# Adaptive Transmit Beamforming for Simultaneous Transmit and Receive



by

**Daniel L. Gerber**

B.S., Massachusetts Institute of Technology (2010)

Submitted to the
Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

**Master of Engineering in Electrical Engineering and Computer Science**

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2011



Author ..............................................................
Department of Electrical Engineering and Computer Science
April 28, 2011

Certified by ..............................................................
Paul D. Fiore
Staff Member, MIT Lincoln Laboratory
Thesis Supervisor

Certified by ..............................................................
David H. Staelin
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by ..............................................................
Dr. Christopher J. Terman
Chairman, Department Committee on Graduate Theses

## Adaptive Transmit Beamforming for Simultaneous Transmit and Receive

by

**Daniel L. Gerber**

Submitted to the Department of Electrical Engineering and Computer Science
on April 28, 2011, in partial fulfillment of the
requirements for the degree of
**Master of Engineering in Electrical Engineering and Computer Science**

## Abstract

Simultaneous transmit and receive (STAR) is an important problem in the field of communications. Engineers have researched many different models and strategies that attempt to solve this problem. One such strategy is to place a transmit-side null at the receiver in order to decouple a system's transmitter and receiver, thus allowing them to operate simultaneously. This thesis discusses the use of gradient based adaptive algorithms to allow for transmit beamforming. Several such algorithms are devised, simulated, and compared in performance. Of these, the best is chosen to study in further detail. A mathematical analysis is performed on this particular algorithm to determine a linearized state space model, which is then used in a noise analysis. An important benefit of these algorithms is that they do not require computationally intensive matrix operations such as inversion or eigen-decomposition. This thesis ultimately provides, explains, and analyzes a viable method that can form a transmit-side null at the receiver and extract a weak signal of interest from the received signal while simultaneously transmitting another signal at high power.

Thesis Supervisor: Paul D. Fiore
Title: Staff Member, MIT Lincoln Laboratory

Thesis Supervisor: David H. Staelin
Title: Professor of Electrical Engineering and Computer Science

# Acknowledgements

First and foremost, I would like to thank my direct supervisor Paul Fiore. This thesis would not have been possible without the constant support Paul gave me through meetings, mathematical discussion, and ideas. I would also like to thank Jeff Herd for his support in organizing and leading the members working on the STAR project at Lincoln Laboratory. Thanks also to Dan Bliss who put me up to date with current research in the field of STAR. In addition, I would like to thank my MIT thesis adviser David Staelin and MIT academic adviser David Perreault for supporting me in my master's degree. Finally, I would like to thank my family and friends, who have always supported me and fueled my ambitions.

# Contents

# List of Figures

# Chapter 1

# Introduction

The simultaneous transmission and reception (STAR) of signals on the same frequency band is an important engineering challenge that still has not been adequately solved. Past attempts have included isolation methods between the transmitter and receiver [1, p. 16]. However, such methods offer minimal improvement in cancellation of the high power transmit signal at the receiver. Antenna isolation can provide a 30dB drop in transmit signal power at the receiver. An additional 30dB drop can result if the transmitter and receiver are cross-polarized with respect to one another [1, p. 18]

Another technique under investigation is that of active STAR. Active STAR involves the use of feed-forward electronics and algorithms to use the transmit signal as an input to the receive control loop. The transmit signal is filtered before it is subtracted from the received signal. As a final result, the transmit signal is cancelled from the received signal and the receiver can now detect other external signals of interest (SOI). In addition, the transmitter will not saturate or damage the analog-to-digital converters (ADC) on the receiver.

Active cancellation greatly improves the system performance at a level comparable to passive techniques [1, p. 20]. In addition, the techniques for active and passive cancellation all complement each other in reducing the amount of transmitted signal at the receiver. For example, an RF canceller can lower the transmit signals' power by up to 35dB [2, p. 8]. With another 60dB of well engineered antenna isolation and cross-polarization, the transmit power at the receiver will be 95dB lower.

One cancellation technique that can be applied in any sort of STAR or full duplex design is the use of multiple input multiple output (MIMO) antenna systems. The MIMO antenna array is very important for interference cancellation and isolation. Single input single output (SISO) antenna isolation techniques include polarizing the transmit and receive antennae in different planes, pointing them in different directions, distancing them, or shielding them from each other. The MIMO system can utilize these SISO cancellation techniques, but can also be set up such that the transmit beampattern places a null at the receive antenna location. In general, null points exist in systems with multiple transmitters because the transmitted signals have regions of destructive interference.

An early proposal [3] discusses interference cancellation for use in STAR. That paper proposed an interference cancellation LMS filter (treated here in Section 2.6) to cancel the transmitted signal from the receiver, after which experiments are performed on such a system. Previous work in wideband interference cancellation was performed in [4], in which several methods were tested in Matlab to compare performance and computational complexity. Experiments were performed in [1] to investigate various strategies of antenna isolation and interference cancellation for use in SISO STAR. These prior experiments differ from the setup in this thesis, wherein MIMO STAR is used to overcome the challenges of receiver saturation and a time variant channel. A similar idea [5] suggests augmenting the conventional SISO time domain cancellation techniques with MIMO spatial domain techniques. The techniques of zero-forcing and MMSE filtering were investigated and simulated. This thesis does not pursue these techniques in order to avoid the computationally intensive matrix inversion (or pseudo-inverse). In addition, the methods of this thesis deal with unknown time-variant channel models.

The work of Bliss, Parker, and Margetts [6] is the most relevant to this thesis. Those authors present the general problem of MIMO STAR communication, but focus specifically on the problems of simultaneous links and a full duplex relay. The analysis in their paper assumes a known time-variant channel that can be modeled as a matrix of channel values at certain delays. With knowledge of the signal and received data, the maximum likelihood channel estimate can be derived using the ordinary least squares method. The paper follows by proving that this method maximizes the signal to noise ratio. In addition, it demonstrates the method's performance through simulations of the simultaneous links and the full duplex relay problems. However, like [5], [6] requires matrix inversion. This thesis project is aimed at developing a robust method that does not require computationally intense calculations to solve for optimal weights. In addition, this thesis accounts for a prior lack of channel data and presents methods to probe the channels without saturating the transmitter.

This thesis will explore the digital portion of the STAR system design, with an emphasis on algorithms that make full duplex multipath communication possible in a changing environment. This STAR system's digital portion revolves around a digital MIMO LMS filter, as shown in Figure 1-1. The MIMO LMS filter will help to protect the ADCs from saturation or noisy signals. In addition, it will be able to filter out the transmitted signal from the received signal after the received signal has passed into the digital domain. The LMS filter will be the first step in the system's digital signal processing [2, p. 9].

The remainder of this thesis is organized as follows. A brief summary of the required background theory is given in Chapter 2. In addition, this chapter will discuss the current research and progress in the field of transmit beamforming. Chapter 3 will describe the methods of STAR that have been proposed and tested. For each method, a brief description of the algorithm and mathematical analysis will be provided and the method results will be discussed. The best method from Chapter 3 will be chosen and analyzed in Chapter 4. Chapter 4 will provide an in-depth mathematical analysis, linearization, and system model for the chosen method. Concluding thoughts are offered in Chapter 5.

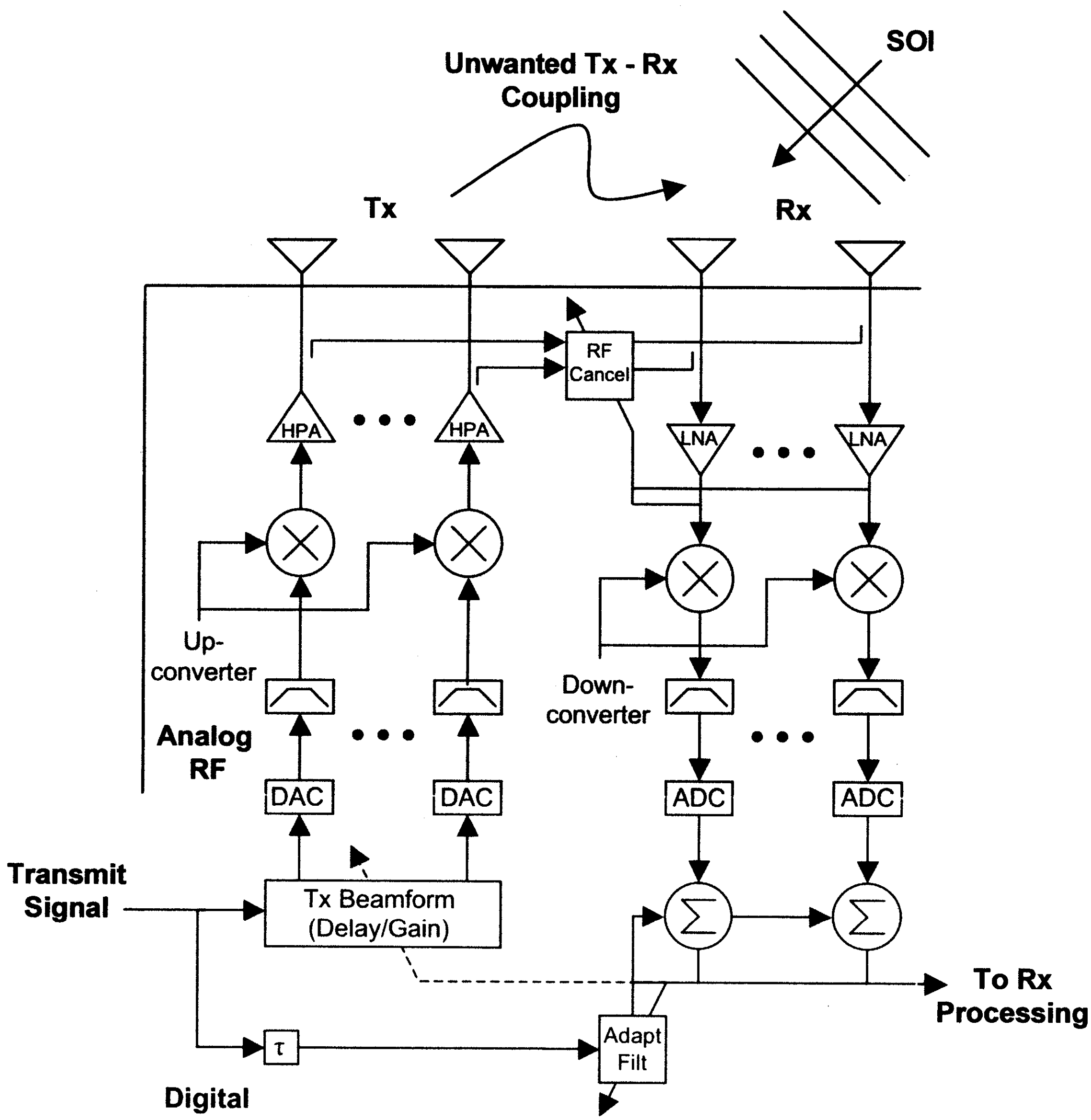


Figure 1-1: Block diagram for the MIT Lincoln Laboratory STAR system proposal [2]. Although this diagram shows plans for the antennae, analog electronics, and RF canceller setup, the focus of this thesis is on digital portion of the system that deals with transmit beamforming. HPA stands for high power amplifier, LNA stands for low noise amplifier, DAC stands for digital to analog converter, and ADC stands for analog to digital converter.

# Chapter 2

# Background Theory

This thesis presents methods for solving the STAR problem with adaptive transmit beamforming. Knowledge of adaptive beamforming is required in order understand the mathematical nature of the algorithms presented in Chapters 3 and 4. This chapter presents the fundamental background theory of adaptive beamforming and contains equations that will be important to the analysis in later chapters.

## 2.1 Control Theory

A system response can be determined by the behavior of its state variables, which describe some type of energy storage within the system. For example, the voltage on a capacitor is a state space variable in a circuit system. For systems with multiple states, the entire set of state variables can be expressed as a vector. Systems with multiple inputs can likewise have their inputs expressed as a vector. The state evolution equations of a linear time-invariant (LTI) state space system with multiple inputs can be written as [7, p. 284]

$$\mathbf{w}[n+1] = \mathbf{A}\mathbf{w}[n] + \mathbf{B}\mathbf{r}[n] \tag{2.1}$$

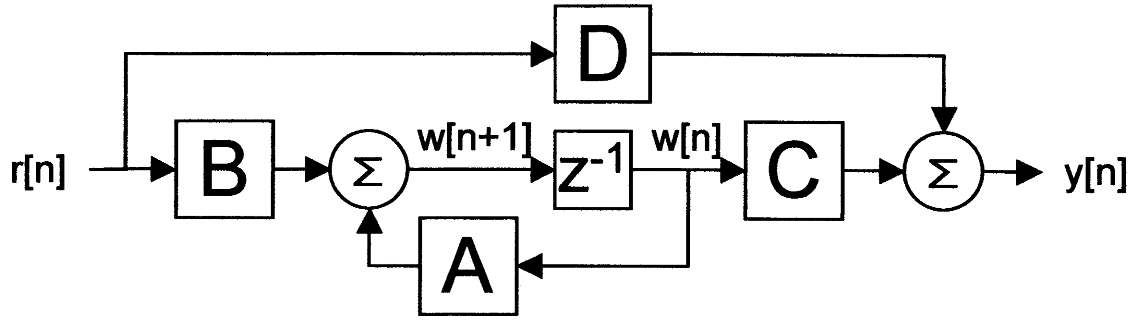


Figure 2-1: Block diagram of a state space system, with input vector $\mathbf{r}[n]$, state vector $\mathbf{w}[n]$, and output vector $\mathbf{y}[n]$.

where $\mathbf{w}[n]$ is a vector containing the state variable at sample time $n$, $\mathbf{r}[n]$ is the vector of inputs, and $\mathbf{A}$ and $\mathbf{B}$ are coefficient matrices of the state equation.

State space systems can have multiple outputs with a similar vector notation of the form

$$\mathbf{y}[n] = \mathbf{C}\mathbf{w}[n] + \mathbf{D}\mathbf{r}[n] \tag{2.2}$$

where $\mathbf{y}[n]$ is the output vector. Together, (2.1) and (2.2) form the state space equations of the system, and can be represented by the block diagram in Figure 2-1.

In the $z$-domain, these equations can be represented as [8, p. 768]

$$\begin{aligned} z\mathbf{w}(z) &= \mathbf{A}\mathbf{w}(z) + \mathbf{B}\mathbf{r}(z) \\ \mathbf{y}(z) &= \mathbf{C}\mathbf{w}(z) + \mathbf{D}\mathbf{r}(z)\,. \end{aligned} \tag{2.3}$$

The transfer function from input to state vector is defined as

$$\frac{\mathbf{w}(z)}{\mathbf{r}(z)} \triangleq (z\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} \tag{2.4}$$

and the transfer function from input to output is defined as

$$\frac{\mathbf{y}(z)}{\mathbf{r}(z)} \triangleq \mathbf{C}(z\mathbf{I}-\mathbf{A})^{-1}\mathbf{B}+\mathbf{D}\,. \tag{2.5}$$

We shall define the Resolvent Matrix $\mathbf{\Phi}(z)$ as

$$\mathbf{\Phi}(z) = (z\mathbf{I}-\mathbf{A})^{-1} \tag{2.6}$$

and note that in the time domain, the State Transition Matrix $\mathbf{\Phi}[n]$ is [7, p. 291]

$$\mathbf{\Phi}[n] = \mathbf{A}^{n} \tag{2.7}$$

for $n \geq 0$. With this information, the state vector becomes [7, p. 289]

$$\mathbf{w}[n] = \mathbf{A}^{n}\mathbf{w}[0] + \sum_{m=0}^{n} \mathbf{A}^{n-m}\mathbf{B}\mathbf{r}[m] \tag{2.8}$$

and the system output function is

$$\mathbf{y}[n] = \mathbf{C}\mathbf{A}^{n}\mathbf{w}[0] + \mathbf{C}\sum_{m=0}^{t} \mathbf{A}^{n-m}\mathbf{B}\mathbf{r}[m] + \mathbf{D}\mathbf{r}[n]\,. \tag{2.9}$$

Note that the second term of (2.8) is a convolution sum.

To determine the steady state behavior of the state vector $\mathbf{w}[n]$, $\mathbf{A}^n$ can be decomposed into [9, p. 294]

$$\mathbf{A}^{n} = \mathbf{V}_{(\mathbf{A})}^{-1}\,\mathbf{\Lambda}_{(\mathbf{A})}^{n}\,\mathbf{V}_{(\mathbf{A})} \tag{2.10}$$

where $\mathbf{V}_{(\mathbf{A})}$ is the right eigenvector matrix of $\mathbf{A}$ and $\mathbf{\Lambda}_{(\mathbf{A})}$ is the eigenvalue matrix. Since $\mathbf{\Lambda}_{(\mathbf{A})}$ is a diagonal matrix, $\mathbf{w}[n]$ will converge only if $|\lambda_{max,(\mathbf{A})}| < 1$, where $\lambda_{max,(\mathbf{A})}$ is the maximum eigenvalue of $\mathbf{A}$.

## 2.2 Gradient Descent and Numerical Methods

Optimization problems involve tuning a set of variables so that some cost function is minimized or maximized [10, p. 16]. This function is referred to as the "performance surface" and describes the system performance with respect to a set of system variables. In many cases, the performance surface is quadratic, and the local critical point is the global minimum or maximum [11, p. 21]. Critical points can be found by determining where the gradient of the performance surface is equal to zero.

Even though an optimal solution to such problems can be mathematically determined, numerical methods are often useful in practice because they are robust [10, p. 277]. In this sense, numerical methods are less likely to be affected by practical difficulties such as modeling errors or poor characterization data.

One particular method of searching the performance surface for local minima is the gradient descent method. This method starts at some point along the performance surface, finds the direction of the surface gradient, and steps in the direction of the (negative) gradient by a small amount [10, p. 277]. In vector form, the gradient of some function $f$ of a vector $\mathbf{w}$ is

$$\nabla f(\mathbf{w}) = \frac{\partial f}{\partial \mathbf{w}} = \begin{bmatrix} \frac{\partial f}{\partial w_1} \\ \vdots \\ \frac{\partial f}{\partial w_K} \end{bmatrix}. \tag{2.11}$$

Many numerical methods employ the negative gradient, which often points in the general direction of a local minimum. By stepping $\mathbf{w}$ in this direction every numerical cycle, the gradient descent method ensures that a local minimum will eventually be reached if the step size is sufficiently small. Any numerical method that uses gradient descent will contain an update equation of the form [11, p. 57]

$$\mathbf{w}[n+1] = \mathbf{w}[n] - \mu \nabla f\big|_{\mathbf{w}[n]} \tag{2.12}$$

where $\mu$ is the growth factor. The growth factor determines the step size, which governs the speed and accuracy of the numerical method.

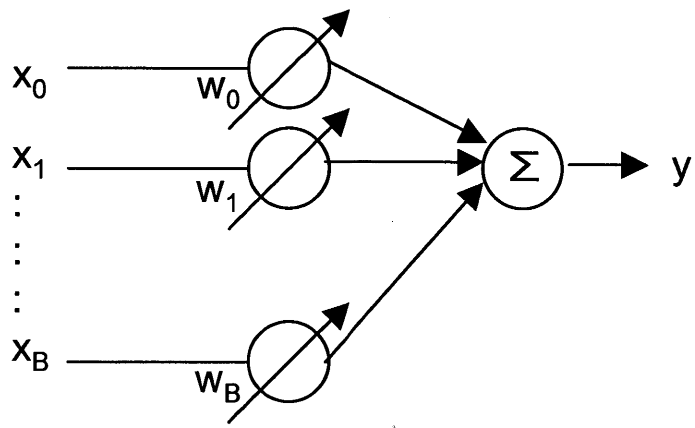


Figure 2-2: Adaptive linear combiner with input $\mathbf{x}$, weights $\mathbf{w}$, and output y.

## 2.3 Adaptive Linear Combiner

One particular use for gradient descent is in optimizing an adaptive linear combiner, shown in Figure 2-2. Here, the purpose of gradient descent is to minimize the mean squared error (MSE) $E[\varepsilon^2[n]]$ between a measurement $r[n]$, and an estimate $y[n]$ of that measurement. Using a gradient descent algorithm, the system error can be fed back into the system input $\mathbf{x}[n]$ through the adjustable weight values $w_k$. The rest of this section follows the derivation from [11, p. 19-22]. In the adaptive linear combiner,

$$\begin{aligned} \varepsilon[n] &= r[n] - y[n] \\ &= r - \mathbf{x}^T\mathbf{w} \end{aligned} \tag{2.13}$$

$$\begin{aligned} \varepsilon^2[n] &= r^2[n] - 2r[n]\mathbf{x}^T[n]\mathbf{w} + \mathbf{x}^T[n]\mathbf{w}\mathbf{x}^T[n]\mathbf{w} \\ &= r^2[n] - 2r[n]\mathbf{x}^T[n]\mathbf{w} + \mathbf{w}^T\mathbf{x}[n]\mathbf{x}^T[n]\mathbf{w}\ . \end{aligned} \tag{2.14}$$

Because $E[\varepsilon^2]$ is being minimized over $\mathbf{w}$, $\mathbf{w}$ is treated like a constant within the expectation operator. In other words,

$$\begin{aligned} E[\varepsilon^2] &= E[r^2] - 2E[r\mathbf{x}^T]\mathbf{w} + \mathbf{w}^T E[\mathbf{x}\mathbf{x}^T]\mathbf{w} \\ &= E[r^2] - 2\mathbf{p}^T\mathbf{w} + \mathbf{w}^T\mathbf{R}\mathbf{w} \\ &= E[r^2] - 2\mathbf{w}^T\mathbf{p} + \mathbf{w}^T\mathbf{R}\mathbf{w} \end{aligned} \tag{2.15}$$

where $\mathbf{R} = E[\mathbf{x}\mathbf{x}^T]$ is the input autocorrelation matrix and $\mathbf{p} = E[r\mathbf{x}^T]$ is the input-to-output crosscorrelation vector.

The gradient of the mean squared error is

$$\nabla E[\varepsilon^2] = \frac{\partial}{\partial \mathbf{w}} E[\varepsilon^2] = -2\mathbf{p} + 2\mathbf{R}\mathbf{w} \ . \tag{2.16}$$

This is the negative of the direction in which a gradient descent algorithm will step on any given cycle of the algorithm. In order to reach the optimal weight value, the algorithm will have to update the gradient every cycle and step in the gradient's direction. Most quadratic performance surfaces such as the MSE have a single global minimum. One exception is the function $\mathbf{x}^T\mathbf{R}\mathbf{x}$ in the case that $\mathbf{R}$ is low-rank. If we assume that $\mathbf{R}$ is of full rank, the optimal weight vector $\mathbf{w}_{opt}$ can be found by determining where the gradient is zero. In this case,

$$\begin{aligned} \mathbf{0} &= -2\mathbf{p} + 2\mathbf{R}\mathbf{w}_{opt} \\ \mathbf{w}_{opt} &= \mathbf{R}^{-1}\mathbf{p} \ . \end{aligned} \tag{2.17}$$

The optimal weight vector, also known as the Weiner solution, is the set of weights that positions the system at the bottom of the performance surface and thus minimizes the MSE. In addition, (2.17) is a variation of the Yule-Walker equations [12, p. 410].

## 2.4 Least Mean Squares Algorithm

The least mean squares (LMS) algorithm is a numerical method that uses gradient descent to minimize the MSE of an adaptive linear combiner. The LMS algorithm uses $\varepsilon^2[n]$ as an unbiased estimate of $E[\varepsilon^2]$ [11, p. 100]. It follows that

$$\begin{aligned} E[\varepsilon^2] &\approx \varepsilon^2[n] = r^2[n] - 2r[n]\mathbf{x}^T[n]\mathbf{w} + \mathbf{x}^T[n]\mathbf{w}\mathbf{x}^T[n]\mathbf{w} && (2.18) \\ \nabla[n] &= -2r[n]\mathbf{x}[n] + 2\mathbf{x}[n]\mathbf{x}^T[n]\mathbf{w} \\ &= -2\mathbf{p}[n] + 2\mathbf{R}[n]\mathbf{w} && (2.19) \end{aligned}$$

where $\nabla[n]$ is the gradient vector. Note that $\mathbf{R}[n]$ and $\mathbf{p}[n]$ differ from $\mathbf{R}$ and $\mathbf{p}$ from Section 2.3 because $\mathbf{R}[n]$ and $\mathbf{p}[n]$ do not use expected values. One particular consequence of this difference is that $\mathbf{R}[n]$ is rank-one because any outer product of the form $\mathbf{x}\mathbf{x}^T$ is rank-one [13, p. 461]. This also causes $\mathbf{R}[n]$ to have only one nonzero eigenvalue.

The LMS algorithm uses the gradient descent weight update equation from (2.12). In this case,

$$\begin{aligned} \mathbf{w}[n+1] &= \mathbf{w}[n] - \mu\nabla[n] \\ &= \mathbf{w}[n] + \mu(2\mathbf{p}[n] - 2\mathbf{R}[n]\mathbf{w}) \\ &= (\mathbf{I} - 2\mu\mathbf{R}[n])\mathbf{w}[n] + 2\mu\mathbf{p}[n]\ . && (2.20) \end{aligned}$$

As shown above, the LMS weight update equation takes on the state space form of (2.1) and (2.2). If the substitutions

$$\begin{aligned} \mathbf{A} &= \mathbf{I} - 2\mu\mathbf{R}[n] \\ \mathbf{B} &= 2\mu\mathbf{x}^T[n] \\ \mathbf{C} &= \mathbf{x}^T[n] && (2.21) \end{aligned}$$

are made, state space analysis may be used to determine the time response and convergence of the LMS algorithm [14]. The steady state convergence and behavior can therefore be analyzed using (2.8) and (2.10). From (2.8), it follows that

$$\mathbf{w}[n] = (\mathbf{I} - 2\mu\mathbf{R}[n])^n\mathbf{w}[0] + \sum_{m=0}^{n}(\mathbf{I} - 2\mu\mathbf{R}[n])^{n-m}(2\mu\mathbf{x}^T[n])\mathbf{r}[m] \,. \tag{2.22}$$

Statistical analysis must be used to determine the convergence of the LMS algorithm, since $\mathbf{R}[n]$ only has one nonzero eigenvalue. Therefore, we shall use the matrix $\mathbf{R} = E[\mathbf{x}\mathbf{x}^T]$ from Section 2.3 to study the algorithm convergence. Using (2.21), we have

$$\begin{aligned}(\mathbf{I} - 2\mu\mathbf{R})^n &= \mathbf{V}_{(\mathbf{R})}^{-1}(\mathbf{I} - 2\mu\mathbf{\Lambda}_{(\mathbf{R})})^n\mathbf{V}_{(\mathbf{R})} \\ 0 &< \mu < \frac{1}{||\lambda_{max,(\mathbf{R})}||}\end{aligned} \tag{2.23}$$

where $\mathbf{V}_{(\mathbf{R})}$ is the right eigenvector matrix of $\mathbf{R}$ and $\mathbf{\Lambda}_{(\mathbf{R})}$ is the eigenvalue matrix. It is easy to see that the weights will not converge if $\mu$ is too large.

