A Study of

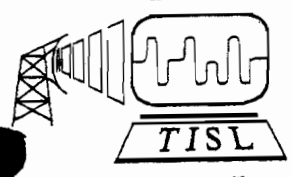
Adaptive Algorithms for HF Antenna Arrays

D. Haegert
R. Spohn
V.S. Frost
K. Sam Shanmugan
G. Sargent
R. Yarbro
B. McClendon
J.C. Holtzman

Final Technical Report TISL-5481-2

Prepared for:
Rome Air Development Center
RADC/DCCL
Griffiss Air Force Base
Rome, New York 13441-5700
Contract no. F-30602-81-C-0205

May 1986



Telecommunications and Information Sciences Laboratory
The University of Kansas Center for Research, Inc.
2291 Irving Hill Drive Lawrence, Kansas 66045

ABSTRACT

The performance of an adaptive array system will not only be dependent upon the controlling algorithm, but also upon the specific environment in which the array is used. Isolating the contributions made solely by the control algorithm represents a formidable task, considering that adaptive array systems are inherently used in situations in which the interference environment is initially unknown, time-variant, or both. If an effective algorithm choice is to be made, it is important to be able to compare array performances in identical environments. The major focus of this study is to compare and contrast the performances of various control algorithms under identical conditions in a computer simulated adaptive HF antenna array system. After examining adaptive array systems and the role played by the controlling algorithms, this study concentrates on the selection and simulation of three specific algorithms. The simulation results will then be presented in the form of a comparative evaluation of the selected control algorithms, and relevant conclusions will be drawn.

TABLE OF CONTENTS

Abstract i

Table of Contents.....ii

List of Figures.....v

List of Symbols.....viii

1.0 Introduction.....1

2.0 Adaptive Array System Model.....5

2.1 Sensor Element Array.....5
2.2 Pattern-Forming Network.....7
2.3 Adaptive Processor.....8

3.0 Least Mean Square Algorithm.....10

3.1 Motivation for Selection of LMS Algorithm.....10
3.2 Mean Square Error (MSE) Performance Criterion.....11
3.3 MSE Performance Criterion Derivation.....13
3.4 LMS Algorithm Description.....16
3.5 LMS Software Modules.....19

4.0 Constrained LMS Algorithm.....21

4.1 Motivation for Selection.....21
4.2 Constrained LMS Algorithm Description.....21
4.3 Derivation of Optimum Constrained Weight Solution.....25
4.4 Derivation of Constrained LMS Algorithm.....28
4.5 Constrained LMS Simulation Model.....29
4.5.2 TAKAO Implementation.....31
4.6 Constrained LMS Software Modules.....32

5.0 Update Covariance Algorithm.....35

5.1 Motivation for Selection.....35
5.2 Update Covariance Algorithm Description.....36
5.3 Update Covariance Software Modules.....38

6.0 Description of the HF Adaptive Array Simulation Model.....40

6.1 User-Definition of Array System and Environment.....40
6.2 Performance Evaluation.....40
6.3 Simulation Software Description.....44
6.3.1 Desired Signal Model.....44
6.3.2 Interference Model.....46
6.3.3 Thermal Noise Model.....46
6.3.4 Correlation Matrices.....46
6.3.5 Simulation Program Operation.....48

7.0 Simulation Results and Conclusions.....50

7.1 TEST1: Negligible Channel Effects.....54
7.2 TEST2: HF Channel with Moderate Delay Characteristics and Poor Attenuation Characteristics.....69
7.3 TEST3: HF Channel with Poor Delay Characteristics and Poor Attenuation Characteristics.....84
7.4 TEST4: Dependence of Convergence of Constrained LMS Algorithm on Signal Presence.....98
7.5 TEST5: Dependence of Adapted Antenna Patterns on Number of Required Iterations.....107
7.6 TEST6: Investigation of an Alternate Antenna Geometry....118
7.7 Conclusions and Recommendations.....141

References.....143

Appendix A: Complex Lowpass Equivalent Representation.....144

Appendix B: User Manual for HF Adaptive Antenna Array
Evaluation Facility.....146

Appendix C: HF Adaptive Antenna Array Simulation Facility:
Software Reference Manual.....165

LIST OF FIGURES

2.1 N-Element Adaptive Array System Model.....6

3.1 LMS-Controlled Adaptive Array System.....12
3.2 MSE Performance Surface.....15
3.3 LMS Software Modules.....20

4.1 Signal-Aligned Broadband Adaptive Array System.....24
4.2 Narrowband Signal-Aligned Array System.....30
4.3 Constrained LMS Software Modules.....33

5.1 Update Covariance Software Modules.....39

6.1 User-Entered Parameters.....41
6.2 Three-Path HF Channel Model.....43
6.3 Flow Diagram of Program Operation.....49

7.1 Constant Test Parameters.....51
7.2 Polar Elevation Plot Description.....52
7.3 Polar Azimuthal Plot Description.....53
7.4 LMS Ideal Channel.....55
7.5 Constrained LMS Ideal Channel.....56
7.6 Update Covariance Ideal Channel.....57
7.7 LMS Channel 2.....70
7.8 Constrained LMS Channel 2.....71
7.9 Update Covariance Channel 2.....72
7.10 LMS Channel 3.....85
7.11 Constrained LMS Channel 3.....86
7.12 Update Covariance Channel 3.....87
7.13 Constrained LMS (no signal) Channel 2.....99
7.14 Constrained LMS (no signal) Channel 3.....100
7.15 Constrained LMS (no signal) Channel 3.....101
7.16 LMS Channel Summary.....114
7.17 Constrained LMS Channel Summary.....115

7.18 Update Covariance Channel Summary.....116
7.19 Constrained LMS (no signal) Channel Summary.....117
7.20 Alternate Antenna Geometry for Rombiod Geometry.....119
7.21 Convergence Histogram of LMS Algorithm in Channel 2 for Rombiod Geometry.....120
7.22 Convergence Histogram of Constrained Algorithm in Channel 2 for Rombiod Geometry.....121
7.23 Convergence Histogram of Update Covariance Algorithm in Channel 2 for Rombiod Geometry.....122
7.24 Convergence Histogram of LMS Algorithm in Channel 3 for Rombiod Geometry.....123
7.25 Convergence Histogram of Constrained LMS Algorithm in Channel 3 for Rombiod Geometry.....124
7.26 Convergence Histogram of Update Covariance Algorithm in Channel 3 for Rombiod Geometry.....125

T1.1-8 Unadapted and Adapted Antenna Plots at the Arrival Angles of the Jamming Signals for TEST1.....61-68

T2.1-8 Unadapted and Adapted Antenna Plots at the Arrival Angles of the Jamming Signals for TEST2.....76-83

T3.1-8 Unadapted and Adapted Antenna Plots at the Arrival Angles of the Jamming Signals for TEST3.....90-97

T4.1-4 Unadapted and Adapted Antenna Plots at the Arrival Angles of the Jamming Signals for TEST4.....103-106

T5.1-4 Comparison of Adapted Antenna Patterns of Fastest-/ Slowest Convergence.....108-111

T6.1-14 Unadapted and Adapted Antenna Plots with Romboid Geometry for TEST2 and TEST3.....127-140

LIST OF TABLES

Table 3.1 Features of Adaptive Algorithms.....22

Table 7.1 TEST1 Summary.....58
Table 7.2 TEST2 Summary.....73
Table 7.3 TEST3 Summary.....88
Table 7.4 TEST4 Summary.....102
Table 7.5 Comparison of Convergence Properties for the Adaptive Algorithms.....113

LIST OF SYMBOLS

N	number of antenna elements
$x_i(t)$	output of i^{th} antenna element
$S_i(t)$	signal component of output of i^{th} antenna element
$n_i(t)$	noise (deliberate and natural) component of output of i^{th} antenna element
$y(t)$	overall array output
w_i	i^{th} antenna weight
$d(t)$	reference signal
$\varepsilon(t)$	error signal
μ	convergence constant
$\underline{R}$	sample covariance matrix
$\underline{I}$	identity matrix
$\underline{C}$	constraint matrix
$\underline{P}$	projection matrix for constraint re-establishment
$\underline{\beta}$	orthogonal vector for constraint re-establishment
α	data de-weighting constant
$\underline{\lambda}$	vector of LaGrange multipliers
$\underline{f}$	response vector
ζ	mean square error

1.0 INTRODUCTION

Conventional antenna receiving systems are susceptible to performance degradation due to the presence of undesired noise signals (deliberate or natural) that enter the system. Extensive research has been conducted in the area of adaptive antenna arrays as a means of compensating for the inevitable presence of these interference signals.

Adaptive arrays can provide a vital element of flexibility to a communication system. They can respond to changes in the interference environment by steering nulls and reducing sidelobes in the directions of the interferences, while maintaining an acceptable level of response in the direction of the desired signal. More importantly, they can do so without prior information pertaining to the interfering signals. These features make adaptive array systems very attractive for applications in which the environment is changing or unknown. For example, it is quite possible that very little descriptive information concerning an intentional jamming signal will be available. Any system requiring such information could not effectively compensate for the interference.

Adaptive arrays also have a reliability advantage over their conventional counterparts. In a conventional array system, if a single sensor element becomes inoperable, the characteristic gain pattern may be noticeably affected, depending on the location of the non-working element and the and the total number of sensor elements. In contrast, an adaptive array system positions nulls and reduces sidelobles according to the monitored external signal/interference environment. Therefore, if a sensor element became incapacitated, the remaining operable elements would be adjusted to produce a pattern that is similar to the original pattern. Simply stated, adaptive array systems fail more gracefully than conventional receiving systems.

The heart of the adaptive array system is the controlling algorithm. It determines not only the method that is used to adapt, but also the speed of adaptation and the amount and complexity of hardware necessary to implement the system. Some of the early pioneering work in the area of adaptive control algorithms began in the 1960's. B. Widrow and others developed a self-training, self-optimizing control algorithm known as the Least Mean Square

(LMS) algorithm [1]. This algorithm uses gradient techniques to asymptotically approach an optimal solution. At approximately the same time, Howells and Applebaum were developing a sidelobe cancelling algorithm for radar applications [2]. This algorithm exploited the fact that the signal of interest was normally absent, and attempts to maximize a generalized signal-to-noise ratio. Numerous algorithms were developed shortly thereafter. Among the more common types are the Differential Steepest Descent algorithm, constrained algorithms such as Griffiths' P-vector and Frost's Constrained LMS, and random search algorithms [3-5].

The direct matrix inversion and recursive processors represent a significant departure from the above algorithms, and they were developed more recently. Although their heavy computational load renders them impractical for many applications, the advancements in cheap, fast digital hardware have spurred an increasing interest in these methods. Both methods involve finding the inverse of the sample covariance matrix to determine an optimum weight solution [7-9].

As mentioned, a wide variety of adaptive algorithms have been developed and utilized. This fact suggests that there is no "clearly superior" adaptive control algorithm that should be implemented without regard to the application. In fact, the performances of adaptive control algorithms are very application dependent. The selection of a specific algorithm is based on many factors including the quantity of a priori information, the convergence speed requirements, implementation cost considerations, and the transmission medium, among others. The process of selecting a control algorithm is further complicated by the fact that adaptive array systems are generally placed in changing surroundings. These natural uncertainties often make it impossible to analytically predetermine how well a specific algorithm will perform. This, combined with the fact that it is often economically unreasonable to build a system in order to test it, makes computer simulation a useful tool for array evaluation.

The importance of application considerations has been stressed. This study is concerned with the simulation of a communication system operating in the HF frequency band. The HF band, which spans the 3-30 MHz range, is commonly used in military communication systems, and has been modeled as a

slowly varying channel. Other factors that played a significant role in the algorithm selection process include:

- Knowledge of interleaved code is periodically available for reference signal generation.

- Knowledge of the angle of arrival of the desired communication signal is known a priori.

- Antenna array geometry.

Three adaptive control algorithms have been selected for computer simulation in this study. They were selected on the basis of their usefulness and applicability in the system of interest as well as the fact that they comprise a fairly representative set of adaptive algorithms in general. These algorithms are:

1. Least Mean Square Algorithm
2. Constrained LMS Algorithm
3. Update Covariance Algorithm (recursive)

The motivation for their selection as well as their attributes and operation will be discussed in detail later in the report.

Once the control algorithms had been selected, attention was turned to the development of the computer simulation models. The overall adaptive array system model was defined first, followed by the software implementation of the chosen control algorithms. In order to obtain relevant results regarding algorithm performance, it was imperative that the control algorithm be isolated in the overall system model. In this way, identical environments could be reproduced, and discrepancies in array performance could be attributed solely to the differences in algorithms. It was also necessary to define performance measures and develop a scheme to monitor the results in order to make a comparative evaluation of the selected control algorithms.

The first section of this report is devoted to the definition of the basic adaptive array system model. The principle system elements and their

operation are explained, and common notation which will be used in following discussions is presented.

Once these fundamental aspects have been examined, each of the selected algorithms is discussed in detail. Each discussion will include the motivations for that algorithm's selection, and a description of how the algorithm operates. It will also contain an explanation of the simulation model that has been used.

The next section gives a presentation of the testing procedure and all parameters that must be specified. The signal models, performance measures, and simulation processes will be explained. The final section presents the results in the form of graphs and offers conclusions concerning the algorithm recommendations.

2.0 ADAPTIVE ARRAY SYSTEM MODEL

The purpose of this section is to introduce the adaptive array system model that is used throughout the study. Although such a model is explicitly required for a computer simulation, it is primarily intended to aid in the understanding of the qualities and objectives of the simulated array sytem. A common representation of an adaptive array system is shown in Figure 2.1. This basic model, while not possessing an abundance of detail, can be used to examine the principle system elements and their functions.

Virtually all adaptive antenna array systems consist of three principle components as depicted in Figure 2.1. These components include: 1) An array of sensor elements; 2) A pattern-forming network; 3) An adaptive processor. Although not explicitly shown, it is also important to realize that the system is located in an environment in which the desired and deliberate jamming signals impinge on the array at various angles of azimuth and elevation in the presence of thermal noise. Each of the components will now be discussed.

2.1 Sensor Element Array

The array consists of N spatially separated sensor elements. Their function is to monitor the external signal and interference environment and distribute this information to the other system components. The sensor elements should be chosen according to their ability to perform in the specified medium. These elements can then be physically located in a planar configuration to produce a suitable gain pattern over the desired region. Care should be taken when choosing the type of sensor element and their locations, as both factors place fundamental limitations on the ultimate capability of the adaptive array system.

The output of each sensor element, $x_i(t)$, is simply the sum of the signal components that arrive at that element.

$$x_i(t) = s_i(t) + n_i(t)$$

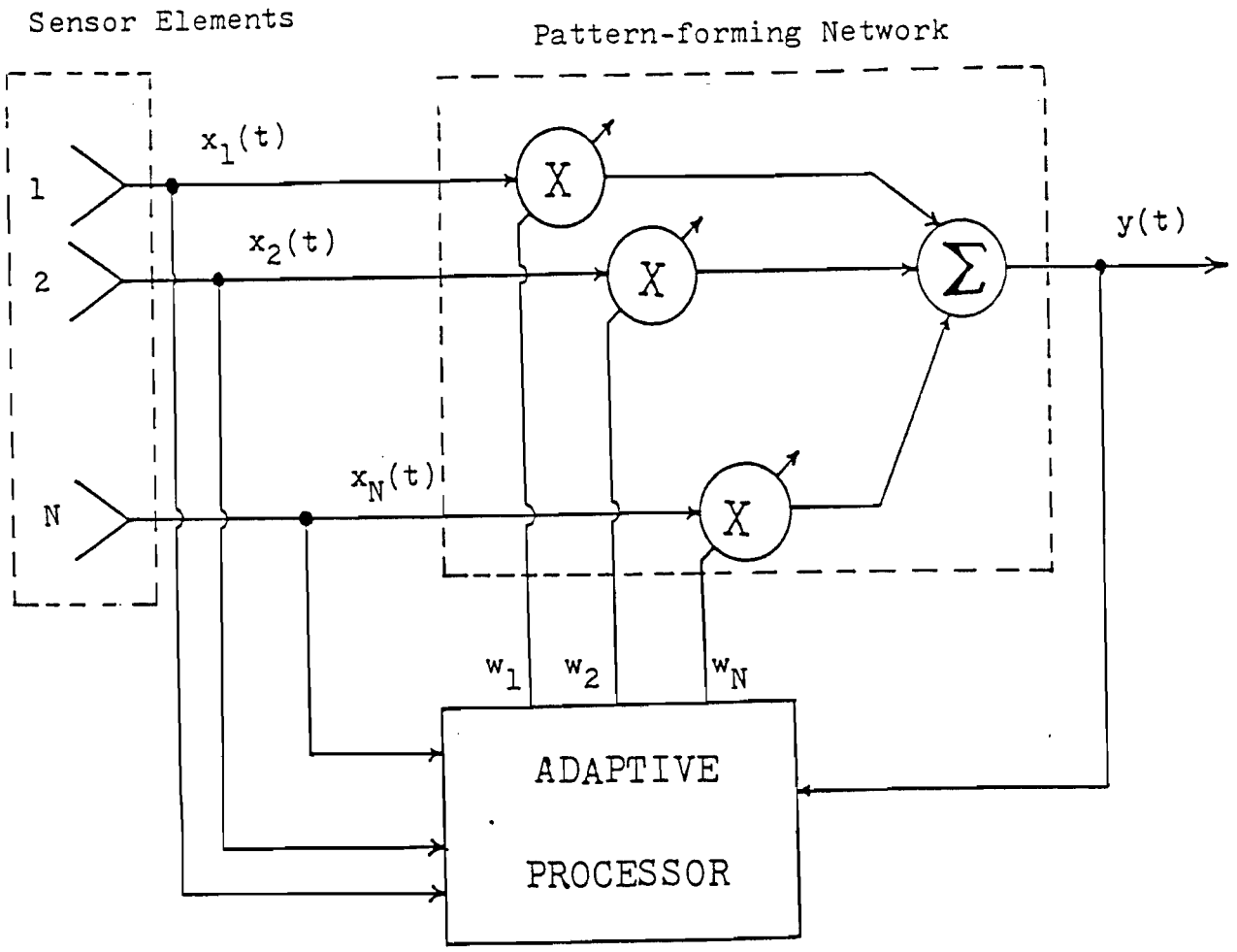


Figure 2.1 N-element Adaptive Array System Model

where

$s_i(t)$ is the desired signal component at element i

$n_i(t)$ is the combined noise components at element i from both deliberate and natural sources.

Also note that components of the same signal will differ from element to element due to phase delay caused by the spatial separation of the sensors.

The sensor element array model that has been utilized in this study contains some differences with models used in much of the related literature. The most notable departure from virtually every simulation conducted, is the use of a dipole model for the sensor elements. The use of isotropic elements is almost universal among simulated studies. Factors such as frequency dependence and directional gain of the elements themselves can then be ignored. Although using dipole elements complicates the model, "reality" is not compromised as severely. Also, many simulations are limited to linear array alignments. Such alignments make phase calculations almost trivial, but may inhibit the array's ability to distinguish signals on the basis of elevation arrival angle.

2.2 Pattern-Forming Network

The pattern-forming network consists of N variable complex weights and a summing device. The function of the network is very straight forward. It accepts a set of weights from the adaptive processor and multiplies each of them with their corresponding sensor element output. This multiplication can be thought of as an amplitude and phase adjustment of the element outputs. The complex weights can be written as

$$w_i = A_i \, e^{j\Theta_i} \tag{2.1}$$

where

$$A_i = \sqrt{(\mathrm{Re}\{w_i\})^2 + (\mathrm{Im}\{w_i\})^2}$$

and

$$\Theta_i = -\tan^{-1} \left(\mathrm{Im}\{w_i\} / \mathrm{Re}\{w_i\} \right)$$

Thus, the output of the sensor element i is assigned a gain, A_i, and delayed by Θ_i radians at the discretion of the adaptive processor.

The sensor output/weight products are summed to produce the overall array output signal, y(t), which can be written as

$$y(t) = \sum_{i=1}^{N} w_i \, x_i(t) = \underline{x}^T \underline{w} \qquad (2.2)$$

where the column vectors $\underline{x}$ and $\underline{w}$ are defined as

$$\underline{x} = \begin{matrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ \cdot \\ \cdot \\ \cdot \\ x_N(t) \end{matrix} \qquad \underline{w} = \begin{matrix} w_1 \\ w_2 \\ w_3 \\ \cdot \\ \cdot \\ \cdot \\ w_N \end{matrix} \qquad (2.3)$$

Throughout the report, matrices will be denoted with an underscored capital letter. Vectors will be denoted with an underscored lower case letter. Complex conjugates will be assigned a superscript asterisk (*), and transposistions will be denoted with a superscript "T". Scalar quantities will be written as a lower case letter with no special notation.

2.3 Adaptive Processor

The adaptive processor controls the operation of the array system. It is comprised of the control algorithm and any supplemental signal processing components required by that particular algorithm. The adaptive processor uses the the sensor element and array system outputs to compute a new set of complex weights. These weights are passed to the pattern-forming network to adjust the amplitude and phase of the sensor element outputs. The resulting array system response, which is hopefully acquiring a greater resemblance to the desired communication signal, is then used by the adaptive processor to help compute the next weighting vector.

3.0 LEAST MEAN SQUARE (LMS) ALGORITHM

The LMS algorithm was introduced and developed by Widrow in the mid 1960's [1]. It uses gradient estimation techniques to arrive at an optimal solution. The LMS algorithm is probably the most popular of all adaptive algorithms. It has been used in a variety of adaptive applications including channel equalization, noise cancellation, and antenna array systems. It has become somewhat of a standard and is frequently used as a performance barometer for other adaptive algorithms studies.

It is important to note that the LMS algorithm requires the generation of an error signal, which in turn requires the generation of a reference signal. This reference signal represents the signal that it is desired to receive and it is generated using some approximation technique. For some applications, the reference signal requirement is unattainable, and the LMS algorithm cannot be used. Communication systems in general, however, lend themselves to reference generation, and systems that involve the use of known codes have been shown to be particularly conducive to the generation of a satisfactory reference signal [11-13].

3.1 Motivation for Selection of LMS Algorithm

As its popularity suggests, the LMS algorithm has many attractive qualities. Probably its most attractive quality is its overall simplicity. Few computations must be performed in order to update the complex weights. It takes on the order of 2N computations, where N is the number of sensor elements, to update the weights. Also, the adjustment of one weight does not affect the adjustment of another. Therefore, the weights can be updated simultaneously. This keeps processing time to a minimum.

A natural outgrowth of the LMS algorithm's computational simplicity is the relatively small hardware requirements. It can be easily implemented in either analog or digital form. For many applications, the LMS algorithm represents a good trade off between speed of convergence* and implementational

---

* The speed of convergence is measured relative to the number of times the complex weighting vector must be updated before the antenna pattern has converged.

feasibility. As a general rule, algorithms with rapid convergence rates are very complex. The LMS algorithm, while not possessing convergence rates as rapid as those offered by recursive processors, have rates that are satisfactory for most slowly varying environments.

Finally, the operation of the LMS algorithm is straight forward and easily understood. The algorithmic steps are clear and well defined. This along with the other factors mentioned, make the LMS algorithm a prime candidate for our study.

The operation of the LMS algorithm is governed by the Mean Square Error performance criterion. Before discussing the inner workings of the LMS algorithm, this criterion will be explained as it adds valuable insight into how the algorithm operates.

3.2 Mean Square Error (MSE) Performance Criterion

The LMS-controlled adaptive array system is shown in Figure 3.1 and will be used to present the fundamental manner in which the MSE criterion is used. For the moment, assume that the reference signal, d(k), is the actual value that is sent by the desired communicator. An error signal, e(k), is defined as the difference between what was sent and what was received.

$$e(k) = d(k) - y(k) = d(k) - \underline{w}^T \underline{x}(k) \qquad (3.1)$$

The LMS algorithm uses this error signal, along with the sensor element output information, to calculate a new set of complex weight values. The weights are computed such that the resulting error signal, and thus the MSE, is reduced. As this process is repeated, the mean square error approaches zero, signifying that the value that was received was approximately equal to the value that was sent.

Although correct in principle, the preceding discussion has one blatant flaw. If the actual value sent by the communicator were truly known at the receiver, no information would be conveyed and there would certainly be no need to perform complicated adaptive techniques. The desired signal cannot be known with certainty at the receiver and therefore must be estimated. As mentioned, systems using known codes, such as the current system of interest, have been shown to be particularly adept at reference signal generation.

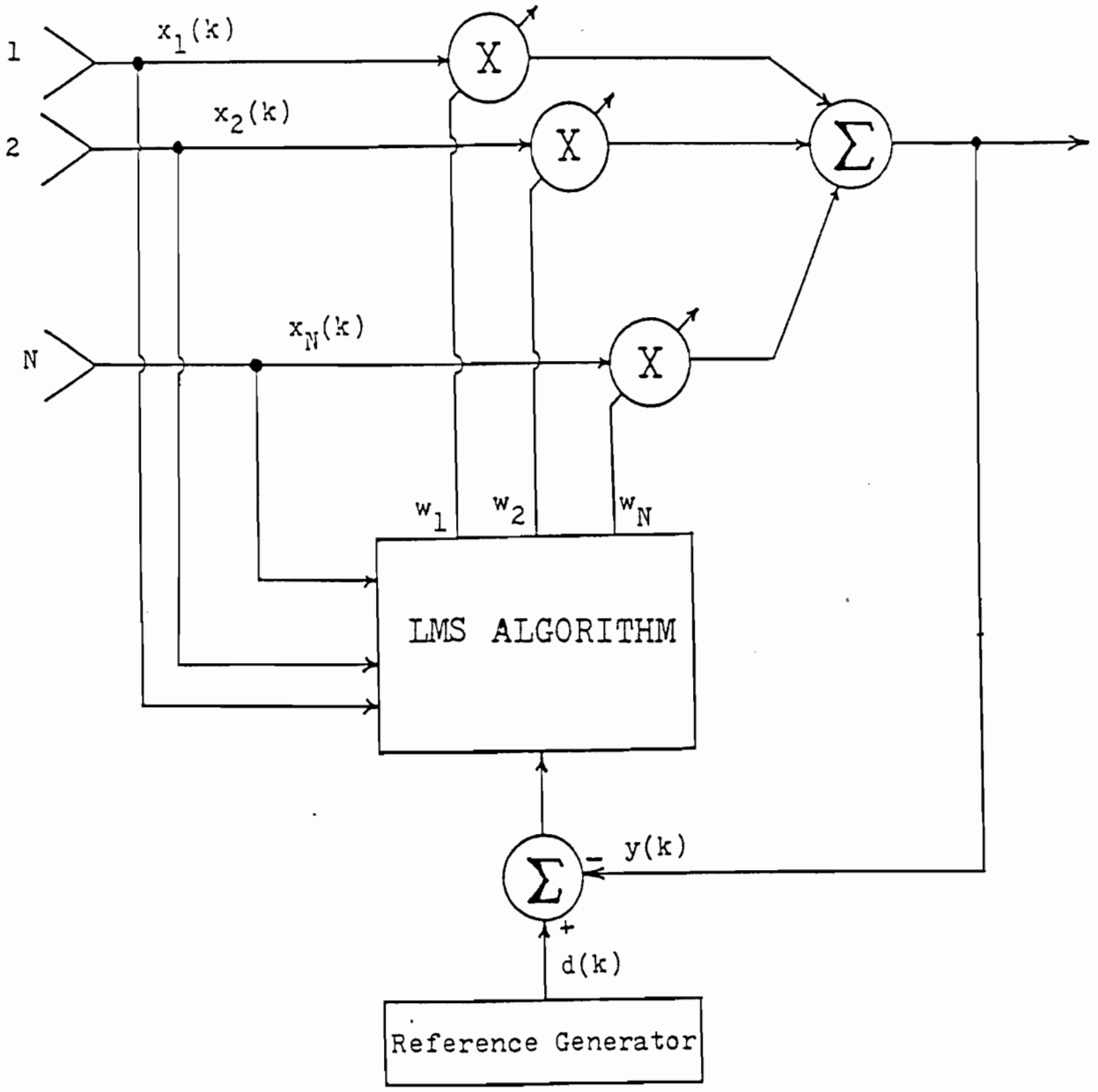


Figure 3.1 LMS-controlled Adaptive Array System

3.3 MSE Performance Criterion Derivation

The goal of the LMS algorithm is to adjust the complex weights in order to minimize mean square error. An expression of mean square error as a function of the weight values will now be derived.

The required error signal is defined as the difference between the generated reference signal and the adaptive array output signal.

$$e(k) = d(k) - y(k) = d(k) - \underline{w}^T \underline{x}(k)$$

The square of the error signal can then be written as

$$e^2(k) = d^2(k) - 2d(k)\ \underline{x}^T(k)\ \underline{w} + \underline{w}^T\ \underline{x}(k)\ \underline{x}^T(k)\ \underline{w} \qquad (3.2)$$

Taking the expected value of both sides yields

$$E\{e^2(k)\} = E\{d^2(k)\} + \underline{w}^T\ E\{\underline{x}(k)\ \underline{x}^T(k)\}\ \underline{w} - 2E\{d(k)x^T(k)\}\ \underline{w}. \qquad (3.3)$$

The above equation represents the mean square error as a function of the complex weights.

In order to simplify the notation, define the vector p as the cross correlation between the desired response and the sensor element output and the matrix R as the input correlation matrix.

$$p = E \begin{bmatrix} d(k)x_1(k) \\ d(k)x_2(k) \\ \cdot \\ \cdot \\ \cdot \\ d(k)x_N(k) \end{bmatrix} \qquad (3.4)$$

where d(k) is the scalar desired response and $x_i(k)$ is the output of sensor element i.

$$R = E\begin{bmatrix} x_1(k)\,x_1(k) & x_1(k)\,x_2(k) & \cdots & x_1(k)\,x_N(k) \\ \vdots & \vdots & & \vdots \\ x_N(k)\,x_1(k) & x_N(k)\,x_2(k) & \cdots & x_N(k)\,x_N(k) \end{bmatrix} \qquad (3.5)$$

Once again, $x_i(k)$ is the output of sensor element i. The mean square error can then be defined as

$$MSE = \xi = E\{e^2(k)\} = \underline{w}^T \underline{R}\,\underline{w} - 2\underline{p}\,\underline{w} + E\{d^2(k)\} \qquad (3.6)$$

It is very important to note that the MSE is a positive quadratic function of the weights. This function is a concave hyperparabaloidal (bowl-shaped) surface that contains no local minima. This is a performance surface and is depicted in Figure 3.2.

Notice that the minimum mean square error corresponds to the global minimum of the performance function. It is also known that the gradient of the function is zero at this minimum. Therefore, in order to find the optimal weight settings (the weights which produce minimum mean square error), the gradient of the function is found and set equal to zero.

The gradient is obtained by differentiating the MSE function with respect to the weights.

$$\nabla = \begin{bmatrix} \frac{\partial E\{\varepsilon^2(k)\}}{\partial w_1} \\ \vdots \\ \frac{\partial E\{\varepsilon^2(k)\}}{\partial w_N} \end{bmatrix} = -2\underline{p} + 2\,\underline{Rw} \qquad (3.7)$$

Setting the gradient equal to zero, the optimal weights are found.

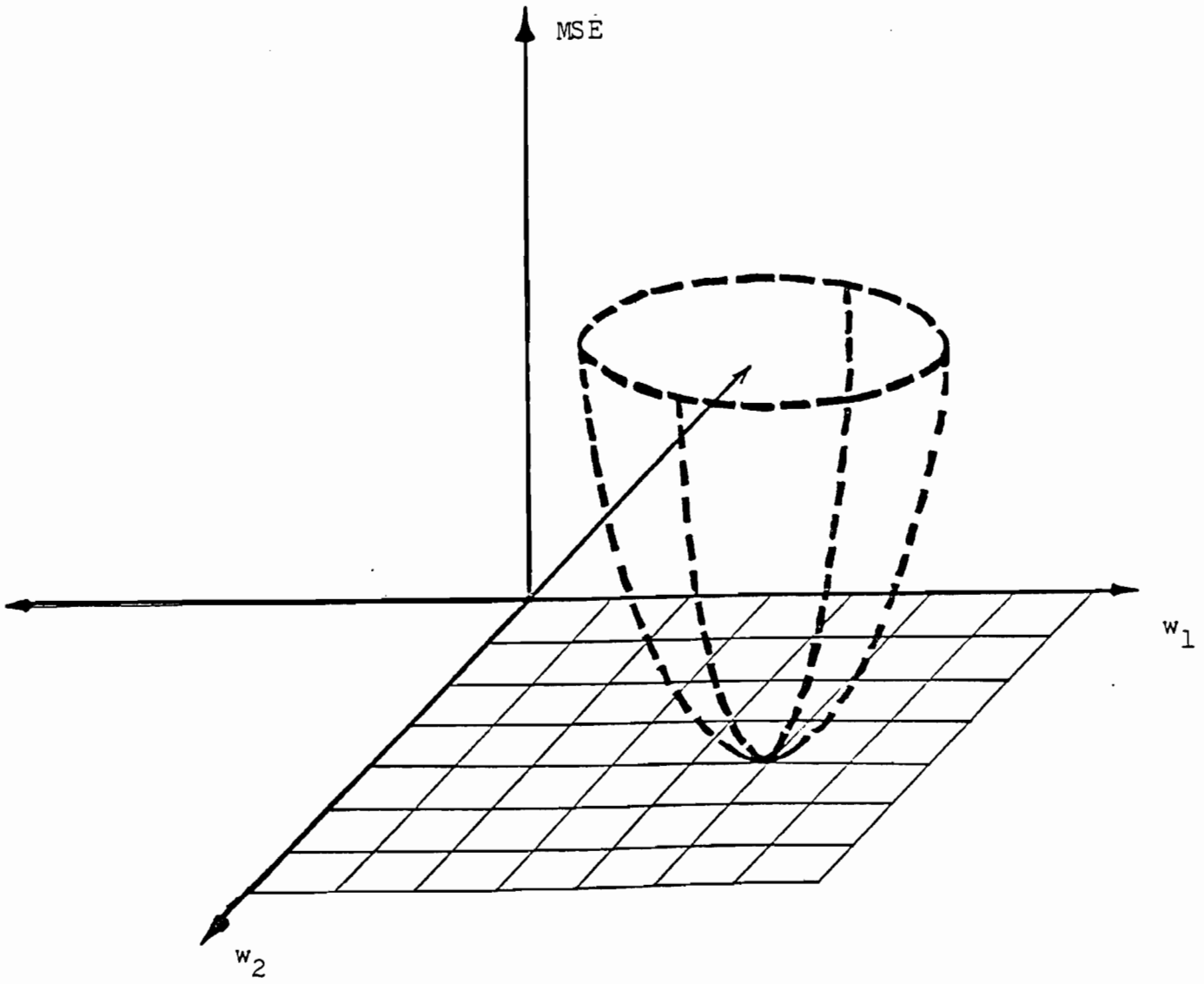


Figure 3.2 MSE Performance Surface

$$-2\underline{p} + 2\,\underline{Rw} = 0$$

$$\underline{w}_{OPTIMUM} = \underline{R}^{-1}\,\underline{p} \qquad (3.8)$$

This is an extremely important result and represents the matrix form of the Weiner-Hopf equation.

The Weiner-Hopf equation defines the optimal weight settings in the mean square sense. Intuitively, it may seem unreasonable to prescribe a complicated adaptive technique when an optimal solution is known. As will be shown, however, the solution is the problem, and the LMS algorithm is a method to avoid actually computing the Weiner-Hopf equation.

3.4 LMS Algorithm Description

An actual computation of the Weiner-Hopf equation would require the explicit measurement of all correlation functions, since it is unreasonable to assume that the correlations of deliberate interferences will be known. This is a plausible task but it would require large amounts of hardware. The requirement of matrix inversion, which for most arrays of practical size is computationally intensive, is more prohibitive. In fact, for most applications of reasonable magnitude, the process of directly inverting the correlation matrix is entirely unfeasible. The LMS algorithm is simply a practical method of finding close approximations to the Weiner-Hopf equation.

The LMS algorithm is an implementation of the Method of Steepest Descent. Using this method, the updated weight vector is equal to the past weight vector plus a change that is proportional to the negative gradient.

$$w(k + 1) = w(k) - \mu\nabla \qquad (3.9)$$

The parameter μ is a factor that controls stability and convergence rate. Updating the weights can be thought of as descending along the aforementioned performance surface in an attempt to reach the "bottom of the bowl."

The LMS algorithm avoids explicit correlation function measurement and matrix inversion by utilizing a crude but effective gradient estimate. Recall that the gradient of the MSE function is given by

$$\nabla = \begin{bmatrix} \frac{\partial E\{\varepsilon^2(k)\}}{\partial w_1} \\ \cdot \\ \cdot \\ \cdot \\ \frac{\partial E\{\varepsilon^2(k)\}}{\partial w_N} \end{bmatrix} = -2\underline{p} + 2\,\underline{Rw}$$

The LMS algorithm estimates the gradient by using the square of a single error sample instead of the MSE and differentiating with respect to the weights.

$$\hat{\nabla} = \begin{bmatrix} \frac{\partial \varepsilon^2(k)}{w_1} \\ \cdot \\ \cdot \\ \cdot \\ \frac{\partial \varepsilon^2(k)}{\partial w_N} \end{bmatrix} = -2\,\varepsilon(k)\,\underline{x}^*(k) \qquad (3.10)$$

Using this gradient estimate in place of the true gradient in (3.9) yields the LMS algorithm.

$$\underline{w}(k+1) = \underline{w}(k) + 2\mu e(k)\,\underline{x}^*(k) \qquad (3.11)$$

Notice that this algorithm does not require squaring, averaging or differentiation. The gradient estimate can be shown to be an unbiased estimator of the true gradient.

$$E\{\hat{\nabla}\} = E\{-2e(k)\ x^*(k)\}$$

$$= E\{-2[d(k) - y(k)]\ \underline{x}^*(k)$$

$$= -2E\{d(k)\ \underline{x}(k) - \underline{x}(k)\ \underline{x}^*(k)\ \underline{w}(k)\}$$

$$= 2(\underline{Rw} - \underline{p}) = \nabla$$

The LMS algorithm does not require the angle of arrival of the desired signal to be known a priori. If it is unknown, the weights are normally initialized to an arbitrary value of $1 < 0°$.

$$w(0) = \begin{bmatrix} 1 < 0^0 \\ 1 < 0^0 \\ \cdot \\ \cdot \\ \cdot \\ 1 < 0^0 \end{bmatrix}$$

If the angle of arrival of the desired signal is known, however, the initial weights can be chosen such that the initial antenna pattern effectively "looks" directly at the desired signal.

$$\underline{w}(0) = \begin{bmatrix} e^{-j\Omega_1} \\ e^{-j\Omega_2} \\ \cdot \\ \cdot \\ \cdot \\ e^{-j\Omega_N} \end{bmatrix}$$

where $-\Omega_i$ is a phase value that exactly compensates for the phase delay due to the spatial separation of the sensor elements. These values can be easily calculated if the angle of arrival and element locations are known.

3.5 LMS Software Modules

Simulation of the LMS algorithm requires two routines. The weights are initialized using the routine WEIGHTINIT. The initial weights are given as

$$w_i(0) = \exp(-j\Omega_i) \quad i = 1, 2, \cdots, N$$

where Ω_i is the phase of the desired signal at element i.

The routine LMS updates the weights according to the update equations that have been given. It requires both the sensor element outputs and the overall array system output to adjust the weights. It also requires a reference signal. In this model, it has been assumed that a known code is available. It is assumed that the code at the receiver is synchronized with the code (preamble) that is being sent. The channel model introduces delay, however, so the reference signal must be delayed accordingly if the synchronization assumption is to be satisfied.

The FORTRAN source code listings are given in Appendix C. Figure 3.3 depicts the modules discussed and illustrates the primary input and output parameters. The variable names used in the program are shown in parenthesis.

The LMS algorithm is not without its drawbacks. It does not converge terribly fast, but more importantly, it requires the presence of a reference signal to adapt. In cases such as this one where the reference signal is not always present, the weight values would have to be effectively frozen during the signal's abscence. Environmental changes in this period could not be tracked.

The severity of this problem will be dependent upon the rapidity of change of the application medium. A slowly varying medium should pose no prohibitive difficulties. An adaptive algorithm which does not encounter the problem of a required reference signal will now be examined.

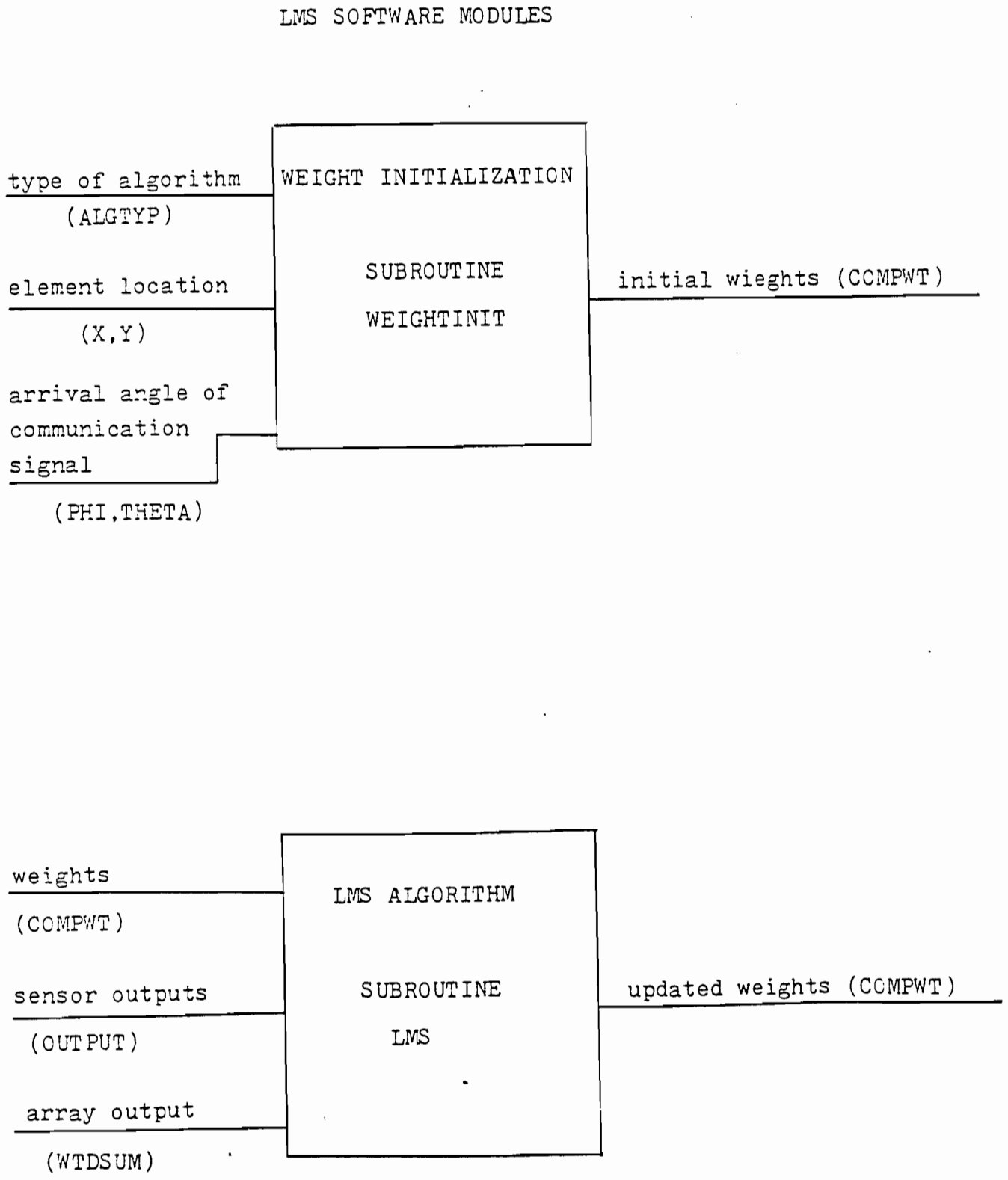


Figure 3.3 LMS Software Modules

4.0 CONSTRAINED LMS ALGORITHM

The second control algorithm selected for this study is the Constrained LMS algorithm. It was developed by O. L. Frost [5]. The name is somewhat misleading as it suggests that this algorithm is simply a permutation of the LMS algorithm. This is definitely not the case, and some of their contrasting features are illustrated below in Table 3.1. The name may have been derived from the fact that, like the LMS algorithm, the Constrained LMS algorithm uses a gradient approach.

4.1 Motivation for Selection

The primary advantage of the Constrained LMS algorithm is the elimination of the reference signal requirement. No reference signal must be present for the algorithm to reduce the effects of interferers or respond to changes in the environment. Complicated reference generation techniques can be avoided. It is, however, required that the arrival angle of the desired signal be known a priori. Secondly, the Constrained LMS algorithm requires relatively few computations in order to update the variable weights. Finally, many adaptive processors tend to degrade their own mainbeam response when attempting to place nulls in the directions of interferences. By explicitly constraining the response, the Constrained LMS algorithm prevents this from occurring.

4.2 Constrained LMS Algorithm Description

As mentioned, the Constrained LMS algorithm places fundamental limits on the adaptive array's response in the direction of interest (look direction). Throughout the adaptation process, the response in the direction of arrival of the desired signal will remain unchanged. The individual variable weights are allowed to take on any values in order to null out interferences provided that the look direction response is maintained. This type of algorithm allows the array to look in the direction of the desired signal (whether it is present or not) while ignoring signals arriving from other directions. It is true that an interfering signal that enters the array system at virtually the same angle as the desired signal will disturb the system. The same could be said for all algorithms because they rely on spatial separation to discriminate.

Table 3.1

Features of Adaptive Algorithms

Feature	LMS	Constrained LMS
Reference Signal	Required	Not Required
Angle of Arrival of Communication Signal	Not Required	Required
Performance Criterion	M.S.E.	Maximum-Likelihood
Optimization Technique	Minimize Error	Minimize output power according to constraints

The adaptive array system model which will be used to explain the operation of the Constrained LMS algorithm was presented in [5] and is shown in Figure 4.1. Although this simulation deals with narrowband signals, the original broadband processor model will be used in this discussion. The model consists of N elements and J taps per element. When narrowband signals are used a simplified model results and it will be described later. Also shown in Figure 4.1 is an "equivalent processor" which aids in the understanding of how the Constrained LMS algorithm operates.

From Figure 4.1, it is evident that the Constrained LMS processor contains an additional component. This component, known as a spatial correction filter, performs a task that is often regarded as preprocessing. This filter compensates for the physical misalignment of the sensor elements by introducing individual delays so that the desired signal effectively arrives at the same time at each element. In other words, the spatial correction filter guarantees that the communication signal component is identical at each element output. The delays can be calculated from the array geometry and the arrival angle of the desired signal. Noise components arriving at the sensors at other angles will not produce equal components at the element outputs.

From the desired signal's vantage point, the processors in Figure 4.1 are equivalent. Each adaptive weight in the equivalent processor is simply equal to the sum of the weights in the vertical column above it. With these values, the signal components at the respective processor outputs are identical. By assigning a value to these equivalent weights, a desired frequency response in the look direction is selected. This introduces J constraint conditions. Since there are N X J adjustable weights, the remaining N X J - J degrees of freedom can be used to minimize the non-look direction noise power. Minimizing non-look direction noise power is equivalent to minimizing total output power because, regardless of how the weights are adjusted, the constraints guarantee that the response in the look direction will not be degraded.

The basic manner in which the Constrained LMS algorithm operates has been discussed. For the purpose of clarity, the primary steps taken by the algorithm will now be re-emphasized. Delays in the spatial correction filter

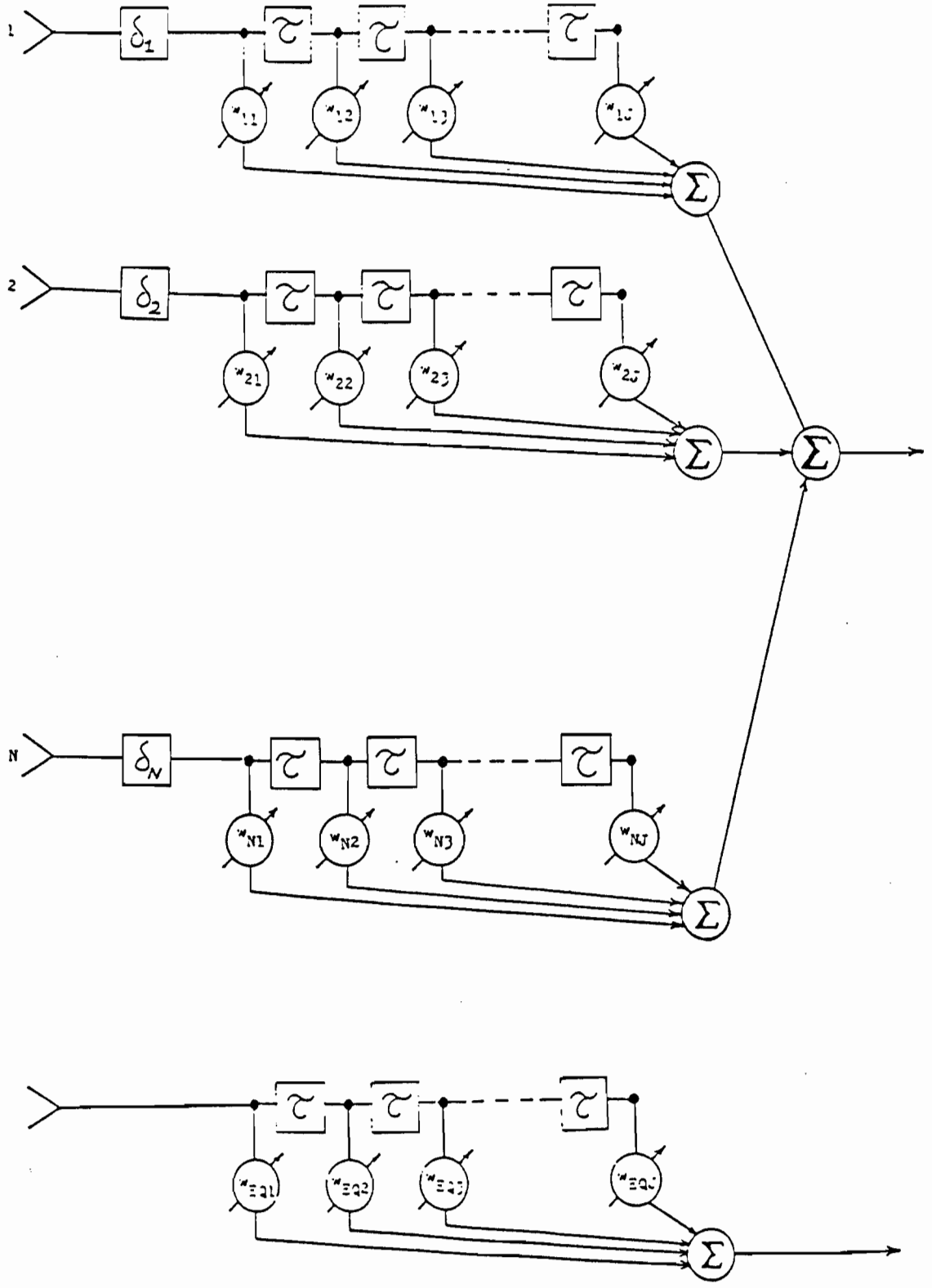


Figure 4.1 Signal-aligned Broadband Adaptive Array System

are calculated to align the communication signal components on the sensors. A desired response in the look direction is selected by assigning weight values to the equivalent processor (the sum-on-column constraints are determined). Once these tasks have been completed, adaptation begins and the processor strives to minimize the total output power. The constraints guarantee that there is no possibility of reducing power contributions made by the communication signal. Mathematical derivations of the optimum constrained weight solution and the Constrained LMS algorithm will now be presented.

4.3 Derivation of Optimum Constrained Weight Solution

The assumptions and definitions will be discussed first. Recall that the signals at the sensor element outputs can be written as the combination of the signal component and noise components

$$\underline{x}(k) = \underline{s}(k) + \underline{n}(k)$$

It is assumed that both the signal and noises can be modeled as zero-mean random processes with unknown second-order statistics. The covariance matrices are defined as follows:

$$E\{\underline{x}(k)\ \underline{x}^T(k)\} = R_{XX}$$

$$E\{\underline{s}(k)\ \underline{s}^T(k)\} = R_{SS}$$

$$E\{\underline{n}(k)\ \underline{n}^T(k)\} = R_{NN}$$

It is also assumed that the signal component is uncorrelated with the noise components.

$$E\{\underline{n}(k)\ \underline{s}^T(k)\} = 0$$

Finally, the expected value of the array output power is given by

$$E\{y^2(k)\} = E\{\underline{w}^T\ \underline{x}(k)\ \underline{x}^T(k)\ \underline{w}\} = \underline{w}^T\ \underline{R_{XX}}\ \underline{w} \qquad (4.1)$$

Recall that the adaptive weights in the equivalent processor dictated the frequency response characteristic in the look direction. Define a J-dimensional vector that guarantees the desired frequency response and represents the summed weight values of the j vertical columns as

$$\underline{f} = \begin{bmatrix} f_1 \\ f_2 \\ \cdot \\ \cdot \\ \cdot \\ f_N \end{bmatrix} \qquad (4.2)$$

The weights in the jth vertical column must sum to the selected number f_j. This constraint condition can be expressed as

$$\underline{c}_j^T \underline{w} = f_j \quad j = 1, 2, 3, \cdots, N \qquad (4.3)$$

where $\underline{c}_j$ is an NJ-dimensional vector consisting of all zeros and N ones given by

$$\underline{c}_j^T \quad [\underset{N}{000..0}....\underset{N}{000..0}....\underset{N}{111..1}....\underset{N}{000..0}....\underset{N}{000..0}] \qquad (4.4)$$

A constraint matrix can then be defined that satisfies all j equations given by (4.3) as

$$\underline{C} = [c_1 \cdots c \cdots c_J] \qquad (4.5)$$

The full set of constraints can then be written as

$$\underline{C}^T \underline{w} = \underline{f} \qquad (4.6)$$

Although it seems like a complicated process, the constraint matrix $\underline{C}$ simply guarantees that the sum of the weights in the vertical columns is equal to the weights in the equivalent processor.

The constrained optimization problem statement can now be formulated. The array output power, $\underline{w}^T \underline{\underline{Rxx}}\, \underline{w}$, must be minimized subject to the constraint condition $\underline{C}^T \underline{w} = \underline{f}$.

The optimum weight vector is found by using LaGrange multipliers. A cost function, similar in purpose to the MSE function of the LMS algorithm, is formed by concatenating the constraint equation with a J-dimensional vector of undetermined LaGrange multipliers $\underline{\lambda}$. This cost function is then minimized with respect to the weights.

$$\text{Cost}(w) = \tfrac{1}{2} \underline{w}^T \underline{\underline{R}}\, \underline{w} + \underline{\lambda}\, [\underline{C}^T \underline{w} - \underline{f}] \tag{4.7}$$

(a factor of $\tfrac{1}{2}$ is added to simplify the arithmetic)

Once again, notice that the cost function is a quadratic function function of the weights. It is known that the gradient of this function is zero at the minimum point. The optimum weights are then found by finding the gradient of the function and setting it equal to zero.

The gradient of the cost function is found by differentiating with respect to the weights.

$$\nabla_{COST} = \underline{\underline{Rxx}}\, \underline{w} + \underline{C}\, \underline{\lambda} \tag{4.8}$$

Setting this result equal to zero yields the optimal weight solution.

$$\underline{\underline{Rxx}}\, \underline{w} + \underline{C}\, \underline{\lambda} = 0$$

$$\underline{w}_{OPT} = - \underline{\underline{Rxx}}^{-1}\, \underline{C}\, \underline{\lambda} \tag{4.9}$$

The LaGrange multipliers are yet to be determined. They can be found by realizing that the optimal weight solution must satisfy the constraint condition.

$$\underline{C}^T \underline{w}_{OPT} = \underline{f} = C^T[-\underline{\underline{Rxx}}^{-1}\, \underline{C}\, \underline{\lambda}]$$

$$\underline{\lambda} = - [\underline{C}^T \underline{\underline{Rxx}}\, \underline{C}]^{-1}\, \underline{f} \tag{4.10}$$

The optimum constrained weight vector can now be expressed as

$$\underline{w}_{OPT} = \underline{\underline{Rxx}}^{-1} \underline{C}[\underline{C}^T \underline{\underline{Rxx}}^{-1} \underline{C}]^{-1} \underline{f} \qquad (4.11)$$

4.4 Derivation of Constrained LMS Algorithm

As in the LMS algorithm, this algorithm uses the Method of Steepest Descent. Recall that this method states that the new weight vector is equal to the previous weight vector plus a change proportional to the negative gradient.

$$w(k + 1) = w(k) - \mu \nabla_{COST}$$

In this case, the update equation can be expressed as

$$w(k + 1) = w(k) - \mu[\underline{\underline{Rxx}}\ \underline{w}(k) + \underline{C}\ \underline{\lambda}(k)] \qquad (4.12)$$

The initial weight vector, w(0), must satisfy the constraint condition. It is chosen as

$$\underline{w}(0) = \underline{C}(\underline{C}^T\ \underline{C})^{-1}\ \underline{f} \qquad (4.13)$$

The updated weight vector must satisfy the constraint condition as well. This can be written as

$$\underline{f} = \underline{C}^T\ \underline{w}(k + 1) = \underline{C}^T[\underline{w}(k) - (\underline{\underline{Rxx}}\ \underline{w}(k) + \underline{C}\ \underline{\lambda}(k))]$$

The LaGrange multipliers, $\lambda(k)$, are then given by

$$\lambda(k) = -[\underline{C}^T\ \underline{C}]^{-1}\underline{C}^T\ \underline{\underline{Rxx}}\ \underline{w}(k) - \frac{1}{\mu}\ [\underline{C}^T\ \underline{C}]^{-1}[\underline{f} - \underline{C}^T\ \underline{w}(k)] \qquad (4.14)$$

and the iterative relation for the update equation is expressed as

$$w(k+1) = w(k) - \mu[\underline{I} - \underline{C}(\underline{C}^T\underline{C})^{-1}\underline{C}^T]\underline{\underline{Rxx}}\ \underline{w}(k) + \underline{C}(\underline{C}^T\underline{C})^{-1}[\underline{f} - \underline{C}^T\ \underline{w}(k)]$$

For the sake of convenience, two definitions are made. Define the NJ-dimensional vector as

$$\underline{\beta} = \underline{C}(\underline{C}^T \underline{C})^{-1} \underline{f} \qquad (4.16)$$

and the NJ X NJ matrix P as

$$\underline{P} = \underline{I} - \underline{C}(\underline{C}^T \underline{C})^{-1}\underline{C} \qquad (4.17)$$

where I is the identity matrix.

The update equation can then be rewritten as

$$w(k + 1) = \underline{P}[\underline{w}(k) - \mu \underline{Rxx}\ \underline{w}(k)] + \underline{\beta}$$

The covariance matrix $\underline{Rxx}$ is unknown, however, so an approximation of $\underline{Rxx}$ at the kth iteration, $\underline{x}(k)\ \underline{x}^T(k)$, is used. Recognizing the fact that $\underline{x}^T(k)\underline{w}(k) = y(k)$, the final update equation becomes

$$w(k + 1) = \underline{P}[\underline{w}(k) - \mu y(k)\ \underline{x}(k)] + \underline{\beta} \qquad (4.18)$$

4.5 Constrained LMS Simulation Model

The tapped delay line in the broadband processor enables the user to select a desired frequency response in the look direction. This study is not concerned with such filtering because narrowband signal models are being used. For the purposes of this simulation, it is only necessary that the response of the adaptive array in the look direction be equal to unity. This response can be achieved in the broadband model by setting one weight in the equivalent processor equal to one and the remaining weights equal to zero. This is somewhat wasteful, however, as the same response can be obtained using the simplified model shown in Figure 4.2. The explanations and definitions that have been presented are still valid, but the tapped-delay line in the broadband model now consist of a single tap (weight). The weight in the equivalent processor is assigned a value of $1 < 0°$ so that the adaptive array has an all-pass distortionless response in the look direction. The sum of the

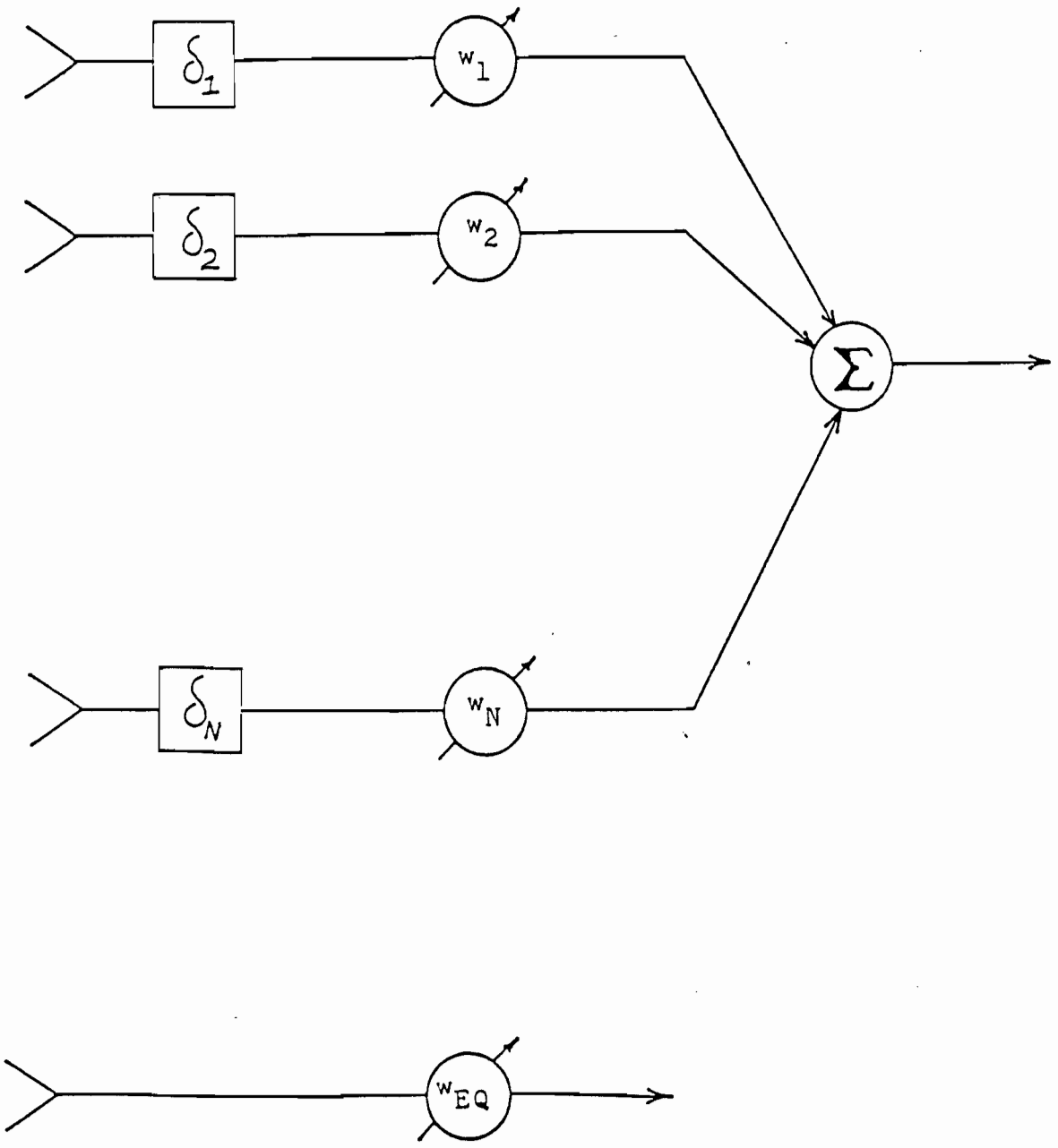


Figure 4.2 Narrowband Signal-aligned array system

weights in the single vertical column of the original processor are still required to equal that of the equivalent processor.

4.5.2 TAKAO Implementation

The Constrained LMS algorithm requires a spatial correction filter to compensate for the misalignment of the sensor elements. A method proposed by Takao et. al., [6] merges the misalignment compensation and the weight computation into a single process. The direction of arrival of the communication signal is used to generate a directional constraint to govern the weights. This method has been used in this simulation. An additional benefit of this implementation is that it helps to isolate the Constrained LMS processor from the other array system components, which was desired from the outset.

A description of the Takao implementation will now be presented. Note the similarities between these and the original equations [5], as they are for the most part equivalent.

$\underline{f} = 1 < 0°$ (only one weight in equivalent processor)

$(\underline{c}^*)^T \underline{w} = \underline{f}$ constraint equation

$\underline{w} = (w_1 + jw_2, w_3 + jw_4, \cdots w_{2N-1} + jw_{2N})$

$\underline{c}^T = (e^{j\Omega_1}, e^{j\Omega_2}, \cdots, e^{j\Omega_N})$

where Ω_i the phase of the desired signal at sensor element i

$\underline{P} = \underline{I} - (\underline{c}^* \underline{c})/N$

$\underline{\beta} = \underline{c}/N$

$\underline{w}(k+1) = \underline{P}\,[\underline{w}(k) - y(k)\,\underline{x}^*(k)] + \underline{\beta}$ update equation

4.6 Constrained LMS Software Modules

The weights are initialized using the WEIGHTINIT routine. They are assigned a value of

$w_i(0)$ = steering delay i/number of elements

The WEIGHTINIT subroutine is also responsible for calculating the individual steering delays that compensate for misalignment. These are given as

steering delay i = $\exp(-j\Omega_i)$

where Ω_i is the phase of the desired signal at element i.

Several quantities are calculated from directional information the first time the weight update routine CONLMS is called. These include

$$P = \underline{I} - (\underline{c}^* \underline{c})/N$$

$$\underline{\beta} = \underline{c}/N$$

$$\underline{c}^T = [e^{j\Omega_1}, e^{j\Omega_2}, \cdots, e^{j\Omega_N}]$$

Each time the routine is called, a new weight is calculated using

$$\underline{w}(k+1) = \underline{P}\,[\underline{w}(k) - \mu y(k)\,\underline{x}^*(k)] + \underline{\beta}$$

These updated weights are then passed to the pattern-forming network.

The FORTRAN source code listings are given in Appendix C. Figure 4.3 depicts the modules discussed and illustrates the primary input and output parameters. The variable names used in the program are shown in parenthesis.

The Constrained LMS algorithm also has some limitations. Although it does not require a reference signal, it does require prior information pertaining to the communication signal. It is vulnerable to steering error. Steering error arises if the supposedly known angle of arrival of the

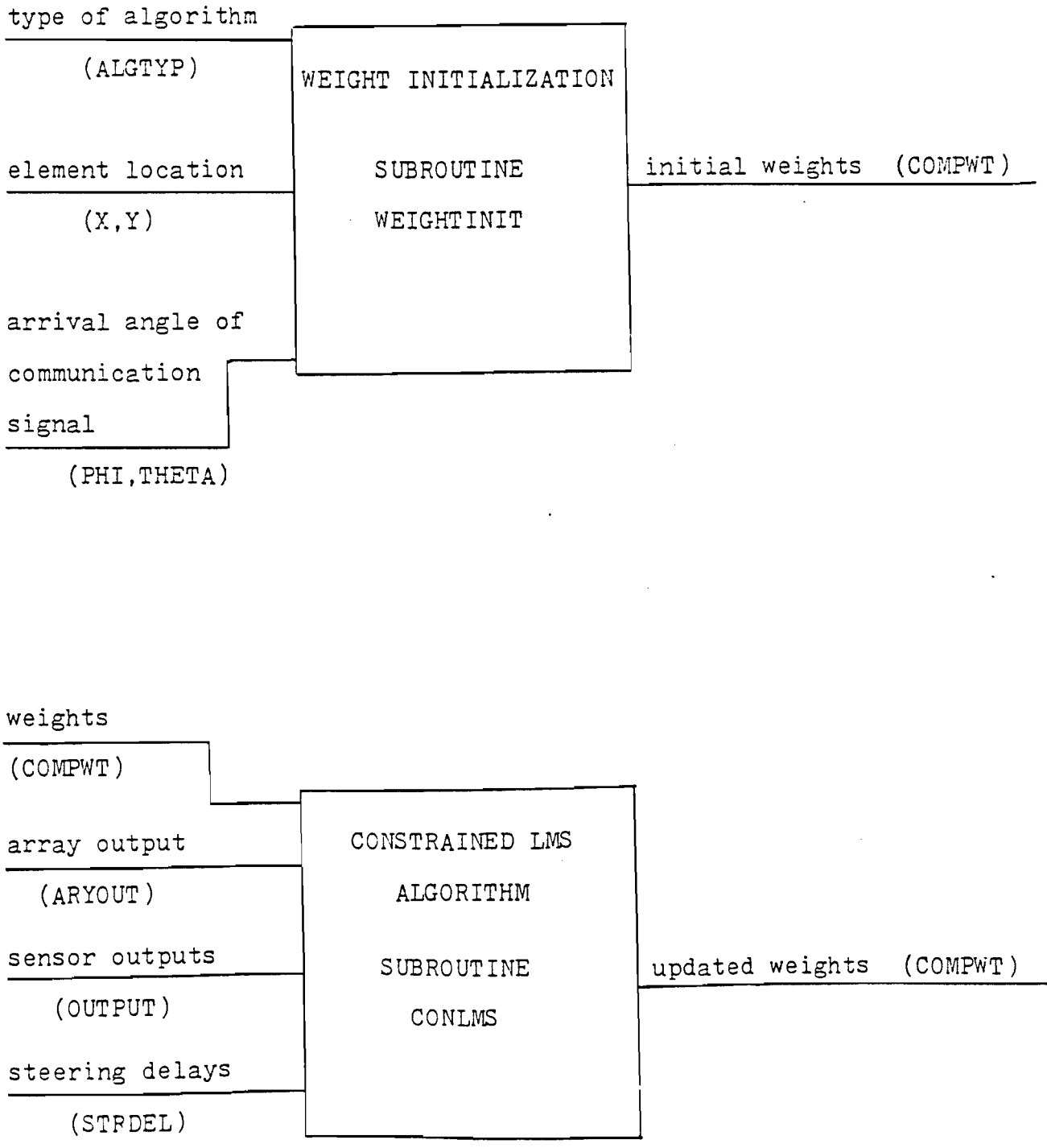


Figure 4.3 Constrained LMS Software Modules

information signal changes appreciably. In that event, the algorithm treats the signal as it would any other non-look direction signal -- by placing a deep null in its direction of arrival. The algorithm would have no way of knowing that the "interference" it is trying to ignore carries desired information. Secondly, its convergence rate is comparable to that of the LMS algorithm. An algorithm that possesses a faster response will now be examined.

