11-16-2004

Antenna Array Output Power Minimization Using Steepest Descent Adaptive Algorithm

Sandra Gomulka Johnson
University of South Florida

Antenna Array Output Power Minimization Using Steepest Descent Adaptive Algorithm

by

Sandra Gomulka Johnson

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Electrical Engineering
College of Engineering
University of South Florida

Major Professor: Arthur David Snider, Ph.D.
Paul Flikkema, Ph.D.
Vijay K. Jain, Ph.D.
Nagarajan Ranganathan, Ph.D.
Mourad Ismail, Ph.D.

Date of Approval:
November 16, 2004

Keywords: signal processing, jammer nulling, GPS reception, exact gradient, Hilbert space gradient

Dedication

I dedicate this dissertation to my husband, William Joel Dietmar Johnson, and to my daughter, Cora Julia Kimberly Johnson. It is through his generosity and motivation, her patience, and their love that this work has been accomplished.

Acknowledgments

I would like to acknowledge my advisor, Dr. A. David Snider, for his patience, encouragement, and persistence during the course of our research. I must also acknowledge Dr. Paul Flikkema for introducing me to this research problem. I thank him as well as Chris Sperandio for their help with LaTeX. The friendships of Carlos and Florence Briceno and the LaPointe family have been invaluable. Also I must include my parents, Richard and Rosemarie Gomulka, for their unwavering encouragement.

Finally, all thanks be to God from Whom all good things come.

Table of Contents

List of Tables

List of Figures

Antenna Array Output Power Minimization Using Steepest Descent Adaptive Algorithm

Sandra Gomulka Johnson

ABSTRACT

A beamforming antenna array is a set of antennas whose outputs are weighted by complex values and combined to form the array output. The effect of the complex valued weights is to steer lobes and nulls of the array pattern to desired directions. These directions may be unknown and so the antenna weights must be adjusted adaptively until some measure of array performance is improved, indicating proper lobe or null placement.

An adaptive algorithm to adjust the complex weights of an antenna array is presented that nulls high power signals while allowing reception of GPS signals as long as the signals arrive from different directions. The GPS signals are spread spectrum modulated and have very low average power, on the order of background thermal noise. Simulations of the adaptive algorithm minimize the output power of the array to within 5 dB of the background noise level.

The adaptive algorithm, named the Hilbert-space-based (HSB) gradient method, is based on the steepest descent algorithm and implements an efficient, *exact* gradient calculation.

With M antennas in the array, only $M-1$ weights are adjustable; one antenna weight is held constant to prevent the algorithm from minimizing the output power trivially by zeroing all weights thus preventing the reception of any signal by the array. It appears that $M-1$ adjustable antenna weights can null $M-1$ unwanted signals (jammers). However,

in the course of the algorithm development, a few configurations of antennas and jammer arrival directions were found where this is not true. Even when the jammer arrival directions are known ('oracle') certain configurations are mathematically impossible to cancel.

The oracle solution has a matrix formulation and under certain conditions an exact solution for antenna weights to annihilate the jammers can be found. This provides an excellent comparison tool to assess the performance of other adaptive algorithms.

The HSB gradient adaptive algorithm and the oracle solution are both implemented in Matlab. Outputs of both are plotted for comparison.

Chapter 1

Introduction

1.1 Motivations

An antenna is a device to transmit and receive electromagnetic radiation. The antenna creates a pattern of radiation that spatially describes areas of gain (lobes) or attenuation (nulls) for the signals it transmits or receives. The radiation pattern of a single isotropic antenna is simpler than that of an array of multiple antennas; the radiation pattern of an array consists of a weighted superposition of each antenna's radiation pattern. The location of lobes and nulls in an array radiation pattern can be controlled with a beamforming network consisting of a signal processor implementing an algorithm that weights the outputs of each antenna and combines them to form the array output [9]. The subject of this dissertation is the signal processing algorithm. The RF data received by the antennas is assumed to be downconverted to a digital baseband stream and the signal processing constituting the proposed algorithm occurs completely in the digital domain.

The objective of the proposed Hilbert-space-based (HSB) gradient adaptive algorithm is to receive signals from the Global Positioning System (GPS) of satellites by nulling all jammer signals received by the array. The desired GPS signals have very low power, on the order of thermal noise, -160 dBw, so if the output power of the array is above this level, it is assumed to come from jammer signals. The adaptive algorithm reduces the power output of the array by adapting weights of the antennas to null power. The adaptive algorithm is not based on forming lobes in the direction of arrival of the GPS signals.

The dissertation is organized as follows. The remainder of Chapter 1 introduces what a beamforming network is and includes an example to illustrate the effect of antenna weights on the array output.

Chapter 2 introduces the vector notation used to describe the antenna array geometry and defines variables describing the input and output signals of the array.

Chapter 3 includes a survey of existing adaptive algorithms related to the HSB gradient algorithm and introduces the oracle solution. The oracle solution is a standard for assessing realistic solutions for antenna array weights based on array geometry and known jammer arrival directions.

Chapter 4 describes the theory of the HSB gradient algorithm including details of the exact gradient calculation and noise simulation.

Chapter 5 describes the implementation of the HSB gradient algorithm in Matlab.

Chapter 6 contains output of the Matlab simulation of the HSB gradient algorithm, illustrating its adaptive performance.

1.2 Antenna Array Fundamentals

This section will briefly describe the radiation pattern of an array of antennas and the effect the antenna weights have on this pattern.

The radiation pattern of an array is the superposition of the individual radiation patterns of each antenna in the array. The principle of pattern multiplication [1] states that the radiation pattern of an array of identical antennas is determined by the product of the *element factor* and the *array factor*. The *element factor* is the radiation pattern of an individual antenna in the array. The *array factor* depends on the geometry of the array antennas as well as on the complex weight of each antenna in the array. By adjusting the complex weights of the antennas in the array, the array factor and thus, the overall array radiation pattern, can be modified. Adaptive algorithms describe how to adjust the weights and a beamforming network implements these weight adjustments.

A beamforming network assigns an adjustable weight to each antenna output signal. Each weight is a complex number which has a variable gain (magnitude) and a variable phase. Adjusting the weight of each antenna modifies the array factor and has the effect of steering lobes and nulls to various positions. Signals arriving at the array in a null location will be severely attenuated or even eliminated by the array; signals arriving at a

lobe will be enhanced by the array. Thus the antenna array performs spatial filtering using the mainlobe, sidelobes, and nulls of its radiation pattern. These are somewhat analogous to the passband, transition band, and stop band from traditional circuit filter theory. The spatial areas in the array pattern of gain and attenuation are frequency dependent. A deep null at ω_1 may not be as deep, or even a null, at another frequency ω_2. The bandwidths of the desired and jammer signals must be taken into account when designing the antenna array as well as the beamforming network. Each antenna weight in a narrowband array is a complex number; the antenna weights controlled by the proposed adaptive algorithm are implemented as such in the simulations to follow. For wideband arrays, the antenna weights must provide the ability to null jammers over a range of frequencies. These weights, being frequency dependent, can be implemented at each antenna as a linear filter or a digital tapped delay line [10]. General notation for a jammer impinging on an antenna array is developed in Chapter 2.

A simple example to illustrate the effect of gain and phase adjustments in a narrowband array follows. Consider a linear antenna array of 3 antennas each separated by a distance r as in Figure 1. The signal arriving at the array is considered to be a plane wave modeled

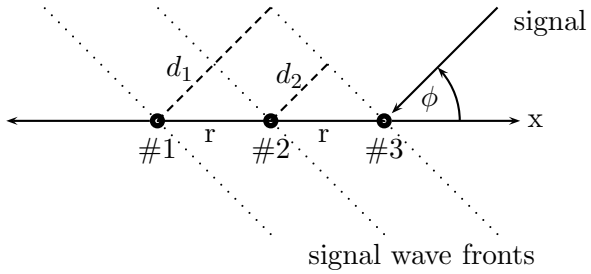

Figure 1. A 3 Element Linear Array.

as a sinusoid with frequency ω arriving at an angle ϕ with respect to the axis of the array. A particular wavefront of the signal will arrive at antenna #3 first and after a delay of $\tau = d_2/c = r\cos\phi/c$ seconds (c is the propagation speed of the signal) the same wavefront of the signal will arrive at antenna #2 and finally, after an additional delay of $(d_1 - d_2)/c = r\cos\phi/c = \tau$ seconds, it arrives at antenna #1. The output of the array is

$$w_3 \cos(\omega t + q_3) + w_2 \cos(\omega(t-\tau) + q_2) + w_1 \cos(\omega(t-2\tau) + q_1), \tag{1}$$

where w_m represents the gain of the mth antenna and q_m represents the phase shift introduced by the mth antenna.

If the antenna phases are adjusted to compensate for the propagation delays so that the signal appears to have arrived at each antenna at the same time, the signal will appear to be boosted in amplitude at the output of the array. This is accomplished by adjusting the phase of antenna #2 to be $q_2 = \omega\tau$ and the phase of antenna #1 to be $q_1 = 2\omega\tau$ while keeping $w_1 = w_2 = w_3 = 1$. This results in the array output of $\cos(\omega t) + \cos(\omega(t-\tau) + \omega\tau) + \cos(\omega(t-2\tau) + 2\omega\tau) = 3\cos(\omega t)$.

The antenna weights can also be adjusted so that the array output is eliminated. With the antenna gains set to $w_2 = -1/2$ and $w_1 = -1/2$ and q_2 and q_1 defined as above, the output of the array is $\cos(\omega t) - \frac{1}{2}\cos(\omega(t-\tau) + q_2) - \frac{1}{2}\cos(\omega(t-2\tau) + q_1) = \cos(\omega t) - \frac{1}{2}\cos(\omega t) - \frac{1}{2}\cos(\omega t) = 0$ and the signal has been eliminated at the output of the array.

These examples show that the choice of antenna weights can significantly affect the array output. Normally, it is up to the weight adjustment algorithm to distinguish friendly signals from jammer signals for proper lobe and null placement. The proposed algorithm does not distinguish the type of incoming signal although it assumes different power levels between the desired and unwanted signals; it simply tracks the output power of the array and adjusts the antenna weights until the output power is minimized. Adjusting antenna weights to achieve the desired array output when the input to the array is not well defined is the dilemma all adaptive algorithms must overcome, including the proposed algorithm to be described in Chapter 4.

Chapter 2

Antenna Array Notation

This chapter will introduce the notation used to describe the configuration of antennas in an array as well as the input and output of the array. The geometry of the antenna array and the incoming signals will be expressed mathematically using vector notation. The incoming signals to the array are of 3 types: the GPS signals, the jammers, and noise. The GPS signals and noise are considered to have low power on the order of background noise levels, while the jammers are assumed to have a much higher power level. The basic strategy of the proposed HSB gradient algorithm focuses on reducing the array output jammer power to a level comparable to the output GPS signal power, so that the latter can be detected with spread-spectrum technology.

An array can consist of any number of antennas (assumed identical) arranged in any configuration. One antenna in the array is selected as the reference antenna. The origin of a reference coordinate system is chosen to coincide with the location of this reference antenna. The other antennas in the array are referred to as peripheral antennas. The antennas are located at positions $\vec{a}_m$, $m = 1, 2, \ldots, N_{ant}$ where N_{ant} is the total number of antennas in the array. All peripheral antennas have adjustable gain w_m and adjustable phase q_m represented as a complex term $A_m = w_m e^{iq_m}$. The weight of the reference antenna, $m = 1$, is fixed at a constant value , $A_1 \equiv 1$.

The total GPS signal received by antenna $\#m$ is denoted $\chi_{GPS}^{(m)}(t)$. The number of individual GPS signals, each from a different GPS satellite, is not taken into account nor are their directions of arrival at the array.

The noise at antenna $\#m$ is $N_m(t)$. Details of the noise model used in the simulation are presented in Chapter 4.

The jammers, modeled as plane waves, arrive from directions $\vec{j}_n$, $n = 1, 2, \ldots, N_{jam}$, where N_{jam} is the total number of jammers impinging on the array, each with frequency ω_n and (complex) amplitude J_n. The phase of jammer $\#n$ at the reference antenna, ε_n, is incorporated into J_n. The $\vec{j}_n$ are unit vectors. A jammer signal received by the reference antenna as a sinusoid with frequency ω is (the real part of)

$$Je^{i\omega t} = |J_n|e^{i(\omega t+\varepsilon_n)}.$$

As stated previously in section 1.2, it takes a finite time for a jammer wavefront to propagate across the array and so the jammer wavefront reaches each peripheral antenna with a corresponding phase delay, with respect to the phase of the jammer wavefront at the reference antenna. These differences in propagation distance between the peripheral antennas and the reference antenna are given by $\vec{a}_m \bullet \vec{j}_n$, the projection of the location vector of antenna $\#m$ onto the vector representing the arrival direction of jammer $\#n$. Figure 2 shows a jammer arriving at a 3 element antenna array. The (negative) distance x_2 equals $\vec{a}_2 \bullet \vec{j}_1$ and the distance x_3 equals $\vec{a}_3 \bullet \vec{j}_1$. Each of these propagation distance differences

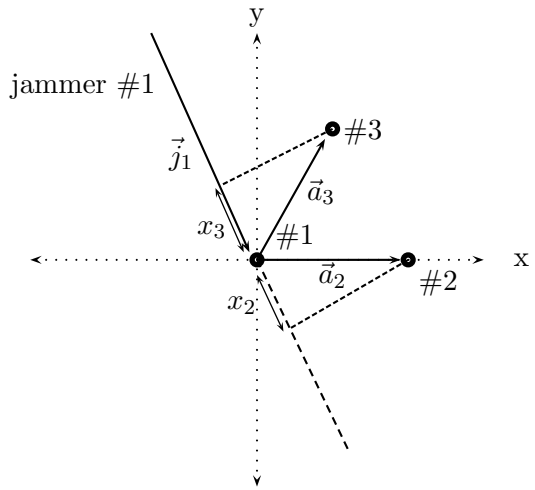

Figure 2. Jammer Propagation Delay.

has an associated time delay of $\tau = \vec{a}_m \bullet \vec{j}_n/c$, where c is the speed of propagation of the jammer signal (taken to be the speed of light), and a corresponding phase difference of

$$\omega\tau = \frac{\omega(\vec{a}_m \bullet \vec{j}_n)}{c} = \frac{2\pi f(\vec{a}_m \bullet \vec{j}_n)}{c} = \frac{2\pi(\vec{a}_m \bullet \vec{j}_n)}{\lambda} \tag{2}$$

where $\omega = 2\pi f$ and $\frac{f}{c} = \frac{1}{\lambda}$. From figure 2, the jammer wavefront reaches antenna #3 x_3/c seconds before it propagates to the reference antenna and the jammer propagates to antenna #2 x_2/c seconds after it has reached the reference antenna.

Each of the m antennas in the array weights its input by A_m and the array output is the sum of these weighted signals. The output signal at antenna $\#m$ consisting of a GPS signal, noise, and N_{jam} jammers is then

$$A_m\chi_{GPS}^{(m)}(t) + A_m N_m(t) + \sum_{n=1}^{N_{jam}} J_n e^{i\omega_n t}\ e^{i\frac{2\pi}{\lambda_n}\vec{a}_m \bullet \vec{j}_n}\ A_m. \tag{3}$$

The total output of the array, with N_{jam} jammers arriving at each antenna in the array, is the sum of the output of each antenna in the array

$$\sum_{m=1}^{N_{ant}} A_m\chi_{GPS}^{(m)}(t) + \sum_{m=1}^{N_{ant}} A_m N_m(t) + \sum_{n=1}^{N_{jam}}\sum_{m=1}^{N_{ant}} J_n e^{i\omega_n t}\ e^{i\frac{2\pi}{\lambda_n}\vec{a}_m \bullet \vec{j}_n}\ A_m, \tag{4}$$

or in matrix form

$$[\chi_{GPS}^{(1)}(t)\cdots\chi_{GPS}^{(N_{ant})}(t)]\begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_{N_{ant}} \end{bmatrix} + [N_1(t)\cdots N_{N_{ant}}(t)]\begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_{N_{ant}} \end{bmatrix}$$

$$+[J_1 e^{i\omega_1 t} J_2 e^{i\omega_2 t}\cdots J_{N_{jam}} e^{i\omega_{N_{jam}} t}]\begin{bmatrix} e^{i\frac{2\pi}{\lambda_1}\vec{a}_1\cdot\vec{j}_1} & \cdots & e^{i\frac{2\pi}{\lambda_1}\vec{a}_{N_{ant}}\cdot\vec{j}_1} \\ \vdots & \ddots & \vdots \\ e^{i\frac{2\pi}{\lambda_{N_{jam}}}\vec{a}_1\cdot\vec{j}_{N_{jam}}} & \cdots & e^{i\frac{2\pi}{\lambda_{N_{jam}}}\vec{a}_{N_{ant}}\cdot\vec{j}_{N_{jam}}} \end{bmatrix}\begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_{N_{ant}} \end{bmatrix}$$

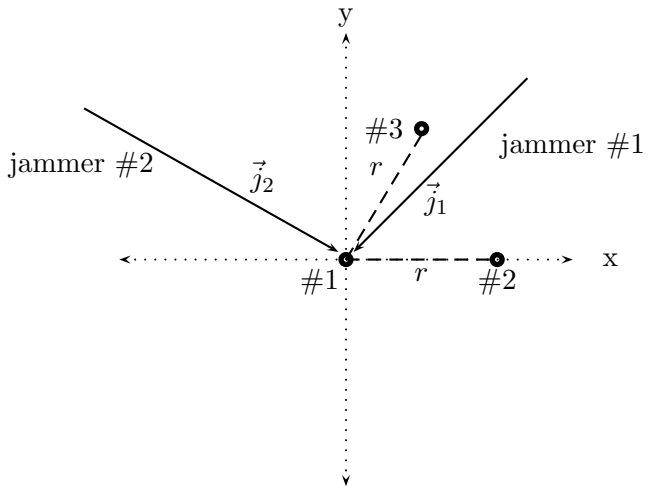

Figure 3. 3 Element Antenna Array with 2 Jammers.

$$= \chi_{GPS}(t)A + N(t)A + J(t)CA = x(t)A \tag{5}$$

where x is the 'generic' input vector to the array, $(\chi^{(m)}_{GPS}(t) + N_m(t) + J(t)C)$, and C is a matrix whose (n, m)th element represents the phase difference of jammer $\#n$ arriving at antenna $\#m$

$$C = \begin{bmatrix} e^{i\frac{2\pi}{\lambda_1}\vec{a}_1\cdot\vec{j}_1} & e^{i\frac{2\pi}{\lambda_1}\vec{a}_2\cdot\vec{j}_1} & \dots & e^{i\frac{2\pi}{\lambda_1}\vec{a}_{N_{ant}}\cdot\vec{j}_1} \\ e^{i\frac{2\pi}{\lambda_2}\vec{a}_1\cdot\vec{j}_2} & e^{i\frac{2\pi}{\lambda_2}\vec{a}_2\cdot\vec{j}_2} & \dots & e^{i\frac{2\pi}{\lambda_2}\vec{a}_{N_{ant}}\cdot\vec{j}_2} \\ \vdots & \vdots & \ddots & \vdots \\ e^{i\frac{2\pi}{\lambda_{N_{jam}}}\vec{a}_1\cdot\vec{j}_{N_{jam}}} & e^{i\frac{2\pi}{\lambda_{N_{jam}}}\vec{a}_2\cdot\vec{j}_{N_{jam}}} & \dots & e^{i\frac{2\pi}{\lambda_{N_{jam}}}\vec{a}_{N_{ant}}\cdot\vec{j}_{N_{jam}}} \end{bmatrix}. \tag{6}$$

A simple example will illustrate the vector notation just introduced. Consider a 3 element planar array with the reference antenna located at the origin ($\vec{a}_1 = 0$) and peripheral antennas located at $\vec{a}_2 = r\hat{i},\ \vec{a}_3 = \frac{1}{2}r\hat{i} + \frac{\sqrt{3}}{2}r\hat{j}$ as in figure 3. The arrival directions of two jammers are $\vec{j}_1 = \frac{1}{\sqrt{2}}\hat{i} + \frac{1}{\sqrt{2}}\hat{j}$ and $\vec{j}_2 = -\frac{\sqrt{3}}{2}\hat{i} + \frac{1}{2}\hat{j}$.

