Development of an Eigenvector Sorting Algorithm for HF Antenna Arrays

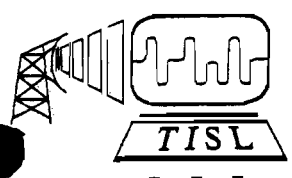
R. Spohn, V.S. Frost, R. Balasubramaniam, R. Moats

Final Technical Report TISL-5481-3

Prepared for:
Rome Air Development Center RADC/DCCD
Griffiss Air Force Base
Rome, New York 13441-5700

Contract No. F-30602-81-C-0205

August 1987



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 depends on both the adaptive algorithm employed and the signal environment in which the system operates. Computer simulation provides a suitable method for the evaluation of such combinations. The major focus of this study is the development of an eigenvector sorting algorithm for the control of an adaptive array. The algorithm makes use of the signal autocorrelation matrix derived from the sensor elements. The adaptive system is integrated into a previously developed simulation package which creates the signal environment and the array characteristics. The relationship between the array system and the adaptive processor is discussed followed by a quick review of three other algorithms supported by the simulation package. These algorithms are used for comparison to the eigenvector algorithm. The mathematical background of the eigensystems are presented as well as a discussion of the algorithm implementation. Results of the simulations will contain comparisons to the other algorithms as well as independent tests of the eigenvector algorithm itself.

Table of Contents

List of Figures

List of Tables

List of Symbols

$x_i(t)$	Time dependent signal at i-th antenna element
w_i	Adaptive weight at i-th antenna element
$y(t)$	Adaptive array output signal
$s_i(t)$	Signal component at i-th antenna element
$n_i(t)$	Noise component at i-th antenna element
$j_i(t)$	Jammer signal component at i-th antenna element
$g_i(t)$	Gaussian noise component at i-th antenna element
A_i	Amplitude of i-th weight
θ_i	Phase of i-th weight
$z_i(t)$	Complex weighted signal from sensor i
$\underline{x}$	Signal vector, Eigenvector
$\underline{w}$	Weight vector
$\underline{A}$	Arbitrary matrix
$\underline{I}$	Identity matrix
λ	Eigenvalue
$\underline{T}$	Arbitrary transform matrix
$\underline{e}_1$	$[1,0,0,\ldots,0]^T$
N	Number of antenna elements
$\underline{P}$	Householder transform matrix
j	$\sqrt{-1}$
$\underline{L}$	Lower triangular matrix

1.0 Introduction

Communication over the HF band (3-30 MHz) has become important in many areas including military applications and ionospheric research. If unprotected, any communications system is susceptible to interference caused by deliberate, hostile jamming signals. In order to cope with this problem, it is possible to employ an array of sensor elements under the control of an adaptive algorithm. With this system, the algorithm is able to monitor the signal environment of the sensors and, according to some pre-defined criteria, force advantageous changes in the antenna pattern. This is accomplished by the correct adjustment of weighting elements at the output of each sensor, altering the amplitude and phase of the signals. The result of a judicious choice of weights is an antenna pattern possessing nulls in the directions of the jammers, while still maintaining adequate response in the desired transmitter direction.

It is the purpose of this study to develop a simulation model of an eigenvector sorting algorithm for the control of an HF antenna array. The algorithm relies on the ability to construct the signal autocorrelation matrix from the sensor array. Furthermore, the eigenvectors of this matrix, when applied as weighting elements, cause the signals incident on the array to be separated. This quality of the algorithm, then, may be used to separate the desired signal from all other

unfriendly signals at the array[11].

The algorithm has been incorporated into a simulation package developed by D. Haegert et. al. [1]. This package was specifically developed for the evaluation of adaptive algorithms employed in the control of array antennas in an HF environment. The simulation supports an array antenna, for which the user may choose up to 36 elements arranged in any planar geometry. Also contained in the simulation is an HF channel model implemented as a tapped delay line. The user may specify the number of signal paths desired as well as the delays and attenuations associated with these paths. Along with other support software, this system forms a suitable environment for testing the eigenvector sorting algorithm. The simulation contains, in addition, three alternative adaptive algorithms, which were previously developed. Specifically, the other supported algorithms are

1) Least mean square (LMS) [6]

2) Constrained LMS [9]

3) Update Covariance [10]

A review of the above algorithms will be presented in a later section. These algorithms are mentioned since a portion of this study deals with the comparison of the performance of these algorithms with that of the eigenvector algorithm.

In an effort to explain the antenna array configuration, and the relationship to the adaptive processors of the

simulation, section 2 of this report is dedicated to that task. The basic system elements and their operations are described. Some notation to be used throughout the study will also be developed.

The purpose of the next section is to provide a quick review of the three alternative algorithms supported by the system model. As was mentioned above, these will be employed to an extent as gauges indicating the performance of the eigenvector algorithm by a comparison of results. The section is not mathematical and does not attempt to derive the algorithm equations. The purpose is simply to make the reader aware of their operation and to form a basis for comparisons.

The mathematical background of the eigenvector algorithm is presented in section 4. To fully understand the algorithm operation, it is necessary to present some properties of matrices and the related eigensystems. Similarity transformations are introduced which are vital to the eigenvalue/eigenvector solutions. The Householder and QL algorithms, which allow an efficient numerical solution to the problem, are also introduced.

The final two sections are devoted to the implementation of the new algorithm and the results of the simulations, respectively. The implementation of the eigenvector algorithm is carefully outlined and the actual operation of the software

modules is described. The final section will present the results of the study in graphical form, and will also contain conclusions relevant to the algorithm performance.

2.0 ADAPTIVE ARRAY SYSTEM MODEL

This section is designed to introduce and explain the adaptive array system model which will be used in this study. In order to achieve the desired goals of the study, it is, of course, necessary to implement and include such a model in the simulation. It is not the purpose here, however, to give an exhaustive explanation of the software operation, but simply to outline the basic function of the unit as applied to adaptive processing. A simple diagram of the system in question is shown in figure 2.1.

As is the case with most adaptive array systems, there are primarily three constituent sub-systems forming the whole. As shown in the diagram, these are: 1) the array of sensor elements; 2) the pattern-forming network; 3) the adaptive processor. Each of these components will, in turn, be discussed in the following sections. Although not explicitly shown, it is also important to realize that the array is imbedded in an environment consisting of a number of incoming signals. For this study, the signals consist of one friendly as well as a number of "unfriendly" or jamming signals impinging on the array from various angles of azimuth and elevation.

Referring to the diagram, it should also be noticed that this system does not represent a closed loop arrangement. Recall that in the case of adaptive processors such as the

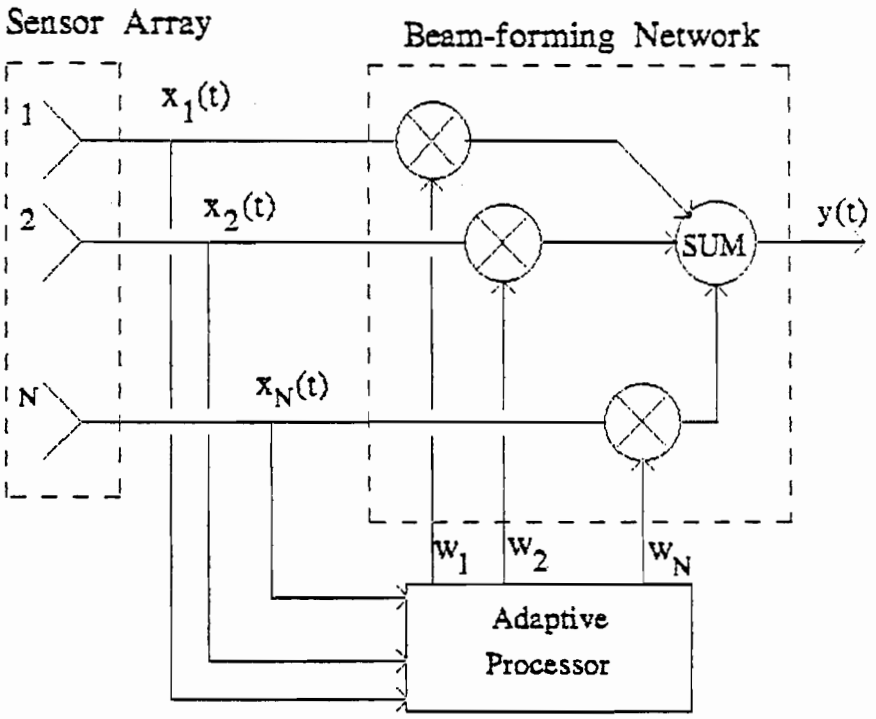


Figure 2.1 Adaptive Array System Model

least mean squared algorithm (LMS), information about the array output, y(t), must be made available to the processor to allow proper adaptation.[1] In the present study, however, the processor employed takes information only from the sensor elements of the array, making for a completely open loop system. The consequences of this arrangement will be discussed in a later section. Let us now expand on the various sub-systems of the array system model.

2.1 Sensor Element Array

The array consists of N spatially separated sensor elements which, for this study, exist in a planar configuration. The function of the sensor elements is to collect information about the signals, both friendly and hostile, which impinge on the array and to distribute them to the other system components. Careful consideration must be given to both the type of elements chosen as well as to their relative locations, since these attributes strongly affect the overall performance of the adaptive system. In many cases, however, complete freedom of choice cannot be maintained due to preassigned constraints on either the element types or the array geometry. In these situations, the performance of the overall system will most probably suffer some degradation.

The output of any particular sensor element, denoted $x_i(t)$, can be expressed as

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

where $s_i(t)$ is the desired signal component at the element i, and $n_i(t)$ is the noise component at element i. The noise component consists of not only the deliberate jamming signals, but also the effects of natural or Gaussian noise present at each sensor. To be strictly correct, then, the output signal from any particular sensor should be written as

$$x_i(t) = s_i(t) + j_i(t) + g_i(t)$$

where $j_i(t)$ and $g_i(t)$ are used to represent the contributions from deliberate and Gaussian noise respectively. In this study, however, the former simpler notation $n_i(t)$ will be used as it is not crucial to distinguish between the two types of interference.

The antenna array used in this simulation study does possess one novel attribute which differs from virtually all other systems presented in the literature. The sensor elements usually employed in such studies are simply isotropic. That is, the reception capability of a single element is independent of the incoming signal direction. In this study, however, dipole elements have been employed which cause a definite directionality in the individual element patterns. This does, of course, complicate the software model for the antenna, but as a result, the model attains a greater degree of realism. It

should also be pointed out, as mentioned above, that the array configuration employed is a planar arrangement as opposed to a linear one. In order to extract as much information about the incoming signals as possible from the array, it is necessary to take advantage of both dimensions. In linear arrangements, it is impossible to uniquely determine both the elevation and azimuth arrival angles of incoming signals. Linear arrangements do allow very simple phase calculations, but the inhibition of signal directional information simply could not be tolerated.

2.2 Pattern-Forming Network

The pattern-forming network of the adaptive array system consists simply of N variable complex weights, one for each sensor, as well as a summing device used to combine the weighted signals. Essentially the variable weights are under the control of the adaptive processor. The pattern-forming network accepts the weighting factors from the processor, performs a complex multiply for each sensor, and transfers the weighted signals to the summing device. Actually, the complex multiply which takes place is really just an adjustment of the signal amplitude and phase. The complex numbers used as weighting factors can be written in amplitude and phase form as

$$w_i = A_i \, e^{j\theta_i}$$

where

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

and

$$\theta_i = \tan^{-1} (Im\{w_i\}/Re\{w_i\})$$

Thus, denoting by $z_i(t)$ the complex weighted signal from sensor i, we can say that

$$z_i(t) = w_i \, x_i(t)$$

To form the output of the array, y(t), we must simply form the sum of the weighted sensor signals as in

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

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

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

As a note on notation, the following conventions will be used throughout. Matrices will be denoted by underscored capital letters while, as shown above, vectors shall be represented as underscored lower case letters. Matrix or vector transpose operations will be denoted by a superscript "T", and the complex conjugate operator by a superscript asterisk (*). Finally, scalar quantities shall be represented by lower case letters with no special extra notation.

2.3 Adaptive Processor

The function of the adaptive processor, in general, is to control the operation of the antenna array in such a way as to produce the desired overall gain pattern. The entities contained in the processor are the adaptive algorithm itself as well as any support components used by the algorithm. The processor accepts information about the state of the environment from the sensor system, and using this information makes choices concerning the signal weighting. These weights are then passed to the patten forming network causing adjustments in the amplitude and phase of the sensor outputs.

In his study, as mentioned earlier, there is no feedback from the array output to guide in the adaptation process. The processor function is essentially to determine the eigenvalues and eigenvectors of the covariance matrix obtained from the outputs of the sensor elements. The eigenvectors which result

from the calculation are the output weight vectors that are utilized within the pattern-forming network. The processor is primarily concerned with the power at the array output, and produces a collection of weight vectors (eigenvectors) to be used at the discretion of the operator. These concepts will be expanded upon and explained in more detail in the next several sections.

3.0 Summary of Previous Work on Adaptive Algorithms

3.1 General Background

In order to insure secure communication over the HF band in the presence of "unfriendly" jamming signals, one alternative is the use of adaptive array systems. By employing a controlling algorithm, the antenna weights may be adaptively chosen to form nulls in the antenna pattern in directions corresponding to the "unfriendly" signal arrival, while simultaneously maintaining communication with the desired transmitter. The feasibility of such architectures may be assessed quite efficiently by using computer simulations of the hardware as well as the underlying controlling algorithms. Simulation techniques allow rapid comparisons to be made according to the performance of particular algorithms in a variety of signal environments. The purpose of this section is to summarize the previous work on adaptive systems which has been carried out at the University of Kansas.

3.2 Simulation History of Adaptive Systems

The first major attempt at a simulation closely related to the present form was conducted by Paul Snow in 1984 [5]. The focus of this study was to determine the feasibility of simulating an adaptive array using the least mean square algorithm (method of steepest decent). The conclusions reached

showed that such simulations were indeed valuable in the prediction of adaptive array performance.

In the second stage of investigation, these ideas were carried on to a somewhat larger scale. D. Haegert, et. al., in 1986 [1], developed a simulation package allowing a much wider range of options to be explored. This package was implemented on a VAX 11/750 under VMS and supported a more flexible antenna system, an HF communication channel, and three distinctly different adaptive algorithms. The antenna system could be simulated with up to 9 elements arranged in any desired planar configuration, as opposed to the linear arrays of the Snow simulation. The HF channel allowed effects such as multi-path propagation to be included in the simulation, adding a higher degree of realism to the scenarios. The adaptive algorithms supported were the least mean square algorithm, as in the Snow simulation, a constrained LMS algorithm [9], and an algorithm known as the Update Covariance [10].

The results from [1] will be compared to the performance of the eigenvector sorting algorithm. A brief explanation on each of the above algorithms will be presented in the interest of completeness. No attempt will be made to mathematically derive the algorithms, but the reader will be given references for that information. The goal here is simply to outline the algorithms such that an intelligent comparison can be made between them and the eigenvector sorting routines.

