\ &= (2\pi)^{-\frac{M}{2}}|\sigma^2 I_M|^{-\frac{1}{2}}e^{-\frac{1}{2}(\mathbf{y}-\mathbf{\Phi w})'\frac{1}{\sigma^2}I_M(\mathbf{y}-\mathbf{\Phi w})}(2\pi)^{-\frac{N}{2}}|\Lambda^{-1}|^{-\frac{1}{2}}e^{-\frac{1}{2}\mathbf{w}'\mathbf{\Lambda w}}. \end{aligned}$$

If we then rearrange these terms to find a density in terms of $\mathbf{w}$ and a density in terms of $\mathbf{y}$, then we will have found our desired densities. To do this we first combine terms in the exponent to get

$$(2\pi)^{-\frac{M}{2}}|\sigma^2 I_M|^{-\frac{1}{2}}(2\pi)^{-\frac{N}{2}}|\Lambda^{-1}|^{-\frac{1}{2}}e^{-\frac{1}{2}(\frac{1}{\sigma^2}\mathbf{y}'\mathbf{y}-\frac{1}{\sigma^2}\mathbf{w}'\mathbf{\Phi}'\mathbf{y}-\frac{1}{\sigma^2}\mathbf{y}'\mathbf{\Phi w}+\frac{1}{\sigma^2}\mathbf{w}'\mathbf{\Phi}'\mathbf{\Phi w}+\mathbf{w}'\mathbf{\Lambda w})}.$$

Focusing on just the exponent, we can factor out $\mathbf{w}'$ and $\mathbf{w}$ from two of the terms

$$-\frac{1}{2}(\frac{1}{\sigma^2}\mathbf{y}'\mathbf{y} - \frac{1}{\sigma^2}\mathbf{w}'\mathbf{\Phi}'\mathbf{y} - \frac{1}{\sigma^2}\mathbf{y}'\mathbf{\Phi}\mathbf{w} + \mathbf{w}'(\frac{1}{\sigma^2}\mathbf{\Phi}'\mathbf{\Phi} + \mathbf{\Lambda})\mathbf{w}).$$

We then introduce the terms $\mu = \frac{\mathbf{1}}{\sigma^{\mathbf{2}}}\mathbf{\Sigma}\mathbf{\Phi}'\mathbf{y}$ and $\mathbf{\Sigma} = [\frac{\mathbf{1}}{\sigma^{\mathbf{2}}}\mathbf{\Phi}'\mathbf{\Phi} + \mathbf{\Lambda}]^{-\mathbf{1}}$

$$-\frac{1}{2}(\frac{1}{\sigma^2}\mathbf{y}'\mathbf{y} - \frac{1}{\sigma^2}\mathbf{w}'\mathbf{\Sigma}^{-1}\mathbf{\Sigma}\mathbf{\Phi}'\mathbf{y} - \frac{1}{\sigma^2}\mathbf{y}'\mathbf{\Phi}\mathbf{\Sigma}\mathbf{\Sigma}^{-1}\mathbf{w} + \mathbf{w}'\mathbf{\Sigma}^{-1}\mathbf{w}).$$

Rewriting the exponent and completing the square gives us

$$-\frac{1}{2}(\frac{1}{\sigma^2}\mathbf{y}'\mathbf{y} - \mathbf{w}'\mathbf{\Sigma}^{-1}\mu - \mu'\mathbf{\Sigma}^{-1}\mathbf{w} + \mathbf{w}'\mathbf{\Sigma}^{-1}\mathbf{w} + \mu'\mathbf{\Sigma}^{-1}\mu - \mu'\mathbf{\Sigma}^{-1}\mu).$$

Separating the exponent then gives us

$$(2\pi)^{-\frac{M}{2}}|\sigma^2 I_M|^{-\frac{1}{2}}e^{-\frac{1}{2}(\mathbf{w}-\mu)'\mathbf{\Sigma}^{-1}(\mathbf{w}-\mu)}(2\pi)^{-\frac{N}{2}}|\Lambda^{-1}|^{-\frac{1}{2}}e^{-\frac{1}{2}(\frac{1}{\sigma^2}\mathbf{y}'\mathbf{y}-\mu'\mathbf{\Sigma}^{-1}\mu)}. \tag{A.8}$$

The expression in (A.8) is then multiplied by $\frac{|\mathbf{\Sigma}|^{-\frac{1}{2}}}{|\mathbf{\Sigma}|^{-\frac{1}{2}}}$ in order to have the correct covariance matrix for one of the forms of the normal.

The next step requires the Woodbury identity referenced in A.1 applied to the term found in the second exponential. Using the notation found in A.1, we let $D = \sigma^2 I_M$, $U = \mathbf{\Phi}$, $E = \mathbf{\Lambda}^{-1}$, and $V = \mathbf{\Phi}'$, we can rewrite the term as

$$\begin{aligned}\frac{1}{\sigma^2}\mathbf{y}'\mathbf{y} - \mu'\mathbf{\Sigma}^{-1}\mu &= \mathbf{y}'\left[\frac{1}{\sigma^2}I_M - \frac{1}{\sigma^2}\mathbf{\Phi}\mathbf{\Sigma}\mathbf{\Phi}'\frac{1}{\sigma^2}\right]\mathbf{y} \\ &= \mathbf{y}'\left[\sigma^2 I_M + \mathbf{\Phi}\mathbf{\Lambda}^{-1}\mathbf{\Phi}'\right]^{-1}\mathbf{y}.\end{aligned} \tag{A.9}$$

Finally, the expression is now in the form of two normals, one normal distribution in terms of $\mathbf{w}$ and the other in terms of $\mathbf{y}$. If we let $C = \sigma^2 I_M + \mathbf{\Phi}\Lambda^{-1}\mathbf{\Phi}'$, then we have:

$$\mathcal{N}(\mu, \mathbf{\Sigma})\mathcal{N}(0, C) = (2\pi)^{-\frac{N}{2}}|\mathbf{\Sigma}|^{-\frac{1}{2}}e^{-\frac{1}{2}(\mathbf{w}-\mu)'\mathbf{\Sigma}^{-1}(\mathbf{w}-\mu)}(2\pi)^{-\frac{M}{2}}|C|^{-\frac{1}{2}}e^{-\frac{1}{2}\mathbf{y}'C^{-1}\mathbf{y}}. \tag{A.10}$$