5.0 UPDATE COVARIANCE ALGORITHM

Both the LMS and Constrained LMS algorithms circumvent computational problems associated with the direct calculation of a set of weights by using effective estimates. The simpler calculations that result allows them to frequently update the weights in order to compensate for the time-varying environment. Recursive processors such as the Update Covariance algorithm presented by Monzingo and Miller [7] can also be used to avoid these computational difficulties. These algorithms recursively perform matrix inversion so that direct matrix inversion is never required. Although they also avoid direct matrix inversion, recursive processors represent a significant departure from the algorithms previously discussed.

Recall that the optimum weight solution given by Weiner-Hopf can be expressed as

$$\underline{W}_{OPT} = \underline{R_{XX}}^{-1} \underline{p}$$

The Update Covariance algorithm and other recursive processors recursively estimate the sample covariance matrix rather than rely on gradient methods that asymptotically approach an optimal solution. These algorithms calculate the optimal set of weights at each sampling instant based on a least-squares fit to the received data.

5.1 Motivation for Selection

The primary reason for the selection of the Update Covariance algorithm is its speed of convergence. This characteristic, which is common among recursive processors, allows the algorithm to respond to changes more rapidly than other methods of adaptation. It is simply a more direct approach to the problem of computing an optimal weight solution. Updating the complex weights does not involve descending along a performance or cost surface at a limited rate.

Another beneficial quality of the Update Covariance algorithm is that no reference signal is required for adaptation. Once again, this eliminates the need for complicated reference generation techniques. It does, however,

require initial directional information pertaining to the communication signal.

Finally, recursive processors hold good promise for the future. They require a digital implementation and this has been, and still is, their primary disadvantage. The great technological strides made in the production of very fast, inexpensive, and compact digital hardware, however, has resulted in the consideration of recursive processors for applications that were previously out of the question. The improved convergence rates (measured in terms of iterations) offered by recursive processors have been documented [8] and, if technological trends continue, it may become implementationally feasible to exploit this advantage.

5.2 Update Covariance Algorithm Description

As mentioned, the Update Covariance processor estimates the sample covariance matrix in order to calculate an optimal weight solution. Unlike the other algorithms described, the operation of this algorithm can be described as a series of complex computations solely intended to calculate the optimal weight solution.

As the name implies, the Update Covariance algorithm uses the sample covariance estimate, $\underline{R}$, to summarize the effect of de-emphasizing the past data. The new sample covariance matrix estimate is given by

$$\hat{\underline{R}}_{\underline{XX}}(k+1) = \alpha\, \underline{R}_{\underline{XX}}(k) + \underline{x}^{*}(k+1)\, \underline{x}^{T}(k+1) \qquad (5.1)$$

The new estimate is equal to the new computed value $\underline{x}^{*}(k+1)\, \underline{x}^{T}(k+1)$ plus the past estimate scaled by a factor of α. α is a number between 0 and 1 that is used to determine the significance of past data. The inverse estimate then becomes

$$\hat{\underline{R}}_{\underline{XX}}^{-1}(k+1) = \frac{1}{\alpha}\left[\underline{R}_{\underline{XX}}(k) + \frac{1}{\alpha}\, \underline{x}^{*}(k+1)\, \underline{x}^{T}(k+1)\right]^{-1} \qquad (5.2)$$

Note that calculating the inverse in this manner however, would require matrix inversion, which is exactly what the algorithm is trying to avoid. Therefore it is useful to invoke the following matrix identity

$$[\underline{P}^{-1} + \underline{M}^{*T}\ \underline{Q}^{-1}\ \underline{M}]^{-1} = \underline{P} - \underline{P}\underline{M}^{*T}\ [\underline{M}\underline{P}\underline{M}^{*T} + \underline{Q}]^{-1}\ \underline{M}\underline{P} \qquad (5.3)$$

This identity is applied to equation 5.2 to obtain $\hat{\underline{R}}_{\underline{XX}}^{-1}(k+1)$ in the form

$$\hat{\underline{R}}_{\underline{XX}}^{-1}(k+1) = \frac{1}{\alpha}\left[\hat{\underline{R}}_{\underline{XX}}^{-1}(k) - \frac{\hat{\underline{R}}_{\underline{XX}}^{-1}(k)\ \underline{x}^{*}(k+1)\ \underline{x}^{T}(k+1)\ \hat{\underline{R}}_{\underline{XX}}^{-1}(k)}{\alpha + \underline{x}^{T}(k+1)\ \hat{\underline{R}}_{\underline{XX}}^{-1}(k)\ \underline{x}^{*}(k+1)}\right] \qquad (5.4)$$

The optimum weight solution can then be found by utilizing the Weiner-Hopf equation

$$\underline{W}_{OPT} = \underline{R}_{\underline{XX}}^{-1}\ \underline{p}$$

Multiplying both sides of equation 5.4 by the vector $\underline{p}$ yields the Update Covariance weight update equation.

$$w(k+1) = \frac{1}{\alpha}\left[w(k) - \frac{\underline{R}_{\underline{XX}}^{-1}(k)\ \underline{x}^{*}(k+1)\ \underline{x}^{T}(k+1)\ \underline{w}(k)}{\alpha + \underline{x}^{T}(k+1)\ \underline{R}_{\underline{XX}}^{-1}\ \underline{x}^{*}(k+1)}\right] \qquad (5.5)$$

The Update Covariance algorithm consists of the following two steps:

1. The inverse sample covariance estimate is formed using equation 5.4.

2. The weight solution is calculated using equation 5.5.

Due to the fact that the Update Covariance algorithm can be thought of as an entirely mathematical process, the simulation model will be dispensed with. The software associated with this algorithm simply performs the computations outlined in equations 5.4 and 5.5.

5.3 Update Covariance Software Modules

The weights are initialized using the WEIGHTINIT subroutine. These weights contain directional information which initially steers the antenna in the direction of the desired signal.

The first time the weight update routine UPDCOVAR is called, the inverse sample covariance estimate initialized to the identity matrix. After the first call, a new sample covariance estimate is formed by performing the necessary computations. This result is then used to calculate the new weight vector which is then passed to the pattern-forming network.

The FORTRAN source code is given in Appendix C. Figure 5.1 depicts the modules that have been discussed and illustrate the primary input and output parameters. The variable names used in the routines are shown in parenthesis.

By examining the update equations, it becomes quite evident that the Update Covariance algorithm, or any recursive processor for that matter, is computationally intensive. Although recursive estimation produces significant processing savings when compared to direct matrix inversion, it still represents somewhat of a quantum leap in terms of complexity when compared to the other algorithms presented.

Recall that the LMS algorithm required on the order of 2N computations to update the weights, where N is the number of sensor elements. The Update Covariance processor requires on the order of $5N^2$ computations to update the weights. Therefore, although the recursive processors require fewer iterations to converge, it may take a great deal of time to complete each iteration. For applications large in size (those having many elements), recursive processing may offer no improvement in actual convergence time over the gradient-based algorithms.

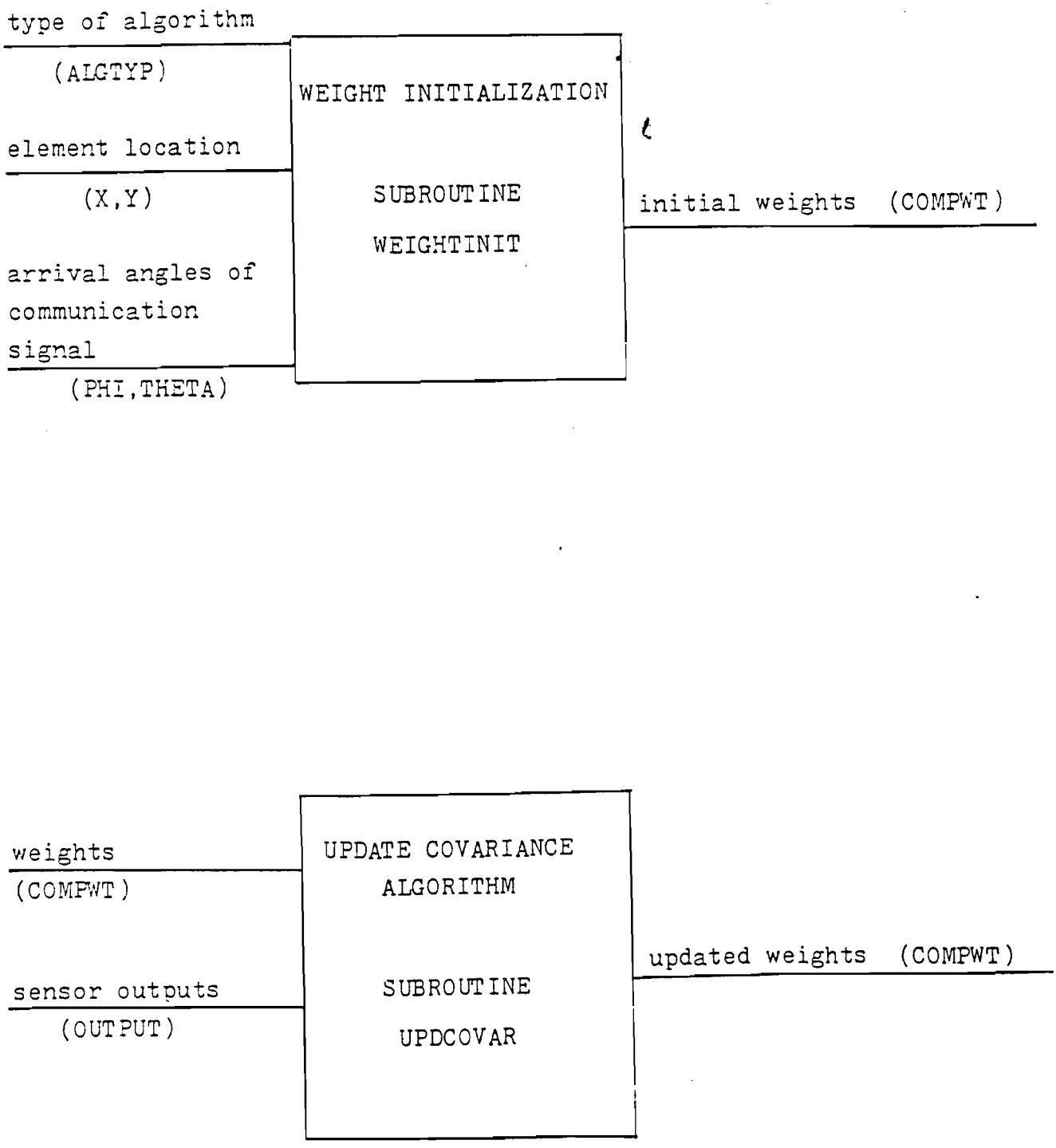


Figure 5.1 Update Covariance Software Modules

6.0 DESCRIPTION OF THE HF ADAPTIVE ARRAY SIMULATION MODEL

The purpose of this section is to explain the computer simulation program that has been developed to study adaptive algorithms for HF antenna arrays. All parameters that the user must specify will be discussed first. These parameters define the signal/interference environment as well as the array system to be tested. Secondly, the method used to evaluate the performance of the algorithms will be examined. Finally, the simulation software including signal models and the operation of the program will be presented.

6.1 User-Definition of Array System and Environment

Before an adaptive array system can be evaluated, the user must define the array system and environment to be studied. The parameters that must be specified can be divided into the following classes:

1. Signal characteristics and environment
2. Interference environment
3. Array system
4. HF channel characteristics
5. Convergence characteristics

A complete listing of the user-specified parameters is given in Figure 6.1. Appendix B contains the user manual for the simulation, giving complete descriptions of the above parameters, as well as the actual user interface. In addition to a description of the input process, the manual also defines user options for viewing the simulation output.

6.2 Performance Evaluation

The performance of the adaptive algorithms will be evaluated using a maximum signal-to-noise criterion. This criterion will now be examined.

Recall that the output of the array can be expressed as

$$y(t) = \underline{w}^T \underline{x}(t)$$

	UNITS
ARRAY SYSTEM	
Number of sensor elements	---
Length of antennas	meters
Location of elements in rectangular coordinates	meters
Adaptive algorithm	---
SIGNAL CHARACTERISTICS	
Arrival angles, azimuth and elevation	degrees
Number of Samples/bit	---
INTERFERENCE CHARACTERISTICS	
Number of Jamming signals	---
Arrival angles, azimuth and elevation	degrees
J/S ratio	dB
S/N(thermal) ratio	dB
HF CHANNEL CHARACTERISTICS	
Number of signal paths	---
delay of each path	msecs.
Attenuation of paths	dB
CONVERGENCE CHARACTERISTICS	
Number of Convergences	---
SNR tolerance	dB

Figure 6.1 User-entered parameters

where x(t) contains both signal and noise components.

The array output can then be divided into signal and noise components.

$$y_s(t) = \underline{w}^T \underline{s}(t) \qquad y_n(t) = \underline{w}^T \underline{n}(t)$$

The expected signal and noise power at the array output is given as

$$E\{|y_s(t)|^2\} = \overline{|\underline{w}^T \underline{s}|^2} = \underline{w}^{*T} [\underline{s}^* \underline{s}^T] \underline{w} = \underline{w}^{*T} \underline{R}_{ss} \underline{w}$$

$$E\{|y_n(t)|^2\} = \overline{|\underline{w}^T \underline{n}|} = \underline{w}^{*T} [\underline{n}^* \underline{n}^T] \underline{w} = \underline{w}^{*T} \underline{R}_{nn} \underline{w} \qquad (6.1)$$

Therefore, the signal-to-noise (noise + interference) ratio can be calculated as

$$SNR = \frac{\underline{w}^{*T} \underline{R}_{ss} \underline{w}}{\underline{w}^{*T} \underline{R}_{nn} \underline{w}} \qquad (6.2)$$

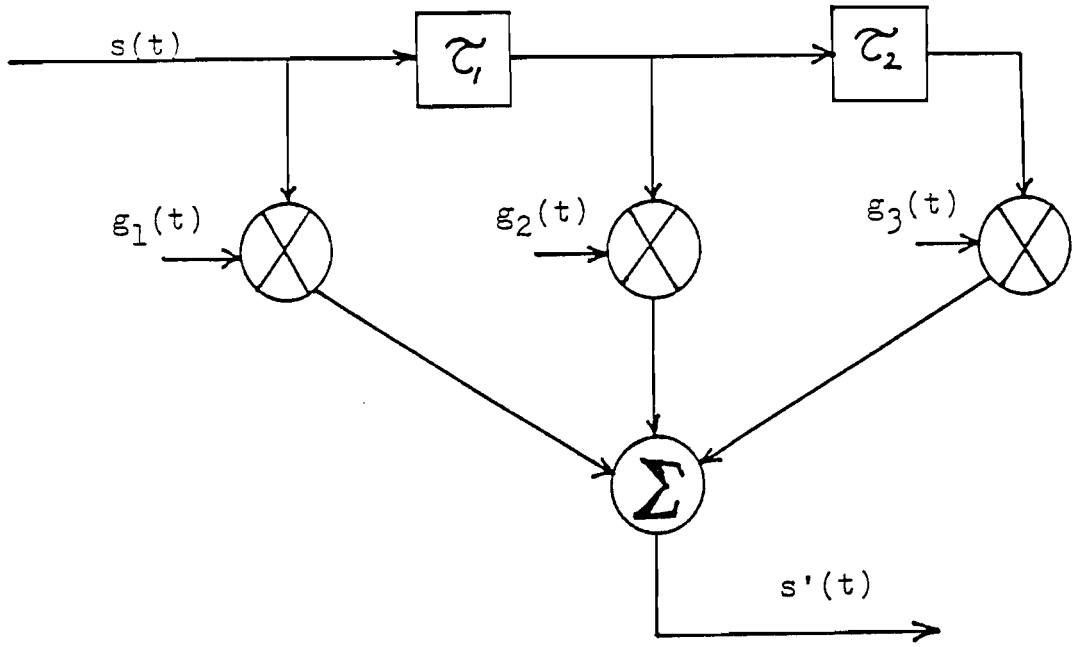
The optimum SNR can be computed using a matrix transformation as given by Monzingo and Miller [7]. The optimum SNR is given as

$$SNR_{OPT} = \underline{s}^T \underline{R}_{nn}^{-1} \underline{s}^* \qquad (6.3)$$

As the name implies, the maximum achievable SNR is used to evaluate algorithm performance. The goal (optimum SNR) is known, and by computing the current SNR, it is possible to observe the degree of success that the algorithm is having in attaining this goal. The model is executed until it has reached a user set SNR goal.

This study is primarily interested in finding the average number of times that a particular algorithm must update the weights (iterations) before it has converged as function of the HF channel. It is useful to first consider a three-path HF channel model as depicted by Figure 6.2. The HF channel is characterized by the delays between the paths and the attenuation of each path (determined by the variances of the random complex numbers, $g_i(t)$). By selecting a set of delays and attenuations, an HF channel model is defined.

THREE-PATH CHANNEL MODEL



$g_i(t)$ is a complex gaussian zero-mean random process

The variance of $g_i(t)$ determines the attenuation of the respective path

Figure 6.2 Three-path HF Channel Model

Notice, however, that it is possible to get different channel "realizations" using the same characteristic channel model (same delays and variances) by simply using different random numbers. This plays an important role in the performance evaluation.

Due to the fact that the HF channel is slowly varying, a single channel realization is selected for each adaptation. The optimum SNR, which is channel dependent, is then calculated and adaptation begins. The current SNR is periodically calculated and if this value is not sufficiently close to the optimum value (proximity to optimum entered by user and known as SNR tolerance), the adaptation process is continued. When the SNR does approach the optimum SNR, the algorithm is said to be "converged" and the number of required iterations is recorded. A new HF channel realization (a new set of random numbers for the tap weights) is then selected. The entire process is repeated until the algorithm has converged the specified number of times (user-entered). At that time, the average number of iterations is calculated.

By performing this simulation for each of the selected algorithms, it will be possible to determine, on the average, which algorithm converges in the fewest number of iterations in an HF environment.

6.3 Simulation Software Description

The purpose of this section is to explain the signal models and the operation of the simulation program used in the study.

6.3.1 Desired Signal Model

The signal that is to be received is assumed to be a BPSK waveform at a carrier frequency f_c. This can be written in complex form as

$$s(t) = A(t) \sqrt{P_s} \exp(jw_c t)$$

where

P_s = desired signal power and
$A(t) = +1, -1$

The signal component at element i can be written as

$$s_i(t) = A(t) \sqrt{P_s} \exp(jw_c t) \exp(j\Omega_i) \qquad (6.4)$$

where Ω_i is the phase of the desired signal at element i.

Ω_i is calculated using the arrival angle information and the location of the elements.

$$\Omega_i = 2*\pi*f_c * xrot(i)*\sin\Theta/c \qquad (6.5)$$

where

$xrot(i) = y(i)\cos\ \phi + x(i)\sin\ \phi$
$x(i),y(i)$ = rectangular coordinates of element i
ϕ = azimuthal arrival angle of desired signal
θ = elevation arrival angle of desired signal
c = speed of light

Using complex envelope representation, the carrier component can be dropped from the notation. The jammer power and thermal noise power are defined as ratios relative to the signal power, P_s. For the sake of simplicity, the signal power is assigned a value of 1.

Equation 6.4 can be written for discrete sampling as

$$s_i = A(k) \sqrt{P_s} \exp(j\Omega_i)$$

The signal vector then becomes

$$\underline{s}(k) = A(k) \sqrt{P_s} \begin{bmatrix} e^{j\Omega 1} \\ e^{j\Omega 2} \\ \vdots \\ e^{j\Omega N} \end{bmatrix}$$

6.3.2 Interference Model

The jamming signals in this study have been modeled as complex Gaussian noises. This can be expressed as

$$nj(t) = \sqrt{Pj/2}\ [E(t) + j\ F(t)] \qquad (6.6)$$

where E(t) and F(t) are zero-mean random processes with a gaussian distribution and a variance of 1. The power of the jammer is calculated from the user-specified jammer-to-signal ratio.

The jamming signal component at each element is defined as

$$nj_i(t) = nj(t)\ \exp(j\Omega_i)$$

where Ω_i is the phase of the jamming signal at element i. It is calculated in the same manner previously discussed using directional and element location information.

6.3.3 Thermal Noise Model

In this study, thermal noise has been modeled by adding complex Gaussian Noise to each element. This can be expressed as

$$n_t(t) = \sqrt{P_n/2}\ [E(t) + j\ F(t)] \qquad (6.8)$$

where E(t) and F(t) are zero-mean processes with a Gaussian distribution and variance of 1.

The power, P_n, is calculated from the user-specified signal-to-noise ratio.

6.3.4 Correlation Matrices

In order to evaluate the performance using a maximum signal-to-noise criterion, it is necessary to compute correlation matrices. These will now be defined.

Recall that the desired signal can be expressed as

$$s_i(k) = A(k)\sqrt{P_s}\exp(j\Omega_i)$$

The correlation matrix can then be written as

$$\underline{R_{ss}} = P_s \begin{bmatrix} e^{-j0} & e^{-j(\Omega_1 - \Omega_2)} & \cdots & e^{-j(\Omega_1 - \Omega_N)} \\ e^{-j(\Omega_2 - \Omega_1)} & e^{-j0} & \cdots & e^{-j(\Omega_2 - \Omega_N)} \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ e^{-j(\Omega_N - \Omega_1)} & e^{-j(\Omega_N - \Omega_2)} & & e^{-j0} \end{bmatrix} \quad (6.9)$$

The noise correlation matrix can be written as the sum of the individual noise matrices.

$$R_{nn} = R_{Jammer} + R_{NTHERMAL} \quad (6.10)$$

where

$$R_J = \begin{bmatrix} e^{-j0} & e^{-j(\Omega_1 - \Omega_2)} & \cdots & e^{-j(\Omega_1 - \Omega_N)} \\ e^{-j(\Omega_2 - \Omega_1)} & e^{-j0} & & e^{-j(\Omega_2 - \Omega_N)} \\ \cdot & \cdot & \cdot & \cdot \\ e^{-j(\Omega_N - \Omega_1)} & & & e^{-j0} \end{bmatrix} \quad (6.11)$$

and $R_{NTHERMAL} = \sigma^2\underline{I}$ (6.12)

As previously mentioned, the optimum SNR is given by

$$SNR_{opt} = \underline{s}^T \underline{R_{nn}}^{-1} \underline{s}$$

The current SNR can be written as

$$SNR = \frac{\underline{w}^{*T} \underline{\underline{R}}_{ss} \underline{w}}{\underline{w}^{*T} \underline{\underline{R}}_{nn} \underline{w}}$$

The optimal value is calculated first. The simulation is started and adaptation begins. The SNR is periodically checked, and when it is sufficiently close to the optimum value the simulation is stopped.

6.3.5 Simulation Program Operation

The flow chart of Figure 6.3 depicts the order in which the described operations are performed. The following discussion is a brief discussion of how the program operates.

The simulation begins with the user specifying the defining parameters such as the element locations, selected algorithm, arrival angles, and relative signal strengths, among others. The program must then calculate the optimum SNR for a particular channel realization. This value will be used to determine when the algorithm has converged. The antenna weights are then initialized and the adaptation process begins.

The desired signal and jamming signals are determined first using the models described. These signals are then passed to the HF channel model. These "channelized" signals are then passed to the antenna routine which calculates the contributions of all signals at each antenna element. The output of each sensor element is multiplied by its corresponding weight, and these products are summed to form an overall array system output. This overall output and the output of each element is then passed to the selected algorithm. The algorithm uses this information to adjust the weights.

The weights are updated by the selected algorithm and the current SNR is periodically computed. If this SNR is not close to the optimum value, adaptation continues. If it is, however, the converged weights and the number of convergence iterations are stored in a file. A new channel realization is selected, the parameters are re-initialized, and the process is repeated.

When the algorithm has converged a sufficient number of times (user-specified), overall statistics are gathered and the simulation is complete.

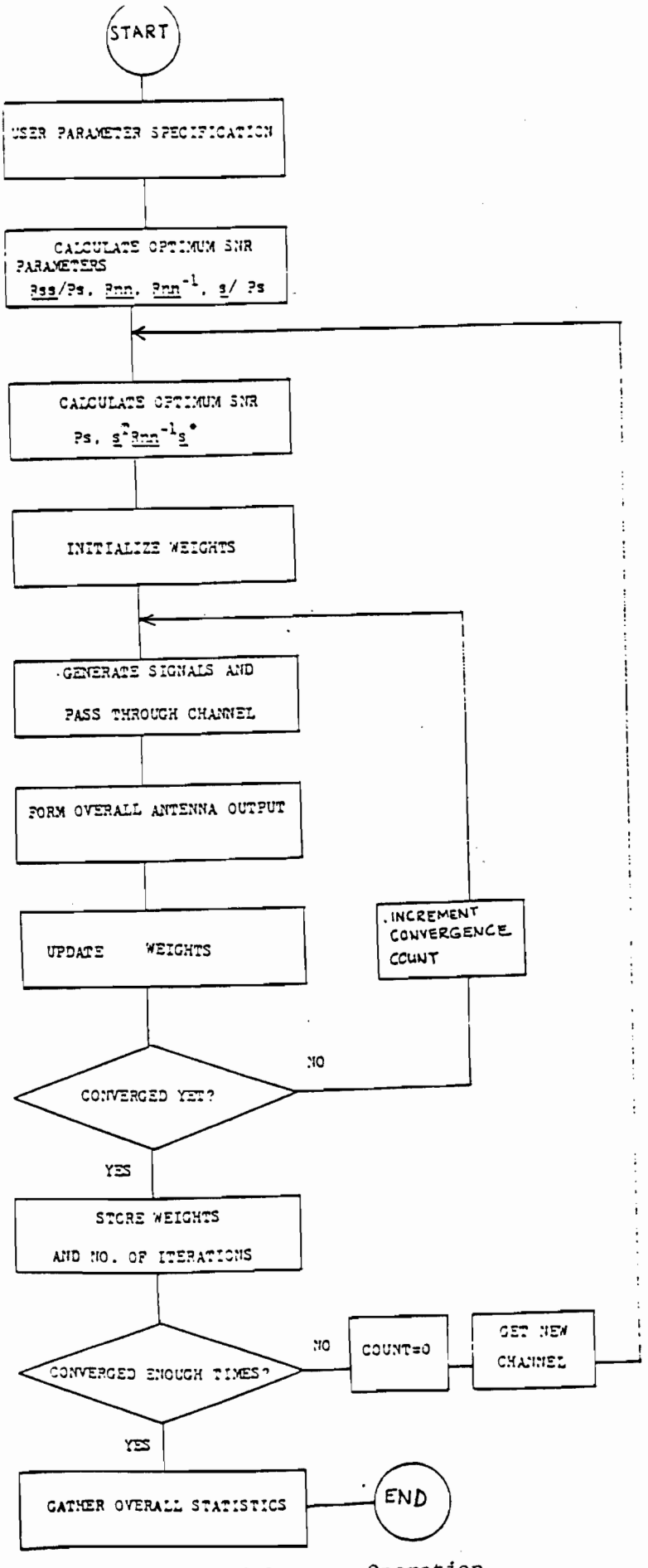


Figure 6.3 Flow-diagram of Program Operation

7.0 SIMULATION RESULTS AND CONCLUSIONS

The purpose of this section is to illustrate the results that have been obtained using the simulation model described in the previous section. From these results, conclusions concerning the applicability of the particular algorithm to the problem of interest can be drawn.

In all of the tests to be conducted, certain parameters will remain constant. These include arrival angles and signal powers and are summarized in Figure 7.1. For each test, the average number of required convergence iterations will be given to demonstrate how quickly the algorithms approch optimality in the maximum-SNR sense. Also, polar plots will be presented in order to pictorally demonstrate the nulling capabilities of the algorithms. These plots consist of two "slices" of each antenna pattern at the azimuth and elevation angles for each incoming signal (see Figures 7.2 and 7.3 for a definition of the geometry).

SIGNAL	AZIMUTHAL ANGLE	ELEVATION ANGLE	POWER
COMMUNICATOR	90.0	60.0	0 dB
JAMMER 1	90.0	80.0	20 dB
JAMMER 2	150.0	75.0	20 dB

S/N(thermal) - 10 dB Carrier frequency - 30 MHz

SENSOR ELEMENT LOCATIONS

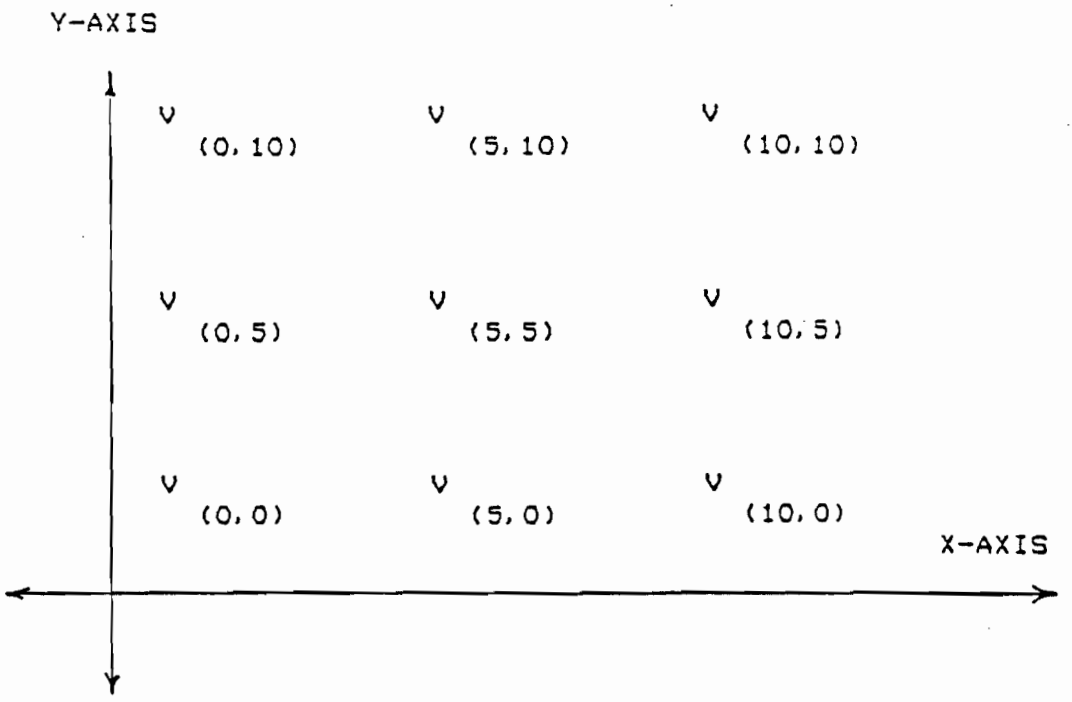
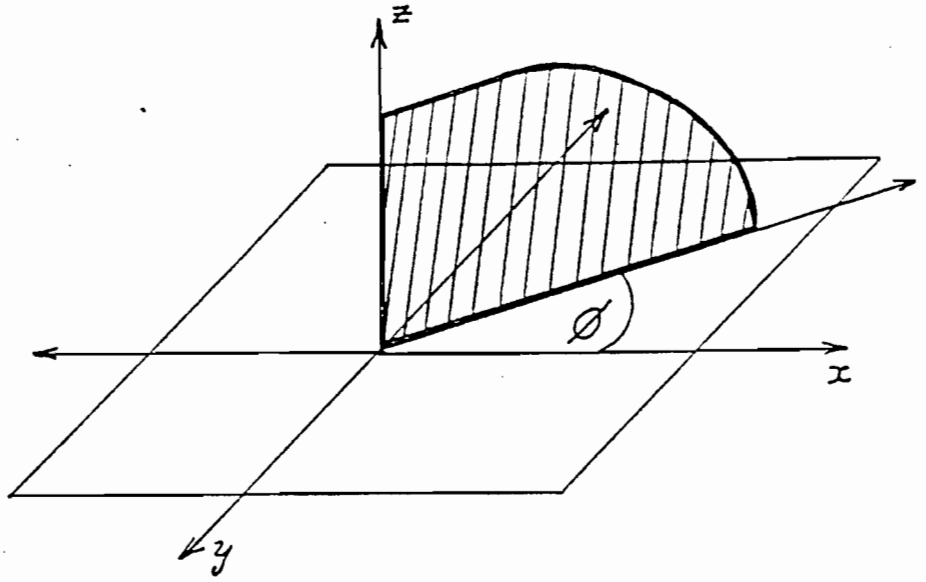


Figure 7.1 Constant Test Parameters



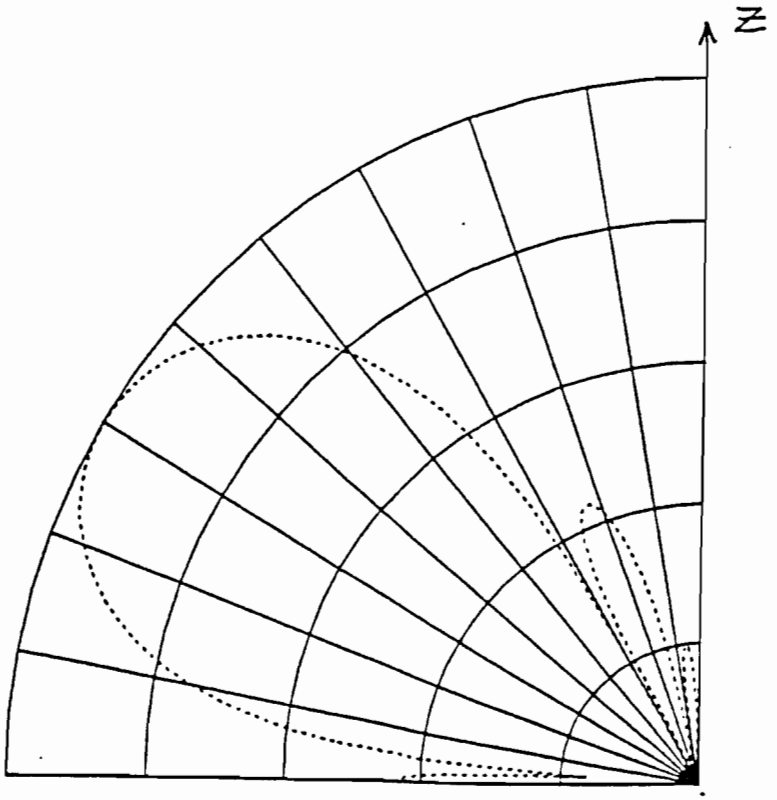
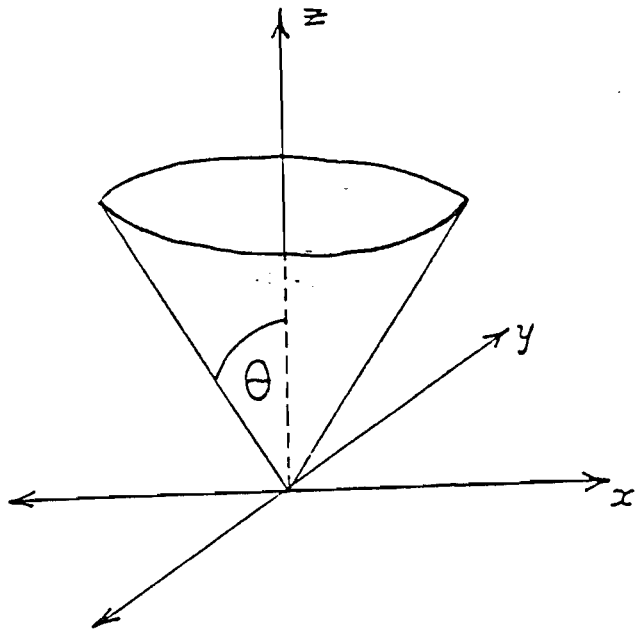


Figure 7.2 Description of Elevation Plots



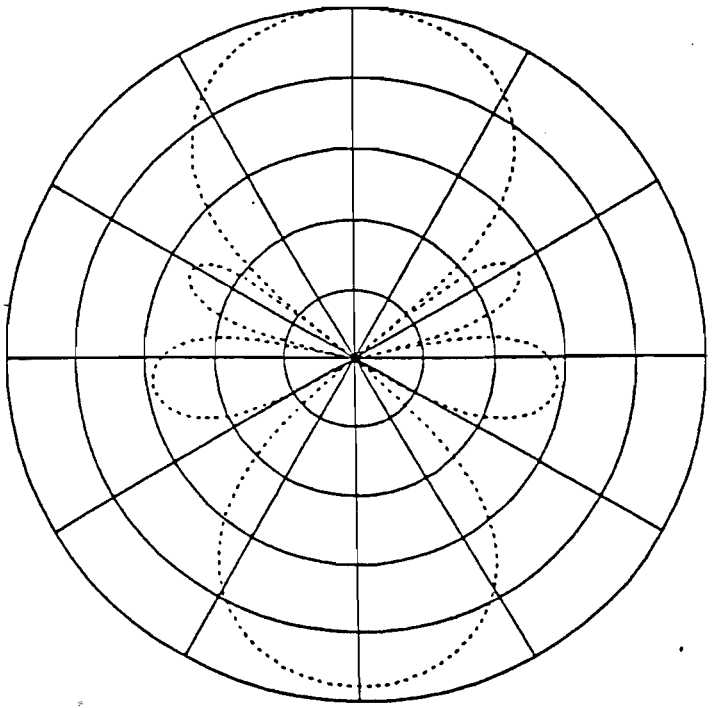


Figure 7.3 Description of Azimuthal Plots

7.1 Test 1: Negligible Channel Effects

It may be enlightening to first consider the performance of the algorithm in an unperturbed channel environment. In this way, the degrading effects of the HF channel can be monitored as well. Although it is not a terribly realistic case, it may become so if some efficient channel compensation technique could be employed. Recall that in this study the array model consists of one weight per element. This allows for beam steering, but the array cannot compensate for the smearing effects of the channel.

This case, although utilizing an ideal channel, is far from a trivial one. In fact, simulation studies frequently limit themselves to an ideal channel, because of the task of nulling out strong jamming signals is a difficult one even in this setting.

The results for all algorithms were averaged over 100 convergences. The number of iterations for each convergence was recorded to produce the histrograms of Figures 7.4 through 7.6. In all cases, the SNR tolerance (the proximity to the optimum SNR that determined covergence) was set to 1 dB. In other words, the algorithms "converged" when the SNR was within 1 dB of the optimum value. The results are tabulated in Table 7.1. The polar antenna plots that follow verify that all of the algorithms did an excellent job in placing nulls in the directions of the interferences.

Probably the most striking result obtained was the incredibly few iterations required by the Update Covariance algorithm to converge. This result can be misleading, however, due to the number of computations it requires per iteration. Recall that the update covariance algorithm requires about 5N complex computations for each iteration, where N is the number of antenna elements. The LMS and Constrained LMS algorithms require on the order of 2N and $5N^2$ computations per iteration respectively. Therefore, the difference in the actual number of required computations is not that great.

The Constrained LMS algorithm converged the slowest of the three algorithms. It is interesting to note that the weight changes of the LMS algorithm approach zero as the weights approach their optimal value. This can beseen from the LMS weight update equation listed below as

Figure 7.4 Convergence Histogram of LMS Algorithm in Ideal Channel

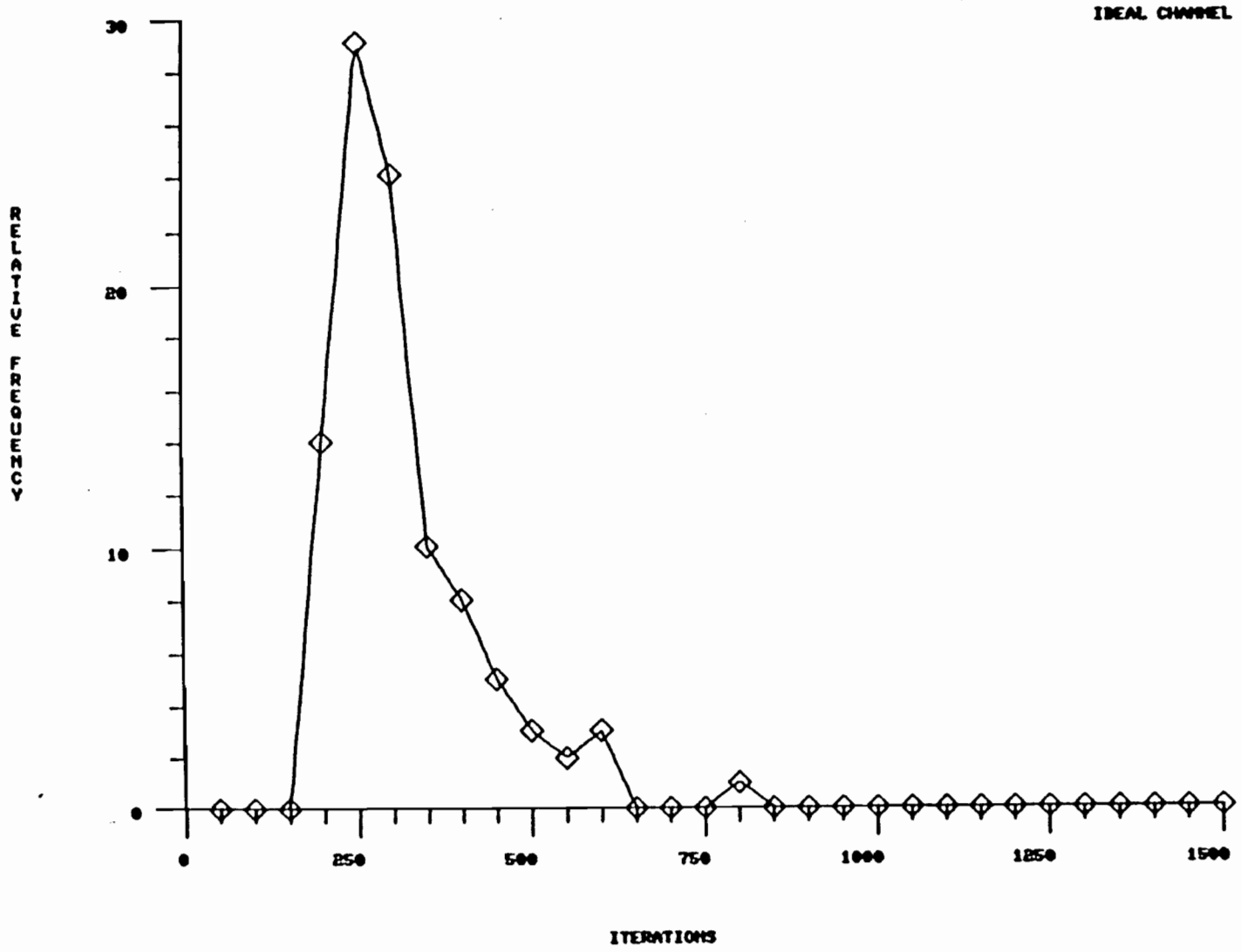


Figure 7.5 Convergence Histogram of Constrained LMS Algorithm in Ideal Channel

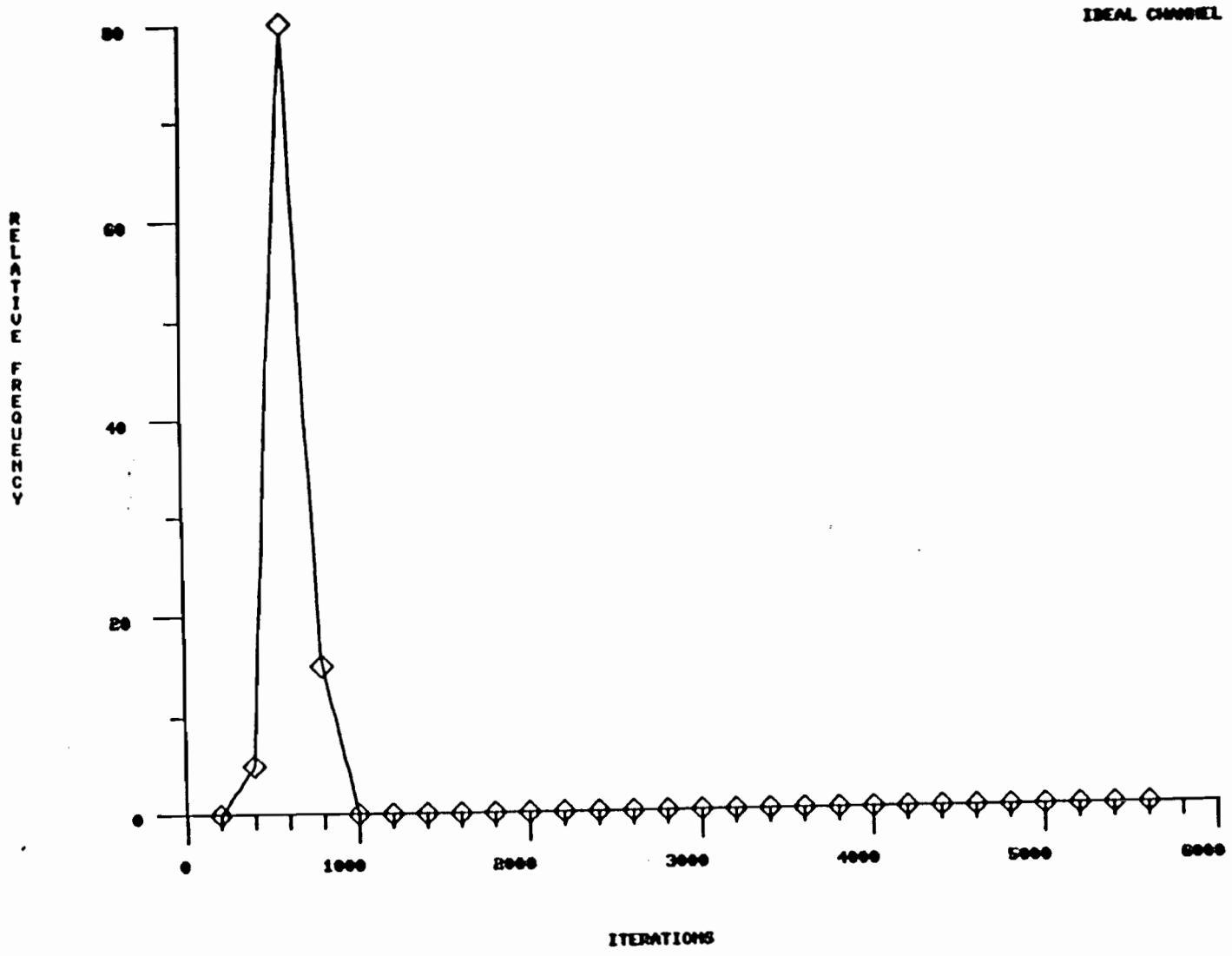


Figure 7.6 Convergence Histogram of Update Covariance Algorithm in Ideal Channel

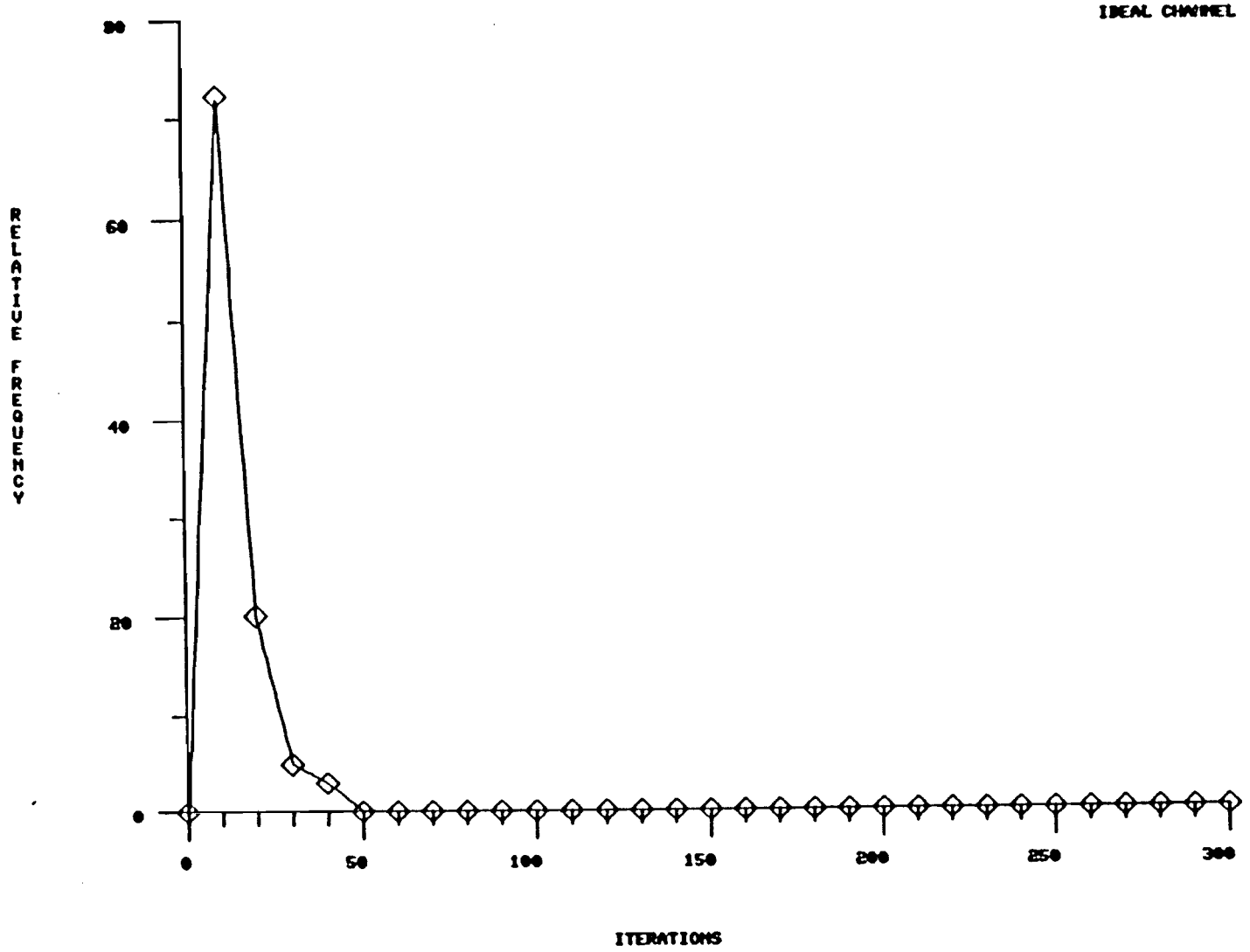


Table 7.1 TEST1 Summary

Adaptive Algorithm	Number of Convergences	Average Number of Iterations	Standard Deviation
LMS	100	318.0	108.5
Constr. LMS	100	536.5	8.14
Update Cov.	100	9.9	6.7

$$W(k+1) = W(k) + \mu\, e(k)\, X^{*}(k)$$

As the error decreases, so does the amount that the weights can change.

The weight changes of the Constrained LMS algorithm however, will never approach zero because the beam is constrained. Signal power will always appear at the output, even if contributions of interferences are negligible. Therefore, the weights will always change an appreciable amount, provided that the signal is present

$$W(k+1) = \text{Factor}*[W(k) - \mu y(k)\, X^{*}(k)]$$

It therefore seems reasonable that the Constrained LMS algorithm will perform better when the signal power is low or when it is absent altogether. This illustrates an important advantage possessed by the Constrained LMS algorithm. Unlike the others, the Constrained LMS algorithm could optimally adjust the weights before actual signal transmission (including preamble) begins. This is the major focus of TEST 4.

SUMMARY OF PLOTS FOR TEST1

Figure T1.1: Unadapted antenna plot at $\phi = 90°$, $\theta = 80°$
Purpose: To indicate initial gain in direction of Jammer 1

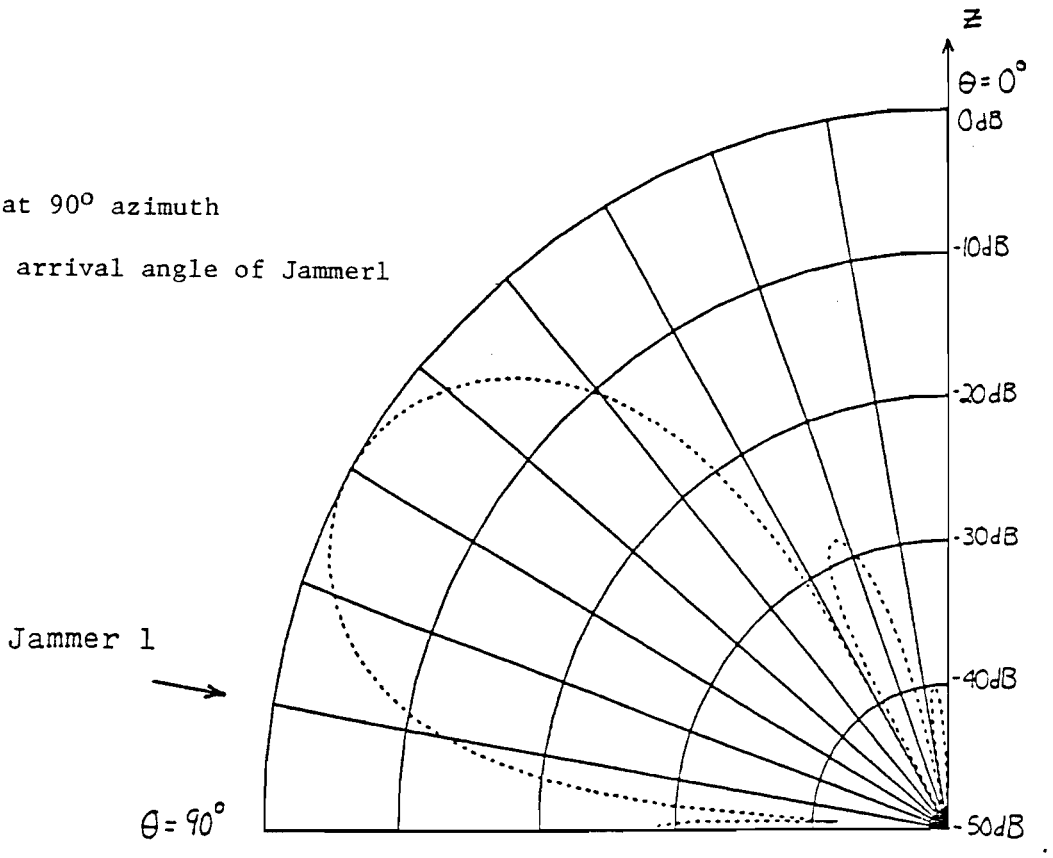
Figure T1.2: LMS-adapted antenna plot at $\phi = 90°$, $\theta = 80°$
Figure T1.3: Constrained LMS-adapted antenna plot at $\phi = 90°$, $\theta = 80°$
Figure T1.4: Update Covariance-adapted antenna plot at $\phi = 90°$, $\theta = 80°$
Purpose: To demonstrate nulling of Jammer 1 by each algorithm

Figure T1.5: Unadapted antenna plot at $\phi = 150°$, $\theta = 75°$
Purpose: To indicate initial gain in direction of Jammer 2

Figure T1.6: LMS-adapted antenna plot at $\phi = 150°$, $\theta = 75°$
Figure T1.7: Constrained LMS-adapted antenna plot at $\phi = 150°$, $\theta = 75°$
Figure T1.8: Update Covariance-adapted antenna plot at $\phi = 150°$, $\theta = 75°$
Purpose: To demonstrate nulling of Jammer 2 by each algorithm

Elevation plot at 90° azimuth

Arrow indicates arrival angle of Jammer1
(θ = 80°)



Azimuthal plot at 80° elevation

Arrow indicates arrival angle of Jammer1
(ϕ = 90°)

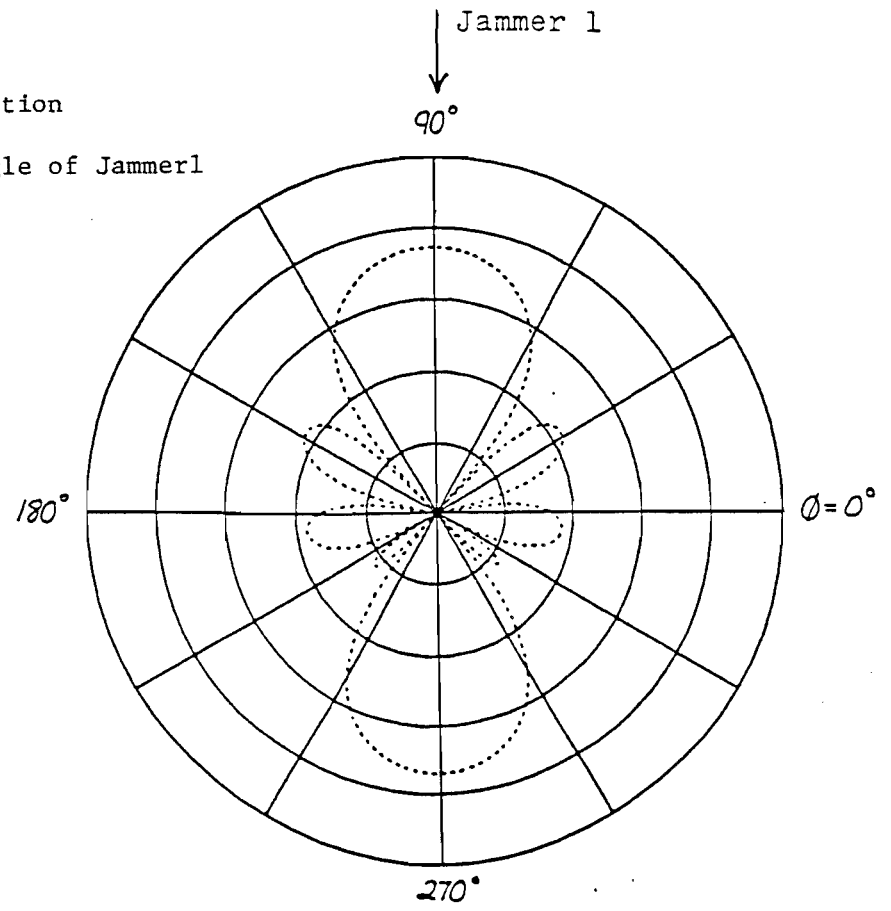


Figure T1. Upadapted Antenna Pattern in Direction of Jammer 1

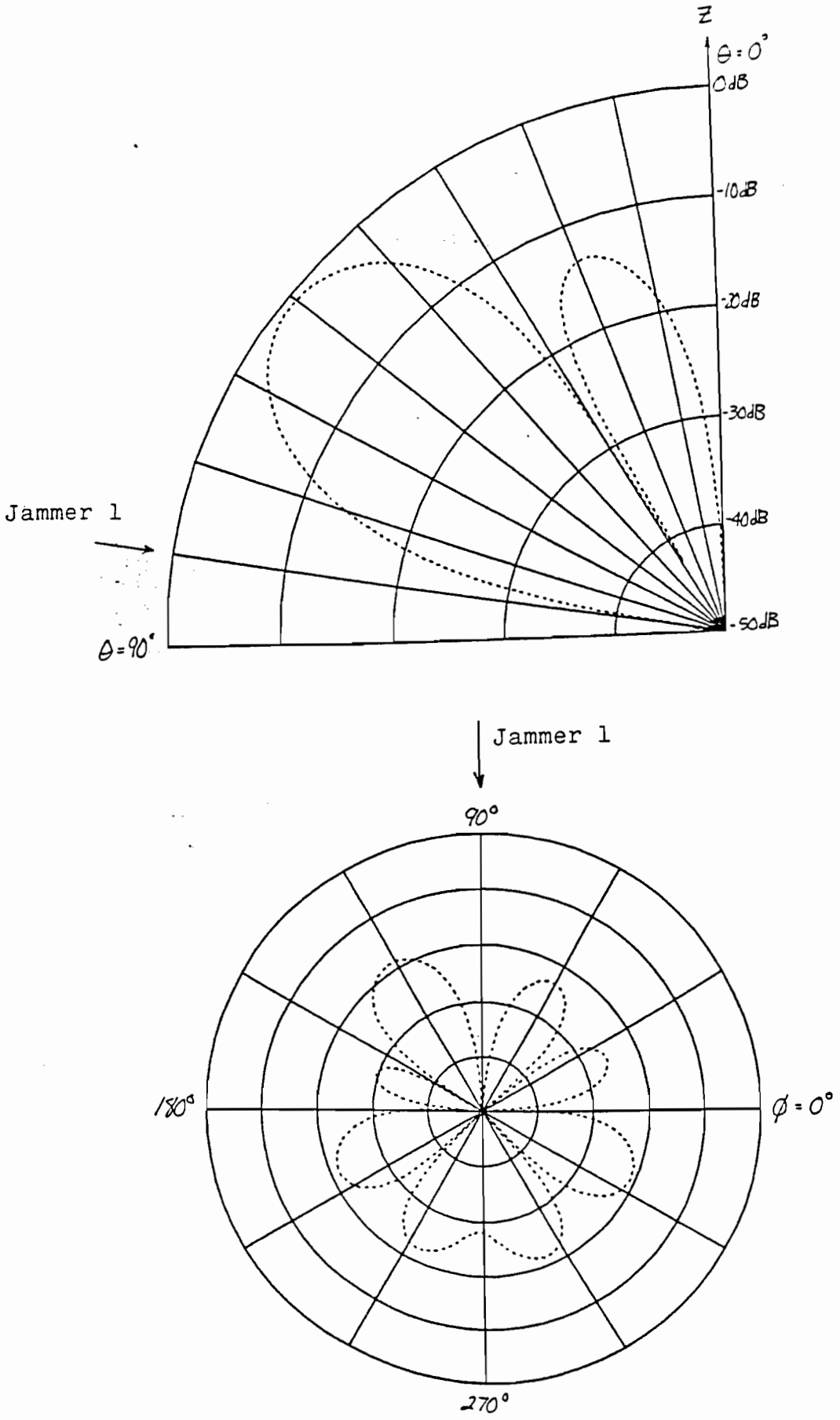


Figure T1.2 LMS-Adapted Pattern in Direction of Jammer 1

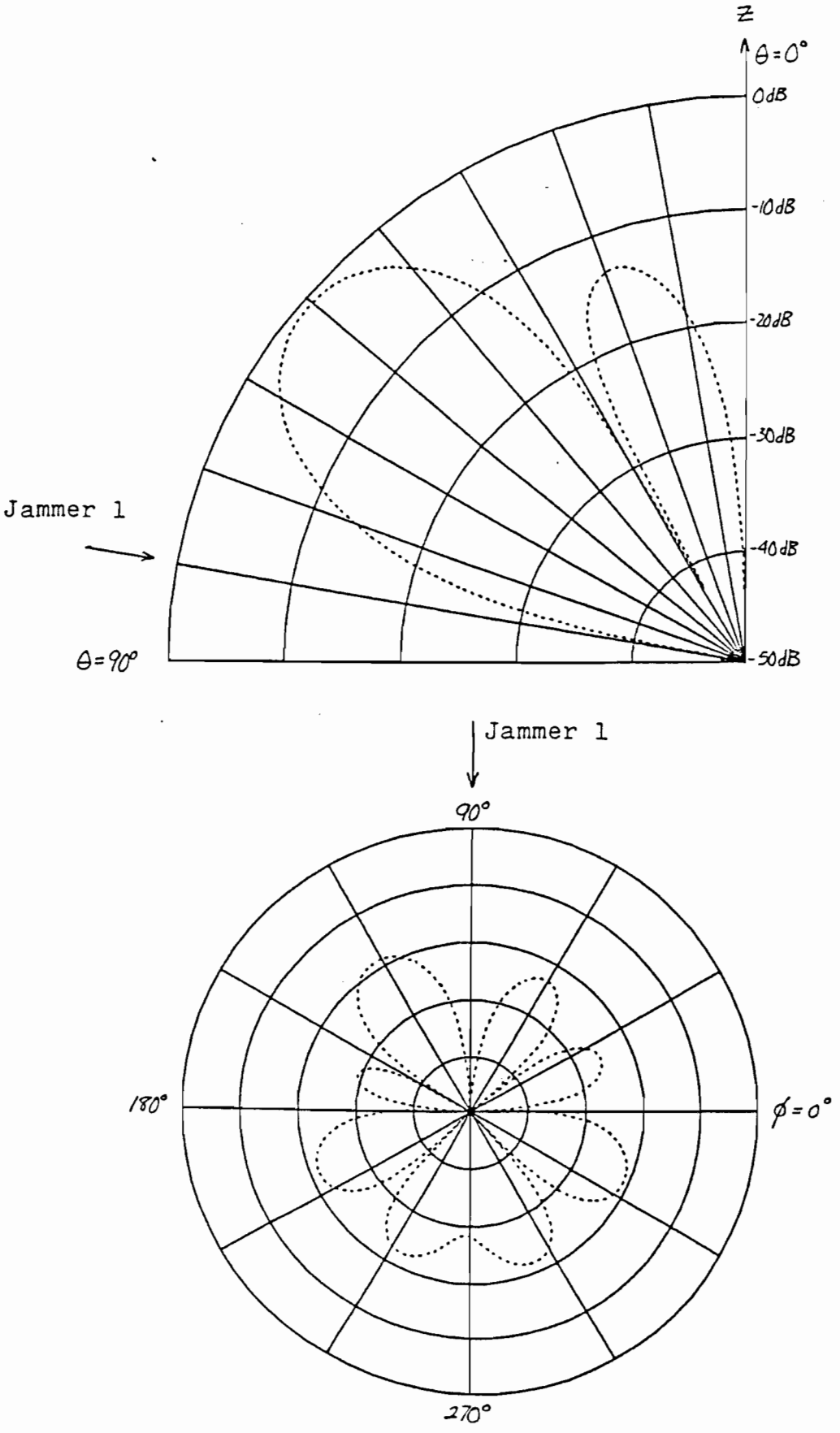


Figure T1.3 Constrained LMS-Adapted Pattern in Direction of Jammer 1

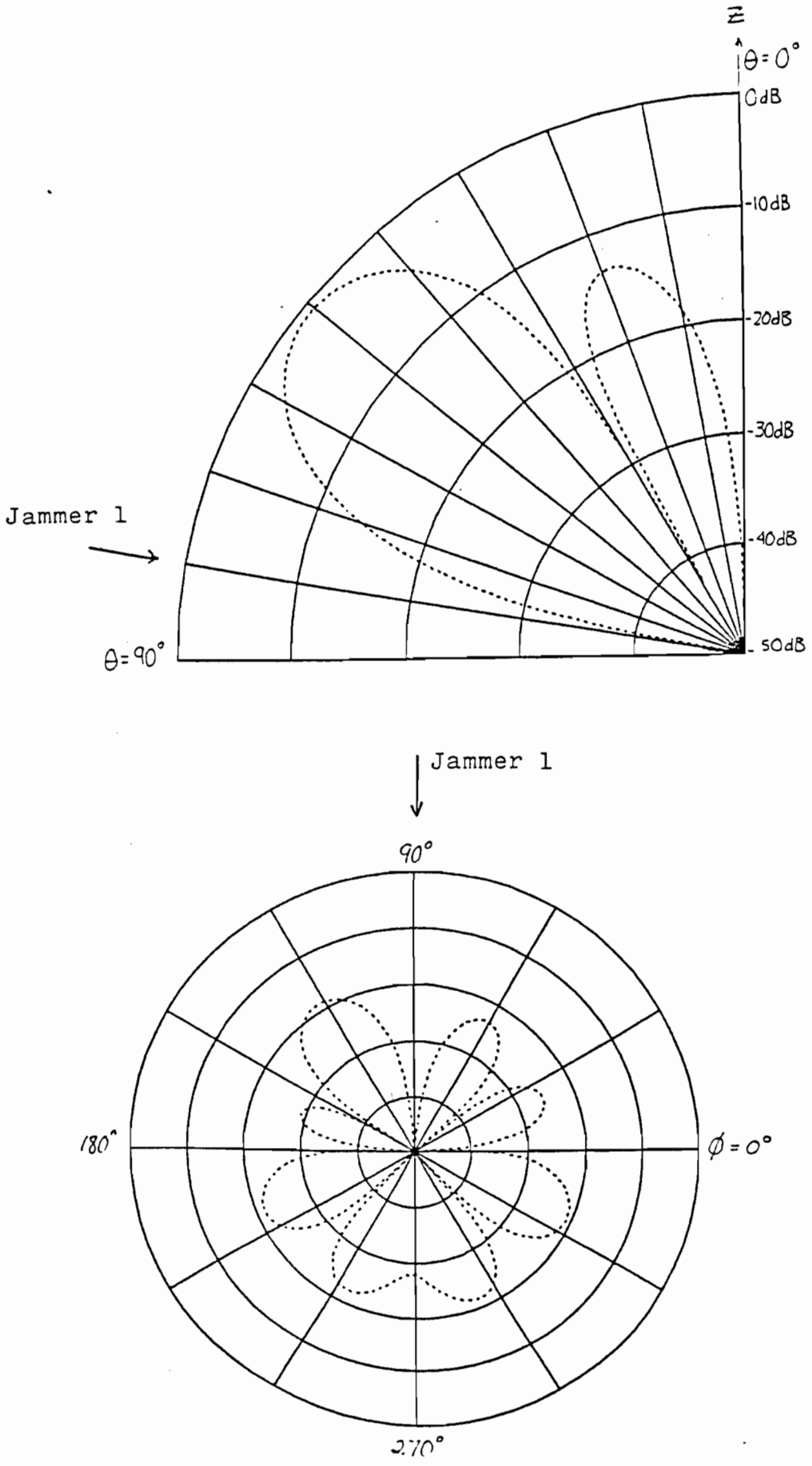
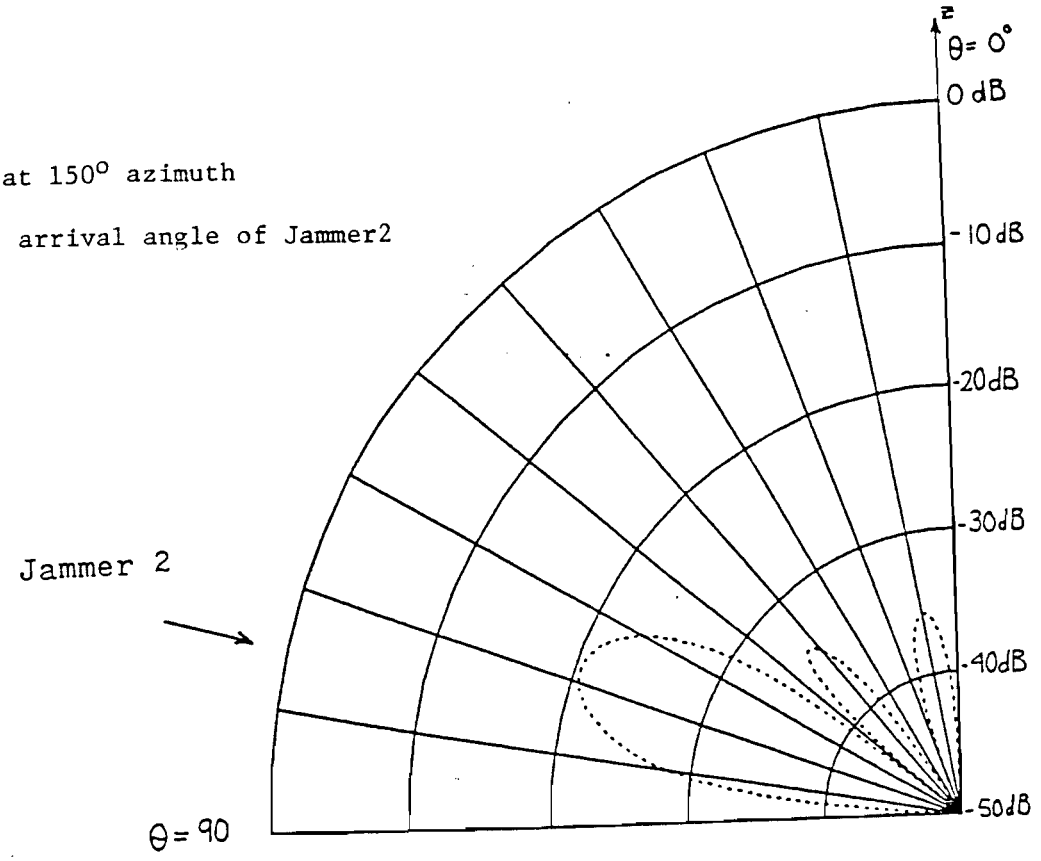