3.3 Least Mean Square (LMS) Algorithm

The least mean square algorithm is perhaps the most popular adaptive scheme known. As the name implies, its performance criteria is based on the mean square error between the controlled system output and an externally generated reference signal. Since its introduction and development by Widrow in the 1960's, it has been used in a vast number of widely varying applications.[6-8]

As mentioned above, the LMS algorithm relies on the generation of what is known as a reference signal. It is the purpose of this signal to mimic, as closely as possible, the "desired" output of the system. In this way, the algorithm is able to measure the error in the output signal by simply forming the difference between it and the reference signal. In other words, the algorithm classifies the error signal as anything remaining after the reference signal has been accounted for. The error signal is subsequently used by the algorithm as an indicator for the proper weight adjustments.

One point mentioned above should perhaps be clarified. It is obviously advantageous for the algorithm to be supplied with a reference signal which closely resembles the desired information. Of course, if an exact copy of the transmitted information is available at the receiver, there is obviously no need for the transmission at all. Luckily, the LMS algorithm is

usually capable of adaptation by making use of reference signals acquired by certain approximation techniques. In some cases, where the desired signal is extremely complex or poorly correlated, reference signals cannot be approximated and consequently, the LMS algorithm cannot be used. In the case of communication systems, however, it has been shown that, in general, the signals lend themselves to reference generation. Especially in the case of systems involving the use of known codes, it is a simple matter to generate a satisfactory reference signal.

The LMS algorithm makes use of gradient estimation techniques to guide in the piecewise adjustment of the adaptive weights. In this context, of course, the weights refer to the amplitude and phase adjustments applied to the individual sensor elements of the array antenna. It can be shown that the mean square error (MSE) surface may be considered as a hyperparaboloid in the weight space. In other words, the mean square error is a quadratic function of the weights [7]. At each step in the adaptation process, the algorithm estimates the gradient of the MSE function and accordingly adjusts the weights such that the MSE is reduced. It is the goal of the algorithm, of course, to move toward the point of minimum MSE. When the algorithm actually attains this point, the difference between the array output and the reference signal is minimum, and the algorithm is said to have converged. No further adjustment of the weights can reduce the MSE.

Actually the LMS algorithm is quite simple as compared to the other adaptive algorithms used, and was chosen for this reason as well as for its notoriety. In addition, the LMS is often used as a benchmark for comparisons to other adaptive algorithms. In this way, the more complex algorithms, to be discussed next, could be analyzed in a comparison mode, allowing a more complete assessment of their performance.

3.4 Constrained LMS Algorithm

The very name of this algorithm is somewhat misleading as it implies that the constrained LMS is simply a different version of the LMS algorithm. This is not the case, as will be seen, since here the performance criteria is maximum likelihood as opposed to least mean square. This algorithm was developed by O. L. Frost, and it is possible that the name was chosen since it also uses a gradient approach for adaptation. [9]

Compared to the LMS, this algorithm has both advantages and disadvantages. First, the requirement of a reference signal is relaxed when using the constrained LMS algorithm. This completely eliminates the need for any approximation techniques involving the generation of the reference signal. The price paid for this luxury, however, is the need to know the direction of the desired signal upon the antenna array. Thus, for the algorithm to perform in the correct and expected fashion, this information must be known a priori. If this is possible

there will be no problem, but in any case where the desired signal direction is unknown, obviously the constrained LMS algorithm cannot be applied.

In order to present a cursory explanation of the algorithm operation, it will be expedient to think of the antenna array as a spatial filter. With this in mind, the steps taken by the algorithm may be explained and understood more readily.

Armed with the knowledge of the desired signal direction, the algorithm creates what is known as a steering vector. This vector, when applied to the element weights, steers the antenna such that the desired signal component at the output of each element is exactly in phase at the output of the array. If the desired signal direction does not change during adaptation, this vector, of course, remains constant and needs no alteration. Speaking in terms of the antenna "filter", the algorithm arbitrarily assigns the filter transfer function a value of 1 in the direction of the desired signal. Once this is done, by use of gradient methods, the algorithm proceeds to minimize the output power of the array while explicitly constraining the response in the desired signal direction. The effect of this is to produce nulls in the antenna pattern in any direction for which power enters the array, except for the desired signal direction.

As can be imagined, this adaptation method is susceptible

to several problems. For example, if the desired signal is not arriving from the direction the algorithm believes, it will be treated exactly as a jammer and will be promptly nulled by the antenna pattern. The algorithm has no way of knowing that this signal is conveying useful information. Also, if a jamming signal arrives at the array at virtually the same angle as the desired signal, it will be allowed to enter the array unchecked. This, however, must be said for any algorithm which uses spatial discrimination.

The constrained LMS algorithm is somewhat more complex than the LMS, but is still relatively simple. Very few computations are required to update the weights, and the convergence rate is comparable to that of the LMS. The algorithm to be discussed next possesses a much greater convergence speed, but also requires a much larger number of computations to update the weight vectors.

3.5 Update Covariance Algorithm

The update covariance algorithm operates by a very different method than the previously mentioned algorithms. Recall that the LMS and constrained LMS algorithms make use of gradient estimation to iteratively approach the optimal solution. The update covariance, on the other hand, employs a much more direct approach in the calculation of the optimum weight vectors. This algorithm falls into a class of processors

known as recursive. This implies that the results of the k+1 iteration are dependent on the outcome of the k-th iteration. The algorithm, therefore, remembers past results and uses them to some extent in formulating the present state of the system. The rate of convergence for the update covariance is extremely fast as compared to either of the gradient algorithms, but the mathematical complexities involved in updating the weights is much greater as well. This is generally true for all adaptive processors. If the convergence rate is high, the mathematical operations for weight updates are usually quite complex.

The update covariance algorithm, as presented by Monzingo and Miller [10], approaches the optimum weight calculation very directly. It makes use of a very important relationship known as the Wiener-Hopf equation [10]. This equation expresses the optimum weight vector in terms of the inverse signal covariance matrix. If it were desired to directly calculate the optimum weights, in one iteration, the Wiener-Hopf equation supplies the proper recipe. This calculation, however, necessitates the direct inversion of the covariance matrix, and it is precisely this operation that the algorithm strives to avoid.

To circumvent this computational problem, then, the update covariance, as the name implies, iteratively updates the inverse correlation matrix in an effort to approach the actual inverse. At each iteration, the inverse approximation is employed to update the weight vector according to a mathematic-

ally simplified form of the Wiener-Hopf equation. The update equations for both the inverse covariance matrix and the weight vectors may be found in [10].

In this section the work on previous adaptive systems has been summarized. In addition, the operation of the algorithms themselves have been quickly explained. Although only cursory treatments have been attempted, the references should provide complete explanations for the interested reader.

4.0 Mathematical Background of Eigensystems

In this section the mathematical development of eigensystems will be reviewed to provide the underlying concepts of the eigenvalue sorting algorithm. Not only will this serve as a guide to the theory of eigensystems used in this study, but will also provide the tools necessary for a subsequent explanation of the actual software implementation of the algorithm. Although the exhaustive set of all eigensystem situations is quite vast, only a small subset will be of interest here. This is due to the fact that we are concerned with the eigenvalues and eigenvectors of the correlation matrix obtained from the array antenna system. Due to their construction, these matrices always fall into a class known as Hermitian, as will be discussed. Because of their symmetry, Hermitian matrices lend themselves to relatively simple eigenvalue/eigenvector analysis as compared to random nonsymmetric matrices [2].

4.1 Basic Definitions and Concepts

As an aid for what is to follow, several terms used to classify matrices and their properties will quickly be introduced. A square matrix $\underline{A}$ is said to be symmetric if it is equal to its transpose, i.e.,

$$\underline{A} = \underline{A}^T \quad \text{or} \quad a_{ij} = a_{ji}$$

A matrix is known as Hermitian, on the other hand, if it is equal to its conjugate transpose,

$$\underline{A} = \underline{A}^{*T} \quad \text{or} \quad a_{ij} = a^*_{ji}$$

Orthogonal refers to the property that the matrix transpose is equal to the inverse. That is, if

$$\underline{A}^{-1} = \underline{A}^{T} \quad \text{or} \quad \underline{A}\,\underline{A}^{T} = \underline{I}$$

then $\underline{A}$ is orthogonal. A matrix is known as unitary if its conjugate transpose is equal to its inverse. In other words, the immediately preceding equation holds when the transpose operations are replaced by the conjugate transpose, i.e.,

$$\underline{A}^{-1} = \underline{A}^{*T} \quad \text{or} \quad \underline{A}\,\underline{A}^{*T} = \underline{I}$$

Lastly, a matrix is known as normal if it commutes with its conjugate transpose, i.e.,

$$\underline{A}^{*T}\,\underline{A} = \underline{A}\,\underline{A}^{*T}$$

These definitions will be useful in the following discussion.

The basic eigenvalue/eigenvector problem is to find a scalar λ and the corresponding vector $\underline{x}$ that together form a solution to the equation

$$\underline{A}\,\underline{x} = \lambda\,\underline{x}$$

where $\underline{A}$ is a given N x N matrix. The above equation may also be written in the form

$$(\underline{A} - \lambda\,\underline{I})\,\underline{x} = \underline{0}$$

which represents a set of N linear homogeneous equations. From linear algebra we know that this system of equations possesses a nontrivial solution if and only if

$$\det |\underline{A} - \lambda \underline{I}| = 0$$

Expanding this equation leads to an n-th order polynomial in λ of the form

$$p_A(\lambda) = (-1)^N \lambda^N + b_1 \lambda^{N-1} + \ldots + b_N$$

The N roots of this characteristic equation are the eigenvalues of the original problem. This shows that there are always N eigenvalues, but depending on the form of the matrix $\underline{A}$, they may or may not be distinct. The above reasoning also shows that for each of the N eigenvalues there necessarily corresponds an eigenvector which, again, may or may not be distinct. This can be shown by considering the above equations. Substituting any of the N given eigenvalues into the expression $\underline{A} - \lambda \underline{I}$, this matrix becomes singular, and it can be shown that any singular matrix contains at least one vector in its null space [2].

As was mentioned earlier, the matrix of interest in this study is the signal correlation matrix taken directly from the sensor array. Due to their construction, correlation matrices may always be classified as Hermitian. These matrices are special in the context of eigensystems for several reasons. First of all, it is known that each of the eigenvalues of a Hermitian matrix are totally real. This is never the case for

either real nonsymmetric or complex non-Hermitian matrices. This property is important in the context of this study because, as will be shown later, the eigenvalues are related to the power at the output of the antenna array.

Another special property of Hermitian matrices is related to the eigenvectors. From the definitions given in the beginning of this section, it is obvious that any Hermitian matrix is also normal. This is important since any normal matrix possessing a nondegenerate set of eigenvalues will produce a complete and orthogonal set of eigenvectors which span the N dimensional vector space. (Nondegeneracy of the eigenvalues refers to the case in which an N x N matrix possesses exactly N distinct eigenvalues.) The significance of the matrix eigenvectors in this work, to be shown later, is that they are used as weighting vectors and are applied to the pattern-forming network of the adaptive array system.

4.2 Matrix Diagonalization and Similarity Transforms

As was mentioned earlier, the eigenvalues of a N x N matrix A are actually the roots of the characteristic equation

$$\det |\underline{A} - \lambda \underline{I}| = 0$$

Since this is an N-th degree polynomial in λ , in theory the eigenvalues could be found by searching the roots of this large polynomial. It is important to realize, however, that this is definitely not the most efficient way to proceed [3]. Aside

from its efficiency, this procedure would do nothing in the way of producing the corresponding eigenvectors of the system. In this section diagonalization methods are briefly introduced which allow efficient and complete computation of both the eigenvalues and eigenvectors of a given square matrix. The basic aim of these methods is to use similarity transforms, to be discussed shortly, in an effort to iteratively shape the matrix to diagonal form. Once this is done, the diagonal elements are the eigenvalues of the system and the eigenvectors may be extracted from the transformation matrices used to achieve the diagonalization.

Although methods do exist for the complete diagonalization of matrices, they are rarely used due to the great amount of computation involved [4]. In general, complete diagonalization cannot be carried out using a finite number of transformations. Thus, instead of complete diagonalization in one stage, virtually all modern eigensystem methods break the problem into two distinct sub-problems. First, the given matrix is reduced to one of several specialized forms such as tridiagonal or Hessenberg, which contain many zero elements, while at the same time preserving the eigenvalues. This is a relatively painless process since such reductions are strictly finite operations. Once this is complete, the eigenvalues and eigenvectors of the specialized matrix may be found with a much lower computational cost. For symmetric matrices the reductions lead to the tridiagonal form, while the Hessenberg form results from

transformations performed on nonsymmetric matrices. In this study we are primarily concerned with the tridiagonal form since the matrices to be transformed are always symmetric. The form of the tridiagonal N x N matrix is shown below in figure 4.1. Notice the symmetry of this matrix. As will be explained more fully later, this is due to both the symmetry of the original matrix as well as to the symmetry preserving properties of the employed transformations.

$$\begin{bmatrix} a_1 & b_1 & & & & & \\ b_1 & a_2 & b_2 & & & & \\ & b_2 & a_3 & b_3 & & & \\ & & b_3 & \cdot & \cdot & & \\ & & & \cdot & \cdot & \cdot & \\ & & & & \cdot & \cdot & \cdot \\ & & & & & \cdot & \cdot & b_N \\ & & & & & & b_N & a_N \end{bmatrix}$$

Figure 4.1 Form of the Tridiagonal Matrix

The impetus for the two stage method can be understood by considering the underlying problem. In essence, the problem of extracting the eigenvalues from the specialized matrix still requires the searching for roots of a high degree polynomial. This is necessarily an iterative procedure since there are no

explicit formulas for the solution [4]. In addition we know that general matrix manipulations, such as multiplication, require on the order of N^3 operations, where N is the side dimension of the matrix. On the other hand, manipulations with the specialized tridiagonal form only require on the order of N operations. It quickly becomes apparent that even with moderately sized matrices, the difference between N^3 and N is extremely significant. In view of this, it is obviously more efficient to postpone the iterative (infinite) stage of the computation until the matrix has been reduced to one of the simpler forms. The two stages of the problem, 1) reducing the general matrix to specialized form, and 2) extracting the eigenvalues and eigenvectors from this transformed matrix are virtually independent. Let us now briefly discuss these two procedures along with the mathematical tools necessary to their formulation.

4.2.1 Similarity Transformations

Practically all matrix reductions to specialized forms, as well as iterative procedures on those forms, may be explained in terms of similarity transforms. A matrix $\underline{A}$ is said to undergo such a transformation if it is pre- and postmultiplied by any other matrix along with its inverse. Thus, the matrix $\underline{C}$ is a similarity transform of $\underline{A}$, through the transformation matrix $\underline{T}$ if

$$\underline{C} = \underline{T}\ \underline{A}\ \underline{T}^{-1}$$

Similarity transformations also apply if the order of the multiplication is reversed as in

$$\underline{D} = \underline{T}^{-1}\ \underline{A}\ \underline{T}$$

It should be pointed out, however, that although both of these operations are classified as similarity transforms, $\underline{C}$ is not equal to $\underline{D}$.