One important simplification to understand when finding $\mathcal{N}(0, C)$ is how the expression $|\Lambda^{-1}||\mathbf{\Sigma}^{-1}||\sigma^2 I_M|$ equals $|C|$. If we combine the first two determinants we get:

$$\begin{aligned} |\Lambda^{-1}||\mathbf{\Sigma}^{-1}| &= |\Lambda^{-1}\mathbf{\Sigma}^{-1}| \\ &= \left|\Lambda^{-1}\left[\frac{1}{\sigma^2}\mathbf{\Phi}'\mathbf{\Phi} + \Lambda\right]\right| \\ &= \left|\frac{1}{\sigma^2}\Lambda^{-1}\mathbf{\Phi}'\mathbf{\Phi} + I_N\right| \end{aligned} \tag{A.11}$$

We then apply the Sylvester determinant identity referenced in A.1, letting $A = \mathbf{\Phi}$ and $B = \mathbf{\Lambda}^{-1}\mathbf{\Phi}'$ to get:

$$\left|\frac{1}{\sigma^2}\Lambda^{-1}\mathbf{\Phi}'\mathbf{\Phi} + I_N\right| = \left|\frac{1}{\sigma^2}\mathbf{\Phi}\Lambda^{-1}\mathbf{\Phi}' + I_M\right| \tag{A.12}$$

Multiplying $\sigma^2 I_M$ through finally gives us $|C|$.

# Appendix B

## B.1 Signal Creation

Note that the original code provided by Dr. Todd Moon is written in `Matlab`. This is an edited translation into `R`.

```
# Make a chirp signal
Fs <- 8000  # sampling frequency
Ts <- 1/Fs  # sample period
Tfinal <- 10  # number of seconds of data
tlist <- seq(0,Tfinal,by=Ts)  # time list
Nt <- length(tlist)
N_eta <- 15  # number of harmonics
alist <- rep(1,N_eta)  # signal amplitudes

# Parameters of the fundamental frequency
toff <- -1.5  # shift of tanh
tfineff <- 1  # final tanh time
Fmin <- 480  # limit initial frequency
Fmax <- 2100  # limit final frequency

# Create fundamental frequencey
tau <- (tfineff - toff)/Tfinal * tlist + toff  # time list
flist <- Fmin + (Fmax-Fmin) * (tanh(tau)+1)/2  # frequency list
phit <- 2*pi*cumsum(flist)*Ts  # phase (integral of freq.)

# Generate signal and all harmonics
s <- rep(0,Nt)

for (i in 1:N_eta){
  s <- s + alist[i]*cos(i*phit)
}
```