Figure T1.4 Update Covariance-Adapted Pattern in Direction of Jammer 1

Elevation Plot at 150° azimuth

Arrow indicates arrival angle of Jammer2
(θ = 75°)



Azimuthal Plot at 75° elevation

Arrow indicates arrival angle of Jammer2
(ϕ = 150°)

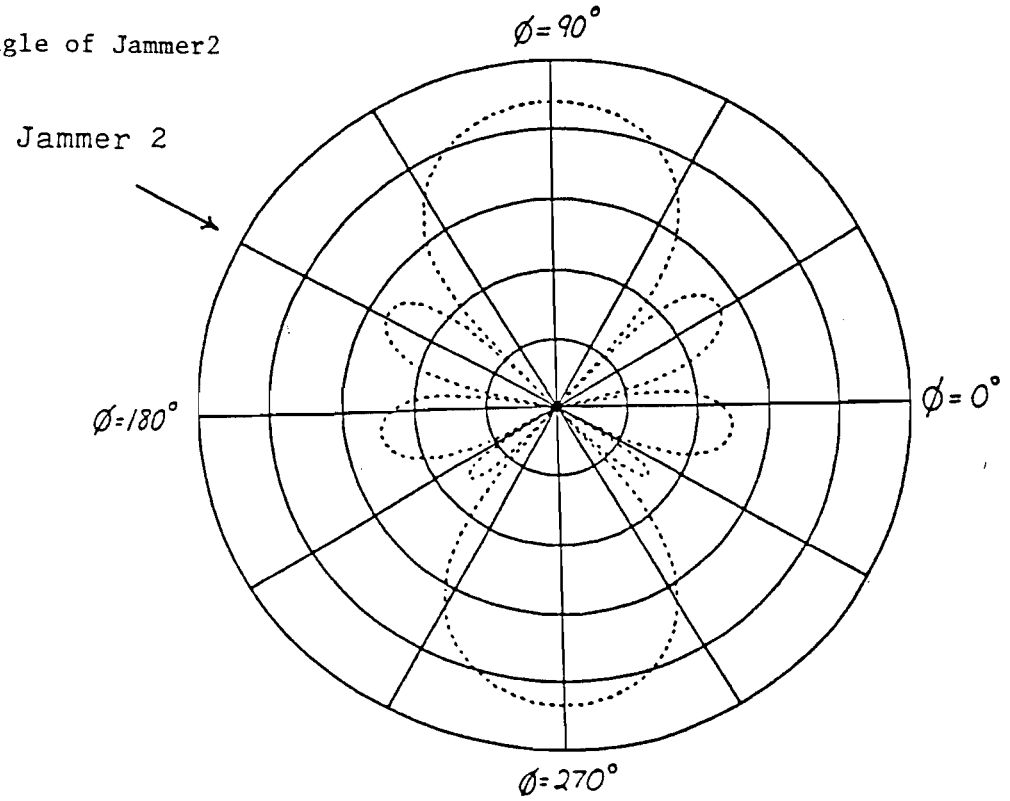


Figure T1.5 Unadapted Antenna Pattern in Direction of Jammer 2

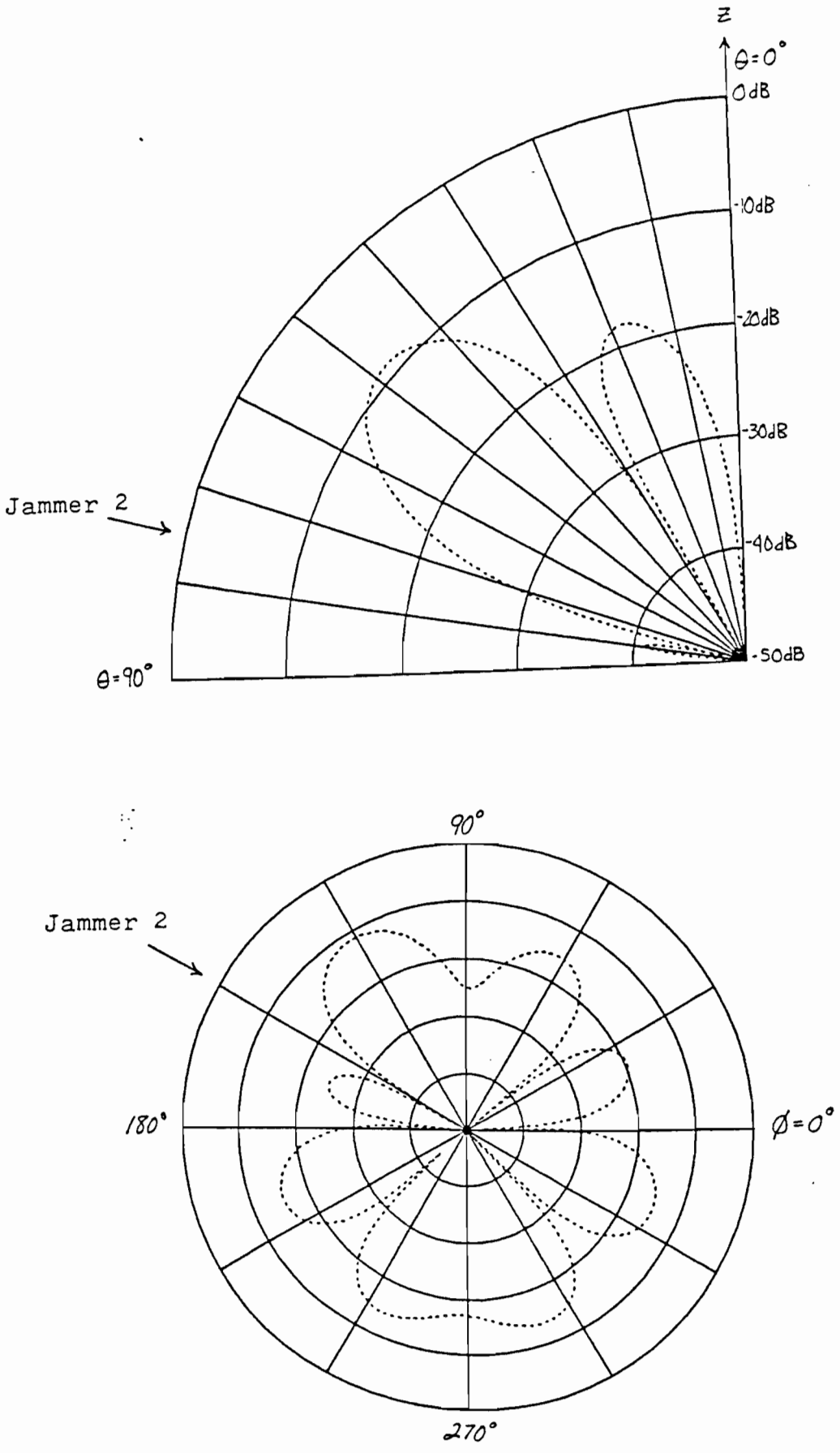


Figure T1.6 LMS-Adapted Pattern in Direction of Jammer 2

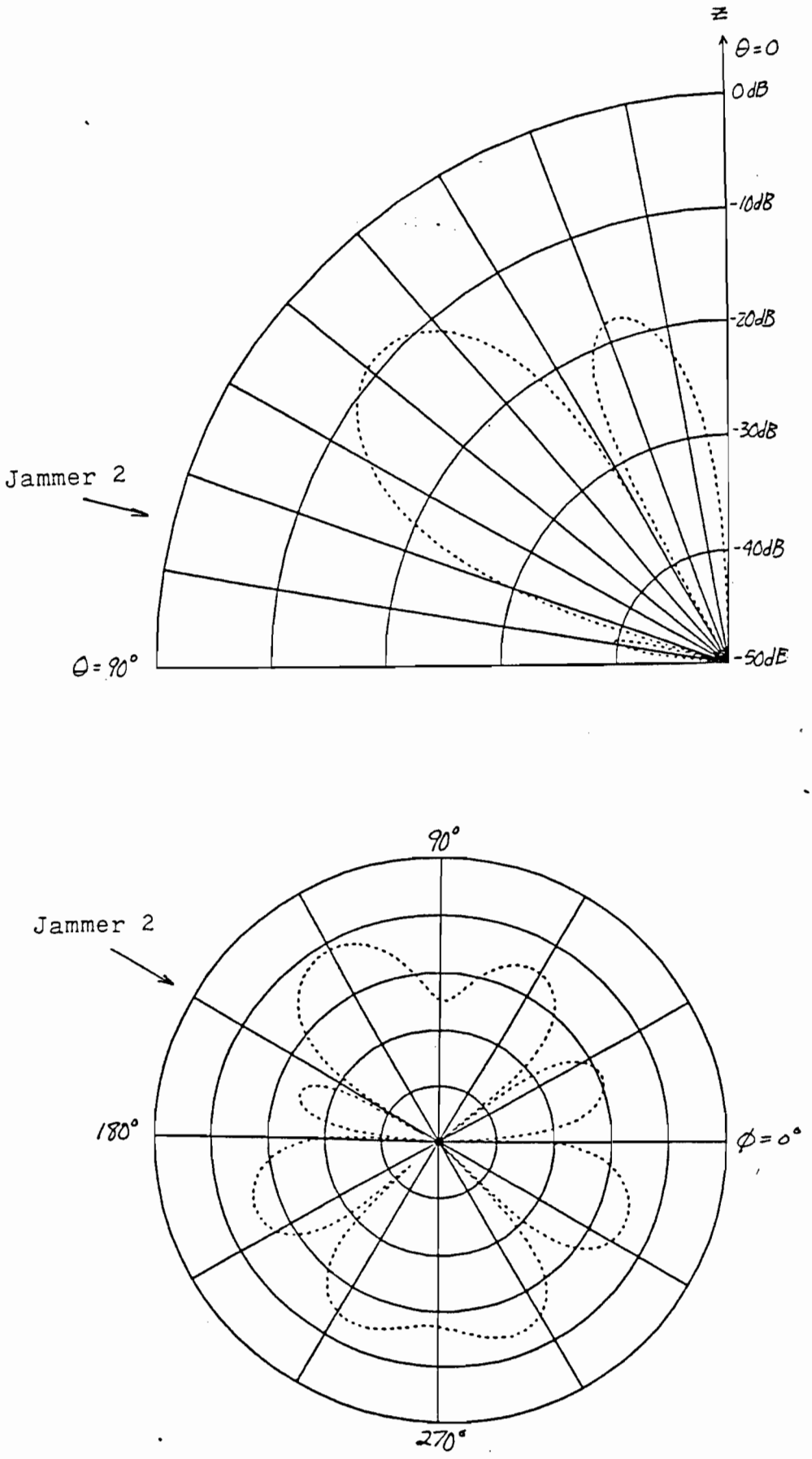


Figure T1.7 Constrained LMS-Adapted Pattern in Direction of Jammer 2

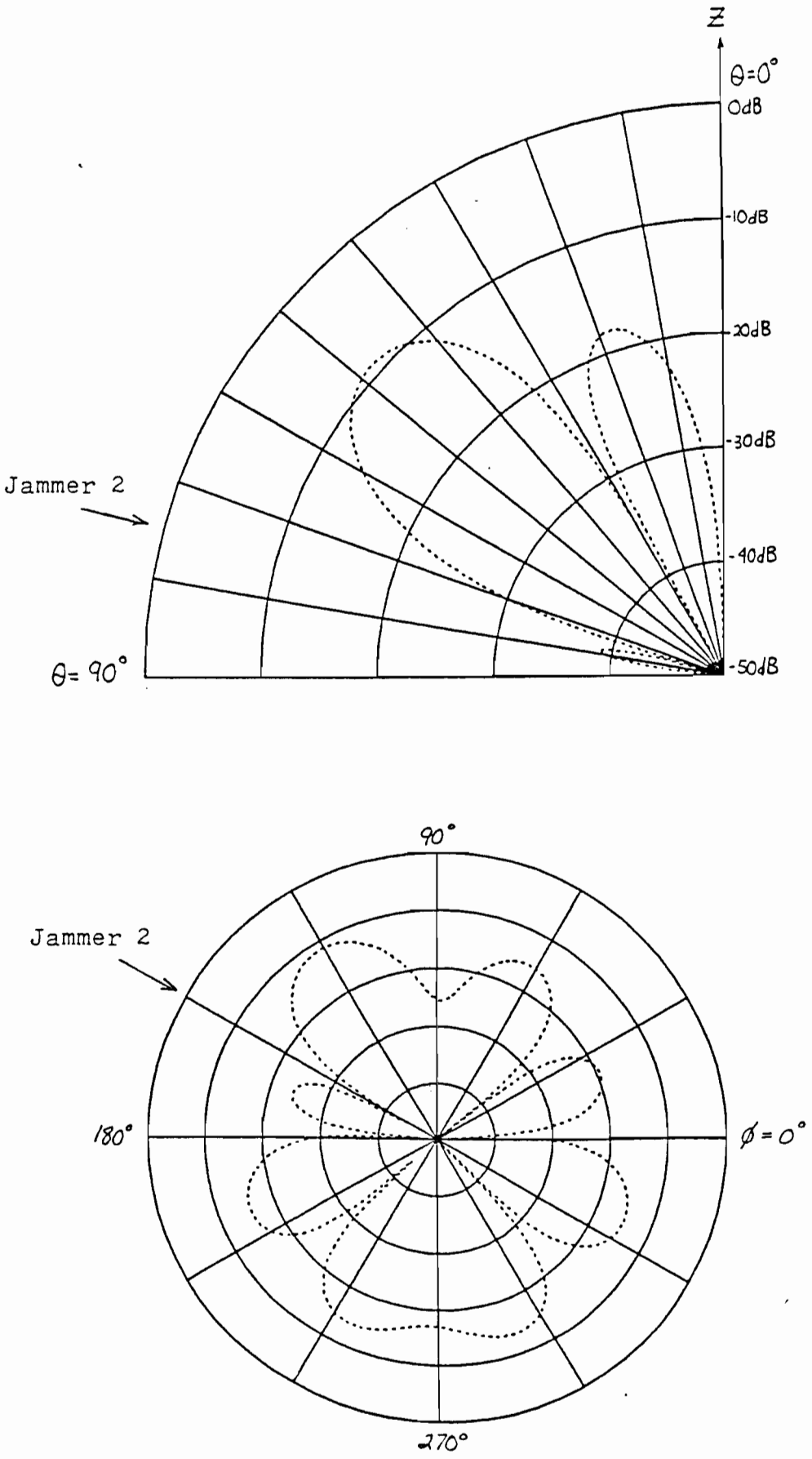


Figure T1.8 Update Covariance-Adapted Pattern in Direction of Jammer 2 ΣP

7.2 Test 2: HF Channel With Moderate Delay Charcteristics and Poor Attenuation Characteristics

The second test represents a significant departure from an ideal channel. The channel consists of three paths separated by 0.83333 msecs. Each of the paths are given equal weighting. In other words, signal components will not be attenuated more in one path than in another.

Once again, results for all algorithms were averaged over 100 convergences. Histograms depicting when the algorithms converged were generated and given as Figures 7.7 - 7.9. The results are tabulated in Table 7.2, and as the plots will verify, all algorithms did an effective job in nulling out the jammers.

The number of iterations required by the Update Covariance algorithm was very small. In this case, however, it is interesting to note the difference in actual computations of this algorithm and the LMS algorithm (9,846 for LMS, 10,651 for Update Covariance). If the computations all required the same amount of time to complete, the actual "convergence time" for the LMS would be shorter. This is why it is important to take the computational complexity of each iteration for the algorithms into account.

Once again, the Constrained LMS algorithm was the slowest in average convergence. The results also indicate that the algorithm had a wide fluctuation in the required number of iteratins. As can be seen from Figure 7.7, the Constrained LMS algorithm had difficulty in attaining the "1 dB" threshold once the SNR had approached the optimal value. This problem can quite possibly be attributed to the weight update problem previously discussed. Even after contributions of interferences were reduced from the array output, the weights will change an appreciable amount due to the signal power (which now may be magnified by the existence of three paths). Once again, this will be studied further in TEST4.

As expected, the LMS algorithm also had more difficulty approaching an optimal value in this test than in the previous one. It should also be noted that in the event that the preamble was severely distorted by a channel, covergence of any kind would be severely hampered. In other words, if the received signal did not look much at all like the reference signal, the

Figure 7.7 Convergence Histogram of LMS Algorithm in Channel 2

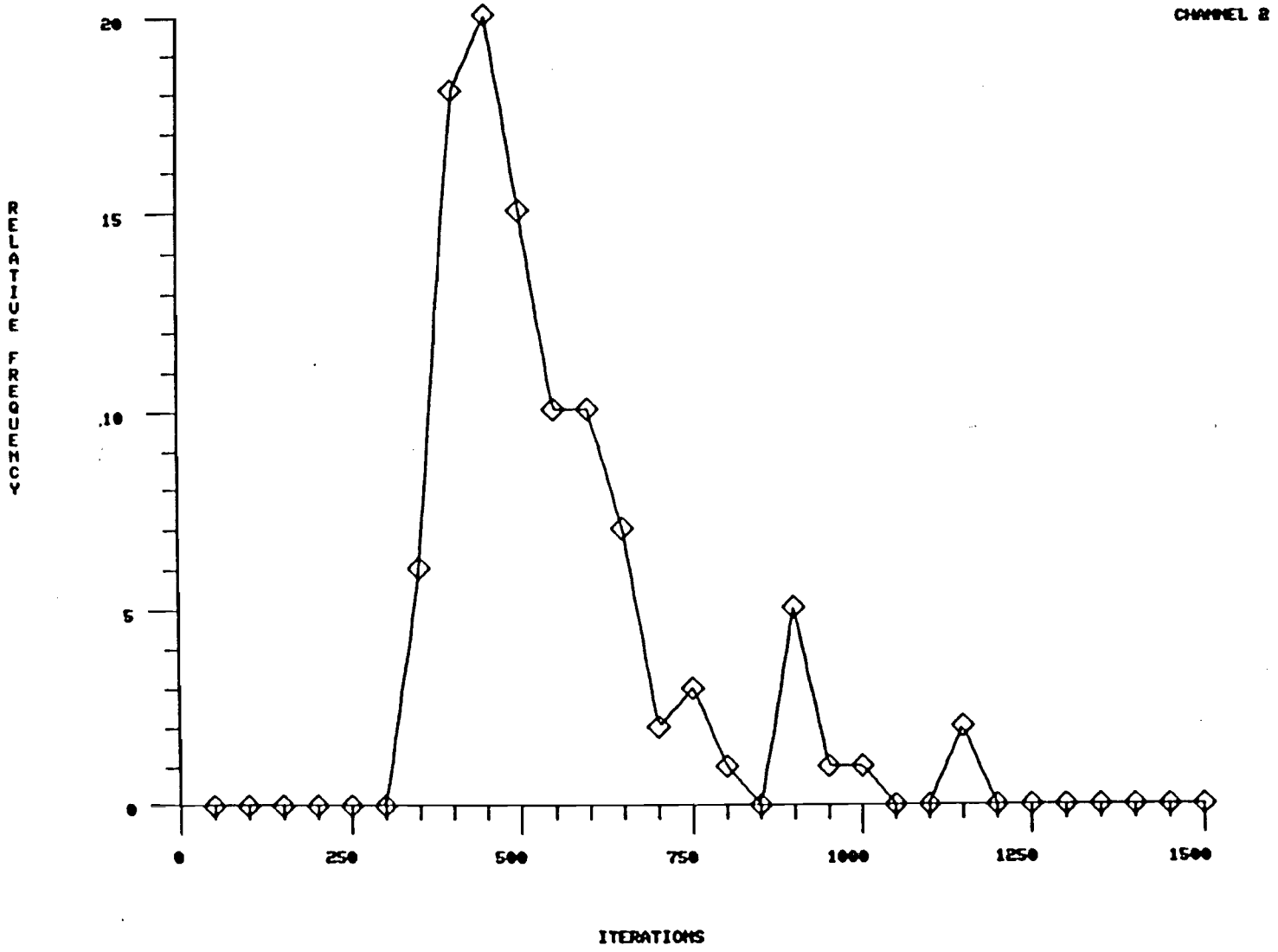


Figure 7.8 Convergence Histogram of Constrained LMS algorithm in Channel 2

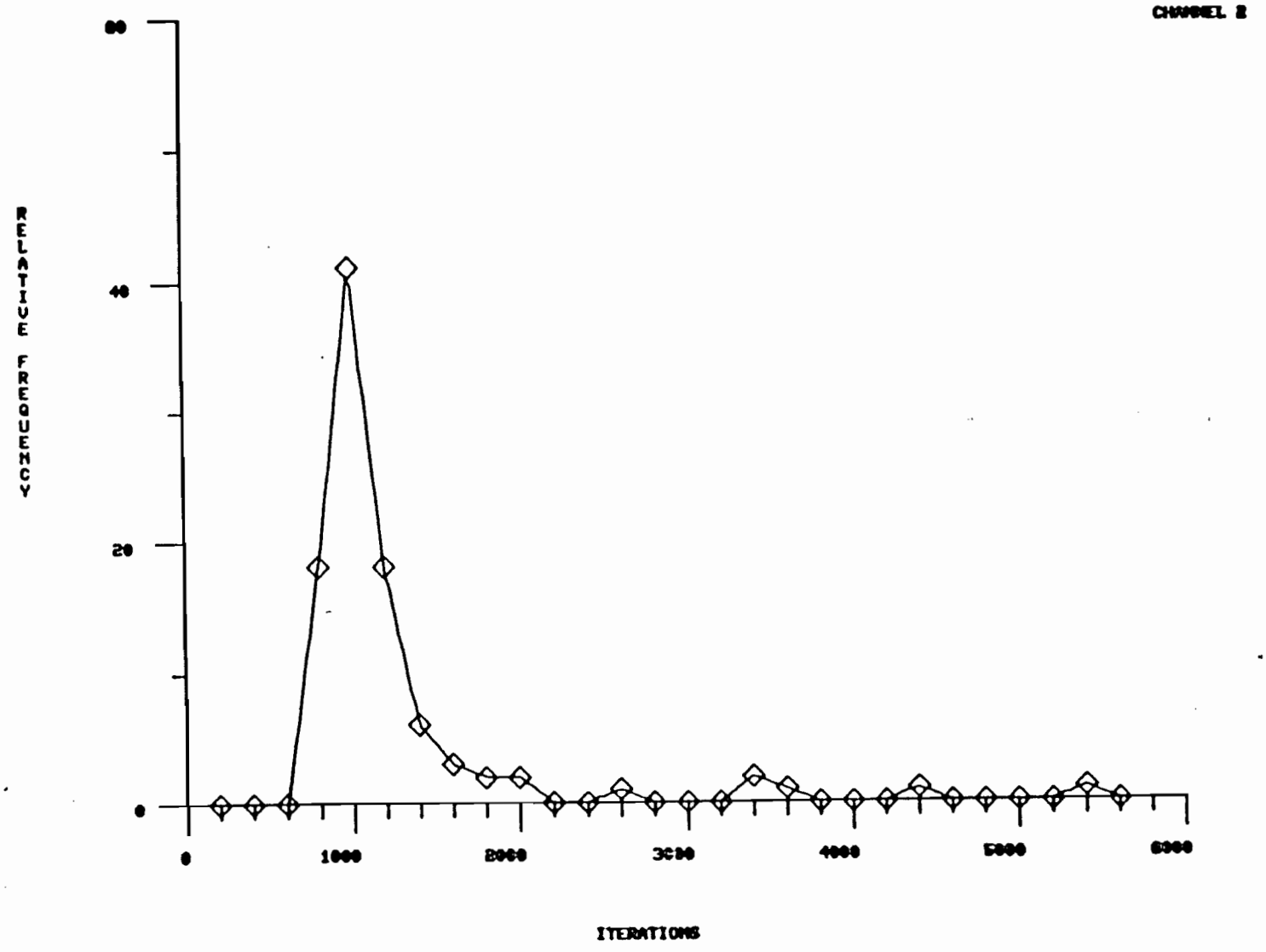


Figure 7.9 Convergence Histogram of Update Covariance Algorithm in Channel 2

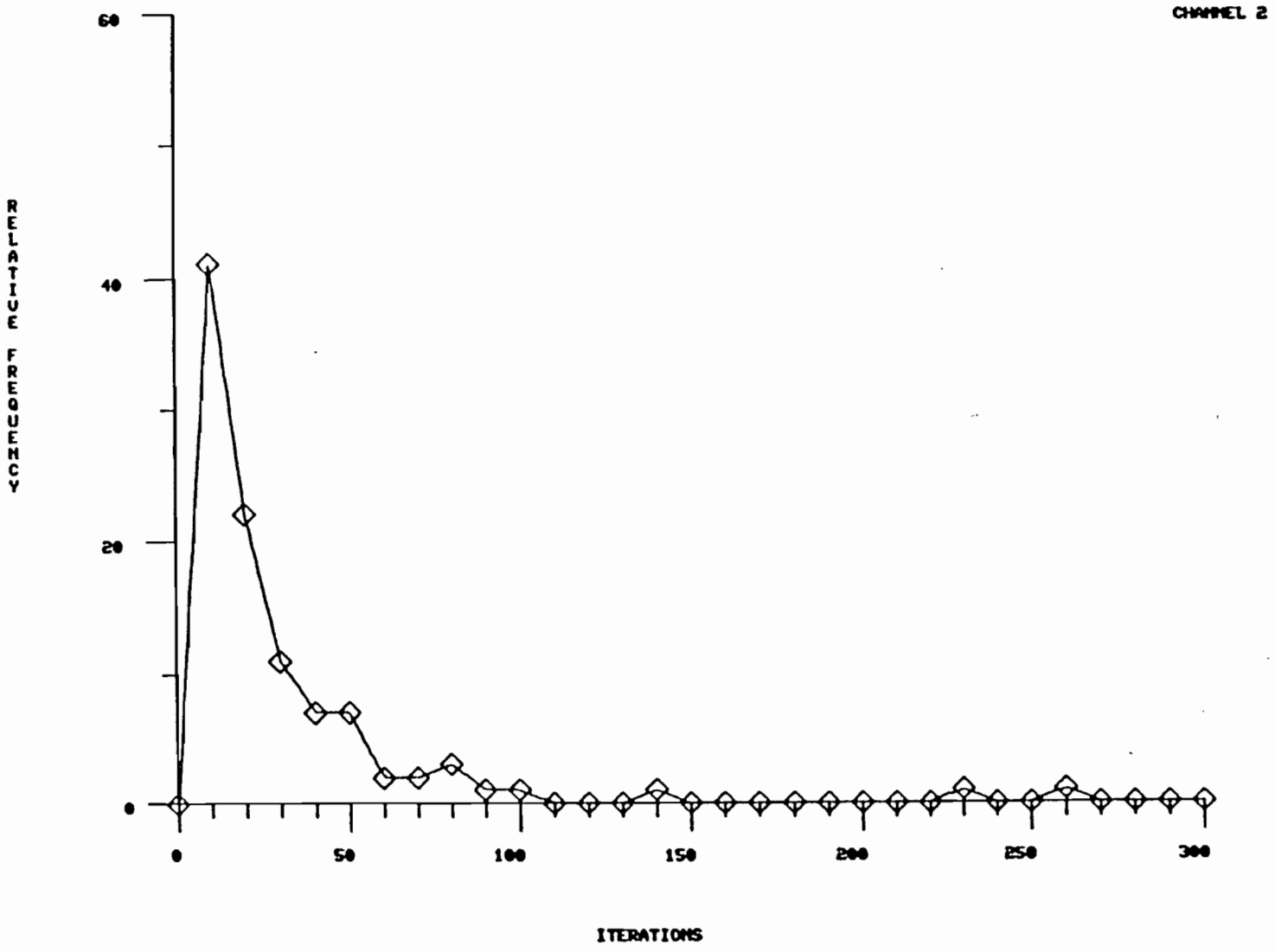


TABLE 7.2 TEST2 SUMMARY

ADAPTIVE ALGORITHM	NUMBER OF CONVERGENCES	AVERAGE NUMBER OF ITERATIONS	STANDARD DEVIATION
LMS	100	547.0	170.8
CONSTR. LMS	100	1155.0	699.2
UPDATE COV.	100	26.3	38.3

algorithm would have an extremely difficult task. After all, algorithms requiring references operate by forcing the array output to be equal to the reference signal. If this is attained, it is assumed that contributions of jammers are negligible. Consider the case, however, when the received signal does not strongly resemble the reference signal. Even if the weights were optimally adjusted, the output of the array would not look like the reference signal, and the algorithm would continue to adjust the weights in an attempt to force the output to be equal to the reference signal. In effect, it would be trying to compensate for the channel effects, although it has no true means of doing so. This is an inherent problem with algorithms that require references and is worth mentioning. It is also worth mentioning that the LMS algorithm had no problem convergning in this case, and the channel produced some fairly bad smearing effects. Therefore, it can be assumed that the channel must get significantly worse than this to prevent convergence.

SUMMARY OF PLOTS FOR TEST2

Figure T2.1: Unadapted antenna plot at $\phi = 90°$, $\theta = 80°$
Purpose: To indicate initial gain in direction of Jammer 1

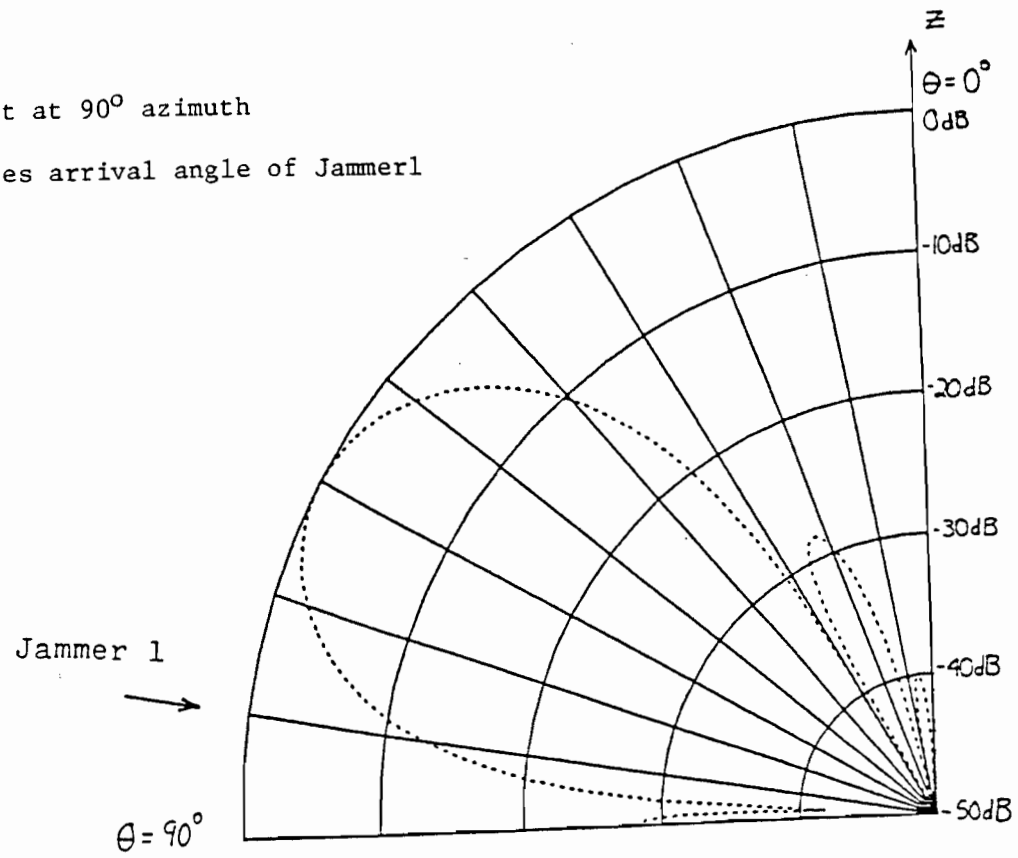
Figure T2.2: LMS-adapted antenna plot at $\phi = 90°$, $\theta = 80°$
Figure T2.3: Constrained LMS-adapted antenna plot at $\phi = 90°$, $\theta = 80°$
Figure T2.4: Update Covariance-adapted antenna plot at $\phi = 90°$, $\theta = 80°$
Purpose: To demonstrate nulling of Jammer 1 by each algorithm

Figure T2.5: Unadapted antenna plot at $\phi = 150°$, $\theta = 75°$
Purpose: To indicate initial gain in direction of Jammer 2

Figure T2.6: LMS-adapted antenna plot at $\phi = 150°$, $\theta = 75°$
Figure T2.7: Constrained LMS-adapted antenna plot at $\phi = 150°$, $\theta = 75°$
Figure T2.8: Update Covariance-adapted antenna plot at $\phi = 150°$, $\theta = 75°$
Purpose: To demonstrate nulling of Jammer 2 by each algorithm

Elevation plot at 90° azimuth

Arrow indicates arrival angle of Jammer1
(θ = 80°)



Azimuthal plot at 80° elevation

Arrow indicates arrival angle of Jammer1
(ϕ = 90°)

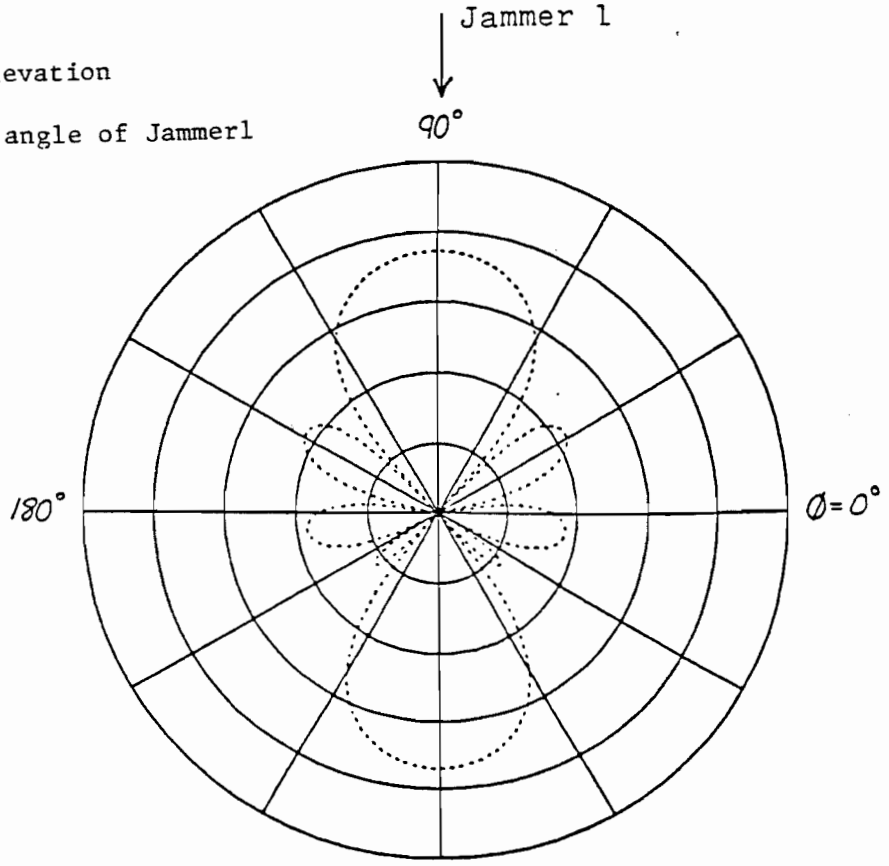


Figure T2.1 Unadapted Antenna Pattern in Direction of Jammer 1

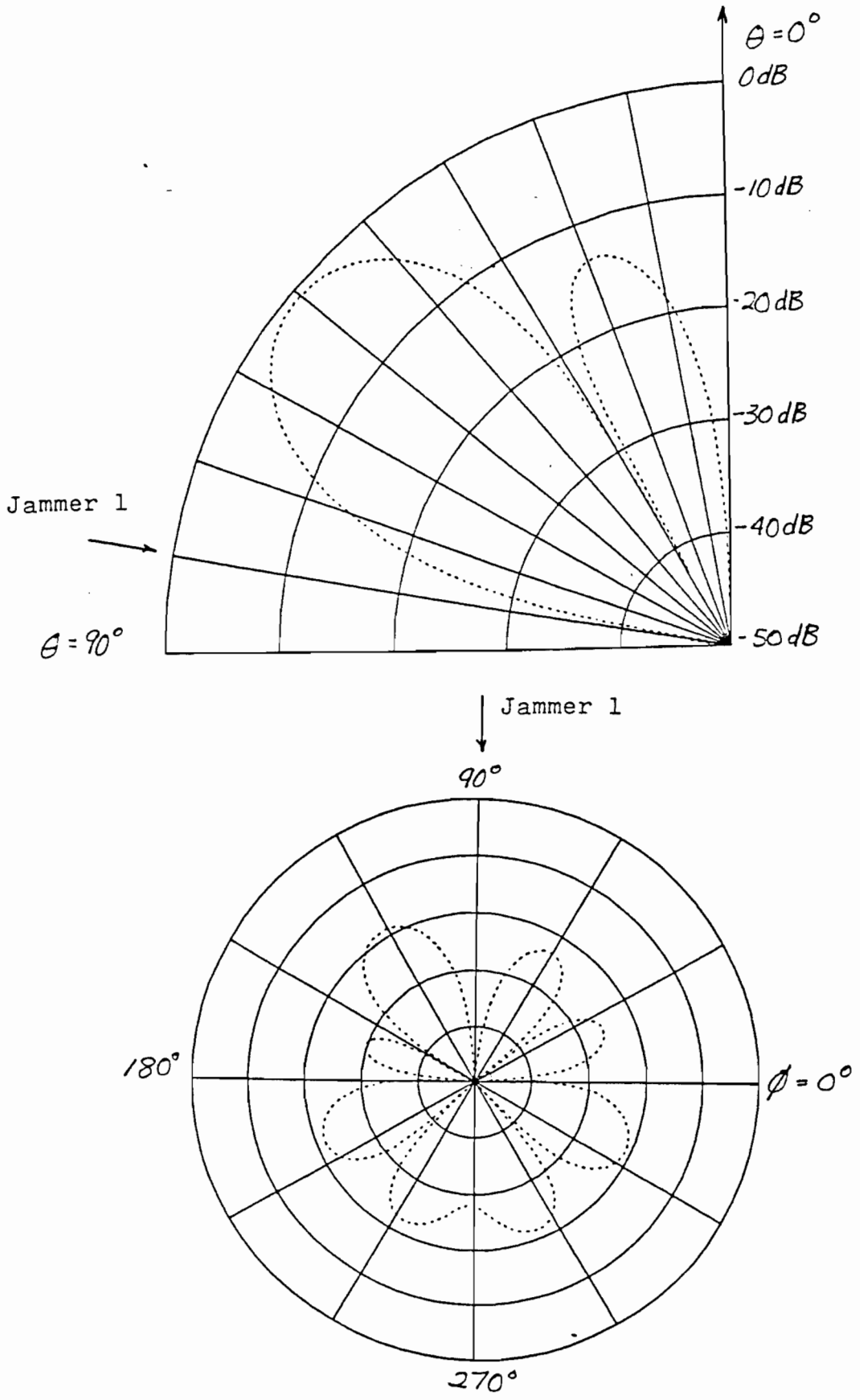


Figure T2.2 LMS-Adapted Pattern in Direction of Jammer 1

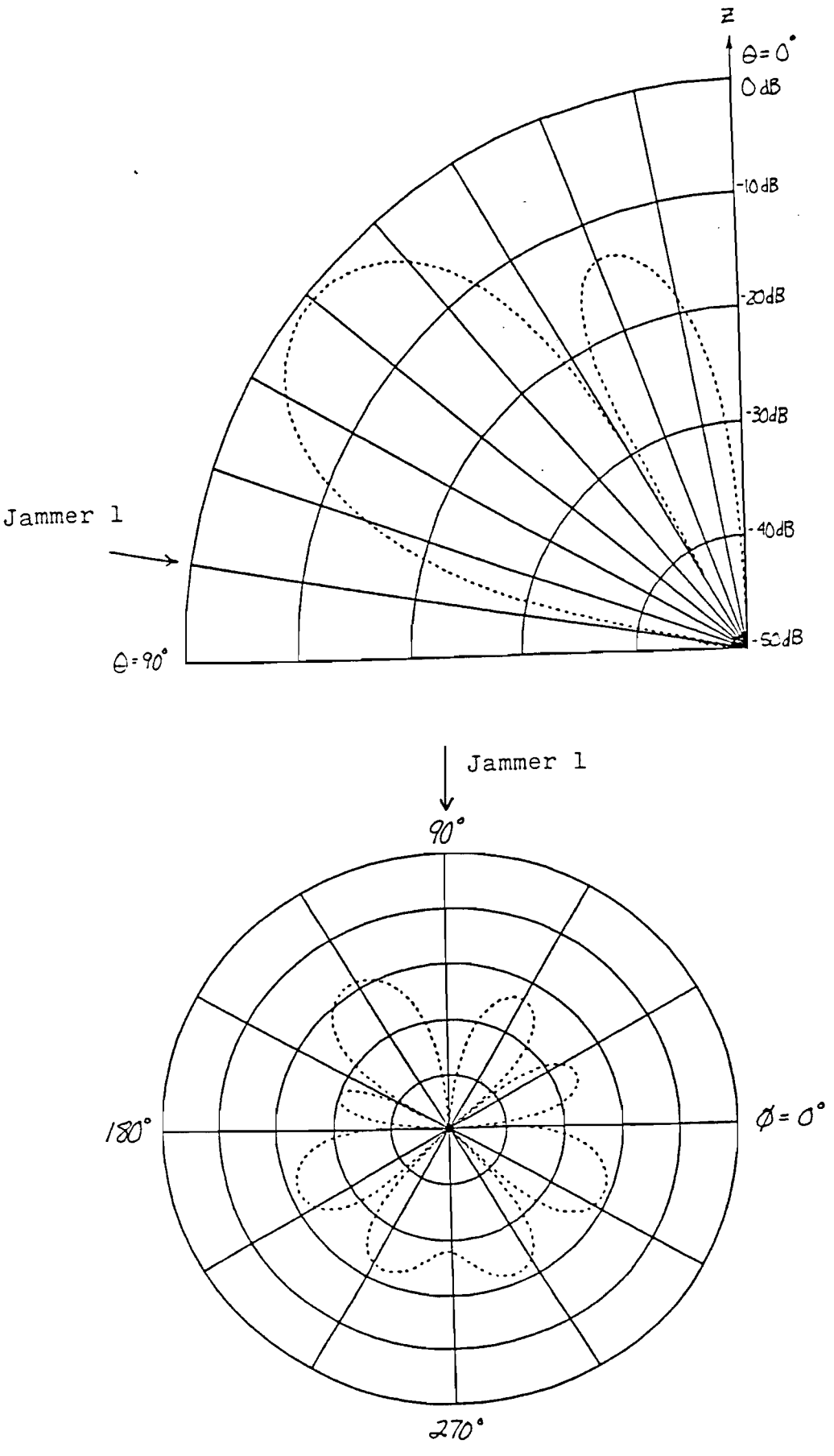


Figure T2.3 Constrained LMS-Adapted Pattern in Direction of Jammer 1

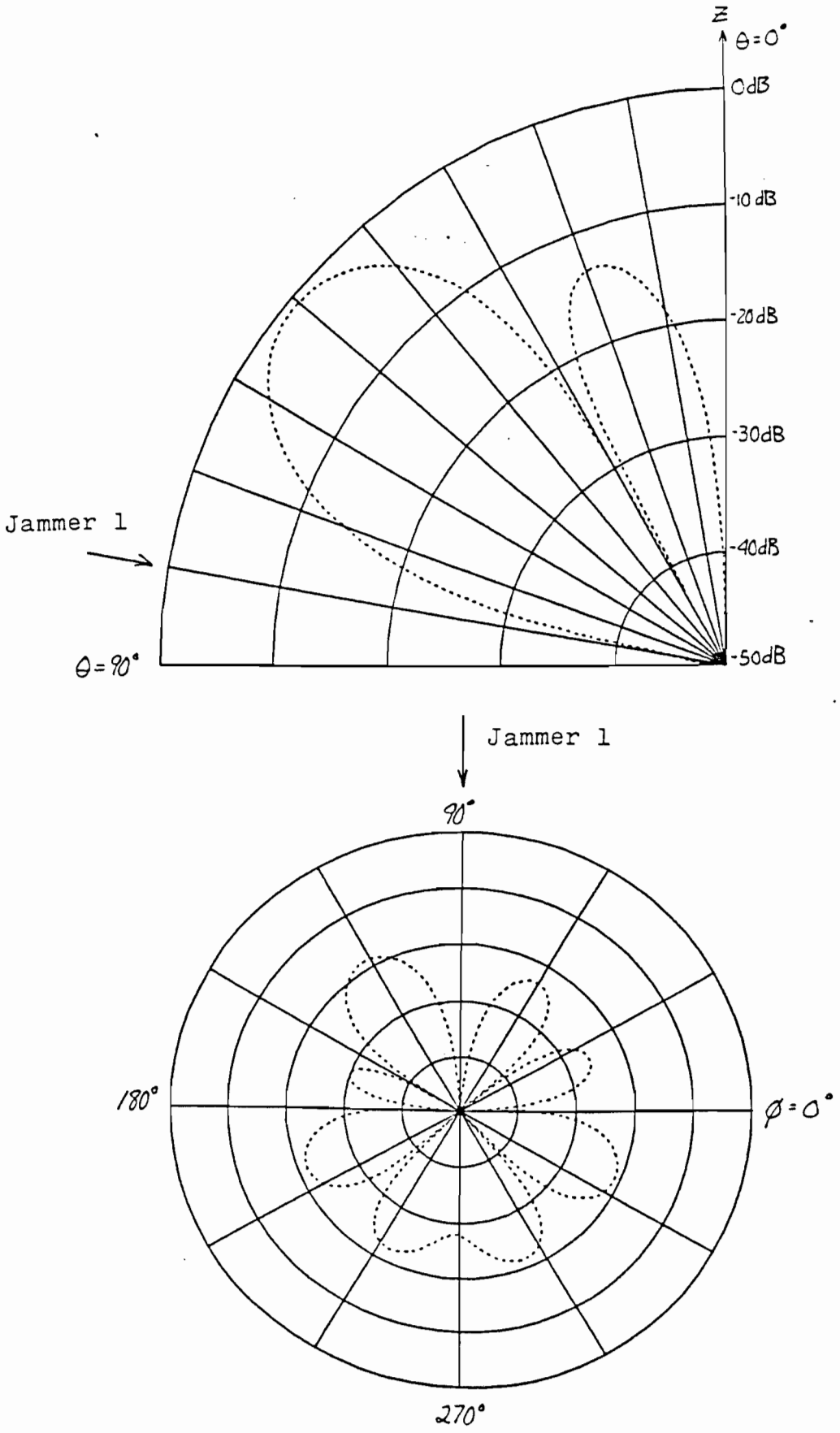
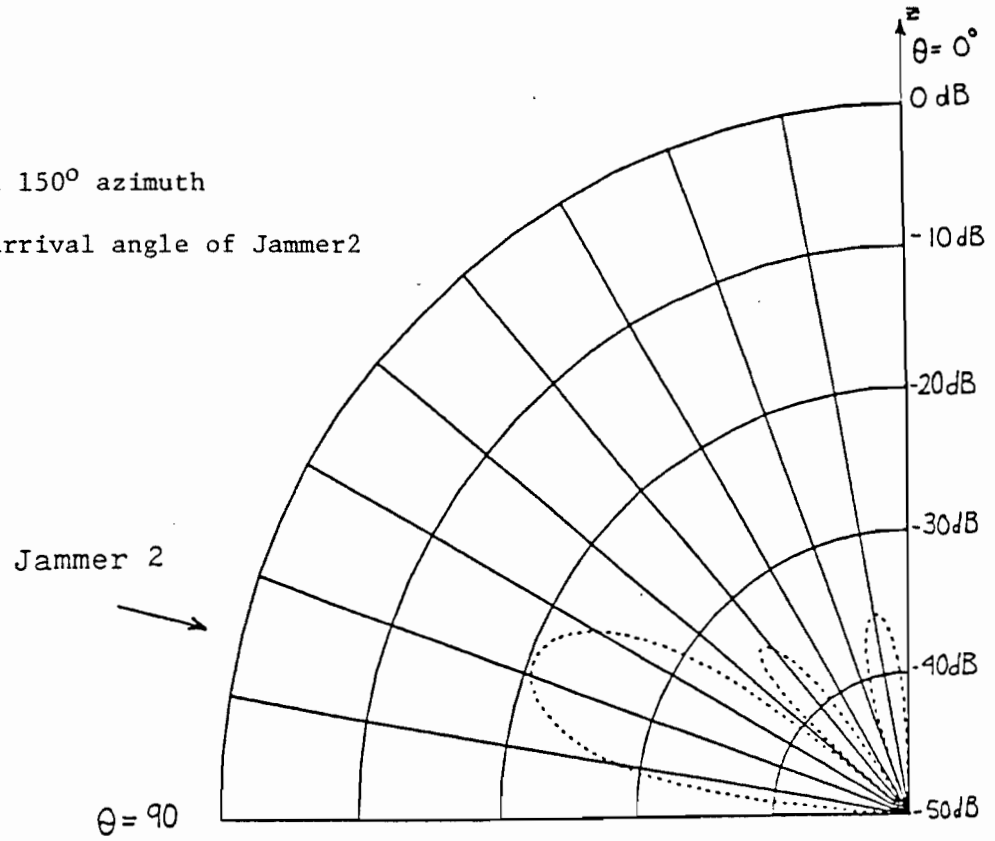


Figure T2.4 Update Covariance-Adapted Pattern in Direction of Jammer 1

Elevation plot at 150° azimuth

Arrow indicates arrival angle of Jammer2
(θ = 75°)



Azimuthal plot at 75° elevation

Arrow indicates arrival angle of Jammer2
(ϕ = 150°)

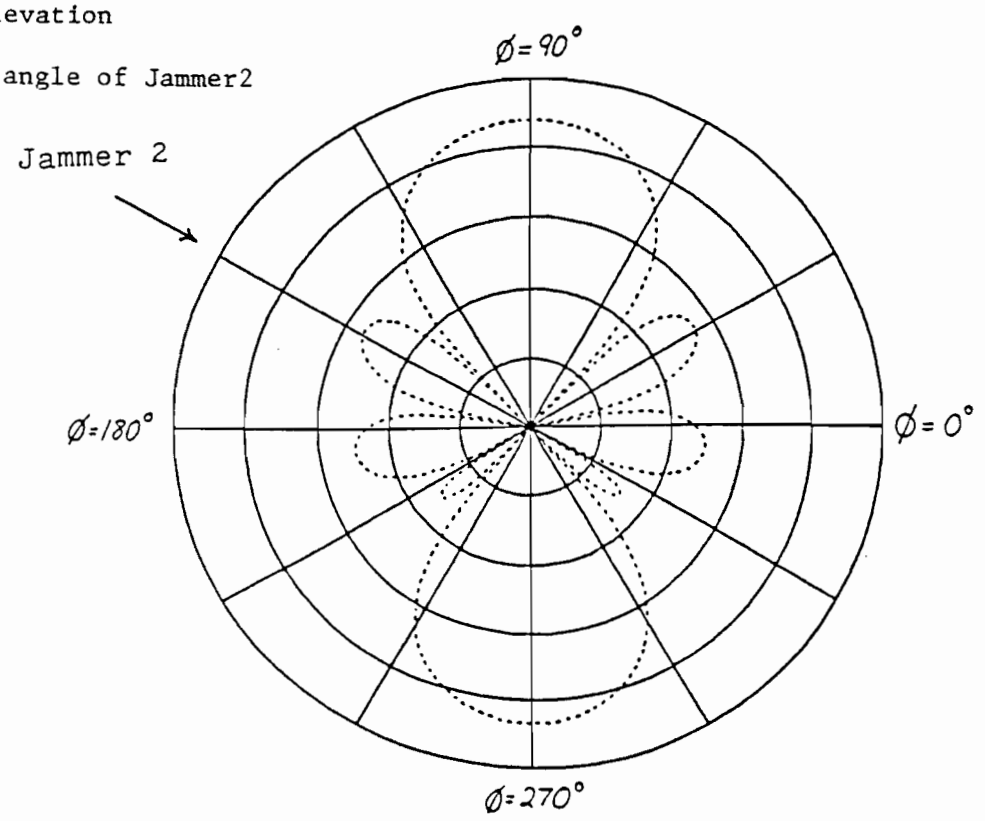


Figure T2.5 Unadapted Antenna Pattern in Direction of Jammer 2

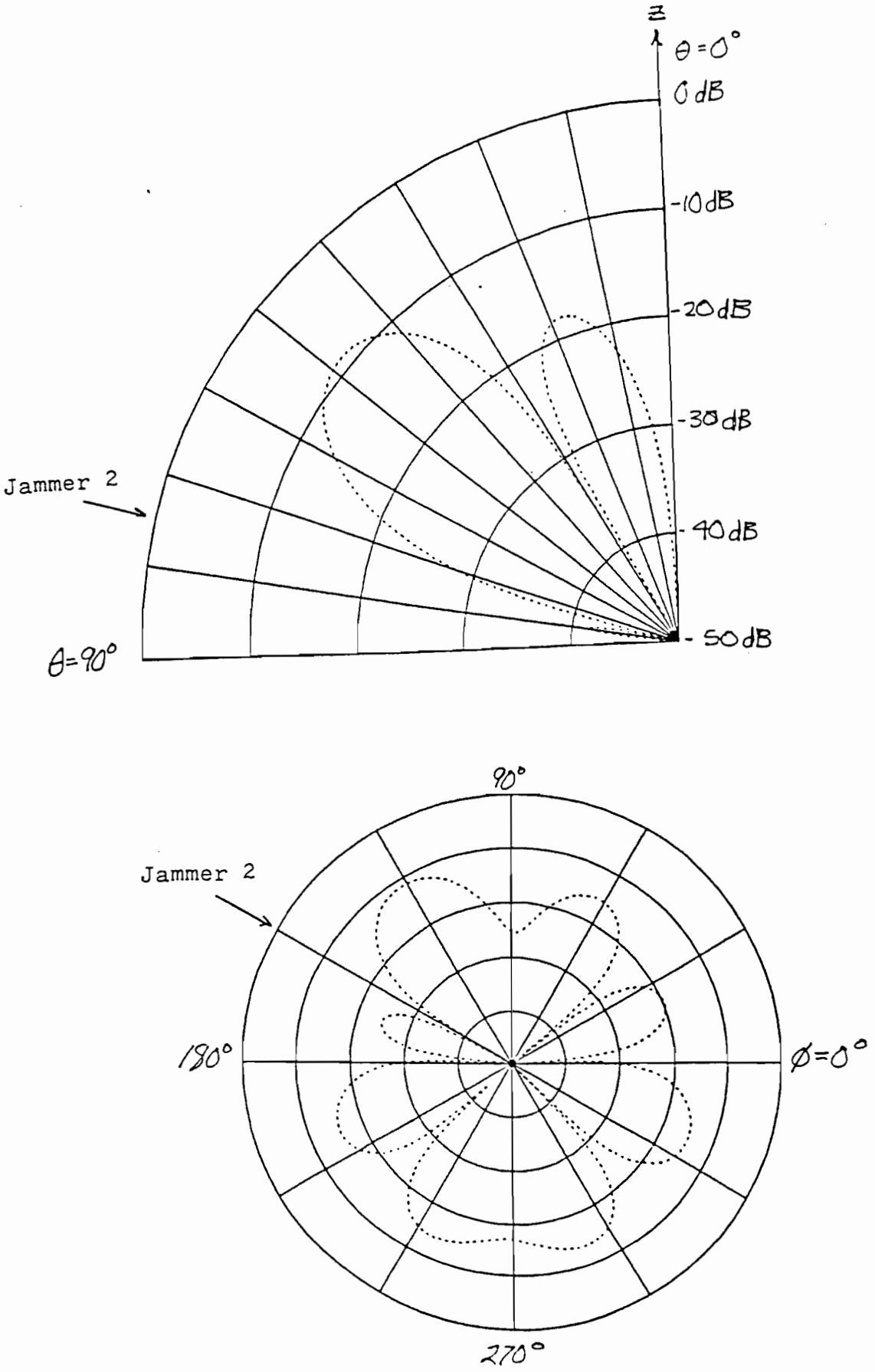


Figure T2.6 LMS-Adapted Pattern in Direction of Jammer 2

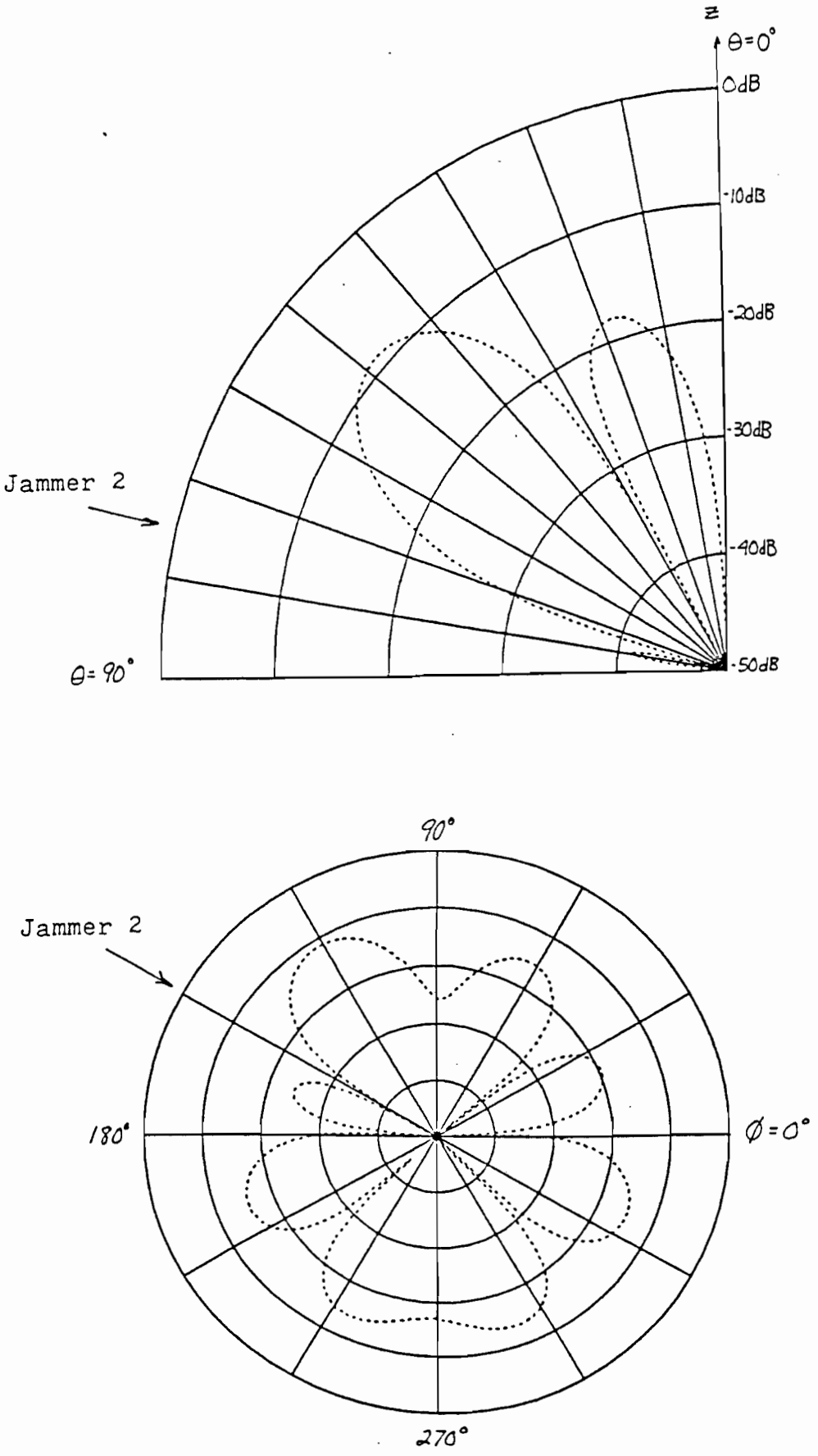


Figure T2.7 Constrained LMS-Adapted Pattern in Direction of Jammer 2

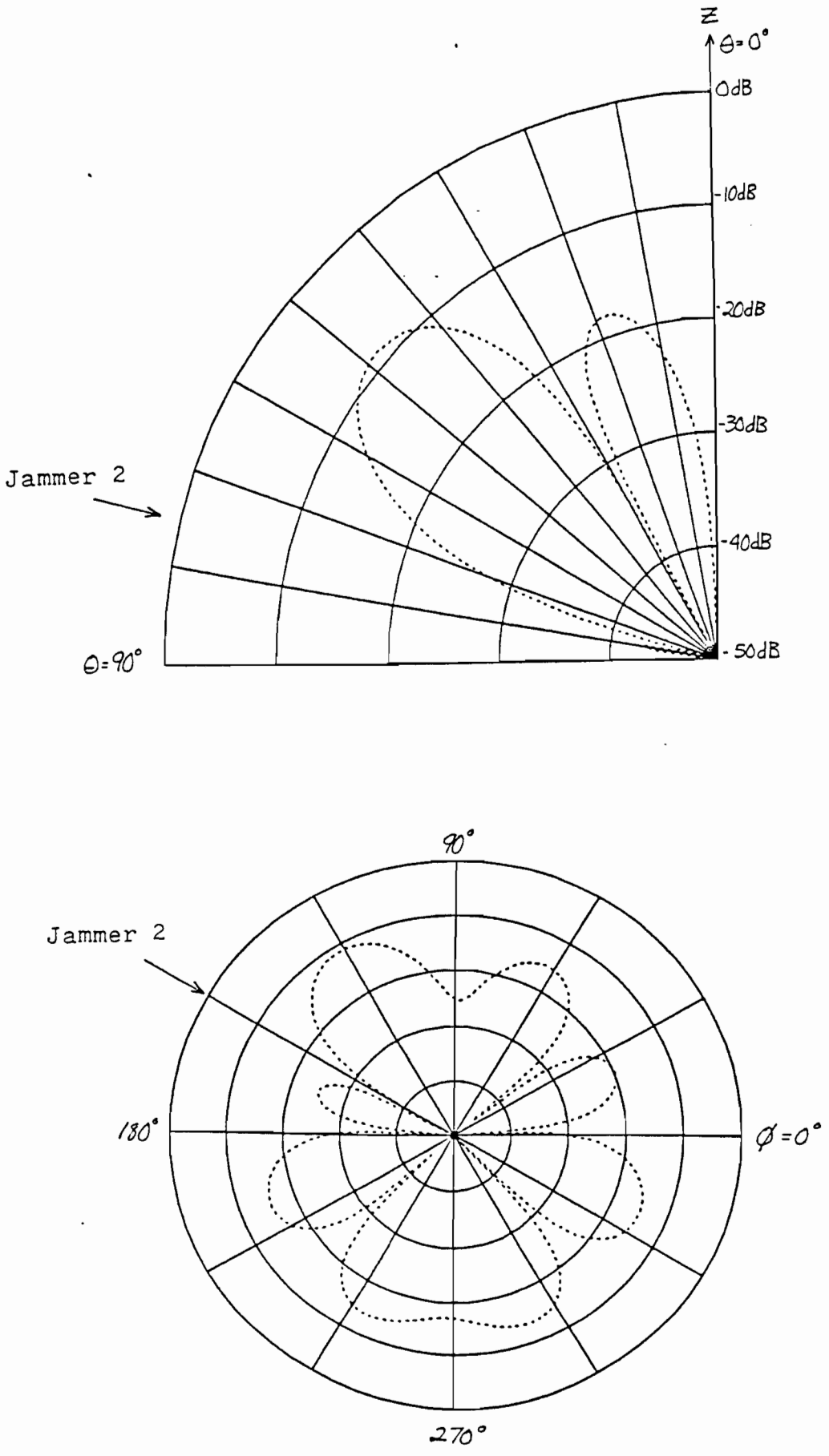


Figure T2.8 Update Covariance-Adapted Pattern in Direction of Jammer 2

7.3 TEST3: HF Channel with Poor Delay Characteristics and Poor Attenuation Characteristics

This test differs from TEST2 in that the delay between signals arriving from different paths is now doubled to 1.6666 msecs.

Once again, results for all algorithms were averaged over 100 convergences. The convergences were monitored and histograms were generated which indicate the nature of when the algorithms converged. The SNR tolerance was set to 1 dB for all trials. All of the algorithms placed deep nulls in the directions of the jamming signals, as the plots that follow will verify. The results are shown below in Table 7.3.

The Update Covariance algorithm again required the fewest number of iterations by a wide margin. It is also important to note that the LMS algorithm actually required fewer computations on the average to converge (9,765 for LMS, 11,664 for Update Covariance). Therefore, in real time, the LMS algorithm converged faster, making the assumption that all computations take the same amount of time.

The Constrained LMS algorithm was the "loser" again. But the average number of convergence iterations did not change significantly from TEST2. This seems to indicate that the covergence difficulties are more related to the magnification of the signal (existence of three paths) than the delay between the paths.

The results indicate that the delay between the paths did not seriously affect the performance of the LMS algorithm. Again, this fact suggests that the existence of three distinct paths plays a more significant role in affecting algorithm performance. The LMS algorithm still converged in a modest number of iterations on the average.