## 2.5 Normalized LMS Algorithm

The normalized LMS (NLMS) algorithm replaces the constant $\mu$ with a time varying growth factor $\mu[n]$. Specifically, [10, p.355]

$$\begin{aligned} \mu[n] &= \frac{\mu}{\mathbf{x}^T[n]\mathbf{x}[n]} \\ \mathbf{w}[n+1] &= \mathbf{w}[n] - \frac{\mu}{\mathbf{x}^T[n]\mathbf{x}[n]}\mathbf{x}[n]\,\varepsilon[n] \end{aligned} \tag{2.24}$$

The benefit of normalizing the growth factor with respect to the input is that it causes the gradient to be more resistant to input noise when the amplitude of the input is relatively large [10, p. 352]. Another benefit of the NLMS algorithm is that the stability conditions for $\mu$ are constant. As shown in (2.23), the limits on $\mu$ for the LMS algorithm are relative to $\lambda_{max,(\mathbf{R})}$, which depends on $\mathbf{x}[n]$. However, for the NLMS algorithm, [10, p. 355]

$$0 < \mu < 2\,. \tag{2.25}$$

## 2.6 Applications of the LMS Algorithm

In signal processing, the LMS algorithm is used as a filter. The applications of the LMS filter all differ with how the inputs and outputs are connected. The two basic LMS filter configurations are parallel (the general form) and traversal [11, p. 16]. Figures 2-3 and 2-4 show the difference between the two configurations. The input vector for the parallel configuration is a set of inputs at a particular time. The input vector for the traversal configuration is a set of delayed time samples from a single input. These two LMS filter configurations can be used in a number of applications. There are four basic classes of adaptive filtering applications: system identification, inverse modeling, prediction, and interference cancellation [10, p. 18]. The system identification filter and the interference-cancelling filter are both important to this thesis.

System identification is useful for modeling an unknown system or channel, as shown in Figure 2-5. In the case of the traversal LMS filter, the unknown system is modeled as a set of gains and delays that form a basic finite impulse response (FIR) filter. The LMS filter's weights will adjust themselves to the taps of an FIR filter that best models the unknown system [11, p. 195]. The performance of the algorithm will decrease if there is noise added to the desired signal. Since the LMS filter ideally adjusts its weights to minimize $E[\varepsilon^2]$, it is theoretically unaffected by noise if the input is uncorrelated to the added noise signal [11, p. 196]. In practice, the noise will affect the system because $\varepsilon^2[n]$ is used in place of $E[\varepsilon^2]$. The system identification filter can also potentially fail if the channel of the unknown system is too long.

Adaptive interference cancellation attempts to subtract an undesired signal from the signal of interest, as shown in Figure 2-6. This generally requires that the undesired signal can be modeled or generated [11, p. 304]. However, it is often possible to obtain a reference input that only contains the undesired signal. For example, noise-cancelling headsets have a microphone outside of the speaker that detects and

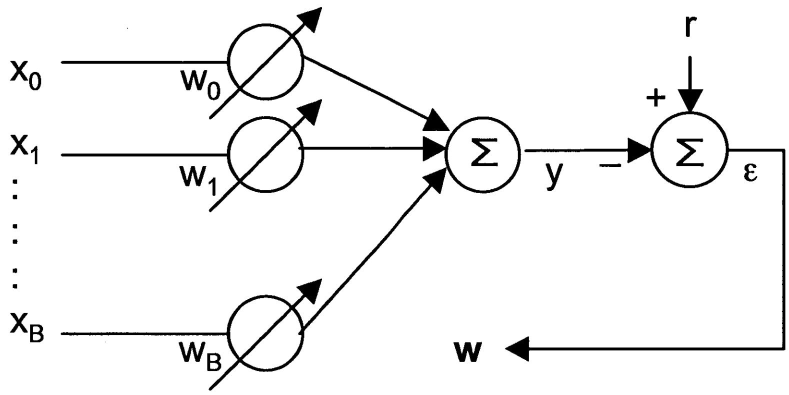


Figure 2-3: Parallel LMS configuration with filter input $\mathbf{x}$, weights $\mathbf{w}$, output $y$, desired input $r$, and error $\varepsilon$ [11, p. 16].

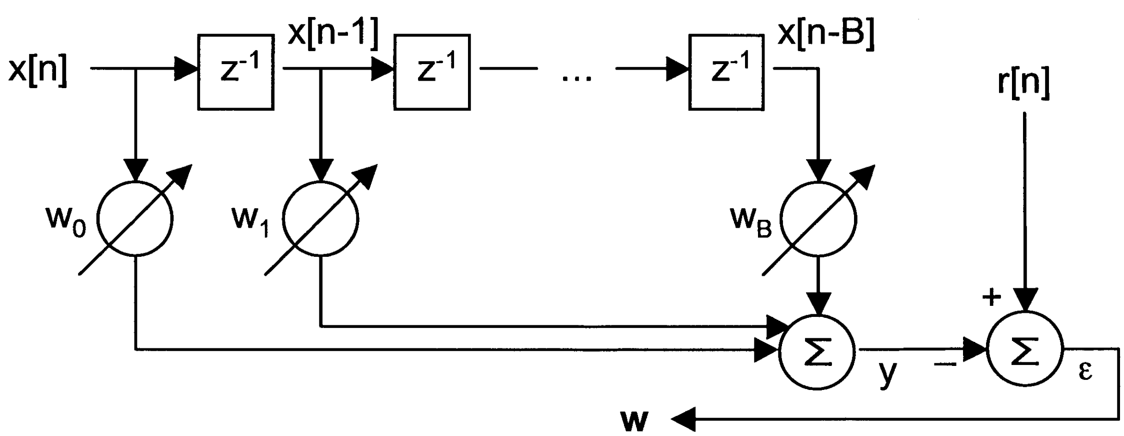


Figure 2-4: Traversal LMS configuration with filter input $\mathbf{x}$, weights $\mathbf{w}$, output $y$, desired input $r$, and error $\varepsilon$ [11, p. 16].

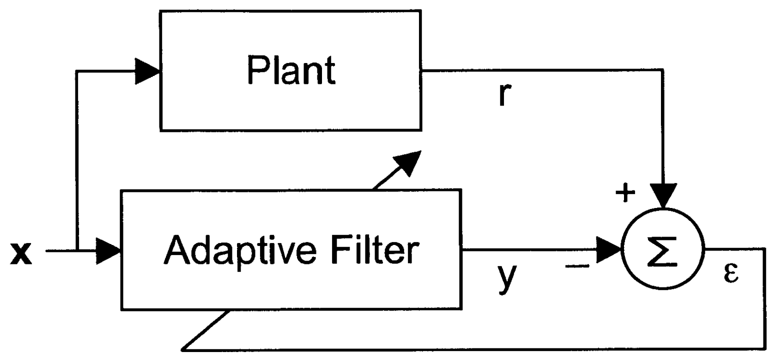


Figure 2-5: System identification LMS configuration with system input $\mathbf{x}$, filter output $y$, plant output $r$, and error $\varepsilon$ [10, p. 19].

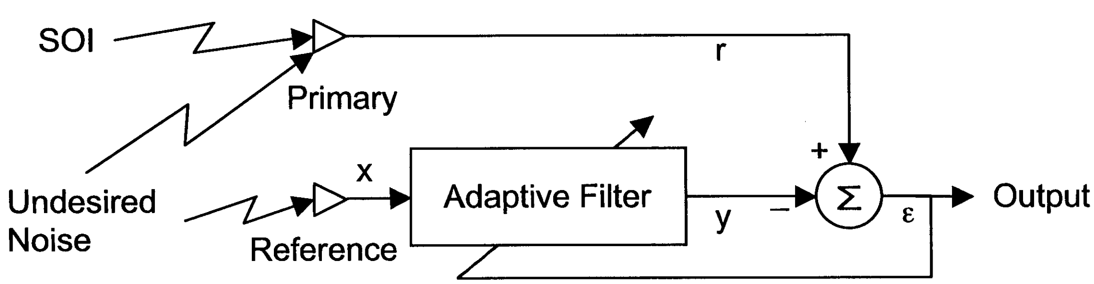


Figure 2-6: Traversal LMS configuration with filter input $\mathbf{x}$, weights $\mathbf{w}$, output $y$, desired input $r$, and error $\varepsilon$ [11, p. 304].

records unwanted noise [11, p. 338], which is then subtracted from the audio signal. However, in many cases, it is difficult to model the channel between the two inputs. Noise-cancelling headsets must delay the noise signal by an amount related to the distance between the microphone and the speaker. The weights of an adaptive noise canceller can sometimes account for this difference.

## 2.7 Signal Orthogonality

Another concept important to signal cancellation is that of orthogonality. Orthogonality occurs when the cross-correlation (the time expectation of the product) of a signal and the complex conjugate of another signal is zero. In general, two sinusoids with different periods are orthogonal over a certain period of integration $L$ if this period is a common multiple of the sinusoid periods. This means that the cross-correlation of sinusoids $r_1(t)$ and $r_2(t)$ with frequencies $\omega_1$ and $\omega_2$ is [8, p. 190]

$$\begin{aligned} E[r_1(t)r_2(t)] &= \frac{1}{L}\int_0^L r_1(t)r_2^*(t)\,dt \\ &= \frac{1}{L}\int_0^L e^{j\omega_1 t}e^{-j\omega_2 t}\,dt \\ &= \frac{1}{L}\int_0^L e^{j(\omega_1-\omega_2)t}\,dt \\ &= \left\{\begin{matrix} 1, & \omega_1=\omega_2 \\ 0, & \omega_1\neq\omega_2 \end{matrix}\right\} \end{aligned} \tag{2.26}$$

if the integration interval $L$ is any integer multiple of both $\frac{2\pi}{\omega_1}$ and $\frac{2\pi}{\omega_2}$. The integral over any number of complete periods of a sinusoid is zero. However, when $\omega_1 = \omega_2$, the integrand $e^{j(\omega_1-\omega_2)t}$ is no longer a sinusoid. This property is what allows the Fourier series formula to separate the frequency components of any periodic signal [8, p. 191]. In the Fourier series formula, $L$ will always be a multiple of all $\frac{2\pi}{\omega_1}$ and $\frac{2\pi}{\omega_2}$. Note that the expectation operator $E[\cdots]$ in (2.26) is a time averaging operator.

Sinusoids of the same frequency $\omega_0$ have a cross-correlation that is dependent on their phase difference $\phi_0$. In this case,

$$\begin{aligned} E[r_1(t)r_2(t)] &= \frac{1}{L}\int_0^L e^{j\omega_0 t}e^{-j(\omega_0 t+\phi_0)}\,dt \\ &= \frac{e^{-j\phi_0}}{L}\int_0^L e^{j(\omega_0-\omega_0)t}\,dt \\ &= e^{-j\phi_0}\,. \end{aligned} \tag{2.27}$$

This property is important for phased arrays, which will be covered in Section 2.8.

Any two sinusoids of different frequencies can still integrate to zero even if $L$ is not a multiple of $\frac{2\pi}{\omega_1}$ or $\frac{2\pi}{\omega_2}$. If $L$ is very large compared to $\frac{2\pi}{\omega_1}$ and $\frac{2\pi}{\omega_2}$, the result will still be

$$\begin{aligned} E[r_1(t)r_2(t)] &= \frac{1}{L}\int_0^L e^{j\omega_1 t}e^{-j\omega_2 t}\,dt \\ &= \frac{1}{jL(\omega_1-\omega_2)}\left(e^{j(\omega_1-\omega_2)L}-1\right) \\ &\approx 0 \end{aligned} \tag{2.28}$$

given that $\omega_1 \neq \omega_2$. Since any signal can be represented as a sum of sinusoids at different frequencies, the cross-correlation of any two deterministic signals will be nearly zero if these signals occupy different frequency bands.

Averaging random signals over a long period of time is also useful in signal cancellation. Let $R_1(t)$ to be an instance of a zero mean white noise process. If the time-average is taken over a sufficiently long period of time, then [12, p. 154-155]

$$E[R_1(t)] \approx 0\,. \tag{2.29}$$

Again, $E[\cdots]$ is a time-average operator. This result assumes that $R_1(t)$ is mean-ergodic [12, p. 427-428]. In any mean-ergodic signal, the time-average of the signal over a long period of time can be used to estimate the expected value of the signal.

The cross-correlation of two ergodic signals will also be ergodic [12, p. 437]. We know from (2.29) that an instance of a white noise process is ergodic. Sinusoids are also mean-ergodic because their average over a long period of time can approximate their expected value of zero. Since $R_1(t)$ and $r_2(t)$ are both ergodic, we know that

$$E[R_1(t)r_2(t)] = \frac{1}{L}\int_0^L R_1(t)r_2(t)\,dt \approx 0\,. \tag{2.30}$$

The frequency domain provides another way of looking at this problem. $R_1(t)$ is an instance of a white noise process, therefore its total power is spread across its wideband frequency spectrum. Since $r_2(t)$ is extremely narrow band, there is very little frequency domain overlap between $R_1(t)$ and $r_2(t)$, thus there is nearly zero correlation between these two signals.

## 2.8 Receive Adaptive Beamforming

Adaptive antenna arrays use beamforming to allow for directional interference cancellation through multiple reference inputs and weights [11, p. 369]. Beamforming allows an antenna array to position its high gain regions and nulls such that it maximizes the gain in the direction of the SOI while minimizing the gain in other directions that may contain noise or other unwanted signals.

RF signals are electromagnetic waves and can be represented as a sinusoid [15, p. 15]

$$A_r cos(\omega_0 t - kz) \tag{2.31}$$

where $A_r$ is the amplitude of the received signal, $\omega_0$ is the carrier frequency, $z$ is the position along the $z$ axis, and $k$ is the wave number. Note that (2.31) is the equation for a narrowband signal with center frequency $\omega_0$. At the primary receiver, the signal's position $z$ is constant. If we set the coordinate system such that $z = 0$ at the primary receiver, the signal at this receiver $r_p(t)$ can be expressed as

$$r_p(t) \quad = \quad A_r cos(\omega_0 t) + n_p(t)\ . \tag{2.32}$$

where the noise on the primary receiver $n_p$ is modeled as a white noise process. The signal at some reference receiver $r_r(t)$ will be

$$r_r(t) = A_r cos(\omega_0 t + \omega_0 \delta_0)) + n_r(t) \tag{2.33}$$

where $\delta_0$ is a time delay that leads to the phase shift $\omega_0 \delta_0$. We will assume that the noise on the reference receiver $n_r$ is independent and uncorrelated with $n_p$. The time delay $\delta_0$ is the amount of time it takes the wave to travel from the reference receiver

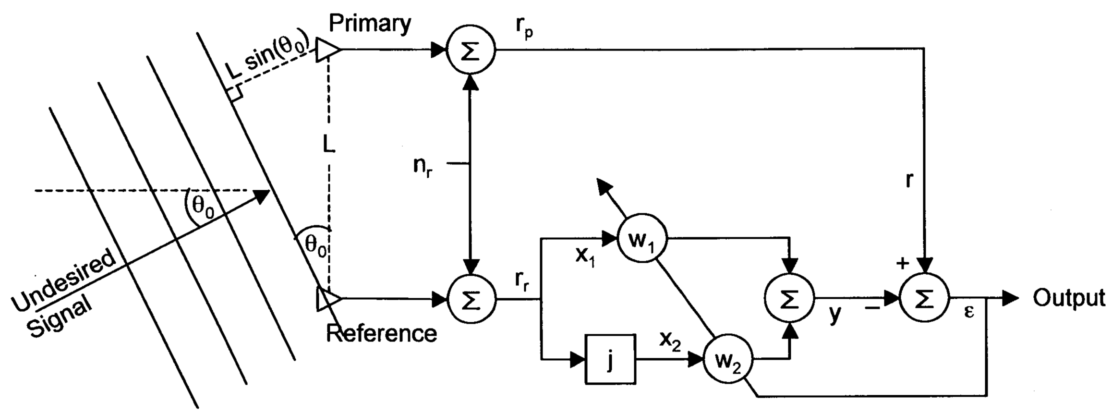


Figure 2-7: Two-antenna receive beamforming that uses LMS interference cancellation with system input $\mathbf{x}$, receiver noise $n_r$, filter output $y$, plant output $r$, and error $\varepsilon$ [11, p. 373].

to the primary receiver in the direction of propagation. It can be expressed as

$$\delta_0 = \frac{2\pi L_0 sin(\theta_0)}{\lambda_0 \omega_0} \tag{2.34}$$

where $\lambda_0$ is the wavelength, $L_0$ is the distance between the primary and reference receivers, and $\theta_0$ is the angle of propagation relative to the receivers shown in Figure 2-7.

Adaptive beamforming uses the LMS algorithm with complex weights. Real weights would allow the gain of the received signal to be modulated. However, the benefit of using complex weights is that both the gain and phase of the received signal can be adjusted. Adjusting the phase of a received signal is vital to beamforming because it allows the two received signals to augment or cancel each other through constructive or destructive interference. Complex weights can be represented by two real weights: one representing the real (in-phase) part, and the other representing the imaginary (quadrature) part. The imaginary weight is simply shifted by 90 degrees, which is equivalent to multiplying it by $j = \sqrt{-1}$ [11, p. 371].

With complex weights configured as a pair of real weights, shown in Figure 2-7, the LMS algorithm can be used to place a null in a certain direction. The LMS beamforming algorithm is set up such that the error is the difference between the primary and reference receivers. Therefore, minimizing the error will cause the weights to configure themselves such that the reference and primary signals cancel each other in the chosen direction.

The optimal weights can be found in a way similar to the method from Section 2.3. This method follows the proof from [11, p. 372]. The weight vector is a complex number that represents the phase shift necessary to form a null. Its parts $w_1$ and $w_2$ form the real and imaginary components of this complex number. As Figure 2-7 shows, the vector $\mathbf{x}[n]$ sent to the weights is

$$\mathbf{x} = r_r[n]\begin{bmatrix}1\\ j\end{bmatrix} = \begin{bmatrix}A_r cos(\omega_0 n+\omega_0\delta_0)+n_r[n]\\ A_r sin(\omega_0 n+\omega_0\delta_0)+n_r[n]\end{bmatrix}. \tag{2.35}$$

The process to find the optimal weights follows by determining the autocorrelation matrix $\mathbf{R}[n]$. This is found to be

$$\begin{aligned}\mathbf{R}[n] &= E[\mathbf{x}\mathbf{x}^H]\\ &= \begin{bmatrix}\frac{A_r^2}{2}+\sigma_r^2 & 0\\ 0 & \frac{A_r^2}{2}+\sigma_r^2\end{bmatrix}.\end{aligned} \tag{2.36}$$

where $\sigma_r^2$ is the noise variance of $n_r$. The result of (2.36) is possible due to the properties of orthogonality from (2.26) and (2.27). Specifically, the cross-correlation of a cosine and sine of the same frequency is zero.

The other vector we must find is $\mathbf{p}[n]$, a vector representing the cross-correlation

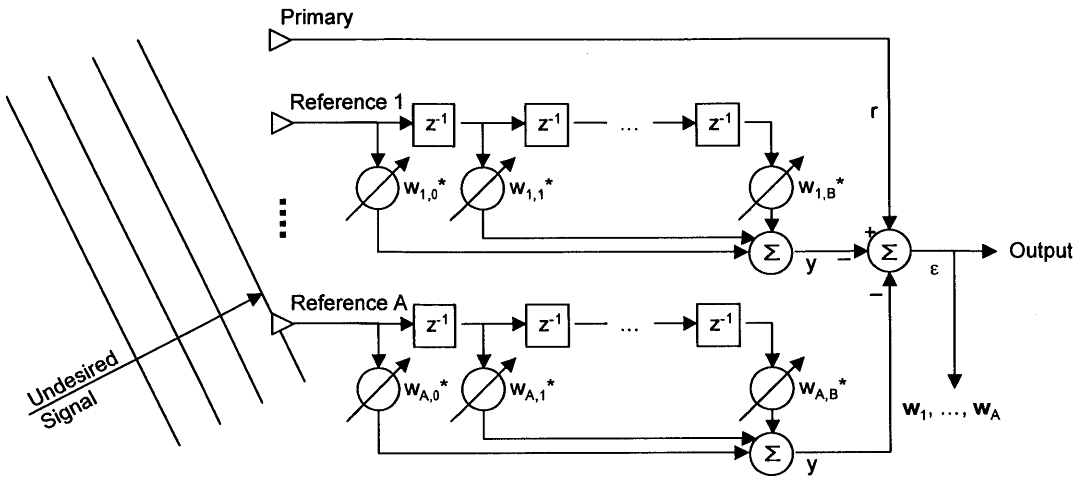


Figure 2-8: Traversal adaptive array for wideband beamforming.

between the input $\mathbf{x}$ and the primary receiver signal $r_p$. This results in

$$\begin{aligned}\mathbf{p}[n] &= E[r_p\mathbf{x}] \\ &= \begin{bmatrix} E[A_r cos(\omega_0 n) A_r cos(\omega_0 n + \omega_0\delta_0)] \\ E[A_r cos(\omega_0 n) A_r sin(\omega_0 n + \omega_0\delta_0)] \end{bmatrix} \\ &= \frac{A_r^2}{2}\begin{bmatrix} cos(\omega_0\delta_0) \\ sin(\omega_0\delta_0) \end{bmatrix} .\end{aligned} \tag{2.37}$$

Again, this result relies on sinusoidal orthogonality. Conveniently, the noise variance does not appear in $\mathbf{p}[n]$ because $r_p$ is uncorrelated with $\mathbf{x}$. With $\mathbf{R}[n]$ and $\mathbf{p}[n]$ determined, the optimal weights $\mathbf{w}_{opt}$ can finally be calculated as

$$\begin{aligned}\mathbf{w}_{opt}[n] &= \mathbf{R}^{-1}[n]\mathbf{p}[n] \\ &= \frac{A_r^2}{A_r^2 + 2\sigma_r^2}\begin{bmatrix} cos(\omega_0\delta_0) \\ sin(\omega_0\delta_0) \end{bmatrix}\end{aligned} \tag{2.38}$$

using the Weiner solution (2.17). These optimal weights will allow the two-antenna narrowband system to form a null in the direction of $\theta_0$.

Additional receivers allow for greater degrees of freedom in placing multiple nulls. In addition, it is possible to place the high gain region in the direction of a SOI in order to maximize its SNR. Wideband signals, however, contain multiple frequencies, and single complex weights and phase shifting may not be sufficient in an adaptive array. For wideband signals, a traversal adaptive array is required [11, p. 399]. As shown in Figure 2-8, the traversal adaptive array not only has the benefit of spatial beamforming, but also has the signal cancellation architecture necessary to process wideband signals.

## 2.9 Transmit Adaptive Beamforming

Transmit adaptive beamforming is another method of adaptive directional cancellation. In this case, each transmitter in an array adjusts its gain and phase such that a null is placed in the direction of the designated receiver. Although a relatively new topic in field of communications, such a strategy can be useful for any wireless application that can be improved by full duplex communication [6].

In this application of transmit beamforming, it is desired that the power at the receiver be zero. Adaptive filtering is naturally useful for such an optimization problem. In a conventional LMS filter, the gradient of the performance surface is derived from reducing the error to zero. Here, the performance surface is the power at the receiver, and a gradient function can be found in terms of the weights in each transmitter's adaptive filter.

To find a function for the power, we must first derive the transmitted signal. Each transmitter contains an adaptive filter with weights $\mathbf{w}$ that act on the reference input $x[n]$. The reference input is common to all of the transmitters but each transmitter's weights can take different values. In any system with a transmit filter, the transmitted signal from transmitter $a$ is

$$t_a[n] = x[n] * \mathbf{w}_a \tag{2.39}$$

where "$*$" stands for convolution. The transmitted signal travels through the air to reach the receiver. On the way, it may reflect off of different surfaces, causing delays in the signal arrival times. The basic natural phenomena that impact signal propagation are reflection, diffraction, and scattering [16]. The entire collection of gains and delays due to these mechanisms is known as the "channel", and can be represented as a transfer function vector $\mathbf{h}$ from the transmitter to the receiver. In a system with $A$ transmitters, the signal at the receiver $r[n]$ is

$$r[n] \quad = \quad \sum_{a=1}^{A} x[n] * \mathbf{w}_a * \mathbf{h}_a \,. \tag{2.40}$$

By definition, the power at the receiver is the square magnitude of the received signal. The power $p[n]$ is

$$p[n] \quad = \quad |r[n]|^2 \,. \tag{2.41}$$

It is often convenient to express the signal $r[n]$ as a vector $\mathbf{r}$. This allows normally complicated operations to be expressed as matrix multiplication. The result from (2.40) can be expressed as

$$\mathbf{r} \quad = \quad \mathbf{E}\mathbf{w} \tag{2.42}$$

where $\mathbf{w}$ is a vector that contains all of the weight values for every transmitter and $\mathbf{E}$ is a matrix that is derived from $\mathbf{w}$ and $\mathbf{h}$. A full derivation of $\mathbf{E}$ is given in Section 4.1. With this notation, the power at the receiver is

$$p = \mathbf{w}^T \mathbf{E}^T \mathbf{E}\mathbf{w} \,. \tag{2.43}$$

Another important condition required for transmit beamforming is that the total transmitted power remains constant. This condition is important because otherwise, the easiest way to achieve zero power would be to set the weight values to zero. This condition can be expressed as

$$\mathbf{w}^T \mathbf{w} = 1 \,. \tag{2.44}$$

The optimal set of weights that minimize (2.43) given the constraint (2.44) can be found via the method of Lagrangian multipliers [7, p. 198]. If we label the functions

$$\begin{aligned} f &= \mathbf{w}^T\mathbf{E}^T\mathbf{E}\mathbf{w} \\ g &= \mathbf{w}^T\mathbf{w} \\ h &= 1\,, \end{aligned} \tag{2.45}$$

then we may specify the Lagrange function $\Lambda$ as

$$\begin{aligned} \Lambda &= f + \lambda(g - h) \\ &= \mathbf{w}^T\mathbf{E}^T\mathbf{E}\mathbf{w} + \lambda(\mathbf{w}^T\mathbf{w} - 1) \end{aligned} \tag{2.46}$$

where $\lambda$ is the Lagrangian multiplier. The Lagrange function takes a form of the Rayleigh quotient [13, p. 440] and it is already apparent that the optimal $\mathbf{w}$ will be an eigenvector of $\mathbf{E}^T\mathbf{E}$. However, we will carry out the rest of the proof by taking the gradient of the Lagrange function and setting it to zero in order to determine its critical points. When we carry out the operation $\nabla\Lambda = 0$, we find that

$$\begin{aligned} \frac{\partial\Lambda}{\partial\lambda} &= 0 = \mathbf{w}^T\mathbf{w} - 1 \\ & \quad\; \mathbf{w}^T\mathbf{w} = 1 \end{aligned} \tag{2.47}$$

$$\begin{aligned} \frac{\partial\Lambda}{\partial\mathbf{w}} &= 0 = \mathbf{w}^T(2\mathbf{E}^T\mathbf{E}) + \lambda(2\mathbf{w}^T) \\ & \quad\; \mathbf{w}^T\mathbf{E}^T\mathbf{E} = -\lambda\mathbf{w}^T \\ & \quad\; \mathbf{E}^T\mathbf{E}\mathbf{w} = \lambda\mathbf{w}\,. \end{aligned} \tag{2.48}$$

The result from (2.48) simply restates the constraint from (2.44). However, the result from (2.48) proves that the optimal $\mathbf{w}$ is an eigenvector of $\mathbf{E}^T\mathbf{E}$. In addition, the corresponding $\lambda$ is an eigenvalue of $\mathbf{E}^T\mathbf{E}$.

We have used the method of Lagrange multipliers to determine the critical points of (2.43) given the constraint (2.44). However, the optimal weight vector must still be chosen from this set of critical points. The power can be determined from (2.48) as

$$
\begin{aligned}
\mathbf{E}^T\mathbf{E}\mathbf{w} &= \lambda\mathbf{w} \\
\mathbf{w}^T\mathbf{E}^T\mathbf{E}\mathbf{w} &= \mathbf{w}^T\lambda\mathbf{w} = \mathbf{w}^T\mathbf{w}\lambda \\
p &= \lambda\,. \qquad (2.49)
\end{aligned}
$$

In other words, the value of the power at a critical point of (2.43) is an eigenvalue of $\mathbf{E}^T\mathbf{E}$. Since we want to minimize the power on the receiver, the optimal weight vector will be the eigenvector that corresponds to the smallest eigenvalue.

# Chapter 3

# Methods and Algorithms for Multiple-Input Single-Output (MISO) Active Cancellation

The basic STAR system uses a single receiver and multiple transmitters. The reason for this is that multiple transmitters allow for beamforming methods to create a null at the receiver. This is desirable because the analog receiver chain and the analog-to-digital converter are intended for low power signals. High signal power will saturate and possibly destroy the receiver's electronics [2, p. 8].