The dot products of the antenna vectors and jammer vectors are

$$\begin{aligned}
\vec{a}_1 \bullet \vec{j}_1 &= 0 \\
\vec{a}_2 \bullet \vec{j}_1 &= \tfrac{r}{\sqrt{2}} \\
\vec{a}_3 \bullet \vec{j}_1 &= (\tfrac{1}{2\sqrt{2}} + \tfrac{\sqrt{3}}{2\sqrt{2}})r \\
\vec{a}_1 \bullet \vec{j}_2 &= 0 \\
\vec{a}_2 \bullet \vec{j}_2 &= -\tfrac{\sqrt{3}}{2}r \\
\vec{a}_3 \bullet \vec{j}_2 &= 0
\end{aligned}$$

and the array output is

$$[\chi^{(1)}_{GPS}(t)\ \chi^{(2)}_{GPS}(t)\ \chi^{(3)}_{GPS}(t)] \begin{bmatrix} A_1 \\ A_2 \\ A_3 \end{bmatrix} + [N_1(t)\ N_2(t)\ N_3(t)] \begin{bmatrix} A_1 \\ A_2 \\ A_3 \end{bmatrix}$$

$$+ \quad \left[J_1 e^{i\omega_1 t}\ J_2 e^{i\omega_2 t}\right] \begin{bmatrix} 1 & e^{i\frac{2\pi}{\lambda}r/\sqrt{2}} & e^{i\frac{2\pi}{\lambda}r(1+\sqrt{3})/2\sqrt{2}} \\ 1 & e^{-i\frac{2\pi}{\lambda}r\sqrt{3}/2} & 1 \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \\ A_3 \end{bmatrix}.$$

Since the array output is dominated by the high power jammers, the task of the proposed algorithm is to adjust the antenna weights, $A_2, \ldots, A_{N_{ant}}$, so that nulls are steered toward the jammer arrival directions, minimizing the jammer power to the level of the noise and GPS signals. This leaves the GPS signals intact to be processed by spread spectrum demodulation techniques. Recall the reference antenna weight is fixed at $A_1 = 1$ so that the algorithm minimizes the output power while avoiding the trivial solution, $A_m = 0,\ m = 1, 2, \ldots, N_{ant}$.

Note that even though the jammer arrival directions appear explicitly in the formulation just presented, in practice one does not know the jammer arrival directions or frequencies. At this point of the exposition, the inclusion of the jammer arrival directions is necessary to formulate the phase differences between the jammers and the antennas as the jammers propagate across the array.

Chapter 3

Existing Algorithms and Assessment Methodology: Comparison with the Oracle Solution

Adaptive algorithms are used in a wide variety of applications. When used with an antenna array, the adaptive algorithm modifies the antenna weights based on some cost function. When the cost function indicates improved array performance, the algorithm is adapting the antenna weights in the desired fashion. The choice of algorithm to adaptively adjust the antenna weights is determined by such factors as the performance characteristics to be improved, known information about the operating environment of the array, and cost of implementation.

This chapter will provide a review of existing techniques used by adaptive algorithms. The proposed algorithm is related to and shares similarities with these existing algorithms. Also, this chapter contains details of the oracle solution. This is an exact solution for antenna weights based on minimizing jammer output power *with* knowledge of the jammer parameters. It is called the 'oracle' because the jammer information is unavailable in practice. The oracle provides a metric upon which the performance of other adaptive algorithms can be assessed.

A direct comparison of antenna weights between one algorithm and another is not always possible. After the first iteration of each algorithm, the weights evolve along different trajectories. This chapter will introduce the scenario that an array may encounter certain configurations which cause the algorithm to stall. The oracle solution is a history-free standard assessing how an algorithm is doing at a particular configuration, regardless of how it got there.

3.1 Existing Adaptive Algorithms

3.1.1 Wiener Solution

A popular cost function that is used in adaptive array algorithms is the mean squared error [10]. The error is a measure between some reference or desired array output signal, d, and the actual array output signal, $x(t)A$ where A is a vector of antenna weights and $x(t)$ is the vector of antenna input signals (assumed real for this exposition). The error signal is

$$e(t) = d(t) - x(t)A$$

and the mean squared error is

$$\text{MSE} = E[e^2(t)] = d^2(t) - 2E[d(t)x(t)]A + A^T E[x^T(t)x(t)]A.$$

The term $E[d(t)x(t)] = r_{xd}$ is the cross-correlation vector of the actual input and the desired signal and the term $E[x^T(t)x(t)] = R_{xx}$ is the autocorrelation matrix of the actual input signal. The MSE is a quadratic function of the antenna weights and can be visualized as a surface with (typically) a unique minimum. The weights correspond to some point on the MSE surface and those that correspond to the minimum of this surface are those that are sought by the adaptive algorithm. A way to approach the minimum is by adjusting the weight values in the direction of the negative gradient, $-\nabla$, on this surface

$$A_{\text{new}} = A_{\text{old}} - \mu\nabla$$

where μ is a constant that determines the fraction of the gradient to be implemented per weight adjustment. The gradient of the MSE (assuming real elements of A) is

$$\nabla = \frac{\partial E[e^2(t)]}{\partial A} = 2r_{xd} + 2R_{xx}A.$$

The gradient has a value of zero at the minimum point on the surface. Setting the gradient to zero and solving for A yields the Wiener solution for the antenna weights,

$$A = -R_{xx}^{-1} r_{xd}.$$

This solution, however, requires the statistics of the input signal to be known, i.e., r_{xd} and R_{xx}. This is rare in practice so, an alternative to the Wiener solution is proposed herein.

3.1.2 Least Mean Square Solution

An alternative to the Wiener solution involves the instantaneous value of the gradient on the squared error surface [11]. The squared error is

$$e^2(t) = (d(t) - x(t)A)^2$$

and its gradient is

$$\nabla(e^2(t)) = \frac{\partial e^2(t)}{\partial A} = -2e(t)x(t).$$

This is only an approximation to the *Mean* Squared Error gradient because the instantaneous values of the error and input signals, instead of their exact statistics as in the Wiener solution, are used in the gradient calculation. The weight update equation becomes

$$A_{\text{new}} = A_{\text{old}} + \mu(2e(t)x(t)).$$

This is usually called the least mean square (LMS) algorithm. Despite the name of this algorithm, the LMS does not seek the least *mean* squared error but rather the least squared error in the solution for A. As stated previously, the squared error is a quadratic function of the weights and can typically be interpreted as a surface with a unique minimum. The LMS algorithm approaches this minimum with iterative weight adjustments in the negative gradient direction. The weight adjustments continue until the gradient is zero or until

further weight adjustments no longer reduce the squared error. At this point, the array output due to these weight values approximates the desired signal with a least squared error.

3.1.3 Power Inversion Solution

Compton has proposed an algorithm whose objective is to minimize the output power of an antenna array and it is called the power-inversion algorithm for adaptive arrays [15]. It improves the signal to interference ratio in an adaptive array and is based on the Howells-Applebaum feedback [12] to minimize array output power. The arrival directions of the desired signals and interference signals need not be known and the best performance occurs when the desired signal is near thermal noise levels. The power-inversion algorithm requires the designer to choose a value for loop gain which depends on three things: 1) the required minimum signal-to-noise ratio (SNR) out of the array, 2) the dynamic range of the signal level that the array must accommodate, and 3) what signals the array is receiving: the desired signal only, the desired and interference signals only, or desired, interference and noise signals. The optimal weights are calculated with an equation dependent on the desired signal power per element, interference power per element, desired SNR per element, and interference-to-noise ratio per element. This algorithm requires a substantial amount of information about the array's operating environment before effective antenna weights can be calculated.

3.1.4 Subgradient Search Technique

Another antenna array power minimization method uses a so-called "subgradient" technique to search a performance surface for the optimal antenna weights [6]. This method assumes an array containing a fixed-weight reference antenna with the remaining antennas, referred to as peripheral antennas, having adjustable weights. The technique sequentially sets the peripheral antenna weights to values from a known, finite list of settings, based on the locations of the antenna elements, until the output power of the array is minimized. The jammers are viewed as phasors whose magnitudes and phases are manipulated by the

peripheral antenna weights. The algorithm adjusts the peripheral antenna weights until the overall jammer phasor from the peripheral antennas cancels or minimizes the jammer phasor from the reference antenna whose weight is fixed.

As an example, consider a planar array in the xy-plane with $N_{ant} = 7$ antennas, 6 peripheral antennas equally spaced around a central reference antenna. Let the origin of the xy axes coincide with the central antenna. The subgradient search sets the peripheral antenna weights to the values from the set $\{\frac{1}{N_{ant}-1}e^{i(\pi-\phi_m)},\ m = 2, 3, \ldots, N_{ant}\}$, where ϕ_m is the angle describing the location of the mth peripheral antenna measured counterclockwise from the $x-$axis. The finite list of settings comprise these values. The technique monitors the output power of the array as each antenna weight takes on each value from the finite set. When all combinations have been tried, the technique chooses the optimal weights as those giving the lowest output power.

The results of this technique were compared in [6] to results of the LMS algorithm for a 4 element planar array receiving GPS and jammer signals. The LMS algorithm outperformed the simple, computationally inexpensive, subgradient technique in static and dynamic cases (roll motion) but with the tradeoff of more complex hardware implementation. Although the subgradient technique was outperformed by the LMS algorithm, it still provided sufficient nulling for successful GPS signal detection.

3.2 A New Candidate: Hilbert-Space-Based (HSB) Gradient Algorithm

The algorithms mentioned above, LMS, Wiener, and Compton, are powerful algorithms when information is known about the array operating environment, and about the input and desired signal characteristics. The desired signal must be approximated in practice and any error in this approximation will propagate to the weight solution. The desired signal for the HSB gradient algorithm to be described in Chapter 4 is a constant (zero); the algorithm adjusts the antenna weights until the output power is minimized.

The statistics of the input signal are rarely known for all potential inputs to the antenna array and the problem is further compounded when the input data is not stationary. The HSB gradient algorithm works successfully with nonstationary data and the dynamic

range of the input signals and output SNR need not be known for successful algorithm performance.

The HSB gradient algorithm efficiently calculates the *exact* gradient on the performance surface based on a time-averaged power output of an array instead of approximating a gradient with instantaneous signal statistics. The HSB gradient algorithm weights are not limited to a finite set of possible values as are the weights from the subgradient search technique. Therefore, the performance of the HSB gradient algorithm is expected to exceed that of the subgradient search technique.

3.3 Algorithm Comparison Considerations

Given the variety of adaptive algorithms that are available, one would like to know which algorithm will result in the best performance for a given application. Comparisons among the outputs of a group of potential algorithms are a way to select the best performing algorithm but this requires that all potential algorithms be implemented, to obtain the outputs to make such comparisons. Therefore, comparing any number of different algorithms can be a very time consuming task.

All algorithms seek a solution that minimizes some measure of the array output *without* knowledge of jammer parameters. This section introduces the 'oracle' solution which seeks to zero the array output *with* knowledge of the jammer parameters. The oracle thus provides a standard against which other algorithms can be measured. If there is no oracle solution, then no other algorithm can possibly succeed in nulling the output for the given jammer parameters.

3.3.1 Oracle Solution

In the course of simulating the HSB gradient algorithm described in Chapter 4 with an array in motion (array motion is simulated with roll, pitch, and yaw), occasionally a configuration of the antennas of the array and the jammers was encountered that confounded the convergence of the algorithm, and the output power was not minimized. The array appeared to move through such regions of divergence and then return to regions of conver-

gence, indicating that the motion of the array through certain jammer arrival configurations caused the divergence of the algorithm.

This opened a new question as to whether it was possible to formulate configurations of antennas and jammers that were mathematically unsolvable. If such configurations could be found and characterized, the configurations that confounded the HSB gradient algorithm might be explained.

The existence of a solution for antenna weights to null the jammers, presuming known antenna locations and known jammer directions, called the "oracle solution", was studied. The configurations of antennas and jammers for which no oracle solution exists were named "confounding configurations".

The oracle solution for the antenna weights will be described here. Recall the output of the array from equation (5) is $\chi_{GPS}\ A + N\ A + J\ C\ A$. The goal of any adaptive algorithm is to minimize the power in this array output. Consider, instead, the task of forcing only the jammer component of the output to be zero, i.e., $J\ C\ A = 0$. Since the jammer amplitude vector, J, cannot be controlled, the oracle cannot exploit any feature about the amplitudes of the jammers, and thus it must solve

$$C\ A = 0$$

(for A). The jammer *amplitudes* do not affect the values of the oracle antenna weights; they provide no information for these weights. The *arrival directions* of the jammers $\vec{j}_n$, however, directly influence the solution. They appear in the elements of C:

$$CA = \begin{bmatrix} e^{i\frac{2\pi}{\lambda_1}\vec{a}_1\cdot\vec{j}_1} & e^{i\frac{2\pi}{\lambda_1}\vec{a}_2\cdot\vec{j}_1} & \cdots & e^{i\frac{2\pi}{\lambda_1}\vec{a}_{N_{ant}}\cdot\vec{j}_1} \\ e^{i\frac{2\pi}{\lambda_2}\vec{a}_1\cdot\vec{j}_2} & e^{i\frac{2\pi}{\lambda_2}\vec{a}_2\cdot\vec{j}_2} & \cdots & e^{i\frac{2\pi}{\lambda_2}\vec{a}_{N_{ant}}\cdot\vec{j}_2} \\ \vdots & \vdots & \ddots & \vdots \\ e^{i\frac{2\pi}{\lambda_{N_{jam}}}\vec{a}_1\cdot\vec{j}_{N_{jam}}} & e^{i\frac{2\pi}{\lambda_{N_{jam}}}\vec{a}_2\cdot\vec{j}_{N_{jam}}} & \cdots & e^{i\frac{2\pi}{\lambda_{N_{jam}}}\vec{a}_{N_{ant}}\cdot\vec{j}_{N_{jam}}} \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_{N_{ant}} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{7}$$

All elements of the first column of C have terms which contain $\vec{a}_1 \bullet \vec{j}_n$ and since $\vec{a}_1 = 0$ for the reference antenna the first column is a column of 1's. Also, $A_1 \equiv 1$ for the reference antenna. Equation (7) can thus be rewritten as

$$\begin{bmatrix} 1 & e^{i\frac{2\pi}{\lambda_1}\vec{a}_2\cdot\vec{j}_1} & \cdots & e^{i\frac{2\pi}{\lambda_1}\vec{a}_{N_{ant}}\cdot\vec{j}_1} \\ 1 & e^{i\frac{2\pi}{\lambda_2}\vec{a}_2\cdot\vec{j}_2} & \cdots & e^{i\frac{2\pi}{\lambda_2}\vec{a}_{N_{ant}}\cdot\vec{j}_2} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & e^{i\frac{2\pi}{\lambda_{N_{jam}}}\vec{a}_2\cdot\vec{j}_{N_{jam}}} & \cdots & e^{i\frac{2\pi}{\lambda_{N_{jam}}}\vec{a}_{N_{ant}}\cdot\vec{j}_{N_{jam}}} \end{bmatrix} \begin{bmatrix} 1 \\ A_2 \\ \vdots \\ A_{N_{ant}} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{8}$$

These equations can be further rearranged by taking the first column of C to the right hand side. The C matrix with the first column removed is now referred to as $C_{reduced}$ with size $N_{jam} \times (N_{ant} - 1)$.

$$\begin{bmatrix} e^{i\frac{2\pi}{\lambda_1}\vec{a}_2\cdot\vec{j}_1} & \cdots & e^{i\frac{2\pi}{\lambda_1}\vec{a}_{N_{ant}}\cdot\vec{j}_1} \\ e^{i\frac{2\pi}{\lambda_2}\vec{a}_2\cdot\vec{j}_2} & \cdots & e^{i\frac{2\pi}{\lambda_2}\vec{a}_{N_{ant}}\cdot\vec{j}_2} \\ \vdots & \ddots & \vdots \\ e^{i\frac{2\pi}{\lambda_{N_{jam}}}\vec{a}_2\cdot\vec{j}_{N_{jam}}} & \cdots & e^{i\frac{2\pi}{\lambda_{N_{jam}}}\vec{a}_{N_{ant}}\cdot\vec{j}_{N_{jam}}} \end{bmatrix} \begin{bmatrix} A_2 \\ \vdots \\ A_{N_{ant}} \end{bmatrix} = C_{reduced} \begin{bmatrix} A_2 \\ \vdots \\ A_{N_{ant}} \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ \vdots \\ -1 \end{bmatrix}. \tag{9}$$

A criterion for the existence of an oracle solution can be stated as:

> Antenna weights annihilating the jammer output power exist if and only if the vector $[-1 \ -1 \cdots -1]^T$ lies in the column space of $C_{reduced}$.