Figure 7.10 Convergence Histogram of LMS Algorithm in Channel 3

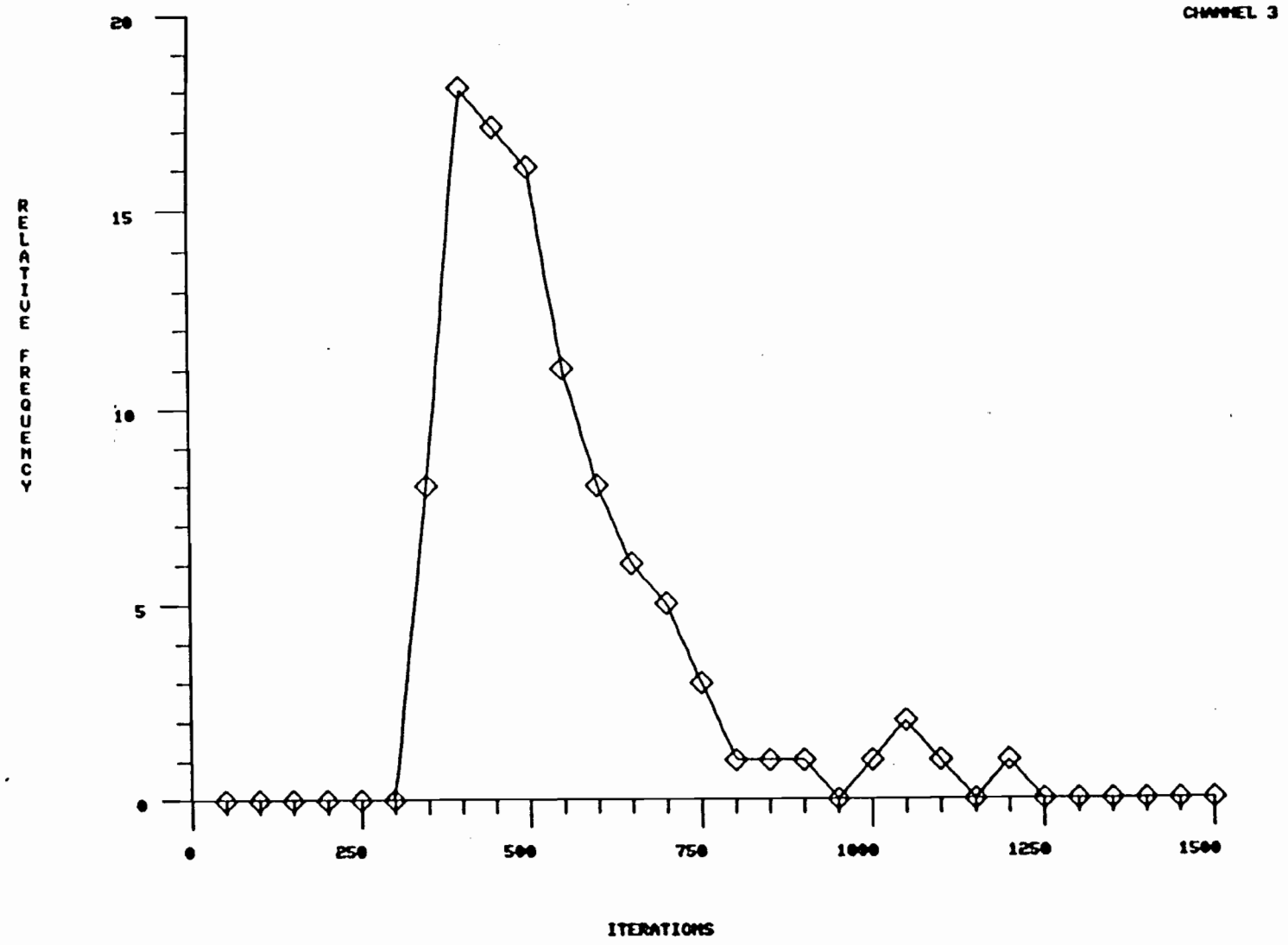


Figure 7.11 Convergence Histogram of Constrained LMS Algorithm in Channel 3

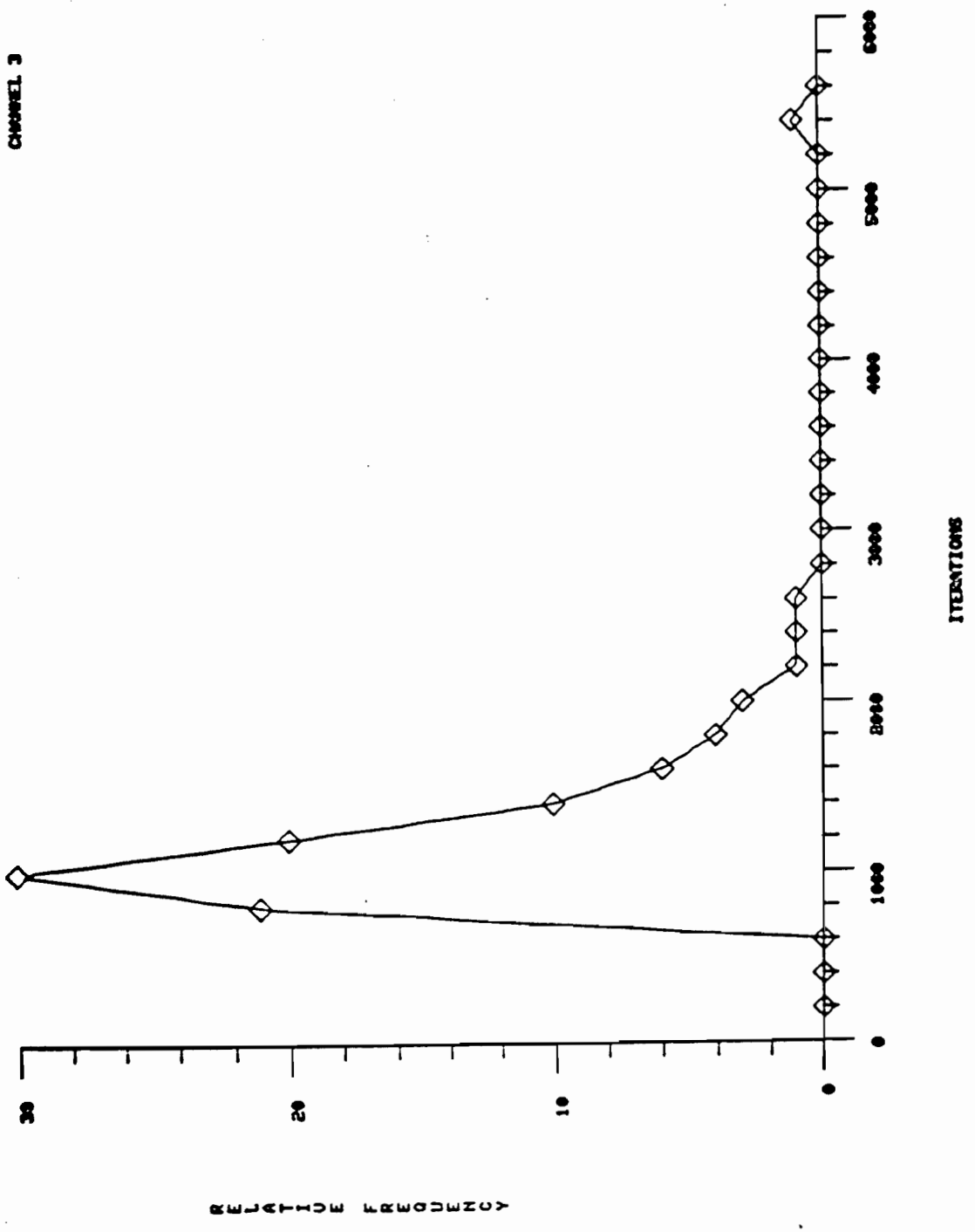


Figure 7.12 Convergence Histogram of Update Covariance Algorithm in Channel 3

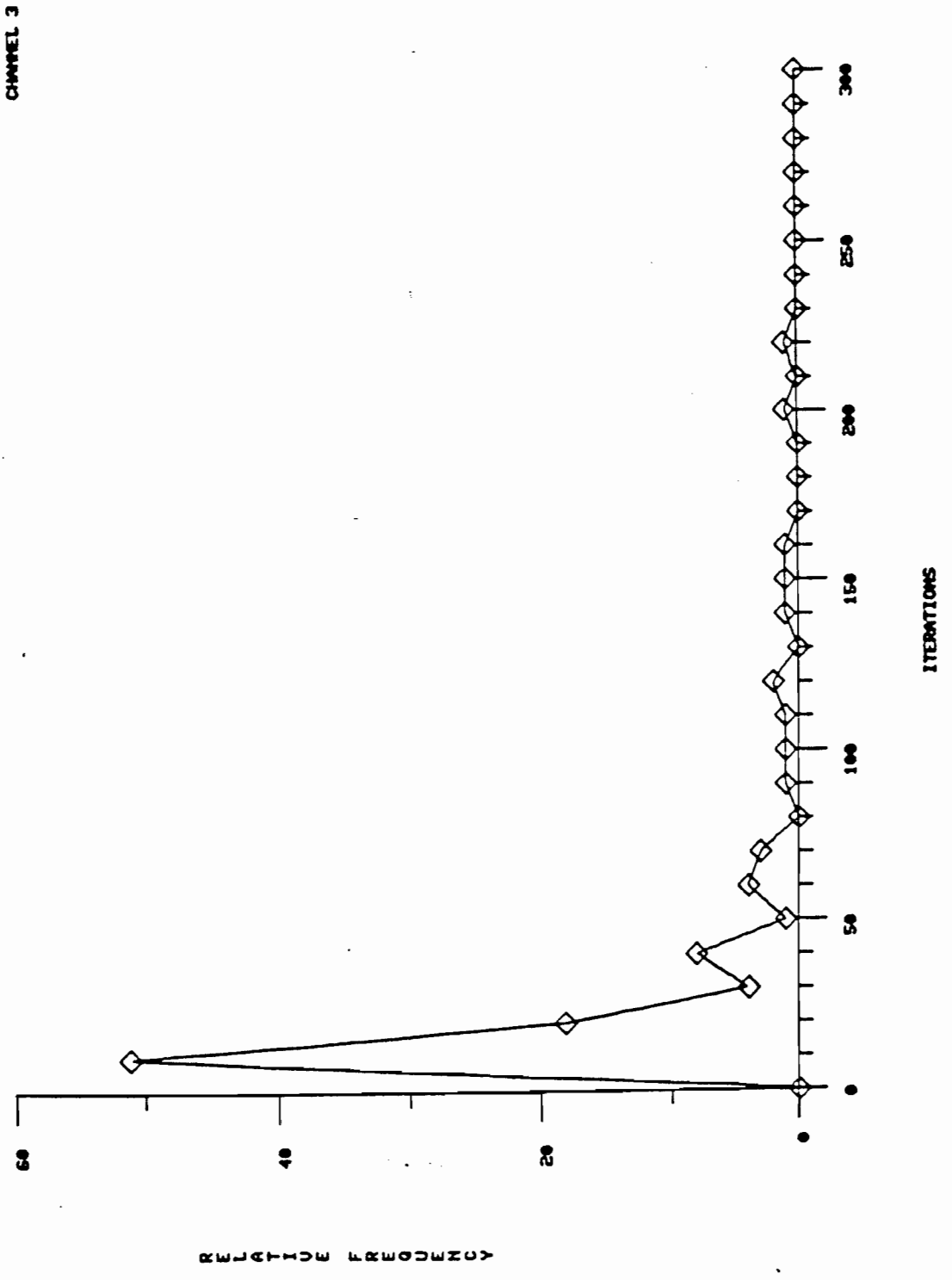


TABLE 7.3 TEST3 SUMMARY

ADAPTIVE ALGORITHM	NUMBER OF CONVERGENCES	AVERAGE NUMBER OF ITERATIONS	STANDARD DEVIATION
LMS	100	542.5	172.6
CONSTR. LMS	100	1143.5	542.6
UPDATE COV.	100	28.8	43.1

SUMMARY OF PLOTS FOR TEST3

Figure T3.1: Unadapted antenna plot at $\phi = 90°$, $\theta = 80°$
Purpose: To indicate initial gain in direction of Jammer 1

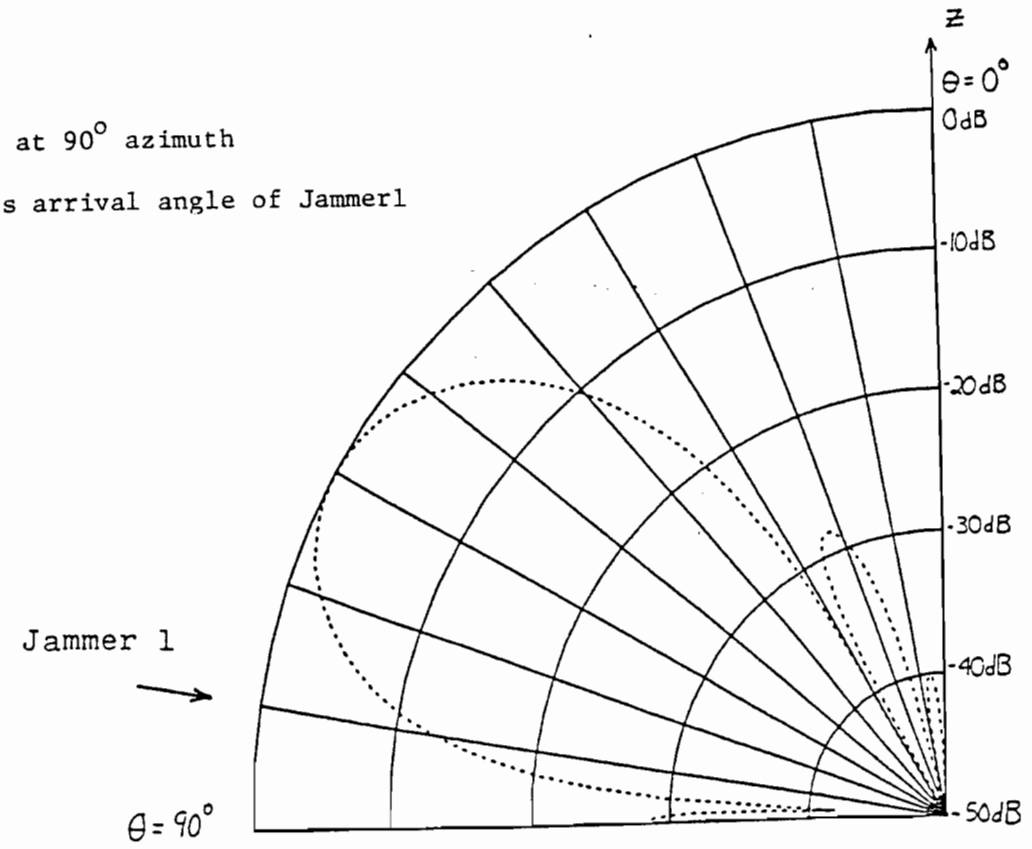
Figure T3.2: LMS-adapted antenna plot at $\phi = 90°$, $\theta = 80°$
Figure T3.3: Constrained LMS-adapted antenna plot at $\phi = 90°$, $\theta = 80°$
Figure T3.4: Update Covariance-adapted antenna plot at $\phi = 90°$, $\theta = 80°$
Purpose: To demonstrate nulling of Jammer 1 by each algorithm

Figure T3.5: Unadapted antenna plot at $\phi = 150°$, $\theta = 75°$
Purpose: To indicate initial gain in direction of Jammer 2

Figure T3.6: LMS-adapted antenna plot at $\phi = 150°$, $\theta = 75°$
Figure T3.7: Constrained LMS-adapted antenna plot at $\phi = 150°$, $\theta = 75°$
Figure T3.8: Update Covariance-adapted antenna plot at $\phi = 150°$, $\theta = 75°$
Purpose: To demonstrate nulling of Jammer 2 by each algorithm

Elevation plot at 90° azimuth

Arrow indicates arrival angle of Jammer1
($\Theta = 80^{\circ}$)



Azimuthal plot at 80° elevation

Arrow indicates arrival angle of Jammer1
($\phi = 90^{\circ}$)

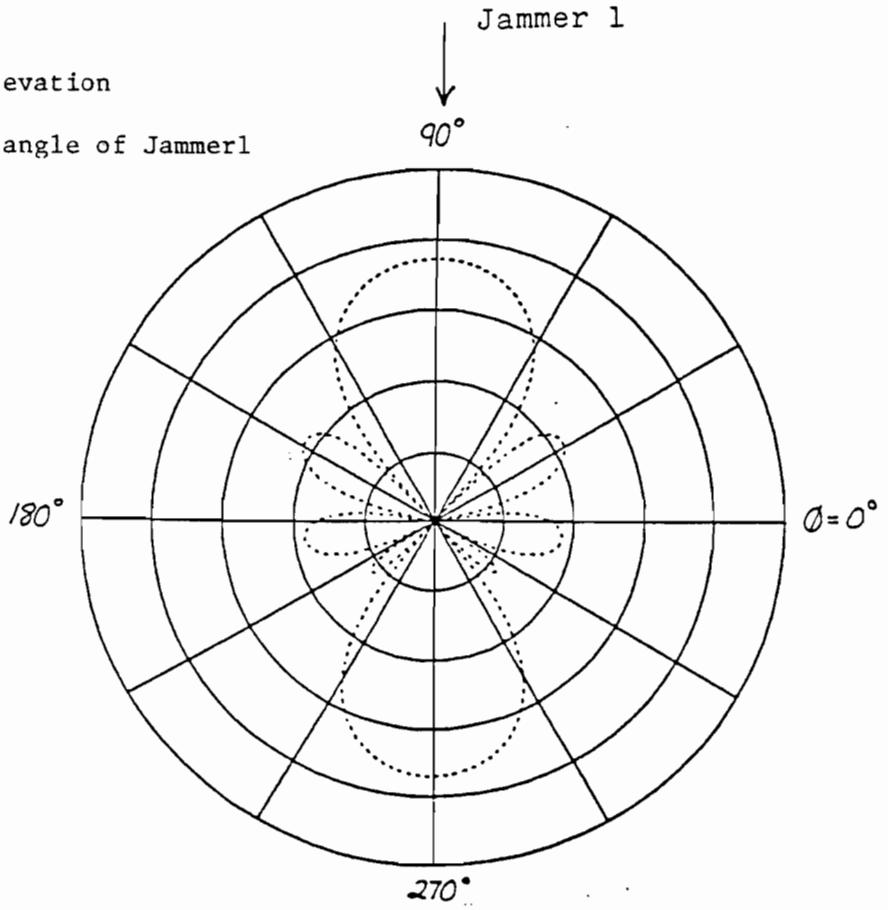


Figure T3.1 Unadapted Antenna Pattern in Direction of Jammer 1

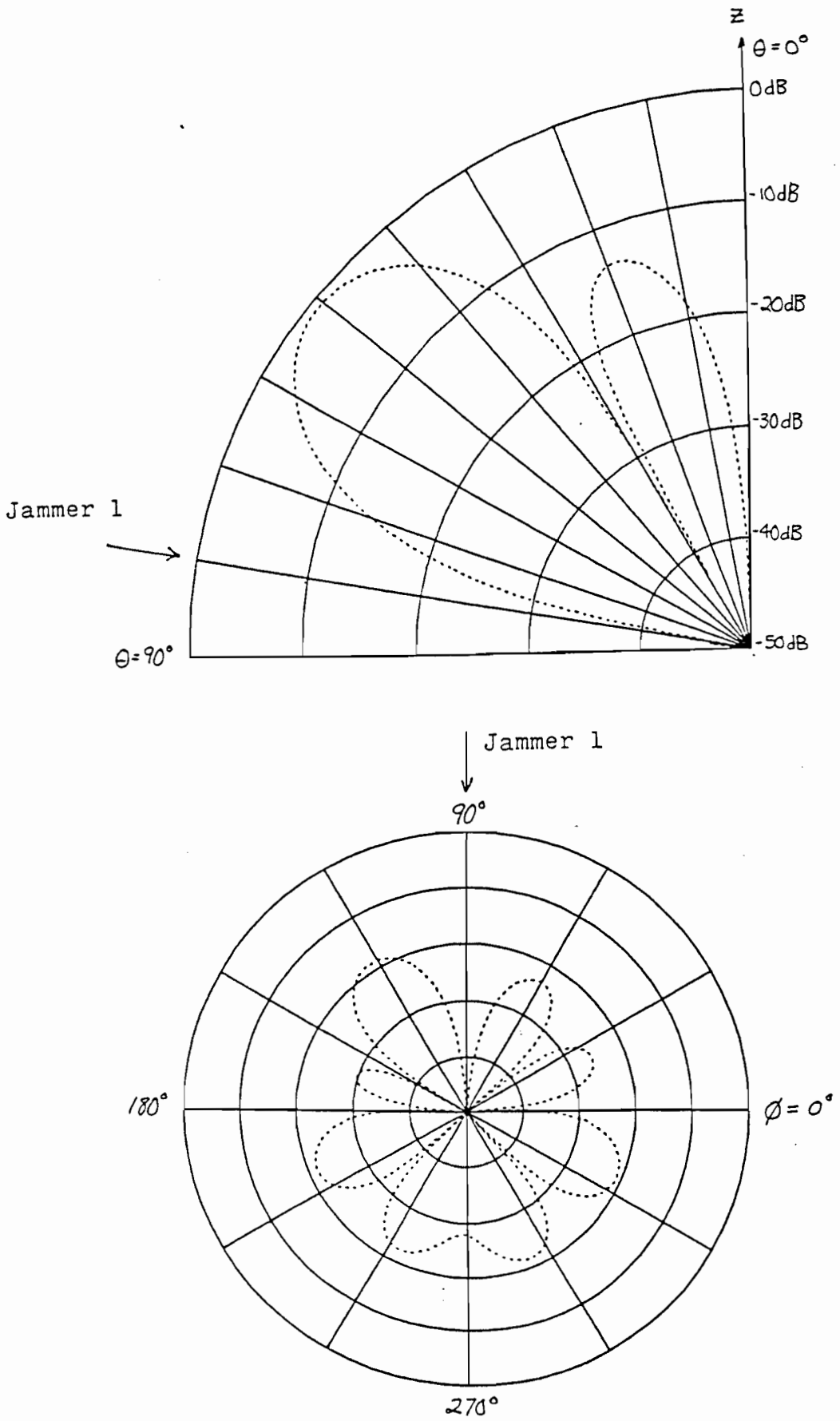


Figure T3.2 LMS-Adapted Pattern in Direction of Jammer1

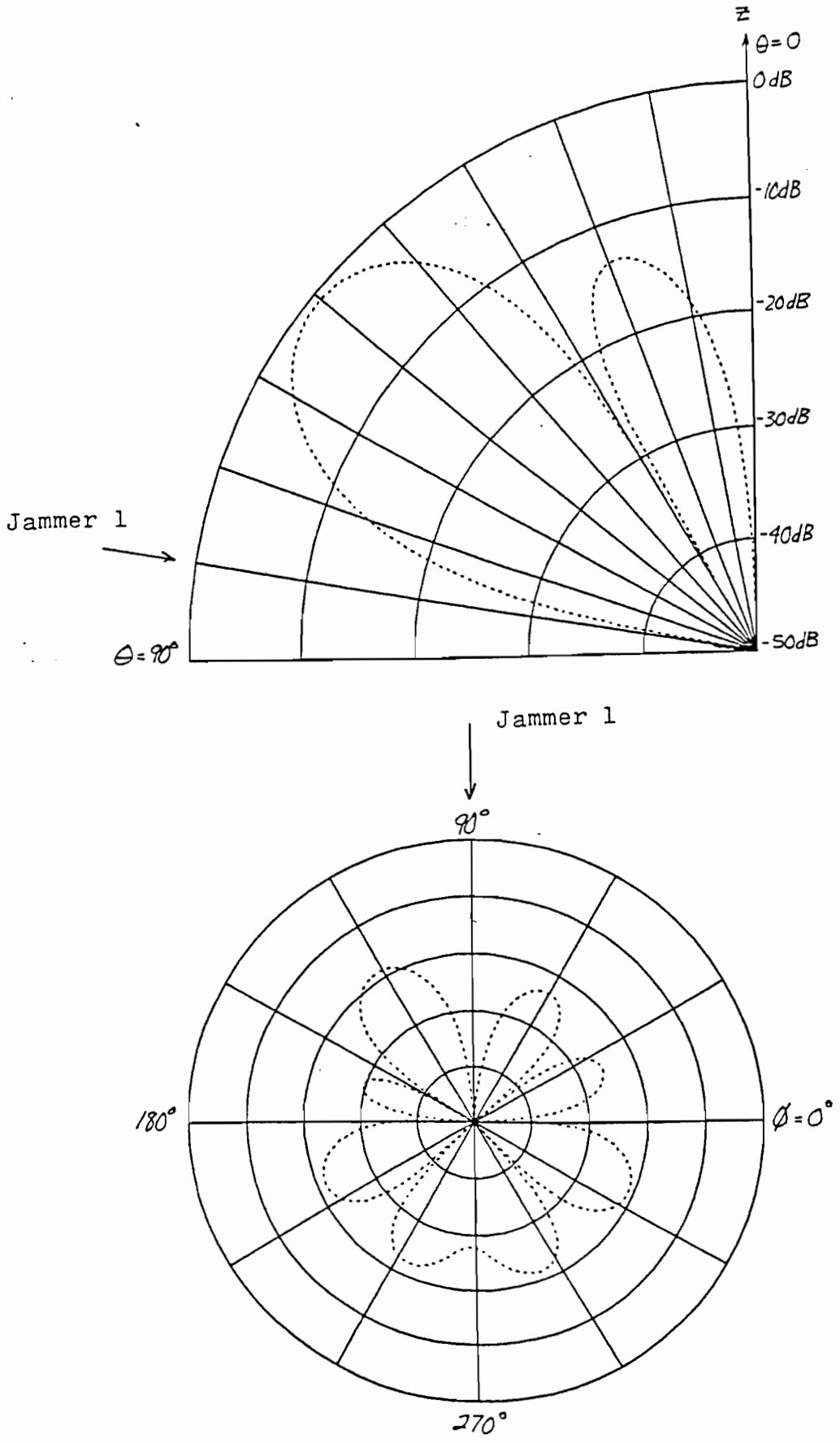


Figure T3.3 Constrained LMS-Adapted Pattern in Direction of Jammer 1

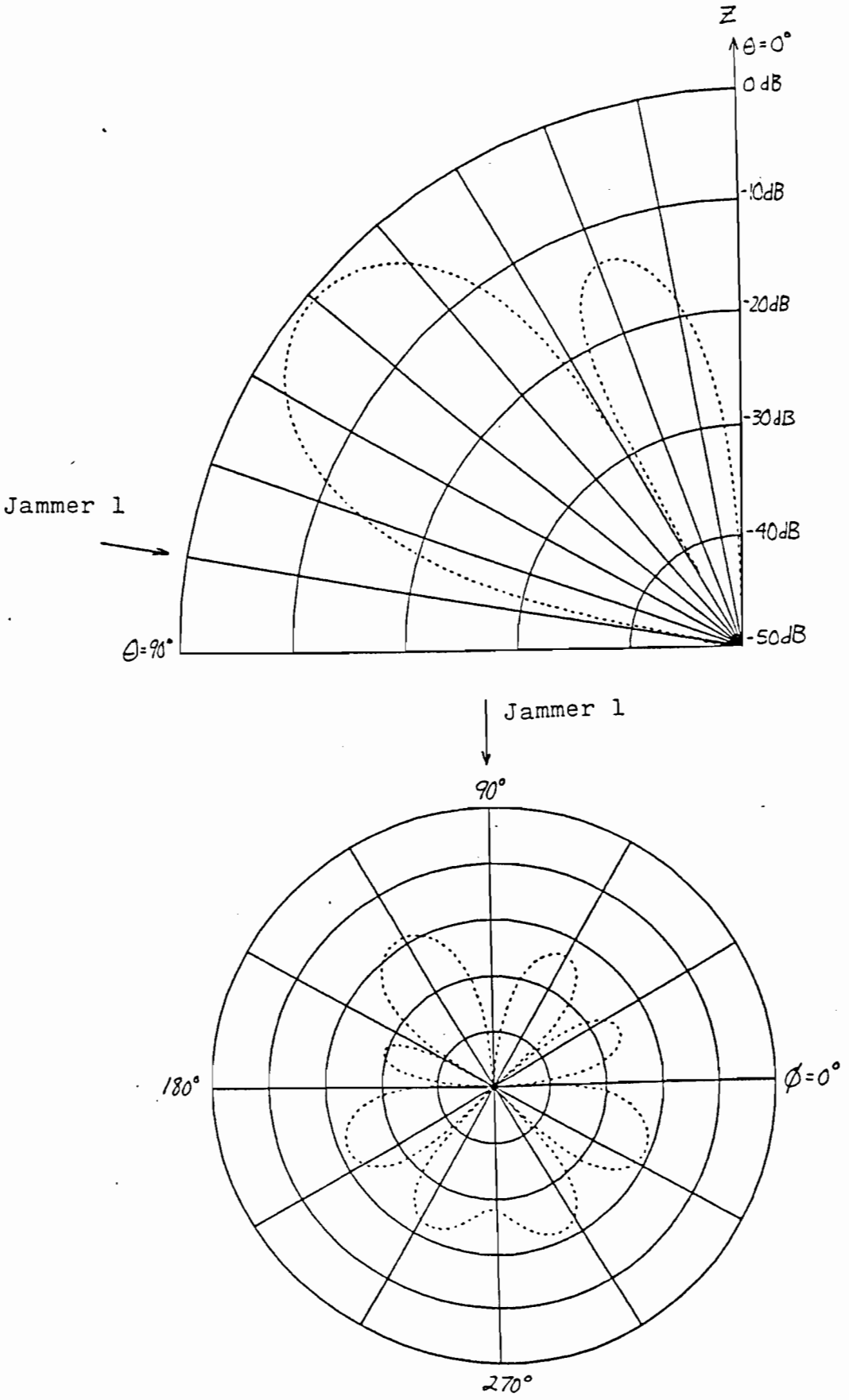
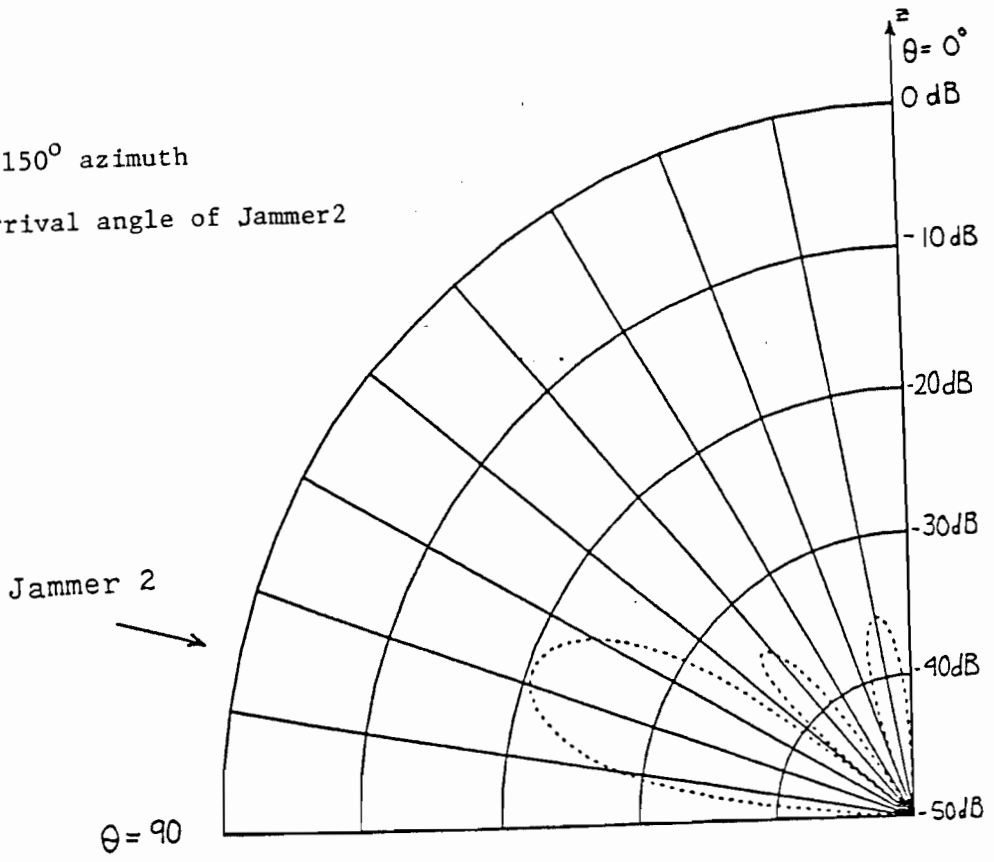


Figure T3.4 Update Covariance-Adapted Pattern in Direction of Jammer 1

Elevation plot at 150° azimuth

Arrow indicates arrival angle of Jammer2
(θ = 75°)



Azimuthal plot at 75° elevation

Arrow indicates arrival angle of Jammer2
(ϕ = 150°)

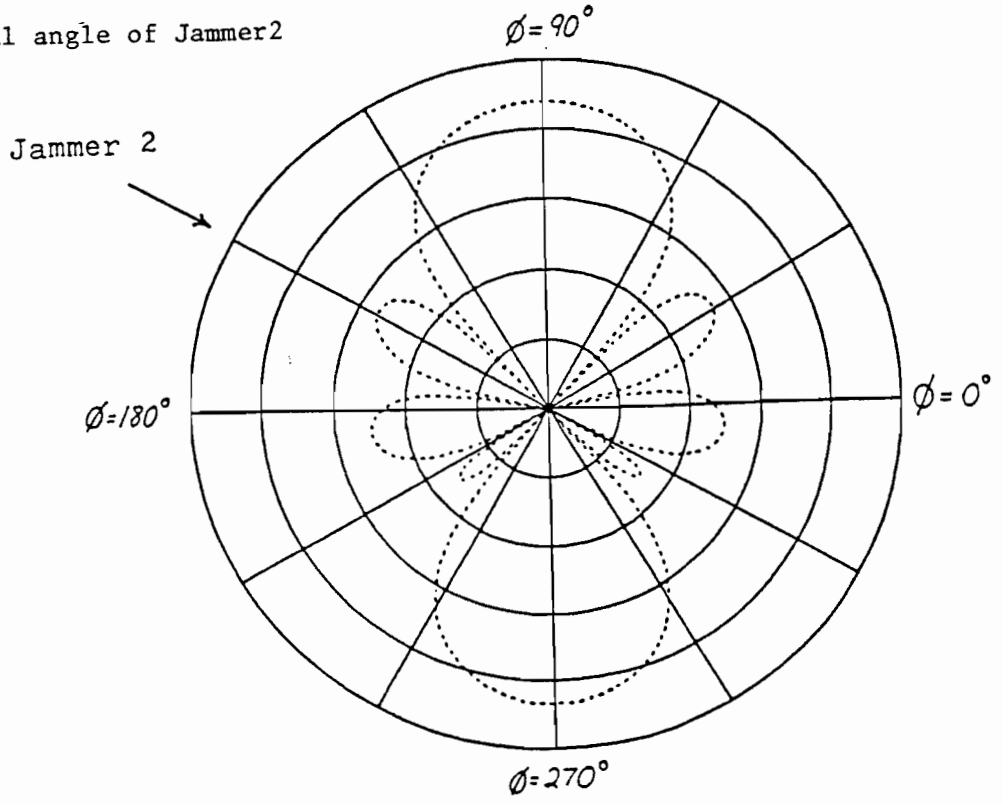
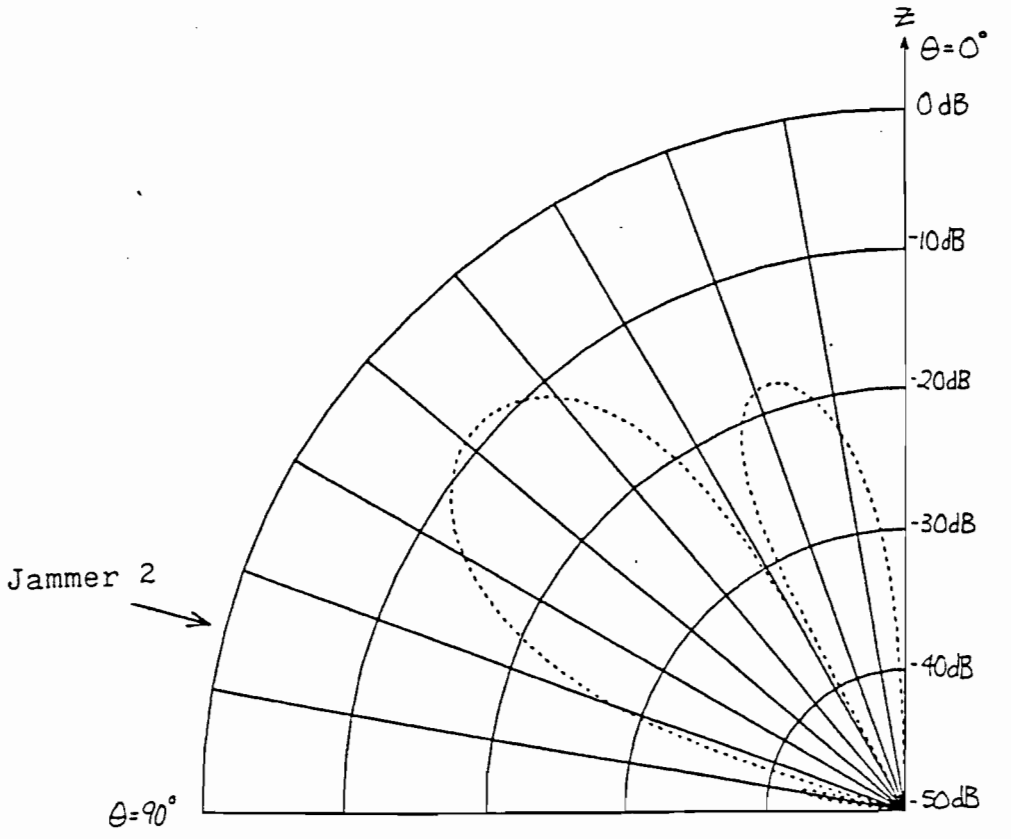


Figure T3.5 Unadapted Antenna Pattern in Direction of Jammer 2



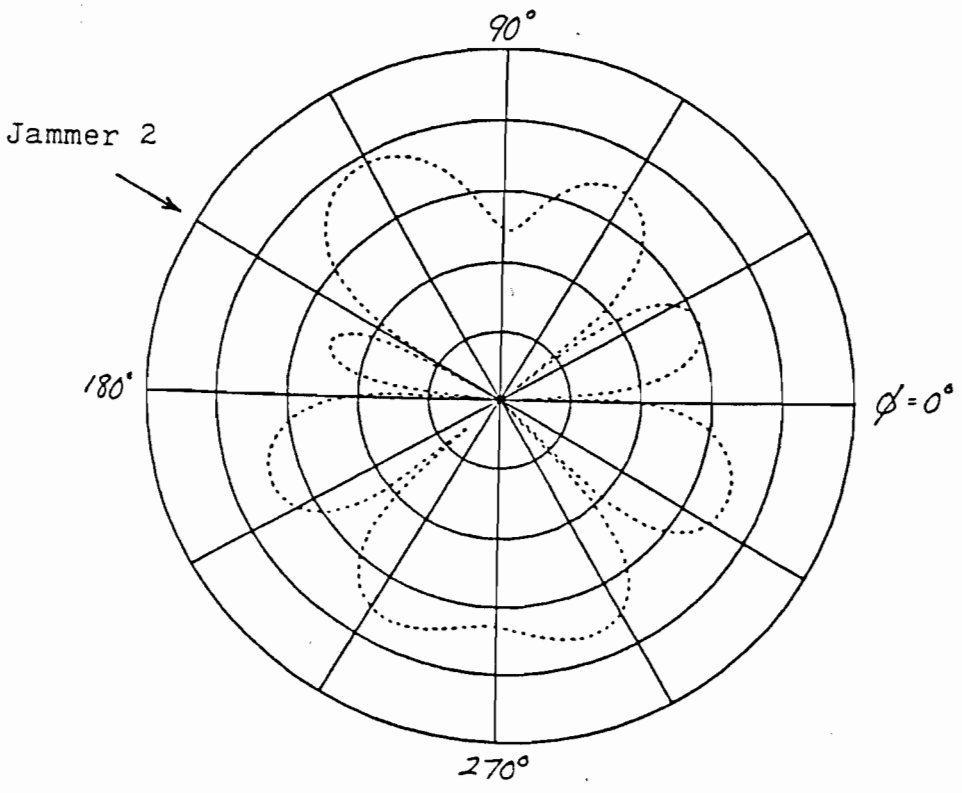


Figure T3.6 LMS-Adapted Pattern in Direction of Jammer 2

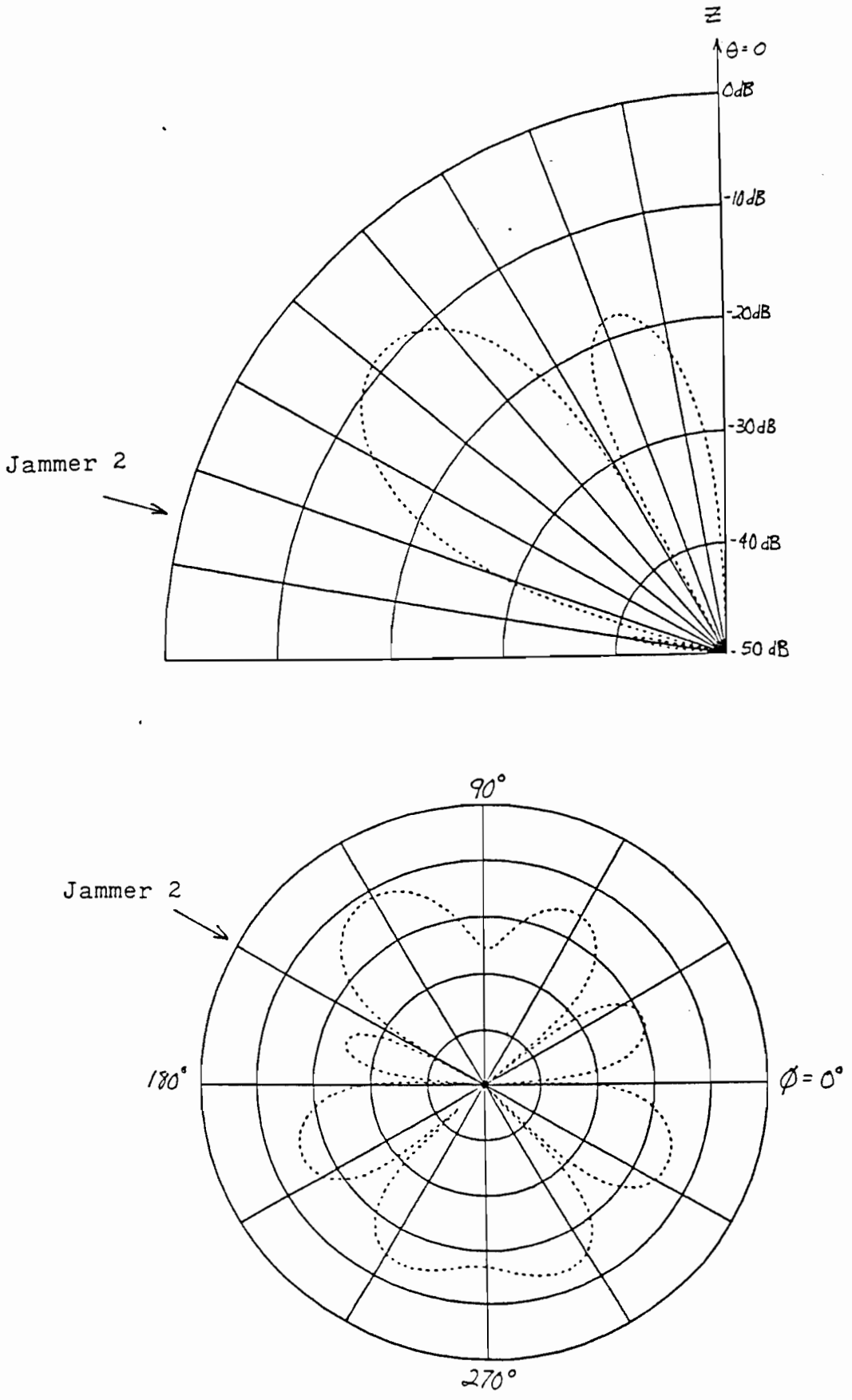


Figure T3.7 Constrained LMS-Adapted Pattern in Direction of Jammer 2

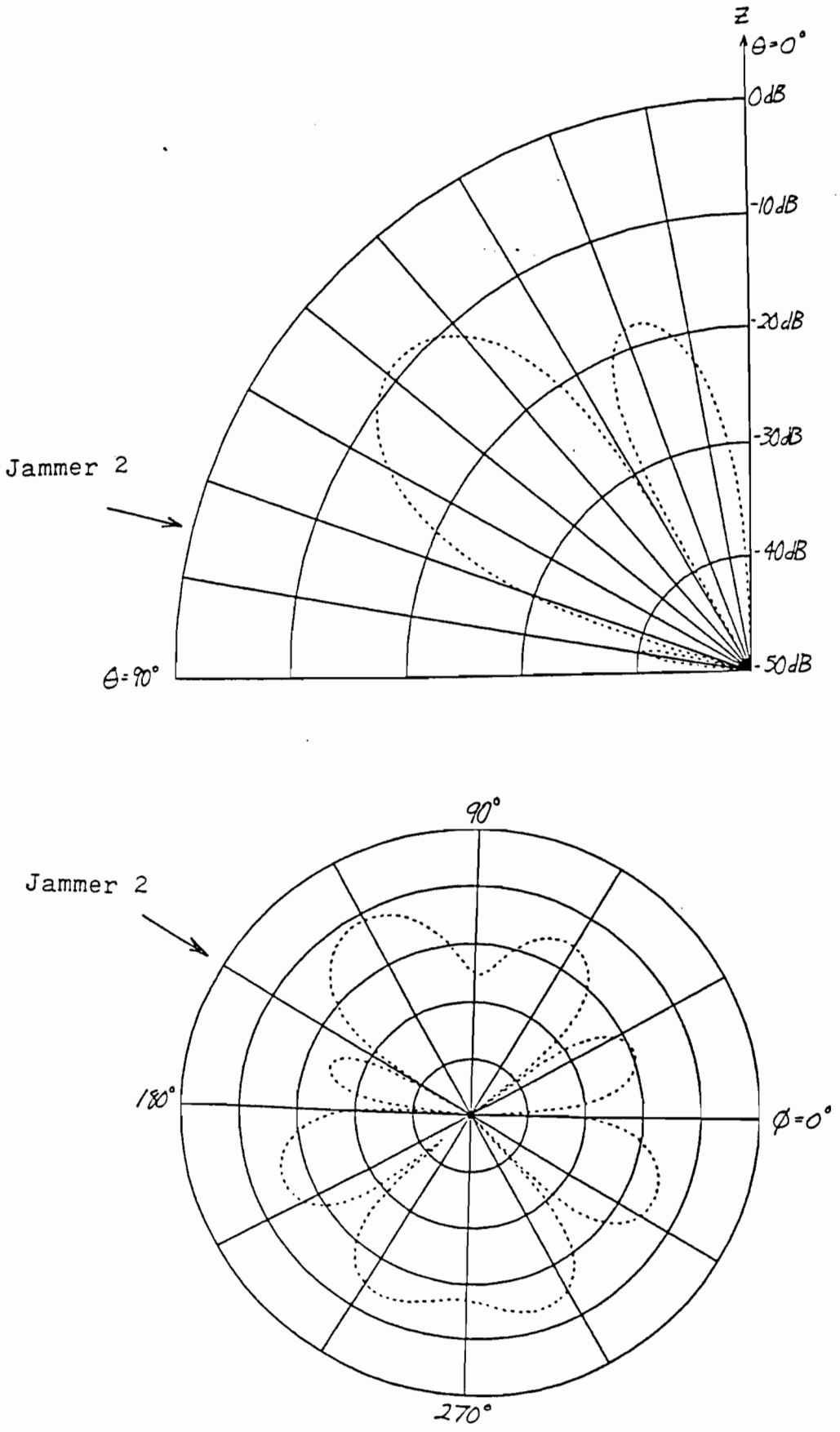


Figure T3.8 Update Covariance-Adapted Pattern in Direction of Jammer 2

7.4 TEST4: Dependence of Convergence of Constrained LMS Algorithm on the Presence of the Signal

It has been mentioned that the Constrained LMS algorithm will perform better when the signal is absent than when it is present. In other words, if the Constrained LMS algorithm was to be selected, sending a preamble would not only be wasteful, it would be detrimental. Tests 1, 2 and 3 will now be repeated for the Constrained LMS with the "desired" signal (preamble) being absent.

In order to perform this test, the simulation had to be effectively "fooled." If the signal power is zero, the SNR is undefined. Therefore, the optimum values will be calculated as if the signal was present (signal power = 1). In other words, the Constrained LMS algorithm will have to attain the same value of SNR as it did before.

The results are tabulated in Table 7.4. The polar antenna plots that follow verify that the Constrained LMS algorithm places deep nulls in the directions of the jammers without the benefit of any preamble whatsoever. The number of average iterations required also validates the assumption that the algorithm will perform better in the absence of the signal.

Notice that the average number of required iterations is now virtually identical to that of the LMS algorithm. Also recall that the Constrained LMS algorithm (with the signal present) suffered large fluctuations in required iterations for channels 2 and 3. When the signal is absent, however, the algorithm actually had the smallest percentage deviation of all the algorithms tested. This is graphically illustrated by the histograms of Figures 7.13 - 7.15. This fact only strengthens the claim that the existence of the three paths (and the signal magnification that results) is the primary culprit of the convergence problems suffered by the Constrained LMS algorithm. When this effect was nullified by eliminating the signal, convergence was fairly rapid and quite consistent.

Figure 7.13 Convergence Histogram of Constrained LMS Algorithm in Ideal Channel (no signal present)

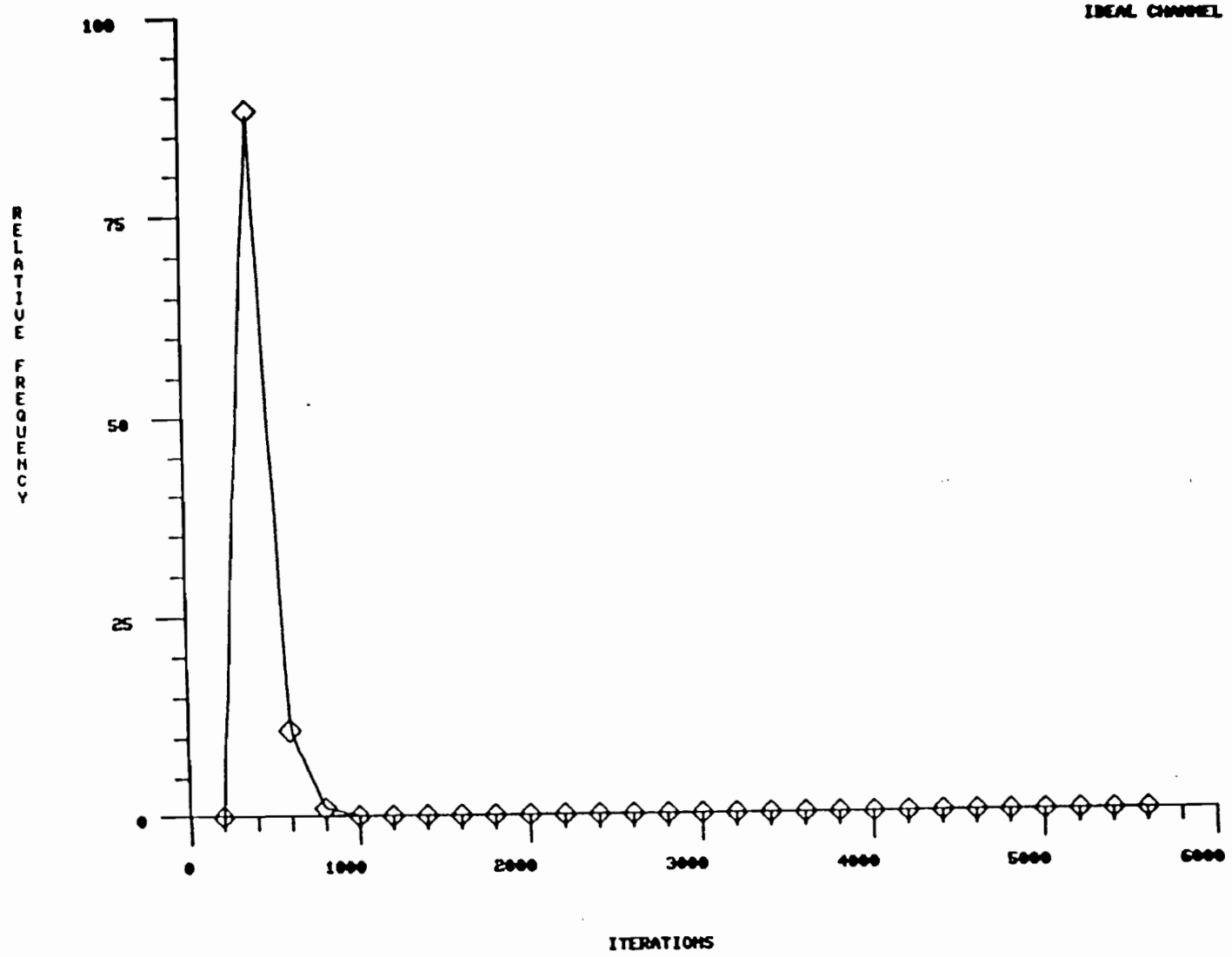


Figure 7.14 Convergence Histogram of Constrained LMS Algorithm in Channel 2 (no signal present)

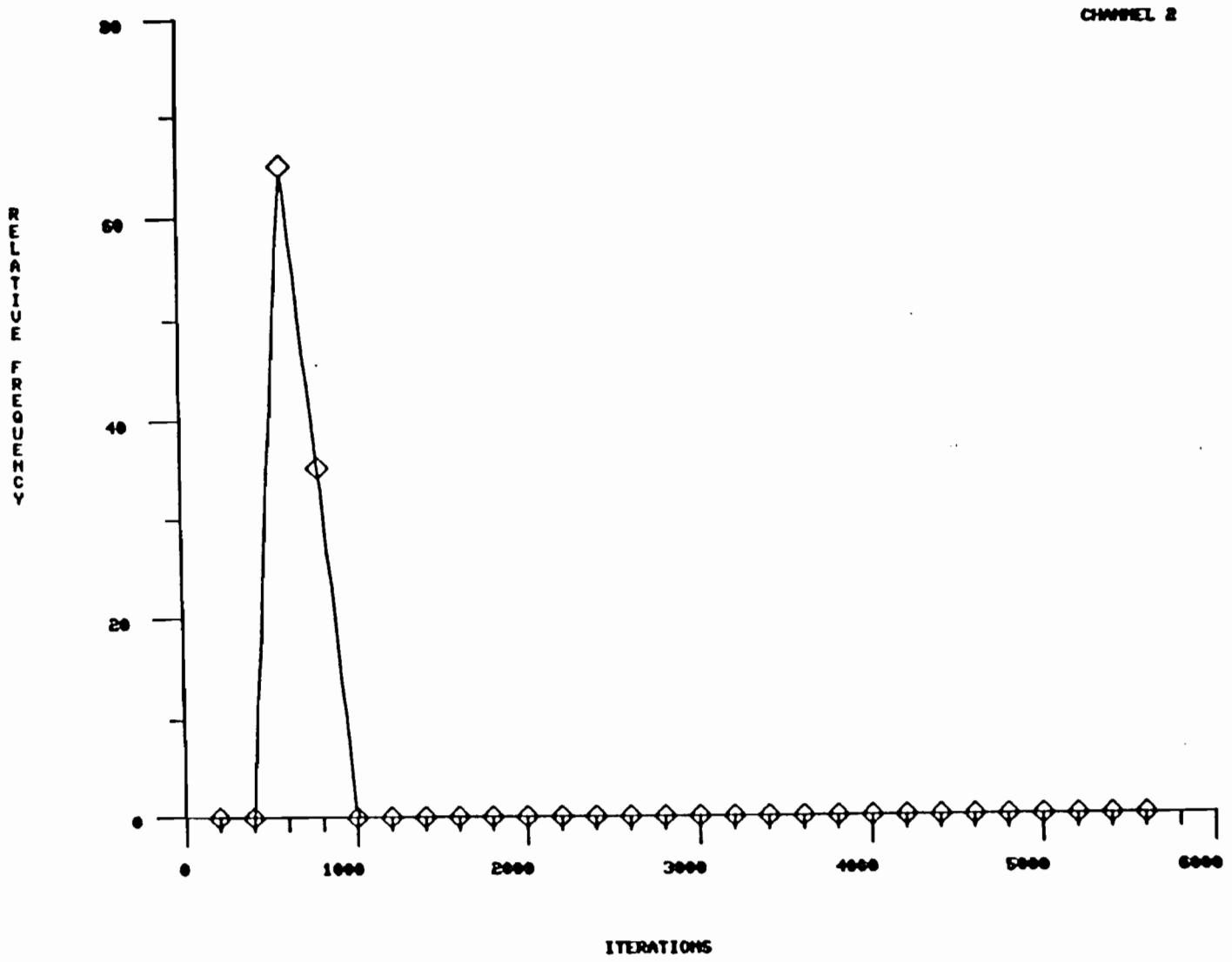


Figure 7.15 Convergence Histogram of Constrained LMS Algorithm in Channel 3
(no signal present)

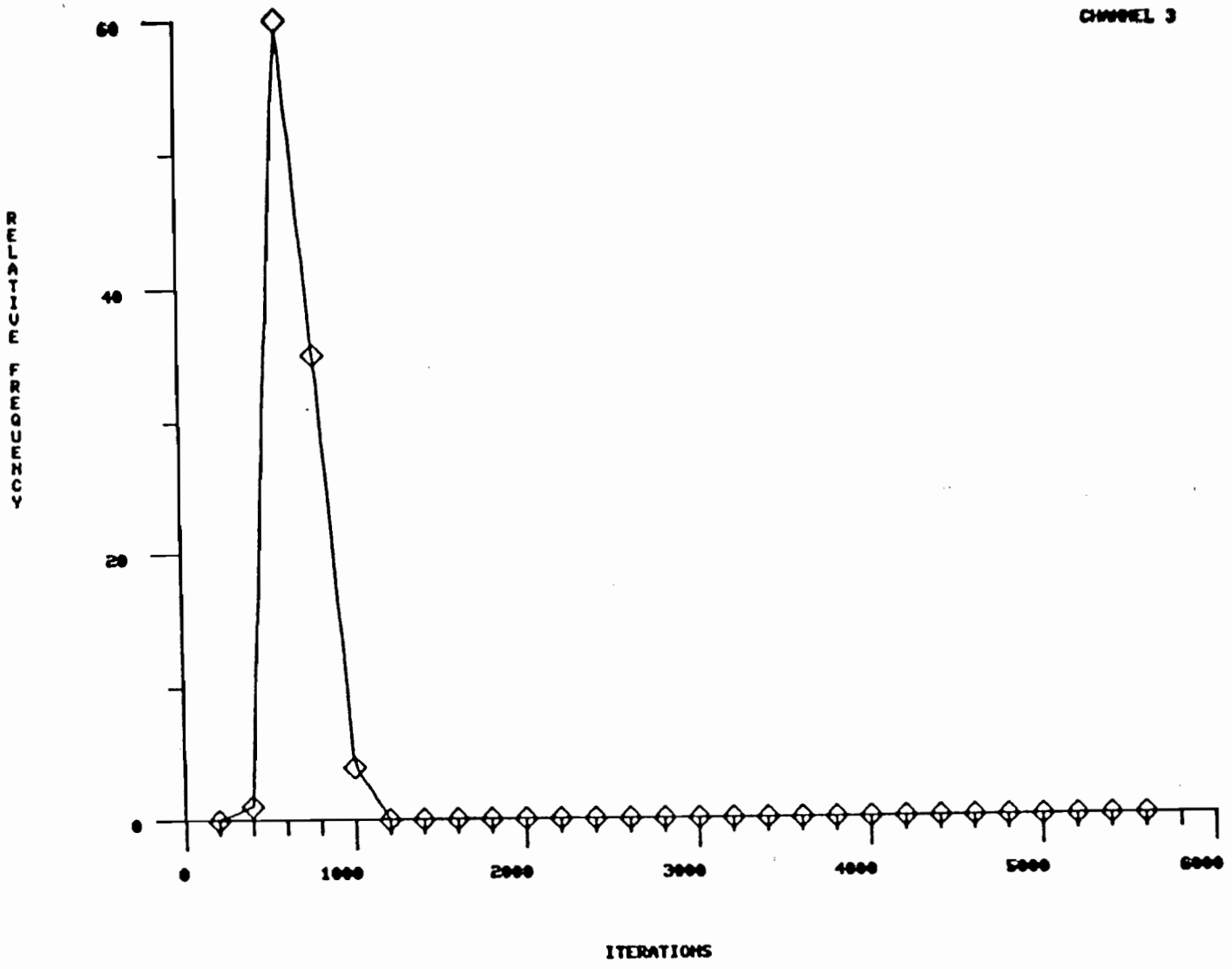
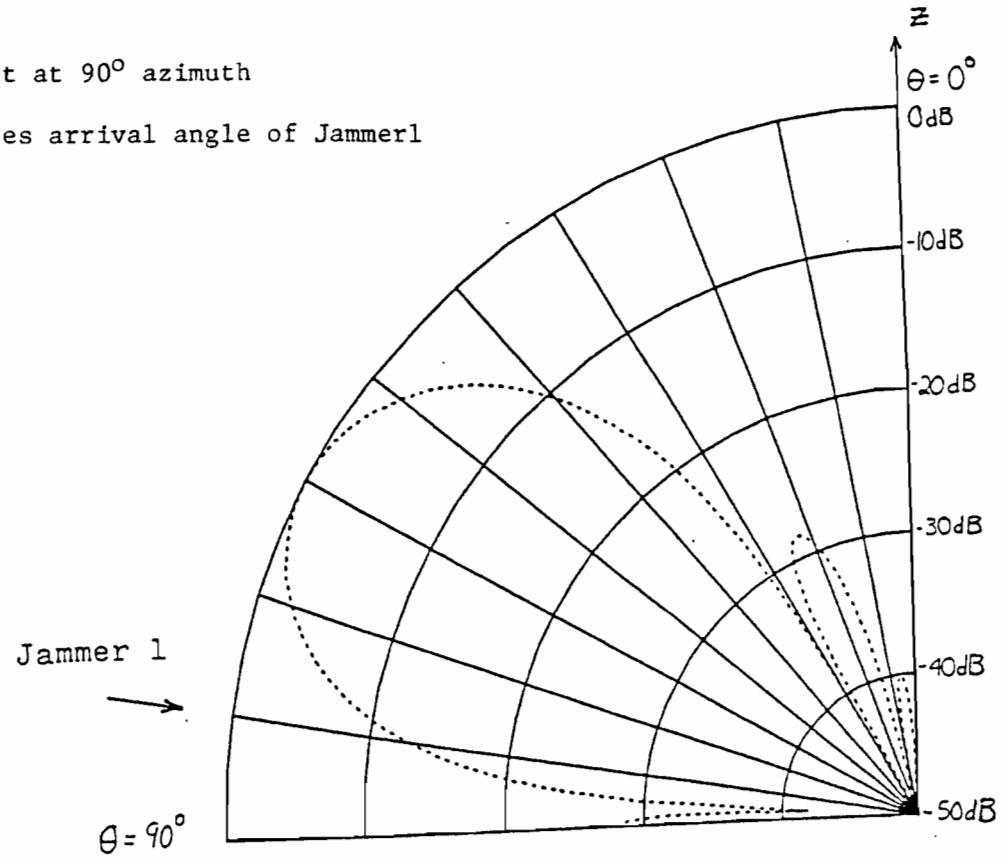


TABLE 7.4 TEST4 SUMMARY

CHANNEL	NUMBER OF CONVERGENCES	AVERAGE NUMBER OF ITERATIONS	STANDARD DEVIATION
IDEAL	100	337.0	81.1
CHANNEL 2	100	593.0	88.1
CHANNEL 3	100	598.5	105.7

Elevation plot at 90° azimuth

Arrow indicates arrival angle of Jammer1
($\theta = 80^{\circ}$)



Azimuthal plot at 80° elevation

Arrow indicates arrival angle of Jammer1
($\phi = 90^{\circ}$)

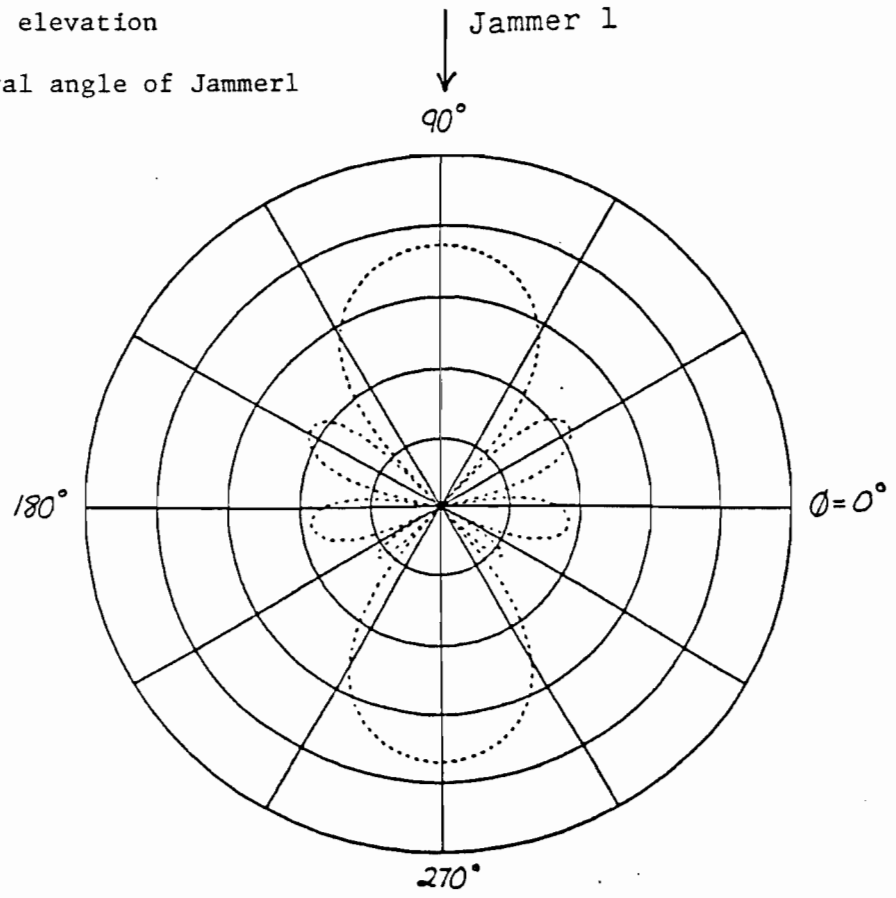


Figure T4.1 Unadapted Antenna Pattern in Direction of Jammer 1

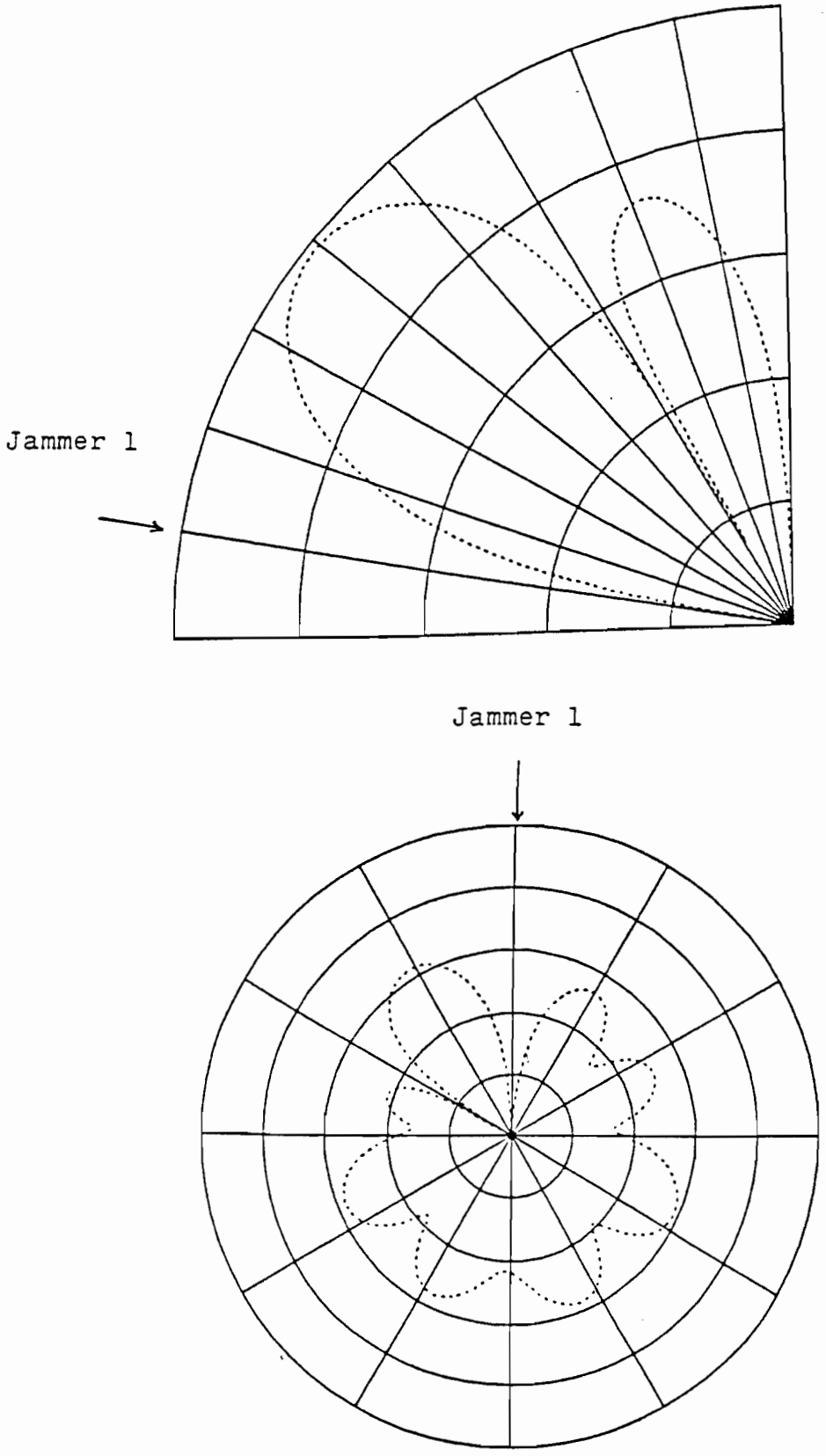
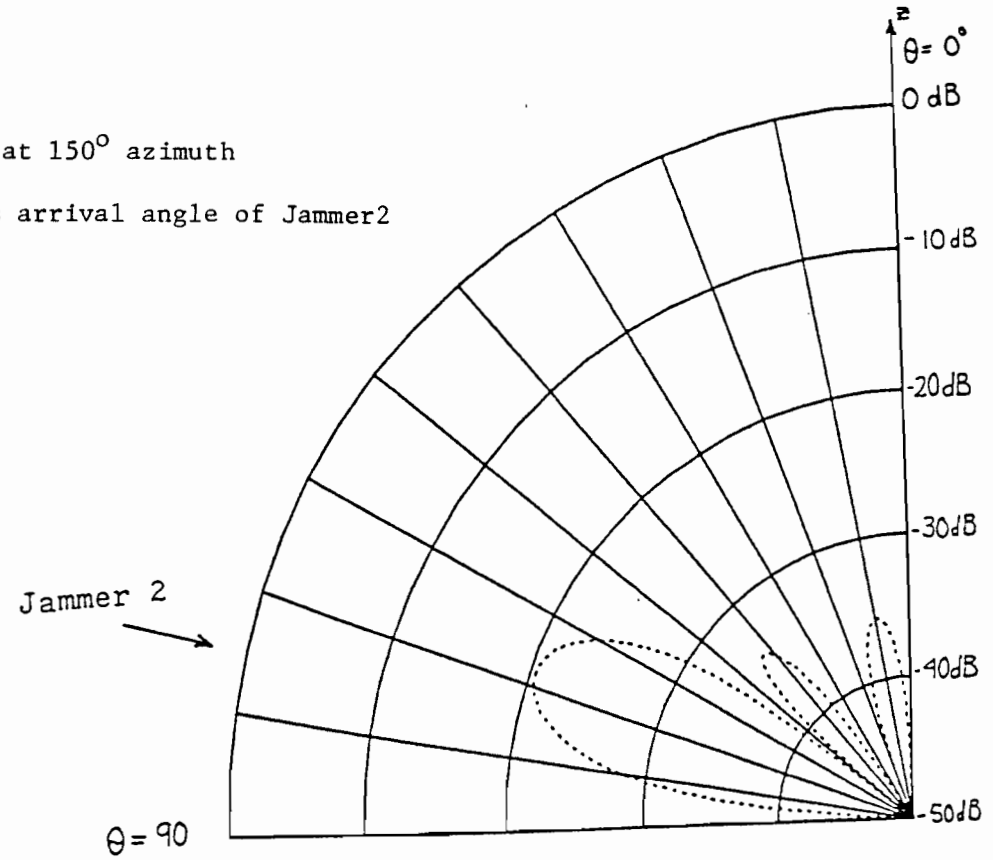