Multiple receive antennae arranged in a phased array can function to create a far-field receive-side null in the direction of the transmitter. However, any signal of interest coming from the direction of the undesired transmitter will not be picked up by the receiver if such a directional null is used. Since the transmitters are stationary with respect to the receiver, these signals of interest will not be detected until the entire platform changes orientation. One exception would be if instead of a directional null, a near-field receive-side null is formed at the transmitter. However, forming a receive-side null at the transmitter is not useful to the STAR system unless the receiver saturation problem is first solved.

In contrast to receive beamforming, transmit beamforming allows a single receiver to function as an isotropic antenna and detect signals of interest from every angle. This is possible because the transmissions can be adjusted to create a null in the interference pattern at the receiver. In fact, there are usually many different far-field interference patterns that result in a null at the receiver [17, p. 81].

One way to form a null at the receiver is to adjust the gain and phase of the transmitters. Under ideal conditions, this would be the best solution for narrow band transmitter signals [4, p. 8]. However, signals with multiple frequencies react differently to fixed delays, and so beamforming becomes difficult with only two adjustable values per transmit antenna [4, p. 12]. In addition, the channel between the transmitters and the receiver is likely to include reflections. Reflections translate to delays and gains in the channel's impulse response [11, p. 201]. For these reasons, the initial multiple-transmitter, single-receiver system uses an adaptive filter on each transmitter. These adaptive filters allow the issues of reflections and wide band signals to be addressed by increasing the number of filter taps. The lower limit to successful transmit beamforming is two transmitters. With knowledge of the channels, two adaptive antennae can theoretically be configured such they cancel at the receiver. This lower limit may not be valid if the channel length is much greater than the number of taps.

This chapter will present three methods of transmit beamforming and two methods of obtaining a channel estimate. These methods all assume multiple transmitters and a channel that can be modeled by its impulse response. In each case, the channel is allowed to vary with time. Practical signal processing applications must account for a time-variant channel even if the antenna platform is stationary [4, p. 62]. It is important to note that none of the methods presented in this chapter require inverting a matrix or finding its eigenvalues. This is very beneficial because large inverse or eigenvalue operations require excessive computation time and resources.

## 3.1 Transmit Beamforming Method: Trial and Error

The most basic method of using adaptive filters in transmit beamforming involves adjusting the filter weights through trial and error. Appendix A.1 gives the Matlab code for this operation. As shown in Figure 3-1, the only adaptive filters in the entire system are located on the transmitters. In this system, there are $A$ transmitters and each transmitter has $B$ weights. Operation of the system requires an arbitrary time period $K$ for which samples can be collected every time a weight is adjusted. The size of $K$ will be mentioned in Section 3.2. Figures 3-2 and 3-3 show that every $K$ cycles, a particular weight from one of the filters is increased by a small step amount. The new average power $p_{avg}$ is then obtained by taking a moving average over the past $K$ measurements of the power seen at the receiver. The received signal $r[m]$ represents a voltage or current. Therefore, the average power is related to the square of the received signal, as shown by [10, p. 116]

$$p_{avg}[n] = \frac{1}{K} \sum_{m=n-K}^{n} |r[m]|^2 \tag{3.1}$$

where $n$ represents the sample number or time. If the new average power is less than the old average power, the weight is left at its new value. Otherwise, it is decreased by twice the step width. At each weight step, all the weights are re-normalized to a specified constant weight power $w_{power}$ so as to keep the transmit power constant. The normalization is accomplished by setting the new weight vector $\mathbf{w}[n+1]$ [7, p. 100]

$$\mathbf{w}[n+1] = \mathbf{w}[n] \left( \frac{w_{power}}{\mathbf{w}^T[n] \cdot \mathbf{w}[n]} \right)^{\frac{1}{2}} . \tag{3.2}$$

This procedure causes all of the weights to change every cycle. However, the overall change in power still largely reflects the change in the stepped weight.

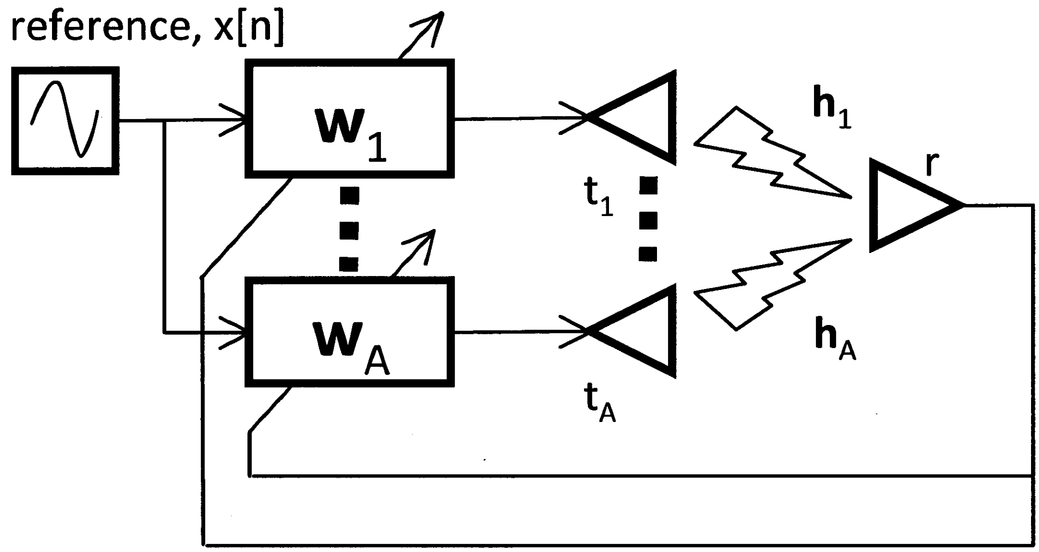


Figure 3-1: System block diagram for a STAR approach using adaptive filters on the transmitters. The transmit weight vector for transmitter $a$ is $\mathbf{w}_a$ and the channel between transmitter $a$ and the receiver is $\mathbf{h}_a$.

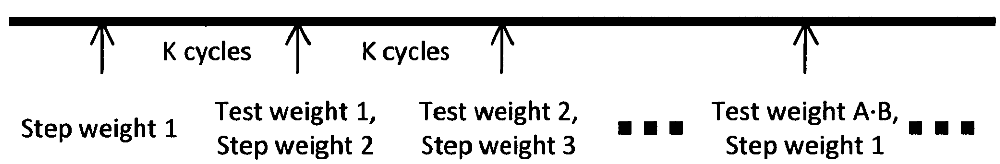


Figure 3-2: Trial and Error algorithm timing diagram. There are $A$ transmitters and $B$ weights per transmitter, therefore, there are $A \times B$ weights total.

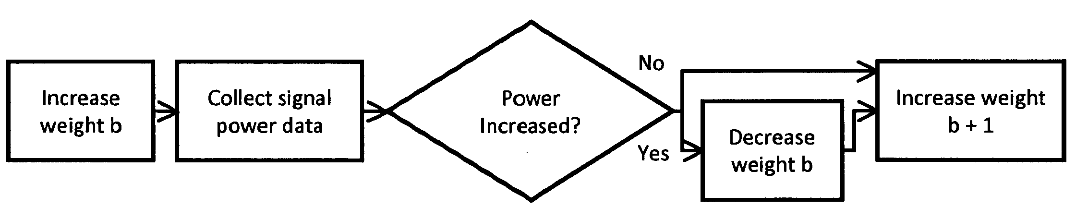


Figure 3-3: Trial and Error algorithm decision flow chart. This shows the algorithm's computation and decision processes every $K$ cycles.

Even with very fine tuning, the performance of this method leaves much to be desired. The weights take a very long time to properly converge, and significant power spikes are produced even at convergence. As Figure 3-4 shows, this method obtains only minimal cancellation. Figure 3-5 reveals that the power spikes are caused by the sharp steps in weight value. It also shows that the weights never really settle. Despite all of its disadvantages, one benefit of this method is that it is virtually immune to noise since all of the calculations use averages of the received data.

The convergence of the weights is largely affected by the speed at which the channel varies. Figure 3-6 shows one realization of the random time-variant channel model that will be used in all the simulations of this chapter. In every method presented, the weights converge more finely if the channel changes slowly.

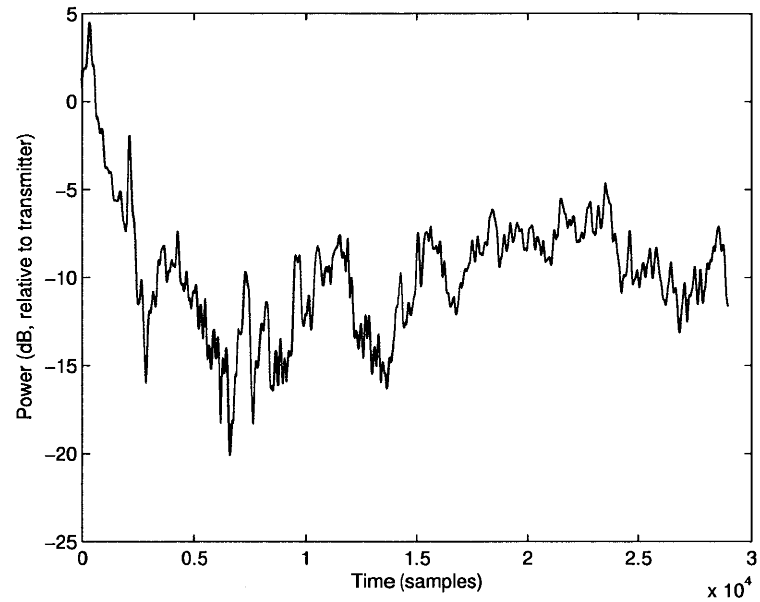


Figure 3-4: Received power relative to transmitted power for the Trial and Error method. In this simulation, $A = 2$, $B = 4$, K = 100, and the weights were stepped by 0.1 every $K$ cycles. The channel model varies with time, as shown in Figure 3-6.

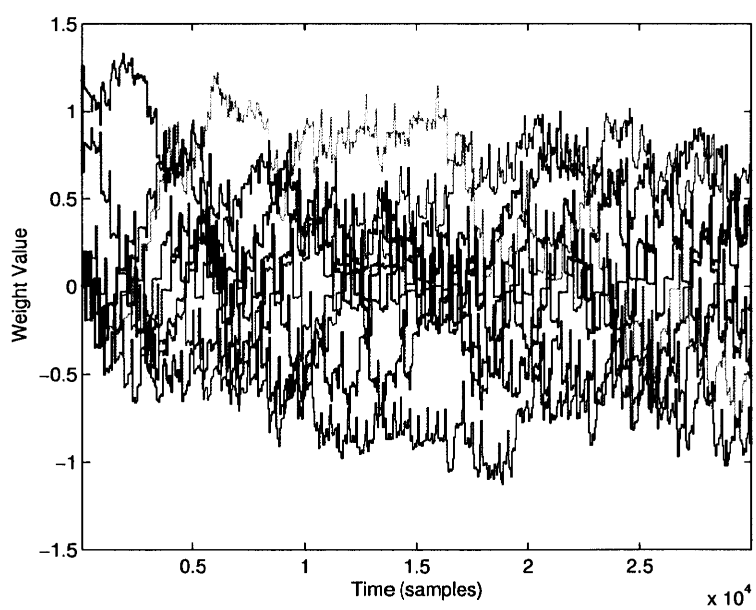


Figure 3-5: Transmitter adaptive filter weights from the simulation in Figure 3-4.

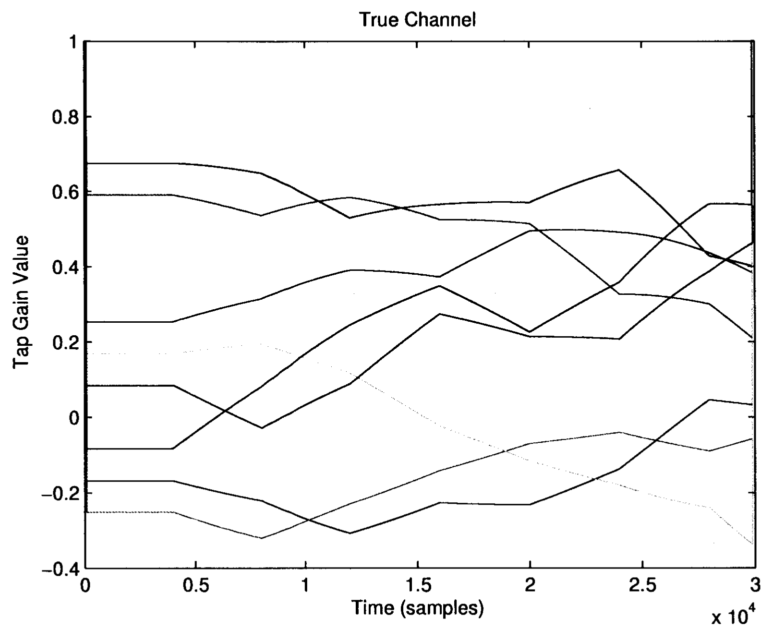


Figure 3-6: Channel realization from the simulation in Figure 3-4. The channel from each transmitter to the receiver has four taps. Each line indicates how the tap values change with time. Every 4000 samples, the slope of these lines changes to a random number between -$\frac{0.1}{4000}$ and $\frac{0.1}{4000}$.

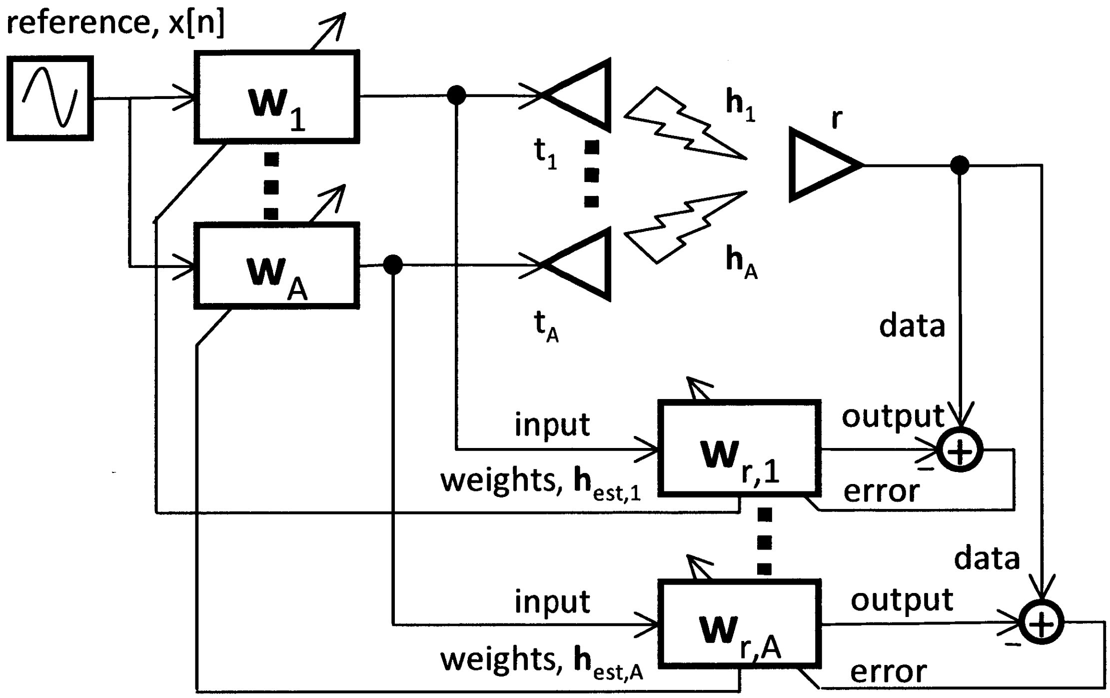


Figure 3-7: System block diagram for a STAR approach that requires a channel estimate for transmit beamforming.

## 3.2 System Identification for Channel Estimation

In contrast to the previous method of transmit beamforming in Section 3.1, all other methods require a channel estimate. The channel estimate is useful because it allows for more sophisticated and intelligent algorithms to be used in the transmitter adaptive filters. For these algorithms, a channel must be estimated from each transmitter to each receiver. A good way to estimate the channel is to use a system identification adaptive filter [18, p. 162]. The system block diagram of Figure 3-7 represents the basic layout used in any method that requires a channel estimate. The issue of actually estimating the channel will be addressed in Section 3.5. For now, the descriptions of these transmit beamforming methods will assume that the channel estimates are perfectly accurate.

An estimate of the average signal power at the receiver is required in order to reduce the actual signal power impinging on the receiver. This estimate can be calculated in terms of known variables and signals. In order to derive the average power estimate, it is important to show how this estimate relates to the actual received signal. The signal $r[n]$ at the receiver is the sum over all $A$ transmitters of each transmitter signal $t_a[n]$ convolved with its respective channel $\mathbf{h}_a$ between transmitter and receiver. The transmitter signal itself is a convolution between the transmitter weights $\mathbf{w}_a$ and some common reference signal $x[n]$. As explained in later sections, the reference signal can be designed to have multiple uses. For now, the reference signal can simply be understood to be any signal input to all the transmitter adaptive filters.

The receiver ultimately sees the sum of each double convolution of the reference, transmitter weights, and channel for each transmitter as shown by

$$\begin{aligned} r[n] &= \sum_{a=1}^{A} t_a[n] \\ &= \sum_{a=1}^{A} x[n] * \mathbf{w}_a * \mathbf{h}_a \,. \end{aligned} \tag{3.3}$$

Every cycle, a prediction of the received signal is calculated by

$$r_{pred}[n] = \sum_{a=1}^{A} x[n] * \mathbf{w}_a * \mathbf{h}_{est,a} \,. \tag{3.4}$$

This prediction $r_{pred}[n]$ is based on the current values of the reference signal, transmitter weights, and channel estimate $\mathbf{h}_{est,a}$. The channel estimate comes from copying the weights of the receiver's system identification LMS filters. Once a prediction is obtained for the received signal, it is used to determine a prediction for the average power as

$$p_{avg,pred}[n] = \frac{1}{K} \sum_{m=n-K}^{n} |r_{pred}[m]|^2 \,. \tag{3.5}$$

In a sense, this average is a moving average whose length is determined by the number

$K$ of elements that were taken from the reference signal. It is necessary for $K$ to be large enough such that (3.5) can average over enough samples to include the delays due to $\mathbf{w}$ and $\mathbf{h}$. The smallest value for $K$ is the length of the impulse response of $\mathbf{w} * \mathbf{h}$, which is equal to the sum of the lengths of $\mathbf{w}$ and $\mathbf{h}$.

## 3.3 Transmit Beamforming Method: Gradient Descent

The transmit beamforming method discussed in this section directly optimizes the received power using a gradient calculation. This method requires a channel estimate. The derivation for the formula used to calculate the weight update vector is similar to that of the LMS algorithm. As explained in Section 2.4, the LMS algorithm uses the instantaneous squared error $\epsilon^2[n]$ as an estimate of the mean squared error $E[\epsilon^2[n]]$. The negative gradient vector of the power, $-\nabla[n]$, points in the direction that most directly minimizes the mean squared error and contains the weight direction and magnitude with which to update each weight [11, p. 21]. Once the gradient is calculated, the weights themselves are updated in the same way as the LMS filter, given in (2.20) where $\mu$ is the adjustable growth factor.

As previously shown in (2.19), $\nabla[n]$ is calculated by taking the partial derivative with respect to each weight. Similar to the LMS algorithm, the gradient vector for the transmit beamforming method of this section is also calculated using partial derivatives. The main difference is that the LMS algorithm attempts to minimize the mean squared error of the filter [11, p. 100], whereas here we attempt to minimize the average signal power at the receiver. In this sense, the performance surface of the algorithm discussed in this section is the average receiver power as a function of each weight in each transmitter's adaptive filter.

The negative gradient vector is the vector of steepest descent along the performance surface. The average power gradient vector for transmitter $a$ is

$$\nabla_a[n] = \frac{\partial p_{avg,pred}[n]}{\partial \mathbf{w}_a} = \begin{bmatrix} \frac{\partial p_{avg,pred}[n]}{\partial w_{a,1}} \\ \vdots \\ \frac{\partial p_{avg,pred}[n]}{\partial w_{a,B}} \end{bmatrix}. \tag{3.6}$$

As shown in (3.5), the predicted average power $p_{avg,pred}[n]$ is a function of the predicted received signal $r_{pred}[n]$. In addition, (3.4) shows that $r_{pred}[n]$ is a function of $x[n]$, $\mathbf{w}_a$, and $\mathbf{h}_{est,a}$. Since $x[n]$ and $\mathbf{h}_{est,a}$ are constant during this partial derivative calculation, $r_{pred}[n]$ is really just a linear function of each $w_{a,b}$. Using the chain rule,

$$\begin{aligned}
\nabla_a[n] &= \frac{\partial p_{avg,pred}[n]}{\partial \mathbf{w}_a} \\
&= \left(\frac{\partial p_{avg,pred}[n]}{\partial r_{pred}[n]}\right)\left(\frac{\partial r_{pred}[n]}{\partial \mathbf{w}_a}\right) && (3.7)\\
&= \frac{2}{K}\sum_{m=n-K}^{n} r_{pred}[m]\frac{\partial r_{pred}[m]}{\partial \mathbf{w}_a}\ . && (3.8)
\end{aligned}$$

Note that the partial derivative in (3.7) only evaluates to the solution in (3.8) when the samples in $r_{pred}$ are real. The original predicted power in (3.5) uses $|r_{pred}|$, which does allow for complex samples in $r_{pred}$, but would result in a much more complicated partial derivative.

The received signal prediction $r_{pred}[n]$ is the sum of the signal contributions from each transmitter. A prediction for the contribution from transmitter $a$ can be expressed as

$$\begin{aligned}
r_{pred,a}[n] &= (x[n] * \mathbf{h}_{est,a}) * \mathbf{w}_a \\
&= g_a[n] * \mathbf{w}_a \\
&= \sum_{b=1}^{B} g_a[n-b]w_{a,b}\ . && (3.9)
\end{aligned}$$

Since each weight $w_{a,b}$ in a traversal filter with weight vector $\mathbf{w}_a$ represents a gain and a delay [10, p. 5], this equation depicts how the weight vector is convolved with the other terms. It is now apparent that the partial derivative of the received signal with respect to each weight is

$$\frac{\partial r_{pred}[n]}{\partial \mathbf{w}_{a,b}} = g_a[n-b]\ . \tag{3.10}$$

From this equation and (3.9), it is finally possible to determine the gradient vector from (3.8) to be

$$\nabla_a[n] = \frac{2}{K} \sum_{m=n-K}^{n} \begin{bmatrix} r_{pred}[m] g_a[m-1] \\ \vdots \\ r_{pred}[m] g_a[m-B] \end{bmatrix} . \tag{3.11}$$

Appendix A.2 gives the Matlab code for the entire operation.

The method of this section performs much better than the method of Section 3.1. As shown by the simulation results of Figure 3.3, the average received power can drop to 20dB below the transmit power within 30000 samples. Another benefit of this method is that it uses direct calculation. This method does not use any conditional statements such as those shown in Figure 3-3 to test or compare, making it linear in terms of the weights and much more mathematically sound overall.

However, this method is not without its flaws. The first potential problem is the speed of convergence. The system has a good steady state average power and has good long term behavior in general. However, if one of the channels were to undergo a large change in a small amount of time, the long transient response could pose problems for achieving the goals of STAR. Another potential problem with this system is that the power at the receiver does not remain constant after the weights have converged. The simulation from Figures 3.3 and 3-9 shows that even after convergence, the received power ranges from -15dB to -25dB relative to the transmitter. Such behavior is no surprise considering that the transmitter weights are also somewhat mobile after convergence.

It should be noted that the performance of this method is heavily dependent on the growth factor. When the growth factor was lowered from 0.02 to 0.005, it caused the weights to be much more stable and lowered the average receive power by an additional 5dB as displayed in Figure 3-10. However, with a smaller growth factor,

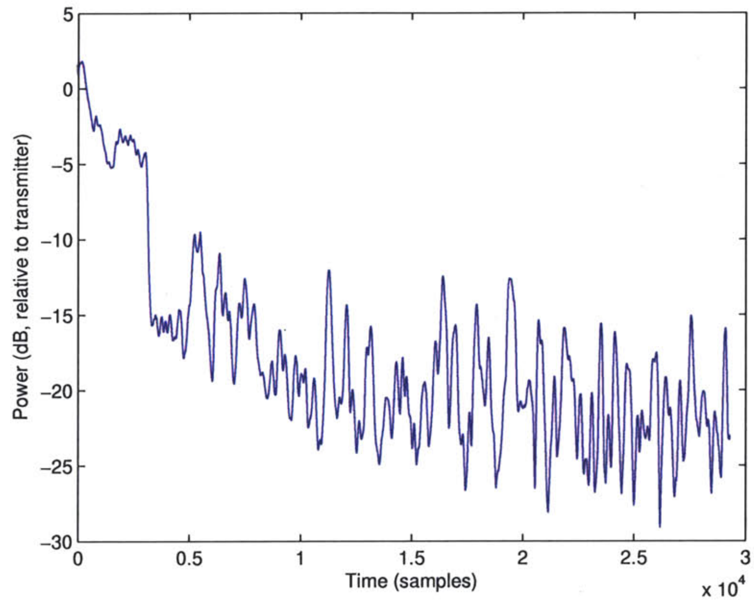


Figure 3-8: Received power relative to transmitted power for the Gradient Descent method. In this simulation, $A = 2$, $B = 4$, and $\mu = 0.05$. The channel model varies with time.

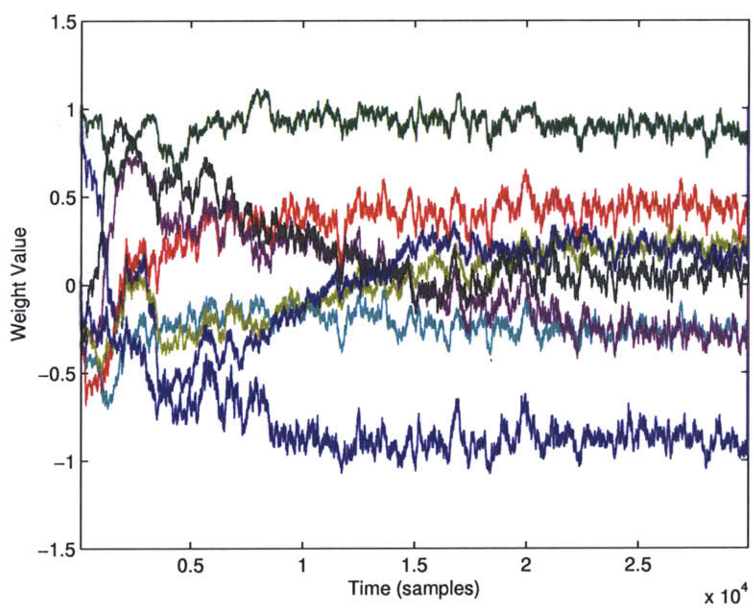


Figure 3-9: The adaptive filter weights from the simulation in Figure 3.3.