When the number of jammers equals the number of peripheral antennas, $N_{jam} = N_{ant} - 1$, $C_{reduced}$ is square. If, further, $\text{rank}(C_{reduced}) = N_{ant} - 1$, the *unique* exact antenna weights

are

$$\begin{bmatrix} A_2 \\ A_3 \\ \vdots \\ A_{N_{ant}} \end{bmatrix} = -C_{reduced}^{-1} \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}. \tag{10}$$

If the number of jammers is less than the number of peripheral antennas, $N_{jam} < N_{ant} - 1$, the system of equations (9) is underdetermined. In this case, if the rank of $C_{reduced}$ is full, $\text{rank}(C_{reduced}) = N_{jam}$, the system has an infinite number of solutions. For the case when there are more jammers than peripheral antennas, $N_{jam} > N_{ant} - 1$, the array is asked to form N_{jam} nulls with only $N_{ant} - 1 < N_{jam}$ adjustable antenna weights. This results in an overdetermined system of equations (9) with the possibility–or rather, likelihood–of no solution for the antenna weights.

(Of course a least squares solution to equation (9) of minimum norm [2] can always be expressed using the pseudoinverse of $C_{reduced}$ [3], written as $C_{reduced}^{-\psi}$, as

$$\begin{bmatrix} A_2 \\ A_3 \\ \vdots \\ A_{N_{ant}} \end{bmatrix} = -C_{reduced}^{-\psi} \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}.) \tag{11}$$

The following is a simple example of an $N_{ant} = 3$ antenna array receiving $N_{jam} = 2$ jammers, ($N_{jam} = N_{ant} - 1$). With 2 adjustable antenna weights, it seems reasonable to expect that 2 nulls can be formed to attenuate 2 jammers. The example will illustrate that this is not always true.

The antennas are located at $\vec{a}_1 = 0,\ \vec{a}_2 = r\hat{i},\ \vec{a}_3 = \frac{1}{2}r\hat{i} + \frac{\sqrt{3}}{2}r\hat{j}$ and the jammer arrival directions are $\vec{j}_1 = \frac{\sqrt{3}}{2}\hat{i} + \frac{1}{2}\hat{j}$ and $\vec{j}_2 = -\frac{\sqrt{3}}{2}\hat{i} - \frac{1}{2}\hat{j}$, see figure 4. The jammer frequencies are

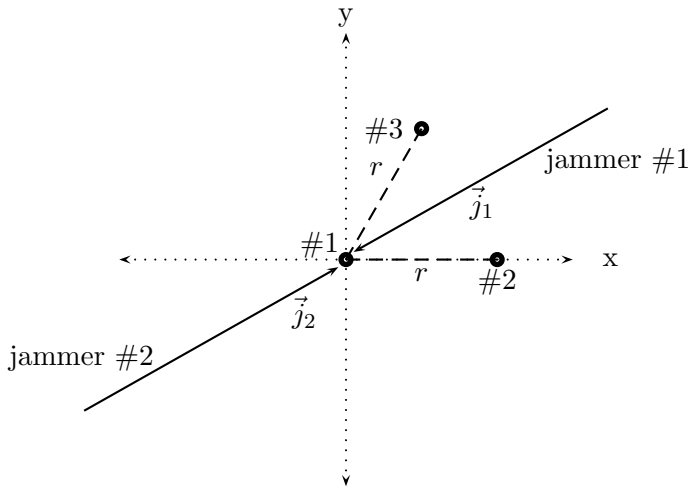

Figure 4. Confounding Configuration Example.

the same, $\omega_1 = \omega_2$. The antenna weights will null the output, from equation (8), if

$$\begin{bmatrix} 1 & e^{i\frac{2\pi}{\lambda}\vec{a}_2\cdot\vec{j}_1} & e^{i\frac{2\pi}{\lambda}\vec{a}_3\cdot\vec{j}_1} \\ 1 & e^{i\frac{2\pi}{\lambda}\vec{a}_2\cdot\vec{j}_2} & e^{i\frac{2\pi}{\lambda}\vec{a}_3\cdot\vec{j}_2} \end{bmatrix} \begin{bmatrix} 1 \\ A_2 \\ A_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

Substituting in the values for $\vec{a}_m$ and $\vec{j}_n$,

$$C_{reduced} = \begin{bmatrix} e^{i\frac{2\pi}{\lambda}\frac{\sqrt{3}}{2}r} & e^{i\frac{2\pi}{\lambda}\frac{\sqrt{3}}{2}r} \\ e^{-i\frac{2\pi}{\lambda}\frac{\sqrt{3}}{2}r} & e^{-i\frac{2\pi}{\lambda}\frac{\sqrt{3}}{2}r} \end{bmatrix}.$$

Let $e^{i\beta} = e^{i\frac{2\pi}{\lambda}\frac{\sqrt{3}}{2}r}$ and equation (9) becomes

$$\begin{bmatrix} e^{i\beta} & e^{i\beta} \\ e^{-i\beta} & e^{-i\beta} \end{bmatrix} \begin{bmatrix} A_2 \\ A_3 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}.$$

Rewrite the left hand side of this equation as

$$A_2 \begin{bmatrix} e^{i\beta} \\ e^{-i\beta} \end{bmatrix} + A_3 \begin{bmatrix} e^{i\beta} \\ e^{-i\beta} \end{bmatrix} = (A_2 + A_3) \begin{bmatrix} e^{i\beta} \\ e^{-i\beta} \end{bmatrix}.$$

But the right hand side of the equation is

$$\begin{bmatrix} -1 \\ -1 \end{bmatrix}.$$

$\left[e^{i\beta}\, e^{-i\beta}\right]^T$ is a scalar multiple of $[-1\ -1]^T$ if and only if $e^{i\beta} = e^{-i\beta}$, that is, $e^{i2\beta} = 1$, $\beta = k\pi$, where k is an integer. Thus, these jammers can be nulled only if $\beta = \frac{2\pi}{\lambda}\frac{\sqrt{3}}{2}r = k\pi$, that is, if the jammer wavelength λ equals $\sqrt{3}$ times the antenna separation, r, divided by an integer, i.e.

$$\lambda = \sqrt{3}r/k. \tag{12}$$

For this example, even though the number of jammers is equal to the number of peripheral antennas in the array, $N_{jam} = N_{ant} - 1$, making $C_{reduced}$ a square matrix, if the jammer wavelength does not satisfy equation (12) the vector $[-1\ -1]^T$ does not lie in the column space of $C_{reduced}$. The jammer power, therefore, cannot be nulled for this configuration.

We propose that whenever a prescribed configuration of jammers and antennas causes any adaptive power-minimizing algorithm to diverge, the configuration should be tested for an oracle solution before the algorithm is "blamed". If the configuration has no oracle solution, it should be removed from the test set in assessing the performance of the adaptive algorithm, because *no* algorithm could possibly give satisfactory performance.

Whenever an oracle solution *does* exist, on the other hand, it completely nulls the jammer power. Thus, it provides a standard against which other algorithms can be measured.

To summarize, if an adaptive algorithm fails to perform as expected, the oracle solution analysis can perhaps provide a potential clue by exposing a confounding configuration. The oracle solution, if it exists, also provides an evaluation of the effectiveness of an adaptive algorithm solution.

3.3.2 Condition Number

When no oracle solution exists, the configuration of jammer arrival directions and antenna locations is labeled a confounding configuration. However, there exist configurations of jammers and antennas for which there is an oracle solution but it is numerically unstable; these such cases are called near-confounding configurations. A metric used during research of confounding configurations was the condition number of $C_{reduced}$ which tests the numerical robustness of an oracle solution. The condition number measures sensitivity to error [16] and it is calculated by taking the ratio of the largest singular value of the matrix to the smallest singular value. The higher the condition number of a matrix, the closer the matrix is to being singular [16]; a condition number of infinity indicates a matrix that is singular, one without full rank.

A confounding configuration can result from a singular $C_{reduced}$. For cases with $N_{jam} = N_{ant} - 1$ ($C_{reduced}$ square), if the condition number of $C_{reduced}$ is large, on the order of 1×10^4, the arrangement of jammers and antennas may confound the HSB gradient algorithm and is labeled a near-confounding configuration.

Therefore, the condition number of $C_{reduced}$ provides a useful indication as to whether any adaptive algorithm will be able to find optimal weights to null jammers. If the condition number is on the order of 1×10^4 or greater the HSB gradient algorithm may not produce viable weights and the antenna/jammer configuration producing the high condition number should be removed from the test set.

3.4 Finding Confounding Configurations

The previous section described how the oracle can be used to test whether given antenna/jammer configurations are confounding or near-confounding. This section examines some aspects of the oracle solution, exploring the ability to construct confounding antenna/jammer configurations. This may be useful for jamming an enemy when the configuration of the enemy's array is known.

The exact antenna weights A to annihilate incident jammers, the oracle solution, exist if the vector $[-1 \ -1 \ \cdots -1]^T$ lies in the column space of $C_{reduced}$

$$C_{reduced} \begin{bmatrix} A_2 \\ A_3 \\ \vdots \\ A_{N_{ant}} \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ \vdots \\ -1 \end{bmatrix}. \tag{9}$$

In this section, it is desired to find jammer arrival directions that defy an oracle solution, i.e., confounding configurations. One way, among many, to explore this starts with introducing the variable

$$\vec{v}_n = \frac{2\pi}{\lambda}\vec{j}_n \tag{13}$$

for more compact notation and factors the matrix $C_{reduced}$ as

$$C_{reduced} = C'_{reduced}\Lambda$$

($'$ indicates prime, and does not indicate transpose) where

$$C'_{reduced} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ e^{i(\vec{v}_2-\vec{v}_1)\bullet\vec{a}_2} & e^{i(\vec{v}_2-\vec{v}_1)\bullet\vec{a}_3} & \cdots & e^{i(\vec{v}_2-\vec{v}_1)\bullet\vec{a}_{N_{ant}}} \\ e^{i(\vec{v}_3-\vec{v}_1)\bullet\vec{a}_2} & e^{i(\vec{v}_3-\vec{v}_1)\bullet\vec{a}_3} & \cdots & e^{i(\vec{v}_3-\vec{v}_1)\bullet\vec{a}_{N_{ant}}} \\ \vdots & \vdots & \ddots & \vdots \\ e^{i(\vec{v}_N-\vec{v}_1)\bullet\vec{a}_2} & e^{i(\vec{v}_N-\vec{v}_1)\bullet\vec{a}_3} & \cdots & e^{i(\vec{v}_N-\vec{v}_1)\bullet\vec{a}_{N_{ant}}} \end{bmatrix}$$

and

$$\Lambda = \begin{bmatrix} e^{i\vec{v}_1\bullet\vec{a}_2} & 0 & 0 & \cdots & 0 \\ 0 & e^{i\vec{v}_1\bullet\vec{a}_3} & 0 & \cdots & 0 \\ 0 & 0 & e^{i\vec{v}_1\bullet\vec{a}_4} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & e^{i\vec{v}_1\bullet\vec{a}_{N_{ant}}} \end{bmatrix}.$$

(Note that the vectors $\vec{v}$ have units of inverse meters, (m^{-1}) and the vectors $\vec{a}$ have unit of meters.) So

$$C_{reduced}\begin{bmatrix} A_2 \\ A_3 \\ \vdots \\ A_{N_{ant}} \end{bmatrix} = C'_{reduced}\Lambda \begin{bmatrix} A_2 \\ A_3 \\ \vdots \\ A_{N_{ant}} \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ \vdots \\ -1 \end{bmatrix},$$

and now the criterion for existence of a solution is that $[-1\ -1\ \cdots, -1]^T$ must lie in the column space of $C'_{reduced}$.

Note that if $C'_{reduced}$ takes the form

$$C'_{reduced} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \cdots & \vdots \\ \gamma & \gamma & \cdots & \gamma \\ \vdots & \vdots & \cdots & \vdots \end{bmatrix},$$

i.e., has a row of identical entries (besides the first row), with $\gamma \neq 1$, then the columns cannot span the vector $[-1\ \ -1\ \ -1 \cdots -1]^T$. So if a configuration of jammers is chosen such that all entries of, say, the second row of $C'_{reduced}$ are equal but different from 1, then there are no solutions. We explore this possibility.

The second row of $C'_{reduced}$ will have equal entries if

$$(\vec{v}_2 - \vec{v}_1) \bullet \vec{a}_2 = (\vec{v}_2 - \vec{v}_1) \bullet \vec{a}_3 + \alpha_3 2\pi = \cdots (\vec{v}_2 - \vec{v}_1) \bullet \vec{a}_{N_{ant}} + \alpha_{N_{ant}} 2\pi, \tag{14}$$

where $\alpha_3, \cdots, \alpha_{N_{ant}}$ are integers. In order to prevent $\gamma = e^{i(\vec{v}_2 - \vec{v}_1)\bullet \vec{a}_2} = 1$ we require

$$(\vec{v}_2 - \vec{v}_1) \bullet \vec{a}_2 \neq \alpha 2\pi \tag{15}$$

for any integer α. Note that equations (14) and (15) are conditions only on the *difference* $\vec{v}_2 - \vec{v}_1$, if there is one solution, there are infinitely many.

Now, consider a specific example of finding confounding jammers for a given array. The array from figure 4 has $\vec{a}_1 = 0, \vec{a}_2 = \hat{i}$, and $\vec{a}_3 = \frac{1}{2}\hat{i} + \frac{\sqrt{3}}{2}\hat{j}$ and we will confound it with 2 jammers. Let $\vec{v}_2 - \vec{v}_1 = \kappa\hat{i} + \nu\hat{j}$. Then

$$(\vec{v}_2 - \vec{v}_1) \bullet \vec{a}_2 = \kappa,$$

$$(\vec{v}_2 - \vec{v}_1) \bullet \vec{a}_3 = (\kappa\hat{i} + \nu\hat{j}) \bullet (\frac{1}{2}\hat{i} + \frac{\sqrt{3}}{2}\hat{j}) = \frac{\kappa}{2} + \frac{\sqrt{3}\nu}{2}.$$

To satisfy equation (14), set

$$(\vec{v}_2 - \vec{v}_1) \bullet \vec{a}_3 + \alpha_3 2\pi = \frac{\kappa}{2} + \frac{\sqrt{3}\nu}{2} + \alpha_3 2\pi = (\vec{v}_2 - \vec{v}_1) \bullet \vec{a}_2 = \kappa$$

and solve for ν

$$\nu = \frac{1}{\sqrt{3}}(\kappa + 4\pi\alpha_3). \tag{16}$$

Thus

$$\vec{v}_2 - \vec{v}_1 = \kappa\hat{i} + \frac{1}{\sqrt{3}}(\kappa + 4\pi\alpha_3)\hat{j}. \tag{17}$$

Pick $\alpha_3 = 1$ and $\kappa = 1$, then from equation (16), $\nu = 7.8325$ and

$$\vec{v}_2 - \vec{v}_1 = \hat{i} + 7.8325\hat{j}.$$

Since the direction and amplitude of $\vec{v}_1$ is not determined, pick $|\vec{v}_1| = 32.987\text{m}^{-1}$ and choose the direction as $\vec{v}_1 = 16\hat{i} + 28.85\hat{j}$. Then $\vec{v}_2$ is, from equation (17), $\vec{v}_2 = -15\hat{i} - 21.0175\hat{j}$ and its magnitude is $|\vec{v}_2| = 25.82\text{m}^{-1}$.

The wavelength of jammer #1 is

$$\lambda_1 = 2\pi/|\vec{v}_1| = 0.19047\,\text{m} \tag{18}$$

and the wavelength of jammer #2 is

$$\lambda_2 = 2\pi/|\vec{v}_2| = 2\pi/|\vec{v}_1 + \kappa\hat{i} + \frac{1}{\sqrt{3}}(\kappa + 4\pi\alpha_3)\hat{j}| = 0.243\,\text{m}. \tag{19}$$

The frequencies corresponding to the wavelengths of these 2 jammers that will confound the 3 element equilateral triangle antenna array are $f_1 = 1.575$ GHz and $f_2 = 1.23$ GHz, which are close to the wavelengths of the GPS transmissions.

Another example of an array that can be confounded with 2 jammers is the centered square array. The antenna location vectors are $\vec{a}_1 = 0, \vec{a}_2 = \hat{i}, \vec{a}_3 = \hat{j}, \vec{a}_4 = -\hat{i}$, and $\vec{a}_5 = -\hat{j}$. Let $\vec{v}_2 - \vec{v}_1 = \kappa\hat{i} + \nu\hat{j}$. Notice that $(\vec{v}_2 - \vec{v}_1) \bullet \vec{a}_2 = \kappa$ and $(\vec{v}_2 - \vec{v}_1) \bullet \vec{a}_4 = -\kappa$ so equation (14) requires

$$\kappa = -\kappa + \alpha_4 2\pi = \alpha_4 \pi.$$

If α_4 is an even integer, this contradicts the requirement (equation (15)) that $(\vec{v}_2 - \vec{v}_1) \bullet \vec{a}_2$ cannot be an integer multiple of 2π; so choose κ to be an odd multiple of π:

$$\kappa = (2N_1 + 1)\pi,$$

where N_1 is any integer. Equation (14) also requires

$$(\vec{v}_2 - \vec{v}_1) \bullet \vec{a}_3 = \nu \quad \text{and} \quad (\vec{v}_2 - \vec{v}_1) \bullet \vec{a}_5 = -\nu$$

so

$$\begin{aligned} \nu + \alpha_3 2\pi &= -\nu + \alpha_5 2\pi \\ \nu &= (\alpha_5 - \alpha_3)\pi \\ &\equiv (2N_2 + 1)\pi \end{aligned}$$

where N_2 is any integer. The centered square array can be confounded by any 2 jammers related by $\vec{v}_2 - \vec{v}_1 = (2N_1 + 1)\pi\hat{i} + (2N_2 + 1)\pi\hat{j}$.

Arbitrarily choosing $N_1 = 1$ and $N_2 = 2$ results in $\kappa = 3\pi$ and $\nu = 5\pi$. Again, the direction and amplitude of $\vec{v}_1$ can be anything. Choosing $|\vec{v}_1| = 32.987$ m^{-1} and its direction as $\vec{v}_1 = 20\hat{i} + 26.23\hat{j}$ results in $\vec{v}_2 = 29.42\hat{i} + 41.94\hat{j}$ and $|\vec{v}_2| = 51.23$ m^{-1}.

The wavelengths of the jammers are

$$\lambda_1 = 2\pi/|\vec{v}_1| = 0.1905 \text{ m}$$

and

$$\lambda_2 = 2\pi/|\vec{v}_1 + (2N_1+1)\pi\hat{i} + (2N_2+1)\pi\hat{j}| = 0.1226 \text{ m}.$$

The frequencies of these jammers are $f_1 = 1.575$ GHz and $f_2 = 2.45$ GHz. Note that we have confounded a *five* antenna array with only *two* jammers.