The most important property of such transformations is that the eigenvalues of the transformed matrix are identical to those of the original. This can easily be shown by comparing their characteristic polynomials.

$$\begin{aligned}
&\det |\underline{T}^{-1}\ \underline{A}\ \underline{T} - \lambda\ \underline{I}| = \det |\underline{T}^{-1}\ \underline{A}\ \underline{T} - \lambda\ \underline{T}^{-1}\ \underline{T}\ \underline{I}| \\
&= \det |\underline{T}^{-1}\ \underline{A}\ \underline{T} - \underline{T}^{-1}\ \lambda\underline{I}\ \underline{T}| = \det |\underline{T}^{-1}\ (\underline{A} - \lambda\underline{I})\ \underline{T}| \\
&= \det |\underline{T}^{-1}|\ \det |\underline{A} - \lambda\underline{I}|\ \det |\underline{T}| = \det |\underline{A} - \lambda\underline{I}|
\end{aligned}$$

Thus, the characteristic polynomial of the transformed matrix is identical to that of the original matrix $\underline{A}$, and consequently the eigenvalues must be identical as well. In addition it can be shown that similarity transforms produce a new set of eigenvectors which are, however, simply related to the originals by the transform matrices [4]. Beginning with the standard eigensystem equation,

$$\underline{A}\ \underline{x} = \lambda\ \underline{x}$$

we can say that

$$\underline{T}\ \underline{A}\ \underline{x} = \lambda\ \underline{T}\ \underline{x}$$

where $\underline{T}$ is the transform matrix. If we now let

$$\underline{T}\ \underline{x} = \underline{z}$$

then

$$\underline{T}\ \underline{A}\ \underline{T}^{-1}\underline{z} = \lambda\underline{z} \quad \text{or} \quad \underline{B}\ \underline{z} = \lambda\ \underline{z}$$

From this we see that a new eigenproblem has been created with the matrix $\underline{B}$ similar to $\underline{A}$, with $\underline{z}$ simply related to $\underline{x}$, and again with the identical eigenvalues.

The sub-class of similarity transformations which are most applicable to this study are those known as orthogonal transforms. This simply means that the matrices used for the transformations are constructed to be orthogonal, and consequently have the property of preserving symmetry through the transformation process. This is, of course, contingent upon the symmetry of the original matrix which, in this study, is always the case. To show that orthogonal transformations do actually preserve symmetry, assume that there exists a symmetric matrix $\underline{A}$ which is to suffer an orthogonal transformation of the form

$$\underline{B} = \underline{T}\ \underline{A}\ \underline{T}^{-1}$$

Now since $\underline{T}$ is orthogonal, the inverse operation may be replaced by the transpose with no change in the mathematical operations. It must then be shown that the transpose of $\underline{B}$ is equal to $\underline{B}$ itself. Thus,

$$\underline{B}^{T} = (\underline{T}\ \underline{A}\ \underline{T}^{T})^{T} = \underline{T}\ (\underline{T}\ \underline{A})^{T} = \underline{T}\ \underline{A}^{T}\ \underline{T}^{T}$$

Since $\underline{A}$ is symmetric, it is equal to its transpose, giving

$$\underline{B}^T = \underline{T}\ \underline{A}\ \underline{T}^T = \underline{T}\ \underline{A}\ \underline{T}^{-1} = \underline{B}$$

Consequently, symmetry is indeed preserved. Also since the above operations, aside from being orthogonal, are still classified as similarity transforms, the eigenvalues and eigenvectors are also preserved. If it is possible, then, to adhere to these orthogonal similarity transforms throughout the reduction process, we are assured that the condition of the eigenproblem is not affected. Let us now quickly examine the specific types of reductions used in this study for these operations.

4.2.2 Householder Reductions

As mentioned earlier, the first step in the reduction process is the use of orthogonal transformations to arrive at the similar tridiagonal form. The Householder method [2-4] accomplishes this quite efficiently by producing transform matrices which simultaneously zero, or annihilate, large portions of a given row and corresponding column, thus forcing the matrix closer to tridiagonal. Through a series of such operations, all unwanted elements are annihilated, resulting in the required tridiagonal form. The actual construction method for the transform matrices will not be presented here, but will be covered in section 5 under the subject of implementation. The aim here is simply to present the overall computational picture.

The main ingredient of the Householder reduction is a symmetric orthogonal matrix $\underline{P}$ which acts on a given vector $\underline{x}$ such that

$$\underline{P}\,\underline{x} = k\,\underline{e}_1$$

Here $\underline{e}_1$ is a vector whose first element is unity with all others being zero, and k is the length of the vector $\underline{x}$. In other words, the matrix $\underline{P}$ annihilates all elements of $\underline{x}$ except the first. Thus, if we are given some particular vector, it is possible to construct the matrix $\underline{P}$ such that the above equation holds. In the context of matrix reduction, then, imagine an N x N symmetric matrix $\underline{A}$ which is to be reduced to tridiagonal form. Let us choose as our vector the last N-1 elements from the first column of $\underline{A}$, and from it construct the N-1 x N-1 matrix $\underline{P}$. Once this is complete, the matrix $\underline{P}$ is augmented as shown in figure 4.2 to produce the N x N matrix $\underline{T}$. Note that the augmentation destroys neither the orthogonality nor the symmetric nature of the matrix. Consider now the effect of the multiplication $\underline{T}\,\underline{A}$ which is shown in figure 4.3. In effect, the first column of the product is now in an acceptable tridiagonal-like form.

At this point it must be recalled that these transformations must also preserve the conditions of the eigenproblem at hand, or in other words must be similarity transforms. The multiplication $\underline{T}\,\underline{A}$ is definitely not such a transform and consequently there must be a postmultiplication by $\underline{T}$ to rectify that problem. This operation is shown in figure 4.4 and, as can

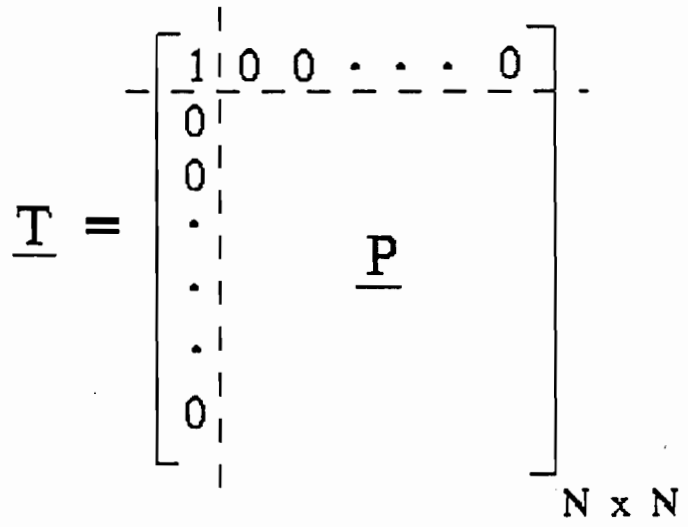


Figure 4.2 Form of Householder Transform Matrix

$$\underline{T}\,\underline{A} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & & & & \\ 0 & & & & \\ \vdots & & \underline{P} & & \\ 0 & & & & \end{bmatrix}_{N \times N} \cdot \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & & & \\ \vdots & & & \\ a_{N1} & & & \end{bmatrix}_{N \times N}$$

$$= \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ k & & & \\ 0 & & & \\ 0 & & \underline{B} & \\ \vdots & & & \\ 0 & & & \end{bmatrix}_{N \times N}$$

Figure 4.3 Effect of Premultiplication of $\underline{A}$ by $\underline{T}$

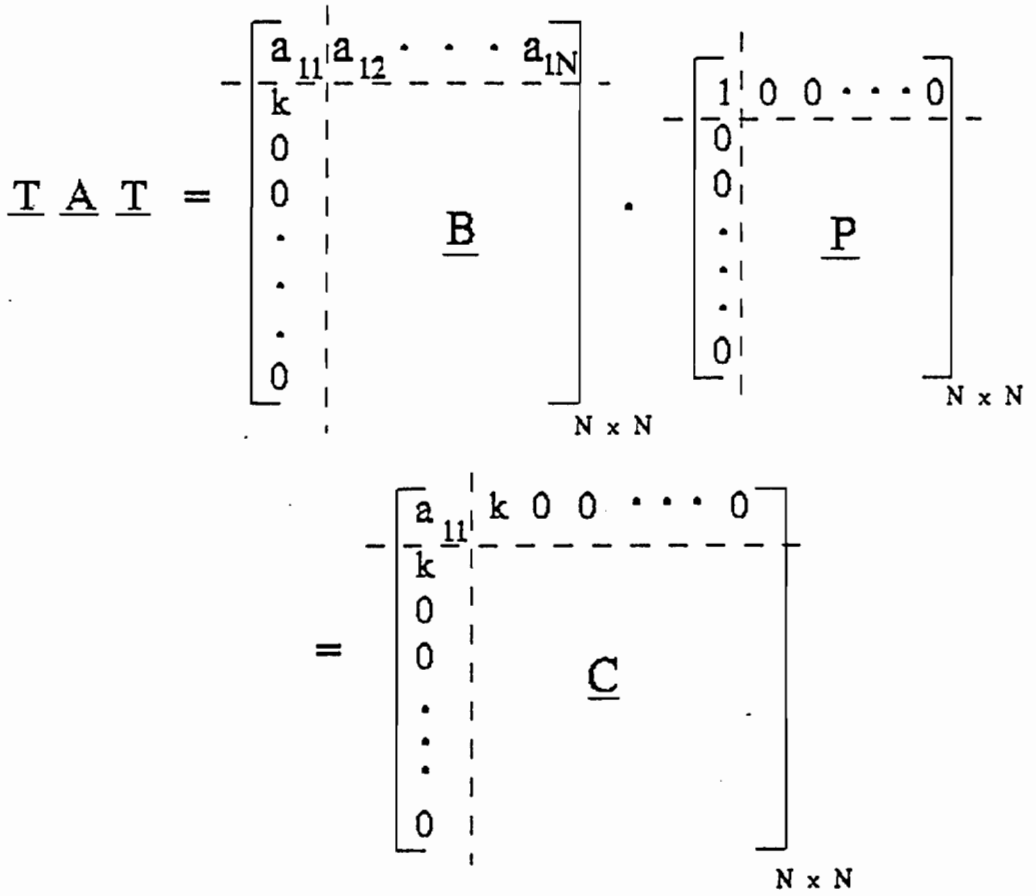


Figure 4.4 Result of First Householder Reduction

be seen, has produced precisely the desired result. Not only has the form of the first column been preserved, but the first row now also appears in the proper form. This symmetry could also have been inferred since the matrix $\underline{A}$ is symmetric and since orthogonal transformations preserve symmetry.

Since we now have produced all the zeros in the first row and column which are needed, we turn our attention to the second row. The procedure is essentially the same as that just performed. Choose our vector to be the last N-2 elements of the second column, and use it to construct the new N-2 x N-2 $\underline{P}$ matrix. This matrix is denoted as $\underline{P}_2$, and is augmented as shown in figure 4.5 to create the new transform matrix $\underline{T}_2$. The resulting products $\underline{T}_2\underline{T}\ \underline{A}\ \underline{T}$ and $\underline{T}_2\underline{T}\ \underline{A}\ \underline{T}\ \underline{T}_2$are shown in figure 4.6 and 4.7 respectively and, as shown, produce another row and column resembling the tridiagonal form.

From these considerations it is obvious that a series of N-2 such transformations will reduce the original N x N matrix completely to tridiagonal form. The Householder algorithm is indisputably the most efficient method known for such matrix reductions [4]. Let us now quickly consider the final step in the reduction process; that of reducing the tridiagonal matrix completely to diagonal form.

4.2.3 QL Algorithm

$$\underline{T}_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdot & \cdot & \cdot & 0 \\ 0 & 1 & 0 & 0 & \cdot & \cdot & \cdot & 0 \\ 0 & 0 & & & & & & \\ 0 & 0 & & & & & & \\ \cdot & \cdot & & & & & & \\ \cdot & \cdot & & & \underline{P}_2 & & & \\ \cdot & \cdot & & & & & & \\ 0 & 0 & & & & & & \end{bmatrix}_{N \times N}$$

Figure 4.5 Transformation Matrix for Second Householder Reduction

$$\underline{T}_2\underline{T}\ \underline{A}\ \underline{T} = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & & & & \\ 0 & 0 & & & & \\ \vdots & \vdots & & \underline{P}_2 & & \\ \vdots & \vdots & & & & \\ 0 & 0 & & & & \end{bmatrix}_{N \times N} \cdot \begin{bmatrix} a_{11}k & 0 & 0 & \cdots & 0 \\ k & c_{11} & c_{12} & \cdots & c_{1,N-1} \\ 0 & c_{21} & & & \\ 0 & c_{31} & & & \\ \vdots & \vdots & & & \\ \vdots & \vdots & & & \\ \vdots & \vdots & & & \\ 0 & c_{N-1,1} & & & \end{bmatrix}_{N \times N}$$

$$= \begin{bmatrix} a_{11}k & 0 & 0 & \cdots & 0 \\ k & c_{11} & c_{12} & \cdots & c_{1,N-1} \\ 0 & k_2 & & & \\ 0 & 0 & & & \\ \vdots & \vdots & & & \\ \vdots & \vdots & & & \\ 0 & 0 & & & \end{bmatrix}_{N \times N}$$

Figure 4.6 Second Stage of Householder Transformation

Premultiplication by Second Transform Matrix

$$\underline{T}_2\underline{T}\,\underline{A}\,\underline{T}\,\underline{T}_2 = \begin{bmatrix} a_{11}k & 0 & 0 & \cdots & 0 \\ k & c_{11} & c_{12} & \cdots & c_{1,n-1} \\ 0 & k_2 & & & \\ 0 & 0 & & & \\ \vdots & \vdots & & & \\ 0 & 0 & & & \end{bmatrix}_{N \times N} \cdot \left[\begin{array}{cc|ccc} 1 & 0 & 0 & 0 & \cdots 0 \\ 0 & 1 & 0 & 0 & \cdots 0 \\ \hline 0 & 0 & & & \\ 0 & 0 & & & \\ \vdots & \vdots & & \underline{P}_2 & \\ & & & & \\ 0 & 0 & & & \end{array}\right]_{N \times N}$$

$$= \left[\begin{array}{cc|ccc} a_{11}k & 0 & 0 & \cdots & 0 \\ k & c_{11} & k_2 & 0 & \cdots 0 \\ \hline 0 & k_2 & & & \\ 0 & 0 & & & \\ \vdots & \vdots & & \underline{D} & \\ & & & & \\ 0 & 0 & & & \end{array}\right]_{N \times N}$$

Figure 4.7 Completed Second Stage of Householder Reduction

Once the matrix in question has been reduced to tridiagonal form, one of the most efficient methods known today for complete diagonalization is the QL algorithm [3]. This procedure is contained in a large class of operations known as factorization methods.