## B.2 Fast Laplace Algorithm

The code here is written in `C++` together with `Rcpp` in order to call the function from `R` while taking advantage of the speed of `C++`. The original algorithm is written in `Matlab` by Derin Babacan, one of the authors in Babacan et al. (2010). That version can be accessed at `http://www.dbabacan.info/software.html` as well as the copyright notice.

```
// [[Rcpp::depends(RcppGSL)]]
// [[Rcpp::depends(RcppArmadillo)]]
#include <RcppArmadillo.h>
#include <RcppGSL.h>
#include <Rmath.h>
#include <RcppArmadilloExtensions/sample.h>
#include <math.h>
#include <stdlib.h>
#include <iostream>
#include <iomanip>
using namespace Rcpp;

//' @useDynLib bcs
//' @importFrom Rcpp sourceCpp

// Finds the intersection between two vectors. Returns a matrix with two columns:
// the first column contains the elements in the intersection and the second
// column contains the indices of those elements in the second vector. Returns
// a sorted matrix, according to the first column.
// [[Rcpp::export]]
arma::umat intersect(arma::umat first, arma::umat second){
  int m = first.n_rows;
  int n = second.n_rows;
  int size;
  if(m<n){
    size = n;
  }
  else{
    size = m;
  }
  int count = 0;
  arma::umat inter;
  inter.ones(size,2);
  // Iterate through both vectors.
  for(int i=0; i<m; i++){
    for(int j=0; j<n; j++){
      if(first(i)==second(j)){
        inter(count,0) = first(i);
        inter(count,1) = j;
        count++;
        break;
      }
    }
  }
  if(count==0){ // If no elements in intersection return null matrix
    arma::umat null_matrix;
    return null_matrix;
  }
  else{
    inter = inter.rows(0,count-1);
    arma::umat s_ind = sort_index(inter.col(0));
    return inter.rows(s_ind);
  }
}

// Finds the set difference between two vectors. The second vector is assumed to
// be a subset of the first vector. Returns a column of the elements in the set
// difference.
// [[Rcpp::export]]
arma::umat setdiff(arma::umat first, arma::umat second){
  int m = first.n_rows;
  int n = second.n_rows;
  int size;
  if(m<n){
    size = n;
  }
  else{
    size = m;
  }
  arma::umat set_bool;
  int count = 0;
  arma::umat set_return;
  set_return.ones(size,1);
  for(int i=0; i<m; i++){
    set_bool = sum(any(second == first(i)),1);
    if(set_bool(0)==0){
      set_return(count) = first(i);
      count++;
    }
  }
  if(count==0){ // If no elements in set difference, return null matrix
    arma::umat null_answer;
    return null_answer;
  }
  else{
    return set_return.rows(0,count-1);
  }
}

//'   Implements the fast Laplace algorithm.
//'
//'   This code implements the fast Laplace algorithm from [1], which is based
//'   on the BCS code available from [2]. The fast Laplace algorithm is a method
//'   used to solve the compressive sensing problem, or in general, a highly
//'   underdetermined system of equations. This system can be written out as:
//'   \deqn{y = \Phiw + n}
//'   where \eqn{w} is the vector of unknown coefficients to solve for and
//'   \eqn{n} is random noise. The method uses a Bayesian framework, and in
//'   particular, uses a Laplace prior to incorporate the information that most
//'   of the coefficients in the solution vector are zero or close to zero.
//'
//' @param PHI measurement matrix.
//' @param y CS measurements.
//' @param sigma2 initial noise variance.
//' @param eta threshold in determining convergence of marginal likelihood.
//' @param roundit whether or not to round the marginal likelihood, in order to
//'       avoid machine precision error when comparing across platforms.
//' @param verbose print to screen which basis are added, re-estimated, or deleted.
//' @return A list containing the following elements:
//' \tabular{lll}{
//'   \code{weights} \tab \tab sparse weights\cr
//'   \code{used} \tab \tab the positions of sparse weights\cr
//'   \code{sigma2} \tab \tab re-estimated noise variance\cr
//'   \code{errbars} \tab \tab one standard deviation around the sparse weights\cr
//'   \code{alpha} \tab \tab sparse hyperparameters (1/gamma)
//' }
//' @references [1] S. D. Babacan, R. Molina and A. K. Katsaggelos, "Bayesian
//' Compressive Sensing Using Laplace Priors," in IEEE Transactions on Image
//' Processing, vol. 19, no. 1, pp. 53-63, Jan. 2010.
//' @references [2] S. Ji, Y. Xue, L. Carin, "Bayesian Compressive Sensing,"
//' IEEE Trans. Signal Processing, vol. 56, no. 6, June 2008.
//' @references [3] M. Tipping and A. Faul, "Fast marginal likelihood maximisation
//' for sparse Bayesian models," in Proc. 9th Int. Workshop Artificial Intelligence