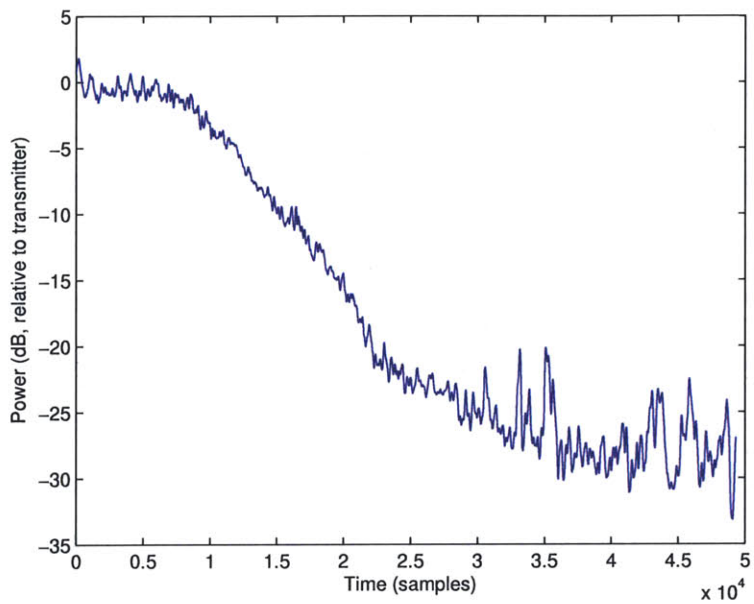


Figure 3-10: Received power relative to transmitted power for the Gradient Descent method. This simulation instead uses $\mu = 0.005$. Note that this simulation was run for 50000 samples.

the system will take even longer to converge. This behavior is similar to that of the LMS filter [11, p. 50] and the user may choose to customize this method for either speed or performance.

## 3.4 Transmit Beamforming Method: Trial and Error Using a Channel Estimate

The method discussed in this section uses a test and comparison approach to find the weight update vector for the transmit beamforming adaptive filter. Like the method of Section 3.3, it involves the use of an adaptive filter on the receiver that estimates the channel between each transmitter and receiver. However, it is also similar to the method of Section 3.1 in that the weight update algorithm uses conditional statements to determine the gradient (as opposed to the strict mathematical approach of Section 3.3). The method of this section essentially improves the structured gradient descent algorithm through the use of nonlinear conditional statements. Appendix A.3 gives the Matlab code for this operation.

A prediction for the average power $p_{avg,pred}$ is determined in the same way as in (3.4) and (3.5). After obtaining this prediction, each weight is individually tested to determine whether a slight increase or decrease in the weight's value will lead to an overall decrease in $p_{avg,pred}$. To test the effect of a slight increase $\alpha$ in a weight's value, a temporary average power prediction $p_{temp,up}$ is obtained. The value of $p_{temp,up}$ can be found by increasing the desired weight by $\alpha$ and using the new weight vector in (3.4) and (3.5). Similarly, $p_{temp,down}$ is calculated to determine the effect of a slight decrease in the weight's value. Once all average power predictions are determined, $p_{avg,pred}$, $p_{temp,up}$, and $p_{temp,down}$ are compared with each other using conditional statements. Depending on which of the average power predictions has the lowest value, the corresponding weight will either be increased, decreased, or left unchanged.

As with the method from Section 3.3, the weight update calculation is the same as that for the LMS filter, given in (2.20). The main difference lies in how the gradient is calculated. For transmitter $a$, the gradient vector $\nabla_a$ is

$$\nabla_a = \frac{\partial p_{avg,pred}}{\partial \mathbf{w}_a} . \tag{3.12}$$

A good estimate for this partial derivative is the change in average power divided by the change in weight $\alpha$. In fact, this estimate is an exact calculation of the partial derivative in the limit $\alpha \to 0$. If it was found that $p_{temp,up} < p_{avg,pred}$ and $p_{temp,up} < p_{temp,down}$ for an increase in weight $b$ of transmitter $a$, then

$$\frac{\partial p_{avg,pred}}{\partial w_{a,b}} \approx \frac{\Delta p_{avg,pred}}{\Delta w_{a,b}} = \frac{p_{temp,up} - p_{avg,pred}}{\alpha} . \tag{3.13}$$

In this case, the change in average power is the difference between $p_{temp,up}$ and $p_{avg,pred}$. If instead $p_{temp,down} < p_{avg,pred}$ and $p_{temp,down} < p_{temp,up}$, then

$$\frac{\partial p_{avg,pred}}{\partial w_{a,b}} \approx \frac{p_{temp,down} - p_{avg,pred}}{-\alpha} . \tag{3.14}$$

This method of transmitter beamforming is the best in both speed and performance. In addition, it is possible to calculate the weights in parallel, allowing for faster computation. The first simulation of this method yielded relative received power as low as -25dB as shown in Figure 3-11. As displayed by Figure 3-12, the weights converge quickly. In Figure 3-12, the weights appear to converge to an exact number. However, due to the nature of this algorithm, what appears to be a convergence is actually an oscillation as shown in Figure 3-13. The weight value oscillates about the theoretical optimal weight value that would yield the lowest received power. This behavior occurs because the weights are always stepped by a constant amount. Since the step is always in the direction of the optimal weight value, the weights will usually overshoot their optimal value.

The weight oscillation can easily be reduced by decreasing the growth factor. To

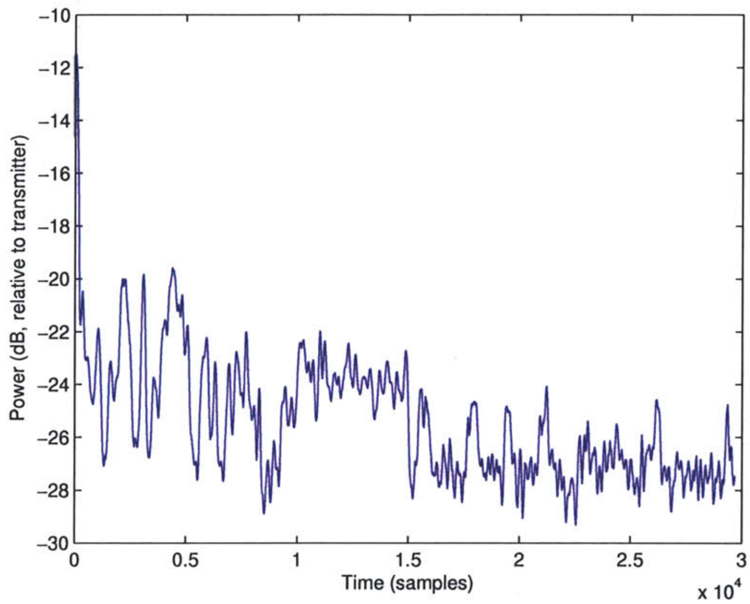


Figure 3-11: Received power relative to transmitted power for the Trial and Error Channel Estimate method. In this simulation, $A = 2$, $B = 4$, $\mu = 0.05$, and $\alpha = 0.05$. The channel model varies with time.

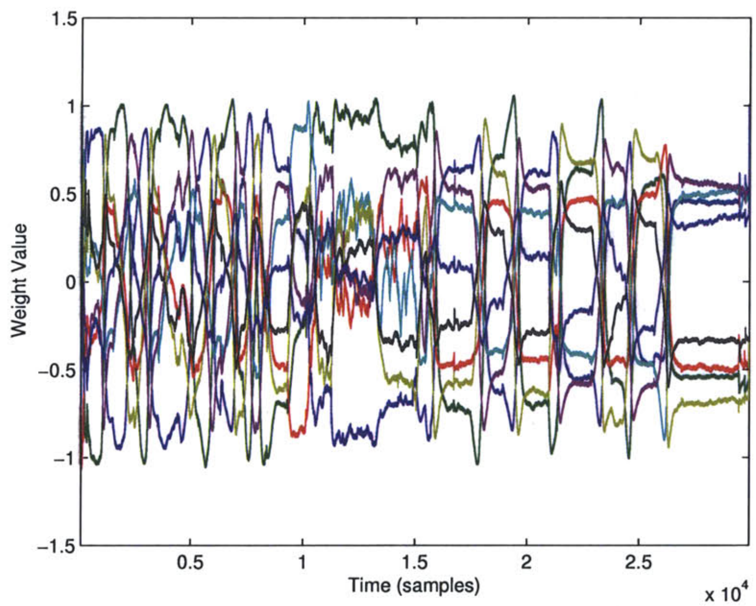


Figure 3-12: Transmitter adaptive filter weights from the simulation in Figure 3-11.

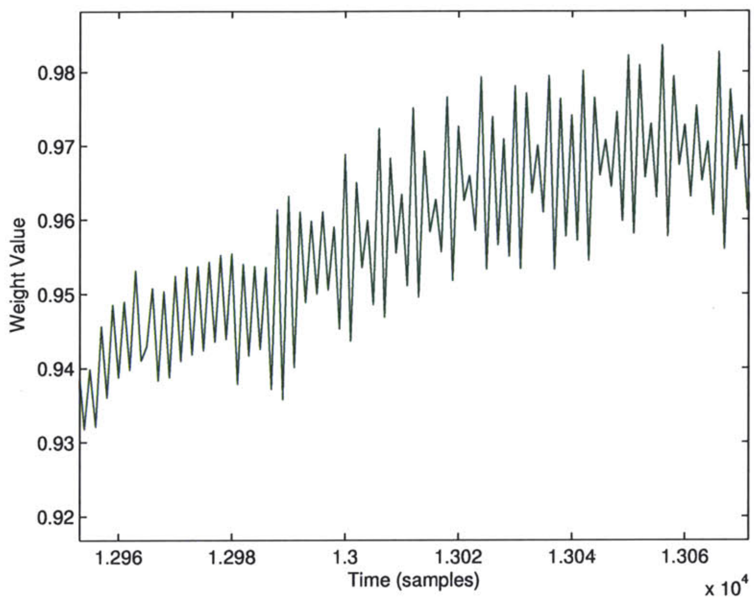


Figure 3-13: Zoomed view of a section of Figure 3-12.

be conservative, $\alpha$ should not be less than $\mu$. However, decreasing both values allows for finer weight step resolution. A finer weight step resolution results in the steady state weight values being much closer to the optimal weight values. As a result, the relative received power improves significantly. Figure 3-14 shows that the relative average received power can be as low as -60dB. However, the time it takes for the weights to converge is greater in the simulation of Figure 3-15 than of Figure 3-12. Such is a characteristic of the tradeoffs involved with adjusting $\mu$ in numerical methods [11, p. 56-57].

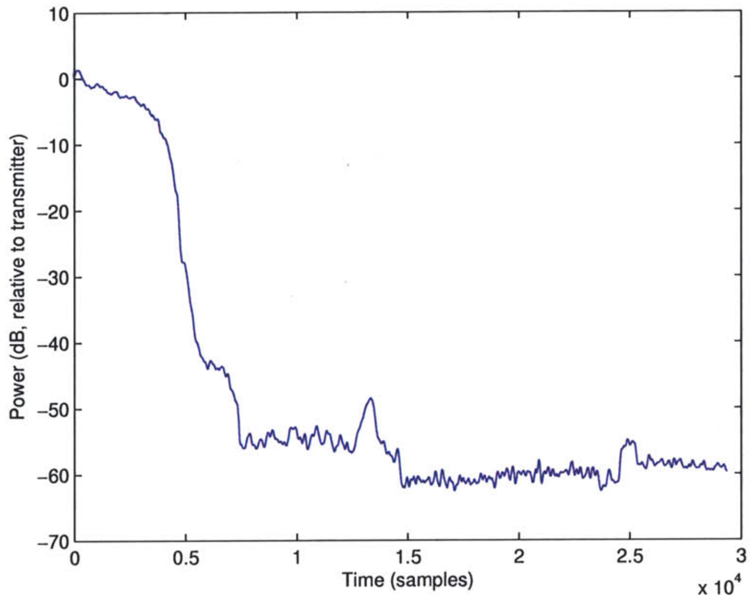


Figure 3-14: Received power relative to transmitted power for the Trial and Error Channel Estimate method. In this simulation, $A = 2$, $B = 4$, $\mu = 0.001$, and $\alpha = 0.001$. The channel model varies with time.

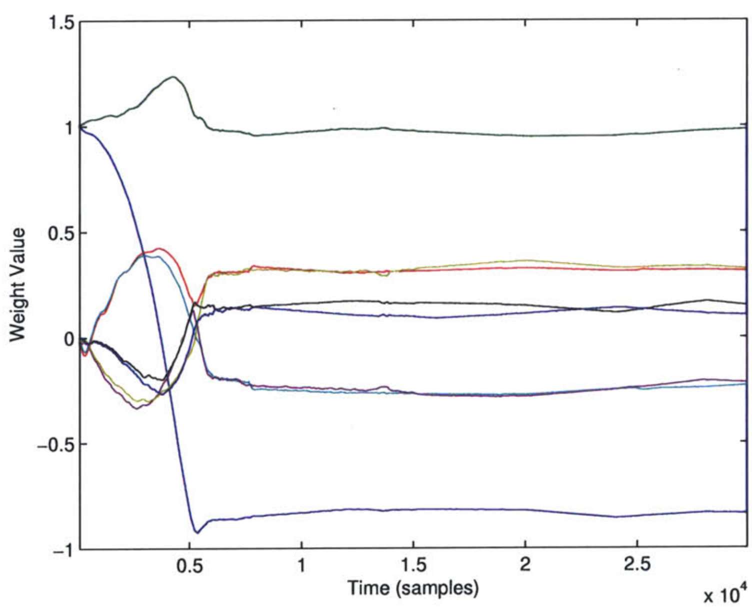


Figure 3-15: Transmitter adaptive filter weights from the simulation in Figure 3-14.

## 3.5 Probe Signals for Channel Estimation

Section 3.4 shows that the best approaches for transmit beamforming require a channel estimate. In the simulations of Sections 3.3 and 3.4, it was assumed that the channel estimate was nearly perfect. However, actually acquiring an accurate estimate is no easy task. As previously shown in Figure 3-7, the transmitter's adaptive filters require adaptive filters on the receiver to identify the channel between each transmitter and the receiver. After the receiver's adaptive filters have successfully identified the channel, their weights can be directly copied as the channel estimate $\mathbf{h}_{est,a}$ for (3.4).

The reason that this task is so complicated is that the receiver contains signals from all of the transmitters. Section 2.6 discusses how the channel of a single transmitter system can be identified by supplying the transmitted signal as input to an LMS filter. However, this will only work if the received signal is solely a result of the transmitted signal. In this system, the received signal is some combination of the transmit beamforming signals, the probe signals, and the signals of interest (SOI). With multiple received signal contributions, special tricks must be used to extract the probe signals from each transmitter in order to identify the channel. Both of the methods that use system identification filters on the receivers require a probe signal. As displayed in Figure 3-16, a different probe signal is added on to each of the transmit beamforming signals. These probe signals are low power and are not designed to be cancelled at the receiver. In addition, each probe signal is given as the input to the corresponding receiver LMS filter. Note that the channel estimation methods to be discussed in Sections 3.6 and 3.7 will use the method discussed in Section 3.4 for transmit beamforming and active cancellation at the receiver.

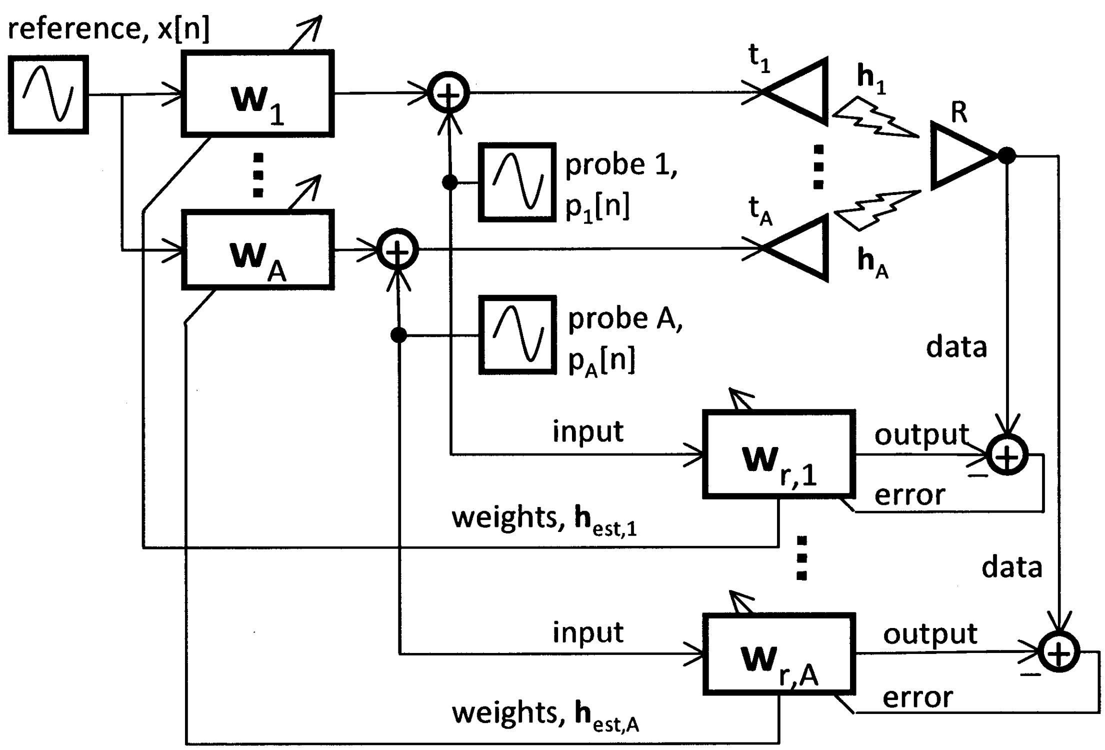


Figure 3-16: System block diagram from Figure 3-7, updated to include probe signals.

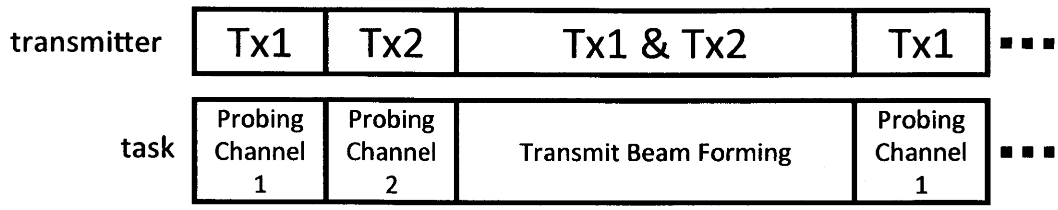


Figure 3-17: Probing Duty Cycle timing diagram that shows which transmitter is active during each phase of the probing period.

## 3.6 Channel Estimation Method: Probing Duty Cycle

This method sequentially probes the channels and cancels the transmit beamforming signal at the receiver. In order to determine a channel estimate, the high power transmit beamforming signal is silenced while probing. Once the channels are properly characterized, the transmit beamforming signal may be reactivated and active cancellation may resume. In this way, the system operates periodically, switching between the phases of channel estimation and transmit beamforming. Appendix A.4 gives the Matlab code for this operation.

During the channel estimation phase, each transmitter is exclusively activated for a small amount of time as shown in Figure 3-17. The signal amplitude for the active transmitter is lowered so as to prevent saturating the receiver. However, this amplitude must be great enough such that it is above the noise and ambient signal level.

Applying a probing duty cycle to the method of Section 3.4 yields a good convergence speed and performance. With the entire system running, the power received can be 30dB or 40dB lower than the transmit power as shown in Figures 3-18 and 3-19.

There are several problems that make this algorithm somewhat impractical. The first problem is that it takes time to probe each channel. Recall from Chapter 1 that

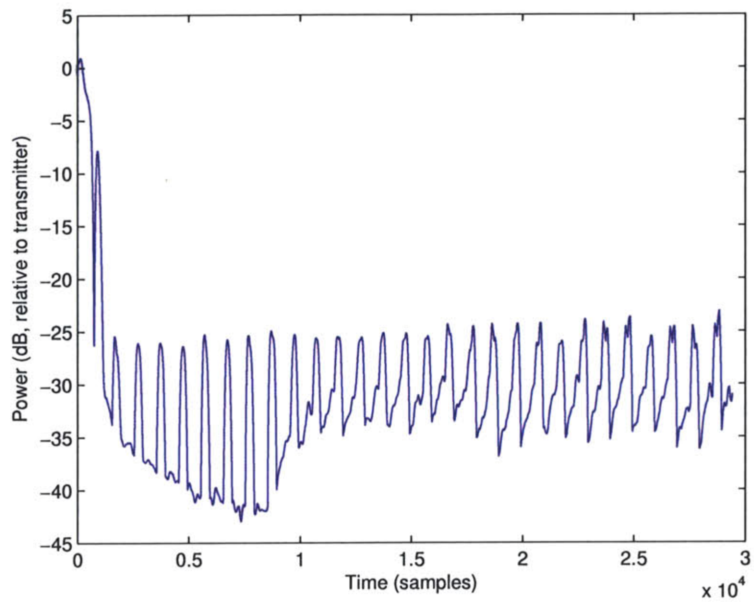


Figure 3-18: Received power relative to transmitted power for the Probing Duty Cycle method. In this simulation, $A = 2$, $B = 4$, $\mu = 0.05$, $\alpha = 0.05$, probing period = 1000 samples, and probing duration = 100 samples. The channel model varies with time.

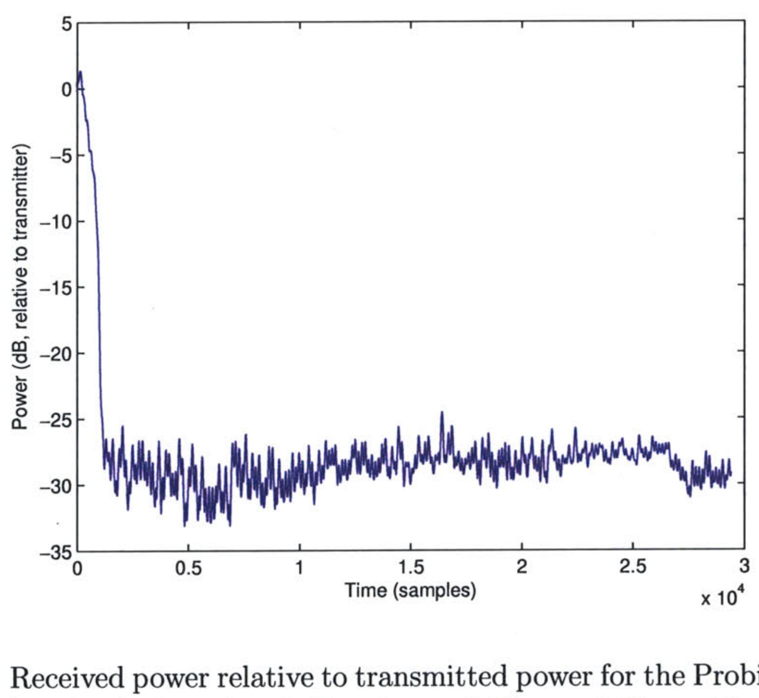


Figure 3-19: Received power relative to transmitted power for the Probing Duty Cycle method. In this simulation, $A = 2$, $B = 4$, $\mu = 0.05$, $\alpha = 0.05$, probing period = 150 samples, and probing duration = 15 samples. The channel model varies with time.

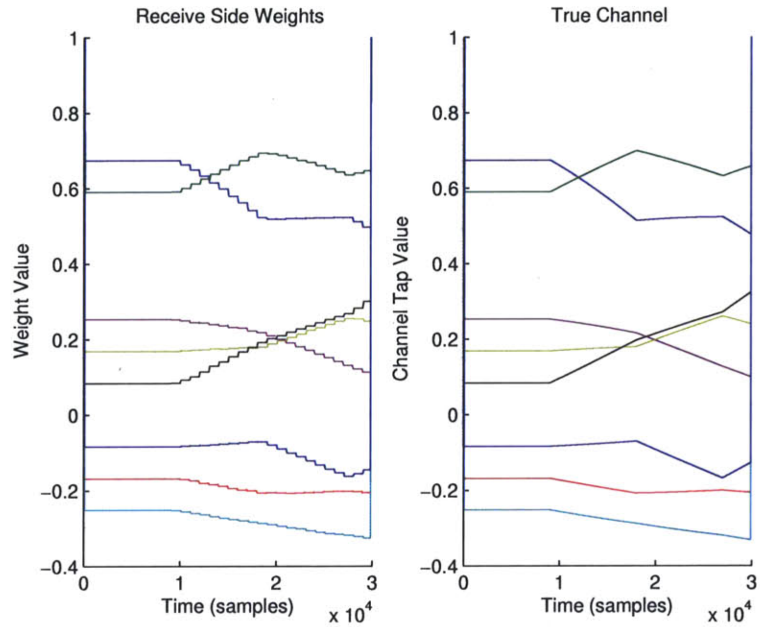


Figure 3-20: Receiver adaptive filter weights (left) and true channel tap values (right) from the simulation in Figure 3-18.

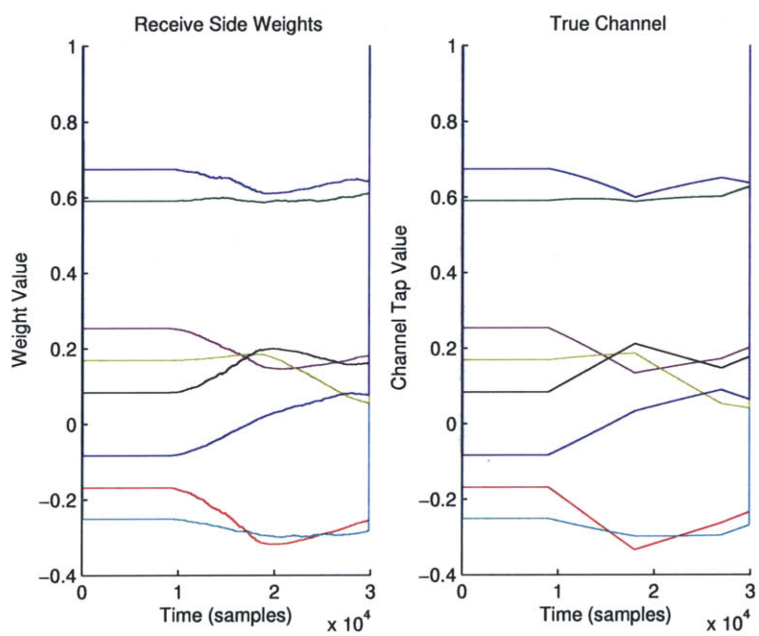


Figure 3-21: Receiver adaptive filter weights (left) and true channel tap values (right) from the simulation in Figure 3-19.

one goal of the STAR system is the high power broadcast of the reference signal [2]. If more time is spent probing the channel, then less time is spent transmitting the reference signal. If the probing time required for an accurate channel estimate is very long, then STAR would be no better than sequentially transmitting and receiving.

Another problem is that the relative received power in Figure 3-18 is littered with periodic power spikes. These spikes come from the fact that the power on the receiver due to the probe signal eventually ends up being 10dB to 20dB greater than that of the transmit beamforming signal. Although these power spikes do not hinder the performance of the transmit beamforming algorithm, they will likely confuse the modules that extract the SOI. The periodic power spikes in Figure 3-18 would essentially add a periodic waveform to the SOI whose fundamental frequency is the probing frequency.

The final problem is the poor characterization of the channels. The receiver LMS filter has a somewhat limited amount of time to converge. Therefore, poorly set growth factors or noise sources can cause problems in convergence. If a weight does not properly converge before it gets locked in, it cannot be corrected until the next receiver LMS update phase. This results in the discontinuous staircase-like weight trajectory displayed in Figure 3-20.

Ways to improve the performance of the receiver LMS filter are to fine-tune its growth factor, lengthen the allowed convergence time, or probe the channels more frequently. Of course, the latter two suggestions would involve taking away from the time that the reference signal can be transmitted. Figures 3-20 and 3-21 both have the same probing duty cycle, but Figure 3-21 probes more often and for a shorter period of time. The simulations from Figures 3-20 and 3-21 demonstrate that the probing period and frequency can be fine tuned together to improve the system performance without increasing the probing duty cycle. It is important to update the channel estimate frequently and also to allow time for the receiver LMS weights to settle.

## 3.7 Channel Estimation Method: Orthogonality-Based Probing Scheme

The method discussed in this section simultaneously probes the channels and cancels the transmit beamforming signal at the receiver. Unlike the method in Section 3.6, this method calls for the constant transmission of a different probe signal from each transmitter. However, its main advantage is that the transmitters may broadcast their transmit beamforming reference signals without interruption. The probe signals are broadcast at much lower power and the system is constantly probing the channels. Appendix A.5 gives the Matlab code for this operation.