This particular method of finding jammers to confound an array will not work if all the antenna location vectors are related by $\vec{a}_i - \vec{a}_j = \vec{a}_k$ for some i, j, k. An example of such an array is a hexagonal array with $\vec{a}_1 = 0, \vec{a}_2 = \hat{i}, \vec{a}_3 = \frac{1}{2}\hat{i} + \frac{\sqrt{3}}{2}\hat{j}, \vec{a}_4 = -\frac{1}{2}\hat{i} + \frac{\sqrt{3}}{2}\hat{j}, \vec{a}_5 = -\hat{i}, \vec{a}_6 = -\frac{1}{2}\hat{i} - \frac{\sqrt{3}}{2}\hat{j}$, and $\vec{a}_7 = \frac{1}{2}\hat{i} - \frac{\sqrt{3}}{2}\hat{j}$. For this array, $\vec{a}_3 - \vec{a}_4 = \vec{a}_2$, $\vec{a}_4 - \vec{a}_3 = \vec{a}_5$, $\vec{a}_3 - \vec{a}_2 = \vec{a}_4$, etc. Substituting one of these, for example $\vec{a}_2 = \vec{a}_3 - \vec{a}_4$, into equation (14) and rearranging terms results in

$$-(\vec{v}_2 - \vec{v}_1) \bullet \vec{a}_4 = \alpha_3 2\pi = (\vec{v}_2 - \vec{v}_1) \bullet (\vec{a}_4 - \vec{a}_3) + \alpha_4 2\pi = \cdots = (\vec{v}_2 - \vec{v}_1) \bullet (\vec{a}_{N_{ant}} - \vec{a}_3) + \alpha_{N_{ant}} 2\pi$$

but this leaves the term $(\vec{v}_2 - \vec{v}_1) \bullet \vec{a}_4$ equal to an integer multiple of 2π. This violates the condition of equation (15) defeating the objective of finding a confounding configuration.

Summarizing, note that the HSB gradient algorithm as well as other algorithms focus on *minimizing* the total power output from the array, $|[\chi_{GPS} + N + JC]A|^2$, while the oracle focuses on *zeroing* the term CA. So if the oracle solution exists and χ_{GPS} and N are small, it is a reasonable standard for assessing the HSB gradient algorithm; if the oracle solution does not exist then the goal of the HSB gradient algorithm may be unachievable.

The next chapter describes the HSB gradient adaptive algorithm that computes antenna weights which seek to minimize the output power of an array. A comparison between the HSB gradient weights and the oracle weights is made by comparing, at each step of the simulation, the array output powers resulting from each set of weights. The comparison

shows that the HSB algorithm exhibits excellent performance in minimizing the output power of an array.

Chapter 4

Hilbert-Space-Based (HSB) Gradient Algorithm

This chapter will describe the theory of the HSB gradient algorithm. The objective of this algorithm is to minimize the output power of the array. The signals of interest received from the GPS satellite network are spread spectrum signals of negligible power, about the same as the background noise level. Reducing the array output power to this level will allow GPS signals to be extracted.

4.1 Power Minimization and Weight Calculation

At time t_γ, the output of antenna $\#m$ consisting of GPS signals, noise, and N_{jam} jammers is

$$A_m \chi_{GPS}^{(m)}(t_\gamma) + A_m N_m(t_\gamma) + \sum_{n=1}^{N_{jam}} J_n e^{i\omega_n t_\gamma}\ e^{i\frac{2\pi}{\lambda_n}\vec{a}_m \bullet \vec{j}_n}\ A_m$$

where $N_m(t_\gamma)$ conceptually represents a random noise component in antenna $\#m$. The details of the nature of the random noise in the array are discussed in section 4.3.

The array output is the sum of all antenna outputs

$$\sum_{m=1}^{N_{ant}} A_m \chi_{GPS}^{(m)}(t_\gamma) + \sum_{m=1}^{N_{ant}} A_m N_m(t_\gamma) + \sum_{m=1}^{N_{ant}} \sum_{n=1}^{N_{jam}} J_n e^{i\omega_n t_\gamma}\ e^{i\frac{2\pi}{\lambda_n}\vec{a}_m \bullet \vec{j}_n}\ A_m$$

which can be expressed in matrix form, from (5), as

$$\chi_{GPS} A + NA + JCA = xA. \tag{20}$$

The power minimization and weight calculations will proceed as follows. For some initial choice of weights A, the output power of the array is sampled every τ seconds and assessed at times labeled by the variable $t_\gamma = \gamma\tau$ (γ is an integer), Γ of these power measurements

are averaged every $\Gamma\tau$ seconds, and the gradient of this average (with respect to the weights) is calculated. The peripheral antenna weights are adjusted in the direction of the negative gradient. This constitutes one algorithm iteration. It is unlikely that one single adjustment in each weight value results in their optimal values so these updated weights are used to accumulate another batch of Γ output power measurements, and the algorithm is repeated. If the procedure converges, the search becomes more precise, with finer adjustments in the weight values toward their optimal values, as the algorithm progresses.

The instantaneous output power of the array, $\mathsf{p}(t_\gamma, A)$, is

$$\mathsf{p}(t_\gamma, A) = (xA)(xA)^\dagger \tag{21}$$

where † represents conjugate transpose. For each value of t_γ, $\mathsf{p}(t_\gamma, A)$ represents one sample of the instantaneous array output power.

During the kth iteration of the algorithm, Γ samples of instantaneous power, $\mathsf{p}(t_\gamma, A)$, are averaged to obtain the average output power of the array, $P(k, A)$, i.e.,

$$P(k, A) = \frac{1}{\Gamma} \sum_{t_\gamma=(k-1)\Gamma+1}^{k\Gamma} \mathsf{p}(t_\gamma, A). \tag{22}$$

The average output power is a quadratic function of the antenna weights. To help picture this power function, consider an array of 3 antennas, all with real weights only, in any configuration (2 peripheral antennas with adjustable weights and 1 reference antenna with a fixed weight). The output power of the array can be plotted along the z-axis as a function of the adjustable antenna weights represented along the x and y axes. The power surface is shaped like a bowl with a single, unique minimum. The antenna weights that correspond to this minimum are those that the algorithm seeks. The strategy is to converge to the minimum by continuously adjusting the values for the antenna weights so as to enforce $P(k+1, A) < P(k, A)$.

Recall, from Chapter 2, A is a $1 \times N_{ant}$ vector with components $w_m e^{iq_m}$, $m = 1, 2, \ldots, N_{ant}$. The power $P(k, A)$ can be expressed explicitly in terms of the magnitudes, w_m, and phases,

q_m, of the antenna weights by reincarnating $A_m = w_m e^{iq_m}$ as a new variable W, where W is a column of antenna weight parameters $[w_1\ w_2 \cdots w_{N_{ant}}\ q_1\ q_2 \cdots q_{N_{ant}}]^T$ (recall since $A_1 \equiv 1$, $w_1 = 1$ and $q_1 = 0$). The objective of the algorithm is to choose the change in the weights, ΔW_m, to reduce $P(k, W_m)$:

$$P(k+1, W_m + \Delta W_m) \leq P(k, W_m), \qquad m = 1, 2, \ldots, 2N_{ant}.$$

Without risk of confusion, we write $P(W)$ for $P(k, A)$. Expanding $P(W_m + \Delta W_m)$ with a (matrix) Taylor series to first order yields

$$P(W_m + \Delta W_m) \approx P(W_m) + \Delta W^T \frac{\partial P}{\partial W} \equiv P(W_m) + \Delta W^T \nabla P \leq P(W_m), \tag{23}$$

where W is the column of weight parameters introduced above and ∇P represents the column of all components of the gradient of the output power with respect to the antenna weight parameters, $\nabla P = [\frac{\partial P}{\partial w_1}, \frac{\partial P}{\partial w_2}, \ldots, \frac{\partial P}{\partial w_{N_{ant}}}, \frac{\partial P}{\partial q_1}, \frac{\partial P}{\partial q_2}, \ldots, \frac{\partial P}{\partial q_{N_{ant}}}]^T$.

For the steepest descent algorithm, the weights from iteration k to iteration $k+1$ are adjusted in the direction of the negative power gradient with a scale factor μ, the "step size"

$$\Delta W = -\mu \nabla P. \tag{24}$$

Substituting equation (24) for ΔW^T in equation (23) results in

$$\Delta P = P(W + \Delta W) - P(W) \approx -\mu \nabla P^T \nabla P. \tag{25}$$

The *desired* change in power is known; it is the difference between the present power, $P(k, A) = P(W)$ and the minimum power, which is approximately zero.

$$\Delta P = \text{change in power} = (0 - P(W)) = -\mu \nabla P^T \nabla P$$

and solving for the step size μ, one obtains

$$\mu = \frac{P(W)}{\nabla P^T \nabla P}. \tag{26}$$

The weight adjustment equation (24) becomes

$$\Delta W = -\mu \nabla P = \frac{-P(W)}{\nabla P^T \nabla P} \nabla P. \tag{27}$$

Equation (27) in terms of the weight magnitudes states

$$\Delta w_i = -\mu \frac{\partial P}{\partial w_i} = -P(W) \times \left(\frac{\partial P / \partial w_i}{\sum_{m=2}^{N_{ant}} \left[(\frac{\partial P}{\partial w_m})^2 + (\frac{\partial P}{\partial q_m})^2 \right]} \right), \; i = 2, 3, \ldots, N_{ant} \tag{28}$$

and in terms of the weight phases it states

$$\Delta q_i = -\mu \frac{\partial P}{\partial q_i} = -P(W) \times \left(\frac{\partial P / \partial q_i}{\sum_{m=2}^{N_{ant}} \left[(\frac{\partial P}{\partial w_m})^2 + (\frac{\partial P}{\partial q_m})^2 \right]} \right), \; i = 2, 3, \ldots, N_{ant}. \tag{29}$$

These weight change calculations occur once per algorithm iteration, each time with updated values for μ and ∇P. Our innovation for calculating the gradient components, $\frac{\partial P}{\partial w_m}$ and $\frac{\partial P}{\partial q_m}$, will be described in the next section.

If the array is in motion, the optimal weight search performed by the algorithm becomes nonstationary because the power surface changes at each iteration. The power surface is a function of the phase matrix C which is a function of the changing jammer arrival directions, $\vec{j}_n$. For an array in motion, the power is time averaged and each weight is updated per algorithm iteration as it is for a stationary array.

4.2 Exact Gradient Calculation

This section will describe the computation of the gradient of the average array output power. What distinguishes the HSB gradient algorithm from all other gradient based adaptive algorithms is that it calculates the *exact* gradient components, *not approximations* of the gradient components. Furthermore, the algorithm calculates the gradient components simultaneously and efficiently.

The power gradient, ∇P, is computed using the Hilbert space inner products of signals which are readily available at the array output by the following method [5]. From equations (21) and (22),

$$P(k,A) = \frac{1}{\Gamma} \sum_{t_\gamma=(k-1)\Gamma+1}^{k\Gamma} \mathsf{p}(t_\gamma, A) = \frac{1}{\Gamma} \sum_{t_\gamma=(k-1)\Gamma+1}^{k\Gamma} (xA)(xA)^\dagger.$$

Rewrite this summation using inner product bra/ket notation

$$< a, b >= \frac{1}{T} \sum_{t=(k-1)T+1}^{kT} a(t)b^\dagger(t)$$

and the power is represented as

$$P(k,A) =< xA, xA > . \tag{30}$$

Since the array output results from the N_{ant} antenna outputs,

$$xA = \sum_{m=1}^{N_{ant}} A_m x_m(k), \tag{31}$$

the array output power becomes

$$P(k,A) =< \sum_{g=1}^{N_{ant}} A_g x_g(k), \sum_{g=1}^{N_{ant}} A_g x_g(k) >= \sum_{g=1}^{N_{ant}} \sum_{h=1}^{N_{ant}} A_g A_h^* < x_g(k), x_h(k) > . \tag{32}$$

Now the mth component of the gradient with respect to the weight magnitude w_m is (the time dependence of x is dropped for clarity) (recall $A_g = w_g e^{iq_g}$)

$$\begin{aligned}\frac{\partial P}{\partial w_m} &= \frac{\partial(\sum_{g=1}^{N_{ant}}\sum_{h=1}^{N_{ant}} A_g A_h^* < x_g, x_h >)}{\partial w_m} \\ &= \sum_g \sum_h < x_g, x_h > A_g \frac{\partial w_h e^{-iq_h}}{\partial w_m} + \sum_g \sum_h < x_g, x_h > \frac{\partial w_g e^{iq_g}}{\partial w_m} A_h^*.\end{aligned}$$

On the right hand side of this equation, the partial derivative with respect to w_m is non-zero only when $h = m$ for the first term and only when $g = m$ for the second term:

$$\begin{aligned}\tfrac{\partial P}{\partial w_m} &= \sum_g \sum_h < x_g, x_h > A_g \delta_{hm} e^{-iq_m} + \sum_g \sum_h < x_g, x_h > \delta_{gm} e^{iq_m} A_h^* \\ &= \sum_g < x_g, x_m > A_g e^{-iq_m} + \sum_h < x_m, x_h > e^{iq_m} A_h^*.\end{aligned}$$

Rearranging terms and identifying the summation of the antenna outputs as the array output signal, xA from equation (31), yields

$$\begin{aligned}\tfrac{\partial P}{\partial w_m} &= \sum_g < A_g x_g, e^{iq_m} x_m > + \sum_h < e^{iq_m} x_m, A_h x_h > \\ &= < xA, A_m x_m > / w_m + < A_m x_m, xA > / w_m.\end{aligned}$$

The terms in the last line are complex conjugates of each other. A quantity added to its conjugate leaves only twice the real part

$$\frac{\partial P}{\partial w_m} = \frac{2\Re < A_m x_m, xA >}{w_m}. \tag{33}$$

The components of the gradient with respect to the weight magnitudes are simply the Hilbert space inner products of the real parts of the appropriately scaled (by $\frac{1}{w_m}$) individual output signals of each antenna $A_m x_m(k)$ with the array signal output, $x(k)A$. That is, they are correlations (at zero delay).

Similarly, the mth component of the gradient with respect to the phase q_m of the weight is

$$\frac{\partial P}{\partial q_m} = \frac{\partial(\sum_g^{N_{ant}} \sum_h^{N_{ant}} A_g A_h^* < x_g, x_h >)}{\partial q_m} \tag{34}$$

$$= \sum_g \sum_h < x_g, x_h > A_g \frac{\partial w_h e^{-iq_h}}{\partial q_m} + \sum_g \sum_h < x_g, x_h > \frac{\partial w_g e^{iq_g}}{\partial q_m} A_h^*. \tag{35}$$

On the right hand side of this equation, the partial derivative with respect to q_m is non-zero only when $h = m$ for the first term and only when $g = m$ for the second term:

$$\begin{aligned} \frac{\partial P}{\partial q_m} &= \sum_g \sum_h < x_g, x_h > A_g \delta_{hm} w_h (-i) e^{-iq_h} \\ &\quad + \sum_g \sum_h < x_g, x_h > \delta_{gm} w_g (i) e^{iq_g} A_h^* \\ &= -i \sum_g < x_g, x_m > A_g A_m^* + i \sum_h < x_m, x_h > A_m A_h^*. \end{aligned}$$

Rearranging terms and identifying the summation of the antenna outputs as the array output signal, xA from equation (31), yields

$$\begin{aligned} \frac{\partial P}{\partial q_m} &= -i \sum_g < A_g x_g, A_m x_m > + i \sum_h < A_m x_m, A_h x_h > \\ &= -i < xA, A_m x_m > + i < A_m x_m, xA > \\ &= 2\Im < A_m x_m, xA > . \end{aligned} \tag{36}$$

The components of the gradient with respect to the weight phases are simply the Hilbert space inner products of the imaginary parts of the scaled individual output signals of each antenna $A_m x_m(k)$ with the array signal output, $x(k)A$.

With all components of the gradient available, they are then used to determine the antenna weights to minimize the output power from the array as in equations (28) and (29).

To reiterate, the components of the gradient are found using the equations

$$\frac{\partial P}{\partial w_m} = \frac{2\Re < A_m x_m, xA >}{w_m}$$

and

$$\frac{\partial P}{\partial q_m} = 2\Im < A_m x_m, xA > .$$

These components, again, are *not approximations* of the gradient, they are the *exact* components of the gradient. The estimates of the gradient used by other algorithms only approximate the gradient. Algorithms that use finite differences to approximate the gradient lose accuracy, particularly near the minimum of the quadratic surface, and take multiple time steps to evaluate, since each term in the finite difference requires a separate power average. The simultaneous, efficient, exact calculation of all components of the gradient by the method proposed herein leads to faster, more robust jammer cancellation performance.

Although we do not exploit it in this study, it is of interest to note that these inner products can be evaluated using power averages alone, for judiciously chosen signal combinations

$$\frac{\partial P}{\partial w_m} = \frac{1}{w_m} \{ < A_m x_m + xA, A_m x_m + xA > - < A_m x_m, A_m x_m > - < xA, xA > \}$$

and

$$\frac{\partial P}{\partial q_m} = -i \{ < A_m x_m + ixA, A_m x_m + ixA > - < A_m x_m, A_m x_m > - < xA, xA > \} .$$

4.3 Noise Simulation

Up to now this chapter has focused on the implementation of the algorithm; this section will describe details about noise in the simulation of the algorithm. Noise is added as random 'input' to the array; it is subject to scaling by the antenna weights as are the jammer and GPS input signals.

The objective of the HSB gradient algorithm is to minimize the output power of the array. During each iteration, noise is simulated as random samples and input to each antenna. These samples are added to the jammer and GPS input signals and these composite signals are scaled by the antenna weights and combined to form the array output.

Referring to equation (20), the sample at time t_γ of the array output signal is modeled as

$$\chi_{GPS}A + NA + JCA = xA.$$

The noise term in this equation is

$$NA = \sum_{m=1}^{N_{ant}} N_m(t_\gamma)A_m$$

where at each time step $N_m(t_\gamma)$ is modeled as an independent sample of a random variable with a Gaussian distribution of zero mean and variance σ^2. The power in this noise at the *output* of the array must equal the (specified) background noise level, ρ. It is known that the sum of N_{ant} random variables, each with the same mean, μ, and variance, σ^2, has mean equal to $N_{ant}\mu$ and variance equal to $N_{ant}\sigma^2$ [8]. Setting the standard deviation of each sample of noise at each antenna equal to $\sqrt{\rho/N_{ant}}$ fixes the power level of the noise at the output of the array to match that of the background noise.

4.4 Differences in Algorithm and Oracle Solution

A direct comparison of the antenna weights of the HSB gradient algorithm with another algorithm is not possible because the antenna weights approach their optimal values via different paths, depending on the details of the adaptive algorithm. However, as discussed in Chapter 3, the results of any iteration of the antenna weights of the HSB gradient algorithm can be compared with the oracle weights for the corresponding antenna position. Some difference between the algorithm and oracle weights is expected because the oracle provides the exact weights annihilating only the output power component due to jammers, $|J\ C\ A|^2$, while the algorithm weights minimize the output power of the array due to all inputs, $|\chi_{GPS}A + N\ A + J\ C\ A|^2$.