//' and Statistics, C. M. Bishop and B. J. Frey, Eds., 2003.
//' @export
// [[Rcpp::export]]
List FastLaplace(arma::mat PHI, arma::vec y, double sigma2, double eta,
                 bool roundit = 0, bool verbose = 0){
  double M = PHI.n_rows;
  double N = PHI.n_cols;

  // Calculates initial alpha, from equation (26) in [3]
  // Note: alpha = 1/gamma where gamma is the parameter used in [1]
  arma::mat PHIy = PHI.t()*y;
  arma::mat PHI2(sum(square(PHI),0)); // Square of 2-Norm of all basis
  PHI2 = PHI2.t();
  arma::mat ratio(square(PHIy)/PHI2);
  arma::uword index;
  // Finds the best basis to start off with, see [3].
  double maxr = ratio.max(index);
  arma::mat alpha; alpha.zeros(N,1);
  alpha(0,0) = PHI2(index)/(maxr-sigma2);

  // Computes initial Sig.
  arma::mat phi = PHI.col(index);
  arma::mat phitphi = (phi.t()*phi);
  arma::mat Sig; Sig << 1/(alpha(0) + phitphi(0)/sigma2); //Equation 25 of [1]

  // Computes initial mu.
  arma::mat mu; mu.zeros(N,1);
  mu(0,0) = Sig(0)*PHIy(index)/sigma2; //Equation 24 of [1]

  // Computes initial S and Q for ALL basis vectors.
  arma::mat left = (PHI.t()*phi)/sigma2;
  arma::mat S = PHI2/sigma2-Sig(0)*square(left); //Equation 51 of [1]
  arma::mat Q = PHIy/sigma2-Sig(0)*PHIy(index)/sigma2*left; //Equation 52 of [1]

  // Indices that are deleted.
  arma::umat deleted; deleted.zeros(N,1);

  int max_it = 1000;
  // This will later be changed if the algorithm converges before reaching max_it.
  int final_count = max_it;
  // Initializes the Maximum Likelihood array. It will later be truncated to the
  // actual number of iterations.
  arma::rowvec ML(max_it);
  arma::umat indices; indices.zeros(N,1);
  indices(0,0) = index;

  arma::mat delta;
  arma::mat ki;
  arma::mat mui;
```

```
arma::mat Sigi;
arma::mat Alpha;
arma::mat round_ml;
// Vector of the possible basis indices.
arma::uvec range_mat = arma::linspace<arma::uvec>(0,N-1,N);
arma::umat range_m = arma::umat(range_mat);
// Keeps track of which index to add basis. Starts at 1 since we added above.
arma::uword add_count = 1;
// Keeps track of how many deleted basis there are in deleted vector
arma::uword delete_count = 0;
for(int count=0; count<max_it; count++){
  if(count % 50 == 0){ //check to make sure there aren't user interrupts
    Rcpp::checkUserInterrupt();
  }

  //***************** First calculate all of the alphas *********************
  // For the alphas that are infinity.
  arma::mat s = S;
  arma::mat q = Q;
  // For the alphas that are not infinity
  //Equation 53 in [1]
  s(indices.head_rows(add_count)) = alpha.head_rows(add_count)
    %S(indices.head_rows(add_count))
    /(alpha.head_rows(add_count)
    -S(indices.head_rows(add_count)));
  //Equation 54 in [1]
  q(indices.head_rows(add_count)) = alpha.head_rows(add_count)
    %Q(indices.head_rows(add_count))
    /(alpha.head_rows(add_count)
    -S(indices.head_rows(add_count)));

  //Equation 35 in [1]
  double lambda = 2*(add_count - 1)/sum(sum(1/alpha.head_rows(add_count)));

  arma::mat A = lambda + s - square(q);
  arma::mat B = 2*lambda*s + square(s);
  arma::mat C = lambda*square(s);

  arma::mat theta = square(q)-s;
  arma::mat discriminant = square(B) - 4*A%C;
  // Sometimes the discriminant can get values that are essentially 0 but are
  // technically negative, therefore outputting nan's in the sqrt expression
  // below in calculating nextAphas. The next line therefore forces these to
  // 0.
  discriminant.elem(find(discriminant<0)).zeros();
```

```
arma::mat nextAlphas = (-B - sqrt(discriminant) ) / (2*A);



// Chooses the next alpha that maximizes marginal likelihood.
double inf = 1.0/0.0;
arma::mat Ones; Ones.ones(N);
arma::mat ml = -inf*Ones;

// Finds the indices of the alphas that are not infinity.
arma::umat ig0 = find(theta>lambda);
arma::umat ire = ::intersect(ig0,indices.head_rows(add_count));
// These are the indices of where the values appear in the vector "indices"
// above.
arma::umat which = ire.col(1);
// Indices for re-estimation.
ire = ire.col(0);

// If there are coefficients to re-estimate.
if(ire.n_rows>0){
  Alpha = nextAlphas(ire);
  // We are subtracting off the marginal likelihood of already calc. alpha
  ml(ire) = pow(q(ire),2)/(Alpha + s(ire)) + log(Alpha/(Alpha + s(ire))) -
            lambda/Alpha - pow(q(ire),2)/(alpha(which) + s(ire)) -
            log(alpha(which)/(alpha(which) + s(ire))) + lambda/alpha(which);
}

// Indices for adding.
arma::umat iad = ::setdiff(ig0,ire);

if(iad.n_rows>0){
  Alpha = nextAlphas(iad);
  ml(iad) = log(Alpha/(Alpha + s(iad))) + pow(q(iad),2)/(Alpha + s(iad))
    - lambda/Alpha;