Figure T4.2 Adapted Pattern in Direction of Jammer 1

Elevation plot at 150° azimuth

Arrow indicates arrival angle of Jammer2
(θ = 75°)



Azimuthal plot at 75° elevation

Arrow indicates arrival angle of Jammer2
(ϕ = 150°)

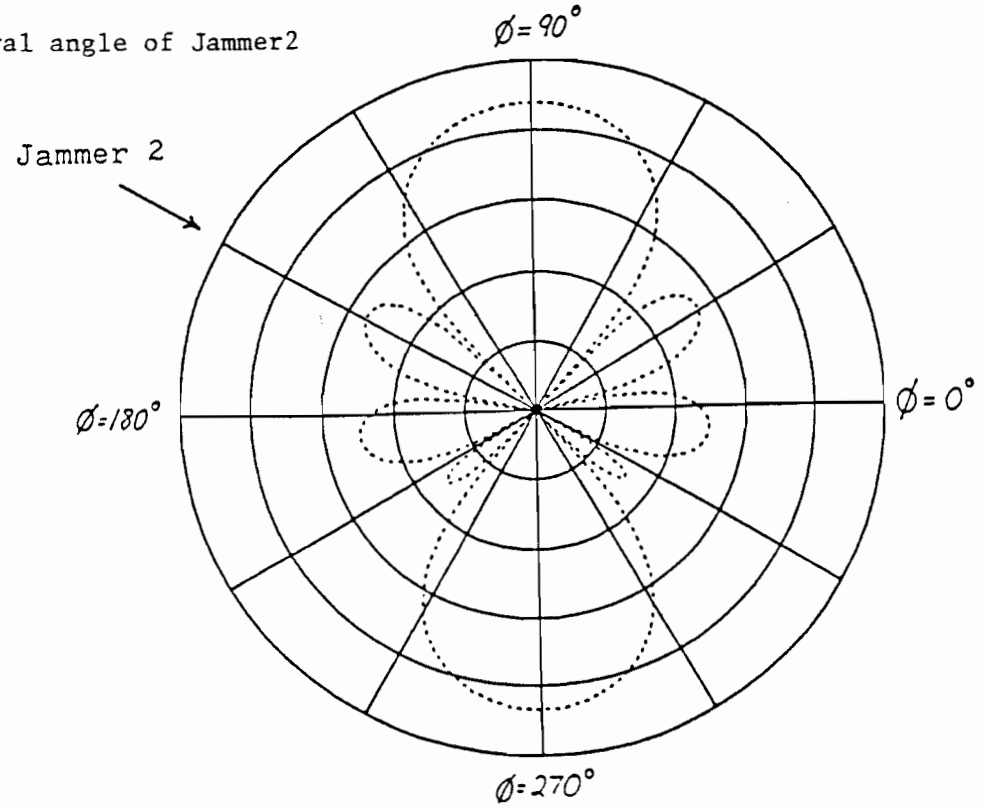


Figure T4.3 Unadapted Antenna Pattern in Direction of Jammer 2

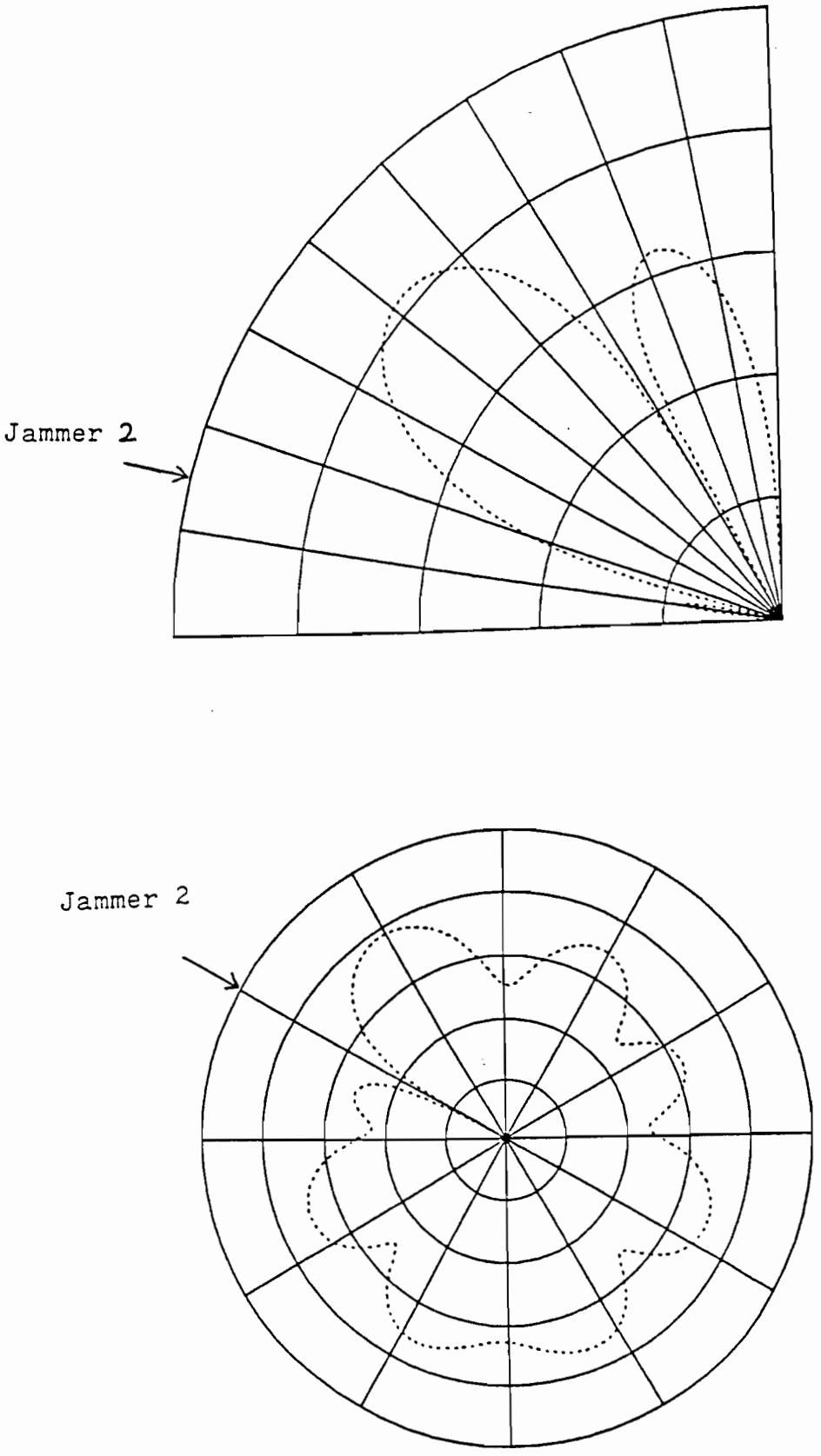
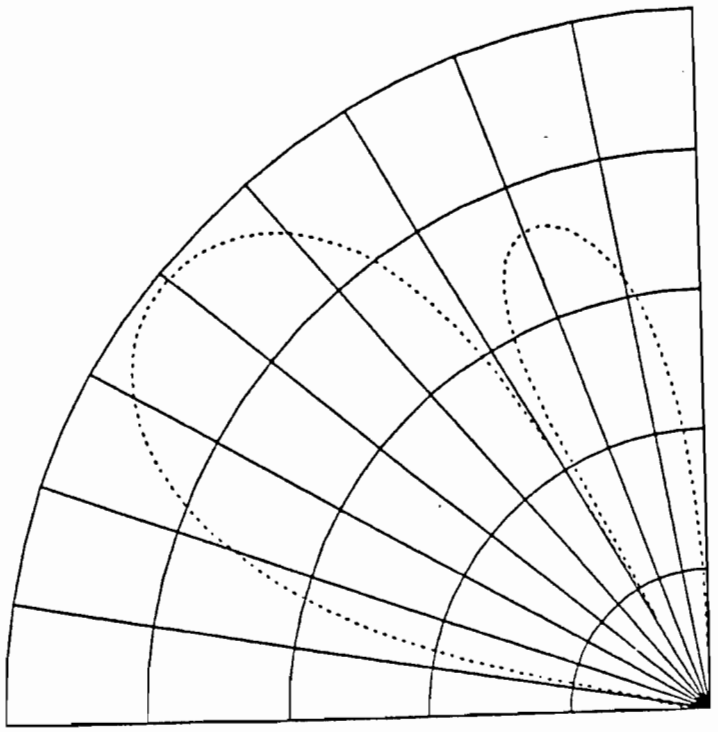


Figure T4.4 Adapted Pattern in Direction of Jammer 2

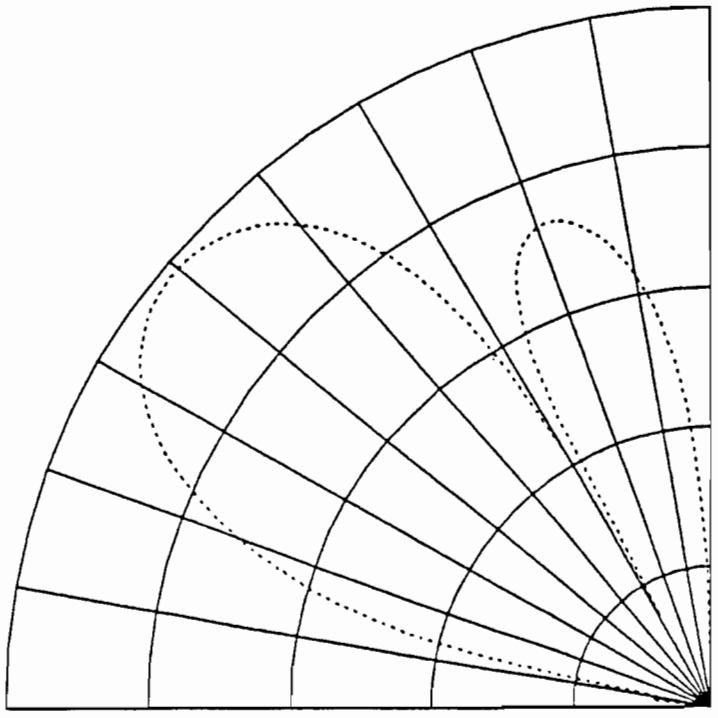
7.5 TEST5: Dependence of Adapted Antenna Pattern on Number of Required Iterations

The histograms that have been presented graphically demonstrate that quite often there is a considerable difference in the number of required iterations for an algorithm to converge. The question may arise whether or not the adapted antenna patterns produced by widely separated convergences are significantly different.

The polar antenna plots that follow were generated using the fastest and slowest convergences of the LMS algorithm of TEST2. It can be seen the antenna plots are virtually indistinguishable. This indicates that there is not a marked difference in the final adapted antenna patterns produced by the widely separated convergences. This is certainly not a surprising result, as the performance measure that has been utilized requires that the pattern be close to optimal before the algorithm is said to be converged.

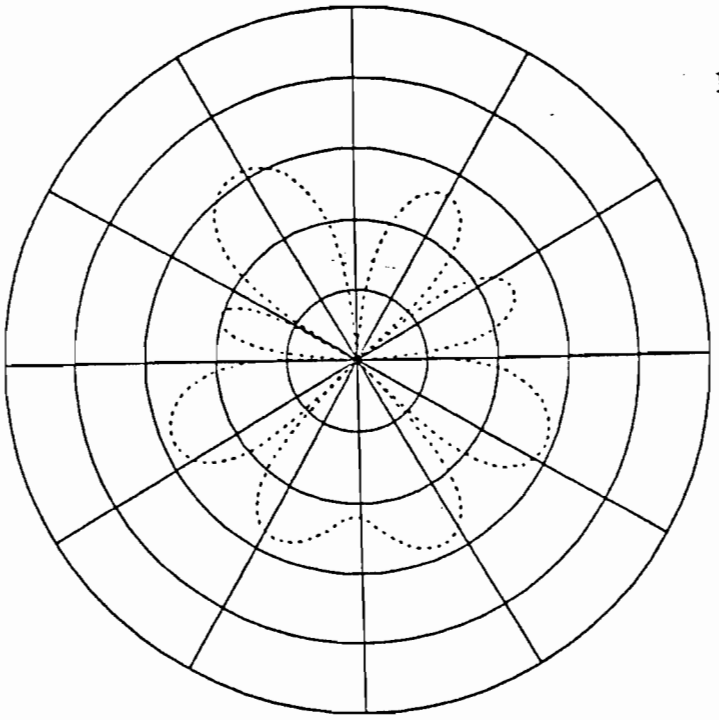


FASTEST CONVERGENCE

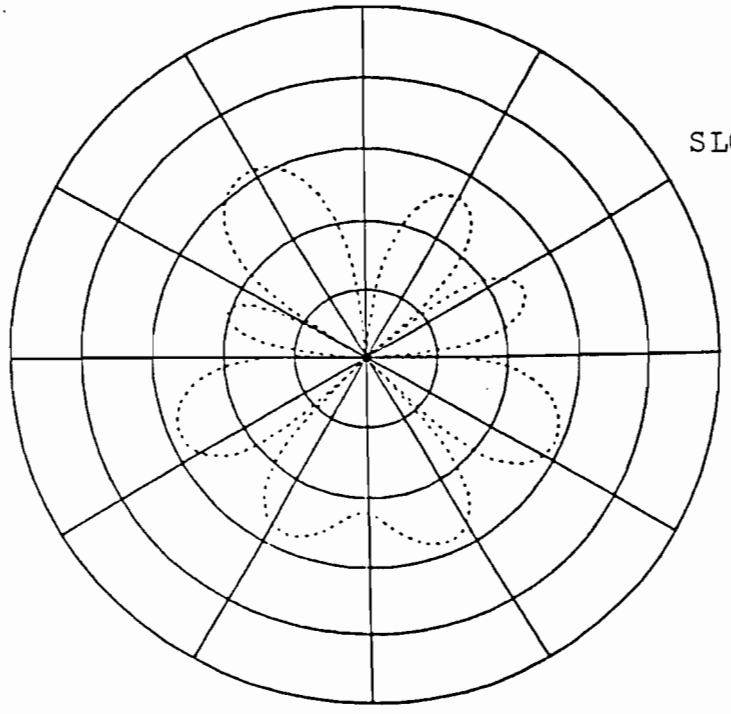


SLOWEST CONVERGENCE

Figure T5.1 Adapted Elevation Patterns at 90^0 Azimuth

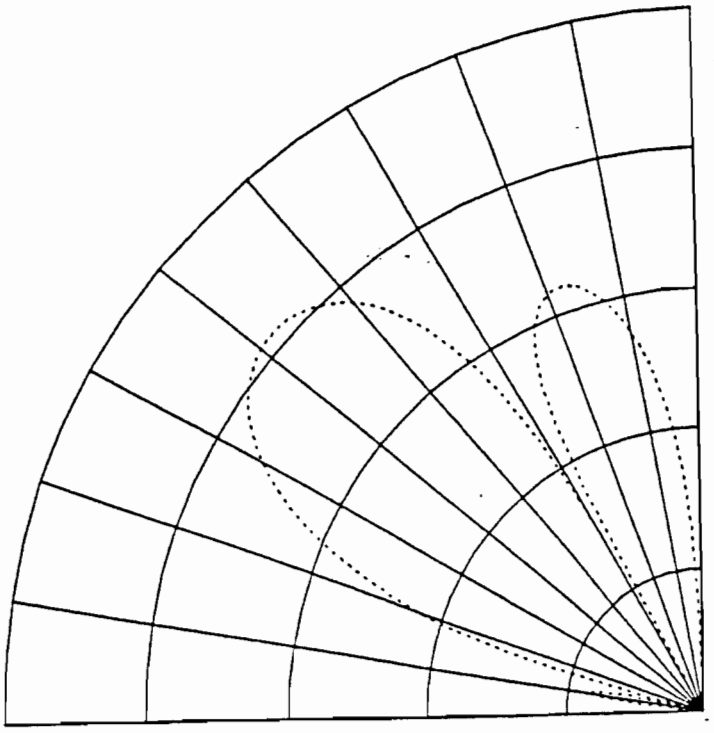


FASTEST CONVERGENCE

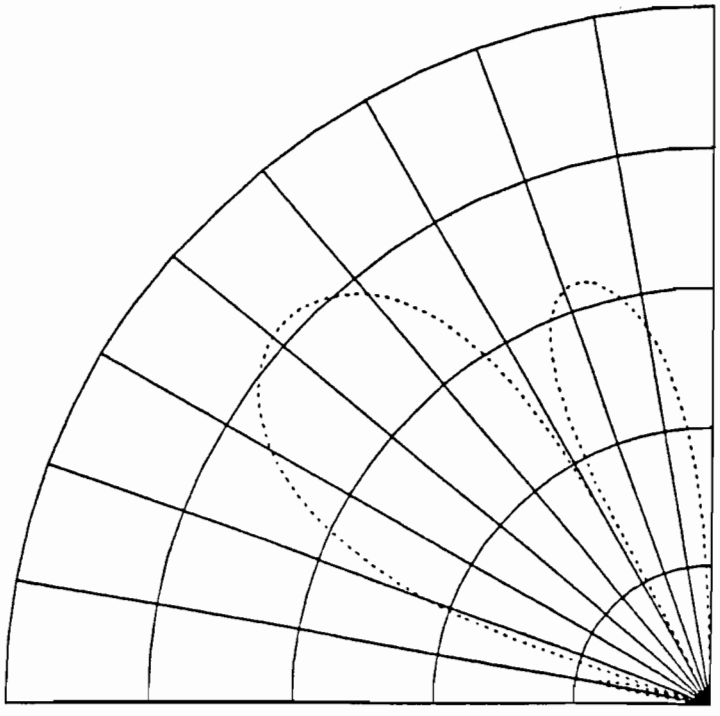


SLOWEST CONVERGENCE

Figure T5.2 Adapted Azimuthal Patterns at 80^{o} Elevation



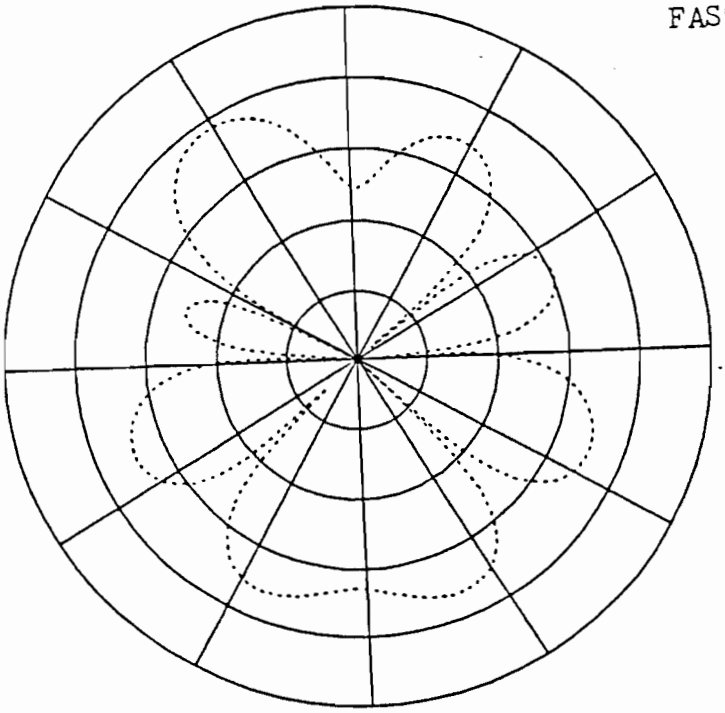
FASTEST CONVERGENCE



SLOWEST CONVERGENCE

Figure T5.3 Adapted Elevation Patterns at 150° Azimuth

FASTEST CONVERGENCE



SLOWEST CONVERGENCE

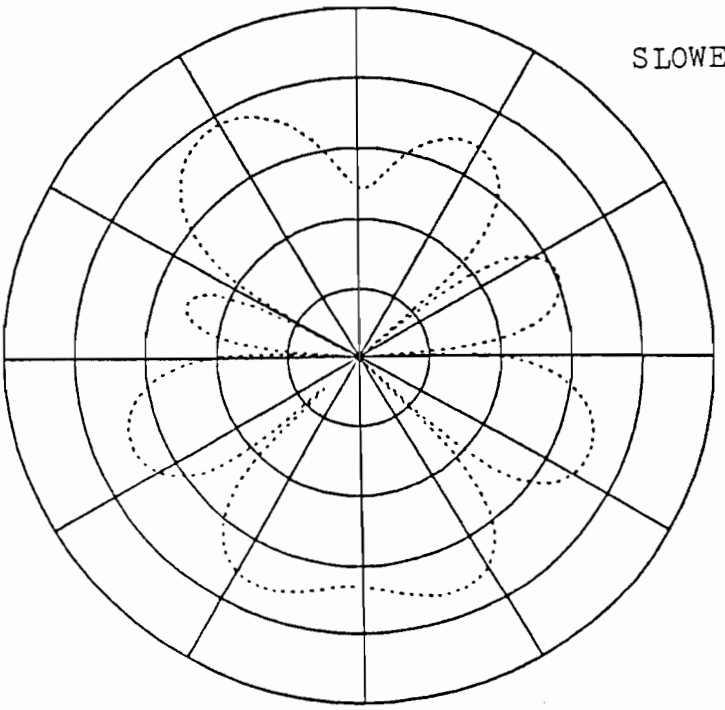


Figure T5.4 Adapted Azimuthal Patterns at 75^{o} Elevation

TEST SUMMARY

For convenience, the results from Tests 1-4 are re-tabulated as shown in Table 7.5. Also, Figures 7.16 - 7.19 track the performances of each algorithm for identical convergences in each of the three channels tested.

Figure 7.16 Convergence Summary for LMS Algorithm

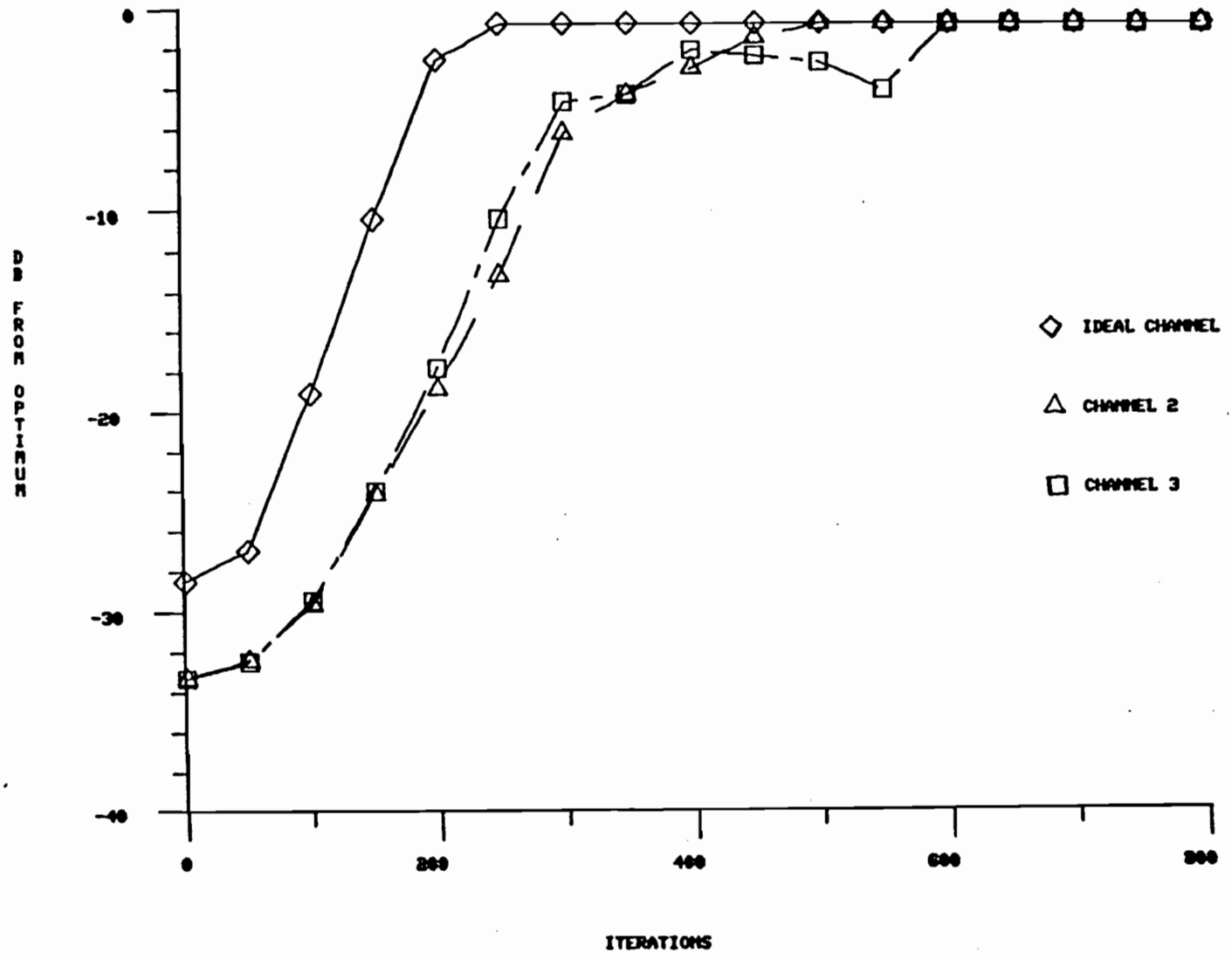


Figure 7.17 Convergence Summary for Constrained LMS Algorithm

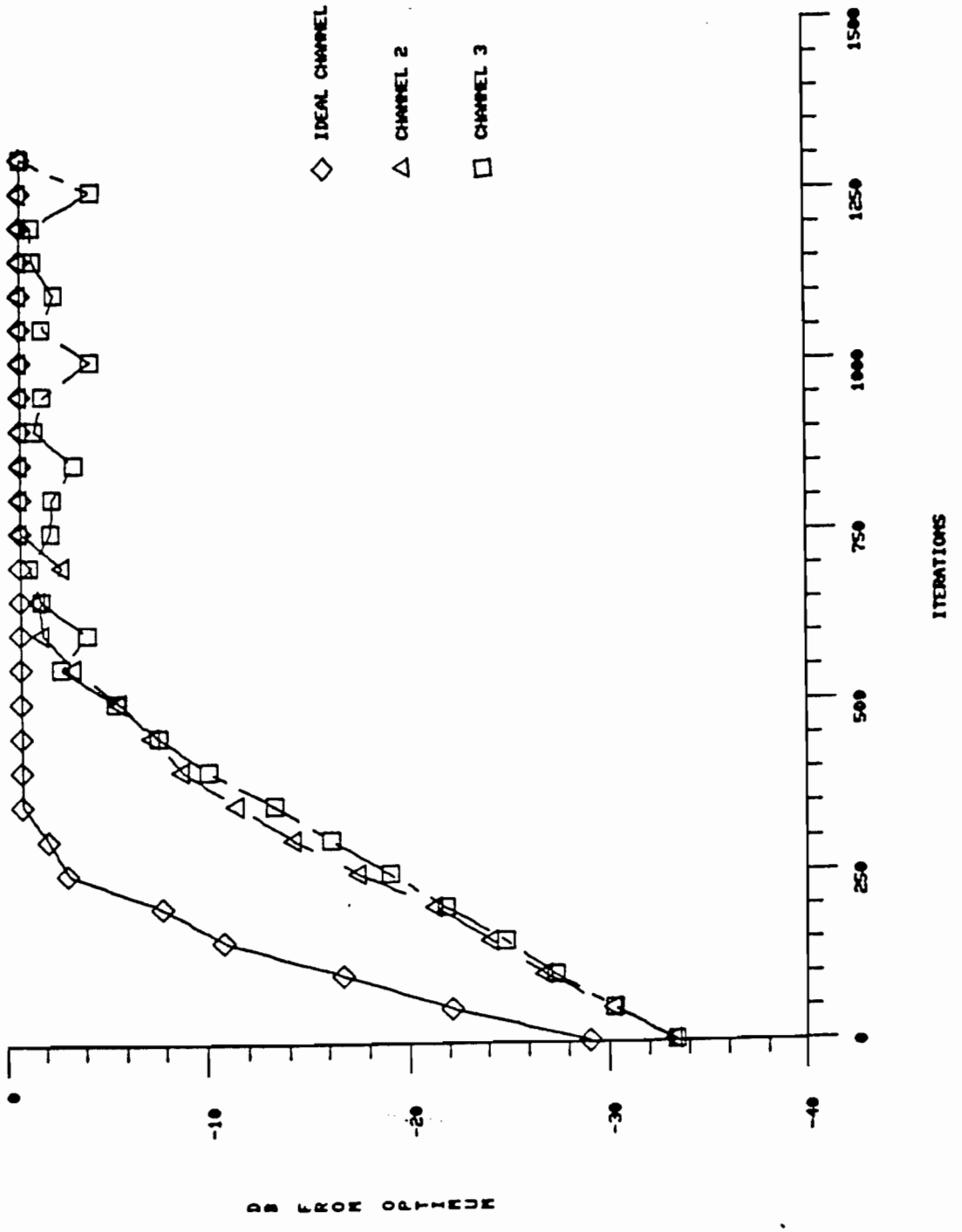


Figure 7.18 Convergence Summary for Update Covariance Algorithm

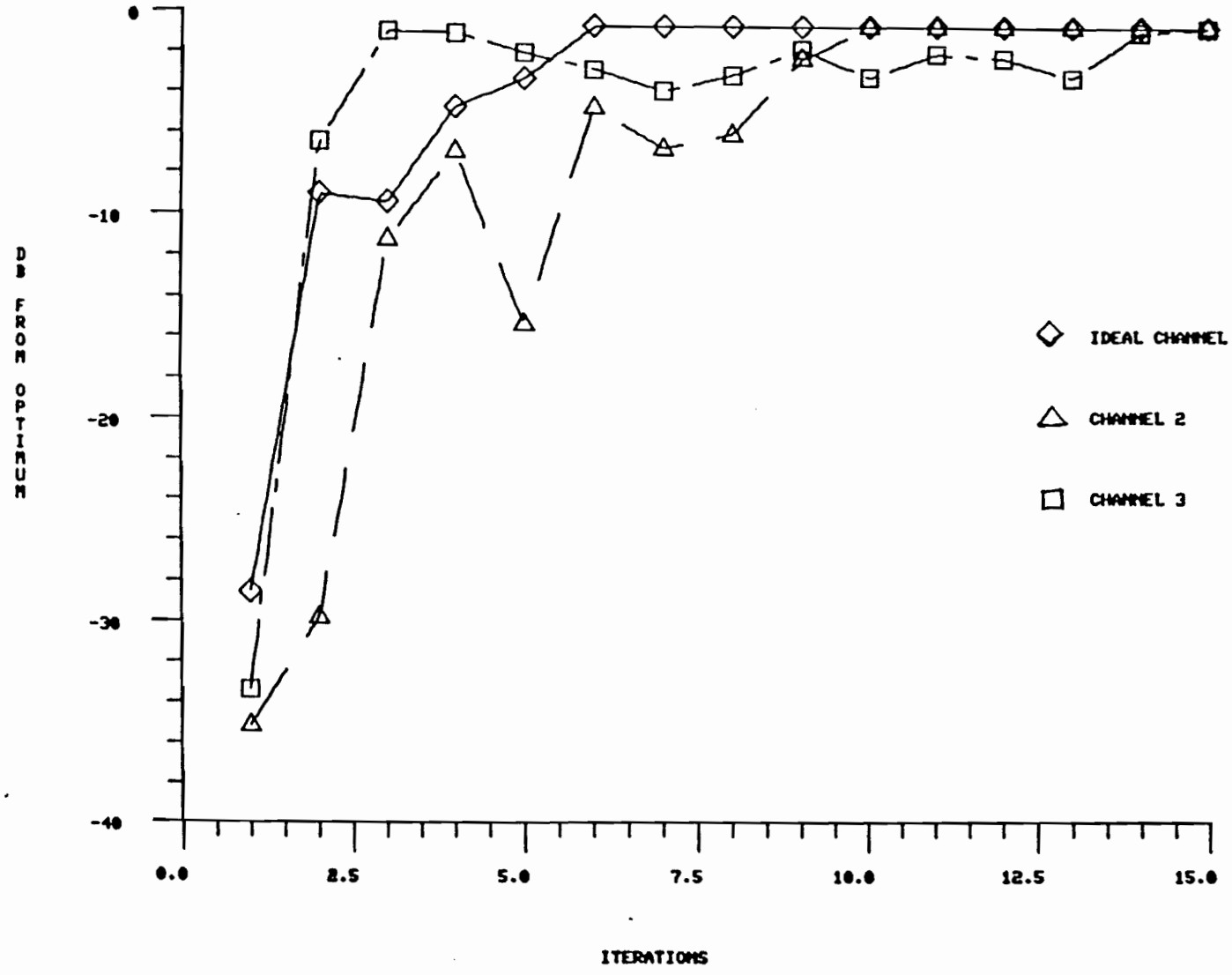
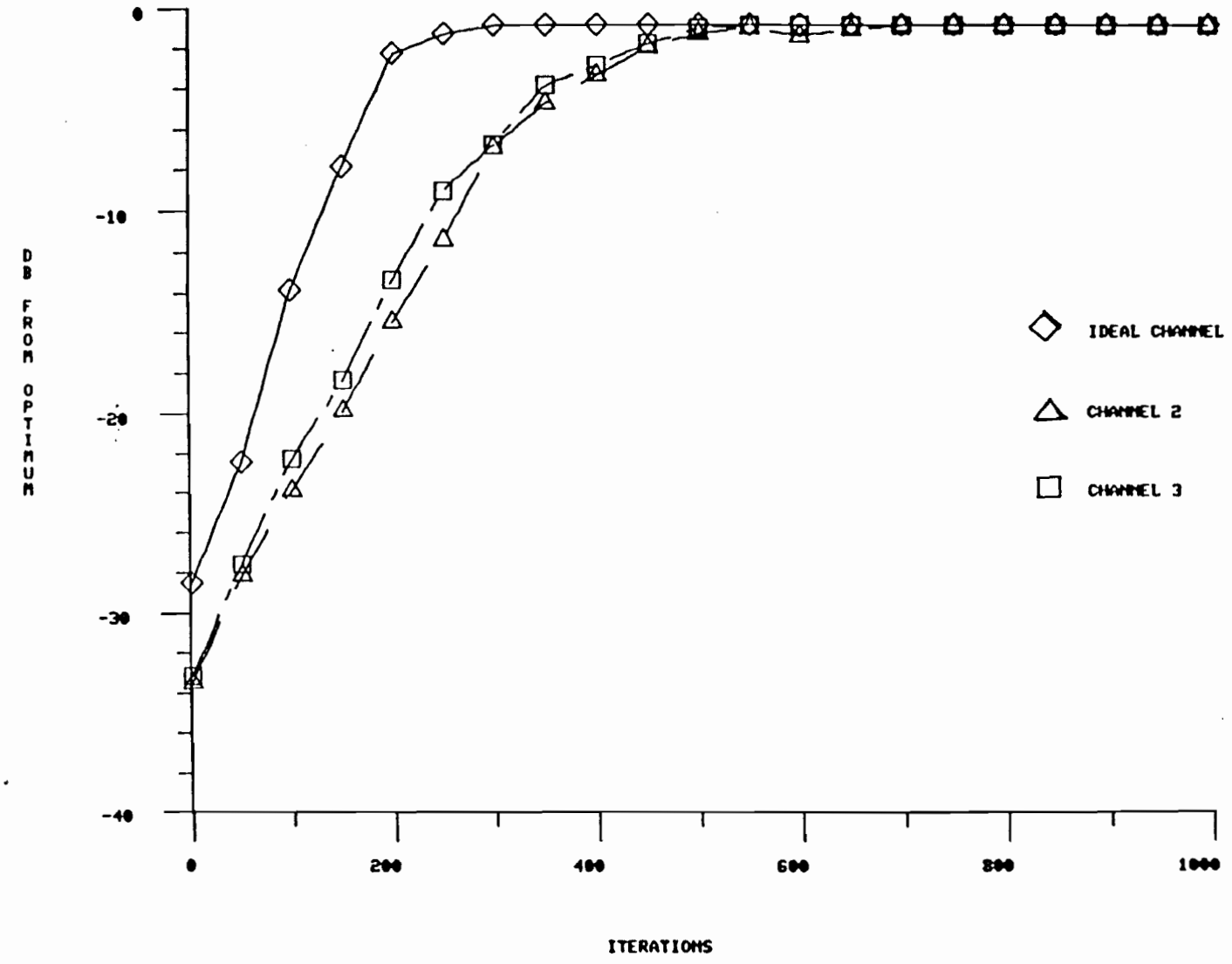


Figure 7.19 Convergence Summary for Constrained LMS Algorithm (no signal present)



7.6 Test6: Investigation of an Alternate Antenna Geometry

In an effort to understand the algorithm performance which results from a change in the antenna geometry, Tests 2 and 3 were repeated for the geometry shown in Figure 7.20. All other parameters from the tests were held fixed so that any differences may be directly attributed to the new geometry.

This rhombiod pattern was chosen somewhat at random, and is not in any way special. It was chosen simply to provide the simulation with a geometry different from the rectangular array used elsewhere.

Perhaps the most striking attribute of these plots, as compared those for the rectangular geometry, is the variation in adapted patterns across the algorithms. Recall that the rectangular array produced virtually identical antenna patterns independent of the controlling algorithm. Here we see a definite alteration of the patterns, especially in the case of the update covariance algorithm.

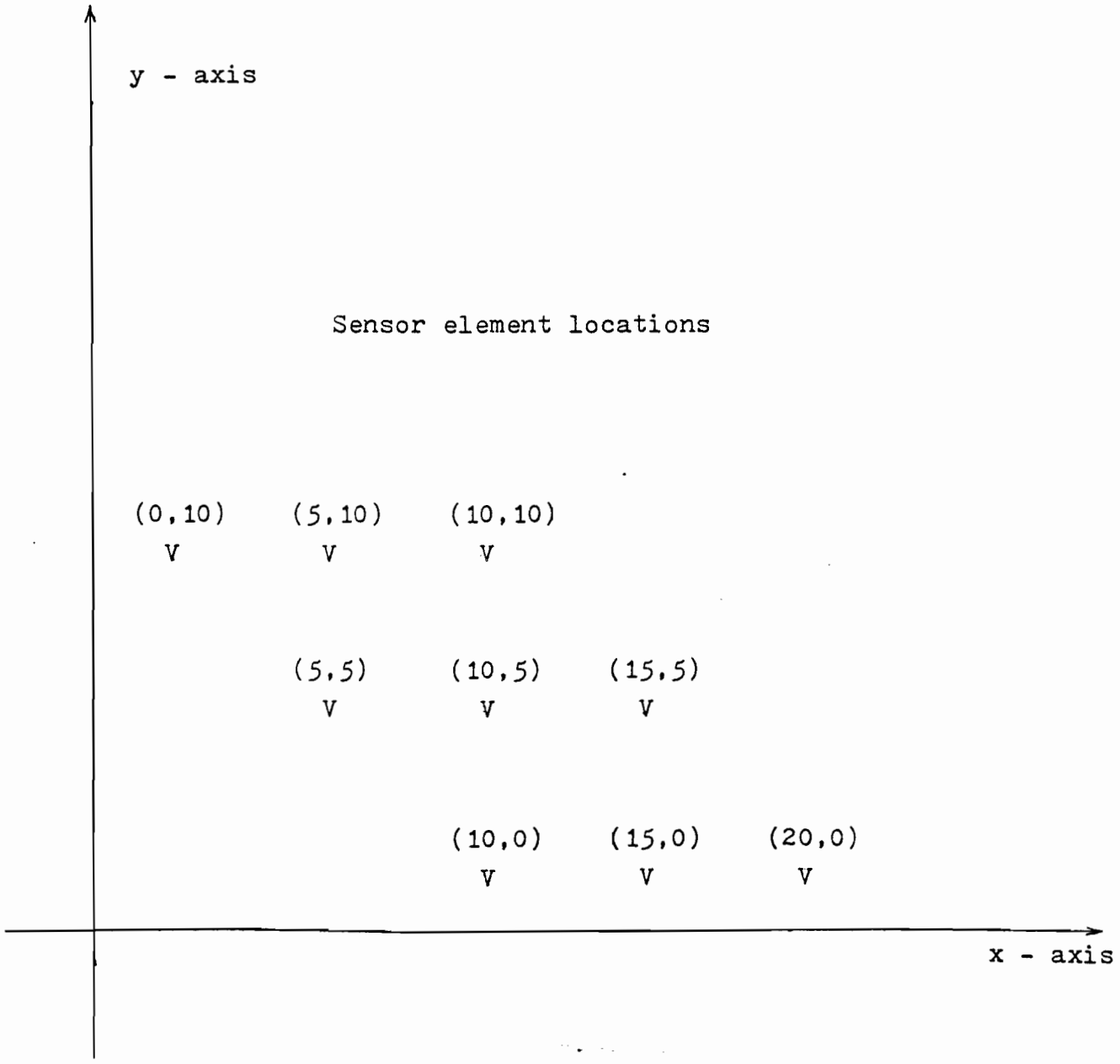


Figure 7.20 Alternate Antenna Geometry for Rombiod Geometry

Figure 7.22 Convergence Histogram of Constrained LMS Algorithm in Channel 2 for Rombiod Geometry

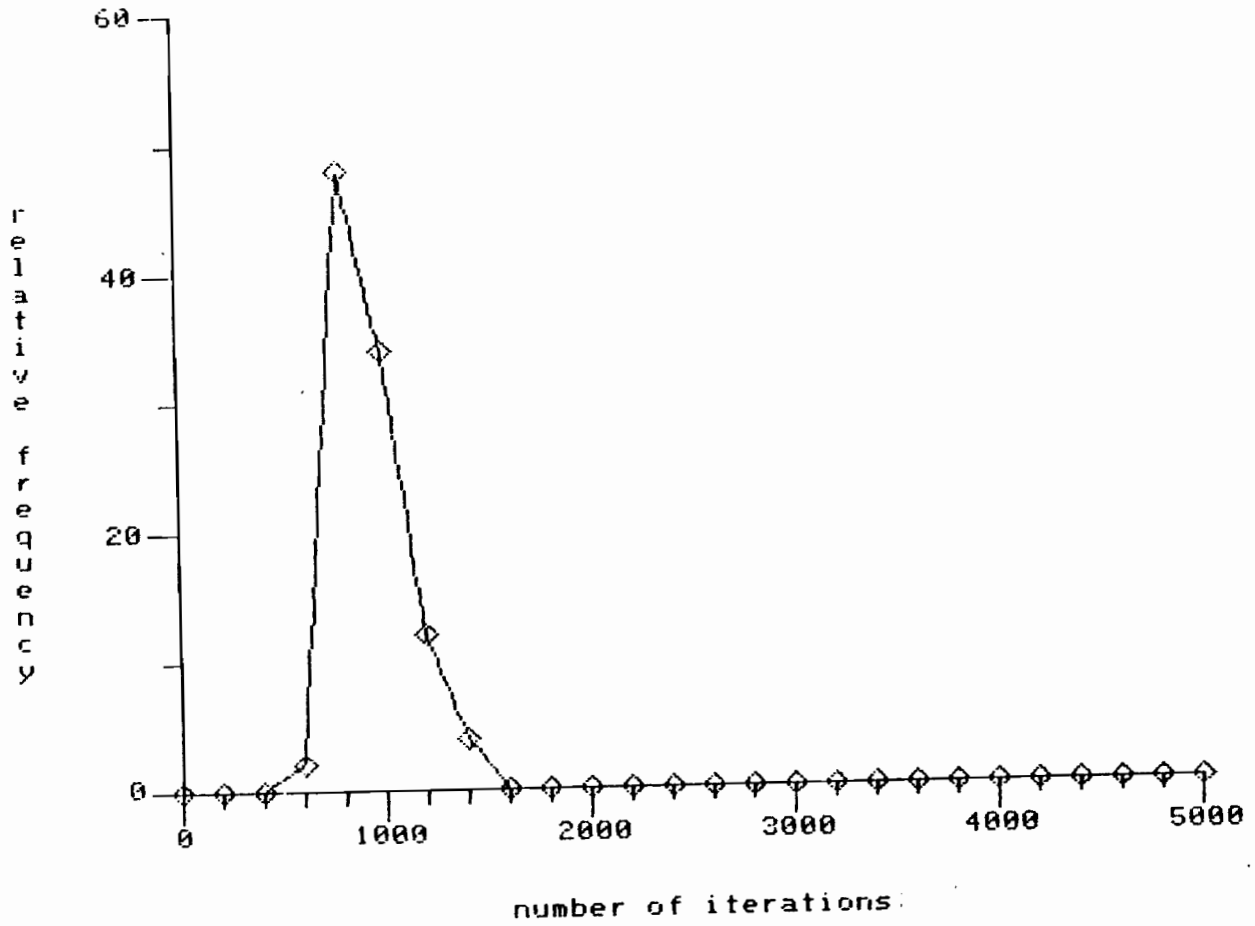


Figure 7.23 Convergence Histogram of Update Covariance Algorithm in Channel 2
for Rombiod Geometry

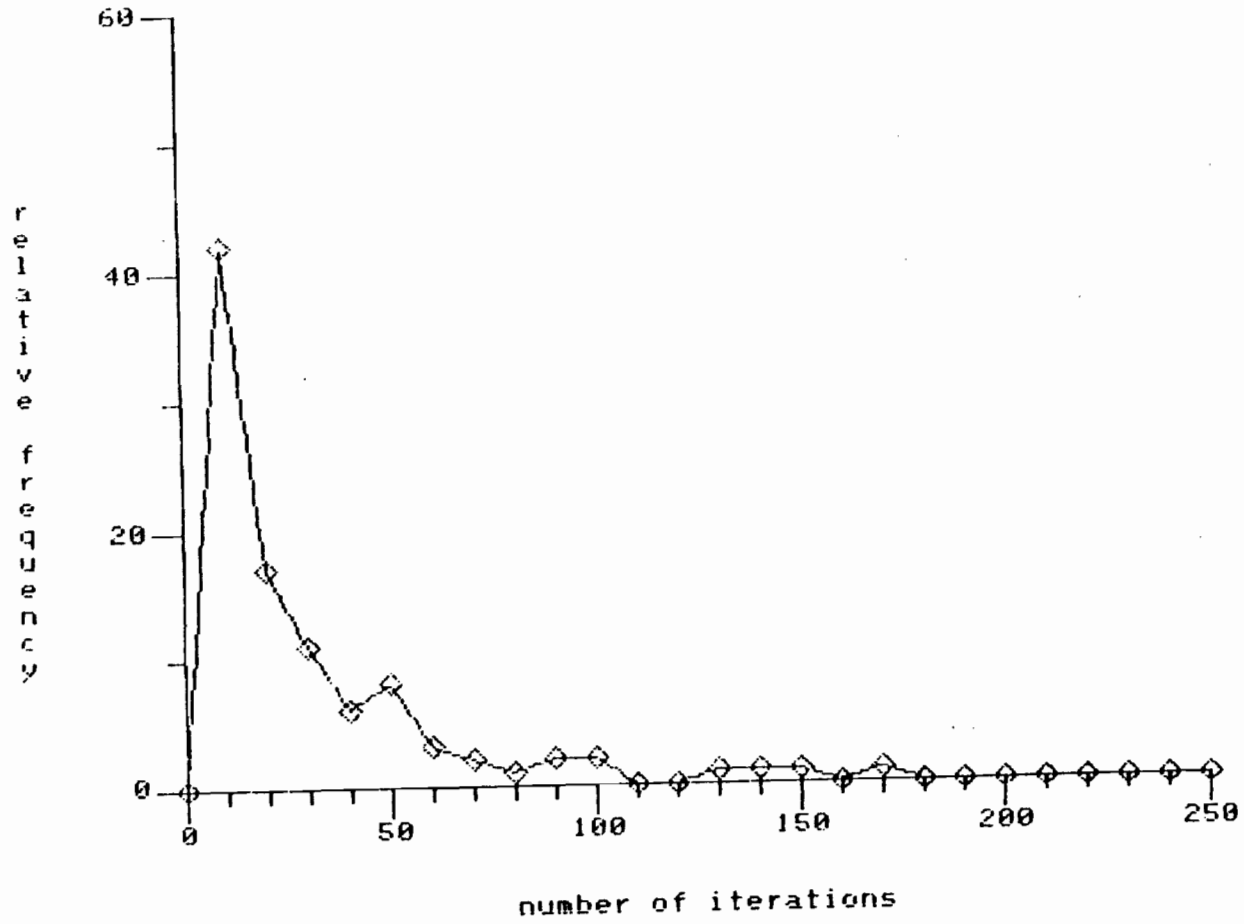


Figure 7.24 Convergence Histogram of LMS Algorithm in Channel 3
for Rombiod Geometry

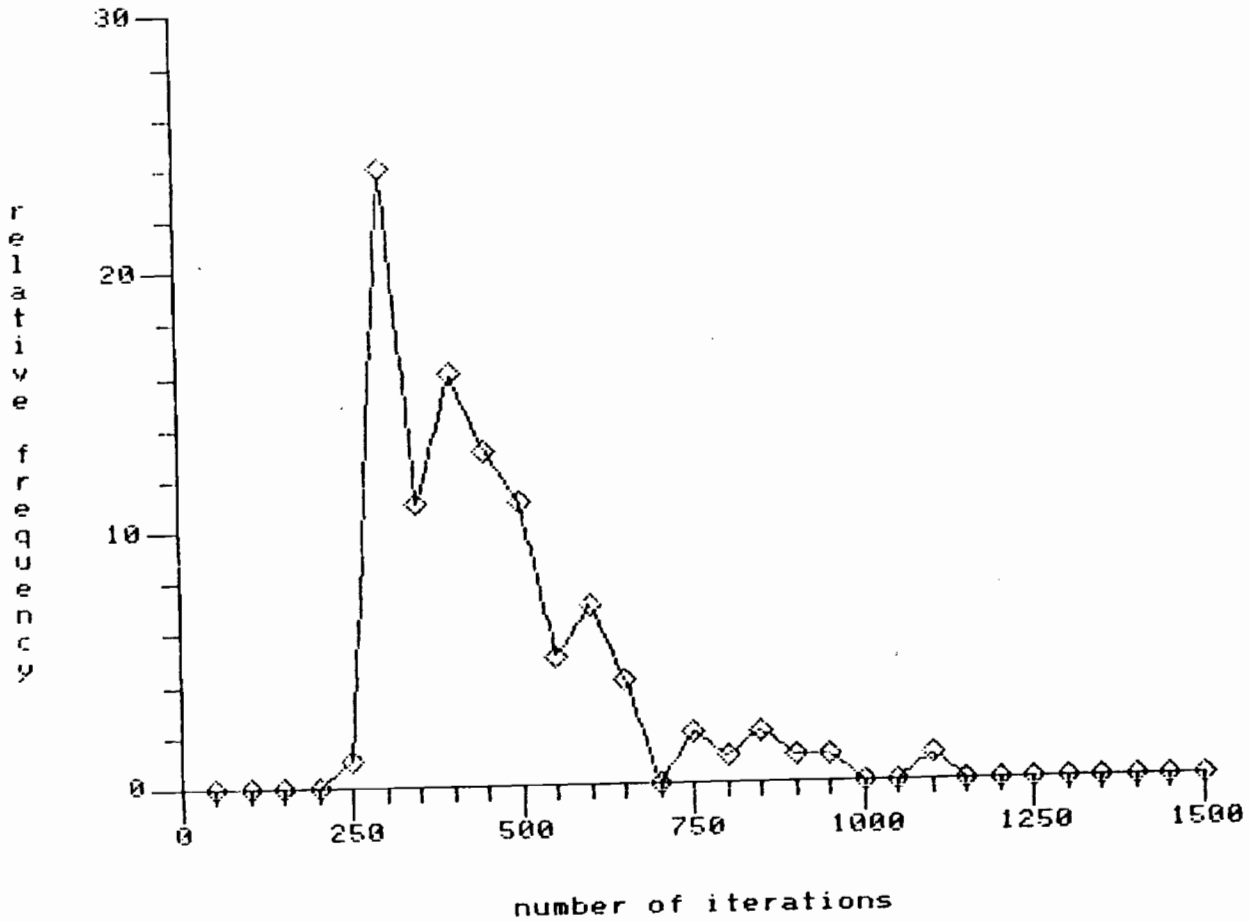


Figure 7.25 Convergence Histogram of Constrained LMS Algorithm in Channel 3
for Rombiod Geometry

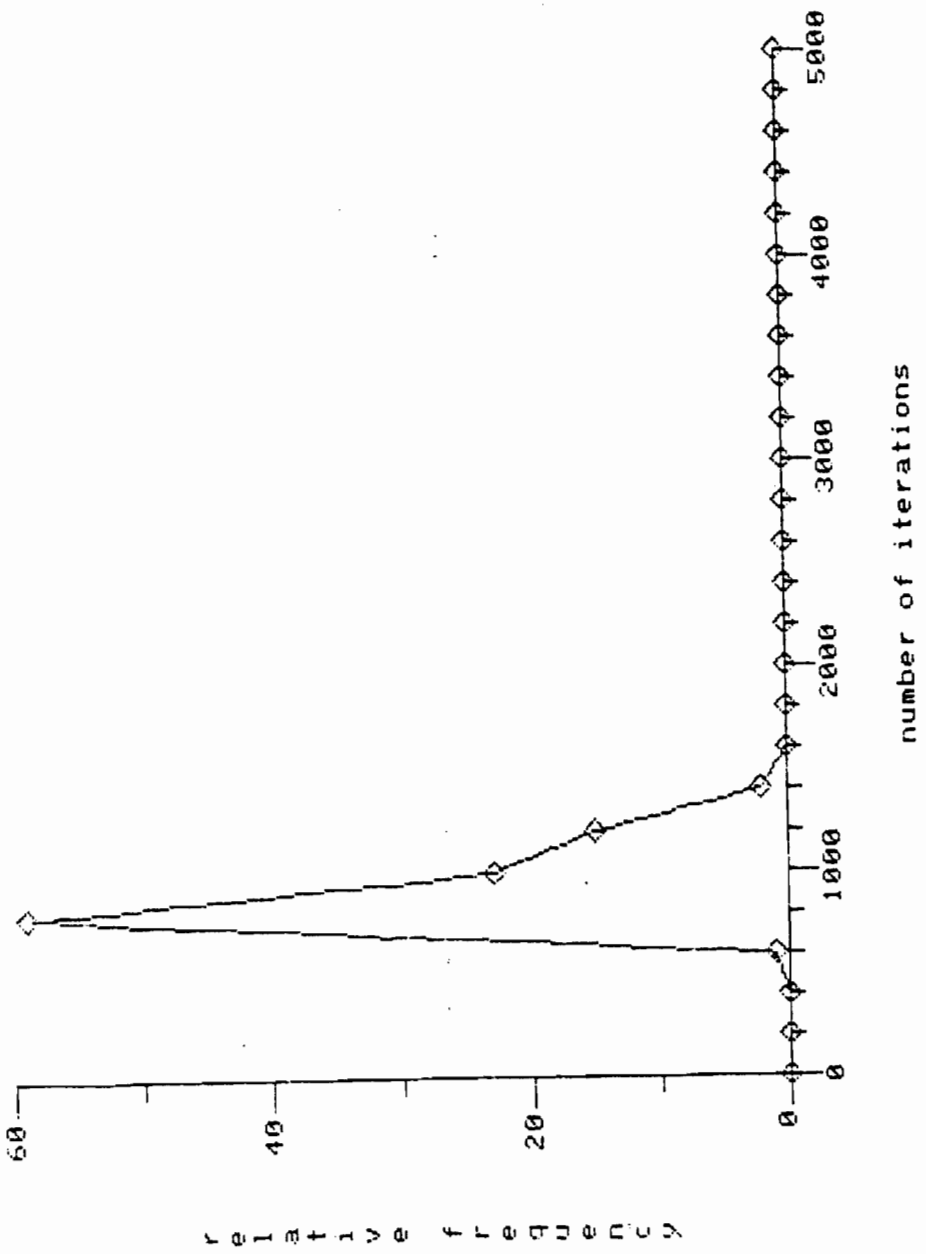
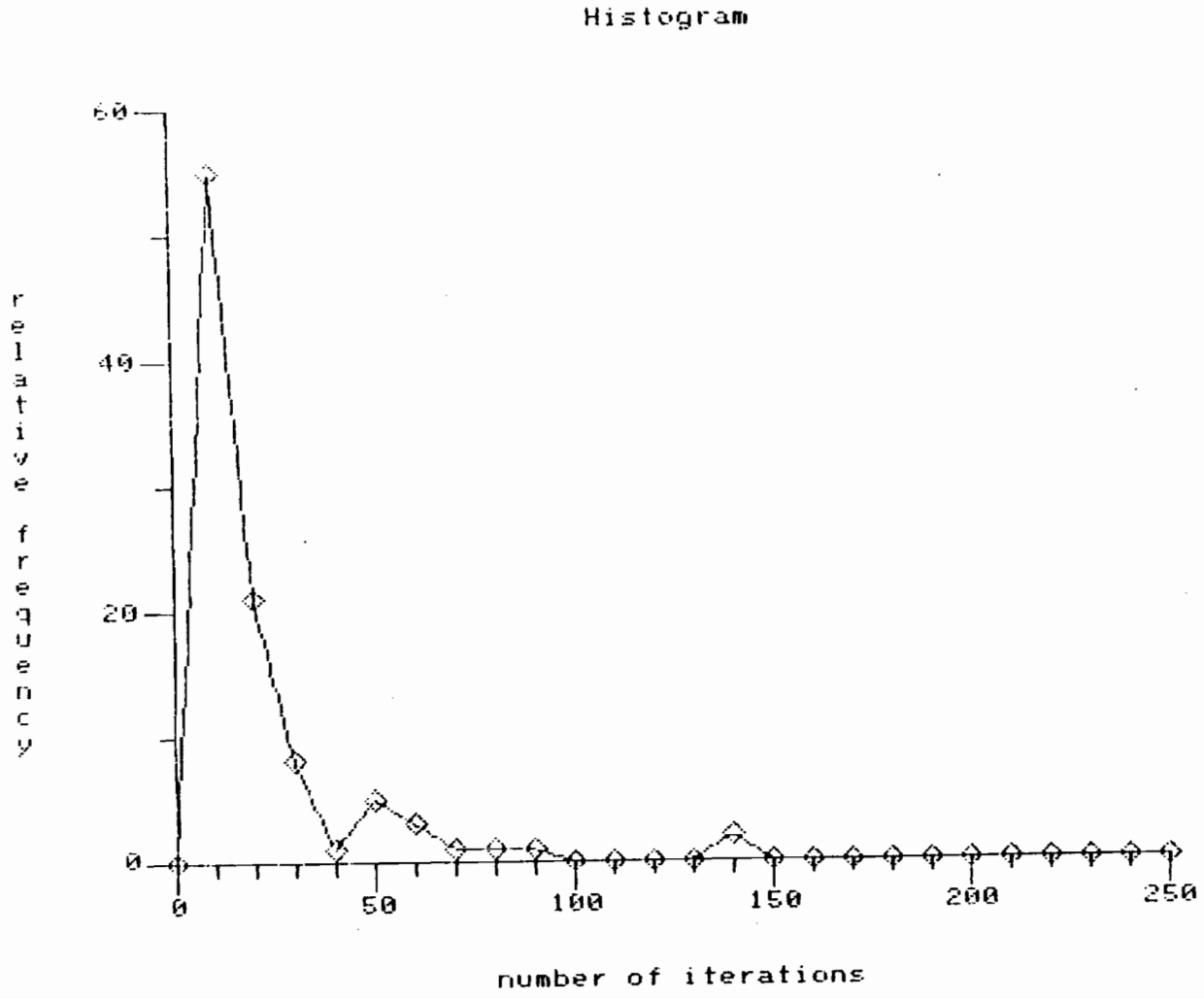


Figure 7.26 Convergence Histogram of Update Covariance Algorithm in Channel 3 for Rombiod Geometry



SUMMARY OF PLOTS FOR TEST6 - ROMBOID GEOMETRY

Figure T6.1: Unadapted antenna plot at $\phi = 90°$, $\theta = 80°$
Purpose: To indicate initial gain in direction of Jammer 1

Figure T6.2: LMS-adapted antenna plot at $\phi = 90°$, $\theta = 80°$
Figure T6.3: Constsrained LMS-adapted antenna plot at $\phi = 90°$, $\theta = 80°$
Figure T6.4: Update Covariance-adapted antenna plot at $\phi = 90°$, $\Theta = 80°$
Purpose: To demonstrate nulling of Jammes 1 (in Channel 2) by each algorithm with new antenna geometry

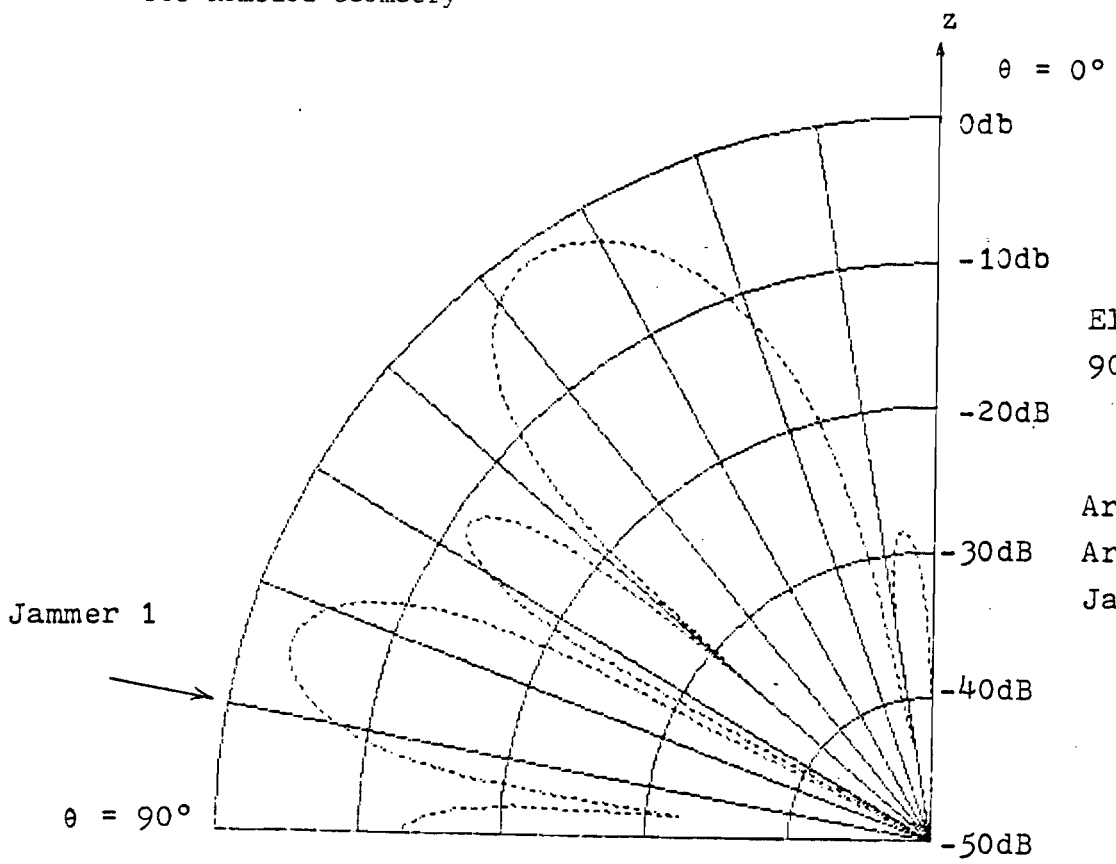
Figure T6.5: LMS-adapted antenna plot at $\phi = 90°$, $\theta = 80°$
Figure T6.6: Constrained LMS-adapted antenna plot at $\phi = 90°$, $\theta = 80°$
Figure T6.7: Update Covariance-adapted antenna plot at $\phi = 90°$, $\theta = 80°$
Purpose: To demonstrate nulling of Jammer 1 (in Channel 3) by each algorithm with new antenna geometry

Figure T6.8: Unadapted Antenna plot at $\phi = 150°$, $\theta = 75°$
Purpose: To indicate initial gain in direction of Jammer 2

Figure T6.9: LMS-adapted antenna plot at $\phi = 150°$, $\theta = 75°$
Figure T6.10: Constrained LMS-adapted antenna plot at $\phi = 150°$, $\theta = 75°$
Figure T6.11: Update Covariance-adapted antenna plot at $\phi = 150°$, $\theta = 75°$
Purpose: To demonstrate nulling of Jammer 2 (in Channel 2) by each algorithm with new antenna geometry

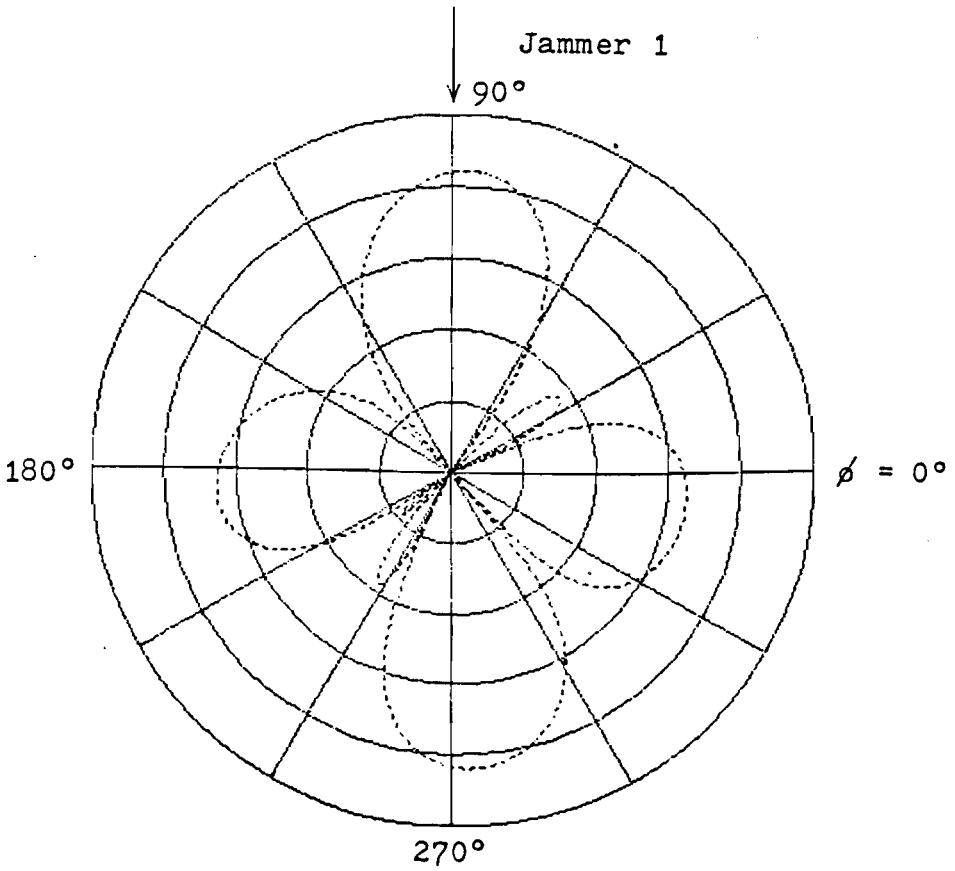
Figure T6.12: LMS-adapted antenna plot at $\phi = 150°$, $\theta = 75°$
Figure T6.13: Constrained LMS-adapted antenna plot at $\phi = 150°$, $\theta = 75°$
Figure T6.14: Update Covariance-adapted antenna plot at $\phi = 150°$, $\theta = 75°$
Purpose: To demonstrate nulling of Jammer 2 (in Channel 3) by each algorithm with new antenna geometry

For Rombiod Geometry



Elevation Plot at 90° Azimuth

Arrow Indicates Arrival Angle of Jammer 1 (θ = 80°)



Azimuthal Plot at 80° Elevation

Arrow Indicates Arrival Angle of Jammer 1 (ø = 90°)