The QL algorithm relies on the fact that any real matrix may be decomposed into the form

$$\underline{A} = \underline{Q}\,\underline{L}$$

where $\underline{Q}$ is orthogonal and $\underline{L}$ is lower triangular. The algorithm makes use of this fact and actually produces a sequence of matrices according to

$$\underline{A}_i = \underline{Q}_i \underline{L}_i \quad , \quad \underline{L}_i \underline{Q}_i = \underline{A}_{i+1} \quad , \quad i = 0,1,2,\ldots$$

Notice that since

$$\underline{A}_{i+1} = \underline{L}_i \underline{Q}_i = \underline{Q}_i^T \underline{A}_i \underline{Q}_i$$

each step of the algorithm is again a similarity transform and thus preserves the conditions of the eigenproblem.

The most important property of this algorithm is that it is based on the following fact. It can be shown that as i approaches infinity, $\underline{A}$ will tend toward a lower triangular matrix with the eigenvalues along the diagonal. Furthermore, if the initial matrix A is symmetric, which is the case in this study, the limiting form will be a purely diagonal matrix (with the eigenvalues along that diagonal). This is, of course, due

to the fact that similarity transforms preserve symmetry.

The algorithm as described thus far will produce the desired results, but the convergence to diagonal is somewhat slow if any of the eigenvalues have nearly identical numerical values [2]. Because of this, the algorithm is modified to take advantage of shifting operations as follows:

$$\underline{A}_i - k_i\underline{I} = \underline{Q}_i\,\underline{L}_i \quad , \quad \underline{L}_i\underline{Q}_i + k_i\underline{I} = \underline{A}_{i+1} \quad , \quad i = 0,1,2,\ldots$$

These operations do not destroy the similarity nature of the transforms, and the convergence is much faster. The algorithm automatically chooses the k parameters at each stage to accelerate the convergence to diagonal form. These aspects of the algorithm will be explained more fully when the actual implementation is discussed. The final result of this algorithm, nevertheless, is to produce the eigenvalues and eigenvectors of the original problem.

4.3 Eigenproblem Review

As a review of the operations which have been discussed in this section, let us consider the original symmetric matrix, $\underline{A}$, which is to be reduced. The matrix is first attacked by the Householder algorithm which, through a finite number of orthogonal transforms, is reduced to tridiagonal form. At this point, the tridiagonal matrix is subjected to the QL algorithm which iterates the matrix (by similarity transforms) ever

closer to purely diagonal form. The result of this operation is to produce the eigenvalues of $\underline{A}$ along the resulting diagonal. The eigenvectors, on the other hand, are retrieved by accumulating the totality of transform matrices used to reach the diagonal form. This applies not only to the QL algorithm, but to the Householder reductions as well.

To be explicit, if we consider the transform matrices from the beginning, and denote them as

$$\underline{T}_1 \, , \, \underline{T}_2 \, , \, \underline{T}_3 \, , \cdots , \, \underline{T}_L$$

then the eigenvectors of the problem are the columns of the product matrix formed by the transformations:

$$\underline{E} = \underline{T}_1 \, \underline{T}_2 \, \underline{T}_3 \cdots \underline{T}_L$$

These operations are, of course, taken care of in a piecewise fashion as the algorithms operate on the input matrix.

4.4 A Numerical Example

In order to better understand the concepts presented in this section, a short numerical example will be presented. A simple 3 x 3 real matrix will be diagonalized using the algorithms mentioned above. Several intermediate steps leading to the diagonalization are explicitly shown with explanations of their significance.

Consider the original matrix, $\underline{A}$, shown in figure 4.8. Notice that the matrix is symmetric, and will consequently be transformed into a tridiagonal form by the Householder algorithm. Note also the use of a real matrix for this example even though the attention of this section has been focused on complex Hermitian matrices. In reality, the routines used in this study operate only on real matrices. Because of this, any complex eigenproblem must first be reduced to a corresponding real problem before the Householder or QL algorithms are administered. Once the algorithms have done their respective operations, the problem is returned to the complex domain. This procedure will be fully explained in section 5, but is pointed out here simply to account for the discrepancy.

The first operation to be performed on the matrix is the reduction to tridiagonal form by use of the Householder algorithm. The transform matrix, $\underline{T}_h$, produced by the algorithm, as well as the resulting tridiagonal matrix, $\underline{B}$, are shown in figure 4.9. As can be seen, the Householder transform matrix is indeed orthogonal, and the resulting tridiagonal form is symmetric. Since the original matrix is only a 3 x 3, this Householder reduction is completed in only one iteration. Recall that an N x N matrix may be reduced to tridiagonal in N-2 iterations.

To reduce the tridiagonal $\underline{B}$ matrix completely to diagonal form, the QL algorithm is now called upon. As shown in figure

$$\underline{A} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & -1 \\ 3 & -1 & 2 \end{bmatrix}$$

Figure 4.8 Original Matrix for Numerical Example

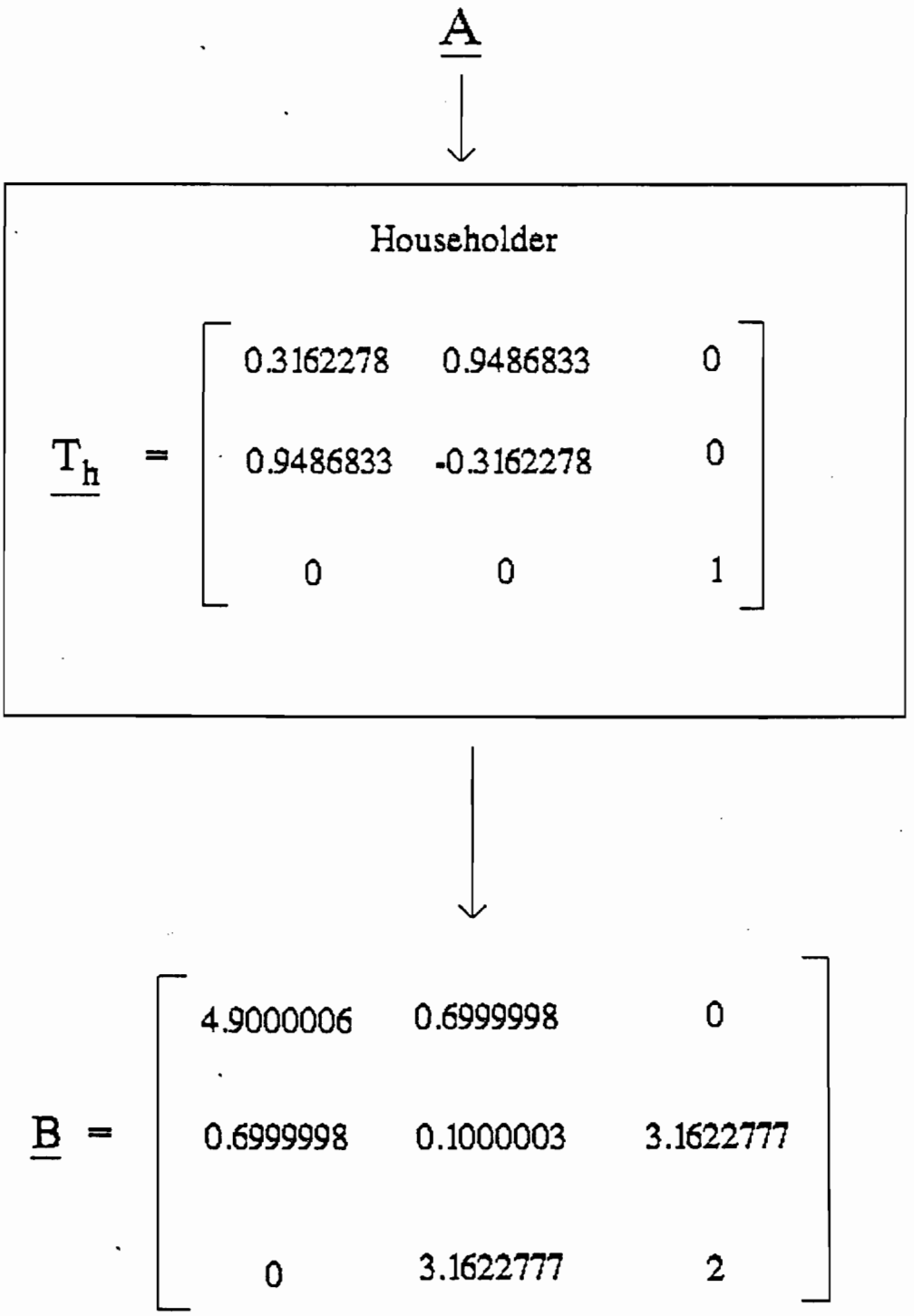


Figure 4.9 Numerical Results of Householder Algorithm

4.10, the tridiagonal matrix is subjected to the QL algorithm, resulting in the final diagonal form denoted as $\underline{C}$. The elements of $\underline{C}$ are actually the three eigenvalues of the original matrix $\underline{A}$. The transformation matrix, $\underline{T}_q$, of the QL algorithm is actually the accumulated transform used to arrive at the diagonal form. Recall that the QL algorithm is iterative in nature, and consequently applies a large number of orthogonal transformations to force matrix $\underline{B}$ into the diagonal form $\underline{C}$. It is because of this property that $\underline{T}_q$ is neither symmetric nor orthogonal. If, however, each of the individual transformations were examined, they would indeed all be orthogonal. In other words, then, $\underline{T}_q$ is simply the matrix product of all transformations used by the QL algorithm.

Finally, the QL algorithm also provides the eigenvector matrix, $\underline{E}$, whose columns are the eigenvectors corresponding to the eigenvalues given in the matrix $\underline{C}$. The $\underline{E}$ matrix is shown in figure 4.11 along with a table giving the eigenvalues (from $\underline{C}$) and the corresponding eigenvectors (from the columns of $\underline{E}$). It is possible to show that for each eigenvalue/eigenvector set, the equality

$$\underline{A}\ \underline{x} = \lambda\ \underline{x}$$

is satisfied. It can also be shown that the eigenvector matrix $\underline{E}$ is related to the transform matrices $\underline{T}_h$ and $\underline{T}_q$ by the relationship

$$\underline{E} = \underline{T}_h\ \underline{T}_q$$

as expected. The theory, then, can indeed be verified in prac-

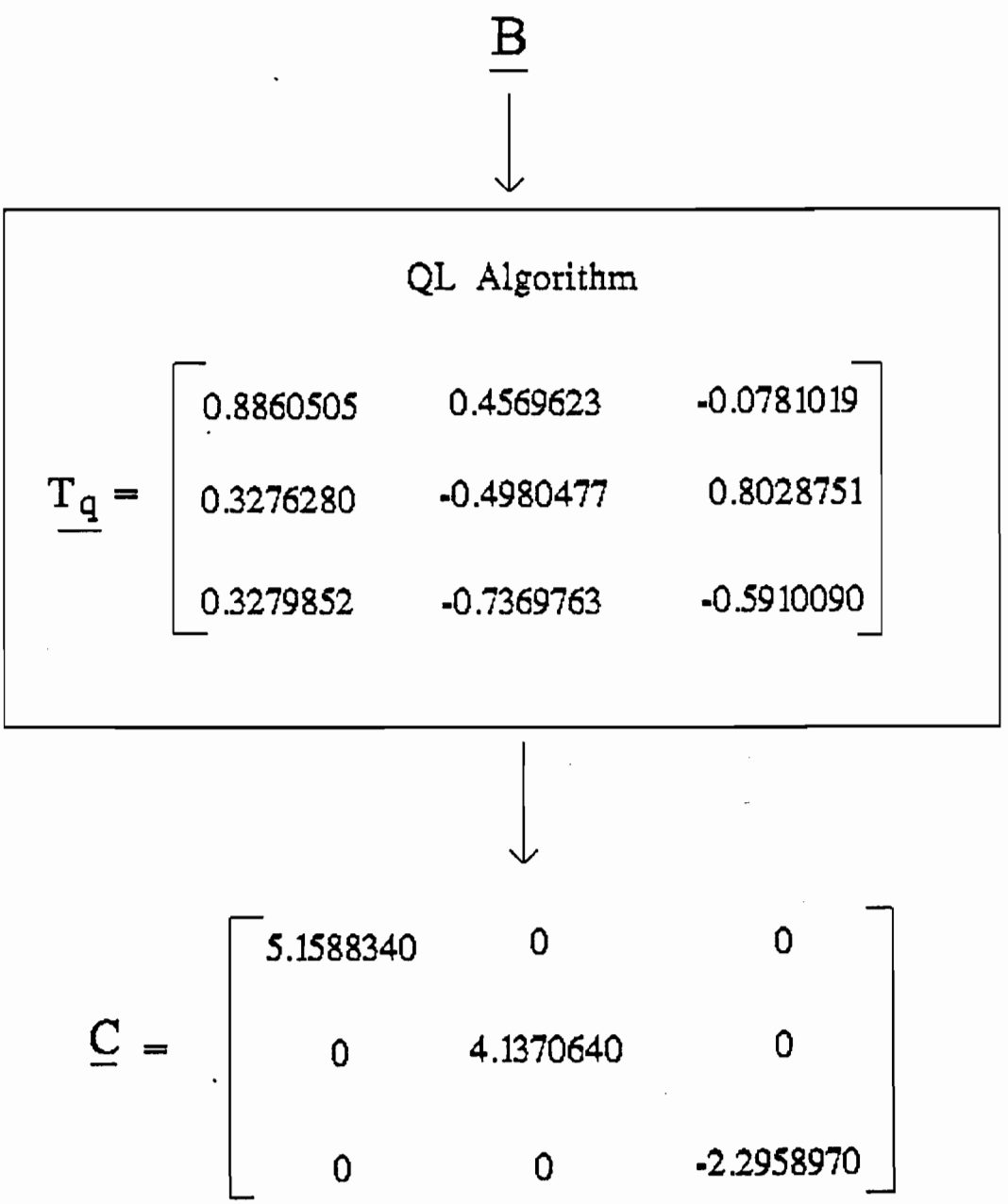


Figure 4.10 Numerical Results of QL Algorithm

$$\underline{E} = \begin{bmatrix} 0.5910090 & -0.3279854 & 0.7369763 \\ 0.7369763 & 0.5910090 & -0.3279854 \\ 0.3279854 & -0.7369763 & -0.5910090 \end{bmatrix}$$

Eigenvalue	5.1588340	4.1370640	-2.2958970
Eigenvectors	0.5910090	-0.3279854	0.7369763
	0.7369763	0.5910090	-0.3279854
	0.3279854	-0.7369763	-0.5910090

Figure 4.11 Eigenvector Matrix and Table of Eigenvalue/Eigenvector Values

tice. This procedure also gives testimony to the accuracy of the numerical routines employed.