  which = ::intersect(deleted.head_rows(delete_count),iad);
  // Makes sure the deleted basis stay deleted
  if(which.n_rows>0){
    which = which.col(0);
    ml(which).fill(-inf);
  }
}

arma::umat is0 = ::setdiff(range_m,ig0);

// Indices for deleting.
arma::umat ide = ::intersect(is0,indices.head_rows(add_count));
```

```
if(ide.n_cols>0){
  which = ide.col(1);
  ide = ide.col(0);
  if(add_count==1){
    ml(ide).fill(-inf);
  }
  else{
    // TODO: Not sure why we don't just put that this equals -inf here
    ml(ide) = -pow(q(ide),2)/(alpha(which) + s(ide)) -
      log(alpha(which)/(alpha(which) + s(ide))) + lambda/alpha(which);
  }
}

// Finds the max of the marginal likelihood.
arma::uword idx;
// NOTE: When comparing this with the original matlab code, the elements of
// 'ml' are only accurate to within seven decimal places or so because of
// machine precision error. So the max of 'ml' can begin to be different.
if(roundit == TRUE){
  round_ml = round(ml*pow(10,7))/pow(10,7);
}
else{
  round_ml = ml;
}
ML(count) = round_ml.max(idx);

// Checks convergence.
if(count>1){
  if(std::abs(ML(count)-ML(count-1)) < std::abs(ML(count)-ML(0))*eta){
    final_count = count + 1;
    break;
  }
}

// Finds any basis that needs to be re-estimated.
arma::uvec w = find(indices.head_rows(add_count)==idx);
// The update formulas below can be found in the appendix of [3].
if(theta(idx) > lambda){
  if(w.n_elem>0){  // Re-estimates basis.
    Alpha = nextAlphas(idx);
    arma::mat Sigii = Sig(w,w);
    mui = mu(w);
    Sigi = Sig.cols(w);
    delta = Alpha-alpha(w);
    ki = delta/(1+Sigii*delta);
    mu.head_rows(add_count) -= ki(0)*mui(0)*Sigi;

    Sig -= ki(0)*Sigi*Sigi.t();
    arma::mat comm = PHI.t()*(phi*Sigi)/sigma2;
    S += ki(0)*square(comm);
    Q += ki(0)*mui(0)*comm;
    alpha(w) = Alpha;

    if(verbose==TRUE){
      Rcout << "Reestimate " << idx << "\n";
    }
  }
  else{  // Add basis.
    Alpha = nextAlphas(idx);
    arma::mat phii = PHI.col(idx);
    double Sigii = 1/(Alpha(0)+S(idx));
    mui = Sigii*Q(idx);
    arma::mat comm1 = Sig*(phi.t()*phii)/sigma2;
    arma::mat ei = phii-phi*comm1;
    arma::mat off = -Sigii*comm1;
    // get dimensions
    arma::uword offN = off.n_cols;
    arma::uword sigM = Sig.n_rows;
    arma::uword sigN = Sig.n_cols;
    arma::mat newSig;
    newSig.zeros((sigM+offN),(sigN+offN)); //TODO: Is off missing sigma2?
    newSig.submat(0,0,(sigM-1),(sigN-1)) = Sig+Sigii*comm1*comm1.t();
    newSig.submat(0,sigN,(sigM-1),(sigN+offN-1)) = off;
    newSig.submat(sigM,0,(sigM+offN-1),(sigN-1)) = off.t();
    newSig.submat((sigM+offN-1),(sigN+offN-1),(sigM+offN-1),(sigN+offN-1))
      = Sigii;
    Sig = newSig;
    mu.head_rows(add_count) -= mui(0)*comm1;
    mu(add_count,0) = mui(0);
    arma::mat comm2 = PHI.t()*ei/sigma2;
    S -= Sigii*pow(comm2,2);
    Q -= mui(0)*comm2;
    indices(add_count,0) = idx;
    alpha(add_count,0) = Alpha(0);
    phi = join_rows(phi,phii);
    add_count++;
    if(verbose==TRUE){
      Rcout << "Add " << idx << "\n";
    }
    }
  }
  else{
    if(w.n_elem > 0 & add_count > 1){ // Deletes basis.
      deleted(delete_count,0) = idx;
      delete_count++;
      arma::mat Sigii = Sig(w,w);
      mui = mu(w);
      Sigi = Sig.cols(w);
      Sig -= Sigi*Sigi.t()/Sigii(0);
      Sig.shed_col(w(0));
      Sig.shed_row(w(0));
      mu.head_rows(add_count) -= mui(0)*Sigi/Sigii(0);
      mu.shed_row(w(0));
      arma::mat comm = PHI.t()*(phi*Sigi)/sigma2;
      S += pow(comm,2)/Sigii(0);
      Q += mui(0)/Sigii(0)*comm;
      indices.shed_row(w(0));
      indices.insert_rows(N-1,1);
      alpha.shed_row(w(0));
      alpha.insert_rows(N-1,1);
      phi.shed_col(w(0));
      add_count -= 1;
      if(verbose==TRUE){
        Rcout << "Delete " << idx << "\n";
      }
    }
    else if(w.n_elem>0 & add_count==1){
      // Something is wrong, trying to delete the only coefficient that has
      // been added.
      break;
    }
  }
} // end of for loop

  arma::mat weights = mu.head_rows(add_count);
  //  Add 1 to put indices in terms of R syntax
  arma::umat used = indices.head_rows(add_count)+1;
  // Re-estimates sigma2
  arma::mat sigma2_re = sum(pow((y-phi*mu.head_rows(add_count)),2))/
  (M-add_count+alpha.head_rows(add_count).t()*Sig.diag());
  arma::mat errbars = sqrt(Sig.diag());

  if(verbose==TRUE){
    Rcout << "Algorithm converged, # iterations : " << final_count <<  "\n";
    }
  return List::create(Named("weights",weights),Named("used",used),
                      Named("sigma2",sigma2_re),Named("errbars",errbars),
                      Named("alpha",alpha.head_rows(add_count)));
}
```