The HSB gradient algorithm and the oracle also differ in their dependence on the jammer amplitudes. The HSB gradient algorithm uses the jammer amplitude information in finding the solution for the antenna weights. The jammer amplitudes and arrival directions affect

the output signal of the array, the output power of the array, the gradient, and the weight update equation. Every antenna in the array is affected by every jammer arriving at the array. A weak jammer cannot remove or reduce the effect that a particular antenna weight has in the array nor can a strong jammer cause a particular antenna weight to dominate the algorithm solution. All jammers affect all antenna weights. What determines the algorithm's solution for effective power minimizing antenna weights is how close the search of the power surface in the negative gradient direction approaches the minimum.

The oracle solution does not take into account any information about the amplitudes of the jammers. The oracle solves $C\ A = 0$ for the optimal weights to annihilate the array output. The jammer amplitudes, J, do not affect the values of the oracle antenna weights. The arrival directions of the jammers, however, are critical to the solution. They are used in the elements of C.

Chapter 5

Matlab Implementation

This chapter will describe the implementation of the HSB gradient algorithm in Matlab. Since the theory of the algorithm has already been presented in the previous chapter, this chapter is included only for readers who are interested in coding details. The results of the simulations are reported in Chapter 6.

The input to the array is modeled as high power jammers and low power noise. The GPS signals are not modeled explicitly, although they would appear similar to the noise. The algorithm does not perform any demodulation of the GPS signals. The HSB gradient algorithm is implemented in Matlab with 3 files: `setup.m`, `adapt.m` and `motion.m`.

5.1 setup.m

The file `setup.m` defines and initializes constants necessary for the adaptive algorithm implementation. See Table 1 for a list of these constants.

`Setup.m` also defines variables that are initialized by the user. Table 2 lists these variables along with typical values they might be assigned.

The peripheral antennas in the array can be arranged in any configuration around a central antenna. Here the antennas are arranged in a plane with the origin of the reference coordinate system coinciding with the central antenna. The peripheral antenna locations are measured counterclockwise from the x-axis at angles $\gamma_m,\ m = 2, 3, \ldots, N_{ant}$. Each jammer arrives at the array with an azimuth angle ϕ measured counterclockwise from the x-axis and with an elevation angle θ measured down from the z-axis.

Table 1. Constants in `setup.m`.

Variable Name	Value	Description
`deg_to_rad`	$\pi/180$	degrees to radians conversion constant
`jmath`	$\sqrt{-1}$	imaginary number definition
`A`	$2 \times$ `Nant` matrix, all row 1 entries = 1, all row 2 entries = 0	initial antenna weight magnitudes and phases
`a`	$1 \times$ `Nant`	complex weight values
`Ann`	$2 \times$ `Nant`	initial antenna weight magnitudes and phases for no noise case
`ann`	$1 \times$ `Nant`	complex weight values for no noise case
`speed_of_light`	3x10^8 m/s	speed of light constant
`P_int`	20x10^{-6} sec	power integral duration
`LOfreq`	1.575x10^9 Hz	local oscillator frequency
`omegaLO`	2π`LOfreq` rad/sec	local oscillator frequency
`jam_freq`	`LOfreq`	frequency of jammers
`omega_jam`	2π`jam_freq`	frequency of jammers
`motion_vector`	[`roll`,`pitch`,`yaw`]	vector of variables indicating array motion
`az`, `el`, `rr`		spherical coordinates of jammer arrival angles
`phi_jam`	`az`	azimuth jammer arrival angle, counterclockwise from x-axis
`theta_jam`	$(\pi/2)$-`el`	elevation jammer arrival angle, down from z-axis
`gamma`	$1 \times$ (`Nant`-1)	vector of angles describing antenna locations, counterclockwise from x-axis

The sampling rate of the jammer and noise signals is set to 20 MHz ($\tau = 50$ ns). Every iteration of the algorithm processes $\Gamma = 400$ samples or 20 μs of data. These 400 samples of instantaneous array output are averaged to form one sample of the average array output power per iteration. The variable `P_int` represents the duration of one iteration.

The phase matrix, C is calculated in `setup.m`. The matrix is made up of the exponential phase differences between the jammer wavefronts impinging on each peripheral antenna with respect to the center antenna in the array. C has size `Njams` $\times$ `Nant` (from Chapter 3, N_{jam}=`Njams` and N_{ant}=`Nant`) and the (n, m)th element has the form using the jammer arrival directions of ϕ and θ and antenna locations of γ

$$C_{nm} = e^{i\frac{2\pi}{\lambda}\vec{a}_m \bullet \vec{j}_n} = e^{i\frac{2\pi}{\lambda}\cos(\gamma_m - \phi_n)\sin\theta_n}$$

Table 2. Variables in `setup.m` Initialized by User.

Variable Name	Typical Value	Description
`Nant`	7	number of antennas in array
`array_radius`	0.1	radius of array in meters
`a_min`	0.1	minimum antenna weight magnitude
`a_max`	10	maximum antenna weight magnitude
`step_control`	1	fraction of gradient correction to be implemented
`noise_floor`	1.25 (1 = 0 dB)	target output power value
`Njams`	6	number of jammers incident on array
`theta_jam`	[90,90,90,90,90,90,-90,90,90]	vector of jammer elevation angles, reduced to 1 × `Njams`
`phi_jam`	[30,90,150,210,270,-330,15,75,135]	vector of jammer azimuth angles, reduced to 1 × `Njams`
`J`	[100,100,100,100,-100,100,100,100,100]	vector of jammer amplitudes, reduced to 1 × `Njams`
`sim_time`	0.5 sec	length of simulation, each iteration of algorithm is `P_int` long
`roll`	0	roll rate in deg/sec, $+y$ onto $+z$
`pitch`	0	pitch rate in deg/sec, $+z$ onto $-x$
`yaw`	360	yaw rate in deg/sec, $+y$ onto $+x$
`beta`	0	initial offset of array in degrees, `-roll`

where, in general, for a planar array

$$\vec{a}_m = \cos\gamma_m \hat{i} + \sin\gamma_m \hat{j},$$

$$\vec{j}_n = \cos\phi_n \sin\theta_n \hat{i} + \sin\phi_n \sin\theta_n \hat{j} + \cos\theta_n \hat{k}$$

and

$$\vec{a}_m \bullet \vec{j}_n = \cos(\gamma_m - \phi_n)\sin\theta_n. \tag{37}$$

5.2 adapt.m

The file `adapt.m` performs the power, gradient, and weight adjustment calculations. The algorithm adapts the antenna weights for the duration `sim_time`, a length of time entered by the user. Since each iteration of the algorithm covers `P_int`=20 μs, the number of weight adjustment iterations performed is `sim_time`/`P_int`. At the start of an iteration,

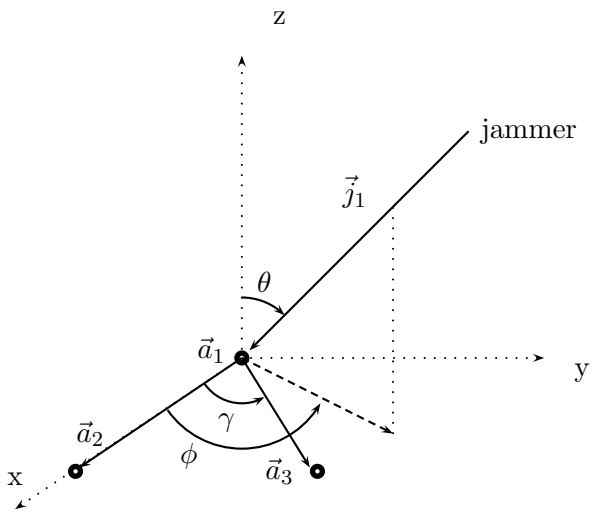

Figure 5. Example of $\vec{a}_m$ and $\vec{u}_n$.

`motion_vector` is checked for a nonzero value. If it is nonzero, the array has roll, pitch, and/or yaw motion and the file `motion.m` is called. This file updates the phase matrix C due to the array motion. Details of `motion.m` are described in the next section.

With the phase matrix components calculated either from `setup.m` or from `motion.m`, the oracle weights are computed as in equation (10)

$$\begin{bmatrix} A_2 \\ A_3 \\ \vdots \\ A_{N_{ant}} \end{bmatrix}_{or} = -C_{reduced}^{-1} \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

or (11)

$$\begin{bmatrix} A_2 \\ A_3 \\ \vdots \\ A_{N_{ant}} \end{bmatrix}_{or} = -C_{reduced}^{-\psi} \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

where the subscript *or* indicates *or*acle weights. With the oracle weights known, the output power corresponding to the oracle weights is calculated using equation (21),

$$\mathsf{p}_{oracle}(t_\gamma, A_{or}) = (xA_{or})(xA_{or})^\dagger.$$

The condition number of $C_{reduced}$ is calculated per iteration using the `cond` function in Matlab.

To calculate the gradient of the power surface, the array output signal and array output power must be calculated first. As stated in section 4.3, the array output signal may contain noise as well as jammers. The simulated output signal of the array is

$$N(t_\gamma)\ A + J(t_\gamma)\ C(t_\gamma)\ A = x(t_\gamma)\ A$$

where the initial values of antenna weights, A, declared in `setup.m`, are used for the first iteration.

During each iteration, 400 samples of array output are simulated, $t_\gamma = \tau, 2\tau, \ldots, 400\tau$. For each value of t_γ, the noise, $N(t_\gamma)$, is a $1 \times N_{ant}$ vector of white noise samples simulated with the `randn` Matlab function. The Matlab `randn` function generates zero mean, unit variance white noise samples. The variance of the noise must be modified as described in section 4.3 to simulate a certain background noise level in the output signal of the array. This is accomplished by multiplying each noise sample at each antenna by the scalar $\sqrt{\frac{\texttt{noise\_floor}}{\texttt{Nant}}}$. Combining all `Nant` antenna outputs results in a noise signal at the output of the array with variance equal to `noise_floor`. The 400 noise samples are added to as many jammer components to form the input to each antenna. These signals are scaled by the antenna weights then summed together to form the array output.

Note that the phase matrix C and the antenna weight vector A are each modified only once per iteration. A snapshot of the array and jammer arrival directions is taken at the start of every algorithm iteration and held constant for the duration of one iteration. Recall that the objective of the algorithm is to minimize the output *power* of the array to within 5 dB of the background noise level. Each of the 400 samples of instantaneous output power

per iteration is computed as

$$(xA)\ (xA)^*.$$

These 400 samples of instantaneous power are averaged to get the average output power of the array, P_{av}. This average power represents the power integrated over a `P_int` $= 20\mu$s interval, one iteration of the algorithm. Finally, the gradient components of the output power are computed as in equations (33) and (36),

$$\frac{\partial P}{\partial w_m} = \frac{2\Re < A_m x_m, xA >}{w_m}$$

and

$$\frac{\partial P}{\partial q_m} = 2\Im < A_m x_m, xA > .$$

The term $A_m x_m$ represents the average signal of 400 samples of output from the mth antenna, $A_m x_m = N\ A_m + J\ C\ A_m$ where N is the average noise signal for the current loop iteration. The term xA is the average output signal of the array. Both $A_m x_m$ and xA are scalars, they are multiplied together and scaled to form $2 < A_m x_m, xA >$. The gradient components are the real (scaled by the reciprocal of each weight magnitude) and imaginary parts of this scaled product. In the Matlab simulation, the gradient components occupy a $2 \times$ (`Nant`-1) matrix

$$\begin{bmatrix} \frac{\partial P}{\partial w_2} & \frac{\partial P}{\partial w_3} \cdots \frac{\partial P}{\partial w_{N_{ant}}} \\ \frac{\partial P}{\partial q_2} & \frac{\partial P}{\partial q_3} \cdots \frac{\partial P}{\partial q_{N_{ant}}} \end{bmatrix}.$$

The vector of antenna weights is adjusted using the steepest descent algorithm as follows. From equation (27), the amount of weight change is calculated as

$$\Delta W = -\mu \nabla P = \frac{(\text{target power} - P_{av})}{\nabla P^T \nabla P} \nabla P$$

where target power is taken to be the constant `noise_floor`. W represents the weights as a $2 \times (N_{ant} - 1)$ vector

$$\begin{bmatrix} w_2\ w_3 \cdots w_{N_{ant}} \\ q_2\ q_3 \cdots q_{N_{ant}} \end{bmatrix}.$$

Finally, the weight vector is updated as

$$W \leftarrow W + \Delta W.$$

This completes one iteration of the algorithm. The process repeats until the number of iterations determined by the user entered variable `sim_time`, has elapsed.

The algorithm is designed to reduce the array output power to within 5 dB of the target level, `noise_floor`, or less. If the output power ever exceeds this level, the average number of iterations it takes for the algorithm to return the output power to within 5 dB of the `noise_floor` level is reported on the output graphs of `adapt.m`.

The file `adapt.m` generates 3 output plots: the array output power versus time, the oracle power versus time, and the condition number of $C_{reduced}$ versus iteration number.

5.3 motion.m

If the array platform moves with roll, pitch, or yaw, the file `motion.m` is called from each iteration of `adapt.m` to update the phase matrix C. As the array rotates, the arrival angles of the jammers change from their original values.

The velocity vector of a jammer with respect to the array is

$$\vec{v} = \vec{\omega} \times \vec{R}$$

where $\vec{\omega}$ is the angular velocity of the array

$$\vec{\omega} = \texttt{roll}\,\hat{i} + \texttt{pitch}\,\hat{j} + \texttt{yaw}\,\hat{k}$$

and $\vec{R}$ is the vector describing the location of a jammer direction of arrival (ϕ represents azimuth arrival direction, θ represents elevation arrival direction) expressed in Cartesian coordinates with respect to the center of the array

$$\vec{R} = \sin\theta\cos\phi\hat{i} + \sin\theta\sin\phi\hat{j} + \cos\theta\hat{k}.$$

The components of $\vec{v}$ are then

$$\begin{aligned}\vec{\omega} \times \vec{R} &= (\texttt{pitch}\cos\theta - \texttt{yaw}\sin\theta\sin\phi)\hat{i} \\ &\quad + (\texttt{yaw}\sin\theta\cos\phi - \texttt{roll}\cos\theta)\hat{j} \\ &\quad + (\texttt{roll}\sin\theta\sin\phi - \texttt{pitch}\sin\theta\cos\phi)\hat{k}\end{aligned}$$

but the velocity of the array is also the change in time of the arrival directions of the jammers

$$\vec{v} = d\vec{R}/dt = -\sin\theta\sin\phi\dot{\phi}\hat{i} + \cos\phi\cos\theta\dot{\theta}\hat{i} + \sin\theta\cos\phi\dot{\phi}\hat{j} + \sin\phi\cos\theta\dot{\theta}\hat{j} - \sin\theta\dot{\theta}\hat{k}.$$

Evaluating the $\hat{k}$ components on both sides of $d\vec{R}/dt = \vec{\omega} \times \vec{R}$ leads to the expression for the change in the elevation angle due to roll, pitch, and yaw motion

$$\dot{\theta} = -\texttt{roll}\sin\phi + \texttt{pitch}\cos\phi. \tag{38}$$

Evaluating the $\hat{j}$ components on both sides of $d\vec{R}/dt = \omega \times \vec{R}$ leads to the expression for the change in the azimuth angle

$$\dot{\phi} = \texttt{yaw} - \texttt{roll}\cot\theta\cos\phi - \texttt{pitch}\cot\theta\sin\phi. \tag{39}$$

Equations (38) and (39) are the change in jammer arrival angles with respect to time. The file `motion.m` is called once per iteration of `adapt.m` and each iteration is `P_int` or 20 μseconds in duration. The amounts of change in the jammer arrival directions during an iteration are then $\dot{\phi}\cdot$`P_int` and $\dot{\theta}\cdot$`P_int`. The values of ϕ and θ are updated by the amounts

$$\phi \leftarrow \phi + \dot{\phi}\cdot\texttt{P\_int}$$

and

$$\theta \leftarrow \theta + \dot{\theta}\cdot\texttt{P\_int}.$$

These new values for ϕ and θ are used to update the components of the phase matrix C for use in the power, gradient, and weight calculations in the next iteration of `adapt.m` as the array rotates.

Chapter 6

Algorithm and Oracle Performance

The HSB gradient algorithm and oracle performance will be presented for various input scenarios in this chapter. The user can configure many inputs when running the algorithm such as the number of jammers impinging on the array, the jammer arrival angles, the jammer frequencies, the number of peripheral antennas, the array configuration, the input noise level, array motion of roll, pitch, or yaw, and any initial array offset in the roll direction. The figures included in this chapter are representative of the performance of the HSB gradient algorithm for a planar circular array. Various input scenarios are included to show the robustness of the HSB gradient algorithm.

6.1 Summary of Performance

This section will provide a brief summary of the performance of the algorithm and the oracle. The performance will be reported for 3 categories: no motion, yaw motion, and full (roll, pitch, and yaw) motion.

The input to the array is modeled as high power jammers and low power noise. The GPS signals are not modeled explicitly, although they would appear similar to the noise. The jammer amplitudes are represented as voltages and the noise amplitude is represented in watts. During each algorithm iteration, the array output power is sampled every $\tau = 50$ ns and $\Gamma = 400$ of these samples are averaged to represent the average array output power. The averaging interval, one algorithm iteration, is 20 μs. The gradient of this average output power is calculated and the antenna weights are updated per iteration of the HSB gradient algorithm.

6.1.1 No Motion

The algorithm reduces the array output power to within 5 dB of the 1 dB noisefloor more than 99.8% of the time. The algorithm takes, on average, 10 iterations or 200 μs to reduce the initial array output power to within 5 dB of the noisefloor. The fewer the number of jammers, the faster the initial convergence. The performance does not deteriorate with an increasing number of jammers, as long as $N_{jam} < N_{ant}$, nor does performance suffer for varying jammer amplitudes, up to 50 dB. The algorithm maintains the output power below the noisefloor after the initial convergence.

The oracle reduces the array output power to within 5 dB of the 1 dB noisefloor 99.9999998% of time when the jammer arrival directions do not result in a confounding configuration or near-confounding configuration (as determined by the condition number of the phase matrix $C_{reduced}$, section 3.3.2).