5.0 Implementation of Eigenvalue/Eigenvector Algorithm

In this section the implementation of the eigenvector algorithm will be discussed in some detail. The aim is to provide some insight into the support software for the algorithm as well as to better establish the actual algorithm operation. As an introduction, let us first examine the significance of the eigenvalues and eigenvectors in the context of adaptive antenna systems.

5.1 Eigenvalue/Eigenvector Significance

To bridge the gap between theory and application, let us first consider the meaning of eigenvalues and eigenvectors in terms of the adaptive system. This will also more clearly define the overall problem at hand. Shown in Figure 5.1 is the heart of the system, consisting of the sensor elements, weighting network, and summer. The output signal, y, can be expressed as

$$y = w_1 x_1 + w_2 x_2 + \ldots + w_N x_N$$

In terms of the weight vector $\underline{w}$ and the signal vector $\underline{x}$, this expression may be written compactly as

$$y = \underline{w}^T \underline{x} = \underline{x}^T \underline{w}$$

The average power of y may be represented as the expected value of y multiplied by its complex conjugate, and consequently,

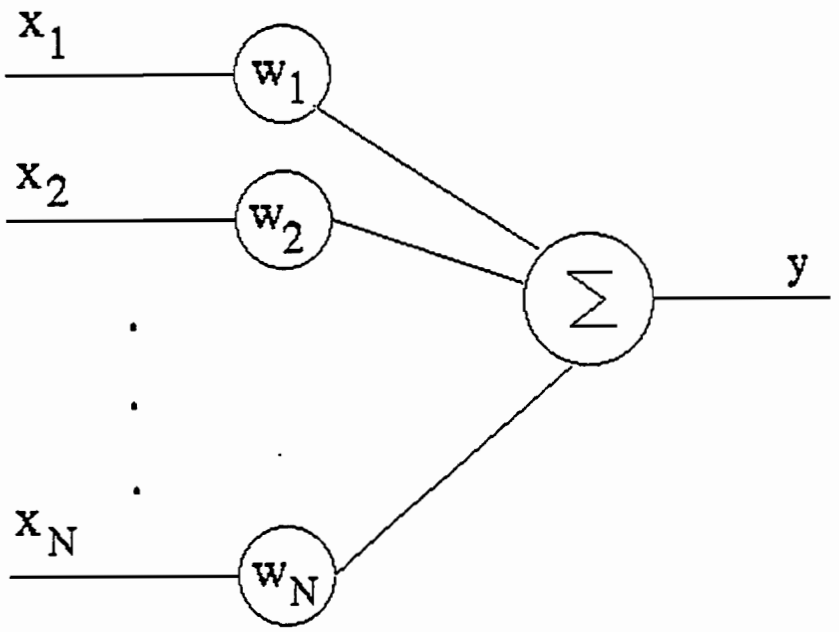


Figure 5.1 Heart of Adaptive System

$$P_{out} = E\{y^* \; y\} = E\{ \underline{w}^{T^*} \; \underline{x}^* \; \underline{x}^T \; \underline{w} \}$$

Since the weight vectors are assumed constants here, the averaging operation may be taken inside to give

$$P_{out} = \underline{w}^{T^*} \; E\{ \underline{x}^* \; \underline{x}^T \} \; \underline{w}$$

The inner quantity can now be recognized as the signal autocorrelation matrix, and thus the output power can be expressed as

$$P_{out} = \underline{w}^{T^*} \; \underline{R}_{xx} \; \underline{w}$$

From its definition, the autocorrelation matrix is Hermitian. As mentioned in section 4, there are some special properties of the eigenvalues and eigenvectors associated with these matrices. The ones of importance here are that 1) the eigenvectors form a complete orthonormal set and 2) there are exactly N eigenvector/eigenvalue pairs if the dimension of the autocorrelation matrix is also N.

Consider the fundamental equation which defines the eigenquantities of the autocorrelation matrix.

$$\underline{R}_{xx} \; \underline{z}_i = \lambda_i \; \underline{z}_i \qquad i = 1,2,\ldots,N$$

Using this equality, the expression for the output power takes on yet another form. Consider the weight vector set equal to one of the eigenvectors, i.e.,

$$\underline{w} = \underline{z}_i$$

where $\underline{z}_i$ is the i-th eigenvector with corresponding eigenvalue λ_i . In this case, then, the output power may be expressed as

$$P_i = \underline{z}_i^{T^*} \underline{R}_{xx} \underline{z}_i = \underline{z}_i^{T^*} \lambda_i \underline{z}_i = \lambda_i \underline{z}_i^{T^*} \underline{z}_i = \lambda_i$$

where P_i is the array output power with weighting vector $\underline{z}_i$. Each eigenvector $\underline{z}_i$,then, will produce an array output power of P_i , which is just equal to λ_i. Since the eigenvectors are normal, we have

$$|\underline{z}|^2 = 1 \quad \text{or} \quad \underline{z}^{T^*} \underline{z} = 1$$

This fact was used in the above expression to show that the output power is exactly equal to the eigenvalues of the system when the weight vectors are set equal to the eigenvectors.

Considering the eigenvalues in decreasing order of magnitude, it can be shown that the largest represents the maximum power which may be obtained from the array by the use of normalized weights [11]. Any deviation from this weight vector will cause a decrease in output power. The second largest eigenvalue represents the maximum power under the constraint that the largest cannot be used. This progresses in the same manner with the remaining eigenvalues, and thus the n-th largest represents the smallest output power.

If the signals incident on the array are completely uncorrelated, the application of the differing eigenvectors as

weighting elements will result in perfect signal sorting. As is usually the case, however, the signals are not completely uncorrelated, resulting in a degradation of the algorithm performance. This does not mean that the solution is ineffective, but only that some of the unwanted signals will be admitted to the array [11].

The eigenvector sorting algorithm, then, involves finding the eigenvalues and eigenvectors of the autocorrelation matrix. Guided by the eigenvalues, the antenna weights are set equal to the corresponding eigenvectors, causing an improvement in the overall system performance. As will be explained later, the workstation software support for the algorithm provides the user with the list of eigenvalues to choose from. Once a particular choice is made, the corresponding eigenvector is applied as the weight vector, and the resulting antenna pattern may be inspected.

The above treatment will hopefully serve to further clarify the problem, and to form a correspondence between theory and application. Attention will now be turned to the actual algorithm implementation. The aim is first to first consider the outermost layers of software support provided for the algorithm followed by an inward movement to the routines which actually perform the algorithm operations.

5.2 Considerations of Software Support

In the interest of completeness, the software support for the eigenvector sorting algorithm will quickly be discussed. This will also serve to orient the reader as to the nature of the environment in which the algorithm operates. Shown in figure 5.2 are the layers of software involved in the present simulation structure. As can be seen, the eigenvector sorting algorithm has been integrated into the simulation package initially developed by D. Haegert, et. al. [1]. The algorithm, then, resides at the same level as the other three adaptive processors discussed in section 3. Above this simulation software is the uppermost layer which is dedicated essentially to I/O operations. This layer was developed by R. Moats [12] and allows the user to input simulation data and inspect results in an interactive environment. The entire collection of software operates on an IBM-PC, and it is this version which contains the most recent software. Let us now quickly examine the function of the two upper layers of support.

The I/O software creates, in effect, a workstation on the PC which allows the user to interact with the simulations. Through a series of graphics screens, the workstation provides means to enter the required information for the simulations. It is also the job of the workstation to graphically present the simulation output data. The actual user interaction is well documented in [12] and will not be discussed here. Finally, the I/O layer is responsible also for input and output information storage. This is to say that the workstation is able to store a

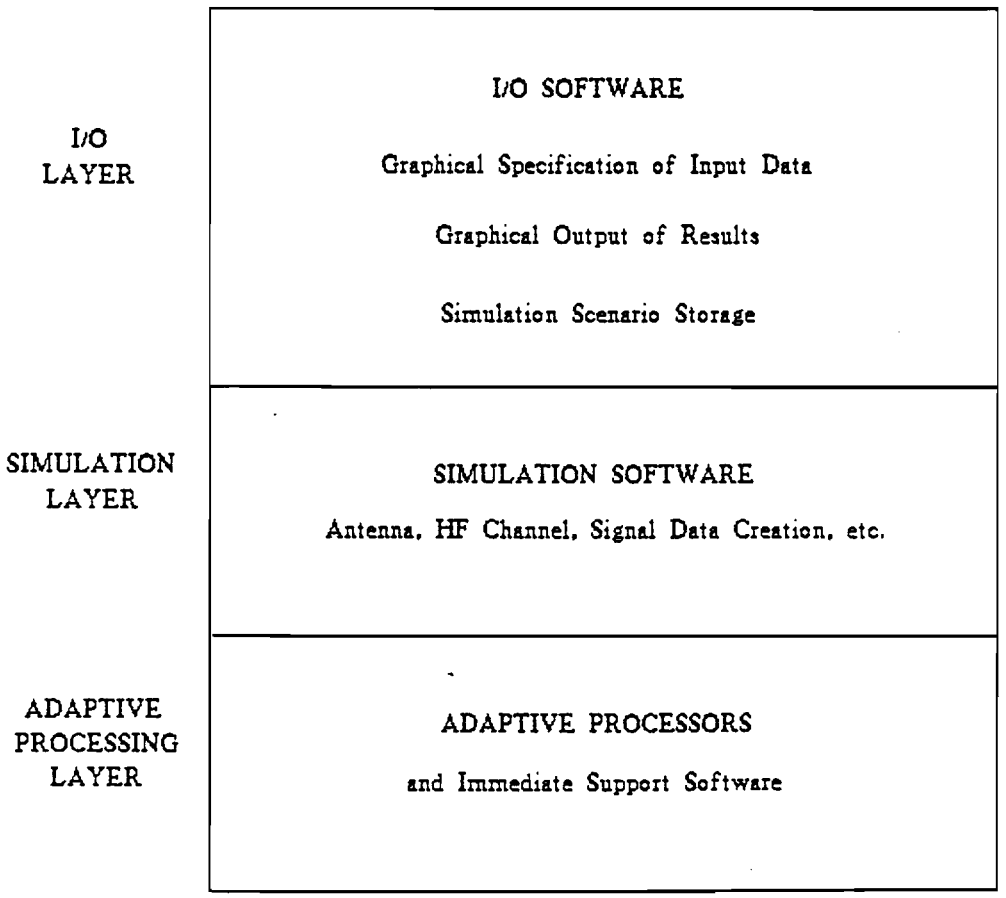


Figure 5.2 Support Software for Adaptive Algorithms

large collection of input parameters and their corresponding results which may then be recalled for future reference. This layer of support is indeed quite useful and was used to create virtually all adapted antenna plots shown in section 6.

The next layer of software, the actual simulation code, is responsible for providing the immediate environment in which the adaptive processors operate. It is at this level that the actual simulation takes place. Example modules contained in this layer are the antenna array, the HF channel, as well as modules which create the information and jamming signals. For the interested reader, the details of this software can be found in [1]. It is not the aim here to completely explain the simulation operations, but simply to provide the necessary information for an understanding of the processor environment.

Perhaps the best way to quickly comprehend the scope of the simulation software is to examine the necessary input data. Shown in Figure 5.3 are the user entered (via the workstation) parameters required by the simulation to run the eigenvector sorting algorithm. As can be seen, these values completely define the antenna, HF channel, as well as the signal and jammer characteristics. Notice that the only piece of data which is directly employed by the eigenvector algorithm is the number of samples used to estimate the autocorrelation matrix. All other information is used by the support software in order to create the proper environment.

ARRAY SYSTEM CHARACTERISTICS

Number of sensors
Length of antennas (m)
Element Locations (rectangular coordinates) (m)
Frequency of operation (MHz)
Adaptive algorithm
1) LMS
2) Constrained LMS
3) Update Covariance
4) (Eigenvector Sort)

HF CHANNEL CHARACTERISTICS

Number of signal paths
Delay of each path (msec)
Attenuation of each path (dB)

SIGNAL CHARACTERISTICS

Arrival angle, Azimuth and Elevation (degrees)
Information rate (bps)
Number of samples per bit

INTERFERENCE CHARACTERISTICS

Number of jammers
Arrival angles, Azimuth and Elevation (degrees)
J/S ratio for each jammer (dB)
S/N thermal ratio (dB)

Number of samples to average Rxx

Figure 5.3 User Entered Parameters for Simulation

Now that the algorithm position in the hierarchy of software has been introduced, let us now turn our attention to the algorithm itself. Included in this discussion will also be the support software dedicated exclusively to the eigenvector algorithm.

5.3 Formulation of the Autocorrelation Matrix

Note: All software described henceforth is included in appendix B of [12].

As shown earlier in this section, the starting point for the algorithm is the estimation of the signal autocorrelation matrix. In order to get a reasonable estimate of this quantity, simple averaging techniques have been employed. Shown in Figure 5.4 is a graphical representation of the procedure. The simulation produces the input signal and jammer samples and sends them through the HF channel and subsequently to the antenna array. The output of the antenna represents the signal values at each of its sensors in complex low pass equivalent form. This information is simply an N dimensional complex vector represented as

$$\underline{x}^T = [\, x_1 \; x_2 \; \ldots \; x_N \,]$$

The signal vector is next employed by the routine "formcorr" which actually formulates a point estimate of the autocor-

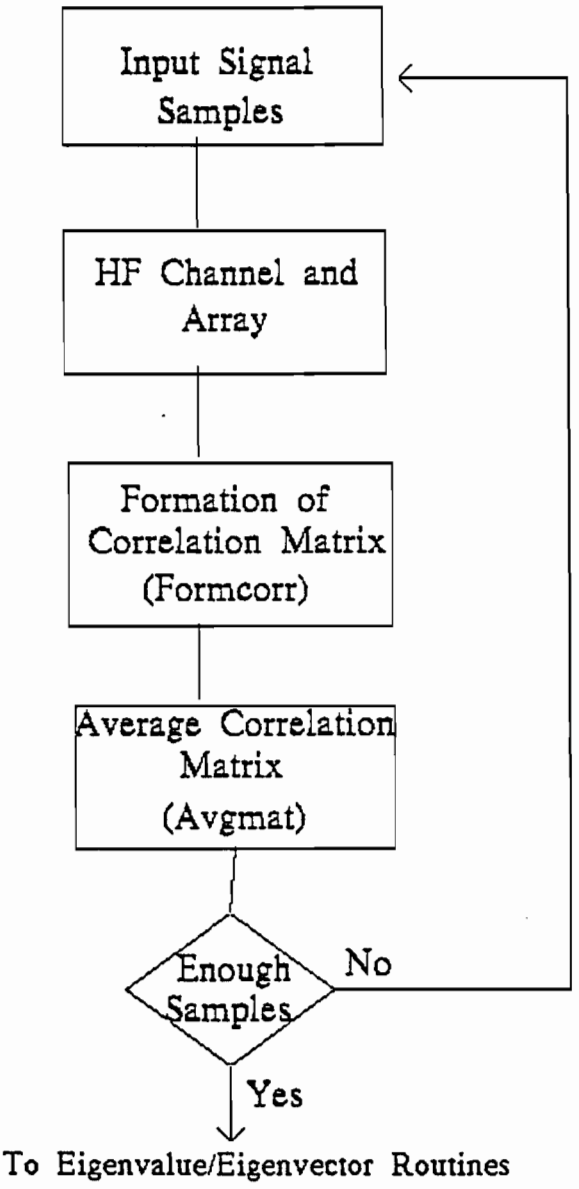


Figure 5.4 Method Used to Obtain R_{xx}

relation matrix according to

$$\underline{R}_{xx}(i,j) = \underline{x}_i^* \underline{x}_j$$

Since we now have a point estimate of the matrix, this quantity is passed to the averaging routine "avgmat". This program contains local memory sufficient for the storage of one 36 x 36 complex matrix. Each time the routine is called, the input matrix is added to that in local storage and the matrix counter is incremented by 1. The counter simply keeps track of the number of matrices which have been summed. On each call, after the addition and counter increment have been performed, the summed matrix is divided by the counter (number of matrices) and output by the routine. In this way, the routine always produces the best average possible at any particular time.