## B.3 Simulations and Plots

Code for the simulations and plots in the results section. Note that the code for the sensing matrix $\mathbf{\Phi}'$ (`A` in the `CompareTrainSize` and `CompareNoise` functions), is translated to `R` from code written in `Matlab` by Derin Babacan, one of the authors in Babacan et al. (2010). That version can be accessed at `http://www.dbabacan.info/software.html` as well as the copyright notice.

The functions in Appendix B.2 and the `WaveletBasis`, `FourierBasis`, `BSplineBasis`, and `FindSparse` functions found here are the core functions of a `R` package tentatively named `bcs` developed in conjunction with this project. Note that the `WaveletBasis` function depends on the `wmtsa` package in `R` as associated with Percival and Walden (2000). See `https://cran.r-project.org/web/packages/wmtsa/wmtsa.pdf` for more information. The `FourierBasis` and `BSplineBasis` functions depend on the functional data analysis `R` package associated with Ramsay et al. (2009). See `https://cran.r-project.org/web/packages/fda/fda.pdf` for more information.

Note that before running these simulations, we first need to run the code from Appendix B.1 to create the signal `s` and time points `tlist`.

```
#' Finds the discrete wavelet transform matrix.
#'
#' Uses the functions \code{\link[wmtsa]{wavDWT}} and
#' \code{\link[wmtsa]{reconstruct}} from the \code{wmtsa} package to find the
#' transformation matrix of the given wavelet basis type. Each column of the
#' matrix is a basis function from the wavelet basis type, evaluated at specified
#' points along the rows of the matrix.
#'
#' @param N number of wavelet basis to keep.
#' @param train indices corresponding to which rows of the matrix to keep.
#'        Default is to keep all rows.
#' @param wavelet the type of wavelet basis to use. See
#'        \code{\link[wmtsa]{wavDaubechies}} from \code{wmtsa} for types.
#' @return A PxN discrete wavelet transform matrix, where P is equal to the
#'        length of \code{train} and N is the number of basis.
#' @export
WaveletBasis <- function(N, train = NULL, wavelet = "Haar"){
  signal <- 1:N
  P <- length(signal)
  # Default for training points is to use all points
  if (is.null(train)){
    train <- 1:P
  }
  M <- length(train)
  w <- wmtsa::wavDWT(signal, wavelet = wavelet)
  lvl <- w$dictionary$n.levels
  basis <- matrix(NA, M, N)
  compt <- 1
  w$data[lvl + 1][[1]] <- 0
  for (i in 1:lvl){
    w$data[[i]] <- rep(0, 2 ^ (lvl - i))
  }
  for (i in 1:(lvl + 1)){
    for (j in 1:2 ^ (lvl - i)){
      w$data[[i]][j] <- 1
      basis[, compt] <- wmtsa::reconstruct(w)[train]
      w$data[[i]][j] <- 0
      compt <- compt + 1
    }
  }
  return(basis)
}

#' Finds the discrete Fourier transformation matrix.
#'
#' Uses the package \code{\link{fda}} to find a transformation matrix where the
#' columns are the different Fourier basis, evaluated at specified
#' points along the rows of the matrix.
#'
#' @param tlist an array of the specific points where the basis are evaluated at.
#' @param N number of basis in the matrix.
#' @param train indices corresponding to which rows of the matrix to keep.
#'        Default is to keep all rows.
#' @return A PxN discrete Fourier transformation matrix where P is equal to the
#'        length of \code{train} and N is the number of basis.
#' @export
```

```
FourierBasis <- function(tlist, N, train = NULL){
  P <- length(tlist)
  # Default for training points is to use all points
  if (is.null(train)){
    train <- 1:P
  }
  # Forces fda to keep an even number of basis
  if (N %% 2 == 0)
    suppressWarnings(b.for <- fda::create.fourier.basis(c(tlist[1], tlist[P]),
                                                        N, dropind = N - 1))
  else
    b.for <- fda::create.fourier.basis(c(tlist[1], tlist[P]), N)
  return(fda::eval.basis(tlist[train], b.for))
}

#' Finds the transformation matrix for the B-spline basis.
#'
#' Uses the package \code{\link{fda}} to find a transformation matrix where each
#' column is a basis function from the B-spline basis, evaluated at specified
#' points along the rows of the matrix.
#'
#' @param tlist an array of the specific points where the basis are evaluated at.
#' @param N number of basis in the matrix.
#' @param train indices corresponding to which rows of the matrix to keep.
#'        Default is to keep all rows.
#' @return A PxN matrix where P is equal to the length of \code{train} and N is
#'        the number of B-spline basis.
#' @export
BSplineBasis <- function(tlist, N, train = NULL){
  P <- length(tlist)
  # Default for training points is to use all points
  if (is.null(train)){
    train <- 1:P
  }
  # The first two lines below shift the time scale so that the difference
  # between knots is roughly equal to 1. To see where the knots are placed take
  # bSpl$params
  tlist.diff <- tlist[2] - tlist[1]
  tlist <- tlist / tlist.diff
  b.spl <- fda::create.bspline.basis(c(tlist[1], tlist[P]), N)
  return(fda::eval.basis(tlist[train], b.spl))
}

#' Implements the fast Laplace algorithm.
#'
#' Given samples from some signal and a measurement matrix, find the sparse
#' respresentation of the signal using compressive sensing. Note that this
#' function is a convenient wrapper for the function \code{\link{FastLaplace}}.
#'
#' The fast Laplace algorithm is a method used to solve the compressive sensing
#' problem, or in general, a highly underdetermined system of equations. This
#' system can be written out as:
#'   \deqn{y = \Phiw + n}
#' where \eqn{w} is the vector of unknown coefficients to solve for and \eqn{n}
#' is random noise. The method uses a Bayesian framework, and in particular, uses
#' a Laplace prior to incorporate the information that most of the coefficients
#' in the solution vector are zero or close to zero. See [1] for details.