This method is based on the principle of orthogonality as discussed in Section 2.7 and uses the properties of cross-correlation to filter the signal on the receiver and extract the desired components. As described in Section 3.5, the received signal consists of the transmit beamforming signals, the probe signals, and the signal of interest. The probe signal is a pseudo-random number (PN) signal that is generated for each transmitter. As shown in (2.29), the probe signal will be uncorrelated with any of the other probe signals. In addition, (2.30) shows that it will be uncorrelated with the signal of interest. In fact, the only component of the received signal that a particular probe signal will be correlated with is the component that solely results from transmitting this probe signal over the channel. Cross-correlating a probe with the received signal allows the adaptive filtering methods in Section 2.6 to identify the channel.

The LMS filter contains a built-in cross-correlation. Recall from (2.20) that the weight vector update equation contains a multiplication of $r[n]$ and $\mathbf{x}[n]$. The multiplication of scalar $r[n]$ with vector $\mathbf{x}[n]$ is effectively a multiplication between the $r$ and $x$ signals. However, the actual LMS filter weight update equation reduces the contribution of $r[n]\mathbf{x}[n]$ by a factor of $\mu$. This reduction is similar to a moving average technique called the "fading memory" technique, which will be discussed later in the section. However, the important concept to understand is that the growth factor acts like a moving average because it reduces the impact of adding $r[n]\mathbf{x}[n]$ to the existing weight values. The weight update equation of LMS filter ultimately cross-correlates $r$ and $x$ through the multiplication $r[n]\mathbf{x}[n]$ and the averaging due to the growth factor $\mu$. This method is similar to the H-TAG method from [19] in that it uses a moving average to alleviate the need for prior channel identification.

The problem with the LMS filter cross-correlation is that the length of the moving average is generally not sufficient for cancelling out uncorrelated signals. Recall from (2.28), (2.29) and (2.30) that uncorrelated signals will only cancel if the length of the moving average $L$ is significantly greater than the period of the signal. In the LMS filter, the moving average length is roughly equal to the number of filter taps, which is generally shorter than the period of $x$ or $r$. Therefore, uncorrelated signal components of $x$ and $r$ will not be absorbed by the LMS filter's cross-correlation.

The solution employed by this method is to use a much longer moving average. Unfortunately, to increase the averaging length of (2.20), one must increase $C$, the number of filter taps. Instead, this method adopts a strategy based on (2.16), the gradient of the expected value of the error $\nabla E[\epsilon^2]$. This gradient can be approximated by averaging its value over many samples. When implemented with a normalized LMS filter, the gradient from (2.16) appears in the weight update equation (2.20) as

$$
\begin{aligned}
&\mathbf{w}[n+1] \\
=\;& E[\mathbf{w}[n]] - \mu\left(\nabla E[\epsilon^2]\right) \\
=\;& E\Big[\mathbf{w}[n]\Big] + \frac{2\mu\left(E\Big[r[n]\mathbf{x}[n]\Big] - E\Big[\mathbf{x}[n]\mathbf{x}^T[n]\mathbf{w}[n]\Big]\right)}{E[\mathbf{x}^T[n]\mathbf{x}[n]]} \\
=\;& \frac{1}{L}\sum_{l=0}^{L}\mathbf{w}[n-l] + \frac{2\mu\left(\sum_{l=0}^{L} r[n]\mathbf{x}[n-l] - \mathbf{x}[n-l]\mathbf{x}^T[n-l]w[n-l]\right)}{\sum_{l=0}^{L}\mathbf{x}^T[n-l]\mathbf{x}[n-l]}
\end{aligned}
\tag{3.15}
$$

where $L$ is the length of the moving average, $r[n]$ is the received signal, and $\mathbf{x}[n]$ is a vector containing the past $C$ values of the LMS input signal $x[n]$. The results of (3.15) cannot be achieved simply by setting $\mathbf{w}[n+1]$ to the average of the past values of $\mathbf{w}[n]$. Although the expectation operation is linear, the division required by (3.15) is nonlinear, and therefore (3.15) cannot be simplified.

Storing values is costly in terms of processing time and resources. Therefore, a much more efficient procedure, known as the fading memory averaging technique is used. This technique assumes the average of the past values of some variable is equal to its previous value. For example, the average of the past $L$ values of $\mathbf{w}[n]$ can be approximated as

$$
\frac{1}{L}\sum_{l=0}^{L}\mathbf{w}[n-l] \approx \frac{(L-1)\mathbf{w}[n-1] + \mathbf{w}[n]}{L}\,. \tag{3.16}
$$

Using the fading memory technique saves on time and resources required to process the moving average.

The method of this section and that of Section 3.6 are two alternatives for an implementation of the system shown in Figure 3-16. In terms received power relative to the transmit power, the method of this section does not perform as well. As shown

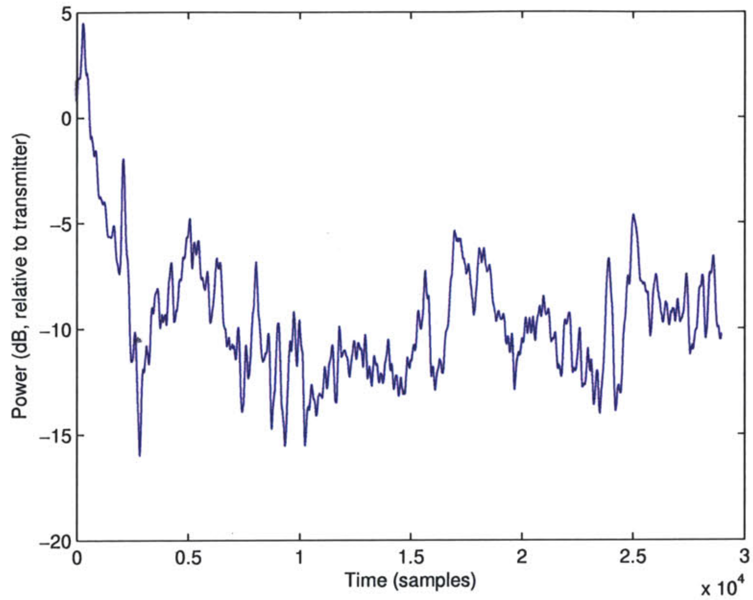


Figure 3-22: Received power relative to transmitted power for the Orthogonality Based Probing method. In this simulation, $A = 2$, $B = 4$, $\mu = 0.05$, $\alpha = 0.05$, $\mu_{Rx} = 1.5$, $L = 5000$. The channel model varies half as fast as the simulations in Section 3.6.

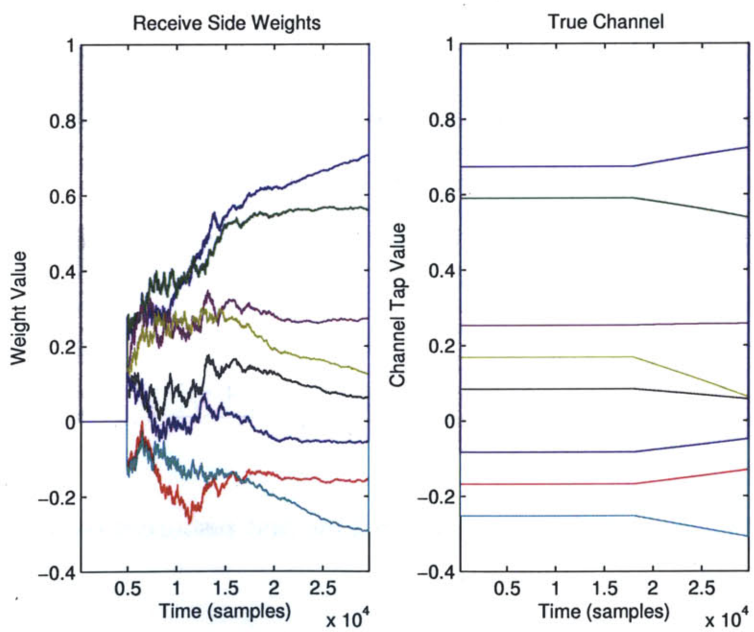


Figure 3-23: Receiver adaptive filter weights (left) and true channel tap values (right) from the simulation in Figure 3-22. The receiver adaptive filter cannot operate until there are $L$ samples of received data available, hence the 5000 sample delay.

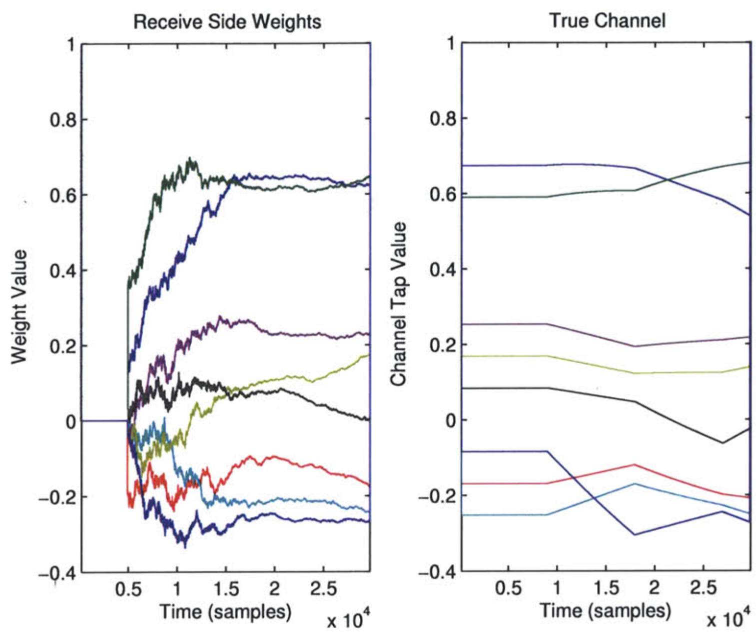


Figure 3-24: Receiver adaptive filter weights (left) and true channel tap values (right). In this simulation, $A = 2$, $B = 4$, $\mu = 0.05$, $\alpha = .05$, $\mu_{Rx} = 1.5$, $L = 5000$. The channel model varies at the same rate as the simulations in Section 3.6.

in Figure 3-22, the relative power settles at -25 dB, which is slightly greater than the -30 dB from Figure 3-19. However, the power level in Figure 3-22 after the weights settle is much steadier than that of Figure 3-19. This result serves to highlight the main advantage that this method has over the method of Section 3.6. The method presented in this section has a full duty cycle of both channel probing and transmit beamforming. Because of this, there will be no power spikes at the receiver and the signal of interest will appear as a clean waveform.

The receiver LMS filter weights of this method, shown in Figure 3-23, are somewhat sloppier than those of the previous method, shown in Figure 3-21. This was expected because an LMS filter that uses a cross-correlation to parse the probe out of the received signal will never be as good as the standard system identification LMS filter whose received signal is directly related to the probe. Despite this disadvantage, it is clear from Figure 3-23 that the receiver's weights settle to the correct values and attempt to track changes in the channel. The simulation from Figure 3-23 uses a slowly changing channel. Due to the moving average, this method would not be able to track a faster channel with much success. Figure 3-24 shows that increasing the moving average length and increasing the rate at which the channel changes results in a set of LMS filter weights that track the channel too slowly to be useful.

## 3.8 Extracting the Signal of Interest

Recall from Chapter 1 that the goal of the system is to extract the SOI from the received signal [2]. When a SOI $s[n]$ is present, the received signal $r[n]$ is

$$r[n] = s[n] + \sum_{a=1}^{A} t_a[n] * h_a \,. \tag{3.17}$$

If we assume that $h_{est,a} \approx h_a$, we can improve the system of Figure 3-16 to become that of Figure 3-25. In this case, we solve for $s[n]$ by calculating

$$s[n] = r[n] - \sum_{a=1}^{A} t_a[n] * h_{est,a} \,. \tag{3.18}$$

The Matlab code for this operation can be found in Appendix A.5.

Figure 3-26 shows the results of applying (3.18) to the orthogonality based probing scheme of Section 3.7. Unfortunately, the channel estimation of this method is not perfect, which causes the estimated $s[n]$ to be noisy. When the $s[n]$ is filtered through an averager, it appears almost identical to the original SOI. It is important to note that higher frequency SOIs will be eliminated if the averager is too long. In addition, the power of the SOI must be somewhere between that of the probe signals and the transmitters. In Figure 3-27, it is clear when the SOI is in operation.

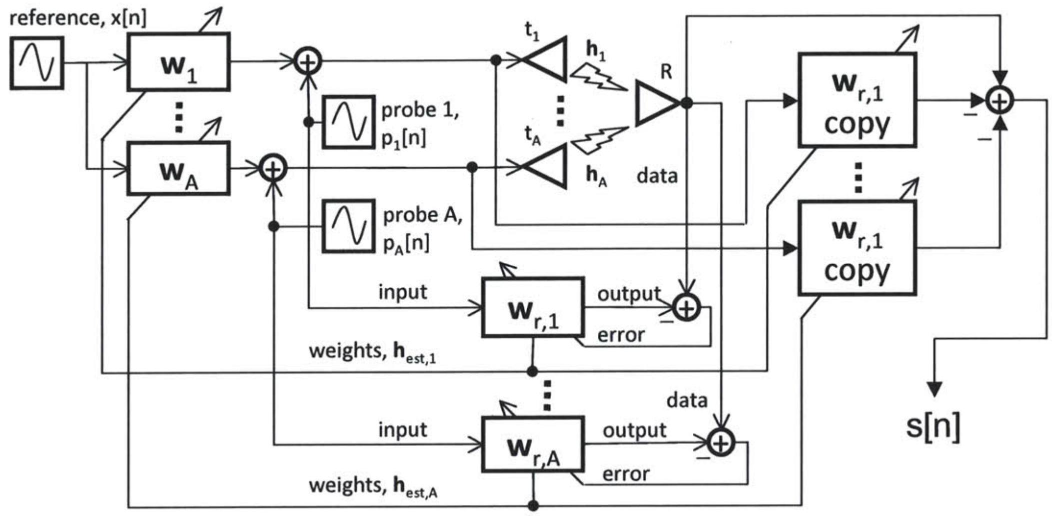


Figure 3-25: System block diagram from Figure 3-16, updated to include $s[n]$.

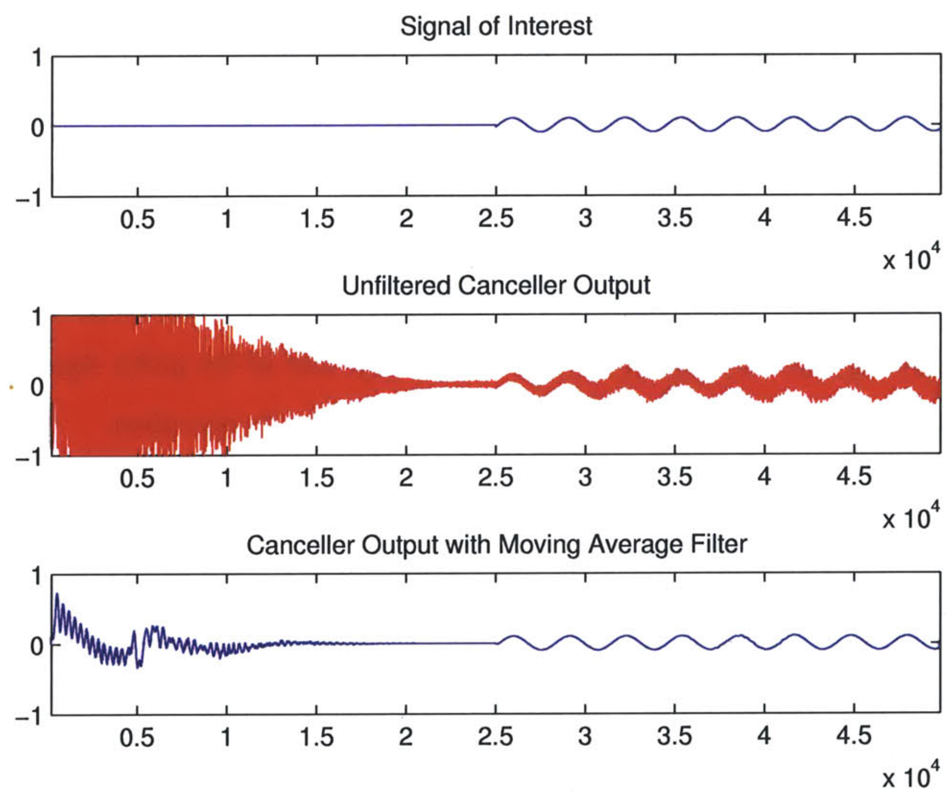


Figure 3-26: SOI, unfiltered $s[n]$, and filtered $s[n]$ for a simulation with conditions identical to that of Figure 3-23.

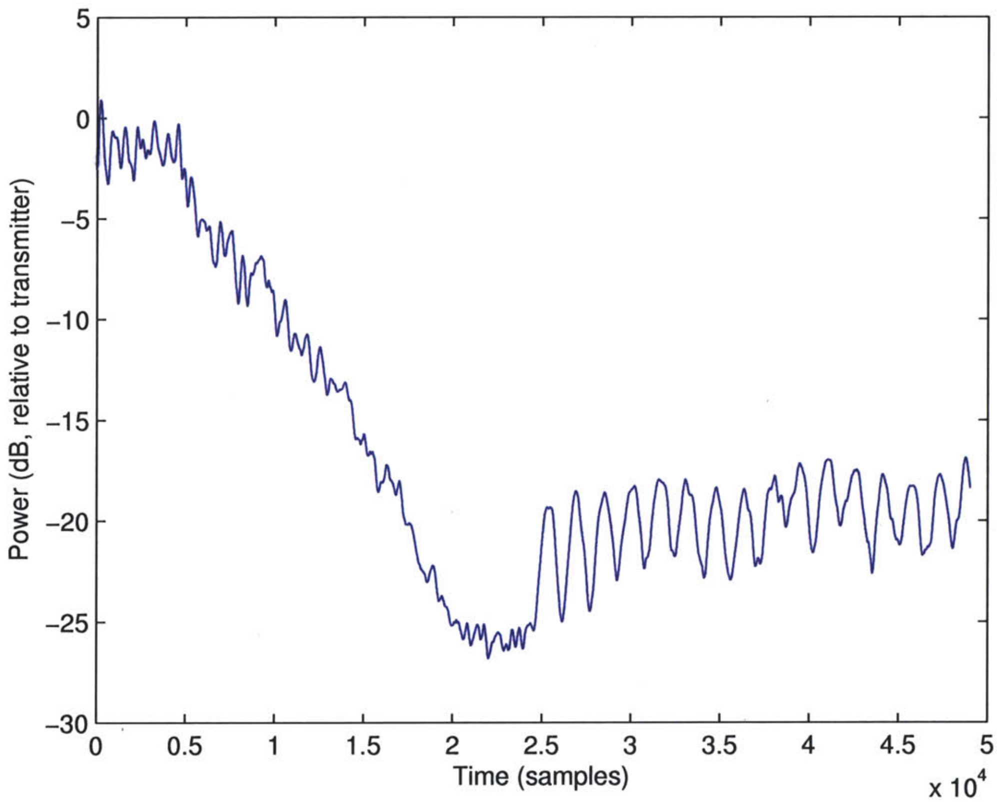


Figure 3-27: Received power for the simulation of Figure 3-26.

# Chapter 4

# Mathematical Analysis of the Trial and Error with a Channel Estimate Method

A mathematical analysis is provided for the method from Section 3.4 titled Trial and Error with a Channel Estimate (TECE). This method was chosen because its simulations produced the best results in both convergence time and steady state receive power. In this chapter, we will develop a linearized state space model for the algorithm. In addition, we will perform a noise analysis of the state space model and verify this analysis via simulation.

## 4.1 Linearized System Dynamics

In this section, we will determine a linearized state space model to describe the weight values. Specifically, we will determine the $\mathbf{A}$ matrix from Section 2.1 for the TECE method. In this context, the $\mathbf{A}$ matrix will describe how $\mathbf{w}[n+1]$ depends on $\mathbf{w}[n]$.

Recall from (3.12) that the weight update equation for the TECE method requires a prediction of the average power $p_{avg,pred}$. The average power prediction, as defined in (3.5), requires the receive signal prediction $r_{pred}$. Recall from (3.4) that

$$\mathbf{r}_{pred} = \sum_{a=1}^{A} \underset{(K\times 1)}{\mathbf{x}} * \underset{(C\times 1)}{\mathbf{h}_{est,a}} * \underset{(B\times 1)}{\mathbf{w}_a} \tag{4.1}$$

where $\mathbf{x}$ is a vector that contains value from $x[n-K]$ to $x[n]$. The vector dimensions have been included in (4.1) and the following equations to assist in analyzing the vector and matrix multiplication.

The convolution of two vectors can be determined by multiplying one vector by a Hankel matrix generated from the other vector. This matrix is related to the Toeplitz matrix, but is more convenient for this analysis [13, p. 183]. The Hankel operation, $han(\mathbf{x})$ is flexible in its dimensions and the resulting Hankel matrix can have an arbitrary number of columns. In this way, the convolutions of (4.1) can be rewritten as

$$\begin{aligned} \mathbf{x} * \mathbf{h}_{est,a} &= \underset{(K_1\times C)}{han(\mathbf{x})} \underset{(C\times 1)}{\mathbf{h}_{est,a}} \\ \mathbf{x} * \mathbf{h}_{est,a} * \mathbf{w}_a &= \underset{(K_2\times B)}{han(\mathbf{x} * \mathbf{h}_{est,a})} \underset{(B\times 1)}{\mathbf{w}_a} \ . \end{aligned} \tag{4.2}$$

where $K_1 = K + C - 1$ and $K_2 = K_1 + B - 1$. To clarify how $han(\mathbf{x})$ designs a

$((K+C-1)\times C)$ matrix,

$$han(\mathbf{x}) = \begin{bmatrix} x[1] & 0 & \cdots & 0 & 0 \\ x[2] & x[1] & & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ x[C] & x[C-1] & \cdots & x[2] & x[1] \\ \vdots & \vdots & & \vdots & \vdots \\ x[K] & x[K-1] & \cdots & x[K-C+2] & x[K-C+1] \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & & x[K] & x[K-1] \\ 0 & 0 & \cdots & 0 & x[K] \end{bmatrix}. \tag{4.3}$$

With the convolution operations of (4.1) simplified to matrix multiplications, we can finally express $\mathbf{r}_{pred}$ as

$$\begin{aligned} \mathbf{r}_{pred} &= \sum_{a=1}^{A} \underbrace{han(\underset{(K_1\times C)}{han(\mathbf{x})}\underset{(C\times 1)}{\mathbf{h}_{est,a}})}_{(K_2\times B)} \underset{(B\times 1)}{\mathbf{w}_a} \\ &= \underbrace{\begin{bmatrix} han(han(\mathbf{x})\mathbf{h}_{est,1}) & \vdots & \cdots & \vdots & han(han(\mathbf{x})\mathbf{h}_{est,A}) \end{bmatrix}}_{(K_2\times AB)} \mathbf{w} \\ &= \mathbf{E}\mathbf{w} \end{aligned} \tag{4.4}$$

where $\mathbf{E}$ is a temporary place holder matrix and $\mathbf{w}$ is an ordered vector that contains the weights of all the transmitters. Specifically,

$$\mathbf{w} = \underbrace{\begin{bmatrix} w_{1,1} & \cdots & w_{1,B} & w_{2,1}\cdots & w_{A,1} & \cdots w_{A,B} \end{bmatrix}^T}_{(AB\times 1)}. \tag{4.5}$$

In terms of these variables, the predicted average power at the receiver is

$$\begin{aligned} p_{avg,pred} &= \frac{1}{K_2}\mathbf{r}_{pred}^T\mathbf{r}_{pred} \\ &= \frac{1}{K_2}(\mathbf{Ew})^T(\mathbf{Ew}) \\ &= \frac{1}{K_2}\mathbf{w}^T\mathbf{E}^T\mathbf{Ew} \ . \end{aligned} \tag{4.6}$$

The previous equations of this section apply for any method that employs a channel estimate $\mathbf{h}_{est}$ and predicted average power $p_{avg,pred}$. The following equations, however, apply specifically for the TECE method. Recall from Section 3.4 that the temporary predicted powers $p_{temp,up}$ and $p_{temp,down}$ are calculated. We will define $p_{temp,n}$ as the temporary predicted power that results from increasing or decreasing weight $n$ by the amount $\alpha$. The value $p_{temp,n}$ does not indicate whether weight $n$ was increased or decreased. However, it will be shown later in this section that stepping weight $n$ up or down will have the same effect on the linearized TECE system. This slight increase or decrease of one of the weights can be expressed as $\mathbf{w} + \mathbf{\Delta w}_n$ where

$$\begin{aligned} \mathbf{\Delta w}_1 &= \alpha\begin{bmatrix}1 & 0 & \cdots & 0 & 0\end{bmatrix}^T \\ \mathbf{\Delta w}_{AB} &= \alpha\begin{bmatrix}0 & 0 & \cdots & 0 & 1\end{bmatrix}^T \\ \mathbf{\Delta w}_{0<n<AB} &= \alpha\begin{bmatrix}0 & \cdots & 1 & \cdots & 0\end{bmatrix}^T \ . \end{aligned} \tag{4.7}$$

To calculate the temporary predicted power, we must substitute $\mathbf{w} + \mathbf{\Delta w}_n$ for $\mathbf{w}$.

With this substitution,

$$
\begin{aligned}
p_{temp,n} &= \frac{1}{K_2}\mathbf{d}_{temp,n}^T\mathbf{d}_{temp,n} \\
&= \frac{1}{K_2}(\mathbf{w}+\mathbf{\Delta w}_n)^T\mathbf{E}^T\mathbf{E}(\mathbf{w}+\mathbf{\Delta w}_n) \\
&= \frac{1}{K_2}(\mathbf{w}^T\mathbf{E}^T\mathbf{E}\mathbf{w}+\mathbf{w}^T\mathbf{E}^T\mathbf{E}\mathbf{\Delta w}_n+\mathbf{\Delta w}_n^T\mathbf{E}^T\mathbf{E}\mathbf{w}+\mathbf{\Delta w}_n^T\mathbf{E}^T\mathbf{E}\mathbf{\Delta w}_n) \\
&\approx \frac{1}{K_2}(\mathbf{w}^T\mathbf{E}^T\mathbf{E}\mathbf{w}+2\mathbf{\Delta w}_n^T\mathbf{E}^T\mathbf{E}\mathbf{w})\,. \qquad (4.8)
\end{aligned}
$$

The final approximation of (4.8) follows by dropping the terms that are quadratic in $\mathbf{\Delta w}_n$.

After applying values from (4.6) and (4.8) to (3.13), we can calculate each component $\nabla_n$ of the gradient vector $\nabla$ to be

$$
\begin{aligned}
\nabla_n &= \frac{1}{\alpha}(p_{pred}-p_{temp,n}) \\
&= \frac{2}{K_2\alpha}(\mathbf{\Delta w}_n^T\mathbf{E}^T\mathbf{E}\mathbf{w})\,. \qquad (4.9)
\end{aligned}
$$

The gradient vector $\nabla$ is a vertical stack of each component $\nabla_n$, and therefore

$$
\begin{aligned}
\nabla &= \begin{bmatrix} \nabla_1 \\ \cdots \\ \vdots \\ \cdots \\ \nabla_{AB} \end{bmatrix} = \frac{2}{K_2\alpha} \begin{bmatrix} \Delta\mathbf{w}_1^T \\ \cdots \\ \vdots \\ \cdots \\ \Delta\mathbf{w}_{AB}^T \end{bmatrix} \mathbf{E}^T\mathbf{E}\mathbf{w} \\
&= \frac{2}{K_2\alpha} \begin{bmatrix} \alpha & 0 & \cdots & 0 & 0 \\ 0 & \alpha & & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & & \alpha & 0 \\ 0 & 0 & \cdots & 0 & \alpha \end{bmatrix} \mathbf{E}^T\mathbf{E}\mathbf{w} \\
&= \frac{2}{K_2}\mathbf{E}^T\mathbf{E}\mathbf{w} \, . \qquad (4.10)
\end{aligned}
$$

From this equation, we can conclude that it does not matter whether weight $n$ was increased or decreased by $\alpha$.