Figure T6.1 Unadapted Antenna Pattern in Direction of Jammer 1

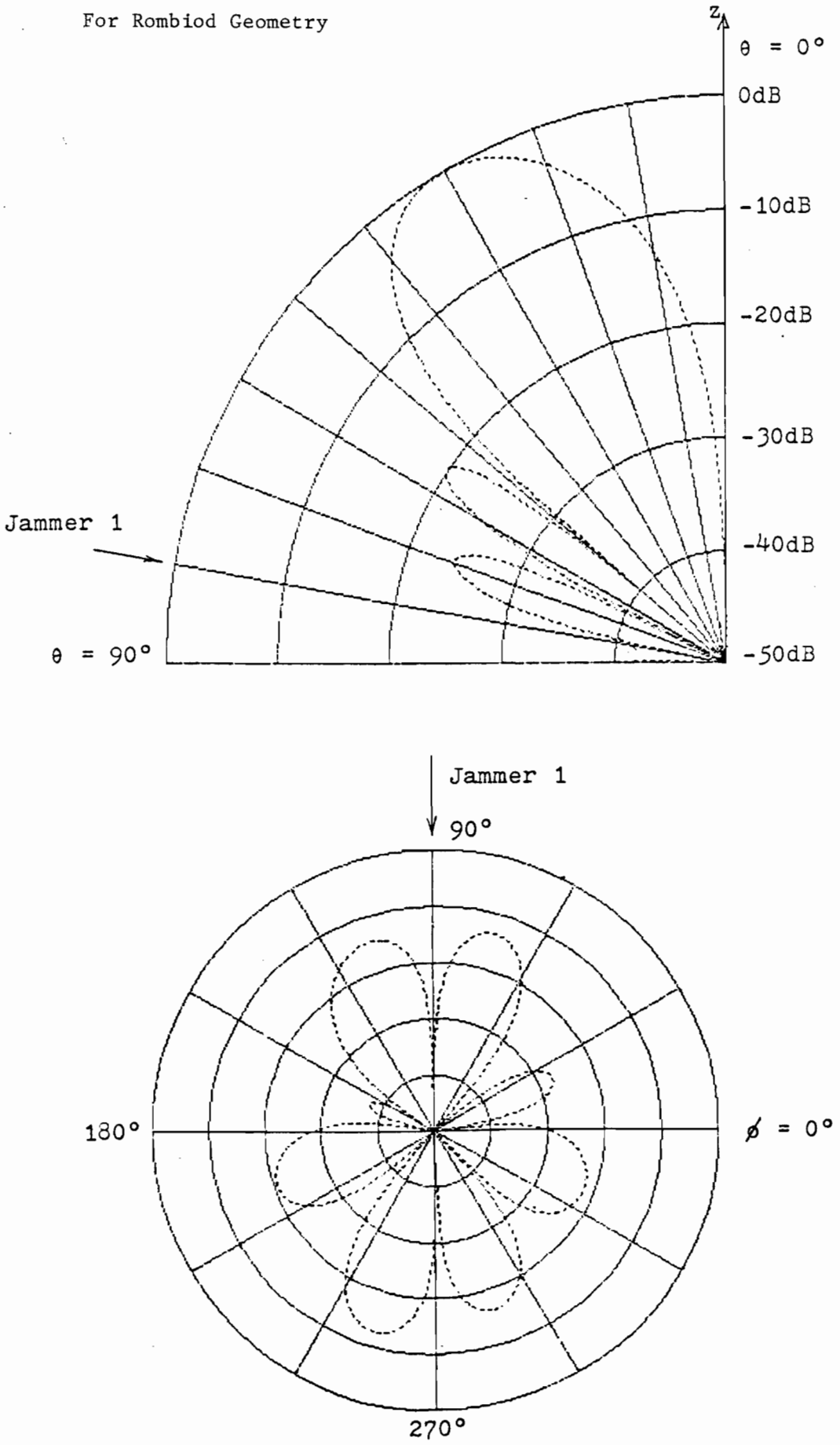


Figure T6.2 LMS-Adapted Pattern in Direction of Jammer 1

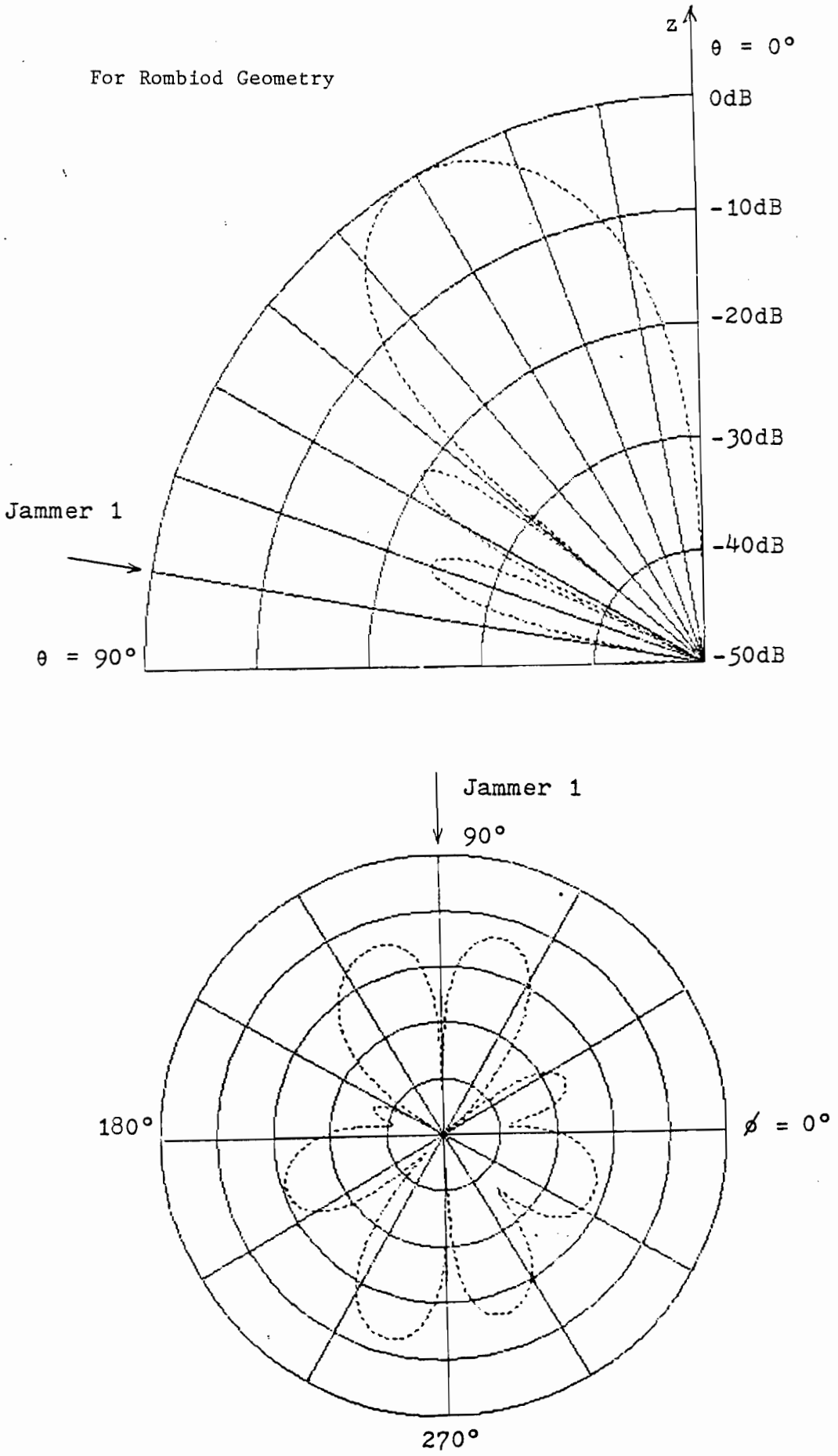


Figure T6.3 Constrained LMS-Adapted Pattern in Direction of Jammer 1

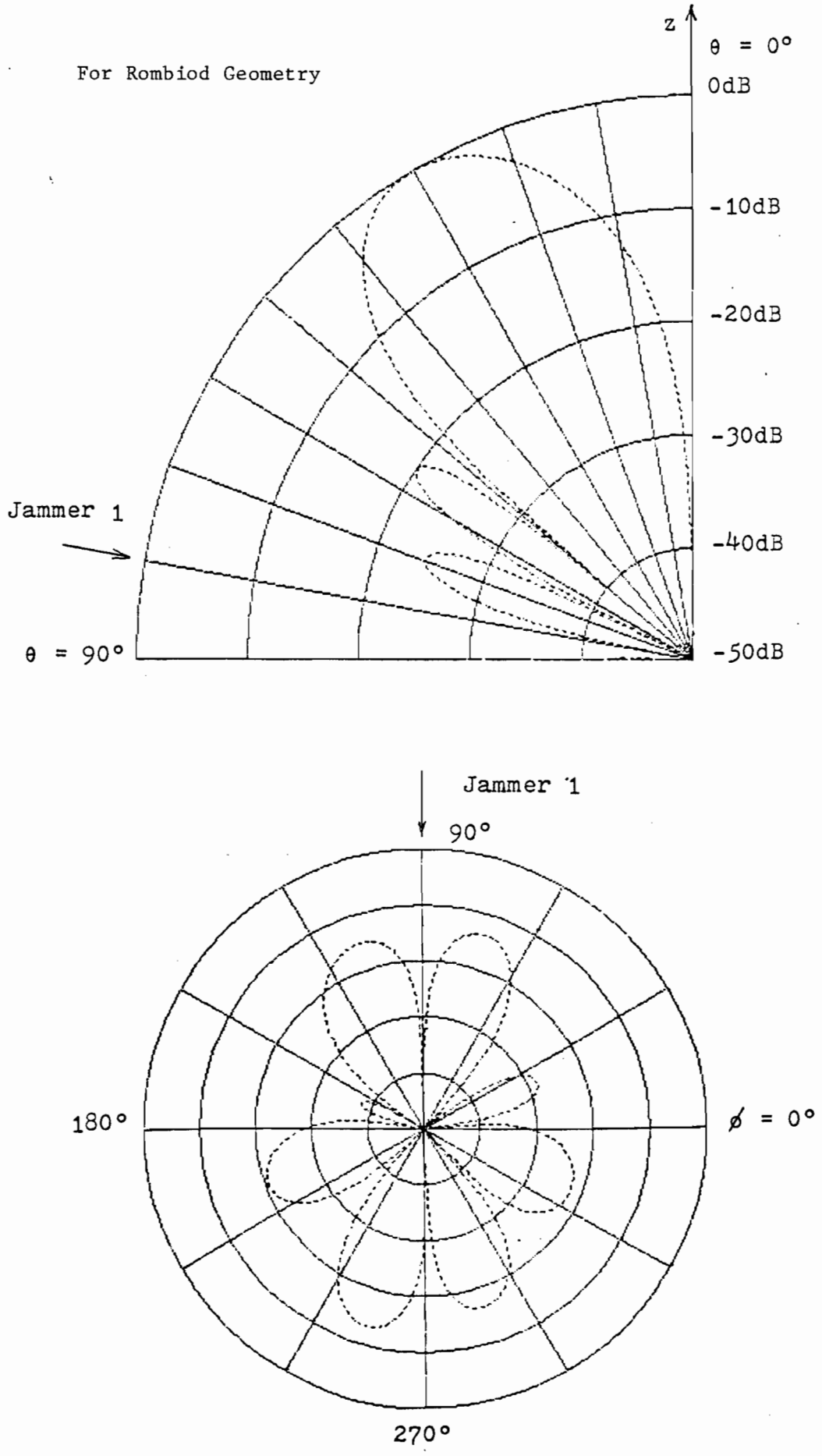


Figure T6.4 Update Covariance-Adapted Pattern in Direction of Jammer 1

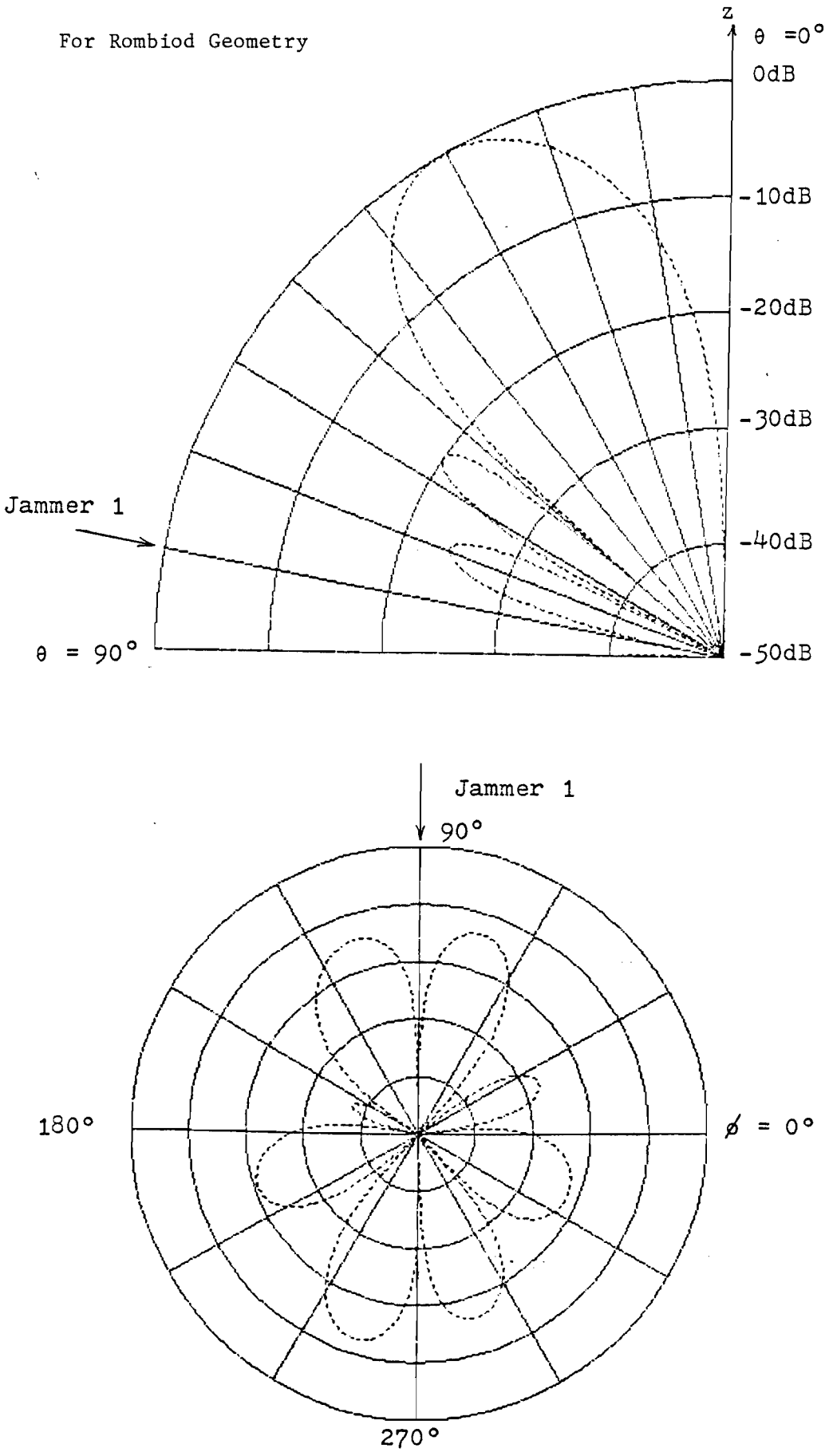


Figure T6.5 LMS-Adapted Pattern in Direction of Jammer 1

For Rombiod Geometry

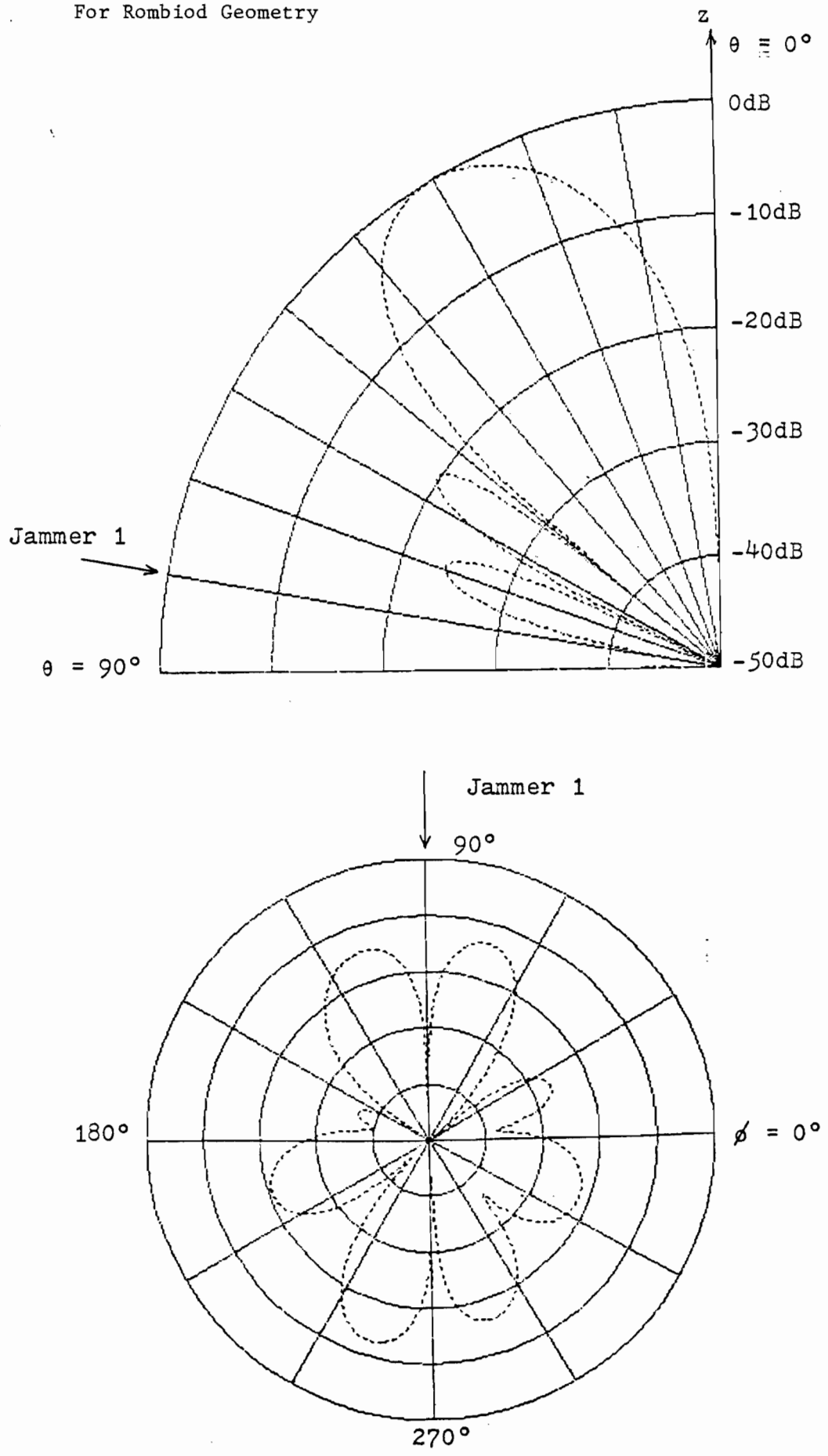


Figure T6.6 Constrained LMS-Adapted Pattern in Direction of Jammer 1

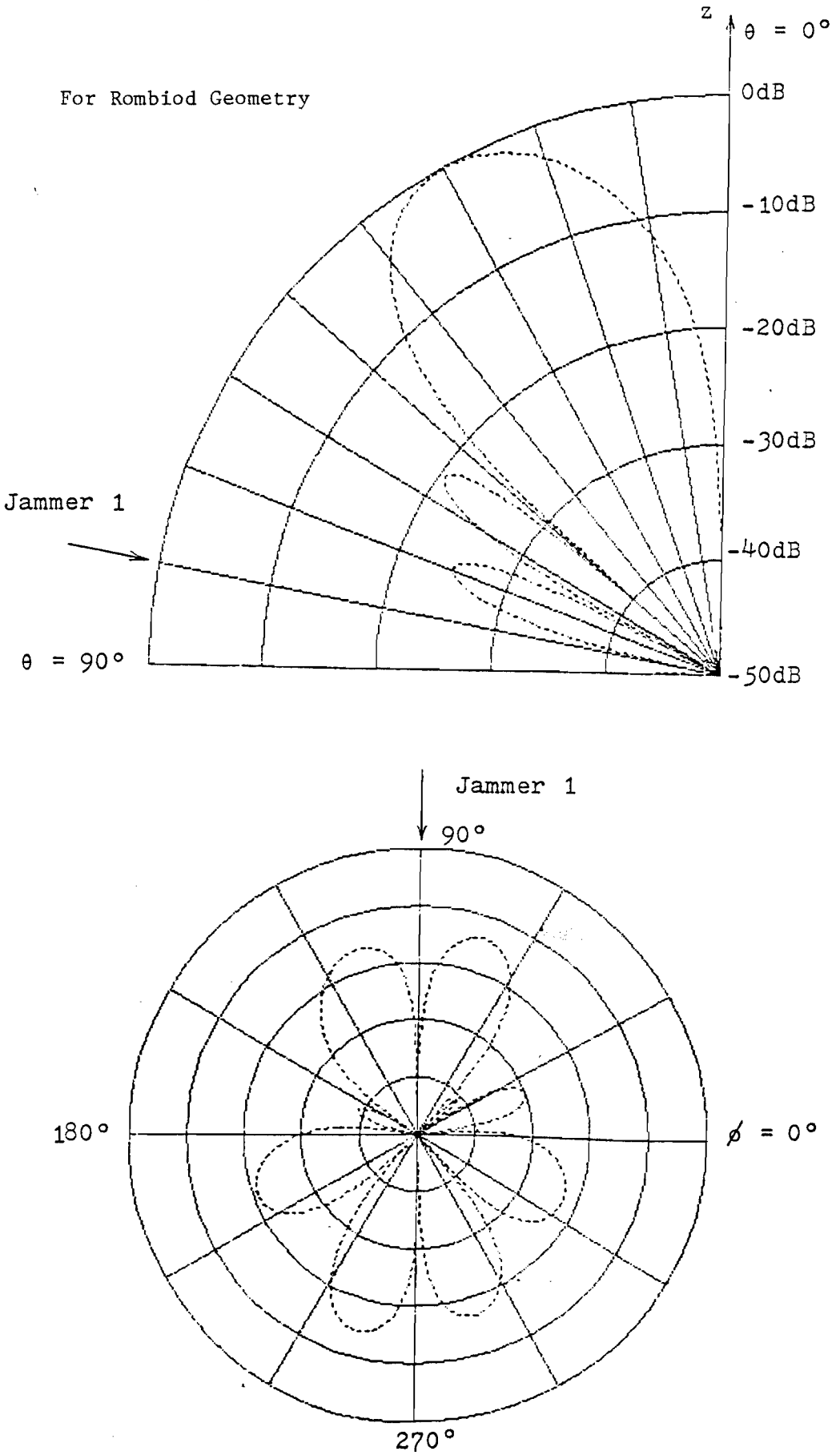
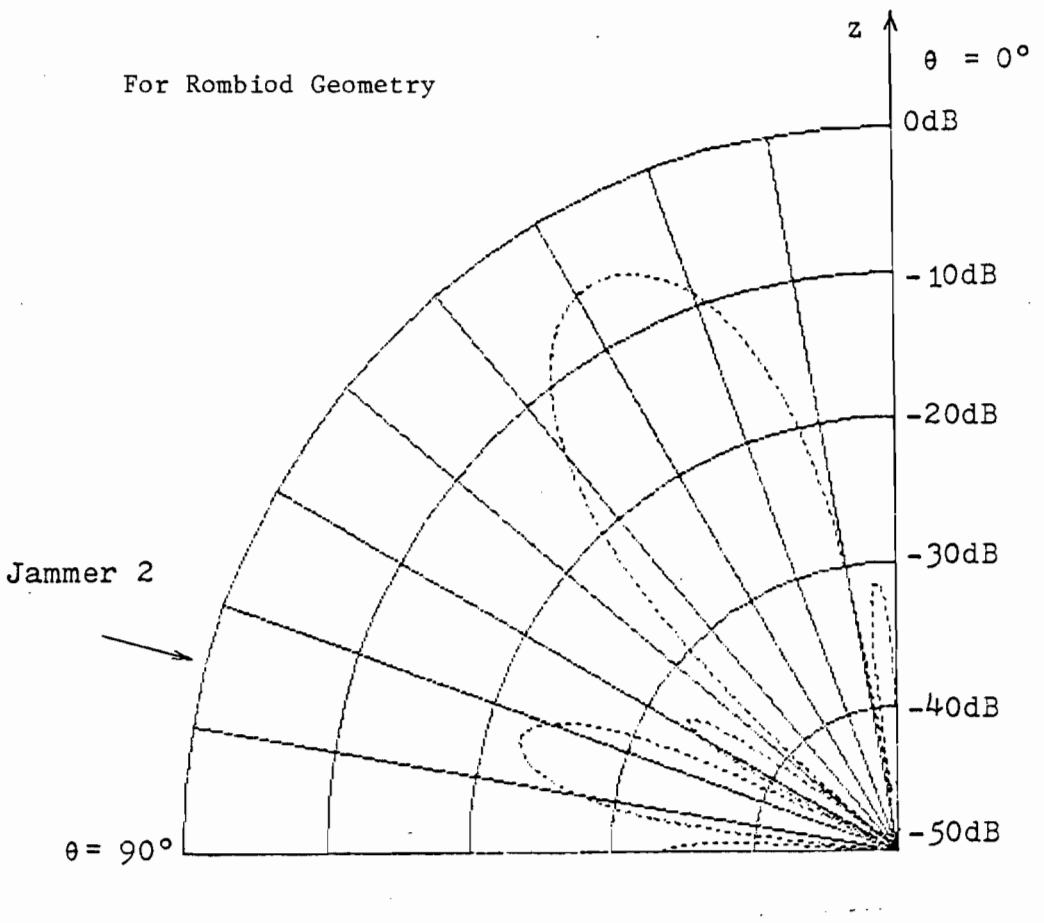
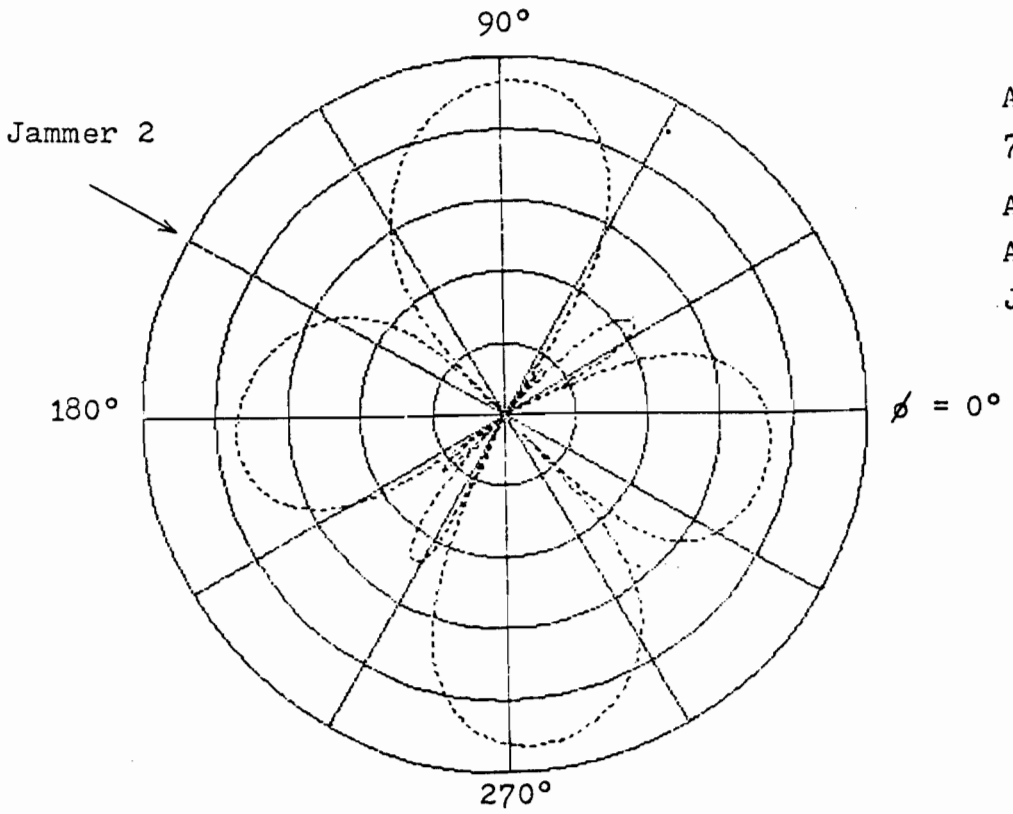


Figure T6.7 Update Covariance-Adapted Pattern in Direction of Jammer 1



Elevation Plot at 50° Azimuth

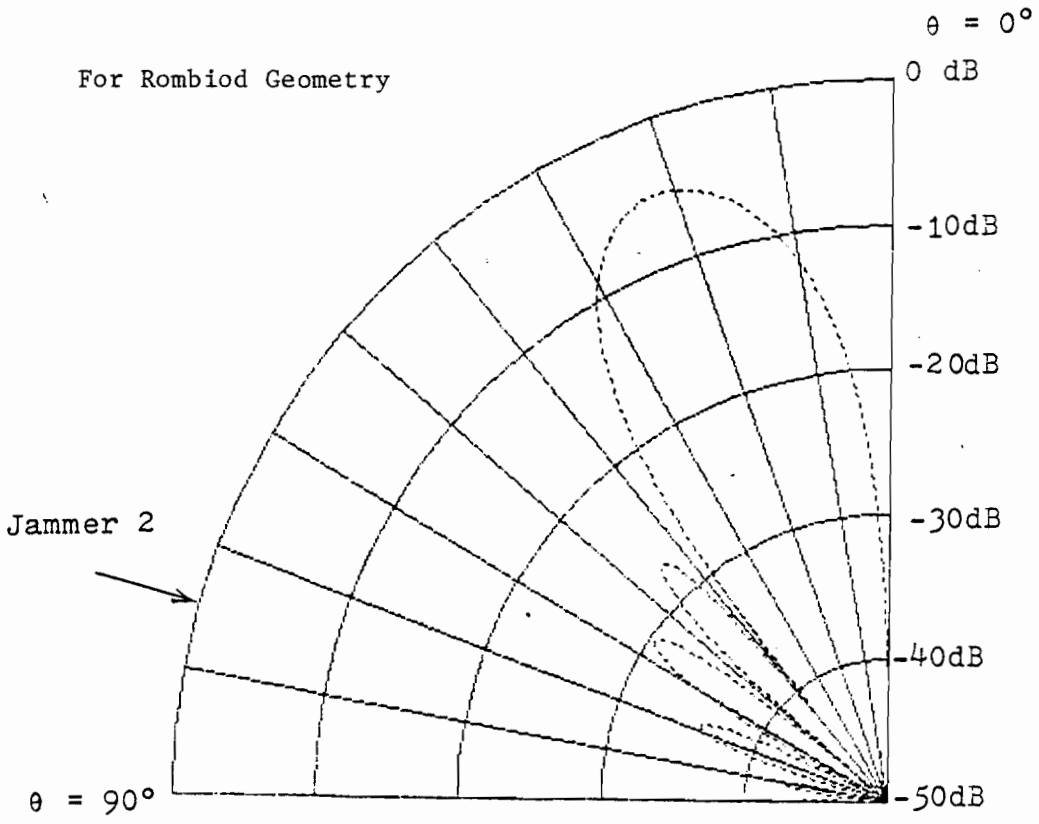
Arrow Indicates Arrival Angle of Jammer 2 (θ = 75°)



Azimuthal Plot at 75° Elevation

Arrow Indicates Arrival Angle of Jammer 2 (ϕ = 150°)

Figure T6.8 Unadapted Antenna Pattern in Direction of Jammer 2



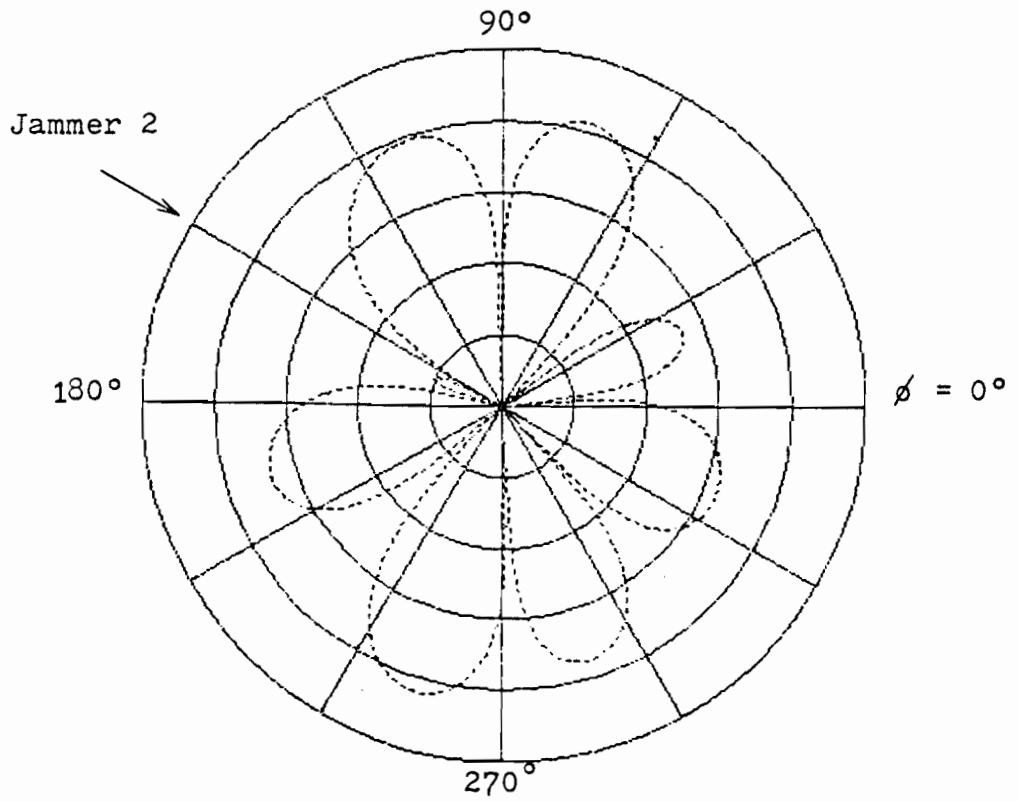
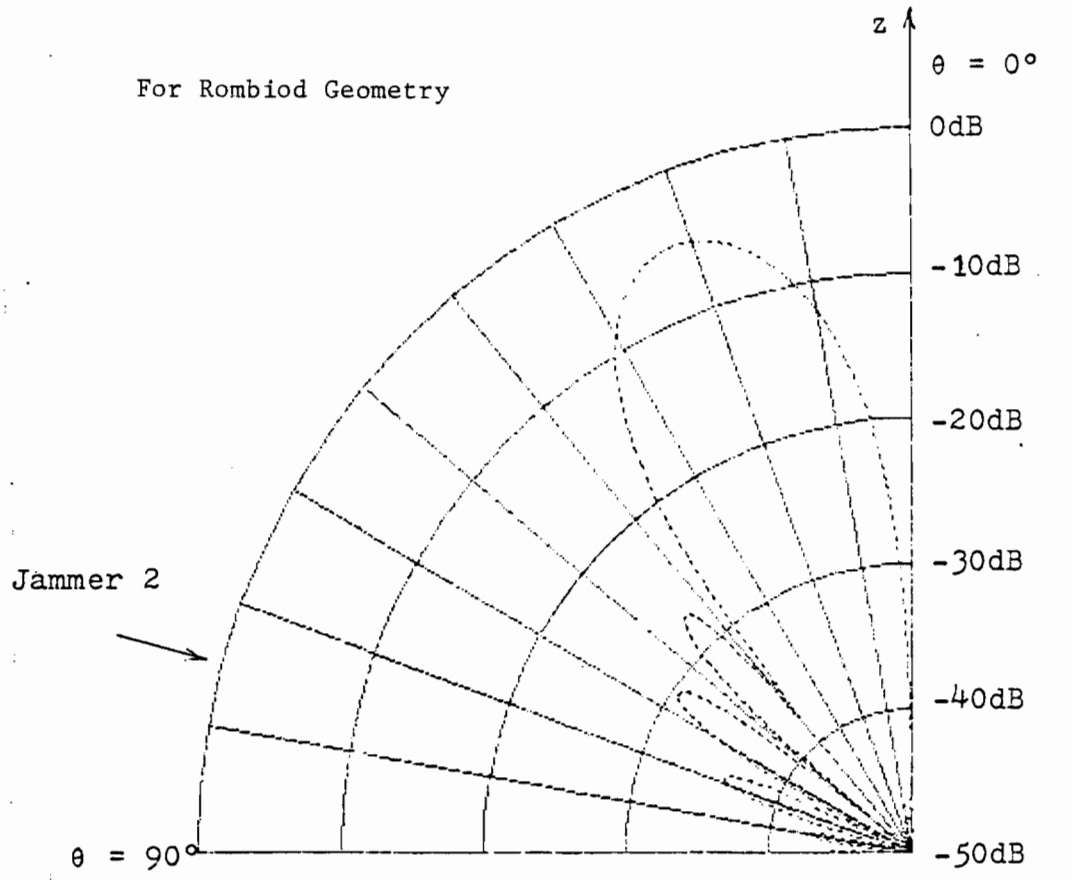


Figure T6.9 LMS-Adapted Pattern in Direction of Jammer 2



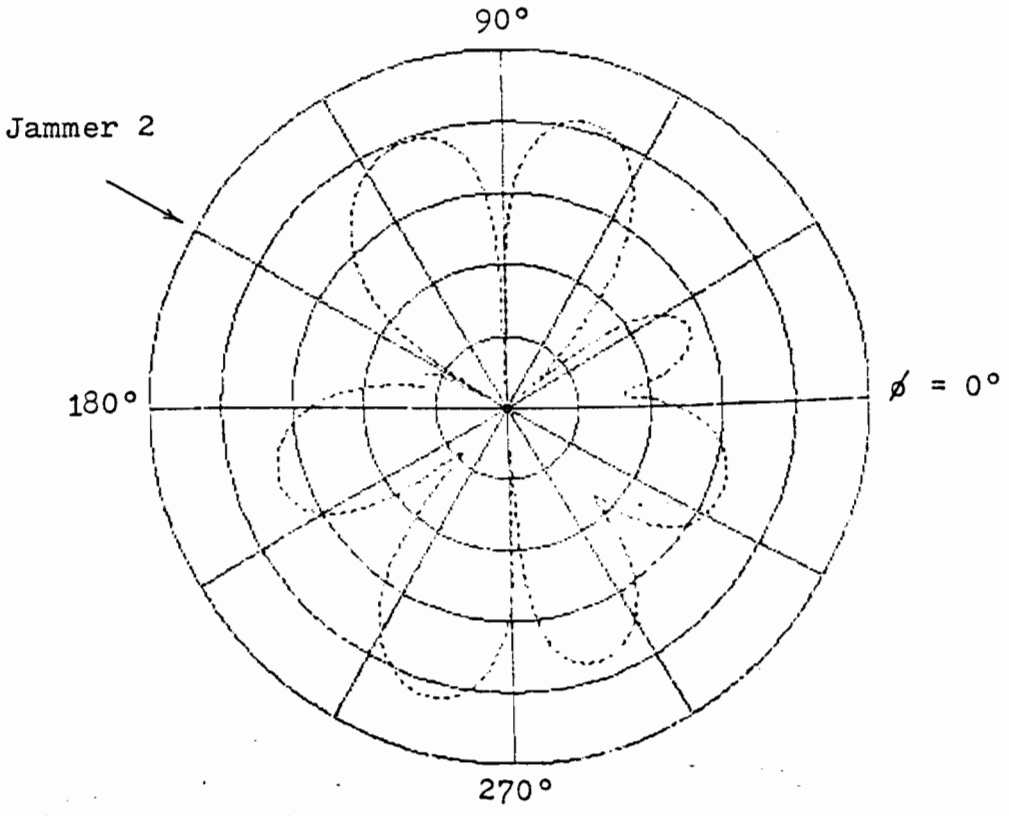
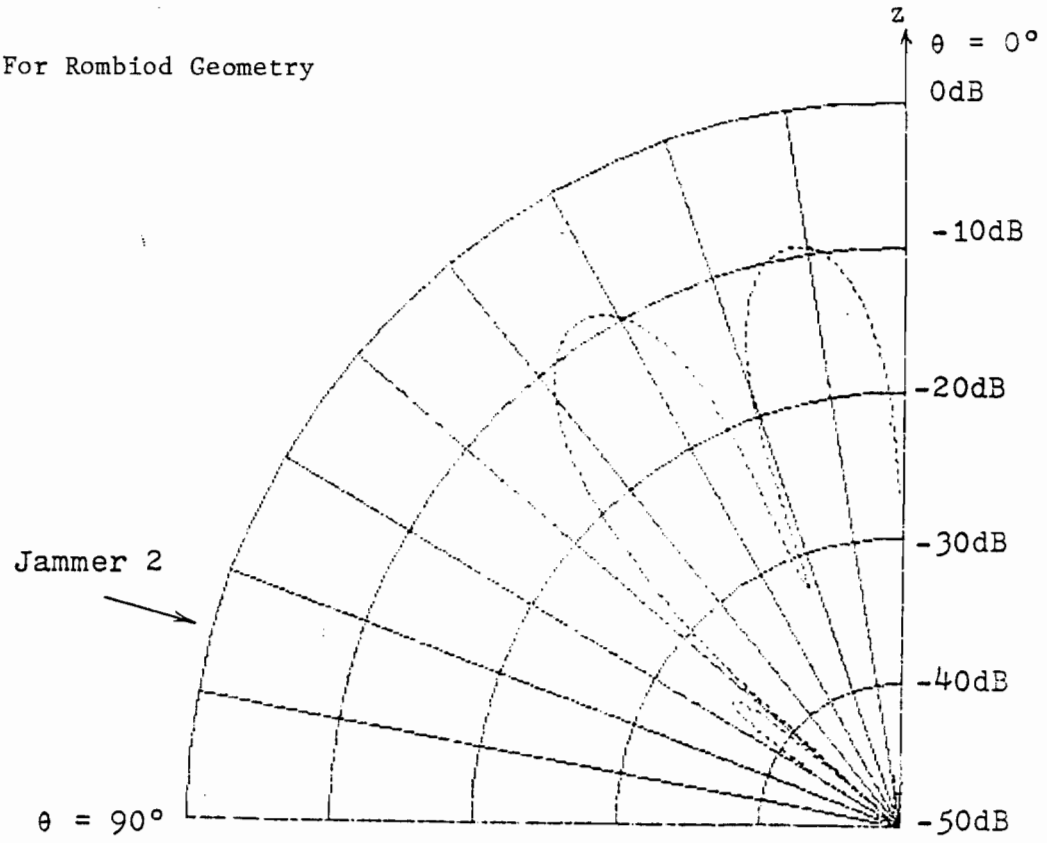


Figure T6.10 Constrained LMS-Adapted Pattern in Direction of Jammer 2



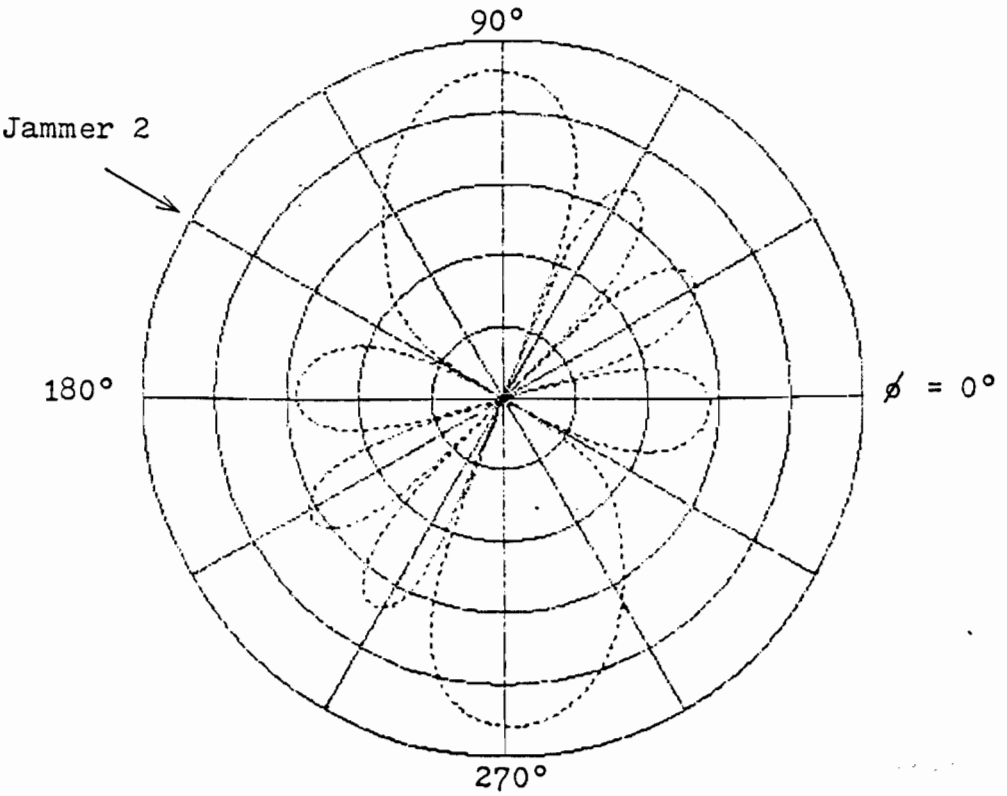


Figure T6.11 Update Covariance-Adapted Pattern in Direction of Jammer 2

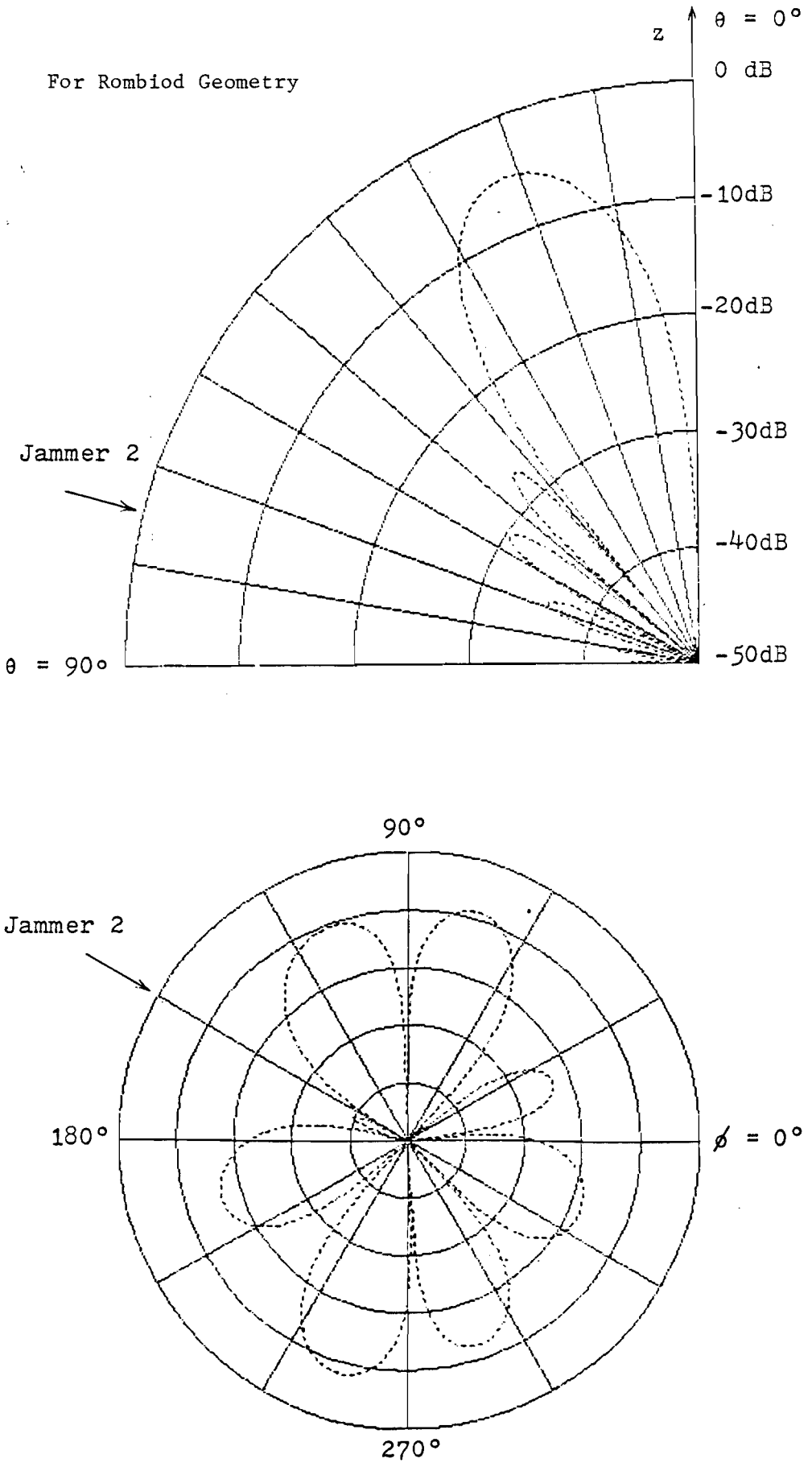


Figure T6.12 LMS-Adapted Pattern in Direction of Jammer 2

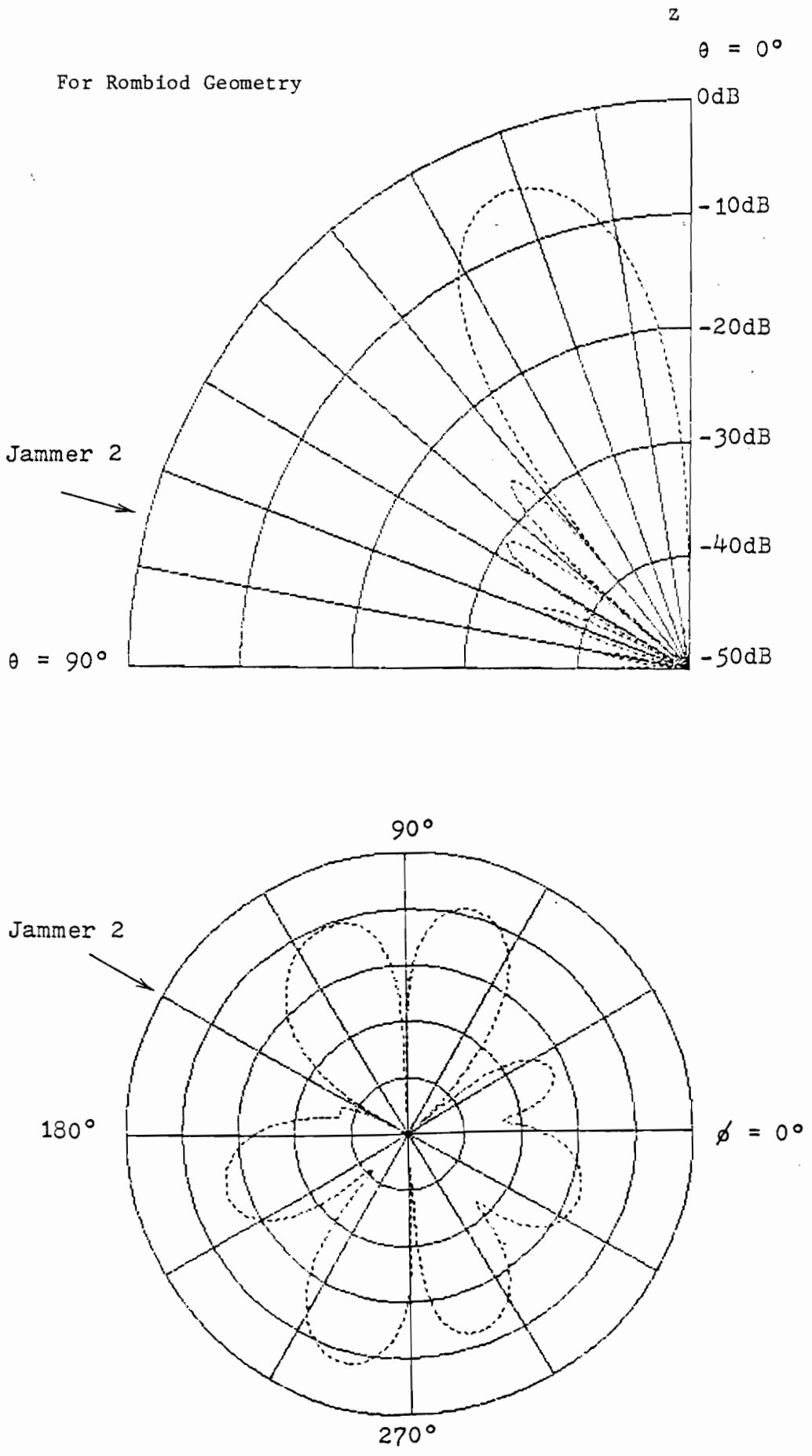


Figure T6.13 Constrained LMS-Adapted Pattern in Direction of Jammer 2

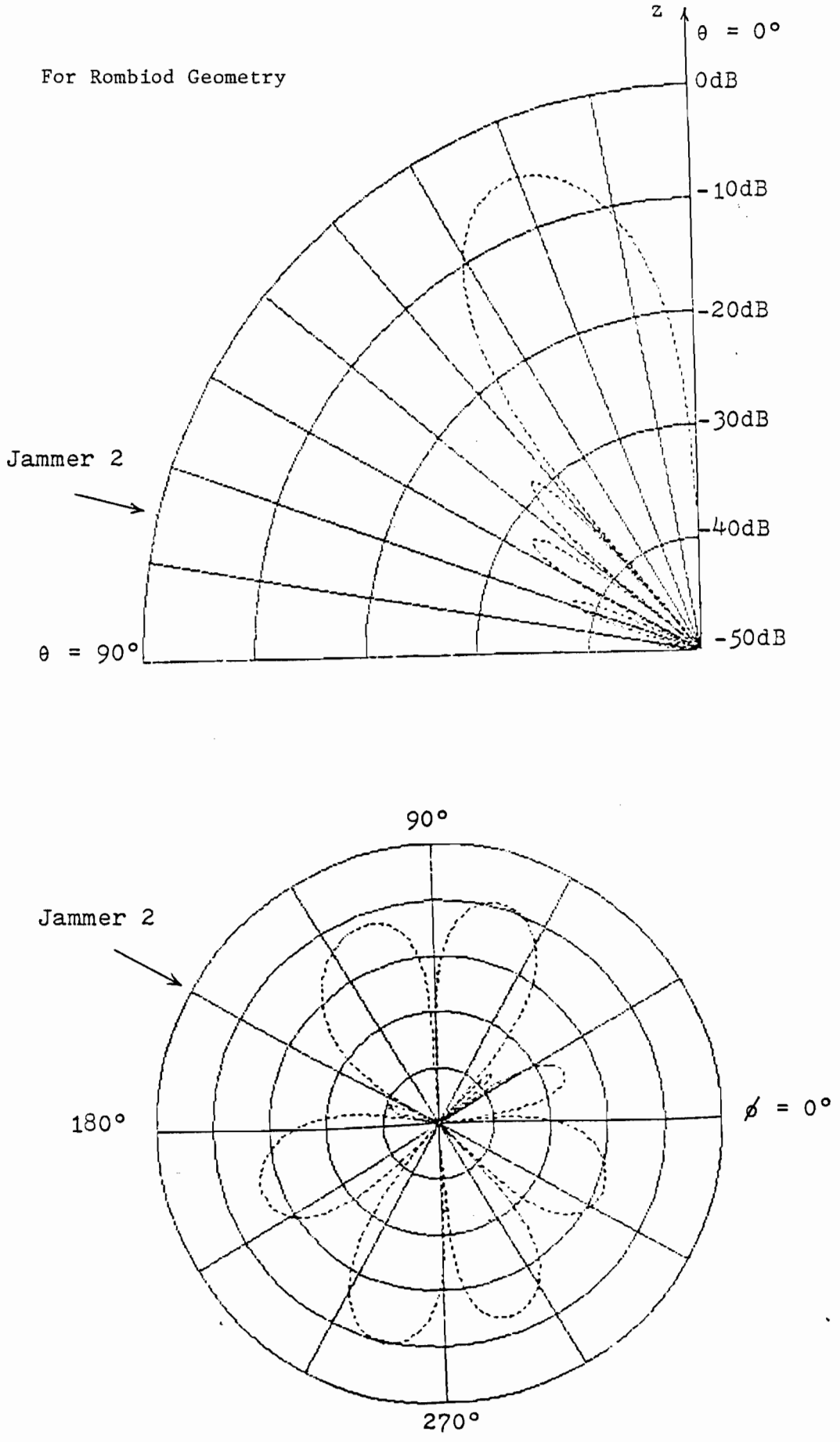


Figure T6.14 Update Covariance-Adapted Pattern in Direction of Jammer 2

7.7 Conclusions and Recommendations

In all of the tests that were conducted, the Update Covariance algorithm consistently outperformed the others in terms of required iterations for convergence. It has also been shown that this can be an extremely misleading quantity, as the algorithm may require more actual computations to converge. Therefore, although the Update Covariance algorithm will converge in fewer iterations, it will undoubtedly require significantly more time to complete each iteration. As an example,, assume that it was desired to build an array consisting of 36 elements, and that the algorithm must converge on a preamble 100 msecs. in duration. The Update Covariance algorithm would then require $5N^2$ or approximately 6500 complex computations for each iteration. If the algorithm needs 100 iterations to converge, a grand total of 650,000 complex computations must be completed in 100 msecs. This allows approximately 0.15 microseconds to perform a complex computation. So despite the comparatively few iterations required, implementing the Update Covariance algorithm may place rigorous, if not unreasonable demands on the hardware. Even if this can be attained, it seems unreasonable to implement such a system when other algorithms can offer similar convergence times with much less complicated hardware. It was shown that the LMS algorithm could converge with fewer computations. And unlike the Update covariance algorithm (which involves matrix arithmetic), the computations of the LMS algorithm can be performed simultaneously. In other words, the required 2N computations can be performed N at a time as the update of one weight is independent of the update of another. The required time for each iteration is therefore equal to the time it takes to perform 2 complex computations. This means that the actual real-time convergence for the LMS algorithm is much less than that of the Update Covariance algorithm in all cases that have been presented.

The LMS algorithm is not without its drawbacks, however, as has been mentioned. Probably the most notable is the reference requirement that has been discussed. The channel model in this simulation posed no prohibitive difficulties for the algorithm and, if it is a good representation of what goes on in the HF channel as it is believed, the reference requirement should not be a major obstacle. The reference signal generation will, however, require precise synchronization between the arrival of the transmitted preamble and the preamble used as the reference. This may be a difficult

task. If not, however, the simulation study indicates that the LMS algorithm would be an effective choice.

The simulation results also indicate that if a preamble is to be sent, the Constrained LMS algorithm, although capable of meeting the convergence requirements, would not be the best choice in light of its performance in comparison to the LMS algorithm in the presence of a signal. It also requires more computations for each iteration, although many of these computations can be performed simultaneously as well.

The performance of the Constrained LMS algorithm in the absence of the signal, however, introduces many interesting possibilities. TEST4 demonstrated that if preamble was not sent, the Constrained LMS algorithm would converge in approximately the same number of iterations as the LMS algorithm. Complicated synchornization and reference generation techniques could be avoided by simply not sending a preamble. Simply initiate the algorithm prior to any transmission. It may also be that the transmitted data had naturally occuring "breaks" in it. During either natural or deliberate breaks, the algorithm could be re-initiated, minimizing the effects of interferences. This would require some type of directional power sensing device. There are many interesting possibilities that could and should be considered. If the preamble and related synchronization devices are being generated solely for the benefit of the algorithm, one would have to consider the advantages of implementing the Constrained LMS algorithm and dispensing with the preamble. This would not seriously degrade the convergence performance, as evidenced by TEST4, and could possibly simplify the overall system.

REFERENCES

[1] B. Widrow, P. Mantey, L. Griffiths, and B Goode, "Adaptive Antenna Systems," Proceedings of the IEEE, Vol. 55, Dec. 1967.

[2] S.P. Applebaum, "Adaptive Arrays," IEEE Transactions on Antennas and Propagation, vol AP-24, no. 5, Sept. 1976.

[3] B. Widrow and J.M. McCool, " A Comparison of Adaptive Algorithms Based on the Method of Steepest Descent and Random Search," IEEE Transactions on Antennas and Propagation, vol AP-24, no. 5, Sept. 1976.

[4] L.J. Griffiths, "A Simple Algorithm for Real-time Processing in Antenna Arrays," Proceedings of the IEEE, vol. 57, Oct. 1969.

[5] O.L. Frost III, "An Algorithm for Linearly Constrained Adaptive Array Processors," Proceedings of the IEEE, vol. 60, no. 8, Aug. 1972.

[6] K. Takao, M. Fujita, and T. Nishi, "An Adaptive Antenna Array Under Directional Constraint," IEEE Transactions on Antennas and Propagation, vol AP-24, no. 5, Sept 1976.

[7] R. A. Monzingo and T. W. Miller, "Introduction to Adaptive Arrays," John Wiley and Sons, New York, 1980.

[8] L.E. Brennan, J.D. Mallet, I.S. Reed,"Adaptive Arrays in Airborne MTI Radar," IEEE Transactions on Antennas and Propagation, vol AP-24, no. 5, Sept. 1976.

[9] M. Brobston, "Simulation of Recursive Processors for Adaptive Antenna Arrays," M.S. project report, Dept. of Electrical and Computer Engineering, Univ. of Kansas, 1981.

[10] John G. Proakis, "Digital Communication," McGraw-Hill, New York, 1983.

[11] R. T. Compton Jr., "An Adaptive Array in a Spread Spectrum Communication System," Proceedings of the IEEE, vol. 66, no. 3, March 1978.

[12] R. L. Reigler and R. T. Compton Jr.," An Adaptive Array for Interference Rejection," Proceedings of the IEEE, vol. 61, June 1973.

[13] P. Snow, " An Antenna Simulation for a Spread Spectrum System," M.S. project report, Dept. of Electrical and Computer Engineering, 1984.

APPENDIX A

COMPLEX LOWPASS EQUIVALENT REPRESENTATION

Digital information signals are often transmitted using some type of carrier modulation. Signals which have a bandwidth that is much smaller than the carrier frequency are known as narrowband bandpass signals. For convenience, it is desirable to reduce the bandpass signals to equivalent lowpass signals. The carrier componenet can be dispensed with because it carries no information.

A real-valued signal s(t) with a frequency concentrated in a narrow band of frequencies about the carrier frequency, f_c, can be expressed as

$$s(t) = a(t) \cos[2\pi f_c t + \Theta(t)]$$

where

$a(t)$ = amplitude of $s(t)$
$\Theta(t)$ = phase of $s(t)$

By expanding the cosine function in the above expression, a second representation is obtained. This is written as

$$s(t) = a(t) \cos\Theta(t) \cos 2\pi f_c t - a(t) \sin\Theta(t) \sin 2\pi f_c t$$

$$= I(t) \cos 2\pi f_c t - Q(t) \sin 2\pi f_c t$$

where

$I(t) = a(t) \cos\Theta(t)$
$Q(t) = a(t) \sin\Theta(t)$

The frequency content of I(t) and Q(t) is concentrated at low frequencies. These are lowpass signals.

Finally, define the complex envelope u(t) as

$$u(t) = I(t) + jQ(t)$$

so that

$$s(t) = re[u(t)\, e^{j2\pi f_c t}]$$

Therefore, a real bandpass signal s(t) is completely specified by its complex envelope, u(t), if its carrier frequency is known.

APPENDIX B

USER MANUAL FOR HF ADAPTIVE ANTENNA ARRAY EVALUATION FACILITY

The following is a brief description of the parameters and procedures needed to operate the simulation. The operations are divided into an input and an output phase, and will be described separately.

B1.0 Input Procedures

As shown in Figure B-1, the input phase is initiated with the command @INPUT. This command file asks the user to input the name of the experiment to be performed, and in this case we have chosen TEST1. The result of this is the creation of a new subdirectory which is given the name EXPERIMENT TEST1. This subdirectory, then, will provide a location for the simulation run, and will contain all important output files produced. This, however, is completely transparent to the user.

Again referring to Figure B-1, we see that as soon as the user determines the name of the experiment, he or she is immediately introduced to the first menu of the actual input program. It is this routine which creates the data file which is subsequently read by the simulation mainline. Before explaining the actual variables appearing in this and the remaining menus, we first consider the methods by which variables are input and new menus acquired.

Concentrating again on the first menu, we see that 4 variables appear (numbered 1 through 4). Assume we wish to change the number of desired convergences from the default value of 5 to a value of 20. As shown in the Figure, this is accomplished simply by typing the variable index (4) followed by a space and the new value (20). The program then returns the instructions < NEXT ENTRY PLEASE (0 TO REVIEW PAGE, -1 TO LEAVE PAGE > and as shown, 0 redisplays the menu with the new value inserted. A -1 at this point (instead of a 0) simply moves to the next menu without displaying the change. Of course, if it is desired to change nothing, it is possible to traverse the entire program simply by typing a -1 after each menu. The data file, then, will simply contain the default values. It should be noted that this program continuously updates the default file. That is to say that if a variable is

changed on a run through the input program, it becomes the new default value for the next run.

Now that we may move through the input program, the variables themselves will be explained. It will be expedient to cover the variables one menu at a time since, in many cases, the variables within a menu are closely related. For the following discussion, refer to Figures B-1 through B-10.

B1.1 Menu 1: Simulation Parameters

These parameters are general and serve to set up the simulation at its most basic level. Here, many of the variables are self-explanatory.

1. Number of samples per bit. Sets up sample rate.
2. Signal to Noise Ratio (dB). This is the thermal S/N ratio which is used to simulate random electrical noise.
3. Simulation Bit Rate. Used to determine the Nyquist bandwidth of the information signal.
4. Number of simulation convergences desired. To arrive at a statistical evaluation of the simulation, the system is forced to converge a number of times. This variable simply determines the number of loops (value = 100).

B1.2 Menu 2: Adaptive Algorithm Options

Here the user is asked to choose the type of algorithm which will control the simulation.

1. Least Mean Squared Algorithm
2. Constrained Lease Mean Squared Algorithm
3. Update Covariance Algorithm

B1.3 Menu 3: Antenna Array Parameters

This menu gives the user several options to construct the antenna array to be used in the simulation. The array elements are dipole antennas (whip above ground).

1. Number of antenna elements (maximum = 9 elements)
2. Number of incoming signals. This parameter is simply the total number of impinging signals, including both the desired signal as well as one or two jammers.
3. Dipole equivalent element length (meters). The length of an equivalent dipole has been assumed to be twice the length of the actual whip antennas.
4. Carrier frequency of friendly communicator (kHz). This is simply the frequency, in kilohertz, of the desired signal. The frequency is used to determine the receiving characteristics of the dipoles.

B1.4 Menu 4: Jammer to Signal Ratios (dB)

Menu 4 is variable in length as dictated by the number of signals entered earlier. Each jammer to signal ratio may be specified independently and are used to adjust the jammer powers.

B1.5 Menu 5: Relative Antenna Element Locations (meters)

Here, the user is allowed to define the actual geometry of the array by specifying the x and y coordinates of each element. This menu, again, is variable in length depending on the number of elements entered earlier.

B1.6 Menu 6: Azimuth (PHI) and Elevation (THETA) Coordinates of Incoming Signals

The user is prompted for the arrival angles of the incoming signals (in degrees). It is always assumed that the friendly communicator is signal #1. The azimuth and elevation angles are specified in terms of the spherical coordinates PHI and THETA. Phi determines the azimuthal coordinate and is defined to be 0 degrees on the x-axis and increasing toward the y-axis. Theta, on the other hand, determines the elevation and is defined to be 0 degrees on the z-axis and increasing toward the x-y plane. Note that in our case, it is only meaningful if theta is in the range from 0° to 90° as this corresponds to the space above ground.

B1.7 Menu 7: HF Channel Parameters

This is just the number of different paths to be simulated with the HF channel. It essentially chooses the number of taps in the tapped delay line model employed by the channel.

B1.8 Menu 8: Delays of Each Propagation Mode - (mS)

Menu 8 is variable in length depending on the number of channel modes entered in menu 7. Here the delay of each path is given in milliseconds and is used in the HF channel model to simulate dispersiveness. Note that the first delay is assumed to be zero, and the others are simply relative to the first. Also, it should be noticed that the path delays should be integer multiples of (1/Nyquist Rate). This is necessary, again, due to the implementation of the HF channel model.

B1.9 Menu 9: Attenuation of Each Propagation Mode (dB)

The length of this menu is determined, again, by the number of modes. The HF channel model contains multipliers which allow signals emerging from different paths to be attenuated separately. This menu lets the user assign a different attenuation to each path.

B.10 Menu 10: Adaptive Algorithm Convergence Constant

The convergence constant is used by the LMS and constrained LMS algorithms to dictate the update step sizes as convergence is taking place. This constant is quite crucial to the convergence time and overall character of the convergence, but is also very difficult to obtain. For the most part, only a trial and error type search will produce the optimum value. A rough estimated value is calculated by the program, but it should not be trusted too far. The default value may be repeatedly changed until the simulation model is executed. Note that only 5 decimal places are provided by the input program. The convergence value may be specified out to 9 decimal locations, but unfortunately only 5 are displayed. The actual value may be viewed simply by listing the parameter file.

At this point, as can be seen from Figure B-10, once the convergence constant is specified, the input program automatically submits the simulation in batch. The important files generated by the mainline are stored, as was mentioned earlier, in the user defined directory EXPERIMENT_TEST1. When the simulation is complete, it is then possible to use the output programs to view the resulting data. The output phase will be discussed next.

B2.0 Output Procedures

The purpose of this section is to explain the use of the output programs which allow both a review of the input parameters, as well as the results, of any test.

To invoke the output procedures, simply type @OUTPUT while stationed at a graphics terminal. The program responds with the question, "For which experiment do you wish to see the output?" After the response is given, which in our case is TEST1, the program returns the menu shown in Figure B-11. We see that there are 8 choices of output, the first 3 simply being a review of the input parameters of the test, while the remaining choices are actual data output from the run. To explain the use of this menu, we will step through it one option at a time.

B2.1 Option 1: Review Antenna Parameters

To invoke this or any other option, simply type the corresponding number. Thus, in this case, after a 1 is typed, a screen similar to Figure B-12 will appear. This screen reminds us of the antenna geometry which was used in the test as well as the carrier frequency and equivalent dipole lengths of the antennas. Also displayed are the amplitude and phase distributions for ech element resulting from the adaptation. To return, then, to the main menu from this option, simply type <return> and figure B-11 will again appear.

B2.2 Option 2: Review Channel Information

The second screen which may be viewed, shown in Figure B-13, is also a review of parameters, but here it is the channel information which is displayed. Several other input quantities are also listed such as data rate,

type of algorithm, number of convergences, and the convergence constant. Again, to return to the main menu, just type <return>.

B2.3 Option 3: Review Incoming Signal Information

As shown in Figure B-14, this screen lets the user review the arrival angles of the friendly signal as well as that of the jammers. Also, the jammer to signal ratios are displayed for each of the jammers.

B2.4 Option 4: View Calculated Results

This screen, arrived at by typing 4 at the main menu level, is shown in Figure B-15. This is the first screen that actually returns some data from the simulation run. The values which appear here are the average number of iterations to converge, as well as the average final signal/interference ratio. The variance of these quantities is also given, to show the user the amount of spread which has resulted in the values.

B2.5 Option 5: View Antenna Plot

Typing 5 at the main meny level allows the user to examine the antenna plots which have resulted from the simulation. This is probably the most useful portion of the output program, as it allows immediate conformation of the algorithm performance. Actually when Option 5 is chosen in the main menu, a new menu appears as shown in Figure B-16. With this menu at hand, the user is given the capability of examining any cut of the antenna pattern simply by changing the menu entries. To better understand the capabilities, we will step through each option of the sub-menu.

B2.5.1 Sub-Option 1: 1) Min Convergence Time 2) Avg. 3) Max.

When the simulation is finished, the convergence iterations of all convergences are examined and the corresponding antenna weights of the slowest, average, and fastest convergences are written to a file. With this option, it is possible to choose which set of weights will be used in computation ofthe antenna patterns. These are very similar, however, and usually indistinguishable in graph form. The default value of this parameter is set to the average.