6.1.2 Yaw Motion

The performance of the algorithm, simulated with yaw motion (500°/second or less), is only slightly weaker than the results when there is no motion. The algorithm reduces the array output power to within 5 dB of the 1 dB noisefloor more than 99.3% of the time and initial convergence takes, on average, 10 iterations or 200 μs. The performance does not deteriorate with an increasing number of jammers or increasing yaw rate, as long as $N_{jam} < N_{ant}$, and varying jammer amplitudes, up to 50 dB, do not have an adverse effect on performance. When the output power is greater than 5 dB above the noisefloor, the number of algorithm iterations to reduce the power below this level increases with the number of jammers but is unaffected by the yaw rate and on average is less than 3 iterations or 60 μs.

The oracle reduces the array output power to within 5 dB of the 1 dB noisefloor 99.9999998% of time when the jammer arrival directions do not result in a confounding configuration or near-confounding configuration.

6.1.3 Full Motion

The performance of the algorithm and oracle with full roll, pitch, and yaw motion mirror the performance of the algorithm and oracle with only yaw motion. The maximum motion rates simulated were 180°/second for roll, 180°/second for pitch, and 500°/second for yaw.

6.2 Catalog of Output

The figures in this section show the output of the Matlab file `adapt.m` which simulates the HSB gradient algorithm. Each figure contains three plots: the array output power due to the HSB gradient algorithm weights, the array output power due to the oracle weights, and the condition number of the phase matrix $C_{reduced}$. The array used in these simulations consisted of 7 antennas in a planar configuration of 1 central, fixed-weight antenna and 6 peripheral antennas equally spaced around the central antenna at 0°, 60°, 120°, 180°, 240°, 300°.

6.2.1 No Motion

For the following plots no array motion is simulated. The number of jammers incident on the array is indicated in the plot title. The jammers arrive in the plane of the array with elevation angle $\theta = 90°$ and azimuth angles $\phi = 120°, 330°, 25°, 204°, 0°$, and 108°. The jammer amplitudes are simulated with constant amplitudes of 100 V (or 40 dB). The noise added to the input of the array is 1.25 (or 1 dB) and the algorithm attempts to reduce the output power level to this level. The number of iterations simulated is 25000, a duration of 0.5 seconds. As the plots show, both the oracle and algorithm successfully reduce the output power of the array to the noisefloor.

Table 3. Typical Data for No Motion Cases.

noisefloor=1 dB	No Motion, 25000 Iterations (.5 sec)				
$J_1 = \cdots = J_{N_{jam}}$ = 40 dB	Output Power > Allowed Level		# iterations to return below allowed level		Maximum Condition Number
# jammers	algorithm	oracle	algorithm	oracle	
2	0.104%	1.6e-07%	3	0	1.27
4	0.032%	1.6e-07%	9	0	3.54
6	0.036%	1.6e-07%	10	0	14.02

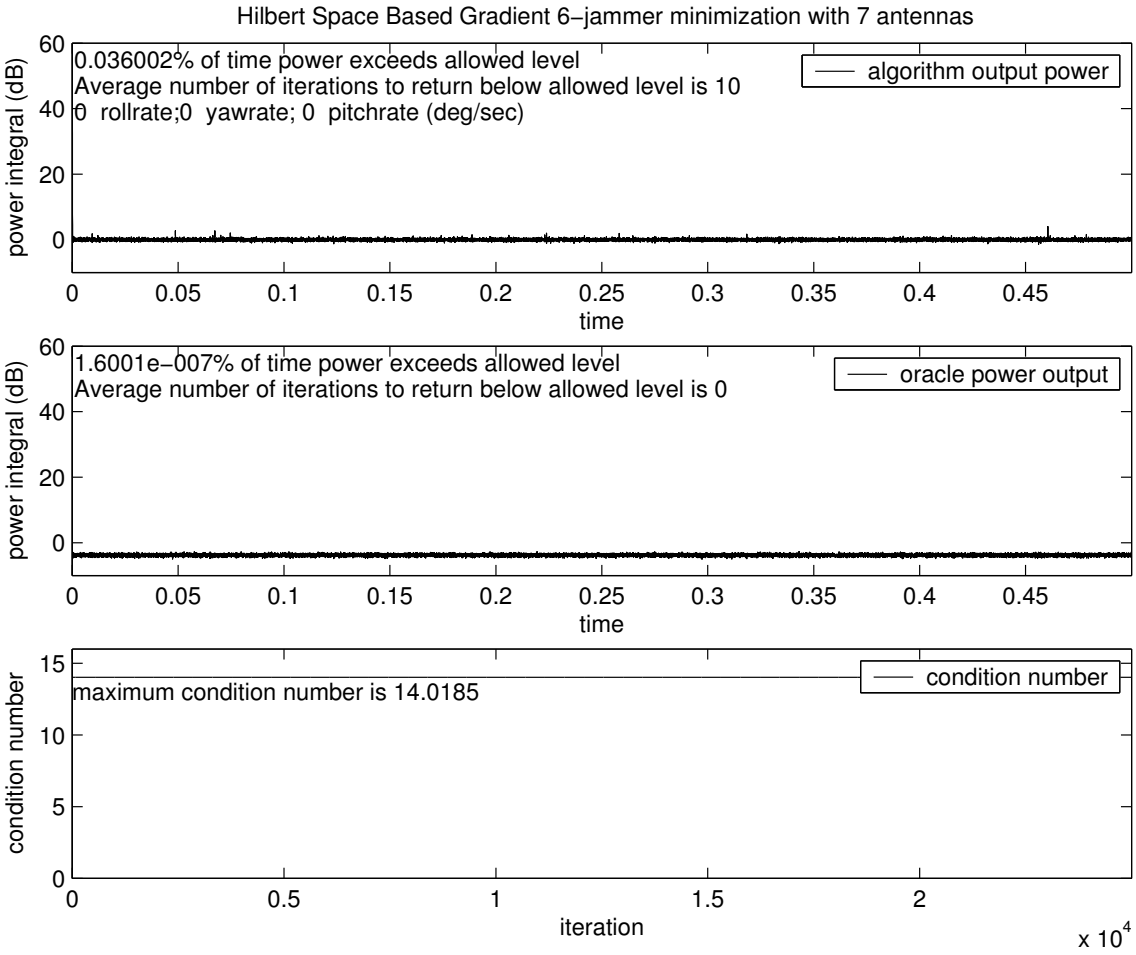

Figure 6. Six Jammers, No Motion.

6.2.2 Yaw Motion

The input conditions for the plots in this section are the same as in the previous section except now the array is simulated with a yaw rate of 360°/second. The number of jammers incident on the array is indicated in the plot title. The jammers arrive in the plane of the array, $\theta = 90°$ and initially with $\phi = 120°, 330°, 25°, 204°, 0°$, and 108°. The jammer amplitudes are all equal to 100 V (40 dB). The noise added to the input of the array is 1 dB and the algorithm attempts to reduce the output power to this level. As the plots show, both the oracle and algorithm successfully reduce the output power of the array to the noisefloor.

Note the occurrence of three near-confounding configurations in figure 7. Although the oracle shows difficulty annihilating the jammers at these near-confounding configurations, the HSB gradient algorithm is able to minimize the jammers at these same configurations. Also, when the output power exceeds 5 dB above the noisefloor, the HSB algorithm requires less weight updates to restore the output power to within 5 dB of the noisefloor than the oracle, see Table 4.

Table 4. Typical Data for Yaw Motion Cases.

noisefloor=1 dB	Yaw Motion (360°/sec), 25000 Iterations (.5 sec)				
$J_1 = \cdots = J_{N_{jam}}$ = 40 dB	Output Power > Allowed Level		# iterations to return below allowed level		Maximum Condition Number
# jammers	algorithm	oracle	algorithm	oracle	
2	0.7%	1.6e-07%	2	0	3.13
4	0.504%	1.6e-07%	3	0	4.84
6	0.564%	.924%	2	34	85590.32

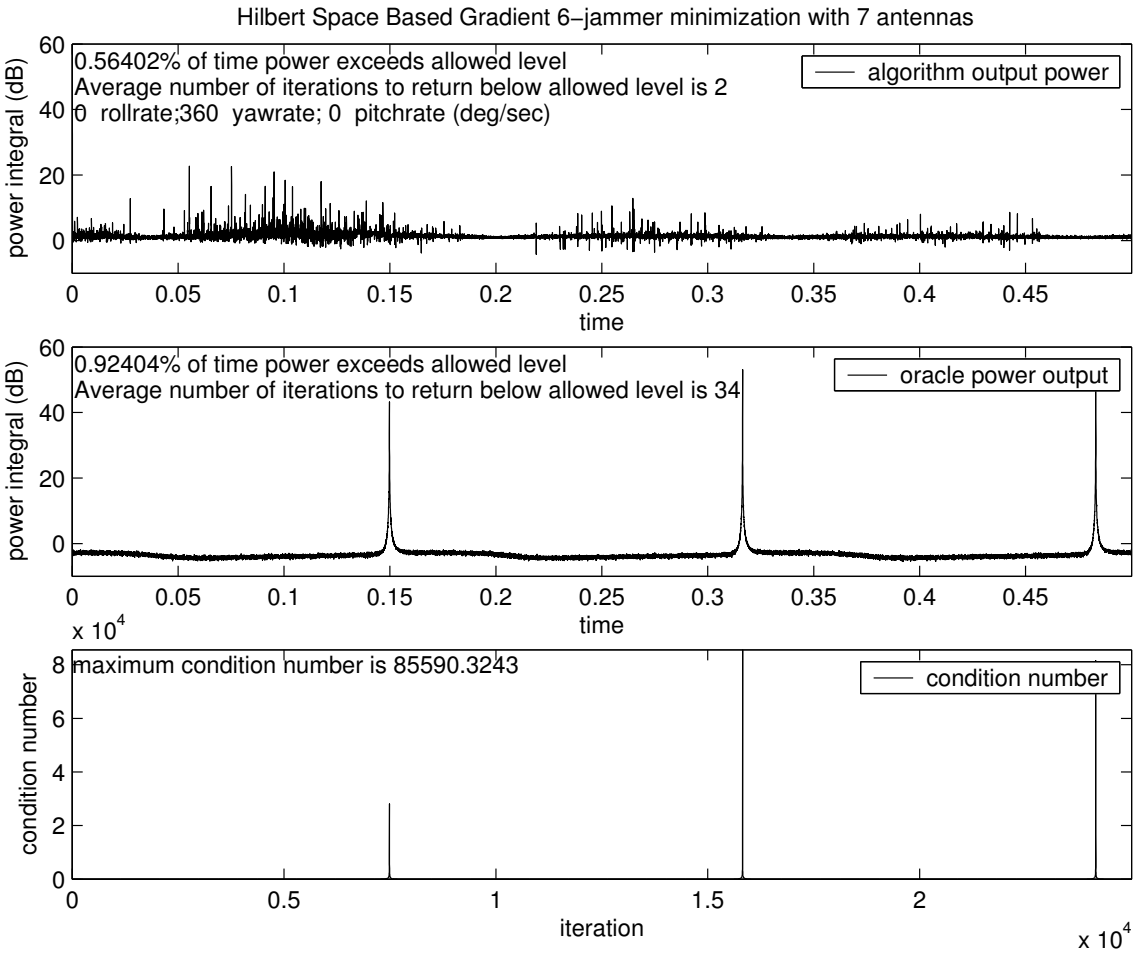

Figure 7. Six Jammers, Yaw Motion, Near Confounding Configuration.

6.2.3 Different Jammer Amplitude Levels

The input conditions for the plots in this section are similar to those of the previous section except that now the jammer amplitudes are at different levels. As noted (section 4.4), this does not affect the oracle solution. The jammers arrive in the plane of the array as before, $\theta = 90°$ and $\phi = 120°, 330°, 25°, 204°, 0°$, and $108°$ but now the jammer amplitude voltages are $316, 31.6, 100, 10, 100, 316$ (50 dB, 30 dB, 40 dB, 20 dB, 40 dB, 50 dB). The array is simulated with and without motion; this is indicated in the plot caption.

Table 5. Typical Data for No Motion Cases with Different Jammer Amplitudes.

noisefloor=1 dB	No Motion, 25000 Iterations (.5 sec)				
different jammer amplitudes	Output Power > Allowed Level		# iterations to return below allowed level		Maximum Condition Number
# jammers	algorithm	oracle	algorithm	oracle	
2 (50, 30 dB)	0.244%	1.6e-07%	2	0	1.27
4 (50, 30, 40, 20 dB)	0.084%	1.6e-07%	3	0	3.54
6 (50, 30, 40, 20, 40, 50 dB)	0.128%	1.6e-07%	3	0	14.02

Table 6. Typical Data for Yaw Motion Cases with Different Jammer Amplitudes.

noisefloor=1 dB	Yaw Motion (360°/sec), 25000 Iterations (.5 sec)				
different jammer amplitudes	Output Power > Allowed Level		# iterations to return below allowed level		Maximum Condition Number
# jammers	algorithm	oracle	algorithm	oracle	
2 (50, 30 dB)	1.956%	1.6e-07%	2	0	3.13
4 (50, 30, 40, 20 dB)	0.988%	1.6e-07%	2	0	4.84
6 (50, 30, 40, 20, 40, 50 dB)	3.088%	.924%	3	34	85590

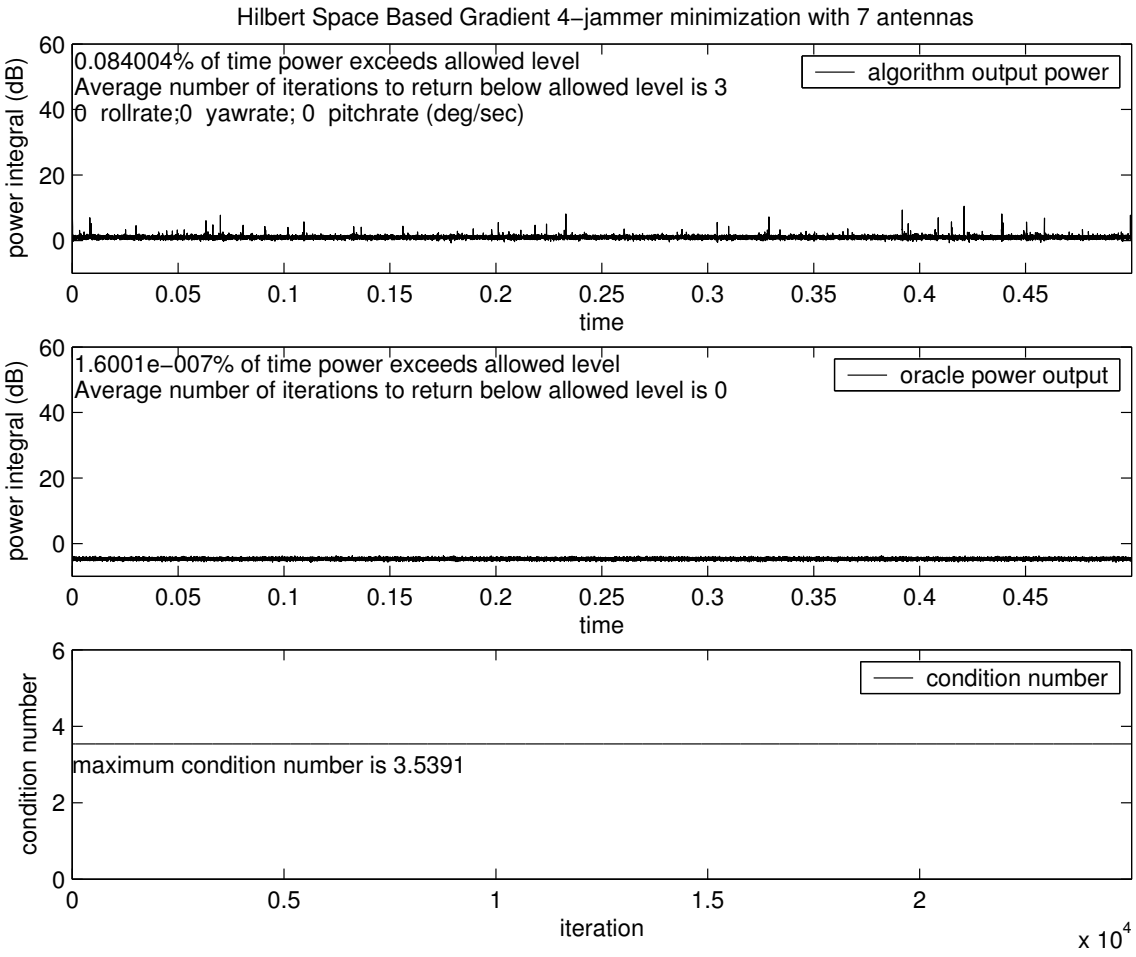

Figure 8. Four Jammers 50 dB, 30 dB, 40 dB, 20 dB; No Motion.

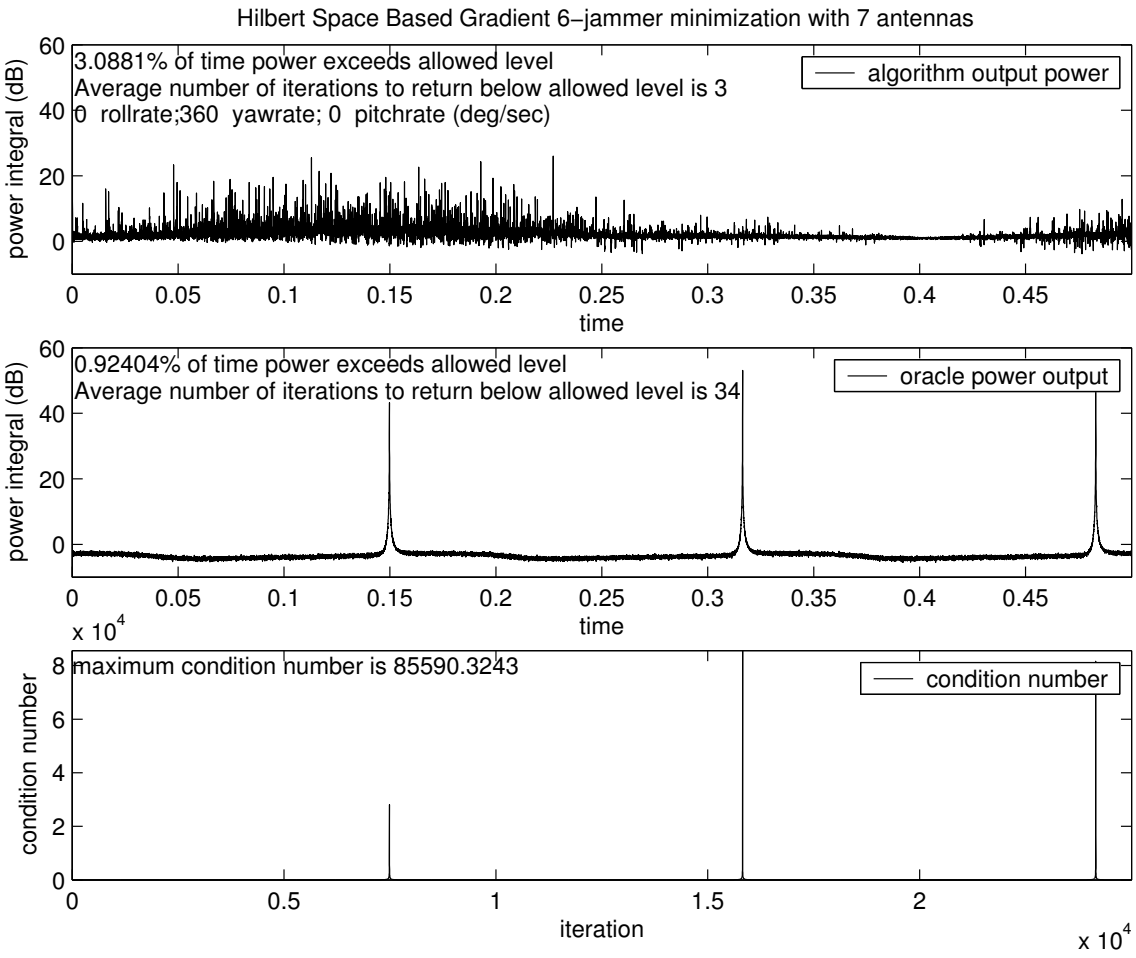

Figure 9. Six Jammers 50, 30, 40, 20, 40, 50 dB; Yaw Motion.