We may finally advance to the weight update equation of the TECE method. Recall from Section 2.5 that a normalized algorithm allows the growth factor $\mu$ and the system input $x[n]$ to be independent. Also recall from (3.2) that the weights must be normalized in order for the transmit power to remain constant. The TECE method normalizes both the gradient and the weights in order to achieve both of the desired effects. However, in our linearized model, the weight normalization from (3.2) makes it difficult to analyze the effect of input from the channel estimate (which will be discussed in Section 4.2). Therefore, the weight update equation only involves the gradient normalization

$$\begin{aligned} \mathbf{w}[n+1] &= \mathbf{w}[n] - \mu \frac{\nabla[n]}{\sqrt{\nabla^T[n]\nabla[n]}} \\ &= \mathbf{w}[n] - \mu \mathbf{g}(\mathbf{w}[n]) \end{aligned} \tag{4.11}$$

where $\mathbf{g}$ is a nonlinear vector function of $\mathbf{w}[n]$.

Further analysis of (4.11) is difficult because it is nonlinear in its current form. For this reason, it is necessary to develop a small signal model [7, p. 324]. From (4.11), it follows that

$$\mathbf{w}_{OP,new} + \widetilde{\mathbf{w}}[n+1] = \mathbf{w}_{OP} + \widetilde{\mathbf{w}}[n] - \mu\left[\mathbf{g}(\mathbf{w}_{OP}) + \left(\frac{\partial \mathbf{g}}{\partial \mathbf{w}}\bigg|_{\mathbf{w}_{OP}}\right)\widetilde{\mathbf{w}}[n]\right] \tag{4.12}$$

where $\mathbf{w}_{OP}$ is the vector containing the weights' operating point (bias) and $\widetilde{\mathbf{w}}[n]$ is the small signal vector of the weights. The weight operating point $\mathbf{w}_{OP,new}$ that corresponds to $\widetilde{\mathbf{w}}[n+1]$ is

$$\mathbf{w}_{OP,new} = \mathbf{w}_{OP} - \mu \mathbf{g}(\mathbf{w}_{OP}) \ . \tag{4.13}$$

For this analysis, $\mathbf{w}_{OP}$ represents the optimal weight values $\mathbf{w}_{opt}$, and $\widetilde{\mathbf{w}}[n]$ is a small deviation in weight value from $\mathbf{w}_{opt}$. When $\mathbf{w}_{OP,new}$ is removed from both sides of (4.12), the remaining small signal equation is

$$\widetilde{\mathbf{w}}[n+1] = \widetilde{\mathbf{w}}[n] - \mu\left(\frac{\partial \mathbf{g}}{\partial \mathbf{w}}\bigg|_{\mathbf{w}_{OP}}\right)\widetilde{\mathbf{w}}[n] \ . \tag{4.14}$$

It is important to note that (4.14) can only be used to analyze the transient produced by very small weight disturbances. If the weight disturbance is on the order of $\mathbf{w}_{OP}$, then the linearized small signal approximation will not hold. The final step in solving

(4.14) is to determine the matrix $\frac{\partial \mathbf{g}}{\partial \mathbf{w}}\big|_{\mathbf{w}_{OP}}$ by calculating

$$\begin{aligned}
\mathbf{g}(\mathbf{w}) &= \mathbf{E}^T\mathbf{E}\big[\mathbf{w}\big(\mathbf{w}^T(\mathbf{E}^T\mathbf{E})^2\mathbf{w}\big)^{-\frac{1}{2}}\big] \\
\frac{\partial \mathbf{g}}{\partial \mathbf{w}} &= \mathbf{E}^T\mathbf{E}\Bigg[\big(\mathbf{w}^T(\mathbf{E}^T\mathbf{E})^2\mathbf{w}\big)^{-\frac{1}{2}}\mathbf{I} + \mathbf{w}\frac{\partial}{\partial \mathbf{w}}\Big(\big(\mathbf{w}^T(\mathbf{E}^T\mathbf{E})^2\mathbf{w}\big)^{-\frac{1}{2}}\Big)\Bigg] \\
&= \mathbf{E}^T\mathbf{E}\Bigg[\big(\mathbf{w}^T(\mathbf{E}^T\mathbf{E})^2\mathbf{w}\big)^{-\frac{1}{2}}\mathbf{I} + \mathbf{w}\Big(-\frac{1}{2}\big(\mathbf{w}^T(\mathbf{E}^T\mathbf{E})^2\mathbf{w}\big)^{-\frac{3}{2}}2\mathbf{w}^T(\mathbf{E}^T\mathbf{E})^2\Big)\Bigg] \\
&= \mathbf{E}^T\mathbf{E}\big(\mathbf{w}^T(\mathbf{E}^T\mathbf{E})^2\mathbf{w}\mathbf{I} - \mathbf{w}\mathbf{w}^T(\mathbf{E}^T\mathbf{E})^2\big)\big(\mathbf{w}^T(\mathbf{E}^T\mathbf{E})^2\mathbf{w}\big)^{-\frac{3}{2}} \\
\frac{\partial \mathbf{g}}{\partial \mathbf{w}}\bigg|_{\mathbf{w}_{OP}} &= \mathbf{F}\,. \qquad (4.15)
\end{aligned}$$

We may prove that $\mathbf{F}$ is symmetric, which will be important in Section 4.3. To start, we have in (4.15) the term $\big(\mathbf{w}^T(\mathbf{E}^T\mathbf{E})^2\mathbf{w}\big)^{-\frac{3}{2}}$, which evaluates to a scalar. We also have the term $\mathbf{w}^T(\mathbf{E}^T\mathbf{E})^2\mathbf{w}\mathbf{I}$, which evaluates to $\mathbf{I}$ multiplied by a scalar. The symmetry of $\mathbf{w}\mathbf{w}^T(\mathbf{E}^T\mathbf{E})^2$ can be proven since $\mathbf{w}$ is an eigenvector of $\mathbf{E}^T\mathbf{E}$, as shown in Section 2.9. With this, we have

$$\begin{aligned}
\mathbf{E}^T\mathbf{E}\mathbf{w} &= \lambda\mathbf{w} \\
\mathbf{w}^T\mathbf{E}^T\mathbf{E} &= \lambda\mathbf{w}^T \\
\mathbf{w}\mathbf{w}^T(\mathbf{E}^T\mathbf{E})^2 &= \lambda\mathbf{w}\mathbf{w}^T\mathbf{E}^T\mathbf{E} \\
&= \lambda^2\mathbf{w}\mathbf{w}^T\,, \qquad (4.16)
\end{aligned}$$

where $\lambda$ is the eigenvalue of $\mathbf{E}^T\mathbf{E}$ that corresponds to $\mathbf{w}$. To prove the symmetry of $\mathbf{F}$, we can show that

$$\begin{aligned}
&\mathbf{E}^T\mathbf{E}\big(\mathbf{w}^T(\mathbf{E}^T\mathbf{E})^2\mathbf{w}\mathbf{I} - \mathbf{w}\mathbf{w}^T(\mathbf{E}^T\mathbf{E})^2\big) \\
=\;& (\mathbf{w}^T(\mathbf{E}^T\mathbf{E})^2\mathbf{w})\mathbf{E}^T\mathbf{E} - \mathbf{E}^T\mathbf{E}\lambda^2\mathbf{w}\mathbf{w}^T \\
=\;& (\mathbf{w}^T(\mathbf{E}^T\mathbf{E})^2\mathbf{w})\mathbf{E}^T\mathbf{E} - \lambda^3\mathbf{w}\mathbf{w}^T\,, \qquad (4.17)
\end{aligned}$$

which is a symmetric matrix expression.

Applying the value of $\mathbf{F}$ to (4.14) allows one to simplify

$$\begin{aligned}\widetilde{\mathbf{w}}[n+1] &= \widetilde{\mathbf{w}}[n] - \mu\mathbf{F}\widetilde{\mathbf{w}}[n] \\ &= \left(\mathbf{I} - \mu\mathbf{F}\right)\widetilde{\mathbf{w}}[n] \\ &= \mathbf{A}\widetilde{\mathbf{w}}[n] \end{aligned} \tag{4.18}$$

into the familiar state space from of Section 2.1 where

$$\mathbf{A} = \mathbf{I} - \mu\mathbf{F}\,. \tag{4.19}$$

Given the symmetry of $\mathbf{F}$, it is clear that $\mathbf{A}$ is also symmetric.

With the linearized form of (4.18), we can apply the state space analysis from Section 2.1 to determine the small signal time response. This time response depicts the transient in $\widetilde{\mathbf{w}}$ due to a small disturbance in the weight vector, represented by the initial conditions $\widetilde{\mathbf{w}}[0]$. Similar to (2.8), the time response of this system is

$$\begin{aligned} z(\widetilde{\mathbf{w}}(z) - \widetilde{\mathbf{w}}[0]) &= \mathbf{A}\widetilde{\mathbf{w}}(z) \\ \widetilde{\mathbf{w}}(z) &= (\mathbf{I} - z^{-1}\mathbf{A})^{-1}\widetilde{\mathbf{w}}[0] \\ \widetilde{\mathbf{w}}[n] &= \mathbf{A}^n\widetilde{\mathbf{w}}[0]\,. \end{aligned} \tag{4.20}$$

In addition, we can use the analysis of (2.10) to determine whether or not the system will converge. The eigen-decomposition of $\mathbf{A}^n$ turns (4.20) into

$$\begin{aligned} \widetilde{\mathbf{w}}[n] &= \mathbf{V}_{(\mathbf{A})}^{-1}\,\mathbf{\Lambda}_{(\mathbf{A})}^{n}\,\mathbf{V}_{(\mathbf{A})}\,\widetilde{\mathbf{w}}[0] \\ &= \mathbf{V}_{(\mathbf{A})}^{-1}\,(\mathbf{I} - \mu\mathbf{\Lambda}_{(\mathbf{F})})^{n}\,\mathbf{V}_{(\mathbf{A})}\,\widetilde{\mathbf{w}}[0]\,. \end{aligned} \tag{4.21}$$

Similar to (2.23), we must have

$$0 \; < \; \mu < \frac{2}{|\lambda_{max,(\mathbf{F})}|} \tag{4.22}$$

in order for the small signal weights to converge back to the origin after being subject to $\widetilde{\mathbf{w}}[0]$.

## 4.2 System Response to an Input

The analysis of Section 4.1 resulted in a state space model that described the effect of a small disturbance in the weights on the small signal transient response. This disturbance will almost always be caused by an external source. Recall from Section 3.2 that systems involving a channel estimate $\mathbf{h}_{est}$ have the transmit-side beamforming algorithm decoupled from the receive-side system identification filter. The only parameter that connects these two systems is $\mathbf{h}_{est}$. In Section 3.4, we assumed that $\mathbf{h}_{est} = \mathbf{h}$. However, most of the noise and disturbance in the transmit-side system is due to inaccurate measurements of the channel from the receive-side filter. For this reason, it is important to find the effect of a small disturbance in the channel estimate $\widetilde{\mathbf{h}}_{est}$ on $\widetilde{\mathbf{w}}$.

In Section 4.1, we found $r_{pred}$ as a linear function of $\mathbf{w}$. Now, we must determine $r_{pred}$ as a linear function of $\mathbf{h}_{est}$. To begin, we rearrange the order of convolution in (4.2) to get

$$\mathbf{x} * \mathbf{w}_a * \mathbf{h}_{est,a} = han(\mathbf{x} * \mathbf{w}_a)\mathbf{h}_{est,a} \;, \tag{4.23}$$

which is a linear function of $\mathbf{h}_{est}$. This result can now be used in (4.1) to find

$$\begin{aligned}
\mathbf{r}_{pred} &= \sum_{a=1}^{A} \underbrace{han(\underset{(K_3\times B)}{han(\mathbf{x})}\ \underset{(B\times 1)}{\mathbf{w}_a})}_{(K_4\times C)}\underset{(C\times 1)}{\mathbf{h}_{est,a}} \\
&= \underbrace{\Big[ han(han(\mathbf{x})\mathbf{w}_1) \ \vdots\ \cdots\ \vdots\ han(han(\mathbf{x})\mathbf{w}_A) \Big]}_{(K_4\times AC)} \mathbf{h}_{est} \\
&= \mathbf{G}\mathbf{h}_{est}
\end{aligned} \tag{4.24}$$

where $K_3 = K + B - 1$, $K_4 = K_3 + C - 1 = K_2$, $\mathbf{G}$ is a temporary place holder matrix and $\mathbf{h}_{est}$ is an ordered vector that contains each tap of each channel estimate for every transmitter. Specifically,

$$\mathbf{h}_{est} = \underbrace{\begin{bmatrix} h_{est,1,1} & \cdots & h_{est,1,C} & h_{est,2,1} \cdots & h_{est,A,1} & \cdots h_{est,A,C} \end{bmatrix}^T}_{(AC\times 1)} . \tag{4.25}$$

In terms of these variables, the predicted average power at the receiver is

$$p_{pred} = \frac{1}{K_4}\mathbf{h}_{est}^T\mathbf{G}^T\mathbf{G}\mathbf{h}_{est} . \tag{4.26}$$

Calculating $\mathbf{r}_{temp}$ in terms of $\mathbf{h}_{est}$ is slightly more complicated than in Section 4.1. Instead of simply substituting $(\mathbf{w} + \mathbf{\Delta w})$ for $\mathbf{w}$ as was done in (4.8), we must calculate

$$\begin{aligned}
\mathbf{d}_{temp,n} &= \sum_{a=1}^{A} \underbrace{han(\underset{(K_3\times B)}{han(\mathbf{x})}(\underset{(B\times 1)}{\mathbf{w}_a} + \underset{(B\times 1)}{\Delta\mathbf{w}_{n,a}}))}_{(K_4\times C)}\underset{(C\times 1)}{\mathbf{h}_{est,a}} \\
&= \sum_{a=1}^{A} han\big(han(\mathbf{x})\mathbf{w}_a\big)\mathbf{h}_{est,a} + \sum_{a=1}^{A} han\big(han(\mathbf{x})\Delta\mathbf{w}_{n,a}\big)\mathbf{h}_{est,a} \\
&= \mathbf{G}\mathbf{h}_{est} + \mathbf{G}_{temp,n}\mathbf{h}_{est} ,
\end{aligned} \tag{4.27}$$

where $\mathbf{G}_{temp}$ is another placeholder matrix. From this, the temporary power that results from stepping one of the weights can be calculated in a way similar to (4.8). The temporary predicted power $p_{temp,n}$ due to stepping weight $n$ is

$$\begin{aligned}
p_{temp,n} &= \frac{1}{K_4}\mathbf{d}_{temp,n}^T\mathbf{d}_{temp,n} \\
&= \frac{1}{K_4}\mathbf{h}_{est}^T(\mathbf{G} + \mathbf{G}_{temp,n})^T(\mathbf{G} + \mathbf{G}_{temp,n})\mathbf{h}_{est} \\
&= \frac{1}{K_4}\mathbf{h}_{est}^T\mathbf{G}^T\mathbf{G}\mathbf{h}_{est} + 2\mathbf{h}_{est}^T\mathbf{G}^T\mathbf{G}_{temp,n}\mathbf{h}_{est} + \mathbf{h}_{est}^T\mathbf{G}_{temp,n}^T\mathbf{G}_{temp,n}\mathbf{h}_{est} \\
&\approx \frac{1}{K_4}\mathbf{h}_{est}^T\mathbf{G}^T\mathbf{G}\mathbf{h}_{est} + 2\mathbf{h}_{est}^T\mathbf{G}^T\mathbf{G}_{temp,n}\mathbf{h}_{est} .
\end{aligned} \tag{4.28}$$

The gradient follows as

$$\begin{aligned}\nabla_n &= \frac{1}{\alpha}(p_{pred} - p_{temp,n}) \\ &= \frac{2}{K_4\alpha}\mathbf{h}_{est}^T\mathbf{G}^T\mathbf{G}_{temp,n}\mathbf{h}_{est}\end{aligned} \tag{4.29}$$

$$\nabla = \frac{2}{K_4\alpha}\begin{bmatrix}\mathbf{h}_{est}^T\mathbf{G}^T\mathbf{G}_{temp,1}\mathbf{h}_{est} \\ \cdots \\ \vdots \\ \cdots \\ \mathbf{h}_{est}^T\mathbf{G}^T\mathbf{G}_{temp,AC}\mathbf{h}_{est}\end{bmatrix}. \tag{4.30}$$

This expression is more complicated than (4.10) because the terms of $\nabla$ are each a quadratic vector function of $\mathbf{h}_{est}$.

The same type of small signal linearization from (4.14) applies to this situation. In this case, however, $\mathbf{g}(\mathbf{w}[n], \mathbf{h}_{est}[n])$ is now a function of both $\mathbf{w}$ and $\mathbf{h}_{est}$. We must therefore add the $\mathbf{h}_{est}$ small signal term to (4.14), resulting in

$$\widetilde{\mathbf{w}}[n+1] = \widetilde{\mathbf{w}}[n] - \mu\left(\left.\frac{\partial \mathbf{g}}{\partial \mathbf{w}}\right|_{\mathbf{w}_{OP}}\right)\widetilde{\mathbf{w}}[n] - \mu\left(\left.\frac{\partial \mathbf{g}}{\partial \mathbf{h}_{est}}\right|_{\mathbf{w}_{OP}}\right)\widetilde{\mathbf{h}}_{est}[n]\ . \tag{4.31}$$

Differentiating $\mathbf{g}$ with respect to $\mathbf{h}_{est}$ can be performed using the chain rule as follows

$$\begin{aligned}
\mathbf{g}(\mathbf{h}_{est}[n]) &= \frac{\nabla[n]}{\sqrt{\nabla^T[n]\nabla[n]}} \\
\frac{\partial \mathbf{g}}{\partial \mathbf{h}_{est}} &= \frac{\partial \mathbf{g}}{\partial \nabla}\frac{\partial \nabla}{\partial \mathbf{h}_{est}} \\
\frac{\partial \mathbf{g}}{\partial \nabla} &= \left((\nabla^T\nabla)^{-\frac{1}{2}}\mathbf{I} - \nabla(\nabla^T\nabla)^{-\frac{3}{2}}\nabla^T\right)\frac{2}{\alpha} \\
\frac{\partial \nabla}{\partial \mathbf{h}_{est}} &= \begin{bmatrix} \mathbf{h}_{est}^T(\mathbf{G}^T\mathbf{G}_{temp,1} + \mathbf{G}_{temp,1}^T\mathbf{G}) \\ \cdots \\ \vdots \\ \cdots \\ \mathbf{h}_{est}^T(\mathbf{G}^T\mathbf{G}_{temp,AC} + \mathbf{G}_{temp,AC}^T\mathbf{G}) \end{bmatrix} .
\end{aligned} \tag{4.32}$$

These values can be calculated via matrix multiplication, but instead we will simply assign

$$\mathbf{B} = -\mu\frac{\partial \mathbf{g}}{\partial \mathbf{h}_{est}}\bigg|_{\mathbf{h}_{est,OP}} . \tag{4.33}$$

With $\mathbf{A}$ from (4.18), and $\mathbf{B}$ from (4.33), (4.31) now becomes

$$\widetilde{\mathbf{w}}[n+1] = \mathbf{A}\widetilde{\mathbf{w}}[n] + \mathbf{B}\widetilde{\mathbf{h}}_{est}[n] . \tag{4.34}$$

This is the familiar state space notation from (2.1) in which $\widetilde{\mathbf{h}}_{est}[n]$ takes the role of the system input. An input of $\widetilde{\mathbf{h}}_{est}[n]$ is appropriate because our model is meant to be used to analyze the effect of a small disturbance on the weights. The frequency and time domain analyses follows similar to (2.4) and (2.8) as [7, p. 289]

$$
\begin{aligned}
z(\widetilde{\mathbf{w}}(z) - \widetilde{\mathbf{w}}[0]) &= \mathbf{A}\widetilde{\mathbf{w}}(z) + \mathbf{B}\widetilde{\mathbf{h}}_{est}(z) \\
\widetilde{\mathbf{w}}(z) &= \left(\mathbf{I} - z^{-1}\mathbf{A}\right)^{-1}\widetilde{\mathbf{w}}[0] + \left(\mathbf{I} - z^{-1}\mathbf{A}\right)^{-1} z^{-1}\mathbf{H}\widetilde{\mathbf{h}}_{est}(z) \\
\widetilde{\mathbf{w}}[n] &= \mathbf{A}^{n}\widetilde{\mathbf{w}}[0] \sum_{m=0}^{n-1} \mathbf{A}^{m}\mathbf{H}\widetilde{\mathbf{h}}_{est}[n-1-m] \,.
\end{aligned} \tag{4.35}
$$

## 4.3 Analysis of Noise on the System Input

The main purpose of developing (4.34) is to analyze the effect of noise or other disturbances in $\widetilde{\mathbf{h}}_{est}$ on $\widetilde{\mathbf{w}}$. To simplify this analysis, we will define the transfer function $\mathbf{\Psi}(z)$ as

$$\begin{aligned}\widetilde{\mathbf{w}}(z) &= z^{-1}\mathbf{\Psi}(z)\widetilde{\mathbf{h}}_{est}(z) \\ \mathbf{\Psi}(z) &= \left(\mathbf{I} - z^{-1}\mathbf{A}\right)^{-1}\mathbf{B}\,. \end{aligned} \tag{4.36}$$

It will also be convenient to calculate the Hermitian of $\mathbf{\Psi}(z)$ as

$$\begin{aligned}\mathbf{\Psi}^H(z) &= \mathbf{B}^H\left(\mathbf{I} - (z^{-1})^{-1}\mathbf{A}^H\right)^{-1} \\ &= \mathbf{B}^T\left(\mathbf{I} - z\mathbf{A}^T\right)^{-1}, \end{aligned} \tag{4.37}$$

where $\mathbf{A}$ was calculated in (4.19) and $\mathbf{B}$ is from (4.33). Note that we assume $z = e^{j\omega}$. In other words, $z$ is on the unit circle, and therefore, $z^* = z^{-1}$.

To study the effect of noise on the system, we will assume that $\widetilde{\mathbf{h}}_{est}$ is a white noise process since it is characteristic of most types of system noise. It will also allow us to ignore the time delay $z^{-1}$ in the first line of (4.36), because time shifts generally have no effect on the outcome of a white noise process. We can use the properties of white noise processes to define the matrices

$$\begin{aligned}\mathbf{R}_{\mathbf{hh}}[n] &= \sigma^2\delta[n]\mathbf{I} \\ \mathbf{S}_{\mathbf{hh}}(z) &= \sigma^2\mathbf{I} \end{aligned} \tag{4.38}$$

where $\mathbf{R}_{\mathbf{hh}}[n]$ is the autocorrelation matrix of $\widetilde{\mathbf{h}}_{est}$, $\mathbf{S}_{\mathbf{hh}}(z)$ is the z-transform of $\mathbf{R}_{\mathbf{hh}}[n]$, and $\sigma^2$ is the noise variance of $\widetilde{\mathbf{h}}_{est}$. With this, $\mathbf{S}_{\mathbf{ww}}(z)$ can be calculated to be [12,

p. 329]

$$
\begin{aligned}
\mathbf{S}_{\mathbf{ww}}(z) &= \mathbf{\Psi}(z)\mathbf{S}_{\mathbf{hh}}\mathbf{\Psi}^H(z) \\
&= \sigma^2\mathbf{\Psi}(z)\mathbf{\Psi}^H(z) \\
&= \sigma^2\left(\mathbf{I} - z^{-1}\mathbf{A}\right)^{-1}\mathbf{B}\mathbf{B}^T\left(\mathbf{I} - z\mathbf{A}^T\right)^{-1} . \qquad (4.39)
\end{aligned}
$$

Instead of painstakingly extracting the noise variance in $\mathbf{w}$ from (4.39), we will do our analysis on the sum of the noise variances of $\mathbf{w}$ using a proof by Paul Fiore. It is known that a SISO system with frequency response $h(z)$, impulse response $h[n]$, and input noise variance $\sigma_h^2$ will have an output noise variance $\sigma_w^2$ equal to [12, p. 312-313]

$$
\begin{aligned}
\sigma_w^2 &= \sigma_h^2(h(z)h(z^{-1})) \\
&= \sigma_h^2(h[n] * h[-n]) . \qquad (4.40)
\end{aligned}
$$

In vector notation, this can be expressed as

$$
\begin{aligned}
\sigma_w^2 &= \mathbf{h}^H\mathbf{h}\sigma_h^2 \\
&= ||\mathbf{h}||^2\sigma_h^2 \qquad (4.41)
\end{aligned}
$$

where the elements of $\mathbf{h}$ are the taps of $h[n]$. This result is the sum of the squares of the taps of $\mathbf{h}$. In the case of MIMO, the noise variance can be derived from (4.36). For element $i$ of $\mathbf{w}$, this translates to

$$
\begin{aligned}
\widetilde{w}_i(z) &= \mathbf{\Psi}_i(z)\widetilde{\mathbf{h}}_{est}(z) \\
&= \sum_{j=1}^{N} \Psi_{ij}(z)\widetilde{h}_{est,j}(z) . \qquad (4.42)
\end{aligned}
$$

Note that $i$ is a row index and $j$ is a column index for $\mathbf{\Psi}$, but for $\widetilde{\mathbf{w}}$ and $\widetilde{\mathbf{h}}_{est}$, $i$ and

$j$ are the respective element indices. Another way of looking at (4.42) is that $\widetilde{\mathbf{w}}_i(z)$ is the sum of pushing each input $\widetilde{h}_{est,j}(z)$ through a transfer function $\Psi_{ij}(z)$. With this, we can use the procedure from (4.40) to analyze the effect of noise on the system from (4.36). Given that the noise variance for all inputs is $\sigma_h^2$, each tap of $\widetilde{\mathbf{w}}$ has a noise variance of

$$\begin{aligned} var(w_i) &= \sum_{j=1}^{N} \Psi_{ij}(z)\Psi_{ij}(z^{-1})var(h_{est,j}) \\ &= \sigma_h^2 \sum_{j=1}^{N} \Psi_{ij}(z)\Psi_{ij}(z^{-1}) . \end{aligned} \tag{4.43}$$

If we extend (4.41) to each transfer function $\Psi_{ij}(z)$, we get

$$var(w_i) = \sigma_h^2 \sum_{j=1}^{N} ||\Psi_{ij}(z)||^2 . \tag{4.44}$$

It follows that the sum of the elements in the noise variance vector $\sigma_{\mathbf{w}}^2$ can be expressed as

$$\begin{aligned} \mathbf{1}^T\sigma_{\mathbf{w}}^2 &= \sigma_h^2 \sum_{i=1}^{M}\sum_{j=1}^{N} ||\Psi_{ij}(z)||^2 \\ &= \sigma_h^2 ||\mathbf{\Psi}(z)||_F^2 \end{aligned} \tag{4.45}$$

where $\mathbf{1}$ is a vector of ones and $||\mathbf{\Psi}(z)||_F^2$ is the Frobenius norm of $\mathbf{\Psi}(z)$ [13, p. 56].