#'
#' @param PHI measurement matrix.
#' @param y sample from original signal.
#' @param eta tolerance level in determining convergence of marginal likelihood.
#' @param roundit whether or not to round the marginal likelihood, in order to
#'       avoid machine precision error when comparing across platforms.
#' @param verbose print to screen which basis are added, re-estimated, or deleted.
#' @return The sparse signal as found by the fast Laplace algorithm.
#' @references [1] S. D. Babacan, R. Molina and A. K. Katsaggelos, "Bayesian
#' Compressive Sensing Using Laplace Priors," in IEEE Transactions on Image
#' Processing, vol. 19, no. 1, pp. 53-63, Jan. 2010.
#' @export
FindSparse <- function(PHI, y, eta = 1e-8, roundit = FALSE, verbose=FALSE){
  M <- dim(PHI)[1]
  N <- dim(PHI)[2]
  fl <- FastLaplace(PHI, y, stats::sd(y) ^ 2 / M, eta, roundit = roundit,
                    verbose = verbose)
  x.lap <- rep(0, N)
  x.lap[fl[[2]]] <- fl[[1]]
  return(x.lap)
}

##########################################
#           Compare Train Size
##########################################
CompareTrainSize <- function(signal, tlist, iters, low_sample, high_sample,
by_sample, P, N){
    # Compare the RMSE between the basis across training set size
    x_scale <- seq(low_sample, high_sample, by = by_sample)
    gran <- length(x_scale)
    mseFor <- matrix(0, iters, gran)
    mseWav <- matrix(0, iters, gran)
    mseSpl <- matrix(0, iters, gran)
    l0For <- matrix(0,iters,gran)
    l0Wav <- matrix(0,iters,gran)
    l0Spl <- matrix(0,iters,gran)
    test <- 1:P
    for(j in seq(1,iters)){
        cat("Iters: ", j, "\n")
        counter <- 1
        for(M in x_scale){
            A <- matrix(runif(M*P),M,P)
            A <- A/matrix(rep(sqrt(apply(A^2,2,sum)),M),M,P,byrow=TRUE);

            # Wavelets
            bWav <- WaveletBasis(N, test)
            xWav <- FindSparse(A%*%bWav, A%*%signal[test], 1e-8, FALSE)
            mseWav[j,counter] <- sqrt(mean((bWav%*%xWav - signal[test])^2))
            l0Wav[j,counter] <- length(xWav[xWav!=0])

            # Fourier
            bFor <- FourierBasis(tlist[1:P], N, test)
            xFor <- FindSparse(A%*%bFor, A%*%signal[test], 1e-8, FALSE)
            mseFor[j,counter] <- sqrt(mean((bFor%*%xFor - signal[test])^2))
            l0For[j,counter] <- length(xFor[xFor!=0])

            # Spline
            bSpline <- BSplineBasis(tlist[1:P], N, test)
            xSpline <- FindSparse(A%*%bSpline, A%*%signal[test], 1e-8, FALSE)
            mseSpl[j,counter] <- sqrt(mean((bSpline%*%xSpline - signal[test])^2))
            l0Spl[j,counter] <- length(xSpline[xSpline!=0])

            counter <- counter + 1
        }
    }
    return(list(mseWav, mseFor, mseSpl, l0Wav, l0For, l0Spl))
}

mse <- CompareTrainSize(s, tlist, 50, 100, 1000, 100, 1024, 8192)
mseWav <- mse[[1]]
mseFor <- mse[[2]]
mseSpl <- mse[[3]]

# Plot of how RMSE changes with increased training set size
pdf(file="rmse.pdf",width=6,height=4)
par(mar=c(5,4,2,2))
plot(seq(100,1000,by=100),colMeans(mseWav),type='o',col='#FF9900',
ylim=c(0,4),xlab='',ylab="",lwd=3,pch=20)
title(ylab="RMSE", line=2.5, cex.lab=1)
title(xlab="Number of Samples", line=3, cex.lab=1)
```

```
legend("topright",legend=c("Wavelet","Fourier","B-spline"),lty=c(1,1,1),
lwd=c(3,3,3),col=c("#FF9900","#0099CC",'#99CC99'),pch=c(20,24,0),
text.font = 2,xjust=1,yjust=1,inset=.025,cex = .75)
par(new=TRUE)
plot(colMeans(mseFor),col="#0099CC",,type='o',ylim=c(0,4),xlab="",
xaxt='n',yaxt='n',ylab="",pch=24,lwd=3,cex=.7)
par(new=TRUE)
plot(colMeans(mseSpl),col='#99CC99',type='o',ylim=c(0,4),xlab="",
xaxt='n',yaxt='n',ylab="",pch=0,lwd=3,cex=.7)
dev.off()

##########################################
#           Compare Noise
##########################################
CompareNoise <- function(signal, tlist, iters, gran, P, M, N){
    l0Wav <- matrix(0,iters,gran)
    l0For <- matrix(0,iters,gran)
    l0Spl <- matrix(0,iters,gran)
    l1Wav <- matrix(0,iters,gran)
    l1For <- matrix(0,iters,gran)
    l1Spl <- matrix(0,iters,gran)
    l0Wav.white <- matrix(0,iters,gran)
    l0For.white <- matrix(0,iters,gran)
    l0Spl.white <- matrix(0,iters,gran)
    l1Wav.white <- matrix(0,iters,gran)
    l1For.white <- matrix(0,iters,gran)
    l1Spl.white <- matrix(0,iters,gran)
    noise.lvl <- seq(0,1,length.out=gran)
    test <- 1:P
    # Basis
    bWav <- WaveletBasis(N, test)
    bFor <- FourierBasis(tlist[1:P], N, test)
    bSpline <- BSplineBasis(tlist[1:P], N, test)
    for(j in 1:iters){
        cat(j,"\n")
        A <- matrix(runif(M*P),M,P)
        A <- A/matrix(rep(sqrt(apply(A^2,2,sum)),M),M,P,byrow=TRUE)
        count <- 1
        for(i in noise.lvl){
            ynoise <- A%*%signal[test] + rnorm(M,0,i*abs(signal[test]))
            ynoise.white <- A%*%signal[test] + rnorm(M,0,i*2)
            xWav <- FindSparse(A%*%bWav, ynoise, 1e-8)
            xFor <- FindSparse(A%*%bFor, ynoise, 1e-8)
            xSpline <- FindSparse(A%*%bSpline, ynoise, 1e-8)
            xWav.white <- FindSparse(A%*%bWav, ynoise.white, 1e-8)
            xFor.white <- FindSparse(A%*%bFor, ynoise.white, 1e-8)
            xSpline.white <- FindSparse(A%*%bSpline, ynoise.white, 1e-8)

            l0Wav[j,count] <- length(xWav[xWav!=0])
            l0For[j,count] <- length(xFor[xFor!=0])
            l0Spl[j,count] <- length(xSpline[xSpline!=0])
            l1Wav[j,count] <- sum(abs(xWav))
            l1For[j,count] <- sum(abs(xFor))
            l1Spl[j,count] <- sum(abs(xSpline))

            l0Wav.white[j,count] <- length(xWav.white[xWav.white!=0])
            l0For.white[j,count] <- length(xFor.white[xFor.white!=0])
            l0Spl.white[j,count] <- length(xSpline.white[xSpline.white!=0])
            l1Wav.white[j,count] <- sum(abs(xWav.white))