6.2.4 Full Motion

The array is simulated with roll=20°/second, pitch=50°/second, and yaw=180°/second. The jammers arrive with different amplitudes, indicated in the plot caption and the arrival directions as in previous sections, $\theta = 90°$ and $\phi = 120°, 330°, 25°, 204°, 0°$, and 108°. The noisefloor the algorithm attempts to reach is 1 dB.

Table 7. Typical Data for Full Motion Cases.

noisefloor=1 dB	Full Motion, 25000 Iterations (.5 sec)				
different jammer amplitudes	Output Power > Allowed Level		# iterations to return below allowed level		Maximum Condition Number
# jammers	algorithm	oracle	algorithm	oracle	
2 (50, 30 dB)	0.784%	1.6e-07%	2	0	3.063
4 (50, 30, 40, 20 dB)	0.516%	1.6e-07%	2	0	4.831
6 (50, 30, 40, 20, 40, 50 dB)	1.664%	0.792%	2	41	62644

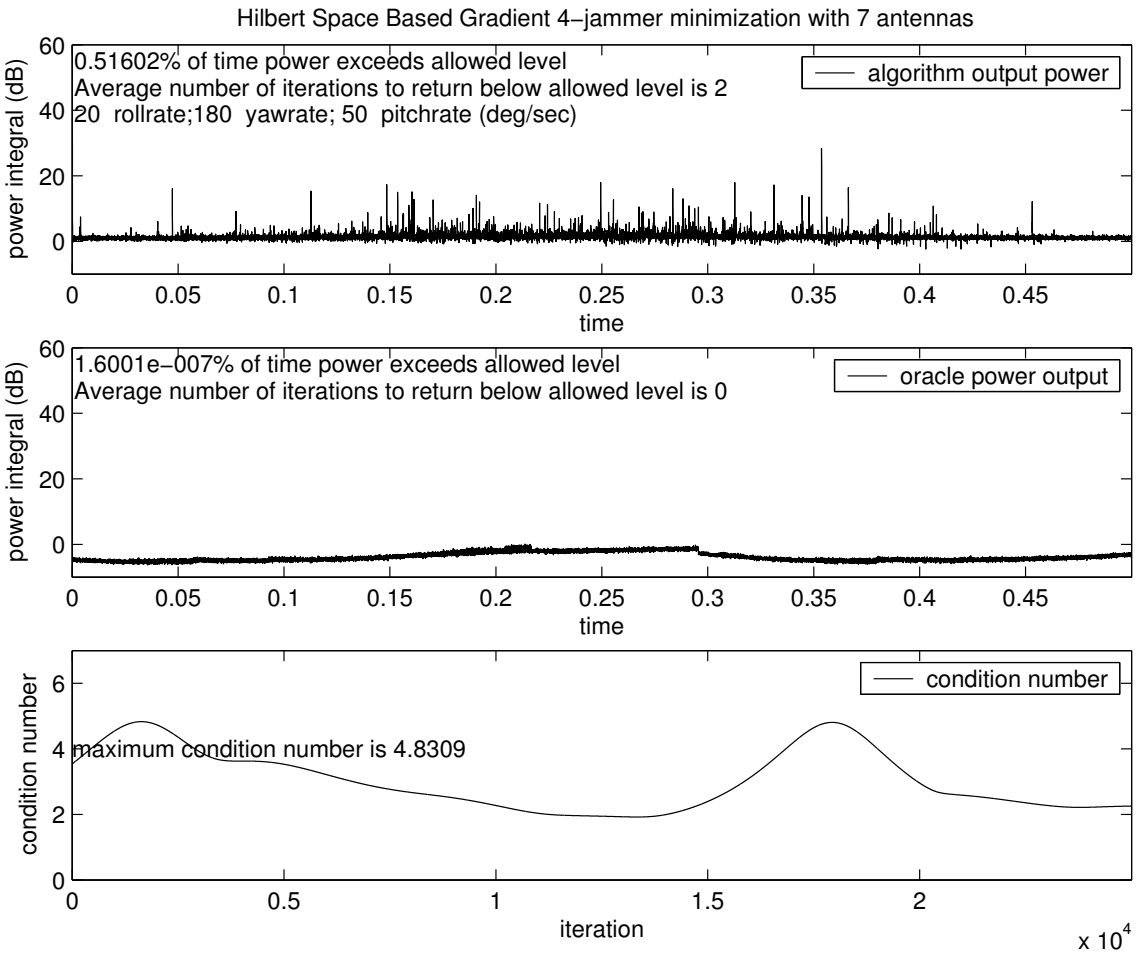

Figure 10. Six Jammers 50, 30, 40, 20, 40, 50 dB; Full Motion.

6.3 A Note on Out-of-Plane Jammers

For a planar array, an out-of-plane jammer is mathematically equivalent to an in-plane jammer at a lower frequency. A jammer at wavelength λ_1, arriving at the array from a direction (ϕ, θ) generates an array output

$$J\ C\ A\ = J_1 e^{i\omega_1 t}[e^{i\frac{2\pi}{\lambda_1}\vec{a}_1\cdot\vec{j}_1}\cdots e^{i\frac{2\pi}{\lambda_1}\vec{a}_{N_{ant}}\cdot\vec{j}_1}]\begin{bmatrix} 1 \\ w_2 e^{iq_2} \\ \vdots \\ w_{N_{ant}} e^{iq_{N_{ant}}} \end{bmatrix}.$$

Recall, from equation (37), that the array output from this jammer impinging on the mth antenna is

$$J_1\ e^{i\frac{2\pi}{\lambda_1} r\cos(\phi-\gamma_m)\sin\theta} w_m e^{iq_m}.$$

We can define a new, longer wavelength $\lambda' = \lambda_1/\sin\theta$ so that the jammer signal at the array output is equivalent to

$$J_1\ e^{i\frac{2\pi}{\lambda'} r\cos(\phi-\gamma_m)} w_m e^{iq_m}$$

in the plane of the array ($\theta = 90°$).

6.4 Noise Immunity

The objective of this section is to investigate how far above the noisefloor any additional input noise can be before the algorithm fails. Recall the desired GPS signals are on the order of or below the noisefloor but are spread spectrum modulated. As long as the array radiation pattern nulls do not coincide with the arrival directions of the GPS signals, the latter will pass through the array and be demodulated accurately. If the input noise and jammer amplitudes are on the order of the noisefloor or are below the noisefloor, the algorithm has nothing to do; the output is already at the noisefloor. The algorithm adaptively reduces the output power of the array only when the input to the array, assumed to be primarily

from undesired jammer signals, is above the noisefloor. Various cases were run to test both the HSB gradient algorithm and the oracle for noise immunity.

The array used in these simulations consisted of 7 antennas in a planar configuration of 1 central, fixed-weight antenna and 6 peripheral antennas at $\gamma = 0°,\ 60°, 120°, 180°, 240°$,-300°. The jammers arrive in the plane of the array, $\theta = 90°$ and $\phi = 120°, 330°, 25°, 204°, 0°$, and 108° with different amplitudes of 316 (50 dB),31.6 (30 dB), 100 (40 dB),10 (20 dB),100 (40 dB), 316 (50 dB). The array is in motion with yaw= 360°/second. Three categories of cases were run in which the algorithm attempted to reduce the array output power to within 5 dB of noisefloors of 0 dB, 1 dB, and 3dB. The noise added as input to the array above the noisefloor is indicated in the tables. The results show that the algorithm is quite sensitive to noise added as input above the noisefloor. Failure is said to occur when the algorithm cannot reduce the output power below the noisefloor. An interesting feature is that the algorithm has a failure rate that decreases with the number of incident jammers. With more nulls in the array radiation pattern, more of the input noise can be attenuated.

Table 8. Algorithm Failure Rate with Increasing Noise Levels (noisefloor=0 dB).

noisefloor=0 dB	Njam					
	2		4		6	
added noise	algorithm	oracle	algorithm	oracle	algorithm	oracle
5 dB	36.113%	1.6e-07%	17.373%	1.6e-07%	23.345%	100% *
6 dB	51.85%	1.6e-07%	31.873%	1.6e-07%	37.213%	100% *
8 dB	80.563%	96.172%	59.586%	96.172%	62.342%	100% *

* indicates near-confounding configuration

Table 9. Algorithm Failure Rate with Increasing Noise Levels (noisefloor=1 dB).

noisefloor=1 dB	Njam					
	2		4		6	
added noise	algorithm	oracle	algorithm	oracle	algorithm	oracle
5 dB	20.709%	1.6e-07%	9.692%	1.6e-07%	14.633%	100% *
6 dB	35.153%	1.6e-07%	17.613%	1.6e-07%	22.677%	100% *
8 dB	71.207%	8.976%	46.722%	8.976%	48.794%	100% *

* indicates near-confounding configuration

Table 10. Algorithm Failure Rate with Increasing Noise Levels (noisefloor=3 dB).

noisefloor=3 dB	Njam					
	2		4		6	
added noise	algorithm	oracle	algorithm	oracle	algorithm	oracle
5 dB	5.284%	1.6e-7%	2.276%	1.6e-7%	4.548%	99.672% *
6 dB	10.64%	1.6e-7%	4.828%	1.6e-7%	7.368%	99.708% *
8 dB	34.381%	1.6e-7%	16.621%	1.6e-7%	20.705%	99.98% *
10 dB	66.139%	7.12%	45.062%	7.12%	48.41%	100% *

* indicates near-confounding configuration

Chapter 7

Summary and Future Work Directions

The main contributions of this work are as follows.

The exact components of the gradient are calculated using inner products of signals readily available at the individual antenna outputs and at the array output. The computation is fast and all of the gradient components are calculated per iteration.

The condition number is used to indicate the likelihood of the existence of the optimal antenna weights. A high condition number indicates a confounding configuration; one without stable oracle weights.

The ability to construct confounding configurations is presented. Research funding is being pursued to study other methods to construct confounding configurations as well as the geometric interpretations, if any, of them.

The oracle solution calculates the exact antenna weights to null the jammer component of the array output. The weights depend only upon the present configuration of antennas and jammers and so provide a check of the effectiveness of weights at any configuration from any adaptive algorithm.

The HSB gradient algorithm, implemented in Matlab, successfully minimizes the power output of the array.

Areas for future work concern the investigation of an adaptive accumulation time for the calculation of the average array output power. As the array motion increases, the accumulation of instantaneous array output power measurements must decrease to capture the dynamic information of the non-stationary array environment.

References

[1] Robert E. Collin. *Antennas and Radiowave Propagation*, section 3.6, pages 107–121. McGraw-Hill Series in Electrical Engineering. McGraw-Hill, Inc., New York, New York, 1985.

[2] Biswa Nath Datta. *Numerical Linear Algebra and Applications*, chapter 7, pages 316–324. Brooks/Cole Publishing Company, Pacific Grove, California, 1996.

[3] Biswa Nath Datta. *Numerical Linear Algebra and Applications*, section 7.9, pages 348–351. Brooks/Cole Publishing Company, Pacific Grove, California, 1996.

[4] Paul G. Flikkema. Spread spectrum techniques for wireless communications. *IEEE Signal Processing Magazine*, 14(3):26–28, May 1997.

[5] Anton Gecan, Paul G. Flikkema, and Arthur David Snider. Jammer cancellation with adaptive arrays for gps signals. *IEEE Southeastcon Conference Proceedings*, pages 320–323, 1996.

[6] Anton S. Gecan and Michael D. Zoltowski. Power minimization techniques for gps null steering antennas. *preprint*, 1995.

[7] Jr. J. J. Spilker. Gps signal structure and performance characteristics. *Navigation*, 25(2):121–146, July 1978.

[8] Harold J. Larson. *Introduction to Probability Theory and Statistical Inference*, section 5.4, pages 178–179. Wiley Series in Probability and Mathematical Statistics. John Wiley and Sons, Inc., New York, New York, 1969.

[9] John Litva and Titus Kwok-Yeung Lo. *Digital Beamforming in Wireless Communications*, section 2.4, pages 28–34. The Artech House Mobile Communications Series. Artech House, Inc., Norwood, Massachusetts, 1996.

[10] Robert A. Monzingo and Thomas W. Miller. *Introduction to Adaptive Arrays*, section 3.3, page 89. A Wiley-Interscience Publication. John Wiley and Sons, New York, 1980.

[11] Robert A. Monzingo and Thomas W. Miller. *Introduction to Adaptive Arrays*, section 4.2, pages 162–178. A Wiley-Interscience Publication. John Wiley and Sons, New York, 1980.

[12] Robert A. Monzingo and Thomas W. Miller. *Introduction to Adaptive Arrays*, chapter 5, pages 217–292. A Wiley-Interscience Publication. John Wiley and Sons, New York, 1980.

[13] Roger L. Peterson, Rodger E. Ziemer, and David E. Borth. *Introduction to Spread Spectrum Communications*, section 2.3, 3.3, pages 52–60, 113–115. Prentice-Hall, Inc., Upper Saddle River, New Jersey, 1995.

[14] John G. Proakis. *Digital Communications*, section 13.2, pages 698–699,707–709. McGraw-Hill Series in Electrical and Computer Engineering. Communications and Signal Processing. McGraw-Hill, Inc., New York, New York, third edition, 1995.

[15] Jr. R. T. Compton. The power-inversion adaptive array: Concept and performance. *IEEE Transactions on Aerospace and Electronic Systems*, 15(6):803–814, November 1979.

[16] Gilbert Strang. *Introduction to Linear Algebra*, section 3.3, 9.2, pages 125–136,386–388. Wellesley-Cambridge Press, Wellesley, Massachusetts, 1993.

Appendices

Appendix A: Symbol Glossary

Symbol	Meaning
i	$\sqrt{-1}$
τ	sampling interval, 50 ns
γ	sampling index
Γ	number of samples to average
t_γ	time
c	speed of light, 3×10^8 meters/second
ω	frequency in radians/second
f	frequency in Hertz
λ	wavelength in meters
r	distance in meters between reference antenna and a peripheral antenna
m	indicates one antenna in an array
N_{ant}	total number of antennas in an array
$\vec{a}_m$	position vector of antenna $\#m$
w	magnitude of antenna weight
q	phase of antenna weight
A_m	complex weight of antenna $\#m$, $w_m e^{iq_m}$
χ_{GPS}	variable representing GPS signal
$N(t)$	random noise input to antennas
n	jammer index
N_{jam}	total number of jammers impinging on array
$\vec{j}_n$	vector indicating direction of arrival of jammer $\#n$
$\vec{v}_n$	unit vector indicating direction of arrival of jammer $\#n$ scaled by $\frac{2\pi}{\lambda_n}$
J_n	amplitude of jammer $\#n$

Symbol	Meaning
ϕ_n	azimuth angle of jammer $\#n$
θ_n	elevation angle of jammer $\#n$
$\vec{x}(t)$	generic input to array
C	$N_{jam} \times N_{ant}$ matrix of exponential phase differences between jammers and antennas
$\hat{i}$	direction unit vector
$\hat{j}$	direction unit vector perpendicular to $\hat{i}$
$d(t)$	desired array output signal
$e(t)$	error between desired and actual signals
r_{xd}	cross-correlation vector of signals x and d
R_{xx}	autocorrelation matrix of signal x
α	time index
k	iteration index
$\mathsf{p}(\alpha, A)$	instantaneous array output power at time α as a function of antenna weights A
$P(k, A)$	average array output power at iteration k as a function of antenna weights A
T	number of instantaneous power samples averaged during each iteration
W	column of antenna weights $[w_1\ w_2 \cdots w_{N_{ant}}\ q_1\ q_2 \cdots q_{N_{ant}}]^T$
μ	gradient step size OR mean of random variable
σ^2	variance of random variable
$\Re$	real part
$\Im$	imaginary part