Now, we must solve for $||\mathbf{\Psi}(z)||_F^2$. It is important to first note that $\mathbf{\Psi}(z)$ can actually be expressed as the geometric sum

$$\mathbf{\Psi}(z) = (\mathbf{I} - z^{-1}\mathbf{A})^{-1}\mathbf{B} = \sum_{k=0}^{\infty} z^{-k}\mathbf{A}^k\mathbf{B} . \tag{4.46}$$

This allows us to express $||\mathbf{\Psi}(z)||_F^2$ as

$$||\mathbf{\Psi}(z)||_F^2 = \sum_{k=0}^{\infty} ||\mathbf{A}^k\mathbf{B}||_F^2 \tag{4.47}$$

After an eigenvalue decomposition, we get

$$\mathbf{A} = \mathbf{V\Lambda V}^{-1} . \tag{4.48}$$

Since we know from Section 4.1 that $\mathbf{A}$ is symmetric, we have

$$\begin{aligned} \mathbf{A} &= \mathbf{V\Lambda V}^H \\ ||\mathbf{\Psi}(z)||_F^2 &= \sum_{k=0}^{\infty} ||\mathbf{V\Lambda}^k\mathbf{V}^H\mathbf{B}||_F^2 . \end{aligned} \tag{4.49}$$

The Frobenius norm has two useful properties. The first is that orthonormal matrices like $\mathbf{V}$ on the outside of a product may be dropped. The second is that the Frobenius norm of matrix $\mathbf{A}$ is equal to the trace of $\mathbf{A}^H\mathbf{A}$. Applying these properties, we get

$$\begin{aligned} ||\mathbf{\Psi}(z)||_F^2 &= \sum_{k=0}^{\infty} ||\mathbf{\Lambda}^k\mathbf{VB}||_F^2 \\ &= \sum_{k=0}^{\infty} tr(\mathbf{B}^H\mathbf{V\Lambda}^{Hk}\mathbf{\Lambda}^k\mathbf{V}^H\mathbf{B}) . \end{aligned} \tag{4.50}$$

Further simplification can be made using the properties of the trace of a matrix. Elements in a trace can be rotated, in that [9, p. 301]

$$tr(\mathbf{XYZ}) = tr(\mathbf{YZX}) = tr(\mathbf{ZXY}) . \tag{4.51}$$

With this property, we get

$$||\mathbf{\Psi}(z)||_F^2 = \sum_{k=0}^{\infty} tr(\mathbf{\Lambda}^{Hk}\mathbf{\Lambda}^k\mathbf{V}^H\mathbf{B}\mathbf{B}^H\mathbf{V}) \,. \tag{4.52}$$

For simplicity, we will make the temporary matrix and vector assignments

$$\begin{aligned} \mathbf{M} &= \mathbf{\Lambda}^H\mathbf{\Lambda} \\ \mathbf{m} &= diag(\mathbf{M}) \\ \mathbf{N} &= \mathbf{V}^H\mathbf{B}\mathbf{B}^H\mathbf{V} \\ \mathbf{n} &= diag(\mathbf{N}) \end{aligned} \tag{4.53}$$

in which $diag(\cdots)$ is the Matlab diagonal function that returns a vector of the diagonal of a matrix. These assignments allow us to express (4.52) as

$$\begin{aligned} ||\mathbf{\Psi}(z)||_F^2 &= \sum_{k=0}^{\infty} tr(\mathbf{M}^k\mathbf{N}) \\ &= \sum_{k=0}^{\infty} \mathbf{m}^{Tk}\mathbf{n} \,. \end{aligned} \tag{4.54}$$

With this result, we now have

$$\begin{aligned} ||\mathbf{\Psi}(z)||_F^2 &= \sum_{k=0}^{\infty} \mathbf{m}^{Tk}\mathbf{n} \\ &= \sum_{k=0}^{\infty}\sum_{a=1}^{A} m_a^k n_a \\ &= \sum_{a=1}^{A}\left(n_a \sum_{k=0}^{\infty} m_a^k\right) \\ &= \sum_{a=1}^{A} \frac{n_a}{1-m_a} \,. \end{aligned} \tag{4.55}$$

These simplifications finally give us an expression that allows us to predict the noise variance on the sum of the weights in terms of the input noise variance. Combining (4.45) and (4.55) gives

$$\begin{aligned} \mathbf{1}^T \sigma^2_{\mathbf{w}} &= \sigma_h^2 ||\boldsymbol{\Psi}(z)||_F^2 \\ &= \sigma_h^2 \sum_{a=1}^{A} \frac{n_a}{1-m_a} . \end{aligned} \tag{4.56}$$

A simulation was created to test the accuracy of (4.55). Appendix A.2 gives the Matlab code for this simulation. To simulate the linearized TECE system, a white noise signal on $\widetilde{\mathbf{h}}_{est}$ is applied as the input to a state space system with random symmetrical positive definite $\mathbf{A}$ and $\mathbf{B}$. This type of input results in a semi-random set of state variables that represent $\mathbf{w}$ in this simulation. Unlike the input, the state variables are not white noise because the time response of a state space system, as shown in (2.8), is not necessarily time invariant.

Each trial of the simulation is run for 20000 samples. The variance of the weights and other such statistics are taken over the set of trial results that correspond to each sample. To test the accuracy of the noise analysis in this section, the sum of the variance of the weights is compared to the predicted value obtained from (4.55). The results, shown in Figure 4-1, clearly verify the prediction. Thus, (4.55) can be used to predict the effect of a noise input on the linearized TECE system.

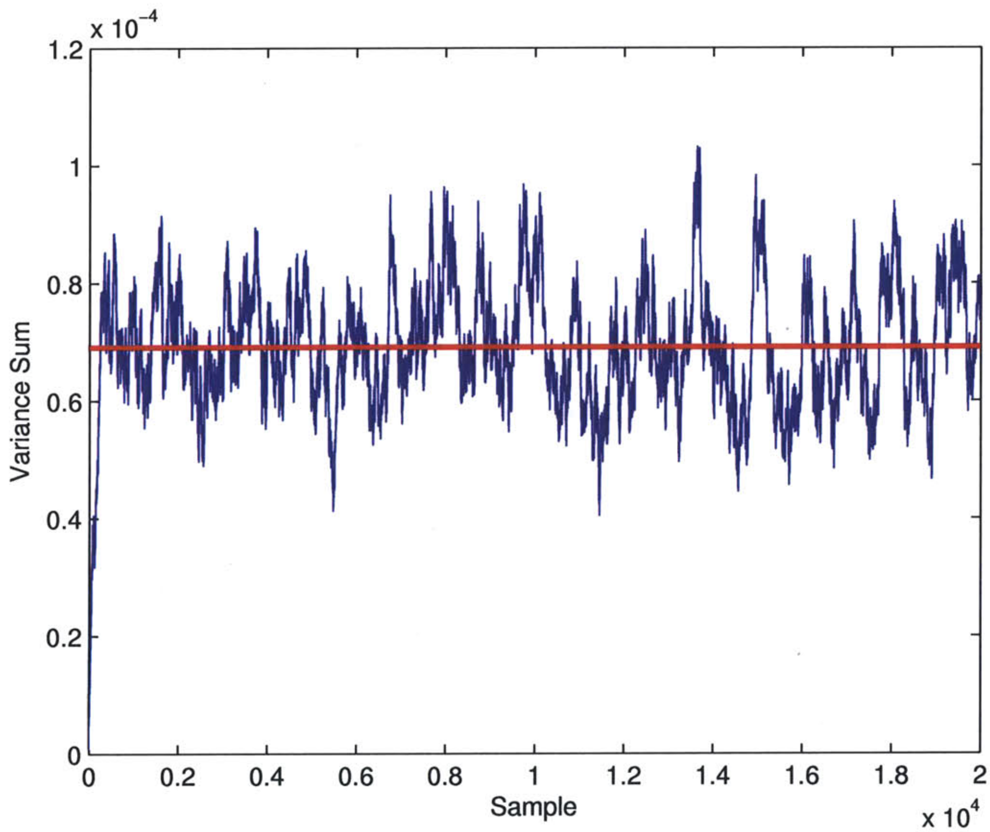


Figure 4-1: Simulation results that compares the sum of the variance of the weights (blue) to the predicted value (red) from (4.55). There were 100 trials, 20000 samples, 4 weights, 4 noise inputs, **A** and **B** are both normalized to 1 in Frobenius norm, and $\sigma_h = .001$.

# Chapter 5

# Conclusion

This thesis discussed three methods for STAR, and two methods of obtaining a channel estimate. Based on the performance results, the TECE method of Section 3.4 was determined to be the best for transmit-side beamforming. However, it is not clear which of the two channel estimation methods are better. The probing duty cycle method of Section 3.6 can achieve a very low relative received power of -30dB in a relatively short span of 1000 iterations. However, the orthogonality based probing scheme of Section 3.7 can probe continuously and thus produces results that are devoid of power spikes.

The mathematical analysis of the TECE method showed that it can be linearized and converted to state space form. The linearized form of this method is only valid if the weights deviate by a small amount from their optimal values. Such a model allows a noise analysis to be performed, and it is possible to predict the effect on the weights of noise in the channel estimate.

This method is to be integrated into a larger project at MIT Lincoln Laboratory. The ultimate goal is to build a system that is capable of STAR. Future work on this project will involve the construction of this system. The first step is to replace the simulated channel values with actual channel data. This will involve collecting realistic RF data and updating the Matlab scripts so that the length of the adaptive filters can more realistically match the length of the channel. The Matlab code for the simulations of this thesis will then be implemented with an FPGA. Finally, the analog system must be constructed, which includes the antenna array, RF canceller, and all of the RF analog electronics.

Successful completion of a STAR system will ultimately provide for technological expansion in the field of communications. Full duplex systems are useful for any application that requires simultaneous communication. Any product that includes a two-way radio can be improved with the ability to simultaneously transmit and receive.

# Appendix A

# Source Code

## A.1 Matlab Code for Trial and Error Method

```
close all
samples = 30000;
transmitters = 2;
taps = 4; %number of taps in the transmit side lms filter
averaging = 100; %length of moving average and convolution (this ...
    is 'K')
update_freq = 50; %how often the weights are updated

t = 1:1:samples;
t2 = 1:1:samples/2;
t3 = samples/2:1:samples;
% ref = sin(.5*t);
ref = sin((.1 + .05*sin(.065*t)).*t); %reference signal
soi = zeros(1, samples);
% soi = [zeros(1, samples/2), .1*sin(.5*t3)];

w = [1 0 0 0; 1 0 0 0]; %weights
grad = ones(transmitters, taps); %gradients
w_change = w; %weight change
u = .1; %weight update growth factor (this is '\mu')
w_power = sum(w(:).^2); %constant weight magnitude level
w_power_old = w_power;
p_old = 0;
w_counter = 1;
noise_factor_tx = 0; %noise added to transmitter
noise_factor_d = 0; %noise added to receiver
% noise_factor_tx = .001; %noise added to transmitter
% noise_factor_d = .01; %noise added to receiver

tx = zeros(transmitters, samples); %what gets transmitted
for a = 1:1:transmitters
    tx(a,1:averaging) = ref(1:averaging);
end

```

```
h_true = [8 -2 3 1; 7 -3 2 -1]; %the channel
h_true = h_true/sqrt(sum(h_true(:).^2));
d_true_part = zeros(transmitters, samples + length(h_true(1,:)));
d_true = zeros(1, samples); %received signal (this is 'r')
h_change_mag = .1; %channel change magnitude
h_change_mod = 4000; %channel change frequency
h_change_factor = 0; %channel change slope (change per cycle)

w_track = ones(numel(w), samples - averaging); %weights
h_track_true = ones(numel(h_true), samples - averaging); %true ...
    channel
t_track = zeros(transmitters, samples - averaging); %transmission ...
    signal
tp_track = zeros(1, samples - averaging); %transmitted power
tp_track_avg = zeros(1, samples - 2*averaging);
dp_track = zeros(1, samples - averaging); %received power
dp_track_avg = zeros(1, samples - 2*averaging);

cput = cputime;
%step through all time
for n = (averaging + 1):1:(samples - averaging - 1)
%change channel if necessary
    if(mod(n, h_change_mod) == 0)
        h_change_factor = ...
            h_change_mag*randn(size(h_true))/h_change_mod;
    end
    h_true = h_true + h_change_factor;
    h_true = h_true/sqrt(sum(h_true(:).^2));
    d_true_temp = 0;
    d_est = 0;
    for a = 1:1:transmitters
%transmit signal for an antenna
        tx_temp = conv(ref(n - averaging:n), w(a,:));
        tx(a,n) = tx_temp(averaging + 1) + noise_factor_tx*randn;
%update true received signal
        d_true_part(a,1:n+length(h_true(a,:))-1) = ...
            conv(h_true(a,:), tx(a,1:n)) + noise_factor_d*randn;
        d_true_temp = d_true_temp + d_true_part(a,:); %receive signal
    end
    d_true(n) = d_true_temp(n) + soi(n);
%update true and estimated received power
    p_true = d_true(n - averaging:n)*d_true(n - averaging:n)'; ...
        %true power
%go through every tap on every transmitter and step weight and ...
    redo calcs
    if(mod(n, update_freq) == 0)
        if(p_old < p_true)
            w = w/sqrt(w_power/w_power_old);
            w(w_counter) = w(w_counter) - 2*u*w_power;
            w = w*sqrt(w_power/sum(w(:).^2));
        end
        p_old = p_true;
        w_counter = w_counter + 1;
        if(w_counter > length(w(:)))
```

```
                w_counter = 1;
            end
            w(w_counter) = w(w_counter) + u*w_power;
            w_power_old = sum(w(:).^2);
            w = w*sqrt(w_power/sum(w(:).^2));
        end
%tracking
        for a = 1:1:transmitters
            t_track(a,n) = tx(a,n);
        end
        h_track_true(:,n) = h_true(:);
        w_track(:,n) = w(:);
        tp_track(n) = sum(w(:).^2);
        d_track(n) = d_true(end);
        dp_track(n) = p_true/averaging;
        if(n > averaging)
            tp_track_avg(n) = mean(tp_track(n - averaging:n));
            dp_track_avg(n) = mean(dp_track(n - averaging:n));
        end
end %for n = (averaging + 1):1:(samples - averaging - 1)
cputime - cput

figure
plot(10*log10(dp_track_avg(round(find(dp_track_avg == ...
    max(dp_track_avg))*.75):(samples - averaging - 1))))
xlabel('Time (samples)')
ylabel('Power (dB, relative to transmitter)')

figure
plot(w_track')
% legend('w(1,1)', 'w(2,1)', 'w(1,2)', 'w(2,2)',...
%     'w(1,3)', 'w(2,3)', 'w(1,4)', 'w(2,4)');
ax = axis;
axis([(averaging + taps + 1), (samples - averaging - 1), ax(3), ...
    ax(4)]);
xlabel('Time (samples)')
ylabel('Weight Value')

figure
plot(h_track_true')
title('True Channel')
% legend('h(1,1)', 'h(2,1)', 'h(1,2)', 'h(2,2)',...
%     'h(1,3)', 'h(2,3)', 'h(1,4)', 'h(2,4)');
xlabel('Time (samples)')
ylabel('Tap Gain Value')
```

## A.2 Matlab Code for Gradient Descent Method

```
close all
samples = 30000;
transmitters = 2;
taps = 4; %number of taps in the transmit side lms filter
averaging = 100; %length of moving average and convolution (this ...
    is 'K')

t = 1:1:samples;
% ref = sin(.5*t);
ref = sin((.5 + .1*sin(.065*t)).*t); %reference signal

w = [1 0 0 0; 1 0 0 0]; %weights
grad = ones(transmitters, taps); %gradients
w_change = w; %weight change
u2 = .005; %weight update growth factor (this is '\mu')
w_power = sum(w(:).^2); %constant weight magnitude level
% noise_factor_tx = .001; %noise added to transmitter
% noise_factor_d = .01; %noise added to receiver
noise_factor_tx = 0; %noise added to transmitter
noise_factor_d = 0; %noise added to receiver

tx = zeros(transmitters, samples); %what gets transmitted
for a = 1:1:transmitters
    tx(a,1:averaging) = ref(1:averaging);
end
u3 = 1.5; %receive side LMS growth factor (this is '\mu_{Rx}')

h_true = [8 -2 3 1; 7 -3 2 -1]; %the channel
h_true = h_true/sqrt(sum(h_true(:).^2));
h_est = h_true; %assume channel estimate perfectly models channel
taps_rlms = size(h_est, 2); %number of taps in the receiver lms ...
    filter
h_ref = zeros(transmitters, length(conv(h_est(1,:),...
    linspace(1,1,averaging + 1)))); %appropriately sized conv holder
d_true_part = zeros(transmitters, length(conv(conv(h_true(1,:),...
    linspace(1,1,averaging + 1)), linspace(1,1,taps))));
d_true = zeros(1, samples); %received signal (this is 'r')
h_change_mag = .02; %channel change magnitude
h_change_mod = 4000; %channel change frequency
h_change_factor = 0; %channel change slope (change per cycle)

w_track = ones(numel(w), samples - averaging); %weights
w_track_rlms = ones(numel(h_est), samples - averaging); %h_est
h_track_true = ones(numel(h_true), samples - averaging); %true ...
    channel
t_track = zeros(transmitters, samples - averaging); %transmission ...
    signal
tp_track = zeros(1, samples - averaging); %transmitted power
tp_track_avg = zeros(1, samples - 2*averaging);
dp_track = zeros(1, samples - averaging); %received power
```

```
dp_track_avg = zeros(1, samples - 2*averaging);

cput = cputime;
%step through all time
for n = (averaging + 1):1:(samples - averaging - 1)
%change channel if necessary
    if(mod(n, h_change_mod) == 0)
        h_change_factor = ...
            h_change_mag*randn(size(h_true))/h_change_mod;
    end
    h_true = h_true + h_change_factor;
    h_true = h_true/sqrt(sum(h_true(:).^2));
    d_true_temp = 0;
    d_est = 0;
    for a = 1:1:transmitters
%transmit signal for an antenna
        tx_temp = conv(ref(n - averaging:n), w(a,:));
        tx(a,n) = tx_temp(averaging + 1) + noise_factor_tx*randn;
%update true received signal
        d_true_part(a,1:n+taps-1) = conv(h_true(a,:), tx(a,1:n))...
            + noise_factor_d*randn;
        d_true_temp = d_true_temp + d_true_part(a,:); %receive signal
    end
    d_true(n) = d_true_temp(n);
    for a = 1:1:transmitters
%update receive side LMS filter, from Paul Fiore's lmsmeth.m
        xvec = flipud(tx(a,n - taps_rlms + 1:n)'); %vector
        normlms = xvec'*xvec; %scalar
        y_rlms = h_est(a,:)*xvec; %scalar
        err_rlms = d_true_part(a,n) - y_rlms;
        h_est(a,:) = h_est(a,:) + u3*conj(err_rlms)*xvec'/normlms;
%update estimated received signal
        h_ref(a,:) = conv(h_est(a,:), ref(n - averaging:n));
        d_est = d_est + conv(h_ref(a,:), w(a,:)); %receive signal ...
            estimate
    end %for a = 1:1:transmitters
%update true and estimated received power
    p_true = d_true(n - averaging:n)*d_true(n - averaging:n)'; ...
        %true power
    p_est = d_est*d_est'; %the estimated receive power
%go through every tap on every transmitter and step weight and ...
    redo calcs
    for a = 1:1:transmitters
        for x = 1:1:taps
            grad(a,x) = 2*sum(d_est(1:averaging + 1)* ...
                h_ref(a,averaging + 1 - (x - 1)));
        end %for x = 1:1:taps
    end %for a = 1:1:transmitters
%update transmit side weights and normalize them
    if(sum(grad(:)) ≠ 0)
        w_change = u2*grad/sqrt(sum(grad(:).^2));
        w = w - w_change;
    end
    w = w*sqrt(w_power/sum(w(:).^2));
```

```
%tracking
    for a = 1:1:transmitters
        t_track(a,n) = tx(a,n);
    end
    h_track_true(:,n) = h_true(:);
    w_track_rlms(:,n) = h_est(:);
    w_track(:,n) = w(:);
    tp_track(n) = sum(w(:).^2);
    dp_track(n) = p_true/averaging;
    if(n > averaging)
        tp_track_avg(n) = mean(tp_track(n - averaging:n));
        dp_track_avg(n) = mean(dp_track(n - averaging:n));
    end
end %for n = (averaging + 1):1:(samples - averaging - 1)
cputime - cput

figure
plot(10*log10(dp_track_avg(round(find(dp_track_avg == ...
    max(dp_track_avg))*.75):(samples - averaging - 1))))
xlabel('Time (samples)')
ylabel('Power (dB, relative to transmitter)')

figure
plot(w_track')
% legend('w(1,1)', 'w(2,1)', 'w(1,2)', 'w(2,2)',...
%     'w(1,3)', 'w(2,3)', 'w(1,4)', 'w(2,4)');
ax = axis;
axis([(averaging + taps + 1), (samples - averaging - 1), ax(3), ...
    ax(4)]);
xlabel('Time (samples)')
ylabel('Weight Value')

figure
subplot(1,2,1);
plot(w_track_rlms')
title('Receive Side Weights');
% legend('w(1,1)', 'w(2,1)', 'w(1,2)', 'w(2,2)',...
%     'w(1,3)', 'w(2,3)', 'w(1,4)', 'w(2,4)');
xlabel('Time (samples)')
ylabel('Weight Value')
ax = axis;
axis([(averaging + taps + 1), (samples - averaging - 1), ax(3), ...
    ax(4)]);
ax = axis;
subplot(1,2,2);
plot(h_track_true')
title('True Channel')
% legend('h(1,1)', 'h(2,1)', 'h(1,2)', 'h(2,2)',...
%     'h(1,3)', 'h(2,3)', 'h(1,4)', 'h(2,4)');
xlabel('Time (samples)')
ylabel('Channel Tap Value')
axis(ax);
```

## A.3 Matlab Code for Trial and Error Method with a Channel Estimate

```
close all
samples = 30000;
transmitters = 2;
taps = 4; %number of taps in the transmit side lms filter
averaging = 100; %length of moving average and convolution (this ...
    is 'K')

t = 1:1:samples;
t2 = 1:1:samples/2;
t3 = samples/2:1:samples;
ref = sin((.5 + .1*sin(.065*t)).*t); %reference signal

w = [1 0 0 0; 1 0 0 0]; %weights
grad = ones(transmitters, taps); %gradients
w_change = w; %weight change
u1 = .001; %weight step factor for gradient (this is '\alpha')
u2 = .001; %weight update growth factor (this is '\mu')
w_power = sum(w(:).^2); %constant weight magnitude level
% noise_factor_tx = .001; %noise added to transmitter
% noise_factor_d = .01; %noise added to receiver
noise_factor_tx = 0; %noise added to transmitter
noise_factor_d = 0; %noise added to receiver

tx = zeros(transmitters, samples); %what gets transmitted
for a = 1:1:transmitters
    tx(a,1:averaging) = ref(1:averaging);
end
u3 = .1; %receive side LMS growth factor (this is '\mu_{Rx}')

h_true = [8 -2 3 1; 7 -3 2 -1]; %the channel
h_true = h_true/sqrt(sum(h_true(:).^2));
h_est = h_true; %assume channel estimate perfectly models channel
taps_rlms = size(h_est, 2); %number of taps in the receiver lms ...
    filter
h_ref = zeros(transmitters, taps_rlms + averaging); %temp conv holder
d_true_part = zeros(transmitters, samples + length(h_true(1,:)));
d_true = zeros(1, samples);
h_change_mag = .02; %channel change magnitude
h_change_mod = 4000; %channel change frequency
h_change_factor = 0; %channel change slope (change per cycle)

w_track = ones(numel(w), samples - averaging); %weights
w_track_rlms = ones(numel(h_est), samples - averaging); %h_est
h_track_true = ones(numel(h_true), samples - averaging); %true ...
    channel
t_track = zeros(transmitters, samples - averaging); %transmission ...
    signal
tp_track = zeros(1, samples - averaging); %transmitted power
```

```
tp_track_avg = zeros(1, samples - 2*averaging);
dp_track = zeros(1, samples - averaging); %received power
dp_track_avg = zeros(1, samples - 2*averaging);

cput = cputime;
%step through all time
for n = (averaging + 1):1:(samples - averaging - 1)
%change channel if necessary
    if(mod(n, h_change_mod) == 0)
        h_change_factor = ...
            h_change_mag*randn(size(h_true))/h_change_mod;
    end
    h_true = h_true + h_change_factor;
    h_true = h_true/sqrt(sum(h_true(:).^2));
    d_true_temp = 0;
    d_est = 0;
    for a = 1:1:transmitters
%transmit signal for an antenna
        tx_temp = conv(ref(n - averaging:n), w(a,:));
        tx(a,n) = tx_temp(averaging + 1) + noise_factor_tx*randn;
%update true received signal
        d_true_part(a,1:n+length(h_true(a,:))-1) = ...
            conv(h_true(a,:), tx(a,1:n)) + noise_factor_d*randn;
        d_true_temp = d_true_temp + d_true_part(a,:); %receive signal
    end
    d_true(n) = d_true_temp(n) + soi(n);
    for a = 1:1:transmitters
%update receive side LMS filter, from Paul Fiore's lmsmeth.m
        xvec = flipud(tx(a,n - taps_rlms + 1:n)'); %vector
        normlms = xvec'*xvec; %scalar
        y_rlms = h_est(a,:)*xvec; %scalar
        err_rlms = d_true_part(a,n) - y_rlms;
        h_est(a,:) = h_est(a,:) + u3*conj(err_rlms)*xvec'/normlms;
%update estimated received signal
        h_ref(a,:) = conv(h_est(a,:), ref(n - averaging:n));
        d_est = d_est + conv(h_ref(a,:), w(a,:)); %receive signal ...
            estimate
    end %for a = 1:1:transmitters
%update true and estimated received power
    p_true = d_true(n - averaging:n)*d_true(n - averaging:n)'; ...
        %true power
    p_est = d_est*d_est'; %the estimated receive power
%go through every tap on every transmitter and step weight and ...
    redo calcs
    for a = 1:1:transmitters
        for x = 1:1:taps
%step weight up
            w_temp1 = w;
            w_temp1(a,x) = w(a,x) + u1*w_power;
            d_temp1 = 0;
            for aa = 1:1:transmitters
                d_temp1 = d_temp1 + conv(h_ref(aa,:), w_temp1(aa,:));
            end
            p_temp1 = d_temp1*d_temp1';
```

```
%step weight down
            w_temp2 = w;
            w_temp2(a,x) = w(a,x) - u1*w_power;
            d_temp2 = 0;
            for aa = 1:1:transmitters
                d_temp2 = d_temp2 + conv(h_ref(aa,:), w_temp2(aa,:));
            end
            p_temp2 = d_temp2*d_temp2';
%find minimum power and set gradient
            if(p_temp1 < p_temp2)
                grad(a,x) = (p_temp1 - p_est)/(u1*w_power);
            else
                grad(a,x) = (p_temp2 - p_est)/(-u1*w_power);
            end
        end %for x = 1:1:taps
    end %for a = 1:1:transmitters

%update transmit side weights and normalize them
    if(sum(grad(:)) ≠ 0)
        w_change = u2*grad/sqrt(sum(grad(:).^2));
        w = w - w_change;
    end
    w = w*sqrt(w_power/sum(w(:).^2));
%tracking
    for a = 1:1:transmitters
        t_track(a,n) = tx(a,n);
    end
    h_track_true(:,n) = h_true(:);
    w_track_rlms(:,n) = h_est(:);
    w_track(:,n) = w(:);
    tp_track(n) = sum(w(:).^2);
    dp_track(n) = p_true/averaging;
    if(n > averaging)
        tp_track_avg(n) = mean(tp_track(n - averaging:n));
        dp_track_avg(n) = mean(dp_track(n - averaging:n));
    end
end %for n = (averaging + 1):1:(samples - averaging - 1)
cputime - cput

figure
plot(10*log10(dp_track_avg(round(find(dp_track_avg == ...
    max(dp_track_avg))*.75):(samples - averaging - 1))))
xlabel('Time (samples)')
ylabel('Power (dB, relative to transmitter)')

figure
plot(w_track')
% legend('w(1,1)', 'w(2,1)', 'w(1,2)', 'w(2,2)',...
%     'w(1,3)', 'w(2,3)', 'w(1,4)', 'w(2,4)');
ax = axis;
axis([(averaging + taps + 1), (samples - averaging - 1), ax(3), ...
    ax(4)]);
xlabel('Time (samples)')
ylabel('Weight Value')
```

```
figure
subplot(1,2,1);
plot(w_track_rlms')
title('Receive Side Weights');
% legend('w(1,1)', 'w(2,1)', 'w(1,2)', 'w(2,2)',...
%      'w(1,3)', 'w(2,3)', 'w(1,4)', 'w(2,4)');
xlabel('Time (samples)')
ylabel('Weight Value')
ax = axis;
axis([(averaging + taps + 1), (samples - averaging - 1), ax(3), ...
    ax(4)]);
ax = axis;
subplot(1,2,2);
plot(h_track_true')
title('True Channel')
% legend('h(1,1)', 'h(2,1)', 'h(1,2)', 'h(2,2)',...
%      'h(1,3)', 'h(2,3)', 'h(1,4)', 'h(2,4)');
xlabel('Time (samples)')
ylabel('Channel Tap Value')
axis(ax);
```