B2.5.2 Sub-Option 2: Fixed Angle PHI for Elevation Plot

Here, the user is allowed to choose a fixed azimuth angle in order to produce an elevation plot. In other words, PHI is held fixed and THETA is allowed to vary over the range 0 to 180 degrees. For example, if PHI is equal to 0, then the resulting elevation plot will exist in the x-z plane.

B.2.5.3 Sub-Option 3: Fixed Angle THETA for Azimuth Plot

This option is very similar to the previous one in this menu, but here it is THETA which we fix instead of PHI. This value is used to produce an azimuth plot at some fixed elevation. For example, if THETA is 90 degrees, the resulting azimuth plot would exist in the x-y plane. For values other than 90 degrees, it should be realized that the resulting pattern does not exist in a plane, but is simply the projection onto a plane of the field values.

B2.5.4 Sub-Option 4: Slice Type 1) Elevation 2) Azimuth

Finally, the user must make a choice to see either an elevation or an azimuth plot. If one chooses, for example, to see an elevation plot, the PHI coordinate is set to the value dictated by Option 2, while THETA is varied to form the pattern. Thus, once elevation or azimuth are chosen, then either the Option 2 or the Option 3 values (respectively) are used; never both.

To actually see an antenna plot, this menu is exited by typing -1 as in the other menu programs. For the default values of Figure B-16, this produces the plot as shown in Figure B-17. Note that for both azimuth and elevation, 0 degrees is to the right on the plot. Thus for an azimuthal plot, right implies the x-axis, while for elevation, it implies THETA = 0 (z-axis).

At the bottom of the screen on each antenna plot, the program asks whether one wishes to see more antenna plots. If the response is yes then control is returned to the menu of Figure B-16. On the other hand, if the answer is no, the user is returned to the main menu (Figure B-11).

B2.6 Option 6: View Histogram of Convergence Counts

Upon choosing this option, one is able to look at the distribution of convergence iterations in histogram form. This is shown in Figure B-18. The bins are fixed in size and equal to 50. This value works very well for the LMS and constrained LMS, but is slightly large for the update covariance algorithm. The fixed value of 50 was chosen simply to provide ease of use.

B2.7 Option 7: View S/N Ratio vs. Time Plot

This option produces a plot of the signal to noise ratio versus the number of iterations. An example of this is shown in Figure B-19. This plot is always stopped at 6250 iterations in order to display a reasonable number of convergences while still providing an acceptable resolution. The purpose of the graph is simply to show the convergence characteristics of the test.

B2.8 Option 8: View Error Signal vs. Time Plot

The final available plot is shown in Figure B-20, and represents the magnitude of the error signal used by the LMS algorithm for adaptation. This option may only be invoked if the controlling algorithm of the test is the LMS, however. It is the only algorithm which makes use of an error signal, and consequently the mainline only produces the output file for that case. This plot is also stopped at 6250 iterations to get a reasonable number of convergences while still providing space for clarity.

```
$ @input
What is the name of this experiment?: test1
                        SIMULATION PARAMETERS

 1  NUMBER OF SAMPLES PER BIT                                  16.00000
 2  SIGNAL TO NOISE RATIO (dB)                                 10.00000
 3  SIMULATION BIT RATE (bps)                                 300.00000
 4  NUMBER OF SIMULATION CONVERGENCES DESIRED                   5.00000
4 20
NEXT ENTRY PLEASE (0 TO REVIEW THE PAGE, -1 TO LEAVE THE PAGE)
0
                        SIMULATION PARAMETERS

 1  NUMBER OF SAMPLES PER BIT                                  16.00000
 2  SIGNAL TO NOISE RATIO (dB)                                 10.00000
 3  SIMULATION BIT RATE (bps)                                 300.00000
 4  NUMBER OF SIMULATION CONVERGENCES DESIRED                  20.00000
```

Figure B-1 Menu #1 of input

```
-1
                    ADAPTIVE ALGORITHM OPTIONS
 TYPE=1    ==> LMS
 TYPE=2    ==> CONSTRAINED LMS
 TYPE=3    ==> UPDATE COVARIANCE ALGORITHM

 1  ADAPTIVE ALGORITHM TYPE (1,2, OR 3)                         1.00000
-1
```

Figure B-2 Menu #2 of input

ANTENNA ARRAY PARAMETERS

1	NUMBER OF ANTENNA ELEMENTS	9.00000
2	NUMBER OF INCOMING SIGNALS (DESIRED + JAMMERS)	3.00000
3	DIPOLE EQUIVALENT ELEMENT LENGTH-(m)	36.57600
4	CARRIER FREQUENCY OF FRIENDLY COMMUNICATOR-(kHz)	30000.00000

-1

Figure B-3 Menu #3 of input

JAMMER TO SIGNAL RATIOS

(dB)

1	J/S RATIO FOR JAMMER 1	20.00000
2	J/S RATIO FOR JAMMER 2	20.00000

Figure B-4 Menu #4 of input

```
                RELATIVE ANTENNA ELEMENT LOCATIONS
                        MEASURED IN METERS

 1 X COORDINATE OF ELEMENT 1                          10.00000
 2 Y COORDINATE OF ELEMENT 1                          10.00000
 3 X COORDINATE OF ELEMENT 2                           5.00000
 4 Y COORDINATE OF ELEMENT 2                          10.00000
 5 X COORDINATE OF ELEMENT 3                           0.00000
 6 Y COORDINATE OF ELEMENT 3                          10.00000
 7 X COORDINATE OF ELEMENT 4                          10.00000
 8 Y COORDINATE OF ELEMENT 4                           5.00000
 9 X COORDINATE OF ELEMENT 5                           5.00000
10 Y COORDINATE OF ELEMENT 5                           5.00000

-1
                RELATIVE ANTENNA ELEMENT LOCATIONS
                        MEASURED IN METERS

11 X COORDINATE OF ELEMENT 6                           0.00000
12 Y COORDINATE OF ELEMENT 6                           5.00000
13 X COORDINATE OF ELEMENT 7                          10.00000
14 Y COORDINATE OF ELEMENT 7                           0.00000
15 X COORDINATE OF ELEMENT 8                           5.00000
16 Y COORDINATE OF ELEMENT 8                           0.00000
17 X COORDINATE OF ELEMENT 9                           0.00000
18 Y COORDINATE OF ELEMENT 9                           0.00000
-1
DO YOU WANT TO REVIEW ALL THE PARAMETERS? (Y/N)
```

Figure B-5 Menu #5 of input

```
n
AZIMUTH (PHI) AND ELEVATION (THETA) COORDINATES OF INCOMING SIGNALS
PHI=0 IMPLIES THE X-AXIS. PHI INCREASES TOWARD THE Y-AXIS.
THETA=0 IMPLIES THE +Z-AXIS. (INCREASES TOWARD THE X-Y PLANE)
SIGNAL #1 IS ASSUMED TO BE THE FRIENDLY COMMUNICATOR.

 1 PHI COORDINATE OF SIGNAL  1                          90.00000
 2 THETA COORDINATE OF SIGNAL  1                        60.00000
 3 PHI COORDINATE OF SIGNAL  2                          90.00000
 4 THETA COORDINATE OF SIGNAL  2                        80.00000
 5 PHI COORDINATE OF SIGNAL  3                         150.00000
 6 THETA COORDINATE OF SIGNAL  3                        75.00000
```

Figure B-6 Menu #6 of input

```
                    HF CHANNEL PARAMETERS

 1  NUMBER OF MODES OF PROPOGATION (TAPS)               3.00000
-1
```

Figure B-7 Menu #7 of input

```
                DELAYS OF EACH PROPOGATION MODE
                    MEASURED IN MILLISECONDS
 ** SHOULD BE INTEGER MULTIPLES OF 1/(NYQUIST RATE) **

 1  DELAY OF PATH NUMBER  1                            0.00000
 2  DELAY OF PATH NUMBER  2                            0.83333
 3  DELAY OF PATH NUMBER  3                            1.66666
```

Figure B-8 Menu #8 of input

```
                ATTENUATION OF EACH PROPOGATION MODE
                              (dB)

 1  ATTENUATION OF PATH NUMBER  1                                  0.00000
 2  ATTENUATION OF PATH NUMBER  2                                  0.00000
 3  ATTENUATION OF PATH NUMBER  3                                  0.00000
-1
```

Figure B-9 Menu #9 of input

```
   ESTIMATED CONVERGENCE CONSTANT=    1.3888889E-05
         ADAPTIVE ALGORITHM CONVERGENCE CONSTANT

 1  DEFAULT CONVERGENCE CONSTANT VALUE                             0.00001
-1
Job MAIN (queue SYS$BATCH, entry 740) started on SYS$BATCH
$
```

Figure B-10 Menu #10 of input

```
OUTPUT- The Main Menu

 1. Review antenna parameters
 2. Review channel information
 3. Review incoming signal information
 4. View calculated results
 5. View antenna plot
 6. View histogram of convergence counts
 7. View S/N vs. time graph
 8. View error signal vs. time graph
-1. End output

Which?
```

Figure B-11 Main menu of output

```
        Antenna Element Parameters

element        x        y        amplitude      phase(in radians)
   1         10.00    10.00        0.31            2.19
   2          5.00    10.00        0.43            2.08
   3          0.00    10.00        0.31            1.99
   4         10.00     5.00        0.03           -0.99
   5          5.00     5.00        0.12            3.55
   6          0.00     5.00        0.03            1.83
   7         10.00     0.00        0.31           -1.15
   8          5.00     0.00        0.43           -1.24
   9          0.00     0.00        0.31           -1.36

Carrier Frequency (in Hz) :    30000000.0
Antenna Length (in meters):           36.6
```

Figure B-12 Antenna parameters

```
          Channel Information

Data Rate (in bits/sec) :          300
Number of Samples/Bit   :           16
Algorithm used:                   LMS
Number of Convergences  :           20
The Convergence Constant:     0.00001493
Channel S/N ratio(in dB):        10.00

              Delay Information
    delay(in msec)        path attenuation (in dB)
         0.000                      0.00
         0.833                      0.00
         1.667                      0.00
```

Figure B-13 Channel and other assorted information

Incoming Signal Information

signal	phi(in degrees)	theta(in degrees)	J/S(in dB)
friendly	90.00	60.00	
jammer 1	90.00	80.00	20.000
jammer 2	150.00	75.00	20.000

Figure B-14 Signal information

Calculated Results

Average Number of Iterations: 326.00
Variance : 13026.32

Average Signal/Interference (in dB): 13.83
Variance : 2.13

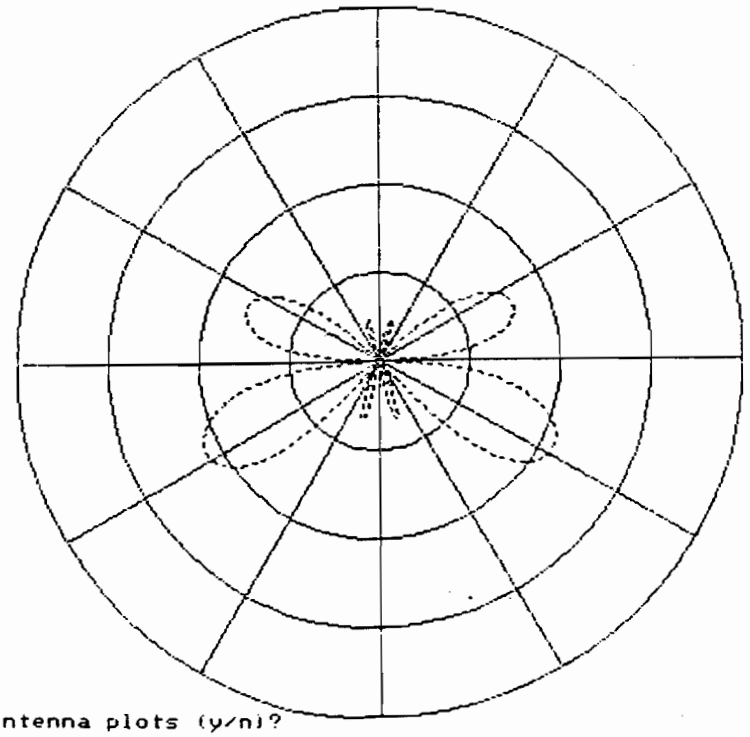
Figure B-15 Calculated results of output

SLICE DESCRIPTION

After exiting this menu, the graph data will be calculated and then plotted in polar form.

1	1) min. convergence time 2) avg. 3) max	2.00000
2	Fixed angle Phi for elevation plot (in deg.)	0.00000
3	Fixed angle Theta for azimuth plot (in deg.)	45.00000
4	Slice type: 1)Elevation 2)Azimuth	1.00000

Figure B-16 Antenna pattern menu



More Antenna plots (y/n)?