As shown in Figure 5.4, the simulation compares the number of times "avgmat" has been called with the number of averages desired by the user. If an insufficient number have been taken, the program flow is directed back to send the next data sample to the channel and array. If, on the other hand, the averaging is complete, the final autocorrelation matrix average is sent on for further processing. Let us now consider the steps taken by the algorithm once the autocorrelation matrix has been constructed.

Shown in Figure 5.5 is the remaining software flow of the eigenvector sorting algorithm. Comments are provided on the right side of the diagram to clarify the quantities produced at

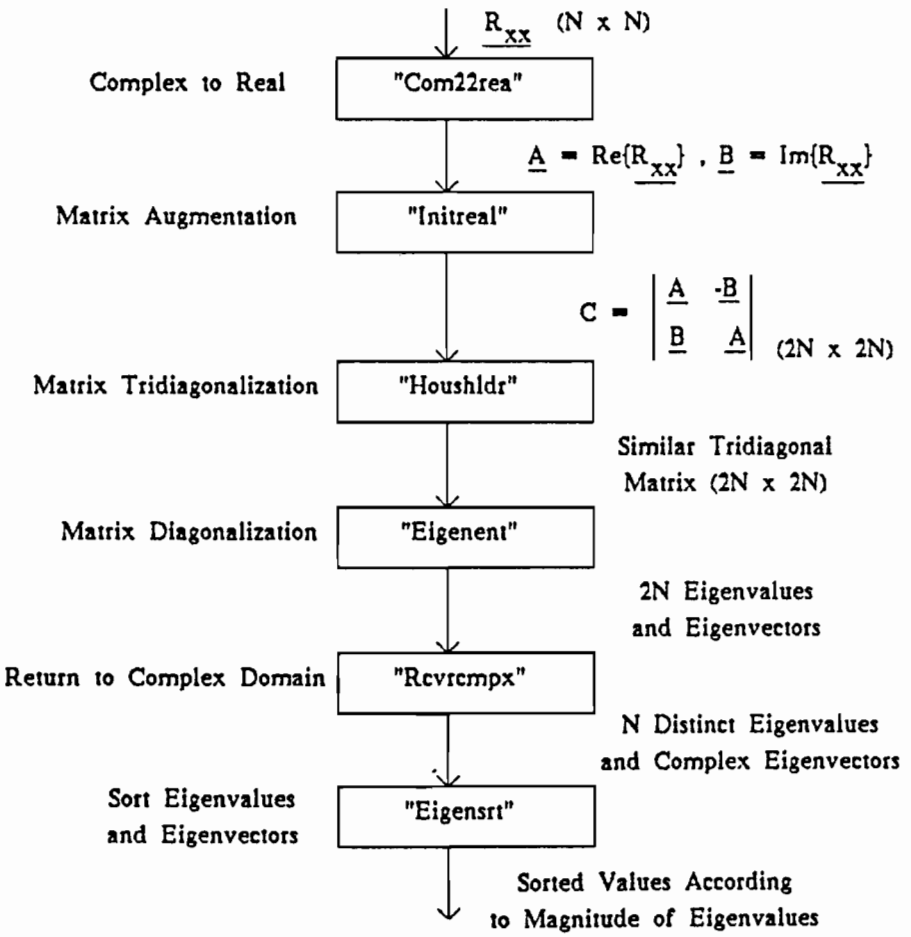


Figure 5.5 Eigenvector Algorithm Central Software

each step. The functional duties of the particular software routines are included to the left of each module, while the actual routine names are within the boxes. Each of the modules will be discussed; some more completely than others, depending on their complexity.

5.4 Transformations From Complex to Real (and Back)

The functions of the first two programs shown in Figure 5.5 are to transform the complex autocorrelation matrix into a real symmetric matrix without destroying the eigenproblem. This transformation is necessary, as mentioned in section 4, since the numerical programs which actually perform the diagonalization operate only on real matrices. There are routines which operate directly on the complex quantities, but they are of no real advantage. To understand the transformation, let us consider the following. We know that the autocorrelation matrix is Hermitian, and thus may be written in the form

$$\underline{R}_{XX} = \underline{A} + j\underline{B}$$

where

$$\underline{A} = \mathrm{Re}\ \{\ \underline{R}_{XX}\ \} \quad \text{and} \quad \underline{B} = \mathrm{Im}\ \{\ \underline{R}_{XX}\ \}$$

It can be shown [2] that the N x N complex problem

$$(A + j\ B)\ (u + j\ v) = \lambda\ (u + j\ v)$$

is equivalent to the 2N x 2N real problem

$$\begin{bmatrix} A & -B \\ B & A \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \lambda \begin{bmatrix} u \\ v \end{bmatrix}$$

Note that the real matrix is symmetric if the autocorrelation matrix is indeed Hermitian. This is due to the fact that

$$\underline{A} = \underline{A}^T \quad \text{and} \quad \underline{B} = -\underline{B}^T$$

The only differences which have been created in the problem by this transformation are the pairing of the eigenvalues and eigenvectors. This means that if the original complex matrix possesses the N eigenvalues

$$\lambda_1, \lambda_2, \ldots, \lambda_N$$

then the augmented matrix has the 2N eigenvalues

$$\lambda_1, \lambda_1, \lambda_2, \lambda_2, \ldots, \lambda_N, \lambda_N$$

The same may be said for the eigenvectors. The real eigenvectors of the augmented problem, for each eigenvalue, are also pairs of the form

$$\begin{bmatrix} u \\ v \end{bmatrix} \text{ and } \begin{bmatrix} -v \\ u \end{bmatrix}$$

In the complex domain, then, the eigenvectors are represented as

$$(u + j v) \quad \text{and} \quad j (u + j v)$$

which are identical except for the inessential phase. To effectively solve the original problem, then, we need only

choose one eigenvalue and one eigenvector from each pair. These will represent the solutions for the original complex autocorrelation matrix.

In view of the above discussion, it is possible to easily explain the operation of several software modules in Figure 5.5. The function of the program "com22rea" is to separate the original complex autocorrelation matrix into its real and imaginary parts. Likewise, the program "initreal" serves to form the augmented 2N x 2N symmetric matrix given the real and imaginary parts created by "com22rea". At the other end of the flow, when the concern is to recover the complex entities, the program "rcvrcmpx" chooses one eigenvalue and eigenvector from each pair as mentioned above. Thus, it is essentially this program which takes the problem back to the complex domain. It should be realized that the function of this program is not only to choose one pair from each set, but to restore the complex nature of the problem by performing transformations on the eigenvectors of the type

$$\begin{bmatrix} u \\ v \end{bmatrix} \longrightarrow \begin{bmatrix} u + j\ v \end{bmatrix}$$

The problem, at this point, has essentially been solved. The complete set of eigenvalues and complex eigenvectors are now available. In order to facilitate subsequent processing, however, it is expedient to sort the eigenvalues (and their

corresponding eigenvectors) in order of decreasing magnitude. This is the function of "eigensrt", the final program shown in Figure 5.5. With the eigenvalues and eigenvectors thus sorted, it is a much simpler matter for the output software to choose the largest, second largest, or any other desired eigenvalue.

5.5 The Householder and QL Algorithms

The two routines shown in Figure 5.5 which have not yet been discussed form the heart of the eigenvector algorithm. These numerical routines designated as "Houshldr" and "Eigenent" perform the actual matrix diagonalization operations, and are thus responsible for the production of the actual eigenvalues and eigenvectors. Due to their importance, a slightly deeper analysis of their operation is in order.

5.5.1 The Householder Algorithm

As mentioned in section 4, the Householder algorithm makes use of orthogonal transformations in order to reduce a real symmetric matrix to tridiagonal form. If the dimension of the square input matrix is N, the reduction can be performed with N-2 transformations. Recall that the algorithm relies on the production of Householder matrices which act to zero out specific rows and columns of the input matrix. Let us examine specifically how this is accomplished.

The matrices used by the algorithm are of the form

$$\underline{P} = 1 - 2\ \underline{w}\ \underline{w}^T$$

where $\underline{w}$ is a real vector possessing the property that

$$|\underline{w}|^2 = 1$$

This matrix can be shown to be orthogonal since

$$\begin{aligned}\underline{P}^2 &= (1 - 2\ \underline{w}\ \underline{w}^T)\ (1 - 2\ \underline{w}\ \underline{w}^T) \\ &= (1 - 4\ \underline{w}\ \underline{w}^T + 4\ \underline{w}\ (\underline{w}^T\ \underline{w})\ \underline{w}^T) \\ &= 1\end{aligned}$$

Because of this , the matrix $\underline{P}$ is equal to its own inverse, proving orthogonality. It is somewhat of a problem, however, to insist that the vector $\underline{w}$ be a unit vector. To take care of this, $\underline{P}$ may be rewritten in the form

$$\underline{P} = 1 - (\underline{u}\ \underline{u}^T\ /L)$$

where L is defined as

$$L = 1/2\ |\underline{u}|^2$$

With this arrangement, $\underline{u}$ can now be any vector, not restricted by its length.

Recall that it is the duty of the $\underline{P}$ matrix to operate on a given vector $\underline{x}$ and zero all elements except the first. To see that this is actually possible, a vector $\underline{x}$ is chosen at random, and the vector $\underline{u}$ is defined as

$$\underline{u} = \underline{x} - |\underline{x}|\ \underline{e}_1$$

where $\underline{e}_1 = [1,0,0,\ldots,0]^T$. With these definitions, then,

$$\begin{aligned}
\underline{P}\ \underline{x} &= (1 - (\underline{u}\ \underline{u}^T / L))\ \underline{x} \\
&= \underline{x} - (\underline{u}/L)\ (\underline{x} - |\underline{x}|\underline{e}_1)\ \ \underline{x} \\
&= \underline{x} - \underline{u}\ (|\underline{x}|^2 - |\underline{x}|\ x_1)/(|\underline{x}|^2 - |\underline{x}|\ x_1) \\
&= \underline{x} - \underline{u} \\
&= |\underline{x}|\ \underline{e}_1
\end{aligned}$$

Thus, we see that the matrix $\underline{P}$ has indeed produced a vector containing all zeros except for the first element.

Following this procedure, as outlined in section 4, the Householder algorithm chooses its $\underline{x}$ vectors from the matrix to be reduced. As shown there, the first vector is comprised of the N-1 lower elements of the first column which, after the complete orthogonal transformation, causes zeros to appear in the last N-2 positions of the first row and column. Likewise, the second transform matrix is constructed, by the same method, from the last N-2 elements of the second column. This, of course, produces zeros in the last N-3 positions of the second row and column. This procedure is continued for N-2 iterations at which point the matrix is in the desired tridiagonal form.

5.5.2 The QL Algorithm

Once the Householder reductions are performed, the reduced matrix is passed on to the QL algorithm for complete

diagonalization. This routine is denoted as "Eigenent" in Figure 5.5. As mentioned in section 4, this is not a finite algorithm, and consequently the victim matrix is iterated through a series of transformations until it is diagonal to machine precision. It is thus not a simple matter to determine a priori the amount of work involved for any given matrix. The convergence rate of the QL algorithm is usually quite good, but does depend on the closeness of the eigenvalues involved. If several eigenvalues are very similar in numerical value, convergence will be slowed somewhat.

The actual operation of the QL algorithm is based on several nonobvious lemmas and theorems and consequently will not be discussed in detail. The algorithm operation is explained in [2], and the interested reader is encouraged to make use of this excellent development. The algorithm has caused a few minor problems in the implementation, however, and as a final note, these will be discussed.

5.6 Numerical Problems

The eigenvector sorting algorithm along with its simulation support software has been implemented on both the VAX 11/750 and the IBM-PC. During the earlier stages of development and testing, only the VAX version had been implemented. The algorithm worked well in the VAX environment and the numerical routines "Houshldr" and "Eigenent" performed with no problems.

When the package was implemented on the IBM, however, some problems did develop with the QL algorithm "Eigenent". Because of its iterative nature, the program is forced to work with extremely small numbers. Since there is no analytic method for the diagonalization, the off diagonal elements are pushed to ever smaller values. Due to a squaring operation in the algorithm, the IBM reported that numbers were being generated which did not fall within its range of representation. For this reason, the algorithm had to be modified slightly to stop the iteration before the quantities became too small. Although this was a mandatory step, the accuracy of the routine must necessarily have suffered. The harm done, however, was not great since very good agreement still exists between the VAX and IBM versions.

It has been shown in this section the methods used to implement the eigenvector sorting algorithm. As mentioned earlier, the actual software which was described is not contained in this report, but resides with all other simulation software in appendix B of the R. Moats project report[12]. In the next section, some results of the eigenvector algorithm will be presented along with conclusions and recommendations.

6.0 Simulation Results

In this section the simulation results for the eigenvector sorting algorithm will be presented. The results were obtained by use of the simulation system described and referenced earlier in this report. Throughout the testing procedure, several input parameters such as the antenna geometry and the channel characteristics will remain constant. These are shown in figure 6.1 which will serve as a reference for all tests. The remaining parameters which are not constant, such as signal directions and powers, will be specified at the beginning of each test.

Referring to Figure 6.1, note that the HF channel model contains three signal paths with roughly 1 ms interpath delays. This model was also used by Haegert [1] in order to simulate an HF channel possessing moderate delay characteristics. The antenna array used by the simulation contains nine whip antennas arranged in a 3 x 3 square arrangement. The spacing between adjacent elements is 5 meters, which corresponds to a half-wavelength at the operating frequency of 30 MHz.