# A.4 Matlab Code for Probing Duty Cycle Method

```
close all
samples = 30000;
transmitters = 2;
taps = 4; %number of taps in the transmit side lms filter
averaging = 100; %length of moving average and convolution (this ...
    is 'K')

t = 1:1:samples;
t2 = 1:1:samples/2;
t3 = samples/2:1:samples;
% ref = sin(.5*t);
ref = sin((.5 + .1*sin(.065*t)).*t); %reference signal
soi = zeros(1, samples);
% soi = [zeros(1, samples/2), .5*sin(.01*t3)]; %signal of interest

w = [1 0 0 0; 1 0 0 0]; %weights
grad = ones(transmitters, taps); %gradients
w_change = w; %weight change
u1 = .005; %weight step factor for gradient (this is '\alpha')
u2 = .005; %weight update growth factor (this is '\mu')
w_power = sum(w(:).^2); %constant weight magnitude level
% noise_factor_tx = .001; %noise added to transmitter
% noise_factor_d = .01; %noise added to receiver
noise_factor_tx = 0; %noise added to transmitter
noise_factor_d = 0; %noise added to receiver

tx = zeros(transmitters, samples); %what gets transmitted
for a = 1:1:transmitters
    tx(a,1:averaging) = ref(1:averaging);
end
% rlms_bd = 0; %receive side lms bulk delay
u3 = .05; %receive side LMS growth factor (this is '\mu_{Rx}')
probe_period = 150; %how often the system probes itself
probe_time = 15; %how long the system probes itself for each time
probe_mag = .1; %amplitude of probing signal

h_true = [8 -2 3 1; 7 -3 2 -1]; %the channel
h_true = h_true/sqrt(sum(h_true(:).^2));
h_est = h_true; %assume channel estimate perfectly models channel
taps_rlms = size(h_est, 2); %number of taps in the receiver lms ...
    filter
h_ref = zeros(transmitters, taps_rlms + averaging); %temp conv holder
d_true_part = zeros(transmitters, samples + length(h_true(1,:)));
d_true = zeros(1, samples); %received signal (this is 'r')
h_change_mag = .1; %channel change magnitude
h_change_mod = 9000; %channel change frequency
h_change_factor = 0; %channel change slope (change per cycle)

w_track = ones(numel(w), samples - averaging); %weights
w_track_rlms = ones(numel(h_est), samples - averaging); %h_est
```

```
h_track_true = ones(numel(h_true), samples - averaging); %true ...
    channel
t_track = zeros(transmitters, samples - averaging); %transmission ...
    signal
tp_track = zeros(1, samples - averaging); %transmitted power
tp_track_avg = zeros(1, samples - 2*averaging);
dp_track = zeros(1, samples - averaging); %received power
dp_track_avg = zeros(1, samples - 2*averaging);

cput = cputime;
%step through all time
for n = (averaging + 1):1:(samples - averaging - 1)
%change channel if necessary
    if(mod(n, h_change_mod) == 0)
        h_change_factor = ...
            h_change_mag*randn(size(h_true))/h_change_mod;
    end
    h_true = h_true + h_change_factor;
    h_true = h_true/sqrt(sum(h_true(:).^2));
    d_true_temp = 0;
    d_est = 0;
    probe_number = floor(mod(n, probe_period)/probe_time) + 1;
    probe_number2 = mod(mod(n, probe_period), probe_time);
    for a = 1:1:transmitters
%transmit signal for an antenna
        if(probe_number > transmitters)
            tx_temp = conv(ref(n - averaging:n), w(a,:));
            tx(a,n) = tx_temp(averaging + 1) + noise_factor_tx*randn;
        else if (probe_number == a)
                tx(a,n) = probe_mag*ref(n);
            else
                tx(a,n) = 0;
            end
        end
%update true received signal
        d_true_part(a,1:n+length(h_true(a,:))-1) = ...
            conv(h_true(a,:), tx(a,1:n)) + noise_factor_d*randn;
        d_true_temp = d_true_temp + d_true_part(a,:); %receive signal
    end
    d_true(n) = d_true_temp(n);
%update receive side LMS filter, from Paul Fiore's lmsmeth.m
    for a = 1:1:transmitters
        if((probe_number == a) && (probe_number2 > 2*taps_rlms))
            xvec = flipud(tx(a,n - taps_rlms + 1:n)'); %vector
            normlms = xvec'*xvec; %scalar
            y_rlms = h_est(a,:)*xvec; %scalar
            err_rlms = d_true(n) - y_rlms;
            h_est(a,:) = h_est(a,:) + ...
                u3*conj(err_rlms)*xvec'/normlms;
        end
    end
%update estimated received signal
    for a = 1:1:transmitters
        h_ref(a,:) = conv(h_est(a,:), ref(n - averaging:n));
```

```
        d_est = d_est + conv(h_ref(a,:), w(a,:)); %receive signal ...
            estimate
    end %for a = 1:1:transmitters
%update true and estimated received power
    p_true = d_true(n - averaging:n)*d_true(n - averaging:n)'; ...
        %true power
    p_est = d_est*d_est'; %the estimated receive power
%go through every tap on every transmitter and step weight and ...
    redo calcs
    if(probe_number > transmitters)
        for a = 1:1:transmitters
            for x = 1:1:taps
%step weight up
                w_temp1 = w;
                w_temp1(a,x) = w(a,x) + u1*w_power;
                d_temp1 = 0;
                for aa = 1:1:transmitters
                    d_temp1 = d_temp1 + conv(h_ref(aa,:), ...
                        w_temp1(aa,:));
                end
                p_temp1 = d_temp1*d_temp1';
%step weight down
                w_temp2 = w;
                w_temp2(a,x) = w(a,x) - u1*w_power;
                d_temp2 = 0;
                for aa = 1:1:transmitters
                    d_temp2 = d_temp2 + conv(h_ref(aa,:), ...
                        w_temp2(aa,:));
                end
                p_temp2 = d_temp2*d_temp2';
%find minimum power and set gradient
                if(p_temp1 > p_est && p_temp2 > p_est)
                    grad(a,x) = 0;
                else
                    if(p_temp1 < p_temp2)
                        grad(a,x) = (p_temp1 - p_est)/(u1*w_power);
                    else
                        grad(a,x) = (p_temp2 - p_est)/(-u1*w_power);
                    end
                end
            end %for x = 1:1:taps
        end %for a = 1:1:transmitters
%update transmit side weights and normalize them
        if(sum(grad(:)) ≠ 0)
            w_change = u2*grad/sqrt(sum(grad(:).^2));
            w = w - w_change;
        end
        w = w*sqrt(w_power/sum(w(:).^2));
    end
%tracking
    for a = 1:1:transmitters
        t_track(a,n) = tx(a,n);
    end
    h_track_true(:,n) = h_true(:);
```

```
    w_track_rlms(:,n) = h_est(:);
    w_track(:,n) = w(:);
    tp_track(n) = sum(w(:).^2);
    d_track(n) = d_true(end);
    dp_track(n) = p_true/averaging;
    if(n > averaging)
        tp_track_avg(n) = mean(tp_track(n - averaging:n));
        dp_track_avg(n) = mean(dp_track(n - averaging:n));
    end
end %for n = (averaging + 1):1:(samples - averaging - 1)
cputime - cput

figure
plot(10*log10(dp_track_avg(round(find(dp_track_avg == ...
    max(dp_track_avg))*.75):(samples - averaging - 1))))
xlabel('Time (samples)')
ylabel('Power (dB, relative to transmitter)')

figure
hold on
plot(w_track')
% legend('w(1,1)', 'w(2,1)', 'w(1,2)', 'w(2,2)',...
%     'w(1,3)', 'w(2,3)', 'w(1,4)', 'w(2,4)');
xlabel('Time (samples)')
ylabel('Weight Value')

figure
subplot(1,2,1);
hold on
plot(w_track_rlms')
title('Receive Side Weights');
% legend('w(1,1)', 'w(2,1)', 'w(1,2)', 'w(2,2)',...
%     'w(1,3)', 'w(2,3)', 'w(1,4)', 'w(2,4)');
xlabel('Time (samples)')
ylabel('Weight Value')
subplot(1,2,2);
hold on
plot(h_track_true')
title('True Channel')
% legend('h(1,1)', 'h(2,1)', 'h(1,2)', 'h(2,2)',...
%     'h(1,3)', 'h(2,3)', 'h(1,4)', 'h(2,4)');
xlabel('Time (samples)')
ylabel('Channel Tap Value')
```

# A.5 Matlab Code for Orthogonality Based Probing Method

```
%transmit beamforming test

close all
clear all

samples = 50000;
soi_mag = .1; %amplitude of the signal of interest
h_true = [8 -2 3 1; 7 -3 2 -1]; %the channel
h_change_mag = .1; %channel change magnitude
h_change_mod = 18000; %channel change frequency
% noise_factor_tx = .001; %noise added to transmitter
% noise_factor_d = .01; %noise added to receiver
noise_factor_tx = 0; %noise added to transmitter
noise_factor_d = 0; %noise added to receiver

transmitters = 2;
taps = 4; %number of taps in the transmit side lms filter
averaging = 200; %length of moving average and convolution
u1 = .001; %weight step factor for gradient
u2 = .001; %weight update growth factor
taps_rlms = 4; %number of taps in the receiver lms filter
averaging_rlms = 5000; %receive side LMS averager length
u3 = 1.5; %receive side LMS growth factor
probe_mag = .05; %amplitude of the probe signal
ref_mag = 1; %amplitude of the reference signal

t = 1:1:samples;
% ref_tx = ref_mag*sin(.5*t);
ref_tx = ref_mag*sin((.01 + .009*sin(t)).*t); %reference signal
ref_probe = zeros(transmitters, length(ref_tx));
ref = zeros(transmitters, length(ref_tx));
for a = 1:1:transmitters
%      ref_probe(a,:) = probe_mag*sin((.1 + .3*(a-1))*t);
%      ref_probe(a,:) = probe_mag*sin((.7 + .2*a + .1*sin(.06*t)).*t);
    ref_probe(a,:) = probe_mag*randn(1,samples); %probe signal
    ref(a,:) = ref_tx; %reference signal is same for both ...
        transmitters
end
% soi = zeros(1, samples);
soi = [zeros(1, samples/2), ...
    soi_mag*sin(.002*(samples/2:1:samples))]; %signal of interest

w = ones(transmitters, taps); %weights
grad = ones(transmitters, taps); %gradients
w_power = ref_mag*transmitters; %constant weight magnitude level
w = w*sqrt(w_power/sum(w(:).^2));

tx = zeros(transmitters, samples); %what gets transmitted
```

```
for a = 1:1:transmitters
    tx(a,1:averaging) = ref(a,1:averaging);
end
y_rlms = zeros(transmitters, samples);
err_rlms = zeros(transmitters, samples);
wsave_rlms = zeros(taps_rlms*transmitters, samples);
xsave_rlms = zeros(taps_rlms*transmitters, samples);
psave_rlms = zeros(taps_rlms*transmitters, samples);
rsave_rlms = zeros(taps_rlms*transmitters, samples);
nxsave_rlms = zeros(transmitters, samples);
output_pre_finale = zeros(1, samples); %canceller output
output_finale = zeros(1, samples); %canceller output after LPF

h_true = h_true/sqrt(sum(h_true(:).^2)); %normalize true channel ...
    to 1
h_est = zeros(2, taps_rlms); %initialize channel estimate
h_ref = zeros(transmitters, taps_rlms + averaging); %temp conv holder
d_true_part = zeros(transmitters, averaging + length(h_true(1,:)));
h_change_factor = 0; %channel change slope (change per cycle)

w_track = ones(numel(w), samples - averaging); %weights
w_track_rlms = ones(numel(h_est), samples - averaging); %h_est
h_track_true = ones(numel(h_true), samples - averaging); %true ...
    channel
t_track = zeros(transmitters, samples - averaging); %transmission ...
    signal
d_track = zeros(1, samples - averaging);
dp_track = zeros(1, samples - averaging); %received power
dp_track_avg = zeros(1, samples - 2*averaging);

cput = cputime;
%step through all time
for n = (averaging + taps + length(h_true(1,:)) + 1):1: ...
        (samples - averaging - 1)
%change channel if necessary
    if(mod(n, h_change_mod) == 0)
        h_change_factor = ...
            h_change_mag*randn(size(h_true))/h_change_mod;
    end
    h_true = h_true + h_change_factor;
    h_true = h_true/sqrt(sum(h_true(:).^2));
    d_true = 0;
    d_est = 0;
    for a = 1:1:transmitters
%transmit signal for an antenna
        tx_temp = conv(ref(a,n - averaging:n), w(a,:));
        tx(a,n) = tx_temp(averaging + 1) + ...
            ref_probe(a,n) + noise_factor_tx*randn;
%update true received signal
        d_true_part(a,:) = conv(h_true(a,:), ...
            tx(a,n - averaging:n));
        d_true_part(a,:) = d_true_part(a,:) + ...
            noise_factor_d*randn(1,length(d_true_part(a,:)));
        d_true = d_true + d_true_part(a,1 + ...
```

```
            length(h_true(1,:)):averaging + 1); %receive signal
    end %for a = 1:1:transmitters
    d_true = d_true + soi(n - length(d_true) + 1:n);
    for a = 1:1:transmitters
%update receive side LMS filter
        save_rlms_start = taps_rlms*(a-1) + 1;
        save_rlms_end = taps_rlms*(a-1) + taps_rlms;
        if(n ≤ averaging_rlms) %prepare initial  values
            xvec=flipud(ref_probe(a,n-taps_rlms+1:n)');
            xsave_rlms(save_rlms_start:save_rlms_end,n)=xvec;
            nxsave_rlms(a,n) = xvec'*xvec;
            psave_rlms(save_rlms_start:save_rlms_end,n) = ...
                d_true(end)*xvec;
            rsave_rlms(save_rlms_start:save_rlms_end,n) =...
                 xvec*xvec'*h_est(a,:)';
            wsave_rlms(save_rlms_start:save_rlms_end,n) = ...
                h_est(a,:)';
        else %if(n ≤ averaging_rlms)
            xvec=flipud(ref_probe(a,n-taps_rlms+1:n)');
            xsave_rlms(save_rlms_start:save_rlms_end,n) = xvec;
            nxsave_rlms(a,n) = xvec'*xvec;
            nx = mean(nxsave_rlms(a,n-averaging_rlms:n-1),2);
            temp1 = mean(wsave_rlms(save_rlms_start:save_rlms_end,...
                 n-averaging_rlms:n-1),2);
            psave_rlms(save_rlms_start:save_rlms_end,n) = ...
                d_true(end)*xvec;
            temp2 = ...
                u3/nx*mean(psave_rlms(save_rlms_start:save_rlms_end,..
                 n-averaging_rlms:n-1),2);
            rsave_rlms(save_rlms_start:save_rlms_end,n)...
                 = xvec*xvec'*h_est(a,:)';
            temp3 = ...
                u3/nx*mean(rsave_rlms(save_rlms_start:save_rlms_end,..
                 n-averaging_rlms:n-1),2);
            h_est(a,:) = temp1 + temp2 - temp3;
            y_rlms(a,n) = h_est(a,:)*xvec;
            err_rlms(a,n) = d_true(end) - y_rlms(a,n);
            wsave_rlms(save_rlms_start:save_rlms_end,n) = ...
                h_est(a,:)';
        end %if(n ≤ averaging_rlms)
%update estimated received signal
        h_ref(a,:) = conv(h_est(a,:), ref(a,n - averaging:n));
        d_est = d_est + conv(h_ref(a,:), w(a,:)); %receive signal ...
            estimate
    end %for a = 1:1:transmitters
%calculate canceller output to find SOI
    output_pre_finale(n) = d_true(end) - sum(y_rlms(:,n)) ...
        - d_est(averaging + 1); %components of canceller output
    output_finale(n) = mean(output_pre_finale(n - averaging:n)); ...
        %with LPF
%update true and estimated received power
    p_true = d_true*d_true'; %true power
    p_est = d_est*d_est'; %the estimated receive power
```

```
%go through every tap on every transmitter and step weight and redo calcs
    for a = 1:1:transmitters
        for x = 1:1:taps
%step weight up
            w_temp1 = w;
            w_temp1(a,x) = w(a,x) + u1*w_power;
            d_temp1 = 0;
            for aa = 1:1:transmitters
                d_temp1 = d_temp1 + conv(h_ref(aa,:), w_temp1(aa,:));
            end
            p_temp1 = d_temp1*d_temp1';
%step weight down
            w_temp2 = w;
            w_temp2(a,x) = w(a,x) - u1*w_power;
            d_temp2 = 0;
            for aa = 1:1:transmitters
                d_temp2 = d_temp2 + conv(h_ref(aa,:), w_temp2(aa,:));
            end
            p_temp2 = d_temp2*d_temp2';
%find minimum power and set gradient
            if(p_temp1 < p_temp2)
                grad(a,x) = (p_temp1 - p_est)/(u1*w_power);
            else
                grad(a,x) = (p_temp2 - p_est)/(-u1*w_power);
            end
        end %for x = 1:1:taps
    end %for a = 1:1:transmitters
%update transmit side weights and normalize them
    if(sum(grad(:)) ≠ 0)
        w = w - u2*grad/sqrt(1 + sum(grad(:).^2));
    end
    w = w*sqrt(w_power/sum(w(:).^2));
%tracking
    h_track_true(:,n) = h_true(:);
    w_track_rlms(:,n) = h_est(:);
    w_track(:,n) = w(:);
    d_track(n) = d_true(end);
    dp_track(n) = p_true/averaging;
    if(n > averaging)
        dp_track_avg(n) = mean(dp_track(n - averaging:n));
    end
end %for n = (averaging + 1):1:(samples - averaging - 1)
cputime - cput

% figure
% plot(tx(1,:))
% title('transmission 1');
% figure
% plot(tx(2,:))
% title('transmission 2');

% figure
% plot(d_track)
```

```
% title('Received Signal');
% xlabel('Time (samples)')
% ylabel('Signal Value')

% figure
% plot(10*log10(dp_track(round(find(dp_track == ...
%     max(dp_track))*.75):(samples - averaging - 1))))
% title('receive power');
% xlabel('Time (samples)')
% ylabel('Power (dB, relative to transmitter)')

figure
plot(10*log10(dp_track_avg(round(find(dp_track_avg == ...
    max(dp_track_avg))*.75):(samples - averaging - 1))))
% title('Average Receive Power');
xlabel('Time (samples)')
ylabel('Power (dB, relative to transmitter)')

figure
plot(w_track')
% title('Weights');
% legend('w(1,1)', 'w(2,1)', 'w(1,2)', 'w(2,2)',...
%     'w(1,3)', 'w(2,3)', 'w(1,4)', 'w(2,4)');
ax = axis;
axis([(averaging + taps + 1), (samples - averaging - 1), ax(3), ...
    ax(4)]);
xlabel('Time (samples)')
ylabel('Weight Value')

figure
subplot(1,2,1);
plot(w_track_rlms')
title('Receive Side Weights');
% legend('w(1,1)', 'w(2,1)', 'w(1,2)', 'w(2,2)',...
%     'w(1,3)', 'w(2,3)', 'w(1,4)', 'w(2,4)');
xlabel('Time (samples)')
ylabel('Weight Value')
ax = axis;
axis([(averaging + taps + 1), (samples - averaging - 1), ax(3), ...
    ax(4)]);
ax = axis;
subplot(1,2,2);
plot(h_track_true')
title('True Channel')
% legend('h(1,1)', 'h(2,1)', 'h(1,2)', 'h(2,2)',...
%     'h(1,3)', 'h(2,3)', 'h(1,4)', 'h(2,4)');
xlabel('Time (samples)')
ylabel('Channel Tap Value')
axis(ax);

figure
subplot(3,1,2);
plot(output_pre_finale, 'r')
title('Unfiltered Canceller Output');
```

```
% ax = axis;
% axis([(averaging + taps + 1), (samples - averaging - 1), ax(3), ...
    ax(4)]);
% ax = axis;
axis([(averaging + taps + 1), (samples - averaging - 1), -1, 1]);
subplot(3,1,3);
plot(output_finale, 'b')
title('Canceller Output with Moving Average Filter')
axis([(averaging + taps + 1), (samples - averaging - 1), -1, 1]);
subplot(3,1,1);
plot(soi);
title('Signal of Interest');
axis([(averaging + taps + 1), (samples - averaging - 1), -1, 1]);
% axis(ax);
```

## A.6 Matlab Code for Noise Variance Simulation

```
TRIALS = 100;
SAMPLES = 20000;
WEIGHTS = 4; %total number of transmit weights
%note: H in the simulation is B in the analysis
FH_FIXED = 1; %1 if F and H are fixed for the entire simulation
NOISE_IMPULSE = 0; %an optional noise impulse function for the input
w = zeros(WEIGHTS,1);
u2 = 1; %(this is '\mu')
sigma = .001; %channel noise standard deviation
w_track = zeros(WEIGHTS,SAMPLES,TRIALS);
h_track = zeros(WEIGHTS,SAMPLES,TRIALS);
%if F and H matrices vary randomly during the simulation
if(FH_FIXED == 0)
    H = zeros(WEIGHTS,WEIGHTS,SAMPLES);
    F = zeros(WEIGHTS,WEIGHTS,SAMPLES);
    expected_sigma = zeros(1, SAMPLES);
    for n = 1:1:SAMPLES
        H(:,:,n) = .1*randn(WEIGHTS);
        F(:,:,n) = .1*randn(WEIGHTS);
        F(:,:,n) = F(:,:,n)*F(:,:,n)';
        H(:,:,n) = H(:,:,n)*H(:,:,n)';

        [V, Lambda] = eig(eye(WEIGHTS)-u2*F(:,:,n));
        G = u2^2*V'*H(:,:,n)*H(:,:,n)'*V;
        g = diag(G);
        E = Lambda'*Lambda;
        e = diag(E);
        psiFN = 0;
        for m = 1:1:WEIGHTS
            psiFN = psiFN + g(m)/(1-e(m));
        end
        expected_sigma(n) = sigma^2*psiFN;
    end
%if F and H matrices are fixed during the simulation
else
%create and normalize F and H
    F = randn(WEIGHTS);
    F = F*F';
    F = 1*F/sqrt(sum(F(:).^2));
    H = randn(WEIGHTS);
    H = H*H';
    H = 1*H/sqrt(sum(H(:).^2));
%calculate expected weight standard deviation (expected_sigma)
    [V, Lambda] = eig(eye(WEIGHTS)-u2*F);
    G = u2^2*V'*H*H'*V;
    g = diag(G);
    E = Lambda'*Lambda;
    e = diag(E);
    psiFN = 0;
    for m = 1:1:WEIGHTS
```

```
        psiFN = psiFN + g(m)/(1-e(m));
    end
    expected_sigma = sigma^2*psiFN*ones(1, SAMPLES);
end

for trials = 1:1:TRIALS
%run simulation for each trial
    for n = 1:1:SAMPLES
        if(NOISE_IMPULSE == 0)
            h_est_noise = sigma*(randn(WEIGHTS,1));
        else
            if n < 100 %optional noise impulse
                h_est_noise = sigma*(rand(WEIGHTS,1));
            else
                h_est_noise = zeros(WEIGHTS,1);
            end
        end
        if(FH_FIXED == 0)
            w = w - u2*F(:,:,n)*w - u2*H(:,:,n)*h_est_noise;
        else
%linearized weight update equation
            w = w - u2*F*w - u2*H*h_est_noise;
        end
        w_track(:,n,trials) = w;
        h_track(:,n,trials) = h_est_noise;
    end
    w = zeros(WEIGHTS,1);
end

%average over the trials
w_avg = mean(w_track, 3);
w_var = var(w_track, 0, 3);
h_avg = mean(h_track, 3);
h_var = var(h_track, 0, 3);

close all
figure
plot(w_var')
ylabel('Variance')
xlabel('Sample')

figure
plot(sum(w_var,1))
hold on
hand = plot(expected_sigma,'r');
set(hand, 'LineWidth', 2);
ylabel('Variance Sum')
xlabel('Sample')

figure
w_track_temp = zeros(SAMPLES,TRIALS);
w_track_temp(:,:) = w_track(1,:,:);
plot(w_track_temp)
```

# Bibliography


[1] H. L. Adaniya. Wideband active antenna cancellation. Master's thesis, Massachusetts Institute of Technology, June 2008.

[2] J. S. Herd, P. D. Fiore, S. M. Duffy, B. T. Perry, and G. F. Hatke. Simultaneous transmit and receive for look-through electronic attack. Technical report, MIT Lincoln Laboratory, February 2010.

[3] K. Abe, H. Watanabe, and K. Hirasawa. Simultaneous transmission and reception system by adaptive cancelling. *Electrical Engineering in Japan*, 123(1):1–7, 1998.

[4] B. D. Maxson. Optimal cancellation of frequency-selective cosite interference. Master's thesis, University of Cincinnati, November 2002.

[5] T. Riihonen, S. Werner, and R. Wichman. Spatial loop interference suppression in full-duplex MIMO relays. In *Asilomar Conf. on Signals, Systems, and Computers*, 2009.

[6] D. W. Bliss, P. A. Parker, and A. R. Margetts. Simultaneous transmission and reception for improved wireless network performance. In *IEEE Wrkshp. on Stat. Sig. Processing*, pages 478–482, 2007.

[7] W. L. Brogan. *Modern Control Theory.* Prentice-Hall, 1985.

[8] A. V. Oppenheim and A. S. Willsky. *Signals and Systems.* Prentice-Hall, 1997.

[9] G. Strang. *Introduction to Linear Algebra.* Wellesley-Cambridge Press, 2003.

[10] S. Haykin. *Adaptive Filter Theory.* Prentice-Hall, 1991.

[11] B. Widrow and S. D. Stearns. *Adaptive Signal Processing.* Prentice-Hall, 1985.

[12] A. Papoulis. *Probability, Random Variables, and Stochastic Processes.* McGraw-Hill, 1984.

[13] G. H. Golub and C. F. Van Loan. *Matrix Computations.* Johns-Hopkins University Press, 3rd edition, 1996.

[14] B. Hassibi, A. H. Sayed, and T. Kailath. $H^\infty$ optimality of the LMS algorithm. *IEEE Trans. Signal Processing*, 44(2):267–280, February 1996.

[15] D. M. Pozar. *Microwave Engineering.* John Wiley and Sons, Inc., 2005.

[16] T. K. Sarkar et al. A survey of various propagation models for mobile communications. *IEEE Antennas and Propagation Magazine*, 45(3):51–82, June 2003.

[17] R. T. Compton. *Adaptive Antennas.* Prentice-Hall, 1988.

[18] L. Ljung. *System Identification.* Prentice-Hall, 1999.

[19] D. L. Kewley and R. L. Clark. Feedforward control using the higher-harmonic, time-averaged gradient descent algorithm. *JASA*, 97(5):2892–2905, May 1995.