Figure B-17 Antenna plot from Menu 5

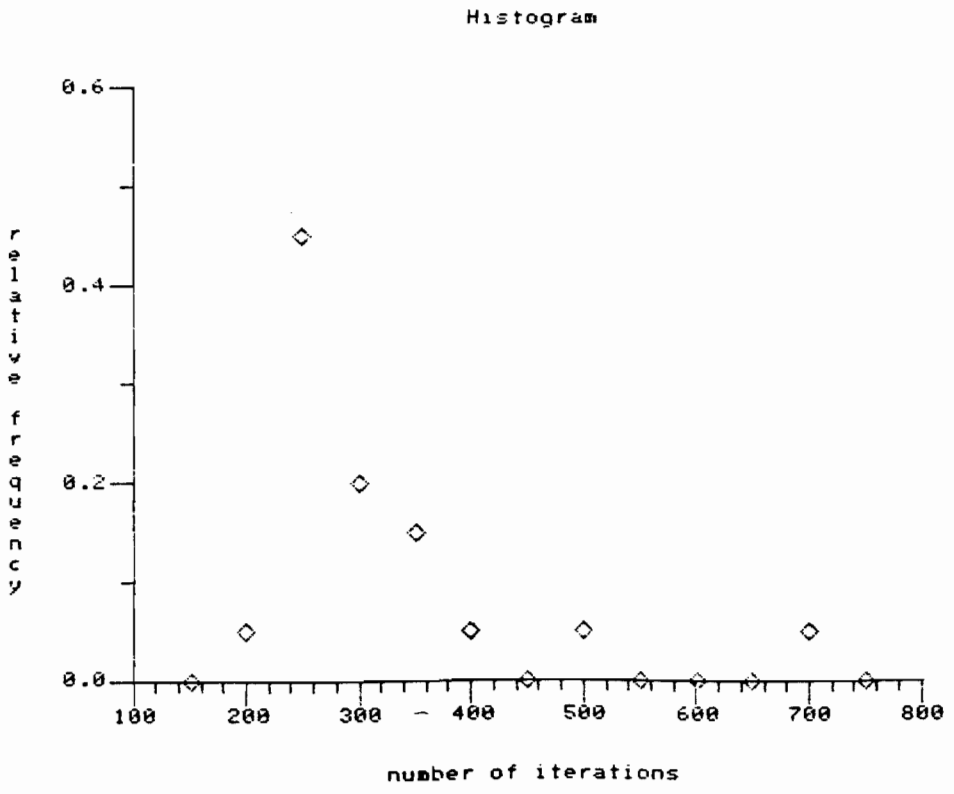


Figure B-18 Convergence histogram

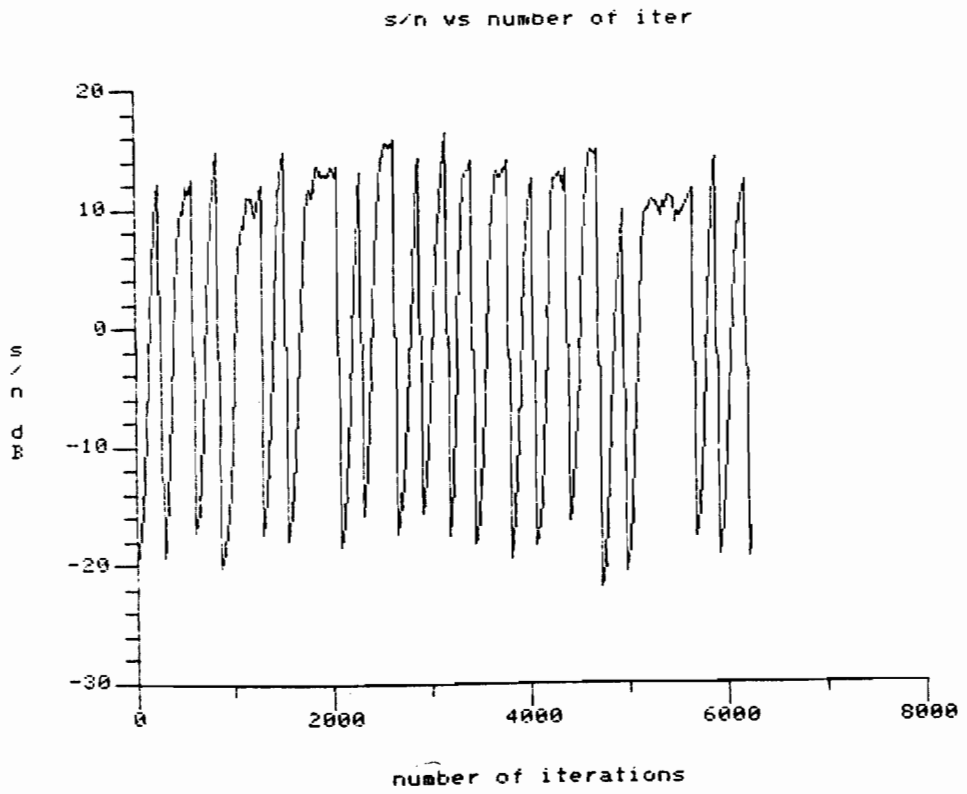


Figure B-19 Signal to noise plot

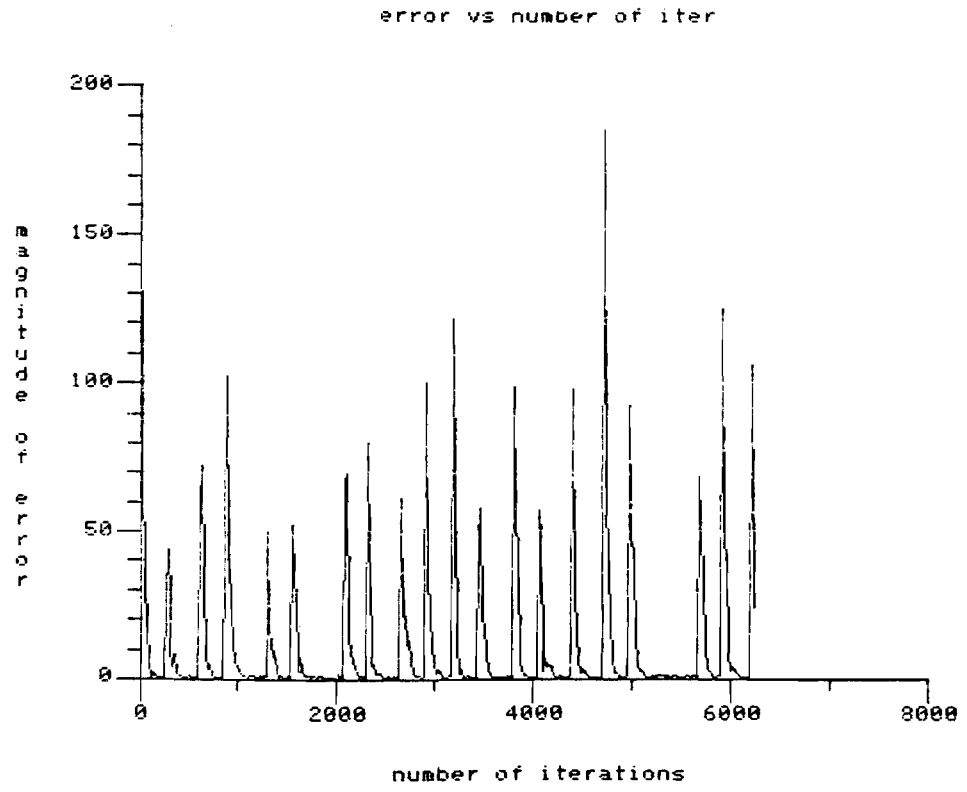


Figure B-20 Error signal plot

APPENDIX C

HF ADAPTIVE ANTENNA ARRAY SIMULATION FACILITY: SOFTWARE REFERENCE MANUAL

```
C ------------------------------------------------------------------
C TELECOMMUNICATION AND INFORMATION SCIENCES LABORATORY, UNV. OF KANSAS
C ------------------------------------------------------------------
C PROGRAM NAME: MAIN          AUTHOR: Roger Spohn     DATE: 11/85
C ------------------------------------------------------------------
C VERSION NUMBER : 1.         AMENDMENT DATE(S) :
C ------------------------------------------------------------------
C PURPOSE :     This program is the mainline routine which controls
C               the simulation. Main invokes all of the subroutines
C               and the description of the variables can be found with
C               those modules.
C ------------------------------------------------------------------
C                       PARAMETER DEFINITION
C  NAME          ! TYPE  ! CLASS! RANGE  !     DESCRIPTION
C ------------------------------------------------------------------
C                !       !      !        !
C                !       !      !        !
C                !       !      !        !
C                !       !      !        !
C                !       !      !        !
C                !       !      !        !
C                !       !      !        !
C                !       !      !        !
C                !       !      !        !
C                !       !      !        !
C                !       !      !        !
C                !       !      !        !
C                !       !      !        !
C                !       !      !        !
C                !       !      !        !
C                !       !      !        !
C                !       !      !        !
C                !       !      !        !
C                !       !      !        !
C                !       !      !        !
C                !       !      !        !
C ------------------------------------------------------------------
C                NON-LOCAL VARIABLES -- FILE DEFINITIONS
C ------------------------------------------------------------------
C                !       !      !        !
C                !       !      !        !
C                !       !      !        !
C                !       !      !        !
C                !       !      !        !
C                !       !      !        !
C                !       !      !        !
C                !       !      !        !
C                !       !      !        !
C ------------------------------------------------------------------
C                    SUBROUTINES REQUIRED
C    NAME        !          DESCRIPTION
C ------------------------------------------------------------------
C                !
C                !
C                !
C                !
C                !
C                !
C ------------------------------------------------------------------
```

```
      IMPLICIT NONE
      INTEGER LUN1, I, J, LUN2, NOCONV, LUN3, LUN4, LUN5
C*****************************************************************
      INTEGER*4 MXNANT, MXNSIG, NANT, NSIG, MXCONV
      PARAMETER(MXNANT=11, MXNSIG=15)
      REAL CARFRQ, ANTLEN, X(MXNANT), Y(MXNANT)
      REAL PHI(MXNSIG), THETA(MXNSIG)
      REAL PI, DTR, J2SRAT(MXNSIG-1)
      COMPLEX COMPWT(MXNANT), STRDEL(MXNANT), IMGUNT
      PARAMETER(IMGUNT=(0., 1.))
      PARAMETER(PI=3.141592654)
      PARAMETER(DTR=PI/180.)
      PARAMETER(MXCONV=100)
C*****************************************************************
      INTEGER BITRT, ALGTYP
      REAL NYBAND, SAMPRT, SNRAT, NSRAT, betal
      COMPLEX RNN(MXNANT, MXNANT), DESIG(MXNANT), RNNINV(MXNANT, MXNANT)
      COMPLEX RSS(MXNANT, MXNANT), SNRO
      complex*16 temp, temp2
      REAL SNROPT, SNR
      CHARACTER*20 ALDESC
C*****************************************************************
      INTEGER MAXMOD, SIGINV, JAMINV(MXNSIG-1)
      PARAMETER (MAXMOD=20)
      INTEGER*4 NUMMOD, NPTHDL(MAXMOD)
      REAL ATTENU(MAXMOD), PHERR, MEAN, VAR, DATA(MXCONV)
      REAL INPOW, OUTPOW, SIGPOW, JAMPOW(MXNSIG-1), SIGAMP
      real amp, ph, elv
      LOGICAL SIGNAL, NEWTAP
C*****************************************************************
      INTEGER INVOKE, iter, COUNT
      INTEGER CODE, BITNUM, PNCODE(31), NSAMPB, jampnt
      REAL TIME, DELTAT, KS, GAIN(10)
      real reljam, imajam, jamdev, jmseed(20)
      LOGICAL NEW, initupd, lmsinit
      COMPLEX SIGIN(MXNSIG), INSIG, OUTSIG
C*****************************************************************
      data (pncode(i), i=1,31)/-1, 1, -1, 1, 1, 1, -1, 1, 1, -1, -1, -1,
     & 1, 1, 1, 1, 1, -1, -1, 1, 1, -1, 1, -1, -1, 1, -1, -1, -1, -1, 1/
C
      data (jmseed(i), i=1, 20)/839205815, 826491047, 228472107,
     &309765101, 208524933, 502968209, 449376019, 348203833, 984961517,
     &449256291, 553735617, 945845675, 644523461, 455678993, 346875421,
     &745966205, 672356833, 935591659, 249793233, 946857263/
C
C     Initialization of variables.
C
      NEW=.TRUE.
      SIGINV=0
      DO I=1, MXNSIG-1
         JAMINV(I)=0
      END DO
C
      INCLUDE 'FORMAT.INC'
C
C     Read data in from parameter file.
C
      CALL OPNCHK(LUN1, 'PARAMS.DAT', 'OLD', 'FOR')
      READ(LUN1, 101) NSAMPB
      READ(LUN1, 100) SNRAT
```

```
      READ(LUN1,103) BITRT,ALGTYP
      READ(LUN1,103) NANT,NSIG
      READ(LUN1,102) ANTLEN,CARFRQ
      READ(LUN1,101) NOCONV
      DO I=1,NANT
         READ(LUN1,102) X(I),Y(I)
      END DO
      DO I=1,NSIG
         READ(LUN1,102) PHI(I),THETA(I)
      END DO
      DO I=1,NSIG-1
         READ(LUN1,100) J2SRAT(I)
      END DO
      READ(LUN1,101) NUMMOD
      DO I=1,NUMMOD
         READ(LUN1,101) NPTHDL(I)
      END DO
      DO I=1,NUMMOD
         READ(LUN1,100) ATTENU(I)
      END DO
      READ(LUN1,100) KS
      CALL UCLOSE(LUN1)
C
      DELTAT=1./FLOAT(BITRT*NSAMPB)
      NYBAND=4*BITRT/1000.
      SAMPRT=BITRT*NSAMPB/1000.
C
C
C     Write input data to the simulation report file.
C
      CALL OPNCHK(LUN2,'SIMREPORT.DAT','NEW','FOR')
      WRITE(LUN2,110)
 110  FORMAT (40X,'SIMULATION INPUT DATA')
      WRITE(LUN2,120) NYBAND,SAMPRT,SNRAT,BITRT
 120  FORMAT(//1X,'NYQUIST BANDWIDTH OF INFORMATION=',F20.6,' kHz'
     &/1X,'SAMPLE RATE=',F20.6,' kHz'/1X,'SIGNAL TO NOISE RATIO='
     &,F20.6,' dB'/1X,'BIT RATE=',I10,' bps')
      IF (ALGTYP .EQ. 1) THEN
         ALDESC='LEAST MEAN SQUARED'
      ELSE IF (ALGTYP .EQ. 2) THEN
         ALDESC='CONSTRAINED LMS'
      ELSE
         ALDESC='RECURSIVE'
      END IF
      WRITE(LUN2,130) ALDESC
 130  FORMAT(1X,'ADAPTIVE ALGORITHM TYPE IS ',A20)
      WRITE(LUN2,140) ANTLEN,CARFRQ
 140  FORMAT(1X,'DIPOLE EQUIVALENT ELEMENT LENGTHS=',F20.6,' meters'
     &/1X,'CARRIER FREQUENCY OF FRIENDLY COMMUNICATOR=',F20.6,' kHz')
      WRITE(LUN2,150)
 150  FORMAT(/////20X,'ANTENNA ELEMENT LOCATIONS - (METERS)'//10X,'I'
     &,20X,'X(I)',20X,'Y(I)'/)
      DO I=1,NANT
         WRITE(LUN2,160) I,X(I),Y(I)
 160     FORMAT(1X,I10,2F20.6)
      END DO
      WRITE(LUN2,170)
 170  FORMAT(/////20X,'INCOMING SIGNAL DIRECTIONS-(DEGREES)'//10X,
     &'I',20X,'PHI(I)',20X,'THETA(I)'/)
      DO I=1,NSIG
```

```
            WRITE(LUN2,180) I,PHI(I),THETA(I)
 180        FORMAT(1X,I10,2F20.6)
         END DO
         WRITE(LUN2,186)
 186     FORMAT(/////20X,'JAMMER/SIGNAL RATIO FOR EACH JAMMER-(dB)'
        &//1X,'JAMMER NO.',10X,'J/S RATIO'/)
         DO I=1,NSIG-1
            WRITE(LUN2,188) I,J2SRAT(I)
 188        FORMAT(1X,I10,F20.6)
         END DO
         WRITE(LUN2,190)
 190     FORMAT(/////20X,'HF CHANNEL PARAMETERS'//1X,'MODE NO.',10X,
        &'DELAY-(msec)'/)
         DO I=1,NUMMOD
            WRITE(LUN2,200) I,NPTHDL(I)/NYBAND
 200        FORMAT(1X,I10,F20.6)
         END DO
         WRITE(LUN2,210)
 210     FORMAT(/////1X,'MODE NO.',10X,'MODE GAIN-(dB)'/)
         DO I=1,NUMMOD
            WRITE(LUN2,220) I,ATTENU(I)
 220        FORMAT(1X,I10,F20.6)
         END DO
C
C        Convert variables to actual values.
C
         NYBAND=NYBAND*1000.
         SAMPRT=SAMPRT*1000.
         CARFRQ=CARFRQ*1000.
         SNRAT=10.**(SNRAT/10.)
         NSRAT=1./SNRAT
         DO I=1,NSIG-1
            J2SRAT(I)=10.**(J2SRAT(I)/10.)
         END DO
         DO I=1,NUMMOD
            ATTENU(I)=10.**(ATTENU(I)/10.)
         END DO
c
c        normalize the antenna gain
c
         betal=2.*pi*carfrq*antlen/(299792458.)
c
c
         do i=1,nsig
         elv=theta(i)*pi/180.
            gain(i)=(cos((betal*cos(elv))/2.)-cos(betal/2.))/
        &          sin(elv)
         end do
C
C        Determine the power in the jammers.
C
         SIGNAL =.FALSE.
         DO I=1,NSIG-1
             INPOW = J2SRAT(I)
            CALL POWER(INPOW,ATTENU,NUMMOD,GAIN(I+1),ANTLEN,CARFRQ,
        &                SIGNAL,OUTPOW)
             JAMPOW(I)=OUTPOW
         END DO
C
C        Signal/jammer optimization
```

```
C
      CALL SNCOVAR(NSIG, NANT, X, Y, PHI, THETA, JAMPOW, NSRAT, CARFRQ, RSS, RNN,
     &             DESIG)
      CALL MATINV(NANT, RNN, RNNINV)
C
C     Create output files.
C
      CALL OPNCHK(LUN3, 'WEIGHTS.DAT', 'NEW', 'FOR')
      CALL OPNCHK(LUN4, 'CONVTIME.DAT', 'NEW', 'FOR')
      CALL OPNCHK(LUN5, 'SNR.DAT', 'SCR', 'FOR')
C
C     Begin actual simulation.
C
      bitnum=0
      iter=0
C
      DO J=1, NOCONV
C
C        Calculate signal power.
C
         INPOW = 1.
         SIGNAL = .TRUE.
         CALL POWER(INPOW, ATTENU, NUMMOD, GAIN(1), ANTLEN, CARFRQ,
     &              SIGNAL, OUTPOW)
         SIGPOW = OUTPOW
         SIGAMP=SQRT(SIGPOW)
         CALL MATMULT(NANT, RNNINV, DESIG, SIGAMP, 1., SNRO)
         WRITE(6,*) CABS(SNRO)
         SNROPT=10.*LOG10(CABS(SNRO))
C
C        Initialize antenna weights subject to algorithm type.
C
         CALL WEIGHTINIT(PHI(1), THETA(1), X, Y, CARFRQ, ALGTYP, COMPWT,
     &                   NANT, STRDEL)
C
C        Determine phase error introduced by the HF channel.
C
         NEWTAP=.TRUE.
         CALL HFPHERROR(SIGINV, NUMMOD, NYBAND,
     &                  SAMPRT, NPTHDL, ATTENU, NEWTAP, PHERR)
C
C
 300     IF(NEW) THEN
           COUNT=0
           initupd=.true.
           lmsinit=.true.
         END IF
         NEW=.FALSE.
         COUNT=COUNT + 1
         iter=iter+1
         TIME=DELTAT*FLOAT(COUNT)
         IF(MOD(iter + NSAMPB-1, NSAMPB) .EQ. 0) THEN
            BITNUM=BITNUM+1
            IF(BITNUM .GT. 31) BITNUM=1
            CODE=PNCODE(BITNUM)
         END IF
         SIGIN(1)=CMPLX(FLOAT(CODE), 0.)
C
C
         jampnt=0
```

```
            DO I=1,NSIG-1
               jampnt=jampnt+2
               jamdev=sqrt(j2srat(i)/2.)
               call rannrm(jmseed(jampnt-1),0.0,jamdev,reljam)
               call rannrm(jmseed(jampnt),0.0,jamdev,imajam)
               sigin(i+1)=cmplx(reljam,imajam)
            END DO
C
C
            CALL HFCHANNEL(SIGINV,NUMMOD,SIGIN(1),OUTSIG,NYBAND,SAMPRT,
     &                     NPTHDL,ATTENU,NEWTAP)
C
C
             SIGIN(1)=OUTSIG*EXP(-1.*IMGUNT*PHERR)
C
C
             DO I=1,NSIG-1
               CALL HFCHANNEL(JAMINV(I),NUMMOD,SIGIN(I+1),OUTSIG,NYBAND,
     &                        SAMPRT,NPTHDL,ATTENU,NEWTAP)
               SIGIN(I+1)= OUTSIG
             END DO
C
C
C
            CALL ADAPTA(INVOKE,ALGTYP,NANT,NSIG,CARFRQ,
     &                  PHI,THETA,KS,X,Y,SIGIN,NSRAT,
     &                  COMPWT,STRDEL,CODE,initupd,lmsinit,gain,nsampb)
C
C
C           Check for convergence.
C
            IF (ALGTYP .EQ. 3) THEN
               CALL MATMULT2(NANT,RSS,COMPWT,1.,SIGPOW,TEMP)
               CALL MATMULT2(NANT,RNN,COMPWT,1.,1.,TEMP2)
               SNR=CDABS(TEMP/TEMP2)
               SNR=10.*LOG10(SNR)
            ELSE
               IF (MOD(COUNT-1,50) .EQ. 0) THEN
                  CALL MATMULT2(NANT,RSS,COMPWT,1.,SIGPOW,TEMP)
                  CALL MATMULT2(NANT,RNN,COMPWT,1.,1.,TEMP2)
                  SNR=CDABS(TEMP/TEMP2)
                  SNR=10.*LOG10(SNR)
               END IF
            END IF
            IF(SNR .LT. SNROPT-1.0) GOTO 300
            NEW=.TRUE.
C
C           Save antenna weights.
C
            DO I=1,NANT
               amp=cabs(compwt(i))
               ph=atand(-aimag(compwt(i))/real(compwt(i)))
               if (real(compwt(i)) .lt. 0.0) ph=ph+180.0
               WRITE(LUN3,*) amp,ph
            END DO
C
C           Save convergence time.
C
            WRITE(LUN4,*) COUNT
C
```

```
C         Save signal/noise ratio.
C
          WRITE(LUN5,*) SNR
C
       END DO
C
C
C
C      Compute average convergence time and convergence time
C      variance.
C
       REWIND LUN4
       DO I=1,NOCONV
          READ(LUN4,*) DATA(I)
       END DO
       CALL STAT(DATA,NOCONV,MEAN,VAR)
       WRITE(LUN2,310) MEAN,VAR
 310   FORMAT(/////1X,'MEAN ITERATIONS TO CONVERGE= ',F20.6//1X,
      &'VARIANCE=',F20.6)
       CALL UCLOSE(LUN4)
C
C
C      Compute average signal to interference ratio.
C
       REWIND LUN5
       DO I=1,NOCONV
          READ(LUN5,*) DATA(I)
       END DO
       CALL STAT(DATA,NOCONV,MEAN,VAR)
       WRITE(LUN2,320) MEAN,VAR
 320   FORMAT(////1X,'MEAN S/I RATIO ',F20.6,'dB'//1X,'S/I VARIANCE= ',
      &F20.6)
       CALL UCLOSE(LUN5)
       END
```

```
C ------------------------------------------------------------------
C       TELECOMMUNICATION AND INFORMATION SCIENCES LABORATORY
C ------------------------------------------------------------------
C PROGRAM NAME: UPDCOVAR       AUTHOR: D. HAEGERT       DATE: 11-85
C ------------------------------------------------------------------
C VERSION NUMBER : 1           AMENDMENT DATE(S) :
C ------------------------------------------------------------------
C PURPOSE :     This routine is a software implementation of an Update
C               covariance recursive processor (adaptive algorithm).
C
C
C ------------------------------------------------------------------
C                       PARAMETER DEFINITION
C  NAME           | TYPE    | CLASS | RANGE  |    DESCRIPTION
C ------------------------------------------------------------------
C NELMTS          |INTEGER*4 |READ   | <11    | # OF ANTENNA ELEMENTS
C OUTPUT          |COMPLEX   |READ   |        | SENSOR ELEMENT OUTPUTS
C COMPWT          |COMPLEX   | R/W   |        | ANTENNA WEIGHTS
C INITUPD         |LOGICAL   | R/W   |        | SIGNIFIES START-UP
C                 |          |       |        |
C                 |          |       |        |
C                 |          |       |        |
C                 |          |       |        |
C                 |          |       |        |
C                 |          |       |        |
C                 |          |       |        |
C                 |          |       |        |
C                 |          |       |        |
C                 |          |       |        |
C                 |          |       |        |
C                 |          |       |        |
C                 |          |       |        |
C                 |          |       |        |
C                 |          |       |        |
C ------------------------------------------------------------------
C               NON-LOCAL VARIABLES -- FILE DEFINITIONS
C ------------------------------------------------------------------
C                 |          |       |        |
C                 |          |       |        |
C                 |          |       |        |
C                 |          |       |        |
C                 |          |       |        |
C                 |          |       |        |
C                 |          |       |        |
C                 |          |       |        |
C                 |          |       |        |
C ------------------------------------------------------------------
C                     SUBROUTINES REQUIRED
C    NAME         |          DESCRIPTION
C ------------------------------------------------------------------
C                 |
C                 |
C                 |
C                 |
C                 |
C                 |
C ------------------------------------------------------------------
      subroutine updcovar(compwt,output,nelmts,initupd)
      implicit none
      integer*4 i,j,k,nelmts,cnt
```

```
      real alpha1,alpha2,rpart,ipart,amp,phase
      complex compwt(11),output(11),Rxxinv(11,11)
      complex xconj,mat1(11,11),mat2(11,11)
      complex z(11),f(11),sum1,sum2,denom
      logical initupd
      save cnt,Rxxinv
*
      if (initupd) then
         initupd=.false.
         cnt=0
         do j=1,nelmts
            do i=1,nelmts
               Rxxinv(j,i)=cmplx(0.0,0.0)
               if(i .eq. j) Rxxinv(j,i)=cmplx(1.0,0.0)
            end do
         end do
      end if
*
      cnt=cnt+1
      alpha1=cnt*cmplx(1.0,0.0)
      alpha2=(alpha1+1.0)/alpha1
*
      denom=(0.0,0.0)
      do j=1,nelmts
         sum1=(0.0,0.0)
         do i=1,nelmts
            sum1=Rxxinv(j,i)*conjg(output(i)) + sum1
         end do
         z(j)=sum1
         denom=denom+z(j)*output(j)
      end do
      denom=denom+alpha1
*
C      write(6,*)'X          Rxx*xconj'
C      do j=1,nelmts
C         write(6,*)output(j),' ',z(j)
C      end do
*
      do j=1,nelmts
         sum2=(0.0,0.0)
         z(j)=z(j)/denom
         do i=1,nelmts
            mat1(j,i)=z(j)*output(i)
            sum2=mat1(j,i)*compwt(i) + sum2
         end do
         f(j)=sum2
      end do
*
      do j=1,nelmts
         compwt(j)=(compwt(j)-f(j))*alpha2
      end do
*
C      write(6,*)'numerator          Weight'
C      do j=1,nelmts
C         write(6,*)f(j),' ',compwt(j)
C      end do
*
*
      do k=1,nelmts
         do j=1,nelmts
```

```
            mat2(k,j)=(0.0,0.0)
            do i=1,nelmts
               mat2(k,j)=mat2(k,j)+mat1(k,i)*Rxxinv(i,j)
            end do
         end do
      end do
*
*
      do j=1,nelmts
         do i=1,nelmts
            Rxxinv(j,i)=(Rxxinv(j,i)-mat2(j,i))*alpha2
         end do
      end do
*
C      write(6,*)'new Rxx              Mat2'
C      do j=1,nelmts
C         do i=1,nelmts
C            write(6,*)Rxxinv(j,i),' ',mat2(j,i)
C         end do
C      end do
*
      if(mod(cnt,100) .eq. 0) then
         do j=1,nelmts
            rpart=real(compwt(j))
            ipart=aimag(compwt(j))
            amp=sqrt(rpart**2+ipart**2)
            phase=atand(-ipart/rpart)
            if(rpart .lt. 0.) phase=phase+180.0
            write(6,*)j,' ',amp,' ',phase
         end do
         write(6,*)' '
      end if
      return
      end
```

```
C ------------------------------------------------------------------
C       TELECOMMUNICATION AND INFORMATION SCIENCES LABORATORY
C ------------------------------------------------------------------
C PROGRAM NAME: WEIGHTINIT     AUTHOR: D. HAEGERT       DATE: 10-85
C ------------------------------------------------------------------
C VERSION NUMBER : 1           AMENDMENT DATE(S) :
C ------------------------------------------------------------------
C PURPOSE :     To initialize the complex antenna weights for the
C               adaptive algorithms.
C
C
C ------------------------------------------------------------------
C                       PARAMETER DEFINITION
C  NAME          | TYPE   | CLASS| RANGE   |    DESCRIPTION
C ------------------------------------------------------------------
C                |        |      |         |
C NELMTS          |INT*4   |READ  |  <11    | # OF ANTENNA ELEMENTS
C ALGTYP          |INT*4   |READ  | 1,2,3   | TYPE OF ADAPTIVE ALGORITHM
C PHI1            |REAL    |READ  |  0-360  | AZIMUTH ANGLE OF COMMUNICATO
C THETA1          |REAL    |READ  |  0-90   | ELEVATION ANGLE OF COMMUNICA
C X               |REAL    |READ  |         | ARRAY OF SENSOR LOCATION ON
C Y               |REAL    |READ  |         |ARRAY OF SENSOR LOCATIONS (Y-
C CARFRQ          |REAL    |READ  |         | CARRIER FREQUENCY
C COMPWT          |COMPLEX |WRITE |         | COMPLEX ANTENNA WEIGHTS
C STRDEL          |COMPLEX |WRITE |         | STEERING DELAYS FOR ALG. 2
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C ------------------------------------------------------------------
C              NON-LOCAL VARIABLES -- FILE DEFINITIONS
C ------------------------------------------------------------------
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C ------------------------------------------------------------------
C                    SUBROUTINES REQUIRED
C   NAME         |          DESCRIPTION
C ------------------------------------------------------------------
C                |
C                |
C                |
C                |
C                |
C                |
C ------------------------------------------------------------------
      subroutine weightinit(phi1, theta1, x, y, carfrq, algtyp, compwt,
     &                      nelmts, strdel)
      implicit none
```

```
      integer*4 i,algtyp,nelmts
      real phi1,theta1,azi,elv,carfrq,dtr,c,rotax,x(11),y(11)
      real pi,phslag,w1,w2
      complex compwt(11),strdel(11)
      parameter (pi=3.14159265, c=3.e8, dtr=0.01745329)
*
      azi=phi1*dtr
      elv=theta1*dtr
*
         do i=1,nelmts
            rotax=y(i)*sin(azi)+x(i)*cos(azi)
            phslag=sin(elv)*2.*pi*carfrq*rotax/c
            w1=cos(phslag)
            w2=sin(phslag)
            if (algtyp .eq. 1) then
               compwt(i)=cmplx(w1,w2)
            end if
            if (algtyp .eq. 2) then
               strdel(i)=cmplx(w1,w2)
               compwt(i)=strdel(i)/nelmts
            end if
            if (algtyp .eq. 3) then
               compwt(i)=cmplx(w1,w2)/nelmts
            end if
         end do
      return
      end
```

```
C ------------------------------------------------------------------
C       TELECOMMUNICATION AND INFORMATION SCIENCES LABORATORY
C ------------------------------------------------------------------
C PROGRAM NAME: tapde1          AUTHOR: c.townsend       DATE: 5/85
C ------------------------------------------------------------------
C VERSION NUMBER : 1            AMENDMENT DATE(S) :
C ------------------------------------------------------------------
C PURPOSE :     to convolve an input signal with the impulse
C response of a channel using a circular tap delay line.
C
C
C ------------------------------------------------------------------
C                       PARAMETER DEFINITION
C  NAME           | TYPE   | CLASS| RANGE   |     DESCRIPTION
C ------------------------------------------------------------------
C invoke          |mod     |int*4 | 0-20    |invokation number.
C insig           |read    |int*4 |         |input signal.
C numtap          |read    |int*4 |0-4096   |number of taps on line.
C tgain           |read    |cmplx |         |channel impulse response.
C                 |        |      |         |dimension tgain(numtap).
C tap             |read    |log   |         |logical flags to indicate
C                 |        |      |         |which taps are on
C outsig          |write   |int*4 |         |output signal.
C                 |        |      |         |
C                 |        |      |         |
C                 |        |      |         |
C                 |        |      |         |
C                 |        |      |         |
C                 |        |      |         |
C                 |        |      |         |
C                 |        |      |         |
C                 |        |      |         |
C                 |        |      |         |
C                 |        |      |         |
C                 |        |      |         |
C ------------------------------------------------------------------
C             NON-LOCAL VARIABLES -- FILE DEFINITIONS
C ------------------------------------------------------------------
C                 |        |      |         |
C                 |        |      |         |
C                 |        |      |         |
C                 |        |      |         |
C                 |        |      |         |
C                 |        |      |         |
C                 |        |      |         |
C                 |        |      |         |
C                 |        |      |         |
C ------------------------------------------------------------------
C                   SUBROUTINES REQUIRED
C    NAME         |          DESCRIPTION
C ------------------------------------------------------------------
C                 |
C                 |
C                 |
C                 |
C                 |
C                 |
C                 |
C ------------------------------------------------------------------
      subroutine tapde12(invoke, insig, numtap, tgain, tap, outsig)
```

```
c
      implicit none
      integer invoke,numtap,maxinv,numinv
      integer arg1,arg2,i
      complex tgain(*),insig,outsig
      parameter (maxinv=20)
      integer j(maxinv)
      complex delay(4096,maxinv)
      logical tap(*)
c
c  save local storage.
c
      save numinv, delay, j
c
c  check to verify numtap is less than 4096.
c
      if (numtap .gt. 4096) then
      write(*,*)'maximum number of taps exceeded in tapdel routine'
      stop
      end if
c
c  initialization.
c
      if (invoke .eq. 0) then
      numinv = numinv + 1
      if (numinv .gt. maxinv) then
         write (*,*) 'max invokes exceeded in tapdel routine'
         stop
      end if
      invoke = numinv
      j(invoke) = 0
      end if
c
c  initialize output to zero.
c
      outsig = cmplx(0.,0.)
c
c  set up circular tap memory locations.
c
      arg1 = mod(j(invoke),numtap) + 1
      delay(arg1,invoke) = insig
c
c  multiply each tap gain by the appropriate input signal
c  and sum to get the output.
c
      do i = 1,numtap
         arg2 = mod(numtap-i+j(invoke)+1,numtap) + 1
         if (tap(i)) then
            outsig = outsig + (delay(arg2,invoke)*tgain(i))
         end if
      end do
c
c
      j(invoke) = j(invoke) + 1
c
c  finished with this point!!
c
      return
      end
```

```
C ------------------------------------------------------------------
C       TELECOMMUNICATION AND INFORMATION SCIENCES LABORATORY
C ------------------------------------------------------------------
C PROGRAM NAME: MATMULT2       AUTHOR: D. HAEGERT       DATE: 10-85
C ------------------------------------------------------------------
C VERSION NUMBER : 1           AMENDMENT DATE(S) :
C ------------------------------------------------------------------
C PURPOSE :          The routine is used to calculate a
C          weight vector * covariance matrix * weight vector product.
C          This product helps determine the SNR.
C
C ------------------------------------------------------------------
C                    PARAMETER DEFINITION
C  NAME          | TYPE   | CLASS| RANGE  |    DESCRIPTION
C ------------------------------------------------------------------
C                |        |      |        |
C NELMTS         |INT*4   |READ  |  <11   | # OF ANTENNA ELEMENTS
C SCALE1         |REAL    |READ  |        | VECTOR SCALE FACTOR
C SCALE2         |REAL    |READ  |        | MATRIX SCALE FACTOR
C COVMAT         |COMPLEX |READ  |        | SQUARE COVARIANCE MATRIX
C COMPWT         |COMPLEX |READ  |        | COMPLEX ANTENNA WEIGHTS
C PRODUCT        |COMPLEX |WRITE |        | VECTOR-MATRIX-VECTOR PRODUCT
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C ------------------------------------------------------------------
C              NON-LOCAL VARIABLES -- FILE DEFINITIONS
C ------------------------------------------------------------------
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C ------------------------------------------------------------------
C                  SUBROUTINES REQUIRED
C    NAME        |          DESCRIPTION
C ------------------------------------------------------------------
C                |
C                |
C                |
C                |
C                |
C                |
C                |
C ------------------------------------------------------------------
      subroutine matmult2(nelmts, A, B, scale1, scale2, product)
      implicit none
      integer*4 i, j, nelmts
```

```
      complex A(11,11),B(11),sum,sum2
      real scale1,scale2
      complex*16 product
*
*
      sum=cmplx(0.,0.)
      do i=1,nelmts
         sum2=cmplx(0.0,0.0)
         do j=1,nelmts
            sum2=scale1*conjg(B(j))*scale2*A(j,i) + sum2
         end do
         sum=sum2*scale1*B(i)+sum
      end do
*
      product = sum
      return
      end
```

```
C ------------------------------------------------------------------
C       TELECOMMUNICATION AND INFORMATION SCIENCES LABORATORY
C ------------------------------------------------------------------
C PROGRAM NAME: CONSTLMS       AUTHOR: D. HAEGERT       DATE: 10-85
C ------------------------------------------------------------------
C VERSION NUMBER : 1           AMENDMENT DATE(S) :
C ------------------------------------------------------------------
C PURPOSE :     This routine is a software implementation of a
C               constrained LMS adaptive algorithm using the
C               Takao narrowband method.
C
C ------------------------------------------------------------------
C                       PARAMETER DEFINITION
C  NAME          | TYPE    | CLASS| RANGE   |    DESCRIPTION
C ------------------------------------------------------------------
C NELMTS         |READ     |READ  | <11     | # OF ANTENNA ELEMENTS
C KS             |REAL     |READ  |         | CONVERGENCE CONSTANT
C WTDSUM         |COMPLEX  |READ  |         | ARRAY OUTPUT
C OUTPUT         |COMPLEX  |READ  |         | SENSOR ELEMENT OUTPUT
C COMPWT         |COMPLEX  |READ/W|         | COMPLEX ANTENNA WEIGHTS
C STRDEL         |COMPLEX  |READ  |         | STEERING DELAYS
C                |         |      |         |
C                |         |      |         |
C                |         |      |         |
C                |         |      |         |
C                |         |      |         |
C                |         |      |         |
C                |         |      |         |
C                |         |      |         |
C                |         |      |         |
C                |         |      |         |
C                |         |      |         |
C                |         |      |         |
C ------------------------------------------------------------------
C               NON-LOCAL VARIABLES -- FILE DEFINITIONS
C ------------------------------------------------------------------
C                |         |      |         |
C                |         |      |         |
C                |         |      |         |
C                |         |      |         |
C                |         |      |         |
C                |         |      |         |
C                |         |      |         |
C                |         |      |         |
C                |         |      |         |
C ------------------------------------------------------------------
C                     SUBROUTINES REQUIRED
C   NAME         |          DESCRIPTION
C ------------------------------------------------------------------
C                |
C                |
C                |
C                |
C                |
C                |
C ------------------------------------------------------------------
      subroutine constlms(nelmts, ks, wtdsum, output,
     &                    compwt, strdel)
      implicit none
```

```
      integer*4 j,nelmts,i,flag
      real ks,amp,phase
      complex compwt(nelmts),output(nelmts),wtdsum,xcon,conchk
      complex factor,strcon(11),strdel(nelmts),P(11,11)
      complex newwt(11),sum
      logical fcall
      data fcall/ true./
      save fcall
*
*     Initialize weights for first call
*
      if (fcall) then
          fcall =.false.
          flag=0
          do j=1,nelmts
             compwt(j)=strdel(j)/nelmts
             strcon(j)=conjg(strdel(j))
          end do
*
          do i=1,nelmts
             do j=1,nelmts
                P(i,j)=cmplx(0.0,0.0)
                if(i .eq. j) P(i,j)=cmplx(1.0,0.0)
                P(i,j)=P(i,j)-strdel(i)*strcon(j)/nelmts
             end do
          end do
      end if
*
*
      do i=1,nelmts
         sum=cmplx(0.0,0.0)
         do j=1,nelmts
            xcon=conjg(output(j))
            factor=ks*wtdsum*xcon
            sum=P(i,j)*(compwt(j)-factor) + sum
         end do
         compwt(i)=sum+strdel(i)/nelmts
       end do
*
*
* for testing purposes
      flag=flag+1
      if(mod(flag,1000) .eq. 0) then
      conchk=cmplx(0.0,0.0)
      do i=1,nelmts
         conchk=conjg(strdel(i))*compwt(i) + conchk
         amp=sqrt(real(compwt(i))**2 +aimag(compwt(i))**2)
         phase=atand(-aimag(compwt(i))/real(compwt(i)))
         if (real(compwt(i)) .lt. 0.0) phase=phase+180.0
         write(6,*)amp,'  ',phase
      end do
      write(6,*)' '
      write(6,*)'Constraint    ',conchk
      write(6,*)'power out = ',sqrt(real(wtdsum)**2)
      write(6,*)' '
*
      end if
      return
      end
```

```
C ------------------------------------------------------------------
C       TELECOMMUNICATION AND INFORMATION SCIENCES LABORATORY
C ------------------------------------------------------------------
C PROGRAM NAME: ANTARR          AUTHOR: ROGER SPOHN      DATE: 9/85
C ------------------------------------------------------------------
C VERSION NUMBER : 1.0          AMENDMENT DATE(S) :
C ------------------------------------------------------------------
C PURPOSE :     This program is used to simulate a planar antenna array.
C
C
C
C ------------------------------------------------------------------
C                       PARAMETER DEFINITION
C  NAME             | TYPE    | CLASS| RANGE   |     DESCRIPTION
C ------------------------------------------------------------------
C PHI               |REAL     |READ  |0-360    |ARRAY OF SIGNAL AZIMUTHS
C THETA             |REAL     |READ  |0-180    |ARRAY OF SIGNAL ELEVATIONS
C X                 |REAL     |READ  |         |ARRAY OF ELEMENT COORDINATES
C Y                 |REAL     |READ  |         |ARRAY OF ELEMENT COORDINATES
C SIGNAL            |COMPLEX  |READ  |         |ARRAY OF INCOMING SIGNAL VALU
C CFREQ             |REAL     |READ  |>0       |CARRIER FREQUENCY OF SIGNALS
C NANT              | INT     |READ  |1-11     |NUMBER OF ANTENNA ELEMENTS
C NSIG              | INT     |READ  |1-15     |NUMBER OF INCOMING SIGNALS
C COMPWT            |COMPLEX  |READ  |         |ELEMENT WEIGHTING FACTORS
C NOISE             |COMPLEX  |READ  |         |ARRAY OF ADDITIVE ELEMENT
C GAIN              |REAL     |READ  |         |ARRAY OF ANTENNA GAINS
C OUTPUT            |COMPLEX  |WRITE |         |OUTPUT SIGNAL ARRAY
C WTDSUM            |COMPLEX  |WRITE |         |SUMMED OUTPUT SIGNAL
C                   |         |      |         |
C                   |         |      |         |
C                   |         |      |         |
C                   |         |      |         |
C                   |         |      |         |
C                   |         |      |         |
C                   |         |      |         |
C ------------------------------------------------------------------
C               NON-LOCAL VARIABLES -- FILE DEFINITIONS
C ------------------------------------------------------------------
C                   |         |      |         |
C                   |         |      |         |
C                   |         |      |         |
C                   |         |      |         |
C                   |         |      |         |
C                   |         |      |         |
C                   |         |      |         |
C                   |         |      |         |
C ------------------------------------------------------------------
C                     SUBROUTINES REQUIRED
C    NAME           |          DESCRIPTION
C ------------------------------------------------------------------
C                   |
C                   |
C                   |
C                   |
C                   |
C                   |
C ------------------------------------------------------------------
      SUBROUTINE ANTARR(PHI, THETA, X, Y, SIGNAL, CFREQ, NANT, NSIG,
     &                  COMPWT, OUTPUT, WTDSUM, NOISE, STRDEL, gain)
      IMPLICIT NONE
```

```
      INTEGER*4 I,J,MXNSIG,NSIG,MXNANT,NANT
      PARAMETER (MXNANT=9,MXNSIG=15)
      COMPLEX SIGNAL(NSIG),OUTPUT(NANT),WTDSUM,COMPWT(NANT)
      COMPLEX IMGUNT,TEMP(MXNANT,MXNSIG),NOISE(11),STRDEL(11)
      COMPLEX TEMP2(MXNANT,MXNSIG)
      REAL PI,DTR,C,ANTLEN,PHI(NSIG),THETA(MXNSIG),CFREQ,TRASH
      REAL X(MXNANT),Y(MXNANT),XROT(MXNANT),GAIN(10),BETAL,SYNC
      REAL GFACT,AZI(11),ELV(11)
      PARAMETER(PI=3.141592654, C=299792458., ANTLEN=36.576)
      PARAMETER(DTR=PI/180.)
      LOGICAL FCALL
      DATA FCALL /.TRUE./
      SAVE FCALL,TEMP
C
C
C
      IF (FCALL) THEN
         FCALL=.FALSE.
C
C
         IMGUNT=(0.,1.)
C
C
         DO J=1,NSIG
C
         AZI(J)=PHI(J)*DTR
         ELV(J)=THETA(J)*DTR
C
C
C
            IF (J .EQ. 1) THEN
               IF (GAIN(J) .LT. 0.) THEN
                  SYNC=-1.
               ELSE
                  SYNC=1.
               END IF
            END IF
C
C
            DO I=1,NANT
               XROT(I)=Y(I)*SIN(AZI(J)) + X(I)*COS(AZI(J))
               TEMP(I,J)=GAIN(J)*SYNC*EXP((-2.*PI*IMGUNT*CFREQ*XROT(I)*
     &         SIN(ELV(J)))/C)
            END DO
         END DO
      END IF
C
C
      DO J=1,NSIG
         DO I=1,NANT
            TEMP2(I,J)=TEMP(I,J)*SIGNAL(J)
         END DO
      END DO
C
C
      DO I=1,NANT
         OUTPUT(I)=(0.,0.)
         DO J=1,NSIG
            OUTPUT(I)=OUTPUT(I) + TEMP2(I,J)
         END DO
```

```
      END DO
*
      WTDSUM=CMPLX(0.0,0.0)
      DO I=1,NANT
         WTDSUM=WTDSUM+COMPWT(I)*(NOISE(I)+OUTPUT(I))
      END DO
      RETURN
      END
```

```
C ------------------------------------------------------------------
C       TELECOMMUNICATION AND INFORMATION SCIENCES LABORATORY
C ------------------------------------------------------------------
C PROGRAM NAME: HFCHAN2        AUTHOR: Roger Yarbro     DATE: 8/85
C ------------------------------------------------------------------
C VERSION NUMBER : 1.0         AMENDMENT DATE(S) :
C ------------------------------------------------------------------
C PURPOSE :     To simulate an hf channel model.
C NOTES:      The desired signal should be the very first signal sent
C             thru the channel.
C             Due to program limitations, the maximum number of signals
C             that can be put thru the channel is equal to 20/nummod.
C ------------------------------------------------------------------
C                        PARAMETER DEFINITION
C  NAME             | TYPE  | CLASS| RANGE  |    DESCRIPTION
C ------------------------------------------------------------------
C invoke           |int*4  |trans |0 - 12  |invocation number
C nummod           |int*4  |read  |0 - 10  |number of modes of propagation
C insig            |int*4  |read  |        |input signal name
C outsig           |int*4  |read  |        |output signal name
C nyband           |real   |read  |        |Nyquist bandwidth of information
C                  |       |      |        |signal in kHz
C samprt           |real   |read  |        |sampling rate of output signal
C                  |       |      |        |in kHz
C pathdl(*)        |int*4  |read  |        |array of number which represent
C                  |       |      |        |the mode delays normalized to
C                  |       |      |        |1/W, where W is the Nyquist rate
C attenu(*)        |real   |read  |        |array of numbers which represent
C                  |       |      |        |the attenuation of each mode
C newtap           |logical|read  |        |if true then new taps are obtained
C                  |       |      |        |for the desired signal only
C                  |       |      |        |
C                  |       |      |        |
C                  |       |      |        |
C                  |       |      |        |
C ------------------------------------------------------------------
C              NON-LOCAL VARIABLES -- FILE DEFINITIONS
C ------------------------------------------------------------------
C                  |       |      |        |
C                  |       |      |        |
C                  |       |      |        |
C                  |       |      |        |
C                  |       |      |        |
C                  |       |      |        |
C                  |       |      |        |
C                  |       |      |        |
C                  |       |      |        |
C ------------------------------------------------------------------
C                     SUBROUTINES REQUIRED
C    NAME          |         DESCRIPTION
C ------------------------------------------------------------------
C tapdel           |tap delay line
C rannrm           |randum number generator
C                  |
C                  |
C                  |
C ------------------------------------------------------------------
      subroutine hfchannel(invoke,nummod,insig,outsig,nyband,
     &                     samprt, pathdl, attenu, newtap)
c
      implicit none
```

```
      integer*4  invoke, numtap, maxinv, maxmod, maxtap, numinv
      parameter  (maxinv = 10, maxtap = 4096, maxmod = 10)
      integer*4  iseed(4*maxmod+2), inv(maxinv), temp
      integer*4  nummod, pathdl(*), point, path, i, j
      real       nyband, samprt, realgn, imaggn, attenu(*)
      complex    tgain(0: (maxtap-1), maxinv), outsig, insig
      logical    tap(0: (maxtap-1)), newtap
      data (iseed(i), i=1, (4*maxmod+2)) /298495611, 384652321,
     &     456532321, 888775653, 664598313, 458741311, 874441031,
     &     545451211, 545612121, 215451133, 542132155, 551515457,
     &     986545455, 656544557, 165465447, 654654001, 968445151,
     &     545454569, 546554555, 654554545, 621555441, 999999999,
     &     654654561, 845556451, 984545133, 545451111, 885555555,
     &     545411211, 666553333, 484146547, 546541123, 545455677,
     &     898754577, 515465487, 546231257, 456231257, 653214787,
     &     525113251, 321321623, 871436567, 222929281, 393829111/
      data       (inv(i), i=1, maxinv) /maxinv*0/
      data       (tap(i), i=0, maxtap-1) /maxtap*.false./
c
      save numinv, inv, iseed, tgain, tap
c
      if (nummod.gt.maxmod) then
          print *, 'max number of modes exceeded in channel routine'
          stop
      end if
c
      if (invoke.eq.0) then
          numinv = numinv + 1
          if (numinv.eq.maxinv) then
              print *, 'max invokes exceeded in hf channel routine'
              stop
          end if
          invoke = numinv
c
c----------------Initialization done only on very first call----------------
c
c Since the number of taps required is equal to the largest delay entered,
c which is normalized to 1/W, determine the number of taps required.  Note,
c if the sampling rate of the output signal is greater than the Nyquist
c bandwidth of the information signal then the taps in between the taps
c which represent a delay of 1/W, need to be set to zero.
c
          if (numinv.eq.1) then
              do i=0, nummod-1
                    temp = int(samprt/nyband)
                    if (mod(samprt, nyband).ne.0) then
                        print 1
1                       format(//10x, 'Warning: an',
     &                                ' approximation has been ',
     &                                'made in which', /10x,
     &                                'taps to turn on.')
                    end if
                    temp = temp*pathdl(i+1)
                    tap(temp) = .true.
                    numtap = temp + 1
              end do
          end if
c
c
c-----------Initialixation done on subsequent initial invocations----------
```

```
c
            if (numinv.gt.1) then
                point = 2*nummod*(numinv - 1) + 1
                path = 1
                do i=0, numtap-1
                   if (tap(i).eq..true.) then
                       call rannrm(iseed(point), 0.0, 1.0, realgn)
                       call rannrm(iseed(point+1), 0.0, 1.0, imaggn)
                       tgain(i, invoke) = cmplx(realgn, imaggn)
                       tgain(i, invoke) = tgain(i, invoke)*
     &                                     (attenu(path)/1.414)
                       point = point + 2
                       path = path + 1
                   end if
                end do
            end if
        end if
c
c-------------------------Initialization complete---------------------------
c
        if (newtap) then
            point = 1
            path = 1
            do i=0, numtap-1
               if (tap(i).eq..true.) then
                   call rannrm(iseed(point), 0.0, 1.0, realgn)
                   call rannrm(iseed(point+1), 0.0, 1.0, imaggn)
                   tgain(i, invoke) = cmplx(realgn, imaggn)
                   tgain(i, invoke) = tgain(i, invoke)*
     &                (attenu(path)/1.414)
                   point = point + 2
                   path = path + 1
               end if
            end do
        end if
c
c
        call tapdel2(inv(invoke), insig, numtap, tgain(0, invoke),
     &               tap, outsig)
c
        return
        end
```

```
C ------------------------------------------------------------------
C       TELECOMMUNICATION AND INFORMATION SCIENCES LABORATORY
C ------------------------------------------------------------------
C PROGRAM NAME: ADAPTA          AUTHOR: Paul Snow        DATE: 1-8-85
C                                Adapted by D. Haegert
C ------------------------------------------------------------------
C VERSION NUMBER : 1            AMENDMENT DATE(S) :
C ------------------------------------------------------------------
C PURPOSE :    A point processing routine that simulates an adaptive
C array receiver.
C
C ------------------------------------------------------------------
C                      PARAMETER DEFINITION
C  NAME          | TYPE    | CLASS | RANGE    |    DESCRIPTION
C ------------------------------------------------------------------
C invoke         | int*4   |read/  | 0 - 4    |invocation number of calling
C                |         |modify |          |routine
C algtyp         | int*4   | read  | 1,2,3    |adaptive algorithm type
C                |         |       |          |1=lms
C                |         |       |          |2=constrained lms
C                |         |       |          |3=update covariance
C nelmts         | int*4   | read  | 2 - 11   |# elements in antenna array
C nsig           | int*4   | read  | 1 - 15   |# signal sources
C                |         |       |          |(communicator + jammers)
C carfrq         | real    | read  | > 0      |carrier frequency (Hz)
C sangaz(*)      | real    | read  | 0-360    |azimuth arrival angles of
C                |         |       |          |signals
C sangel(*)      | real    | read  | 0-180    |elevation arrival angles of
C                |         |       |          |signals
C ks             | real    | read  |  ?       |adaptive algorithm converg-
C                |         |       |          |ence constant
C x(*)           | real    | read  |   ?      |x coordinates of antenna
C                |         |       |          |elements
C y(*)           | real    | read  |   ?      |y coordinates of antenna
C                |         |       |          |elements
C sigin(*)       |complex  | read  | |x|<=1    |CLPE of signal sources, 1st
C                |         |       |          |address is communicator
C nsrat          | real    | read  |   ?      |thermal noise to signal ratio
C compwt(*)      |complex  | write |   ?      |adapted antenna weights
C strdel(*)      |complex  | read  |   ?      |antenna steering information
C                |         |       |          |for constrained lms
C code           | int*4   | read  |  -1,1    |bit of preamble code
C initupd        |logical  | read  | T, F     |if T, initialize update
C                |         |       |          |covariance algorithm
C lmsinit        |logical  | read  | T, F     |if T, initialize lms alg.
C gain(*)        |real     | read  |   ?      |antenna gains at signal
C                |         |       |          |arrival angles
C nsampb         | int*4   | read  |  >0      |number of samples/bit
C                |         |       |          |
C                |         |       |          |
C ------------------------------------------------------------------
C              NON-LOCAL VARIABLES -- FILE DEFINITIONS
C ------------------------------------------------------------------
C                |         |       |          |
C                |         |       |          |
C                |         |       |          |
C                |         |       |          |
C                |         |       |          |
C ------------------------------------------------------------------
C                    SUBROUTINES REQUIRED
```

```
C     NAME          !        DESCRIPTION
C ------------------------------------------------------------------
C ANTARR            !antenna array module
C LMS               !lms algorithm
C CONSTLMS          !constrained lms algorithm
C UPDCOVAR          !update covariance algorithm
C                   !
C ------------------------------------------------------------------
      subroutine adapta(invoke,algtyp,nelmts,nsig,
     1carfrq,sangaz,sangel,ks,x,y,sigin,
     &          nsrat,compwt,strdel,code,initupd,lmsinit,gain,nsampb)
      implicit none
      integer*4 invoke,nelmts,nsig,maxinv,nsampb
      real carfrq,sangaz(*),sangel(*)
      real x(nelmts),y(nelmts),ndev,nsrat,ks,gain(10)
      parameter (maxinv=4)
      integer*4 invk(3,maxinv),ninvk,i
      integer*4 iseed(22),point,algtyp,code
      real realgn,imaggn
      complex sigin(*),antsig(11),compwt(11),noise(11)
      complex prcsig,strdel(11)
      logical initupd,lmsinit
      data ninvk/0/
      data (iseed(i), i=1,22)/298495611,395710473,
     &  958194679,223085413,846294087,957773611,631049855,
     &  224719593,193753447,309759367,234652987,739294873,
     &  833957111,623429719,964197565,449658119,566520895,
     &  193393733,452945627,285628933,934655745,786345477/
      save invk,ninvk
      point=0
      if (invoke .eq. 0) then
         ninvk=ninvk+1
         invoke=ninvk
         do i=1,3
            invk(i,invoke)=0
         end do
      end if
*
      ndev=sqrt(nsrat/2)
        do i=1,nelmts
             point=point+2
             call rannrm(iseed(point-1),0.0,ndev,realgn)
             call rannrm(iseed(point),0.0,ndev,imaggn)
             noise(i)=cmplx(realgn,imaggn)
        end do
*
* Call submodules to process signals.
*
      call antarr(sangaz,sangel,x,y,sigin,carfrq,nelmts,
     &            nsig,compwt,antsig,prcsig,noise,strdel,gain)
      if (algtyp .eq. 1) then
      call lms(nelmts,compwt,prcsig,antsig,code,ks,lmsinit,nsampb)
      end if
      if (algtyp .eq. 2) then
      call constlms(nelmts,ks,prcsig,antsig,compwt,strdel)
      end if
      if (algtyp .eq. 3) then
      call updcovar(compwt,antsig,nelmts,initupd)
      end if
      return
```

```
end
```

```
C ------------------------------------------------------------------
C TELECOMMUNICATION AND INFORMATION SCIENCES LABORATORY, UNV. OF KANSAS
C ------------------------------------------------------------------
C PROGRAM NAME: DELAYR          AUTHOR: Paul Snow        DATE: 9/84
C ------------------------------------------------------------------
C VERSION NUMBER : 1.           AMENDMENT DATE(S) :
C ------------------------------------------------------------------
C PURPOSE :     This routine allows the user to delay data by some
C               specified amount.
C
C ------------------------------------------------------------------
C                       PARAMETER DEFINITION
C  NAME          | TYPE  | CLASS | RANGE  |    DESCRIPTION
C ------------------------------------------------------------------
C invoke         | int   | read  | >=0    |user invoke number
C                |       | /mod  |        |
C samdly         | int   | read  |0-4096  |sample delay
C insam          | real  | read  |  ?     |input sample
C outsam         | real  |write  |  ?     |delayed sample
C intout         | real  | read  |  ?     |initial queue output
C                |       |       |        |
C                |       |       |        |
C                |       |       |        |
C                |       |       |        |
C                |       |       |        |
C                |       |       |        |
C                |       |       |        |
C                |       |       |        |
C                |       |       |        |
C                |       |       |        |
C                |       |       |        |
C                |       |       |        |
C                |       |       |        |
C                |       |       |        |
C                |       |       |        |
C                |       |       |        |
C ------------------------------------------------------------------
C              NON-LOCAL VARIABLES -- FILE DEFINITIONS
C ------------------------------------------------------------------
C                |       |       |        |
C                |       |       |        |
C                |       |       |        |
C                |       |       |        |
C                |       |       |        |
C                |       |       |        |
C                |       |       |        |
C                |       |       |        |
C                |       |       |        |
C ------------------------------------------------------------------
C                    SUBROUTINES REQUIRED
C    NAME        |          DESCRIPTION
C ------------------------------------------------------------------
C                |
C                |
C                |
C                |
C                |
C                |
C ------------------------------------------------------------------
```

```
      subroutine delayr(invoke,samdly,insam,outsam,intout)
      implicit none
      integer*4 invoke,numinv,maxinv,samdly,maxdly,i,pntr
      data numinv/0/
      save numinv,memory,qpntr
      parameter (maxinv=10,maxdly=4097)
      real memory(maxdly,maxinv),insam,outsam,intout
      integer*4 qpntr(maxinv)
      if (samdly .le. 0) then
         outsam=insam
         return
      end if
      if (invoke .eq. 0) then
C
C     ERROR - too many invokes
C
         if (numinv .ge. maxinv) then
            print *,'too many callers using delayr'
            stop
         end if
C
C     ERROR - delay requested is too long
C
         if (samdly .gt. maxdly) then
            print *,'maximum delayr delay exceeded'
            stop
         end if
C
C     Initialize variables for new user
C
         numinv=numinv+1
         invoke=numinv
         qpntr(invoke)=1
         do i=1,samdly
            memory(i,invoke)=intout
         end do
      end if
C
C     Use circular queue to output delayed sample and store
C     new sample
C
      pntr=qpntr(invoke)
      outsam=memory(pntr,invoke)
      memory(pntr,invoke)=insam
      qpntr(invoke)=mod(pntr,samdly)+1
      return
      end
```

```
C ------------------------------------------------------------------
C       TELECOMMUNICATION AND INFORMATION SCIENCES LABORATORY
C ------------------------------------------------------------------
C PROGRAM NAME: hfpherror      AUTHOR: Roger Yarbro     DATE: 7/85
C ------------------------------------------------------------------
C VERSION NUMBER : 1.0         AMENDMENT DATE(S) :
C ------------------------------------------------------------------
C PURPOSE :     To determine the phase error of a signal sent thru an
C               hf channel routine.
C
C
C ------------------------------------------------------------------
C                       PARAMETER DEFINITION
C  NAME             | TYPE   | CLASS| RANGE   |    DESCRIPTION
C ------------------------------------------------------------------
C invoke           |int*4   |trans |0 - 20   |invocation number
C insig            |int*4   |read  |         |input signal name
C outsig           |int*4   |read  |         |output signal name
C nummod           |int*4   |trans |0 - 20   |number of modes of propag-
C                  |        |      |         |tion
C maxmod           |int*4   |trans |0 - 20   |maximum number of modes of
C                  |        |      |         |propagation
C nyband           |real    |trans |   ?     |Nyquist bandwidth of information
C                  |        |      |         |signal in kHz
C samprt           |real    |trans |   ?     |sampling rate of output signal
C                  |        |      |         |in kHz
C pathdl(maxmod)   |int     |trans |   ?     |array of numbers which represent
C                  |        |      |         |the mode delays normalized to
C                  |        |      |         |1/W, where W is the Nyquist rate
C attenu(maxmod)   |real    |trans |   ?     |array of numbers which represent
C                  |        |      |         |the attenuation of each mode
C newtap             logical mod               should be true when this routine
C                                              is called
C pherr            |real    |trans |0 - 2*pi |phase error in a signal sent
C                  |        |      |         |thru the hf channel routine
C ------------------------------------------------------------------
C              NON-LOCAL VARIABLES -- FILE DEFINITIONS
C ------------------------------------------------------------------
C                  |        |      |         |
C                  |        |      |         |
C                  |        |      |         |
C                  |        |      |         |
C                  |        |      |         |
C ------------------------------------------------------------------
C                     SUBROUTINES REQUIRED
C    NAME          |          DESCRIPTION
C ------------------------------------------------------------------
C hfchannel        | To simulate an hf channel model
C siglib           |
C                  |
C                  |
C                  |
C ------------------------------------------------------------------
c
c
c  This routine calculates the phase shift in a BPSK signal which is sent th
c  the hf channel routine.  The complex low pass equivalent of a BPSK signal
c  is either +1 at 0 degrees or +1 at 180 degrees.  This routine performs th
c  function of determining the phase shift in the signal and returning that
c  value in order that the calling routine can achieve synchronization.
```

```
c
c
c-------------------------------------------------------------------------
c
c
      subroutine hfpherror(invoke, nummod,
     &                     nyband, samprt, pathdl, attenu,
     &                     newtap, pherr)
c
c
      implicit        none
      integer*4       nummod, maxmod, lastap, invoke, i
      parameter       (maxmod=10)
      integer*4       pathdl(maxmod)
      real            attenu(maxmod)
      real            nyband, samprt, ipt, rpt, pherr, phase, pi
      parameter       (pi = 3.14159)
      complex         insig, outsig
      logical         newtap
c
c
      do i=1, nummod
              lastap = pathdl(i)*int(samprt/nyband) + 1
      end do
c
      insig = (1.0, 0.0)
      do i=1, lastap
              call hfchannel(invoke, nummod, insig, outsig, nyband,
     &                       samprt, pathdl, attenu, newtap)
              newtap = .false.
      end do
c
      rpt = real(outsig)
      ipt = aimag(outsig)
      if ((rpt.eq.0.).and.(ipt.eq.0.)) then
              phase = 0.0
      else
              phase = atan2(ipt, rpt)
      end if
c
      if (phase.lt.0) phase = phase + 2*pi
      pherr = phase
c
      insig = (0.0, 0.0)
      do i=1, lastap
              call hfchannel(invoke, nummod, insig, outsig, nyband,
     &                       samprt, pathdl, attenu, newtap)
      end do
c
      return
      end
```

```
C ------------------------------------------------------------------
C       TELECOMMUNICATION AND INFORMATION SCIENCES LABORATORY
C ------------------------------------------------------------------
C PROGRAM NAME: LMS            AUTHOR: D. HAEGERT      DATE: 12-85
C ------------------------------------------------------------------
C VERSION NUMBER : 1           AMENDMENT DATE(S) :
C ------------------------------------------------------------------
C PURPOSE :     This routine is an implementation of an Least Mean
C               Square adaptive algorithm.
C
C
C ------------------------------------------------------------------
C                     PARAMETER DEFINITION
C  NAME           | TYPE   | CLASS| RANGE  |    DESCRIPTION
C ------------------------------------------------------------------
C NELMTS          |INT*4   |READ  | <11    | # OF ANTENNA ELEMENTS
C CODE            |INT*4   |READ  | 1,-1   | PN PREAMBLE CODE
C KS              |REAL    |READ  |        | CONVERGENCE CONSTANT
C WTDSUM          |COMPLEX |READ  |        | ARRAY OUTPUT
C OUTPUT          |COMPLEX |READ  |        | SENSOR ELEMENT OUTPUTS
C COMPWT          |COMPLEX | R/W  |        | ANTENNA WEIGHTS
C LMSINIT         |LOGICAL |READ  |        | SIGNIFIES START-UP
C NSAMPB          |INT*4   |PASS  |        | NUMBER OF SAMPLES/BIT
C                 |        |      |        |
C                 |        |      |        |
C                 |        |      |        |
C                 |        |      |        |
C                 |        |      |        |
C                 |        |      |        |
C                 |        |      |        |
C                 |        |      |        |
C                 |        |      |        |
C                 |        |      |        |
C                 |        |      |        |
C                 |        |      |        |
C ------------------------------------------------------------------
C             NON-LOCAL VARIABLES -- FILE DEFINITIONS
C ------------------------------------------------------------------
C                 |        |      |        |
C                 |        |      |        |
C                 |        |      |        |
C                 |        |      |        |
C                 |        |      |        |
C                 |        |      |        |
C                 |        |      |        |
C                 |        |      |        |
C                 |        |      |        |
C ------------------------------------------------------------------
C                   SUBROUTINES REQUIRED
C    NAME         |        DESCRIPTION
C ------------------------------------------------------------------
C DELAYR          | DELAYS PNCODE TO SERVE AS REFERENCE
C                 |
C                 |
C                 |
C                 |
C                 |
C ------------------------------------------------------------------
      subroutine lms(nelmts, compwt, wtdsum, output, code, ks, lmsinit,
     &               nsampb)
```

```
      implicit none
      integer*4 i,nelmts,code,delinv,flag,nsampb
      real ks,incode,outcode,strsig,amp,phase
      complex compwt(11),wtdsum,output(11),error,refsig
      logical lmsinit,fcall
      data fcall/.true./
      save flag
*
      if (fcall) then
          fcall=.false.
          delinv=0
      end if
*
      if (lmsinit) then
c         delinv=0
         flag=0
         lmsinit=.false.
      end if
*
      flag=flag+1
      incode=float(code)
      call delayr(delinv,nsampb/4,incode,outcode,-1.0)
      refsig=cmplx(outcode,0.0)
c       refsig=cmplx(incode,0.0)
c     *
c     *
      error= refsig - wtdsum
*
      do i=1,nelmts
         compwt(i)=compwt(i) + ks*error*conjg(output(i))
      end do
*
c      if (mod(flag-1,1000) .eq. 0) then
c        do i=1,nelmts
c        amp=cabs(compwt(i))
c        phase=atand(-aimag(compwt(i))/real(compwt(i)))
c        if(real(compwt(i)) .lt. 0.0) phase=phase+180.0
c        write(6,*) i,amp,' ',phase
c        end do
c      end if
      return
      end
```

```
C ------------------------------------------------------------------
C       TELECOMMUNICATION AND INFORMATION SCIENCES LABORATORY
C ------------------------------------------------------------------
C PROGRAM NAME: MATINV          AUTHOR: D. HAEGERT       DATE: 10/85
C                               (adapted from thesis
C                                by Michael Brobston)
C ------------------------------------------------------------------
C VERSION NUMBER : 1.           AMENDMENT DATE(S) :
C ------------------------------------------------------------------
C PURPOSE :     This routine implements a matrix inversion.
C
C
C
C ------------------------------------------------------------------
C                       PARAMETER DEFINITION
C  NAME           | TYPE   | CLASS| RANGE   |     DESCRIPTION
C ------------------------------------------------------------------
C nelmts         | int    |read  |2 - 11   |matrix size, i.e. for a 3x3,
C                                             nelmts=3
C B              | real   |read  |  ?      |input matrix
C C              | real   |write |  ?      |inverted matrix
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C ------------------------------------------------------------------
C              NON-LOCAL VARIABLES -- FILE DEFINITIONS
C ------------------------------------------------------------------
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C                |        |      |         |
C ------------------------------------------------------------------
C                    SUBROUTINES REQUIRED
C    NAME        |          DESCRIPTION
C ------------------------------------------------------------------
C                |
C                |
C                |
C                |
```

```
C                 !
C                 !
C ----------------------------------------------------------------------
      subroutine matinv(nelmts,B,C)
      implicit none
      integer*4 nelmts,n1,n2,jerr,i,j,k,j1,k1,np1,l
      complex A(11,22),B(11,11),C(11,11),ca
*
      n1=nelmts-1
      n2=2*nelmts
      jerr=1
      do 20 i=1,nelmts
      do 10 j=1,nelmts
      A(i,j)=B(i,j)
      j1=j+nelmts
   10 A(i,j1)=cmplx(0.,0.)
      j1=i+nelmts
   20 A(i,j1)=cmplx(1.,0.)
      do 100 k=1,n1
      ca=A(k,k)
      if (cabs(ca) - .00000001) 3,3,5
    5 k1=k+1
      do 6 j=k1,n2
    6 A(k,j)=A(k,j)/ca
      do 100 i=k1,nelmts
      ca=A(i,k)
      do 100 j=k1,n2
  100 A(i,j)=A(i,j)-ca*A(k,j)
      np1=nelmts+1
      if (cabs(A(nelmts,nelmts))-.00000001) 3,3,19
   19 do 21 j=np1,n2
   21 A(nelmts,j)=A(nelmts,j)/A(nelmts,nelmts)
      do 175 l=1,n1
      k=nelmts-l
      k1=k+1
      do 175 i=np1,n2
      do 175 j=k1,nelmts
  175 A(k,i)=A(k,i)-A(k,j)*A(j,i)
      do 180 i=1,nelmts
      do 180 j=1,nelmts
      j1=j+nelmts
  180 C(i,j)=A(i,j1)
      go to 190
    3 jerr=0
  190 continue
*
      return
      end
```

```
C ------------------------------------------------------------------
C       TELECOMMUNICATION AND INFORMATION SCIENCES LABORATORY
C ------------------------------------------------------------------
C PROGRAM NAME: MATMULT        AUTHOR: D. HAEGERT       DATE: 10-85
C ------------------------------------------------------------------
C VERSION NUMBER : 1           AMENDMENT DATE(S) :
C ------------------------------------------------------------------
C PURPOSE :     To calculate the optimum SNR ratio for an adaptive
C               array.
C
C
C ------------------------------------------------------------------
C                      PARAMETER DEFINITION
C  NAME            | TYPE   | CLASS| RANGE  |    DESCRIPTION
C ------------------------------------------------------------------
C NELMTS           |INT*4   |READ  |  <11   | # OF ANTENNA ELEMENTS
C SCALE1           |REAL    |READ  |        | VECTOR SCALE FACTOR
C SCALE2           |REAL    |READ  |        | MATRIX SCALE FACTOR
C COVMAT           |COMPLEX |READ  |        | INVERSE COVARIANCE MATRIX
C DESIG            |COMPLEX |READ  |        | SIGNAL VECTOR
C PRODUCT          |COMPLEX |WRITE |        | OPTIMUM SNR
C                  |        |      |        |
C                  |        |      |        |
C                  |        |      |        |
C                  |        |      |        |
C                  |        |      |        |
C                  |        |      |        |
C                  |        |      |        |
C                  |        |      |        |
C                  |        |      |        |
C                  |        |      |        |
C                  |        |      |        |
C                  |        |      |        |
C ------------------------------------------------------------------
C              NON-LOCAL VARIABLES -- FILE DEFINITIONS
C ------------------------------------------------------------------
C                  |        |      |        |
C                  |        |      |        |
C                  |        |      |        |
C                  |        |      |        |
C                  |        |      |        |
C                  |        |      |        |
C                  |        |      |        |
C                  |        |      |        |
C                  |        |      |        |
C ------------------------------------------------------------------
C                    SUBROUTINES REQUIRED
C    NAME          |        DESCRIPTION
C ------------------------------------------------------------------
C                  |
C                  |
C                  |
C                  |
C                  |
C                  |
C ------------------------------------------------------------------
      subroutine matmult(nelmts,A,B,scale1,scale2,product)
      implicit none
      integer*4 i,j,nelmts
```

```
      complex A(11,11),B(11),sum,sum2,product
      real scale1,scale2
*
*
      sum=cmplx(0.,0.)
      do i=1,nelmts
         sum2=cmplx(0.0,0.0)
         do j=1,nelmts
            sum2=scale1*B(j)*scale2*A(j,i) + sum2
         end do
         sum=sum2*scale1*conjg(B(i))+sum
      end do
*
      product = sum
      return
      end
```

```
C ------------------------------------------------------------------
C       TELECOMMUNICATION AND INFORMATION SCIENCES LABORATORY
C ------------------------------------------------------------------
C PROGRAM NAME: STAT            AUTHOR: ROGER L. SPOHN  DATE: 11/85
C ------------------------------------------------------------------
C VERSION NUMBER : 1.           AMENDMENT DATE(S) : /
C ------------------------------------------------------------------
C PURPOSE :     To compute the mean and variance of a set of data.
C
C
C
C ------------------------------------------------------------------
C                      PARAMETER DEFINITION
C  NAME          | TYPE  | CLASS| RANGE   |     DESCRIPTION
C ------------------------------------------------------------------
C DATA           |REAL   |READ  |  ?      |Input data
C DATLEN         |INT    |READ  |2 - 100  |Number of input points
C MEAN           |REAL   |WRITE |  ?      |Mean of input data
C VAR            |REAL   |WRITE |  ?      |Variance of input data
C                |       |      |         |
C                |       |      |         |
C                |       |      |         |
C                |       |      |         |
C                |       |      |         |
C                |       |      |         |
C                |       |      |         |
C                |       |      |         |
C                |       |      |         |
C                |       |      |         |
C                |       |      |         |
C                |       |      |         |
C                |       |      |         |
C                |       |      |         |
C                |       |      |         |
C                |       |      |         |
C                |       |      |         |
C                |       |      |         |
C ------------------------------------------------------------------
C             NON-LOCAL VARIABLES -- FILE DEFINITIONS
C ------------------------------------------------------------------
C                |       |      |         |
C                |       |      |         |
C                |       |      |         |
C                |       |      |         |
C                |       |      |         |
C                |       |      |         |
C                |       |      |         |
C                |       |      |         |
C                |       |      |         |
C ------------------------------------------------------------------
C                    SUBROUTINES REQUIRED
C    NAME        |          DESCRIPTION
C ------------------------------------------------------------------
C                |
C                |
C                |
C                |
C                |
C                |
C                |
C ------------------------------------------------------------------
```

```
      SUBROUTINE STAT(DATA,DATLEN,MEAN,VAR)
      INTEGER DATLEN,I,MXLEN
      REAL MEAN,VAR,SUM
      PARAMETER(MXLEN=100)
      REAL DATA(MXLEN)
C
      SUM=0.
      DO I=1,DATLEN
         SUM=SUM + DATA(I)
      END DO
      MEAN=SUM/DATLEN
C
      VAR=0.
      DO I=1,DATLEN
         VAR=VAR + (DATA(I)-MEAN)**2.
      END DO
      var=var/(datlen-1)
      RETURN
      END
```

```
C ------------------------------------------------------------------------
C       TELECOMMUNICATION AND INFORMATION SCIENCES LABORATORY
C ------------------------------------------------------------------------
C PROGRAM NAME: POWER          AUTHOR: Roger, Roger, Darin DATE: 10/85
C ------------------------------------------------------------------------
C VERSION NUMBER : 1.0         AMENDMENT DATE(S) :
C ------------------------------------------------------------------------
C PURPOSE :     To calculate the output power in a signal which goes thru
C               a HF channel antenna array.
C
C
C ------------------------------------------------------------------------
C                       PARAMETER DEFINITION
C  NAME          | TYPE   | CLASS| RANGE  |     DESCRIPTION
C ------------------------------------------------------------------------
C inpow          |real    |read  |        |input signal power
C attenu(*)      |real    |read  |        |array containing the
C                |        |      |        |          |attenuators
C                |        |      |        |factors in the HF channel
C nummod         |int*4   |read  |        |number of taps in HF model
C theta          |real    |read  |        |elevation angle
C antlen         |real    |read  |        |antenna lengh in meters
C carfrq         |real    |read  |        |carrier frequency in kHz
C outpow         |real    |mod   |        |output signal power
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C ------------------------------------------------------------------------
C             NON-LOCAL VARIABLES -- FILE DEFINITIONS
C ------------------------------------------------------------------------
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C                |        |      |        |
C ------------------------------------------------------------------------
C                   SUBROUTINES REQUIRED
C   NAME         |          DESCRIPTION
C ------------------------------------------------------------------------
C   rannrm       | Gaussian random number generator
C                |
C                |
C                |
C                |
C                |
C                |
C ------------------------------------------------------------------------
      subroutine power(inpow, attenu, nummod, sigain, antlen,
```

```
     &                        carfrq, signal, outpow)
C
C
          implicit      none
         integer*4     nummod, iseed(10), point, i
         real          inpow, outpow, sigain, antlen, attenu(*)
         real          sum, carfrq, pi, rpt, ipt, mag
         parameter         (pi = 3.141592654)
         complex       tgain
         logical signal
         data (iseed(i), i=1, 10) /298495611, 384652321, 456532321,
     &       888775653, 664598313, 458741311, 874441031,
     &       545451211, 545612121, 215451133/
C
C
         save iseed
         sum = 0.
C
         point = 0
         do i=1, nummod
            if (signal) then
                 call rannrm(iseed(point+1), 0.0, 1.0, rpt)
                 call rannrm(iseed(point+2), 0.0, 1.0, ipt)
                 tgain = cmplx(rpt, ipt)
                 mag = cabs(tgain)
            else
                 mag=1.0
            end if
                 sum = sum +  attenu(i)*mag
                 point = point + 2
         end do
C
         outpow = inpow * sum
C
         return
         end
```

```
C ------------------------------------------------------------------
C       TELECOMMUNICATION AND INFORMATION SCIENCES LABORATORY
C ------------------------------------------------------------------
C PROGRAM NAME: SNCOVAR        AUTHOR: D. HAEGERT      DATE: 10-85
C ------------------------------------------------------------------
C VERSION NUMBER : 1           AMENDMENT DATE(S) :
C ------------------------------------------------------------------
C PURPOSE :     To calculate the signal and noise covariance matrices
C               necessary to determine an optimum SNR performance
C               measure.
C
C ------------------------------------------------------------------
C                       PARAMETER DEFINITION
C  NAME          | TYPE   | CLASS | RANGE  |    DESCRIPTION
C ------------------------------------------------------------------
C NSIG          |INT*4   |READ   |  <10   | # OF SIGNALS AT ARRAY
C NELMTS        |INT*4   |READ   |  <11   | # OF ANTENNA ELEMENTS
C PHI           |REAL    |READ   | 0-360  | ARRAY OF AZIMUTH ANGLES
C THETA         |REAL    |READ   | 0-90   | ARRAY OF ELEVATION ANGLES
C X             |REAL    |READ   |        | ARRAY OF ELEMENT LOCATIONS
C Y             |REAL    |READ   |        | ARRAY OF ELEMENT LOCATIONS
C CARFRQ        |REAL    |READ   |        | CARRIER FREQUENCY
C JAMPOW        |REAL    |READ   |        | ARRAY OF JAMMER POWERS
C NSRAT         |REAL    |READ   |        | NOISE-TO-SIGNAL RATIO
C DESIG         |COMPLEX |WRITE  |        | SIGNAL VECTOR
C RSS           |COMPLEX |WRITE  |        | COMM. SIGNAL COVARIANCE MATR
C RNN           |COMPLEX |WRITE  |        | NOISE COVARIANCE MATRIX
C               |        |       |        |
C               |        |       |        |
C               |        |       |        |
C               |        |       |        |
C               |        |       |        |
C               |        |       |        |
C               |        |       |        |
C               |        |       |        |
C ------------------------------------------------------------------
C               NON-LOCAL VARIABLES -- FILE DEFINITIONS
C ------------------------------------------------------------------
C               |        |       |        |
C               |        |       |        |
C               |        |       |        |
C               |        |       |        |
C               |        |       |        |
C               |        |       |        |
C               |        |       |        |
C               |        |       |        |
C               |        |       |        |
C ------------------------------------------------------------------
C                   SUBROUTINES REQUIRED
C   NAME        |        DESCRIPTION
C ------------------------------------------------------------------
C               |
C               |
C               |
C               |
C               |
C               |
C ------------------------------------------------------------------
      subroutine sncovar(nsig, nelmts, x, y, phi, theta, jampow,
     &                   nsrat, carfrq, Rss, Rnn, desig)
      implicit none
```

```
      integer*4 i,j,k,m,nsig,nelmts
      real dtr,azi,elv,carfrq,pi,c,rotax
      real theta(3),phi(3),x(11),y(11),jampow(5),nsrat
      complex IMT,Rss(11,11),Rnn(11,11),sig(3,11)
      complex desig(11),Rnnint,Rn
      parameter (pi=3.14159265, c=3.e8, dtr=0.01745329)
      IMT=cmplx(0.,1.0)
*
*
*
      do i=1,nsig
         azi=phi(i)*dtr
         elv=theta(i)*dtr
         do j=1,nelmts
            rotax=y(j)*sin(azi)+x(j)*cos(azi)
            sig(i,j)=exp(-2.*pi*IMT*rotax*sin(elv)
     &                  *carfrq/c)
         end do
      end do
*
      do i=1,nelmts
         desig(i)=sig(1,i)
         do j=1,nelmts
            Rss(i,j)=conjg(sig(1,i))*sig(1,j)
            Rnnint=cmplx(0.,0.)
            do k=1,nsig-1
            m=k+1
              Rnnint=conjg(sig(m,i))*sig(m,j)*jampow(k) + Rnnint
            end do
            Rn=cmplx(0.,0.)
            if (i .eq. j) Rn=cmplx(nsrat,0.)
            Rnn(i,j)=Rnnint+Rn
         end do
      end do
*
      return
      end
```

```
! ------------------------------------------------------------------------
! TELECOMMUNICATION AND INFORMATION SCIENCES LABORATORY, UNV. OF KANSAS
! ------------------------------------------------------------------------
! PROGRAM NAME: Delete_experimAUTHOR: Brian McClendon DATE: 3/20/86
! ------------------------------------------------------------------------
! VERSION NUMBER : 1.0          AMENDMENT DATE(S) :
! ------------------------------------------------------------------------
! PURPOSE :     to delete a given experiment including all created
! files and the directory.
!
! ------------------------------------------------------------------------
!                      PARAMETER DEFINITION
!  NAME             ! TYPE   ! CLASS! RANGE  !     DESCRIPTION
! ------------------------------------------------------------------------
! p1                !text    !      !         !the experiment name
!                   !        !      !         !
! ------------------------------------------------------------------------
!                      SUBROUTINES REQUIRED
!    NAME           !          DESCRIPTION
! ------------------------------------------------------------------------
! none              !
!                   !
! ------------------------------------------------------------------------

$ if (p1.nes."") then goto OK
$ AGAIN: inquire p1 "Which experiment to delete?"
$ if (p1.eqs."") then goto END
$ OK: p1="EXPERIMENT_"+p1
$ if ((f$search(p1+".dir")).eqs."")  then goto NOTFOUND
$ dirname="[."+p1+"]"
$ set def 'dirname'
$ delete *.*.*
$ set def [-]
$ p1=p1+".dir.*"
$ set prot=(owner:rwed) 'p1'
$ delete 'p1'
$ END: exit
$ NOTFOUND: write sys$output "The experiment was not found from this directory,"
$ write sys$output "Try again."
$ goto AGAIN
```

```
! ------------------------------------------------------------------------
! TELECOMMUNICATION AND INFORMATION SCIENCES LABORATORY, UNV. OF KANSAS
! ------------------------------------------------------------------------
! PROGRAM NAME: Input.com      AUTHOR: Brian McClendon DATE: 3/20/86
! ------------------------------------------------------------------------
! VERSION NUMBER : v1.0        AMENDMENT DATE(S) :
! ------------------------------------------------------------------------
! PURPOSE :    This command file will prompt the user for a experiment
! name(or take one as an argument), create a directory with the name:
! .EXPERIMENT_'name', run the parameter input program, and submit a
! batch file which runs the main number crunching routines.  It then
! exits and puts the user back in the previous directory.
! ------------------------------------------------------------------------
!                      PARAMETER DEFINITION
!  NAME           ! TYPE   ! CLASS! RANGE  !     DESCRIPTION
! ------------------------------------------------------------------------
! p1              !string  !       !        !the name of the experiment
!                 !        !       !        !
!                 !        !       !        !
! ------------------------------------------------------------------------
!                    SUBROUTINES REQUIRED
!    NAME         !          DESCRIPTION
! ------------------------------------------------------------------------
! input.exe       !the FORTRAN program that prompts the user for various
!                  parameters required to calculated the convergences.
! main.com        !the command file to be submitted in batch which has
!                  the time consuming FORTRAN routines.
!                 !
! ------------------------------------------------------------------------
$ if (p1.nes."") then goto OK
$ inquire p1  "What is the name of this experiment?"
$ if (p1.eqs."") then goto END
$ OK: p1="EXPERIMENT_"+p1
$ dirname="[."+p1+"]"
$ on error then continue
$ create/dir 'dirname'
$ set prot=(w:rewd)/default
$ copy [.paramdir]*.* 'dirname'
$ define/user sys$input sys$command
$ run [.input]input.exe
$ copy params.dat 'dirname'
$ dirname=f$directory()
$ dirname=f$extract(0,(f$locate("]",f$directory())),dirname)
$ p1=dirname+"."+p1+"]"
$ submit/notify/noprint main.com/parameters=('p1')
$ END: set prot=(w:rwe)/default
$  exit
```

```
! ------------------------------------------------------------------------
! TELECOMMUNICATION AND INFORMATION SCIENCES LABORATORY, UNV. OF KANSAS
! ------------------------------------------------------------------------
! PROGRAM NAME: Output.com     AUTHOR: Brian McClendon DATE: 3/20/86
! ------------------------------------------------------------------------
! VERSION NUMBER : 1.0         AMENDMENT DATE(S) :
! ------------------------------------------------------------------------
! PURPOSE :     This command file is used to view the output of the
! simulation that was run using input.com.  It takes an experiment
! name and then goes into the appropriate directory and executes the
! input.exe file which shows all of the data created by the simulation.
! ------------------------------------------------------------------------
!                      PARAMETER DEFINITION
!  NAME            ! TYPE   ! CLASS! RANGE  !     DESCRIPTION
! ------------------------------------------------------------------------
! p1               !text    !      !         !experiment name
!                  !        !      !         !
! ------------------------------------------------------------------------
!                     SUBROUTINES REQUIRED
!    NAME          !         DESCRIPTION
! ------------------------------------------------------------------------
! input.exe        !shows the data files to the user in a readable format
!                  !
! ------------------------------------------------------------------------

$ if (p1.nes."") then goto OK
$ AGAIN: inquire p1 "For which experiment do you want to see the output?"
$ if (p1.eqs."") then goto END
$ OK: p1="EXPERIMENT_"+p1
$ if ((f$search(p1+".dir")).eqs."") then goto  NOTFOUND
$ p1="[."+p1+"]"
$ set def 'p1'
$ set prot=(w:rwed)/default
$ write sys$output "Switch to Tek 4014 mode now!"
$ write sys$output "(For the HP, hit F3)"
$ define/user sys$input sys$command
$ run [-.output]outputmain.exe
$ set def [-]
$ END: set prot=(w:rwe)/default
$  exit
$ NOTFOUND: write sys$output "No such experiment was found from this directory,"
$ write sys$output "Try again."
$ goto AGAIN
```

```
! ------------------------------------------------------------------
! TELECOMMUNICATION AND INFORMATION SCIENCES LABORATORY, UNV. OF KANSAS
! ------------------------------------------------------------------
! PROGRAM NAME: MAIN.COM       AUTHOR: Brian McClendon DATE: 3/20/86
! ------------------------------------------------------------------
! VERSION NUMBER : 1.0         AMENDMENT DATE(S) :
! ------------------------------------------------------------------
! PURPOSE :     This command file is run in batch mode to execute the
! main simulation program.  It also uses the experiment name for
! the directory name when files are created.
!
! ------------------------------------------------------------------
!                      PARAMETER DEFINITION
!  NAME           ! TYPE  ! CLASS! RANGE  !    DESCRIPTION
! ------------------------------------------------------------------
! p1              !       !      !        !the experiment name
!                 !       !      !        !
! ------------------------------------------------------------------
!                    SUBROUTINES REQUIRED
!    NAME         !         DESCRIPTION
! ------------------------------------------------------------------
! main.exe        !the actual simulation data creator
!                 !
! ------------------------------------------------------------------

$ set default 'p1'
$ set prot=(w:rwed)/default
$ run [-.main]main.exe
$ set prot=(w:rwe)/default
$ EXIT
```

```
C ------------------------------------------------------------------------
C TELECOMMUNICATION AND INFORMATION SCIENCES LABORATORY, UNV. OF KANSAS
C ------------------------------------------------------------------------
C PROGRAM NAME: antenna_params.for AUTHOR: Brian McClendonDATE: 3/20/86
C ------------------------------------------------------------------------
C VERSION NUMBER : 1.0          AMENDMENT DATE(S) :
C ------------------------------------------------------------------------
C PURPOSE :     this routine will simply show the parameters associated
C with the antenna in a table.  When <CR> is hit, it returns to the
C mainline.
C
C ------------------------------------------------------------------------
C                       PARAMETER DEFINITION
C  NAME            | TYPE  | CLASS| RANGE  |     DESCRIPTION
C ------------------------------------------------------------------------
C none             |       |      |        |
C                  |       |      |        |
C ------------------------------------------------------------------------
C                     SUBROUTINES REQUIRED
C    NAME          |          DESCRIPTION
C ------------------------------------------------------------------------
C none             |
C ------------------------------------------------------------------------

      subroutine antenna_params

      integer  type
      integer i,n
      real x(50),y(50),a(50),alpha(50),carfrq,antenna_length
      real DTR

      DTR=3.14159276/180.0

C ** Read parameter data file for useful default values
      open(unit=11,file='antennaparams2.dat',status='old')
      rewind 11
      read(11,*,end=220) N
      do I=1,N
       read(11,*,end=220) x(I)
       read(11,*,end=220) y(I)
       read(11,*,end=220) A(I)
       read(11,*,end=220) alpha(I)
       alpha(I)=alpha(I)*DTR
      enddo
      read(11,*,end=220) carfrq
      read(11,*,end=220)
      read(11,*,end=220) antenna_length
220   close(unit=11)



      call initt(120)
      write(6,221)
221   format(1x,'      Antenna Element Parameters')
      write(*,*)
      write(6,222)
222   format
     &(1x,'element      x       y       amplitude     phase(in radians)')
      do i=1,n
```

```
         write(6,223) i,x(i),y(i),a(i),alpha(i)
223      format(3x,I2,6x,f6.2,1x,f6.2,5x,f6.2,6x,f6.2)
      enddo
      write(*,*)
      write(*,224) carfrq
224   format(' Carrier Frequency (in Hz) : ',F12.1)
      write(*,225) antenna_length
225   format(' Antenna Length (in meters): ',F12.1)
      read(*,*)
      end
```

```
C ------------------------------------------------------------------
C TELECOMMUNICATION AND INFORMATION SCIENCES LABORATORY, UNV. OF KANSAS
C ------------------------------------------------------------------
C PROGRAM NAME: plotter.for    AUTHOR: Darren Haegert  DATE: unknown
C ------------------------------------------------------------------
C VERSION NUMBER : 1.5         AMENDMENT DATE(S) : 3/20/86(BAM)
C ------------------------------------------------------------------
C PURPOSE :     this program uses PLOT10 to plot X-Y graphs of data.
C It loads in the graph parameters from a .DFT file.
C It was modified to be a subroutine with one parameter which specified
C which of three graphs were to be plotted.
C ------------------------------------------------------------------
C                     PARAMETER DEFINITION
C  NAME           ! TYPE   ! CLASS! RANGE   !     DESCRIPTION
C ------------------------------------------------------------------
C graph           !integer!       !1..3     !the graph type
C                 !        !      !         !
C ------------------------------------------------------------------
C                     SUBROUTINES REQUIRED
C    NAME         !          DESCRIPTION
C ------------------------------------------------------------------
C PLOT10          !various routines
C                 !
C ------------------------------------------------------------------

      subroutine plotter(graph)
      implicit none
      integer graph
      character*25 xfil,yfil1,yfil2,yfil3,xtypec,titlec,xlab1,ylab1
      character*25 lab1c,lab2c,lab3c,ytypec,point,conect
      character*1 dummy,charac
      character*25 dftfil
      real loc1x,loc1y,loc2x,loc2y,loc3x,loc3y,numy,xminc,xmaxc
      real yminc,ymaxc,comget,rms1,rms2,rms3
      integer*4 tlen,xlen,ylen,len1,len2,len3,i,numin
      integer*4 ibasex,ibasey,ibasec,inumy,numpts
      integer*4 astit(72),asx(72),asy(72),as1(72),as2(72),as3(72)
      parameter(numpts=300,numin=901)
      real y1ary(numin),y2ary(numin),y3ary(numin),xmin,xmax,ymin,ymax
      real xarray(numin),uxmin,uxmax,uymin,uymax,minx
      real xout(numin),y1out(numin),y2out(numin),y3out(numin)
      logical dflag,s1,s2,s3,con1,con2,con3
      character*30 dftfile
c
c

      if (graph.eq.1)  dftfile='histogram.dft'
      if (graph.eq.2)  dftfile='signalnoise.dft'
      if (graph.eq.3)  dftfile='errorsignal.dft'
      open(unit=12,file=dftfile,status='old')
      rewind 12
      READ (12,10) xfil
      READ (12,20) numy
      READ (12,10) yfil1,yfil2,yfil3,xtypec,ytypec,titlec,
     &    xlab1,ylab1,lab1c
      READ (12,20) loc1x,loc1y
      READ (12,10) lab2c
      READ (12,20) loc2x,loc2y
      READ (12,10) lab3c
```

```
      READ (12,20) loc3x,loc3y
      read (12,10)point
      read (12,10)conect
      read (12,20)rms1,rms2,rms3
10    FORMAT( A25 )
20    FORMAT(6F20.6)
      close(unit=12)
c
c determine which (if any) outputs are to have data pts marked
c
        call extrct(point,s1,s2,s3)
c
c  determine which plots are connected
c
        call extrct(conect,con1,con2,con3)
c
c convert numy to integer
c
        inumy = int(numy+.1)
c
c initialize outline
c
        call initt(120)
        call binitt
        call xfrm(2)
        call yfrm(2)
        call xmtcs(1)
        call ymtcs(1)
c
c set up max and min values
c
        xmax = -1e+22
        xmin = 1e+22
        ymax = -1e+22
        ymin = 1e+22
c
c read in data vectors
c
        call readfi(xfil,yfil1,inumy,yfil2,yfil3,xarray,y1ary,
     &          y2ary,y3ary)
c
c make sure there is no more than 900 points
c
        if(int(xarray(1)+.1) .ge. numin) then
                write(*,*)'ERROR, NUMBER OF DATA POINTS EXCEEDS
     & 900, =',int(xarray(1)+.1)
                stop
        end if
c
c determine which arrays need to have min max values entered
c into the plot 10 common, and which need to have smoothed values
c calculated.
c
        if(s1) then
                call mnmx(y1ary,ymin,ymax)
        end if
        if(con1) then
        call splbuf(xtypec,ytypec,rms1,numin,xarray,y1ary,numpts,xout,
     &  y1out)
                call mnmx(y1out,ymin,ymax)
```

```
        end if
c
        if(s2) then
                call mnmx(y2ary,ymin,ymax)
        end if
        if(con2) then
        call splbuf(xtypec,ytypec,rms2,numin,xarray,y2ary,numpts,xout,
     &  y2out)
                call mnmx(y2out,ymin,ymax)
        end if
c
        if(s3) then
                call mnmx(y3ary,ymin,ymax)
        end if
        if(con3) then
        call splbuf(xtypec,ytypec,rms3,numin,xarray,y3ary,numpts,xout,
     &  y3out)
                call mnmx(y3out,ymin,ymax)
        end if
c
        if(s1 .or. s2 .or. s3) then
                call mnmx(xarray,xmin,xmax)
        end if
        if(con1 .or. con2 .or. con3) then
                call mnmx(xout,xmin,xmax)
        end if
c
c access common to get calculated max and min values
c
        call dlimx(xmin,xmax)
        call dlimy(ymin,ymax)
        call check(xarray,y1ary)
        xminc = comget(ibasex(26))
        xmaxc = comget(ibasex(27))
        yminc = comget(ibasey(26))
        ymaxc = comget(ibasey(27))
c
c
        minx= comget(ibasex(11))
c
c reinitilaize
c
        call initt(120)
        call binitt
        call xfrm(2)
        call yfrm(2)
        call xmtcs(1)
        call ymtcs(1)
c
c set up graph types
c
        call grtyp(xtypec,ytypec)
c
c convert labels to ascii equivalents, and determine # of characters
c
        call chrcod(titlec,tlen,astit)
        call chrcod(xlabl,xlen,asx)
        call chrcod(ylabl,ylen,asy)
        call chrcod(lab1c,len1,as1)
        call chrcod(lab2c,len2,as2)
```

```
        call chrcod(lab3c,len3,as3)
c
c plot arbitrary labels
c
        call labels(loc1x,loc2x,loc3x,loc1y,loc2y,loc3y,
     &              len1,as1,len2,as2,len3,as3)
c
c center the x label, y label, and title
c
        call xytlbl(xlen,ylen,tlen,asx,asy,astit)
c
c set density of tic marks
c
        call xden(7)
        call yden(7)
c
c plot points
c note the calls to comset(ibasec(1),0.) reset plot 10 common memory
c to no data symbol.  each following if block is then exicuted
c if sysmbols are requested for that particular yaxis data.
c
        call dlimx(xmin,xmax)
        call dlimy(ymin,ymax)
        call check(xarray,y1ary)
        call comset(ibasec(0),-1.)
        call comset(ibasec(1),0.)
        if(s1) then
                call symbl(6)
                call dsplay(xarray,y1ary)
        end if
        if(con1)then
                call comset(ibasec(1),0.)
                call comset(ibasec(0),0.)
                call dsplay(xout,y1out)
        end if
c
        if(inumy .ge. 2) then
                call comset(ibasec(0),-1.)
                call comset(ibasec(1),0.)
                if(s2) then
                        call symbl(3)
                        call cplot(xarray,y2ary)
                end if
        end if
        if(con2)then
                call comset(ibasec(1),0.)
                call comset(ibasec(0),76.)
c dashed line
                call cplot(xout,y2out)
        end if
c
        if(inumy .eq. 3) then
                call comset(ibasec(0),-1.)
                call comset(ibasec(1),0.)
                if(s3)then
                        call symbl(4)
                        call cplot(xarray,y3ary)
                end if
        end if
        if(con3)then
```

```
                call comset(ibasec(1),0.)
                call comset(ibasec(0),7434.)
                call cplot(xout,y3out)
        end if
c
        call finitt(0,700)
        read(5,555) dummy
 555    format(a1)
        end
```

```
C ------------------------------------------------------------------
C TELECOMMUNICATION AND INFORMATION SCIENCES LABORATORY, UNV. OF KANSAS
C ------------------------------------------------------------------
C PROGRAM NAME: plot_polar.forAUTHOR: Rafie            DATE: unknown
C ------------------------------------------------------------------
C VERSION NUMBER : 1.5        AMENDMENT DATE(S) : 3/20/86(B.A.M.)
C ------------------------------------------------------------------
C PURPOSE :    this routine will take a file containing the 360 data
C points and plot them on a polar graph using PLOT10 calls.  In this
C case, the plot parameters are fixed at: 0-=> -40 'dB' with angle
C markers every 60 degrees.
C ------------------------------------------------------------------
C                    PARAMETER DEFINITION
C  NAME           | TYPE   | CLASS| RANGE  |     DESCRIPTION
C ------------------------------------------------------------------
C filnam          |string  |      |        |the polar data filename
C                 |        |      |        |
C                 |        |      |        |
C ------------------------------------------------------------------
C                    SUBROUTINES REQUIRED
C    NAME         |         DESCRIPTION
C ------------------------------------------------------------------
C PLOT10 library  |various different routines
C                 |
C ------------------------------------------------------------------
      subroutine plot_polar(filnam)
      implicit none

      real rdata(360)
      real pi,cnt,rmax,rmin,theta_max,theta_min
      integer theta_step
      real degree
      integer number_of_rings,i,j,ncnt
      character*40 filnam
      pi = 3.141592
      open(unit=11,file=filnam,status='old')
      rmin=60.0
      rmax=100.0
      number_of_rings=4
      theta_min=0.0
      theta_max=360.0
      theta_step=30
      do 10 i = 1,360
      read(11,*) rdata(i)
      rdata(i)=rdata(i) + 100.0
      if(rdata(i) .eq. 100.) rdata(i) = 99.99
10    continue
      call initt(480)
      call twindo(100,1000,10,700)
      call dwindo(rmin,rmax+.001,theta_min,theta_max+.001)
      call poltrn(theta_min,theta_max,rmin)
      call gridpol(rmin,rmax,number_of_rings,theta_min,
     &             theta_max,theta_step)
      call movea(rdata(1),1.)
      do 20 j = 1,360
      ncnt = j
      degree = float(ncnt)
      call dashsa(rdata(ncnt),degree,12)
```

```
20      continue
        call finitt(0,0)
        return
        end
```

```
C ------------------------------------------------------------------
C TELECOMMUNICATION AND INFORMATION SCIENCES LABORATORY, UNV. OF KANSAS
C ------------------------------------------------------------------
C PROGRAM NAME: output.for     AUTHOR: Brian McClendon DATE: 3/20/86
C ------------------------------------------------------------------
C VERSION NUMBER : 1.0         AMENDMENT DATE(S) :
C ------------------------------------------------------------------
C PURPOSE :     this mainline displays the main menu and calls the
C appropriate subroutine selected.  If <CR> is entered, it will
C redisplay the menu, and if -1 is entered, the output is ended.
C
C ------------------------------------------------------------------
C                     PARAMETER DEFINITION
C  NAME            ! TYPE  ! CLASS! RANGE  !    DESCRIPTION
C ------------------------------------------------------------------
C none             !       !      !        !
C                  !       !      !        !
C                  !       !      !        !
C ------------------------------------------------------------------
C                     SUBROUTINES REQUIRED
C    NAME          !         DESCRIPTION
C ------------------------------------------------------------------
C histovert        !crunches the numbers for the histogram
C antenna_params   !displays all parameters associated with the antenna
C other_params     !shows 3 diff. displays for other params
C antenna_plot     !plots various antenna patterns
C plotter          !displays the 3 xy graphs necessary: histo, s/n, noise
C                  !
C ------------------------------------------------------------------

C This program will handle the output possibilities for the antennas
C currently, they are:
C  1) The antenna plots: with the various phi/theta slices as

C  2) A histogram of the number of attempts/ convergence

C   (3) time vs. S/N

C   (4) time vs error signal

C   (5) review the parameters
C
C
      implicit none

      integer input

      call histovert


C The main menu
      input=1
      do while(input.ne.-1)
         call initt(120)
         write(*,*) 'OUTPUT- The Main Menu'
         write(*,*)
         write(*,*) ' 1. Review antenna parameters'
         write(*,*) ' 2. Review channel information'
```

```
            write(*,*) ' 3. Review incoming signal information'
            write(*,*) ' 4. View calculated results'
            write(*,*) ' 5. View antenna plot'
            write(*,*) ' 6. View histogram of convergence counts'
            write(*,*) ' 7. View S/N vs. time graph'
            write(*,*) ' 8. View error signal vs. time graph'
            write(*,*) '-1. End output'
            write(*,*)
            write(*,998)
998         format(1x,'Which?',$)
            read(*,999) input
999         format(I4)

            if(input.eq.1) call antenna_params
            if((input.gt.1).and.(input.lt.5)) call other_params(input)
            if(input.eq.5) call antenna_plot
            if((input.gt.5).and.(input.lt.9)) call plotter(input-5)
         enddo
         stop
         end

C ********************************************************
```

```
C ------------------------------------------------------------------------
C TELECOMMUNICATION AND INFORMATION SCIENCES LABORATORY, UNV. OF KANSAS
C ------------------------------------------------------------------------
C PROGRAM NAME: other_params.for AUTHOR: Brian McClendon DATE: 3/20/86
C ------------------------------------------------------------------------
C VERSION NUMBER : 1.0          AMENDMENT DATE(S) :
C ------------------------------------------------------------------------
C PURPOSE :     this routine will display one of three tables containing
C data from the input or the output.  When <CR> is hit, it will return
C to the mainline.
C
C ------------------------------------------------------------------------
C                       PARAMETER DEFINITION
C  NAME             | TYPE   | CLASS| RANGE   |     DESCRIPTION
C ------------------------------------------------------------------------
C type              |integer|       |2..4     |tells which table to show


C ------------------------------------------------------------------------
C               NON-LOCAL VARIABLES -- FILE DEFINITIONS
C ------------------------------------------------------------------------
C params.dat        |file    |       |         |contains the input parameters
C                   |        |       |         |
C                   |        |       |         |
C ------------------------------------------------------------------------
C                     SUBROUTINES REQUIRED
C    NAME           |          DESCRIPTION
C ------------------------------------------------------------------------
C none              |
C                   |
C ------------------------------------------------------------------------

      subroutine other_params(type)

      implicit none

      integer type
      real rdummy
      integer idummy,i,j
      integer number_incoming,element_count,bit_rate,sample_bit,alg_type
      integer convergence_count,delay_count
      character*40 algorithm
      real incoming_theta(21),incoming_phi(21),delay(20),path_atten(20)
      real jammer_sn(20),channel_sn,convergence_constant
      real convergence_time,variance,average_si,si_variance

      call initt(480)
      open(unit=11,file='params.dat',status='old')
      rewind(unit=11)
      read(11,*) sample_bit
      read(11,*) channel_sn
      read(11,*) bit_rate,alg_type
      read(11,*) element_count,number_incoming
      read(11,*)
      read(11,*) convergence_count
      do i=1,element_count
         read(11,*)
      enddo
      do i=1,number_incoming
```

```
         read(11,*) incoming_phi(i),incoming_theta(i)
      enddo
      do i=1,(number_incoming-1)
         read(11,*) jammer_sn(i)
      enddo
      read(11,*) delay_count
      do i=1,delay_count
         read(11,*) delay(i)
      enddo
      do i=1,delay_count
         read(11,*) path_atten(i)
      enddo
      read(11,*) convergence_constant
      close(unit=11)



      if (type.eq.2) then
         write(*,*) '          Channel Information'
         write(*,*)
         write(*,370) bit_rate
370      format(1x,'Data Rate (in bits/sec) :     ',I8)
         write(*,371) sample_bit
371      format(1x,'Number of Samples/Bit   :     ',I8)
         algorithm='Update Covariance'
         if (alg_type.eq.1) algorithm='LMS'
         if (alg_type.eq.2) algorithm='Constrained LMS'
         write(*,372) algorithm
372      format(1x,'Algorithm used:                    ',A20)
         write(*,393) convergence_count
393      format(1x,'Number of Convergences  :     ',I8)
         write(*,373) convergence_constant
373      format(1x,'The Convergence Constant:    ',F16.8)
         write(*,374) channel_sn
374      format(1x,'Channel S/N ratio(in dB):     ',F8.2)
         write(*,*)
         write(*,*) '                Delay Information'
         write(*,375)
375      format(1x,'    delay(in msec)        path attenuation (in dB)')
         do i=1,delay_count
            write(*,376) delay(i),path_atten(i)
376         format(9x,F6.3,19x,F6.2)
         enddo
      endif

      if (type.eq.3) then
         write(*,*) '             Incoming Signal Information'
         write(*,*)
         write(*,377)
377      format
     & (1x,'signal      phi(in degrees)  theta(in degrees)  J/S(in dB)')
         write(*,378) incoming_phi(1),incoming_theta(1)
378      format(1x,'friendly           ',F5.2,12x,F5.2)
       do i=2,number_incoming
        write(*,379) i-1,incoming_phi(i),incoming_theta(i),jammer_sn(i-1)
379     format(1x,'jammer ',I2,8x,F6.2,12x,F5.2,9x,F6.3)
       enddo
       endif
```

```
      if (type.eq.4) then
         open(unit=11,file='results.dat',status='old')
         rewind(unit=11)
         read(11,*) convergence_time,variance
         read(11,*) average_si,si_variance
         close(unit=11)
         write(*,*) '              Calculated Results'
         write(*,*)
         write(*,380) convergence_time
380      format(1x,'Average Number of Iterations:        ',F8.2)
         write(*,381) variance
381      format(1x,'     Variance                :       ',F9.2)
         write(*,*)
         write(*,382) average_si
382      format(1x,'Average Signal/Interference (in dB): ',F8.2)
         write(*,383) si_variance
383      format(1x,'     Variance                :       ',F9.2)
      endif
      read(*,*)

      return
      end
```

```
C ------------------------------------------------------------------
C TELECOMMUNICATION AND INFORMATION SCIENCES LABORATORY, UNV. OF KANSAS
C ------------------------------------------------------------------
C PROGRAM NAME: histovert.for AUTHOR: Brian McClendon DATE: 3/20/86
C ------------------------------------------------------------------
C VERSION NUMBER : 1.0         AMENDMENT DATE(S) :
C ------------------------------------------------------------------
C PURPOSE :     This is a somewhat generic FORTRAN routine that will
C read in a file of data(one entry/line) and, given a box size, fill
C the appropriate boxes.  When finished it will write out a pair of
C files, the box values and the box totals.  This data can then be used
C by PLOT10 to put a graph on the TEk 4014(or whatever) screen.
C ------------------------------------------------------------------
C                     PARAMETER DEFINITION
C  NAME            ! TYPE   ! CLASS! RANGE   !     DESCRIPTION
C ------------------------------------------------------------------
C box_size         !integer!       !          !the size of each box.
C                  !        !      !          !
C                  !        !      !          !
C                  !        !      !          !
C ------------------------------------------------------------------
C                NON-LOCAL VARIABLES -- FILE DEFINITIONS
C ------------------------------------------------------------------
C input_file       !string !       !          !the input filename
C x_file           !string !       !          !the xdata output filename
C y_file           !string !       !          !the ydata output filename
C                  !        !      !          !
C ------------------------------------------------------------------
C                     SUBROUTINES REQUIRED
C    NAME          !          DESCRIPTION
C ------------------------------------------------------------------
C none             !
C ------------------------------------------------------------------

      subroutine histovert

C This program will take the file containing the convergence counts and convert
C it to the pair of files necessary for plot10 to plot a histogram.  It will
C scale the data correctly for reasonable output.

      implicit none
      character*50 input_file       !the file with the convergence listing
      character*50 x_file           !the file with the iteration x coordinates
      character*50 y_file           !the file with the histogram data
      integer data(1000),iterations(50)  !correspond to the above
      integer minimum,maximum       !the largest and smallest #'s in the file
      integer box_size              !the size of each box
      integer count,total_boxes,total_data,num,iter
      real histogram(50)

C********** Parameters ****************
      input_file='convergence.dat'
      x_file='xdata.dat'
      y_file='ydata.dat'
      box_size=50
C**************************************
      maximum=-9999999
      minimum=9999999
```

```
      open(unit=11,file=input_file,status='old')
      rewind 11
      count=1
60    read(11,*,end=70) data(count)
      if (data(count).lt.minimum) then
         minimum=data(count)
      endif
      if (data(count).gt.maximum) then
         maximum=data(count)
      endif
      count=count+1
      goto 60
70    close(unit=11)
      total_data=count-1

      open(unit=12,file=x_file,status='new')
      rewind 12
      open(unit=13,file=y_file,status='new')
      rewind 13

C Allocate enough boxes for 2 empties on each side of the histogram

      total_boxes=int((maximum-minimum)/box_size)+4

      count=0
      iter=(int(minimum/box_size)-2)*box_size
      do while (iter.lt.(maximum+2*box_size))
         iterations(count)=iter
         iter=iter+box_size
         count=count+1
      enddo

C *** initialize the histogram data ***
      do count=1, total_boxes
         histogram(count)=0
      enddo

      do count=1,total_data
         num=int(data(count)/box_size)-int(minimum/box_size)+2
         histogram(num)=histogram(num)+1
      enddo

      do count=1,total_boxes
         histogram(count)=histogram(count)/total_data
      enddo
C write out the xy data to the files
      do count=1,total_boxes
         write(12,*) iterations(count)
         write(13,*) histogram(count)
      enddo

      close(unit=12)
      close(unit=13)
      end
```

```
C ------------------------------------------------------------------------
C       TELECOMMUNICATION AND INFORMATION SCIENCES LABORATORY
C ------------------------------------------------------------------------
C PROGRAM NAME: ARRTEST        FROM: ANTENNA THEORY BY BALANIS
C MODIFIED BY: ROGER L. SPOHN                  DATE: JUN '85
C modified by: Brian McClendon                 Date: March '86
C ------------------------------------------------------------------------
C VERSION NUMBER : 1.4         AMENDMENT DATE(S) :
C ------------------------------------------------------------------------
C PU
C Purpose: This program produces pattern plots in user selected planes
C for any 2 dimentional dipole array geometry. The dipoles are fixed
C in length (70 ft. = 21.34 m) but the positions, amplitudes, and
C phases of each antenna are input variables. The operating frequency
C is also an input variable, as it determines the element pattern.
C      After the user is prompted for the above information, the
C program begins to ask about the plots. The program accepts an angle
C phi which defines a plane containing the z-axis. Phi is the angle of
C the plane with respect to the x-axis. Within this plane, then, a
C polar plot is performed resulting in gain as a function of the
C elevation angle theta. The same thing is then accomplished in the
C x-y plane, giving gain as a function of phi.
C     The above two plots, at the users request, can also be produced
C in rectangular form using pltsig. The program will store the data
C points in files for subsequent pltsig use.
C
C Modifications(3/86): The length of the dipoles is no longer fixed, but
C is passed to ARRTEST4 as a parameter.
C                      the entire program has been converted to a
C subroutine that is called by then antenna_plot subroutine.  It also has
C been altered so that EMAX is calculated in the batch routine main.exe
C to shorten the time required for one antenna pattern calculation.
C
C ------------------------------------------------------------------------
C                  Variable DEFINITIONs
C  NAME      | TYPE  | CLASS| RANGE  |    DESCRIPTION
C ------------------------------------------------------------------------
C N          |INT    |READ  |        |NUMBER OF ELEMENTS IN ARRAY
C UNUMOUT    |INT    |READ  |        |THE FILE OUTPUT NUMBER
C DATA1      |REAL   |ARRAY |        |POLAR DATA
C DATADB     |REAL   |ARRAY |        |RECTANGULAR DATA
C X          |REAL   |ARRAY |        |X LOCATION OF ELEMENT CENTER
C Y          |REAL   |ARRAY |        |Y LOCATION OF ELEMENT CENTER
C A          |REAL   |READ  |        |ELEMENT AMPLITUDE
C ALPHA      |REAL   |READ  |        |ELEMENT PHASE
C PHI        |REAL   |READ  |        |(PHI=0 IS THE X-AXIS) (INCREASING
C            |       |      |        |TOWARD Y-AXIS)
C THETA      |REAL   |READ  |        |(THETA=0 IS THE Z-AXIS)
C CARFRQ     |REAL   |READ  |        |FREQUENCY OF OPERATION
C TEMPPL     |REAL   |ARRAY |        |TEMPORARY STORAGE FOR POLAR DATA
C TEMPRC     |REAL   |ARRAY |        |TEMPORARY STORAGE FOR RECTANGULAR
C            |       |      |        |DATA
C            |       |      |        |
C ------------------------------------------------------------------------
C                SUBROUTINES REQUIRED
C    NAME       |      DESCRIPTION
C ------------------------------------------------------------------------
C ARFACT     | CALCUATES THE ARRAY FACTOR
C ELPAT      | CALCUATES THE ELEMENT PATTERN VALUE
C            |
```

```
C           |
C ------------------------------------------------------------------
C
      subroutine arrtest4(phi, theta, weight)
C
      IMPLICIT NONE
      CHARACTER NWGEOM, RECPLT, NWPHS, PLPLT
      CHARACTER*30, FILNAM
      REAL DATA1(360), DATADB(360), TEMPPL(360), TEMPRC(360)
      INTEGER N, M, I, MAXN, LUN1, weight
      REAL PI, DTR, ALPDEG, EMAX, PHI, THETA, X, Y, A, ALPHA, GAMMA, TEMP
      REAL ARFACT, ELPAT, CARFRQ, DELTA, PHIT, THETAT, antlen
      PARAMETER (PI=3.141592654, MAXN=99)
      PARAMETER (DTR=PI/180.)
      DIMENSION X(MAXN), Y(MAXN), A(MAXN), ALPHA(MAXN)
C
C ** Read parameter data file for useful default values
      filnam='antennaparams1.dat'
      filnam(14:14)=char(48+weight)
      open(unit=11, file=filnam, status='old')
      rewind 11
      read(11,*,end=220) N
      do I=1,N
       read(11,*,end=220) x(I)
       read(11,*,end=220) y(I)
       read(11,*,end=220) A(I)
       read(11,*,end=220) alpha(I)
       alpha(I)=alpha(I)*DTR
      enddo
      read(11,*,end=220) carfrq
      read(11,*,end=220) emax
      read(11,*,end=220) antlen
220   close(unit=11)
      gamma=phi*DTR
      delta=theta*DTR


C
C     Perform the pattern calculations for the particular, user
C     defined, vertical plane disection(s).
C

      DO I=1,360
         THETAT=I*DTR
         DATA1(I)=ELPAT(THETAT, CARFRQ, antlen)*ARFACT(THETAT, GAMMA,
     &    X, Y, A, ALPHA, N, CARFRQ)
         IF (ABS(DATA1(I)) .LT. 1.E-6) DATA1(I)=1.E-6
         DATADB(I)=20.*LOG10(ABS(DATA1(I)/EMAX))
         IF (DATADB(I) .LT. -100.) DATADB(I)=-100.
         TEMPRC(I)=DATADB(I)
      ENDDO
C
C     Perform the azimuth pattern calculation for the angle theta.
C
      DO I=1,360
         PHIT=I*DTR
         DATA1(I)=ELPAT(DELTA, CARFRQ, antlen)*
     &              ARFACT(DELTA, PHIT, X, Y, A, ALPHA, N, CARFRQ)
         IF (ABS(DATA1(I)) .LT. 1.E-6) DATA1(I)=1.E-6
```

```
            DATADB(I)=20.*LOG10(ABS(DATA1(I)/EMAX))
            IF (DATADB(I) .LT. -100.) DATADB(I)=-100.
         ENDDO
C
C
C
            FILNAM='DISECTION.DAT'
            CALL OPNCHK(LUN1,FILNAM,'NEW','FOR')
            DO I=1,360
               WRITE(LUN1,192) TEMPRC(I)
 192           FORMAT(1X,F10.4)
            ENDDO
            CALL UCLOSE(LUN1)

            FILNAM='AZIMUTH.DAT'
            CALL OPNCHK(LUN1,FILNAM,'NEW','FOR')
            DO I=1,360
               WRITE(LUN1,195) DATADB(I)
 195           FORMAT(1X,F10.4)
            ENDDO
            CALL UCLOSE(LUN1)

         END
C
C
C
         FUNCTION ARFACT(THETA,PHI,X,Y,A,ALPHA,N,CARFRQ)
C   THIS SUBPROGRAM RETURNS THE ARRAY FACTOR VALUE OF A GENERAL ARRAY
C        AT THE ANGLE (THETA,PHI)
C
C   THE ARRAY FACTOR IS UNNORMALIZED
C
         IMPLICIT NONE
         INTEGER I,MAXN,N
         PARAMETER (MAXN=99)
         REAL ARFACT,C,CARFRQ,PI,X,Y,A,ALPHA,THETA,PHI
         PARAMETER (C=299792458.)
         COMPLEX TEMP,CEXP,IMAG,CABS
         DIMENSION X(MAXN),Y(MAXN),A(MAXN),ALPHA(MAXN)
         PARAMETER (PI=3.141592654)
C
C
         IF (N .EQ. 1) THEN
            ARFACT=1.
            RETURN
         END IF
         IMAG=(0.,1.)
         TEMP=(0.,0.)
         DO 10 I=1,N
            TEMP=TEMP+A(I)*CEXP((IMAG*2.*PI*CARFRQ/C)*(X(I)*SIN(THETA)
     &      *COS(PHI)+Y(I)*SIN(THETA)*SIN(PHI))+IMAG*ALPHA(I))
 10         CONTINUE
         ARFACT=CABS(TEMP)
         IF (ARFACT .LT. 1.E-06) ARFACT=0.
         RETURN
         END
C
C
C
         FUNCTION ELPAT(THETA,CARFRQ,antlen)
```

```
C
C      This program returns the element pattern value for the
C      angle THETA and the particular carrier frequency CARFRQ.
C
       REAL ELPAT, THETA, PI, CARFRQ, ANTLEN, MULT, C
       PARAMETER (PI=3.141592654, C=299792458.)


       MULT=2.*PI*ANTLEN/C
       IF (ABS(SIN(THETA)) .LT. 1.E-5) THEN
          ELPAT=0.
          RETURN
       END IF
       ELPAT=(COS((MULT*CARFRQ*COS(THETA))/2.) - COS(MULT*CARFRQ/2.
      &))/SIN(THETA)
       RETURN
       END
```

```
C ------------------------------------------------------------------
C TELECOMMUNICATION AND INFORMATION SCIENCES LABORATORY, UNV. OF KANSAS
C ------------------------------------------------------------------
C PROGRAM NAME: antenna_plot.for AUTHOR: Brian McClendon DATE: 3/20/86
C ------------------------------------------------------------------
C VERSION NUMBER : 1.0         AMENDMENT DATE(S) :
C ------------------------------------------------------------------
C PURPOSE :     this routine displays a menu of choices for an antenna
C plot.  It uses GETIN for standard parameter acceptance, calls
C ARRTEST4, and then displays the graph using plot_polar.
C
C ------------------------------------------------------------------
C                     PARAMETER DEFINITION
C  NAME            ! TYPE  ! CLASS! RANGE  !     DESCRIPTION
C ------------------------------------------------------------------
C none             !       !      !        !
C                  !       !      !        !
C ------------------------------------------------------------------
C                     SUBROUTINES REQUIRED
C    NAME          !         DESCRIPTION
C ------------------------------------------------------------------
C GETIN            !standard parameter input routine
C arrtest4         !routine that calculates the antenna pattern
C plot_polar       !the routine that displays the polar data
C                  !
C                  !
C                  !
C ------------------------------------------------------------------

      subroutine antenna_plot

      implicit none


C     ****************************************************************
      INTEGER HDRLNG,PARCNT,I,J,ALGTYP,LUN1
      REAL NUMVAL(2*9),RMIN(2*9),RMAX(2*9)
      CHARACTER*72 HEADER(5)
      CHARACTER*50 PARDES(2*9)
      CHARACTER*25 CHRVAL(2*9)
      CHARACTER*1 PARTYP(2*9)
      CHARACTER*2 ELNUM,SIGNUM
      LOGICAL INIT(2*9),FSTAT(2*9)
C     ****************************************************************
      include 'format.inc'

      integer maxn
      parameter(maxn=99)
      integer n,weight
      real x(maxn),y(maxn),a(maxn),alpha(maxn),carfrq
      real phi,theta
      character*40 filnam
      character*1 chr

      hdrlng=4
      header(1)='                        SLICE DESCRIPTION'
      header(2)='    After exiting this menu, the graph data will be'
      header(3)='      calculated and then plotted in polar form.'
```

```
      header(4)=' '
      parcnt=4

      pardes(1)='1) min. convergence time 2) avg.  3) max'
      partyp(1)='D'
      init(1)=.true.
      rmin(1)=1
      rmax(1)=3
      numval(1)=2
      pardes(2)='Fixed angle Phi for elevation plot (in deg.)'
      partyp(2)='D'
      init(2)=.true.
      rmin(2)=0
      rmax(2)=360
      numval(2)=0
      pardes(3)='Fixed angle Theta for azimuth plot (in deg.)'
      partyp(3)='D'
      init(3)=.true.
      rmin(3)=0
      rmax(3)=360
      numval(3)=45
      pardes(4)='Slice type: 1)Elevation 2)Azimuth'
      partyp(4)='D'
      init(4)=.true.
      rmin(4)=1
      rmax(4)=2
      numval(4)=1

333   call initt(120)
      CALL GETIN(HEADER,HDRLNG,PARCNT,PARDES,PARTYP,CHRVAL,NUMVAL,
     &           INIT,FSTAT,RMAX,RMIN)

      weight=numval(1)
      phi=numval(2)
      theta=numval(3)
      if (numval(4).eq.1) then
          filnam='disection.dat'
      else
          filnam='azimuth.dat'
      endif

      call arrtest4(phi,theta,weight)
      call plot_polar(filnam)
      write(*,334)
334   format(1x,'More Antenna plots (y/n)?',$)
      read(*,335) chr
335   format(a1)
      if ((chr.eq.'Y').or.(chr.eq.'y')) goto 333
      return
      end
```