The results of the simulations are given in the form of antenna patterns which show the action of the adaptive algorithm. The antenna patterns are produced in polar form with the angular coordinates theta and phi described in Figure 6.2. As seen, theta refers to the elevation angle of a particular point

Channel Characteristics

Number of Paths	3
Path Delays (ms)	0.0000
	0.8333
	1.6667
Path Attenuations (dB)	0.0
	0.0
	0.0

System Characterisics

Number of Samples/Bit	16
Information Rate (bps)	300
S/N (dB)	10

Antenna Characteristics

Number of Elements	9
Operating Frequency (MHz)	30
Element Type	Dipole
Element Length-1/2 Dipole (m)	18.288
Element Locations (m)	Shown Below

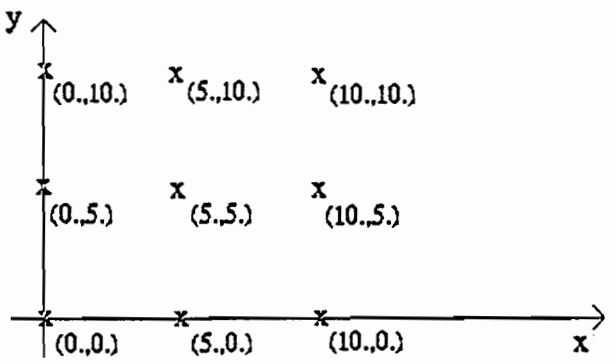


Figure 6.1 Constant Simulation Parameters

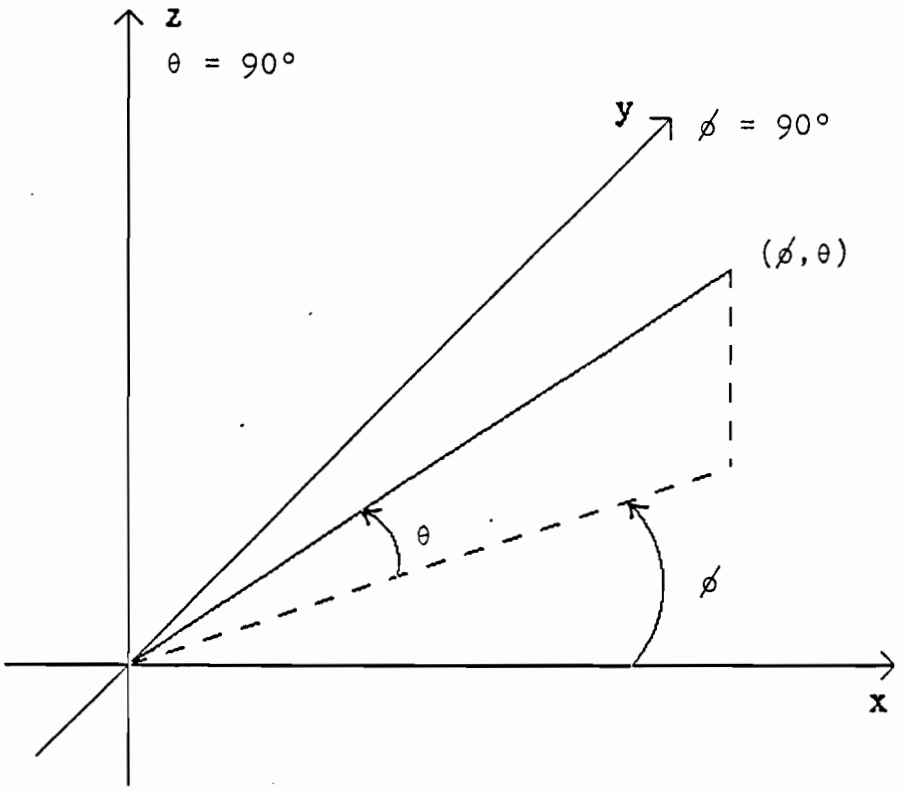


Figure 6.2 Angular Coordinates Used in Plots

from the x-y plane, while phi describes the azimuthal location of the point. The plots consist of both constant azimuth and constant elevation patterns. These are generated by simply holding one coordinate fixed while sweeping over the other.

As mentioned in section 5, it was necessary to incorporate the eigenvector sorting algorithm into a simulation system which had previously supported three other algorithms. The operation of these algorithms is quite dissimilar from the eigenvector sort, and consequently the simulation system is somewhat less than ideal for the complete testing of the new algorithm. The simulation is constrained to operate with one friendly communicator, which is BPSK, and a user specified number of white noise jammers. It would be interesting to supply the algorithm with a wider range of signal environments for a further assessment of its capabilities.

Collectively, the tests are designed to investigate the capability of the eigenvector sorting algorithm for the separation of signals incident on the array. In particular, however, most of the emphasis is directed at an evaluation of the algorithm's ability to eliminate jamming signals from the environment. In other words, we will be most interested in the cases for which the algorithm causes a definite nulling in the antenna pattern at the jammer directions. In these cases, the desired signal, as opposed to an unwanted jammer, has been chosen by the algorithm. Some attention will be given to the

eigenvalues and eigenvectors which result in the selection of jamming signals, however. These really have no significance in terms of system performance improvement for this simulation, but in other environments this action could be useful.

In order to clarify a possible source of confusion before the results are presented, it is necessary first to establish an important difference in the eigenvector sort operation as compared to the other algorithms discussed. As a reference, the first test compares the adapted antenna patterns of the eigenvector sort to those obtained by Haegert [1] using the LMS, Constrained LMS, and Update Covariance algorithms. All signal directions and simulation parameters are identical to the Haegert tests. The LMS, Constrained LMS, and Update Covariance algorithms are primarily closed loop systems which periodically update the antenna weights as the adaptation process progresses. The eigenvector sort algorithm, on the other hand, does not continually update the weights, but operates in an open loop fashion to form an estimate of the autocorrelation matrix. It is the responsibility of the user to specify the number of samples to use in this matrix estimation. Once the simulation has performed the required number of averages, the estimated autocorrelation matrix is then used in a one-time operation which computes the required set of eigenvectors. This set, when applied as weights, hopefully causes acceptable signal sorting. The concept of an iteration, then, refers to an updating of the weights or an improvement of the autocorrelation matrix, depen-

ding upon the algorithm in question.

As mentioned above, the first test compares the operation of the eigenvector sorting algorithm to that of the three algorithms of the Haegert tests. Since those algorithms, especially the LMS, are somewhat of a standard, this gives a solid point of reference for the evaluation of the eigenvector sorting algorithm. This test will also point out that even though the results for the LMS, Constrained LMS, and Update Covariance algorithms are quite similar, the results for the eigenvector sort are very different indeed.

The second test is similar to the first except that another jammer is added to the scenario. This test is no longer concerned with the previously investigated algorithms, and as for the remaining tests, the eigenvector sort is tested on its own merit. The total number of signals present at the array for this investigation is four, consisting of a friendly communicator and three jamming signals.

To determine the effect of autocorrelation matrix averaging, the third test examines the adapted antenna plots as a function of the number of samples used. As would be expected, the larger averaging times do lead to more definite signal separations, not only for the desired signal but also for the jammers. It is also apparent from this test that the resulting antenna pattern is actually quite sensitive to small changes in

the number of averages.

As a final test of the algorithm, the sensitivity in terms of signal separation is investigated. It is determined that higher numbers of averages are required as the signals are moved into closer proximity to one another. Since the algorithm only obtains information from the signal autocorrelation matrix, this does seem logical.

6.1 Test 1: Comparison of Algorithms

In this preliminary test, the results of the eigenvector sorting algorithm are compared with those of the LMS, Constrained LMS, and Update Covariance algorithms. This should be instructive due to the entirely novel operation mode of the eigenvalue sort as compared to the other algorithms. Examination of the plots reveals this fact, since there is indeed very little similarity present.

The signals for this test consist of a friendly communicator whose transmission mode is BPSK, as well as two white noise jamming signals. The angles of incidence and the signal powers for this test are shown in Table 6.1

It should be noted that the unadapted antenna patterns are different for the eigenvector sorting algorithm. In the case of the three other algorithms, the initial pattern is adjusted such that the desired signal is given an advantage over the hostile jammers. For the eigenvector algorithm, however, it was determined that this method is unacceptable, and consequently, the weights are simply adjusted to unity gain and zero phase.

Perhaps the most striking aspect of the eigenvector algorithm, in comparison to the others, is the method by which the signals are chosen by the adapted patterns. The antenna patterns for the largest three eigenvalues are shown and, in fact,

SIGNAL	AZIMUTHAL ANGLE	ELEVATION ANGLE	POWER
COMMUNICATOR	90.0	30.0	0 dB
JAMMER 1	90.0	10.0	20 dB
JAMMER 2	150.0	15.0	20 dB

Table 6.1 Signal Characteristics for Test 1

these are the only three which show any physical significance. This is true not only for this test, but for all others as well. The patterns corresponding to the largest two eigenvalues seem to choose signals (jammers in this case) by pointing the antenna as opposed to performing nulling operations. The third eigenvalue, however, which separates the desired signal, does exhibit a nulling action directed at the jammers. This is logical since the jammers are much stronger than the friendly communicator. In order to provide a reasonable signal separation, the strong jammers must be eliminated. As will be seen, this trend is followed throughout all of the tests.

The first eight plots of the test show the unadapted and adapted patterns, in the direction of jammer 1, for the LMS, constrained LMS, and update covariance algorithms. Likewise, the next set of eight patterns show the same succession of unadapted and adapted plots, but in the direction of jammer 2. From these plots, it can easily be seen that the algorithms produce nulls in the antenna patterns in the directions corresponding to the jammer arrivals.

As an introduction to the plots created using the eigenvector algorithm, a total of five unadapted patterns in the direction of all incident signals are shown. A comparison of these plots with the unadapted plots for the other algorithms reveals that the initial patterns are not the same.

The remaining plots show the appearance of the adapted antenna patterns from the eigenvector sort algorithm. The first five of these (Figures T1.22 - T1.26) show the results of setting the weights equal to the eigenvector corresponding to the largest eigenvalue. An examination of these plots reveals that this weight set steers the antenna pattern in the direction of jammer 2, and gives this jammer an advantage of approximately 15 dB over jammer 1. The antenna pattern also attenuates the desired signal approximately 24 dB with respect to jammer number 2. As mentioned earlier, this weight set corresponding to the largest eigenvalue has no significance in terms of desired signal separation, but it does give an indication that the algorithm is attempting to separate the incoming signals regardless of their nature. It should also be pointed out here that although jammer 2 is definitely chosen by the antenna, "perfect" signal sorting is not achieved. That is, jammer 1, which statistically contains the same power as jammer 2, is attenuated by only 15 dB with respect to the chosen jammer, leading to a relatively large amount of interference. This phenomenon will be seen as a general rule in the remaining tests.

The next five plots (T1.27 - T1.31) are the azimuth and elevation patterns for the same signal directions, but in this case the weight set for the antenna is the eigenvector corresponding to the second largest eigenvalue. As can be seen here, the antenna pattern points more directly at jammer 1, and

presents an advantage over the other strong jammer of approximately 10 dB. Again we see that the jammers are not extremely well separated.

Finally, the remaining plots for test 1 are concerned with the third largest eigenvalue. Applying the corresponding eigenvector as weights produces an antenna pattern which effectively nulls the jamming signals and consequently selects the desired communicator. It is interesting that not until the third largest eigenvalue is there any evidence of a nulling operation. The jammers are attenuated by approximately 30-33 dB with respect to the desired signal. For a better signal to interference ratio, the nulls could be somewhat deeper. This could perhaps be obtained by further averaging of the autocorrelation matrix. The number of samples averaged in this test is just 32, which amounts to two bits of information. This does show, however, that the algorithm is attempting to neutralize the strong jamming signals. It will be seen that the algorithm is quite sensitive to changes in the number of averages and, in fact, this is the focus of test 3. Preceding all tests, the number of samples averaged will be explicitly stated.

Summary of Plots for Test 1

Fig. T1.1: Unadapted antenna plot at $\phi = 90°$

Fig. T1.2: Unadapted antenna plot at $\theta = 10°$

Fig. T1.3: LMS-adapted plot at $\phi = 90°$

Fig. T1.4: LMS-adapted plot at $\theta = 10°$

Fig. T1.5: Constrained LMS-adapted plot at $\phi = 90°$

Fig. T1.6: Constrained LMS-adapted plot at $\theta = 10°$

Fig. T1.7: Update Covariance-adapted plot at $\phi = 90°$

Fig. T1.8: Update Covariance-adapted plot at $\theta = 10°$

Fig. T1.9: Unadapted antenna plot at $\phi = 150°$

Fig. T1.10: Unadapted antenna plot at $\theta = 15°$

Fig. T1.11: LMS-adapted antenna plot at $\phi = 150°$

Fig. T1.12: LMS-adapted antenna plot at $\theta = 15°$

Fig. T1.13: Constrained LMS-adapted plot at $\phi = 150°$

Fig. T1.14: Constrained LMS-adapted plot at $\theta = 15°$

Fig. T1.15: Update Covariance-adapted plot at $\phi = 150°$

Fig. T1.16: Update Covariance-adapted plot at $\theta = 15°$

Eigenvector Algorithm

Fig. T1.17: Unadapted antenna plot at $\phi = 90°$

Fig. T1.18: Unadapted antenna plot at $\theta = 10°$

Fig. T1.19: Unadapted antenna plot at $\theta = 30°$

Fig. T1.20: Unadapted antenna plot at $\phi = 150°$

Fig. T1.21: Unadapted antenna plot at $\theta = 15°$

(Largest Eigenvalue)

Fig. T1.22: Adapted plot at $\phi = 90°$

Fig. T1.23: Adapted plot at $\theta = 10°$

Fig. T1.24: Adapted plot at $\theta = 30°$

Fig. T1.25: Adapted plot at $\phi = 150°$

Fig. T1.26: Adapted plot at $\theta = 15°$

(Second Largest Eigenvalue)

Fig. T1.27: Adapted plot at $\phi = 90°$

Fig. T1.28: Adapted plot at $\theta = 10°$

Fig. T1.29: Adapted plot at $\theta = 30°$

Fig. T1.30: Adapted plot at $\phi = 150°$

Fig. T1.31: Adapted plot at $\theta = 15°$

(Third Largest Eigenvalue)