Appendix B: Matlab Source Code

setup.m

```
 %  This script is a tool for initializing the adapt simulation.
%  Not all parameters listed herein are used in any given version
%   of the simulation
%  September 24, 2004

deg_to_rad = pi/180;
jmath = sqrt(-1);
more off;
format short;
Nant = 7;          % number of antennas in array
array_radius = .1; %radius of array (meters)
A=zeros(Nant,2);   %Set up weights
A(:,1)=ones(Nant,1);
A(:,2)=zeros(Nant,1);
a=zeros(Nant,1);
Ann=zeros(Nant,2);  %weights for noise free simulation

Ann(:,1)=ones(Nant,1);
Ann(:,2)=zeros(Nant,1);
ann=zeros(Nant,1);

a_min = 0.1; a_max = 10;  % limits on weight mags
step_control = 1;  %fraction of gradient correction to be implemented
step_control_nn = 1;
noise_floor = 1.25;%target value for power;
```

```
                %background noise level ( 1 = 0 dB)
speed_of_light = 3.0e+08; %speed of light, m/s
P_int = 20e-6;     %integration time for power computation
LOfreq = 1.575e+09; %local oscillator frequency
omegaLO = 2*pi*LOfreq;

disp('How many jammers? Enter 1,2,3,...,9 to use these preset values. ');
Njams=input('Otherwise, enter -1 [minus one].   ');
theta_jam =[90,90,90,90,90,90,90,75,100] * deg_to_rad;
theta_jam(Njams+1:9)=[];
phi_jam = 359*rand(1,9)*deg_to_rad;
%phi_jam = [120,330,25,204,0,108,351,136,225]*deg_to_rad;
phi_jam(Njams+1:9)=[];
phi_init=phi_jam;
del_freq = [0 0 0 0 0 0 0 0 0];
del_freq(Njams+1:9)=[];
jam_freq = LOfreq + del_freq;
omega_jam = jam_freq*2*pi;
psijam = [0 0 0 0 0 0 0 0 0]' * deg_to_rad;
psijam(Njams+1:9)=[];
%J=[100,100, 100,100,100, 100,100,100,100];
J=[316,31.6, 100,10,100, 316,100,100,100];
J(Njams+1:9)=[];
Ajamdot = [0 0 0 0 0 0 0 0 0];
Ajamdot(Njams+1:9)=[];

disp('Enter simulation time in seconds; recommend 15e-3; note that')
```

```
disp('simulations take ~10-20 seconds of computer time per millisec')
disp('of simulation time.')
sim_time=input('Simulation time in sec.;  ');
disp('Enter vehicle angular velocity, deg/sec.');
roll=input('roll rate: ');  %+y onto +z
pitch=input('pitch rate: '); % +z onto -x
yaw=input('yaw rate: '); %+y onto +x
motion_vector=[roll, pitch, yaw];
beta=input('Set initial rotation angle (deg): ') %-roll
disp('Data Summary')
disp('Number of antennae')
disp(Nant)
disp('Number of jammers; initial theta''s; initial phi''s')
disp(Njams)
disp(theta_jam/deg_to_rad)
disp(phi_jam/deg_to_rad)
disp('relative frequencies; phases; amplitudes; amplitude rates')
disp(del_freq)
disp(psijam'/deg_to_rad)
disp(J)
disp(Ajamdot)
disp('roll, pitch, yaw rates; initial rotation')
disp(motion_vector)
motion_vector=motion_vector*deg_to_rad; roll=roll*deg_to_rad;...
 pitch=pitch*deg_to_rad; yaw=yaw*deg_to_rad;
disp(beta)
beta=beta*deg_to_rad;
```

```
[az,el,rr]=cart2sph((sin(theta_jam).*cos(phi_jam)), ...
 (sin(theta_jam).*sin(phi_jam)*cos(beta)+...
 cos(theta_jam)*sin(beta)),...
 (cos(theta_jam)*cos(beta) - ...
 sin(theta_jam).*sin(phi_jam)*sin(beta)));
phi_jam= az; theta_jam = pi/2 - el;
disp('Rotated jammer theta"s, phi"s');
disp(theta_jam/deg_to_rad)
disp(phi_jam/deg_to_rad)

%  Set up initial array manifold vector for each source
%(relative phases of each jammer at each antenna).
zeta = array_radius*omega_jam.*sin(theta_jam)/speed_of_light;
gamma = linspace(0, 2*pi*(Nant-2)/(Nant-1), (Nant-1));
for m=2:Nant,
   for n=1:Njams,
        C(n,1) = 1;
        C(n,m)=exp(jmath*zeta(n)*cos(phi_jam(n)-gamma(m-1)));
    end
end
```

adapt.m

```
%adapt.m
%
% This script computes the average array output power and
% its exact gradient.  The antenna weights are adapted
```

```
% and the oracle weights are computed.
% Noise is simulated as input to each antenna with
% zero mean and variance equal to the noise_floor.
% Plots of the array output power are generated
% corresponding to the algorithm weights and the
% oracle weights.  The condition number of the
% phase matrix is also plotted.
Niterations = floor(sim_time/P_int);
time = zeros(Niterations,1);
seed = input('Enter a random number seed.   ')
randn('seed', seed);
% Set dynamic variables flag.
moving=norm(motion_vector);
modulator = Ajamdot'*Ajamdot;
%initialize variables/counters/flags
cntr=0;
m=1;
n=0;
count = 0;
sp=1;
kj=1;
oj=1;
acc=1;
occ=1;
below=0;
under=0;
convg=0;
```

Appendix B (Continued)

```
its=0;
prev_sample=0;
ops=0;  %oracle previous sample
allowed_level=10^((5+10*log10(noise_floor))/10);
%Compute power plus noise
for i = 1:Niterations,
    if moving>0;
      motion;     %if array in motion, call motion.m
    end;
    if modulator>0;
        modulate;   %if modulated jammers, call modulate.m
    end;
    a = A(:,1).*exp(jmath*A(:,2));
    weight_prev=abs(a);
    sig_only=J*C*a;
    Cred=C(:,2:Nant);
    a_opt = Cred\(-C(:,1)); %oracle solution
    a_opt = [1; a_opt];   %add central antenna weight
    sig_opt = J*C*a_opt;
    for integral = 1:400,     %create noise samples
       nois_sample=sqrt(noise_floor/Nant)*randn(1,Nant);
       nois_vec(integral,:)=nois_sample;
       nois_only=nois_sample*a;
       Vn=sum(nois_sample);
       pn(integral)=Vn*conj(Vn);
       V = sig_only+nois_only;
       nois_opt=nois_sample*a_opt;
```

```
   V_opt = sig_opt+nois_opt;
   pwr(integral)=V*conj(V);
   pwr_opt(integral)=V_opt*conj(V_opt);
end
nois_ran=sum(nois_vec)/sqrt(400); %average noise
%average power signals
powerno(i)=sum(pn); %noise only power
power_hsb(i)=sum(pwr)/400;
power_orac(i)=sum(pwr_opt)/400;
cn(i) = cond(Cred);     %condition number of Cred
%count iterations to reconverge below allowed level
if (power_hsb(i) >= allowed_level)
    kj=kj+1;
elseif (prev_sample >= allowed_level)
        if (power_hsb(i) < allowed_level)
            nosta(acc)= kj; %number of iterations to adapt
            acc=acc+1;
            kj=1;
            below=1;
        end
end
if (below == 0)  %power never went  below allowed level
    nosta = 0;
end
prev_sample=power_hsb(i);
%count iterations for oracle to reconverge below allowed level
if (power_orac(i) >= allowed_level)
```

```
        oj=oj+1;
    elseif (ops >= allowed_level)
        if (power_orac(i) < allowed_level)
            rto(occ)=oj;
            occ=occ+1;
            oj=1;
            under=1;
        end
    end
    if (under==0) %oracle power never went below allowed level
        rto=0;
    end
    ops=power_orac(i);    %ops is previous oracle sample
    rel_w_err(i)=norm(a_opt - a)/norm(a_opt);
    time(i) = P_int*i;
    sample(i)=i;
    %Compute the gradient components
    for k=2:Nant,
      a_grad = zeros(Nant,1);
      a_grad(k) = A(k,1)*exp(jmath*A(k,2));
      f=J*C*a_grad + nois_ran*a_grad;  % first term in <f,g>
      g=J*C*a + nois_ran*a;
      G=2*conj(f)*g;
      grad(k,1) = real(G)/A(k,1);
      grad(k,2) = imag(G);
      grad1(k,i)=grad(k,1);
      grad2(k,i)=grad(k,2);
```

```
    end
    %Update the weight vector
    delta_wt=step_control*grad...
              *(noise_floor-power_hsb(i))/norm(grad,'fro')^2;
    magwt(i)=norm(delta_wt,'fro');
    A=A+delta_wt;
    count = count + 1;
    for k=2:Nant,  %  Repair unacceptable weights
        if (A(k,1)<0)
            A(k,1)=-A(k,1);
            A(k,2)=A(k,2)+pi;
        end
        if (A(k,1)<a_min)
            A(k,1)=a_min;
        end
        if (A(k,1)>a_max)
            A(k,1)=a_max;
        end
    end
    wAmp(:,i) = A(:,1);
    wPh(:,i) = A(:,2);
    wOpt(:,i)=a_opt;
end
if (kj > 1)
        nosta(acc)=kj;
end
if (oj > 1)
```

```
    rto(occ)=oj;
end
for rows = 1:size(wOpt,1),
          %change matrix of complex optimal weights to
    for cols = 1:size(wOpt,2),
          %matrix of magnitudes and matrix of phases
        mwO(rows,cols) = sqrt(real(wOpt(rows,cols))^2...
                          +imag(wOpt(rows,cols))^2);
        pwO(rows,cols) = atan(imag(wOpt(rows,cols))...
                          /real(wOpt(rows,cols)));
    end
end
%Output power in dB
power_orac_dB = 10*log10(power_orac);
power_hsb_dB = 10*log10(power_hsb);
%Failure rate (power > allowed_level)
b=find(power_hsb_dB >= 10*log10(allowed_level));
err=100*size(b)/size(power_hsb_dB);
orac_over=find(power_orac_dB >= 10*log10(allowed_level));
err_orac=100*size(orac_over)/size(power_orac_dB);
%Initial adaption iteration count
for abc = 1:length(power_hsb_dB),
    if convg == 0
        its = its+1;
    end
    if (power_hsb_dB(abc) <= 10*log10(allowed_level))
        convg = 1;      %algorithm has reduced power below
```

```
    end                 % allowed level
end
its=its-1;
%Average re-convergence counts
if (acc > 1)
    str=sum(nosta)/(acc-1);
else
    str=0;   %power never went above allowed level
end
if (occ > 1)
    osc=sum(rto)/(occ-1);    %oracle sample count
else
    osc=0;  %oracle never went above allowed level
end
%Plot output
figure;
subplot(3,1,1)
    plot(time, power_hsb_dB)
    legend('algorithm output power')
    axis([0 P_int*Niterations -10 60]);
    rep=[num2str(err) '% of time power exceeds allowed level '];
    text(.001,55,rep);
    rul=['Average number of iterations to return below allowed level is '...
            num2str(round(str))];
    text(.001,47,rul);
    rots=[num2str(roll/deg_to_rad) '  rollrate;' num2str(yaw/deg_to_rad)...
            '  yawrate; ' num2str(pitch/deg_to_rad) '...
```

```
                pitchrate (deg/sec)'];
    text(.001,39,rots);
    xlabel('time')
    ylabel('power integral (dB)');
    title(['Hilbert Space Based Gradient ',num2str(Njams),...
           '-jammer minimization with ',num2str(Nant), ' antennas']);
subplot(3,1,2)
    plot(time,power_orac_dB)
    axis([0 P_int*Niterations -10 60]);
    rprt=[num2str(err_orac), '% of time power exceeds allowed level '];
    text(.001,55,rprt)
    orl=['Average number of iterations to return below allowed level is '...
           num2str(round(osc))];
    text(.001,47,orl);
    legend('oracle power output')
    ylabel('power integral (dB)');
    xlabel('time');
subplot(3,1,3)
    ul=round(max(cn));
    plot(cn);
    axis([0 Niterations 0 ul+2]);
    legend('condition number');
    cng=['maximum condition number is ',num2str(max(cn))];
    text(.001,ul/2,cng)
    xlabel('iteration');
    ylabel('condition number');
```

motion.m

```
%  motion.m
%  This script is a tool for updating
%  the orientation matrix d when
%  the array is rotating.

%  August 5, 2003

theta_dot = -roll*sin(phi_jam) ...
            + pitch*cos(phi_jam);
phi_dot = cot(theta_jam).*(-pitch*sin(phi_jam) ...
            - roll*cos(phi_jam)) + yaw;

theta_jam = theta_jam + theta_dot*P_int;
phi_jam = phi_jam + phi_dot*P_int;

%  These can be replaced by
%  nonlinear functions if desired.

%  Set up array manifold vector for
%  each source (relative phases of
%  each jammer at each antenna).

zeta = array_radius*omega_jam.*sin(theta_jam)/speed_of_light;
for mm=2:Nant,
    for nn=1:Njams,
```

```
        C(nn,1)=1;
        C(nn,mm)=exp(jmath*zeta(nn)*cos(phi_jam(nn)-gamma(mm-1)));
    end
end
```

Appendix C: GPS Signal Characteristics

The Hilbert space based gradient algorithm is designed to adjust the weights of antennas in an array to detect signals from the Global Positioning System (GPS) Satellites while simultaneously nulling jammer signals. The GPS signals originate from one of 24 satellites. This network of satellites provides extremely accurate location and time data for navigational purposes. The signals transmitted from the satellites have very low power due to the limited fuel source onboard each satellite. The modulation scheme for the satellite signals is called Direct Sequence Spread Spectrum modulation. Spread spectrum modulates a narrowband signal into a wideband signal that appears to be random in nature while conserving the power in the signal. With the same amount of average power now over a much larger bandwidth, the instantaneous power of the signal is reduced. This is accomplished by modulating the data with a sequence of bits called chips that take on only two values +1 or -1. The sequence of chips appears as random data but, in fact, is not. The sequence is called a pseudorandom or PN sequence because it can be generated from an equation, a primitive polynomial in GF(2). The sequence can be generated by implementing the primitive polynomial with a Linear Feedback Shift Register. For demodulation, the PN sequence that modulates the data must be reproduced at the receiver. Unintended receivers who do not know the PN sequence will not be able to demodulate the data. The receiver works by correlating the received signal with the locally generated PN sequence. The sequence has a high correlation with itself only at zero shift. When a high correlation is detected, the receiver integrates the signal over a data bit interval to recover the value of the data bit. This despreads the signal from wideband back to narrowband. The average power in the signal remains the same but the instantaneous power is increased due to the smaller bandwidth, thus improving correct bit detection [4], [13].

Each chip of the PN sequence is of fixed length, T_c seconds, but this is much shorter than the length of a data bit, T_b seconds. Modulating the data bits by this PN sequence increases the bandwidth or spreads the spectrum of the data signal from $B = 1/T_b$ Hz to

$W = 1/T_c$ Hz. The ratio W/B is called the processing gain and is how much the bandwidth has increased due to spreading [14].

The nonspread GPS data rate is 50 bps BPSK, $T_b = 1/50$ seconds. The transmitted, spread GPS signal consists of signals in quadrature. The I and Q components are spread spectrum modulated each with a distinct PN sequence. For the I component, the data is modulo-2 added to a PN sequence called the P (Precision) code and for the Q component, the data is modulo-2 added to another PN sequence called the C/A (Clear/Acquisition) code [7]. The chip length for the P code is $T_c = 1/10.23\text{x}10^6$ seconds giving a processing gain of

$$\frac{W}{B}_P = \frac{T_b}{T_c} = 204600.$$

Therefore, the spectral height of the I component is 204600 times lower than it would be if it were not spread. The chip length for the C/A code is $T_c = 1/1.023\text{x}10^6$ seconds giving a processing gain of

$$\frac{W}{B}_{C/A} = \frac{T_b}{T_c} = 20460.$$

and so the spectral height of the Q component is 20460 times lower than it would be if it were not spread.

The ratio of signal energy per data bit, $\mathcal{E}_b = P_{av}T_b$, to jammer power spectral density, $J_0 = J_{av}T_c$ is

$$\frac{\mathcal{E}_b}{J_0} = \frac{P_{av}T_b}{J_{av}T_c} = \frac{T_b/T_c}{J_{av}/P_{av}} \tag{40}$$

The ratio $\mathcal{E}_b/J_0$ can be interpreted as the SNR, while T_b/T_c is the processing gain, and J_{av}/P_{av} is the jamming margin [14]. If the processing gain and desired SNR level are known, the jamming margin from equation (40) represents the largest value a jammer can take with reference to the signal of interest for the system to still meet the desired SNR level.

As an example, a desired SNR level of 10.5 dB per bit or 10^{-6} error probability for a BPSK signal is chosen. The processing gain for the P code yields a jamming margin of

$$10\log_{10}\frac{J_{av}}{P_{av}} = 10\log_{10}\frac{T_b}{T_c} - SNR = 10\log_{10} 204600 - 10.5 = 42.61 \text{ dB}$$

For the C/A code, the jamming margin is

$$10\log_{10}\frac{J_{av}}{P_{av}} = 10\log_{10}\frac{T_b}{T_c} - SNR = 10\log_{10} 20460 - 10.5 = 32.61 \text{ dB}$$

Given that the GPS signals received by a 0 dBIC antenna are on the order of -160 dBw [7] and the jamming margin of 32.61 dB, the jammer signal cannot be stronger than $-160 + 32.61 = -127.4$ dBw for this error probability to be achieved. This illustrates the advantage of using spread spectrum modulation in that the unwanted signal may actually be higher in power than the desired signal while still providing a low probability of bit error. This demonstrates that the goal of the algorithm to reduce the output power of the array to within 5 dB of the background noise level will allow a high probability of detection and demodulation of the GPS signals.

The signal processing required to obtain accurate navigational measurements from GPS signals is not relevant to this document. The interested reader will find a description of this in [7].

Appendix D: **Alternative Gradient Formula**

Dr. V. K. Jain has suggested the alternative derivation of the gradient formulas: Since

$$\begin{aligned} P(k) &= < xA, xA > \\ &= A^H x^H x A \\ &= A^H \Phi A \end{aligned}$$

is a real-valued function of A,

$$\begin{aligned} \nabla_A P(k) &= 2\Phi A \\ &= 2(F + iG)A \end{aligned}$$

where Φ is Hermitian symmetric: $F^T = F$; $G^T = -G$. Also, if we let $A = a + ib$, then (since $P(k)$ is a real-valued function of A)

$$\nabla_A = \nabla_a + i\nabla_b.$$

Thus,

$$\begin{aligned} \nabla_A P &= 2(F + iG)(a + ib) \\ &= 2[(Fa - Gb) + i(Fb + Ga)] \end{aligned}$$

and

$$\nabla_a P = 2(Fa - Gb)$$

$$\nabla_b P = 2(Fb + Ga).$$

The gradient of the power with respect to the magnitude and phase of the antenna weights in terms of the gradient with respect to the real and imaginary parts of the antenna

weights can be found using the chain rule:

$$\frac{\partial P}{\partial w} = \frac{\partial P}{\partial a}\frac{\partial a}{\partial w} + \frac{\partial P}{\partial b}\frac{\partial b}{\partial w}$$

and

$$\frac{\partial P}{\partial q} = \frac{\partial P}{\partial a}\frac{\partial a}{\partial q} + \frac{\partial P}{\partial b}\frac{\partial b}{\partial q}$$

where $\frac{\partial a}{\partial w} = e^{iq} = \frac{A}{w}$, $\frac{\partial b}{\partial w} = -ie^{iq} = -i\frac{A}{w}$, $\frac{\partial a}{\partial q} = iwe^{iq} = iA$, and $\frac{\partial b}{\partial q} = we^{iq} = A$.

About the Author

Sandra Gomulka Johnson was born in Johnstown, Pennsylvania. She attended the University of New Mexico in Albuquerque and obtained her BSEE in 1990. She attended graduate school at the University of Southern California in Los Angeles obtaining her MSEE in 1992. She was employed as a signal processing engineer at the Los Alamos National Laboratory, Los Alamos, New Mexico and at Raytheon, St. Petersburg, Florida before obtaining her PhD in Electrical Engineering at the University of South Florida in Tampa.