            l1For.white[j,count] <- sum(abs(xFor.white))
            l1Spl.white[j,count] <- sum(abs(xSpline.white))

            count <- count + 1
        }
    }
    return(list(l0Wav, l0For, l0Spl, l1Wav, l1For, l1Spl, l0Wav.white, l0For.white,
    l0Spl.white, l1Wav.white, l1For.white, l1Spl.white))
}

l100 <- CompareNoise(s, tlist, 50, 20, 1024, 100, 8192)
l0_Wav <- l100[[1]]
l0_For <- l100[[2]]
l0_Spl <- l100[[3]]
l1_Wav <- l100[[4]]
l1_For <- l100[[5]]
l1_Spl <- l100[[6]]
l0_Wav_white <- l100[[7]]
l0_For_white <- l100[[8]]
l0_Spl_white <- l100[[9]]
l1_Wav_white <- l100[[10]]
l1_For_white <- l100[[11]]
l1_Spl_white <- l100[[12]]

# Plot for L0-norm with noise proportional to the mean
pdf(file="l0_noise_mean.pdf",width=6,height=4)
par(mar=c(5,4,2,2))
plot(seq(0,1,length=20),colMeans(l0_Wav),type='o',
ylab='',xlab="",col="#FF9900",lwd=3,pch=20,ylim=c(60,130))
title(ylab="L0 Norm", line=2.5, cex.lab=1)
title(xlab="k", line=3, cex.lab=1)
```

```
legend("topright",legend=c("Wavelet","Fourier","B-spline"),lty=c(1,1,1),
lwd=c(3,3,3),col=c("#FF9900","#0099CC",'#99CC99'),pch=c(20,24,0),
text.font = 2,xjust=1,yjust=1,inset=.025,cex = .75)
par(new=TRUE)
plot(colMeans(l0_For),type='o',ylim=c(60,130),xlab="",xaxt="n",
ylab="",yaxt="n",col='#0099CC',lwd=3,pch=24,cex=.75)
par(new=TRUE)
plot(colMeans(l0_Spl),type='o',ylim=c(60,130),xlab="",xaxt="n",
ylab="",yaxt="n",col='#99CC99',lwd=3,pch=0,cex=.75)
dev.off()

# Plot for L0-norm with white gaussian noise
pdf(file="l0_white_noise.pdf",width=6,height=4)
par(mar=c(5,4,2,2))
plot(seq(0,2,length=20),colMeans(l0_Wav_white),type='o',
ylab='',xlab="",col="#FF9900",lwd=3,pch=20,ylim=c(60,130))
title(ylab="L0 Norm", line=2.5, cex.lab=1)
title(xlab=expression(sigma), line=3, cex.lab=1)
legend("topright",legend=c("Wavelet","Fourier","B-spline"),lty=c(1,1,1),
lwd=c(3,3,3),col=c("#FF9900","#0099CC",'#99CC99'),pch=c(20,24,0),
text.font = 2,xjust=1,yjust=1,inset=.025,cex = .75)
par(new=TRUE)
plot(colMeans(l0_For_white),type='o',ylim=c(60,130),xlab="",xaxt="n",
ylab="",yaxt="n",col='#0099CC',lwd=3,pch=24,cex=.75)
par(new=TRUE)
plot(colMeans(l0_Spl_white),type='o',ylim=c(60,130),xlab="",xaxt="n",
ylab="",yaxt="n",col='#99CC99',lwd=3,pch=0,cex=.75)
dev.off()

l400 <- CompareNoise(s, tlist, 50, 20, 1024, 400, 8192)
l0_Wav400 <- l400[[1]]
l0_For400 <- l400[[2]]
l0_Spl400 <- l400[[3]]
l1_Wav400 <- l400[[4]]
l1_For400 <- l400[[5]]
l1_Spl400 <- l400[[6]]
l0_Wav.white400 <- l400[[7]]
l0_For.white400 <- l400[[8]]
l0_Spl.white400 <- l400[[9]]
l1_Wav.white400 <- l400[[10]]
l1_For.white400 <- l400[[11]]
l1_Spl.white400 <- l400[[12]]

# Plot for L0-norm with noise proportional to the mean
pdf(file="l0_noise_mean400.pdf",width=6,height=4)
```

```
par(mar=c(5,4,2,2))
plot(seq(0,1,length=20),colMeans(l0_Wav400),type='o',
ylab='',xlab="",col="#FF9900",lwd=3,pch=20,ylim=c(250,350))
title(ylab="L0 Norm", line=2.5, cex.lab=1)
title(xlab="k", line=3, cex.lab=1)
legend("topleft",legend=c("Wavelet","Fourier","B-spline"),lty=c(1,1,1),
lwd=c(3,3,3),col=c("#FF9900","#0099CC",'#99CC99'),pch=c(20,24,0),
text.font = 2,xjust=1,yjust=1,inset=.025,cex = .75)
par(new=TRUE)
plot(colMeans(l0_For400),type='o',ylim=c(250,350),xlab="",xaxt="n",
ylab="",yaxt="n",col='#0099CC',lwd=3,pch=24,cex=.75)
par(new=TRUE)
plot(colMeans(l0_Spl400),type='o',ylim=c(250,350),xlab="",xaxt="n",
ylab="",yaxt="n",col='#99CC99',lwd=3,pch=0,cex=.75)
dev.off()

# Plot for L0-norm with white gaussian noise
pdf(file="l0_white_noise400.pdf",width=6,height=4)
par(mar=c(5,4,2,2))
plot(seq(0,2,length=20),colMeans(l0_Wav.white400),type='o',
ylab='',xlab="",col="#FF9900",lwd=3,pch=20,ylim=c(250,350))
title(ylab="L0 Norm", line=2.5, cex.lab=1)
title(xlab=expression(sigma), line=3, cex.lab=1)
legend("topright",legend=c("Wavelet","Fourier","B-spline"),lty=c(1,1,1),
lwd=c(3,3,3),col=c("#FF9900","#0099CC",'#99CC99'),pch=c(20,24,0),
text.font = 2,xjust=1,yjust=1,inset=.025,cex = .75)
par(new=TRUE)
plot(colMeans(l0_For.white400),type='o',ylim=c(250,350),xlab="",xaxt="n",
ylab="",yaxt="n",col='#0099CC',lwd=3,pch=24,cex=.75)
par(new=TRUE)
plot(colMeans(l0_Spl.white400),type='o',ylim=c(250,350),xlab="",xaxt="n",
ylab="",yaxt="n",col='#99CC99',lwd=3,pch=0,cex=.75)
dev.off()


# Plot for L1-norm with noise proportional to the mean
pdf(file="l1_noise_mean400.pdf",width=6,height=4)
par(mar=c(5,4,2,2))
plot(seq(0,1,length=20),colMeans(l1_Wav400),type='o',
ylab='',xlab="",col="#FF9900",lwd=3,pch=20,ylim=c(0,7000))
title(ylab="L1 Norm", line=2.5, cex.lab=1)
title(xlab="k", line=3, cex.lab=1)
legend("topright",legend=c("Wavelet","Fourier","B-spline"),lty=c(1,1,1),
lwd=c(3,3,3),col=c("#FF9900","#0099CC",'#99CC99'),pch=c(20,24,0),
text.font = 2,xjust=1,yjust=1,inset=.025,cex = .75)
```

```
par(new=TRUE)
plot(colMeans(l1_For400),type='o',ylim=c(0,7000),xlab="",xaxt="n",
ylab="",yaxt="n",col='#0099CC',lwd=3,pch=24,cex=.75)
par(new=TRUE)
plot(colMeans(l1_Spl400),type='o',ylim=c(0,7000),xlab="",xaxt="n",
ylab="",yaxt="n",col='#99CC99',lwd=3,pch=0,cex=.75)
dev.off()

# Plot for L1-norm with white noise
pdf(file="l1_white_noise400.pdf",width=6,height=4)
par(mar=c(5,4,2,2))
plot(seq(0,1,length=20),colMeans(l1_Wav.white400),type='o',
ylab='',xlab="",col="#FF9900",lwd=3,pch=20,ylim=c(0,7000))
title(ylab="L1 Norm", line=2.5, cex.lab=1)
title(xlab=expression(sigma), line=3, cex.lab=1)
legend("topright",legend=c("Wavelet","Fourier","B-spline"),lty=c(1,1,1),
lwd=c(3,3,3),col=c("#FF9900","#0099CC",'#99CC99'),pch=c(20,24,0),
text.font = 2,xjust=1,yjust=1,inset=.025,cex = .75)
par(new=TRUE)
plot(colMeans(l1_For.white400),type='o',ylim=c(0,7000),xlab="",xaxt="n",
ylab="",yaxt="n",col='#0099CC',lwd=3,pch=24,cex=.75)
par(new=TRUE)
plot(colMeans(l1_Spl.white400),type='o',ylim=c(0,7000),xlab="",xaxt="n",
ylab="",yaxt="n",col='#99CC99',lwd=3,pch=0,cex=.75)
dev.off()
```