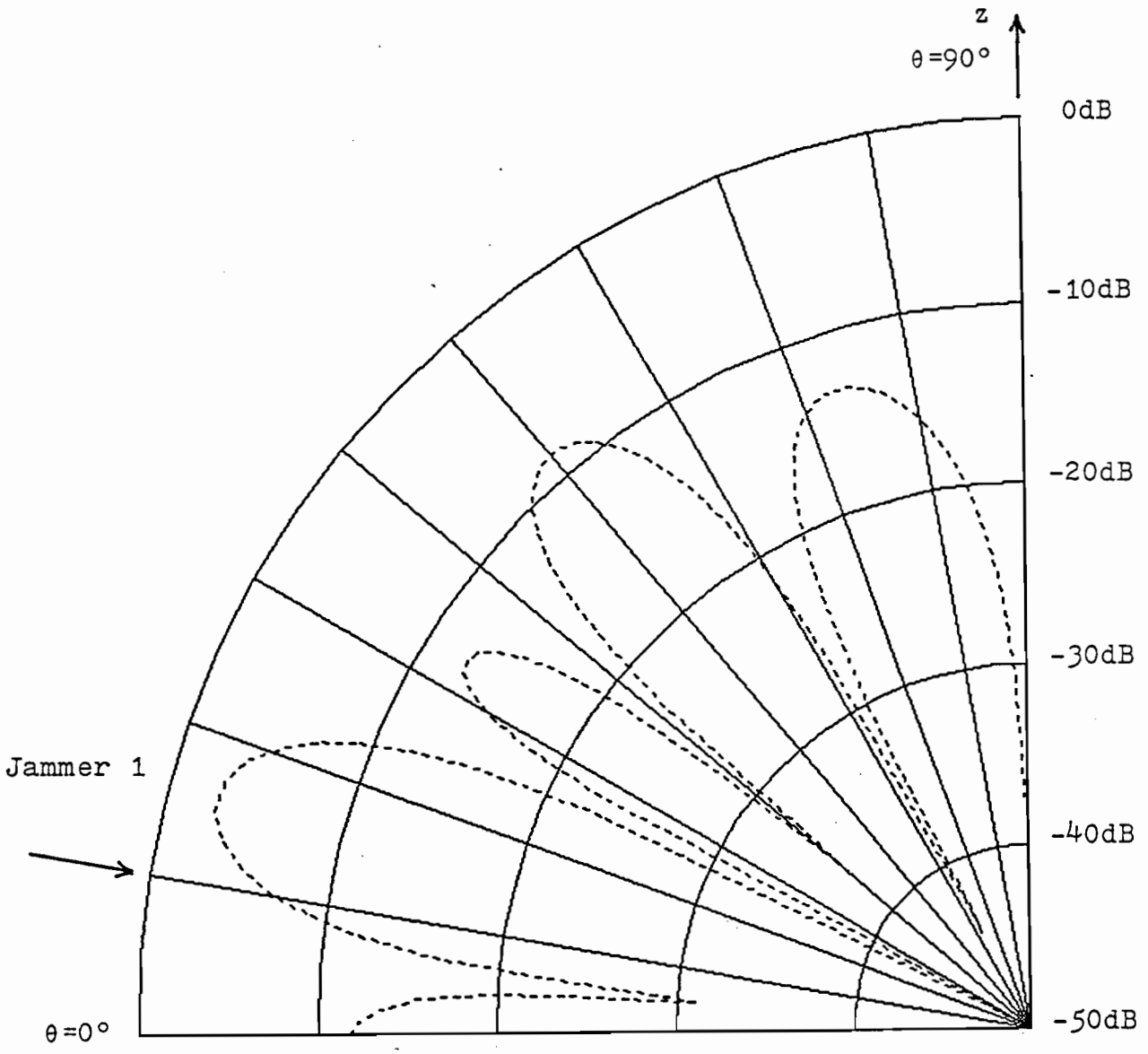
Fig. T1.32: Adapted plot at $\phi = 90°$

Fig. T1.33: Adapted plot at $\theta = 10°$

Fig. T1.34: Adapted plot at $\theta = 30°$

Fig. T1.35: Adapted plot at $\phi = 150°$

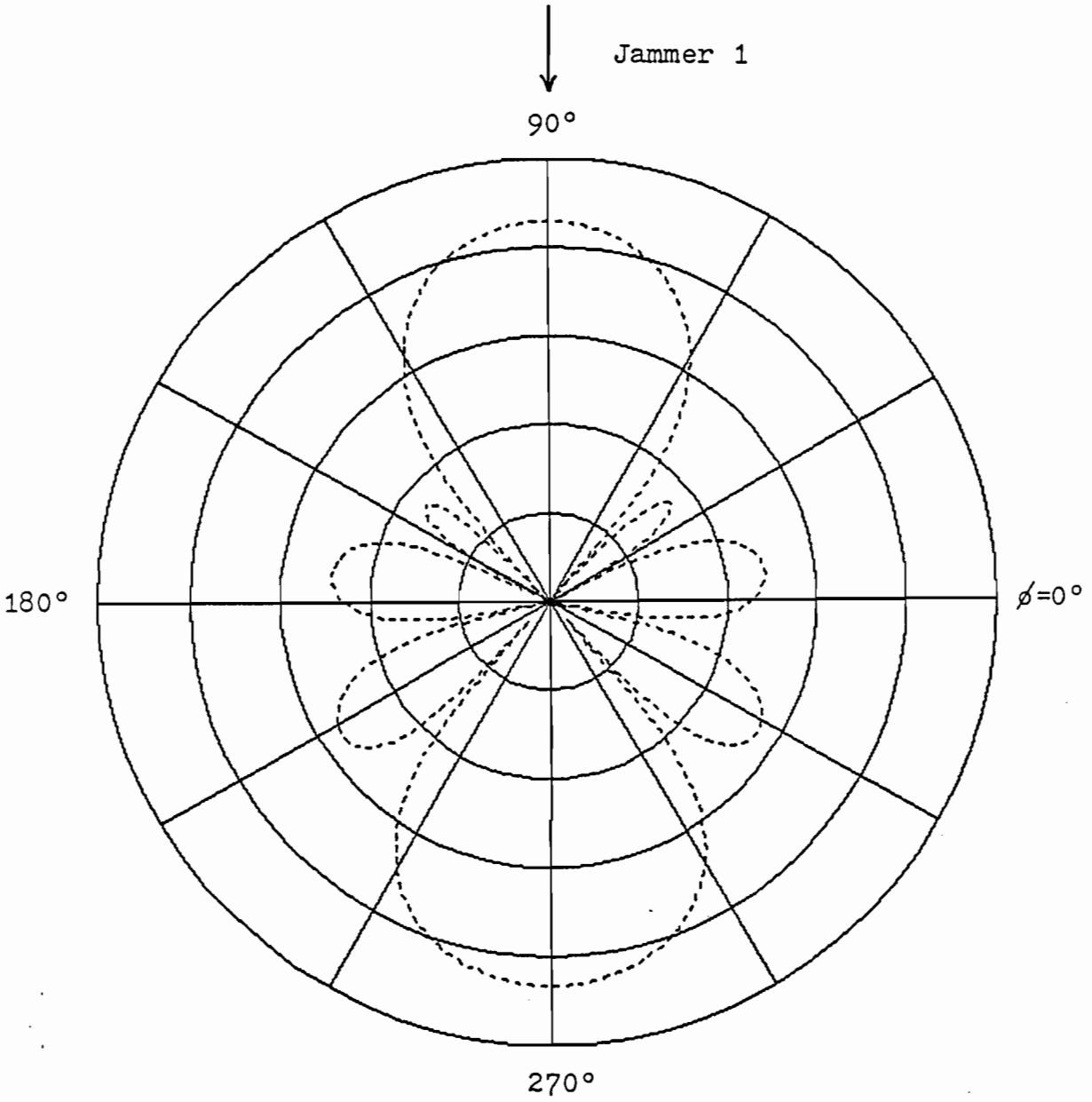
Fig. T1.36: Adapted plot at $\theta = 15°$



Elevation plot at 90° azimuth

Arrow indicates arrival angle of Jammer 1 (θ=10°)

Figure T1.1 Unadapted Antenna Pattern in Direction of Jammer 1



Azimuthal plot at 10° elevation

Arrow indicates arrival angle of Jammer 1 (ø=90°)

Figure T1.2 Unadapted Antenna Pattern in Direction of Jammer 1

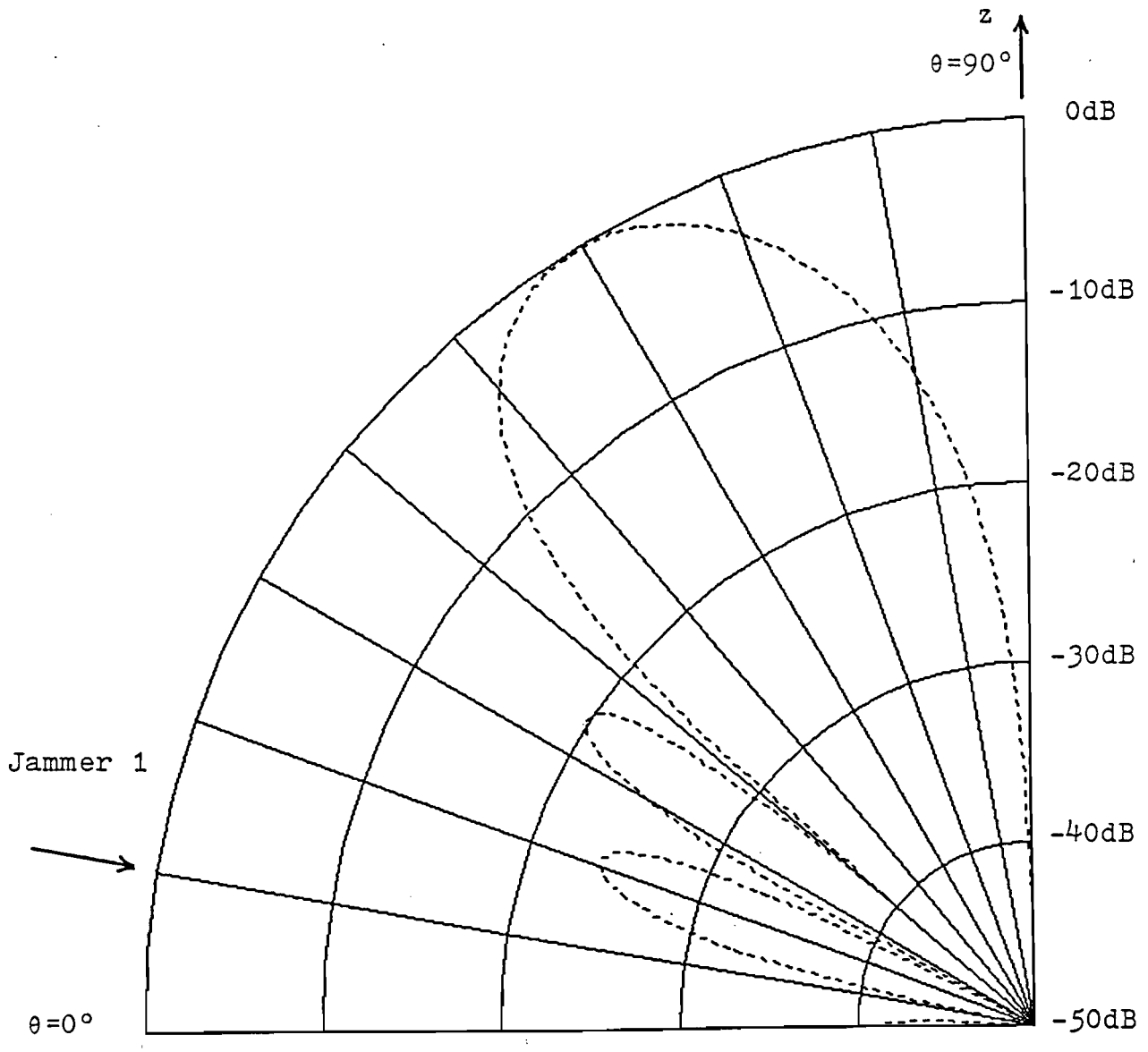


Figure T1.3 LMS-Adapted Pattern in Direction of Jammer 1

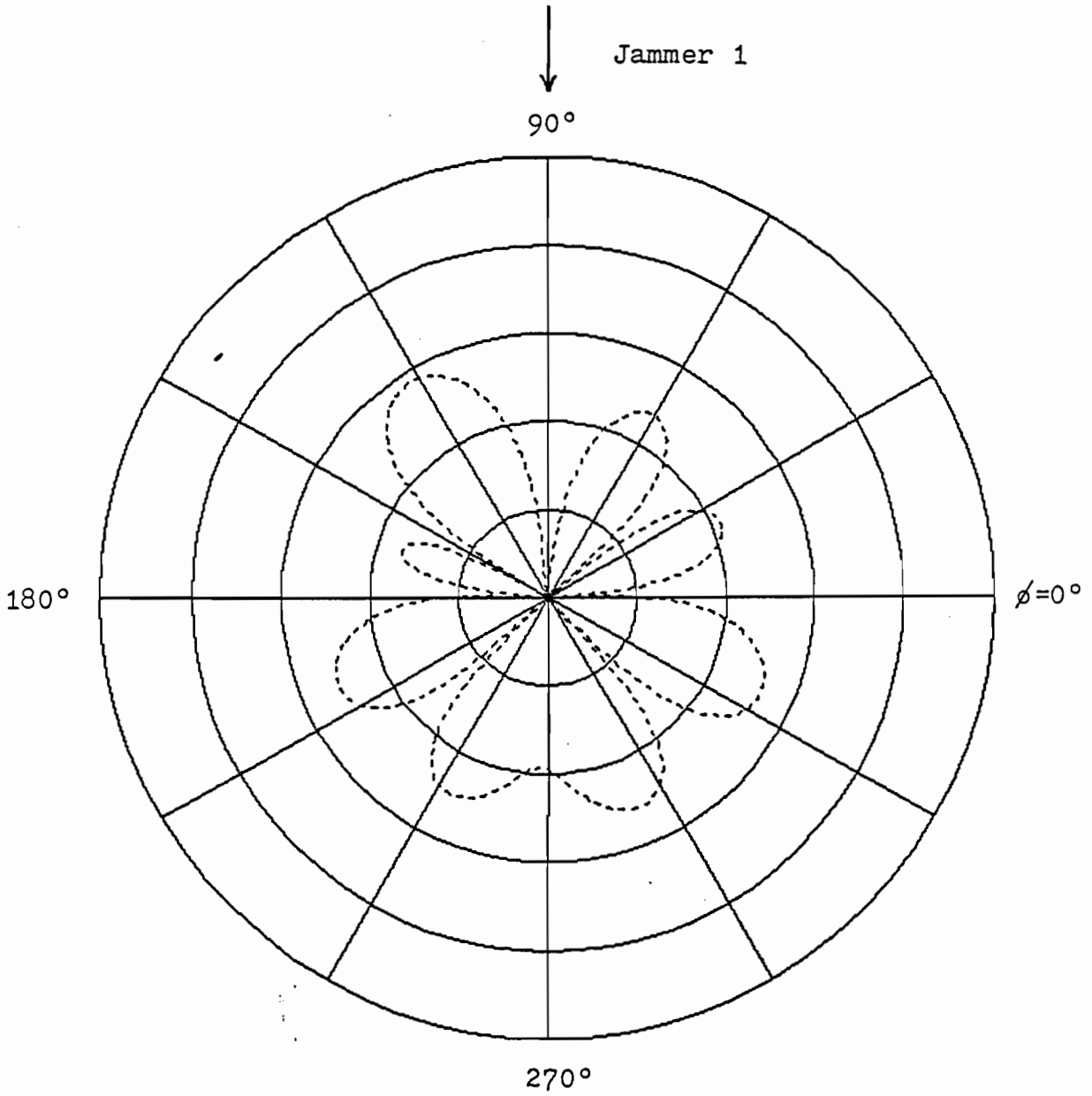


Figure T1.4 LMS-Adapted Pattern in Direction of Jammer 1

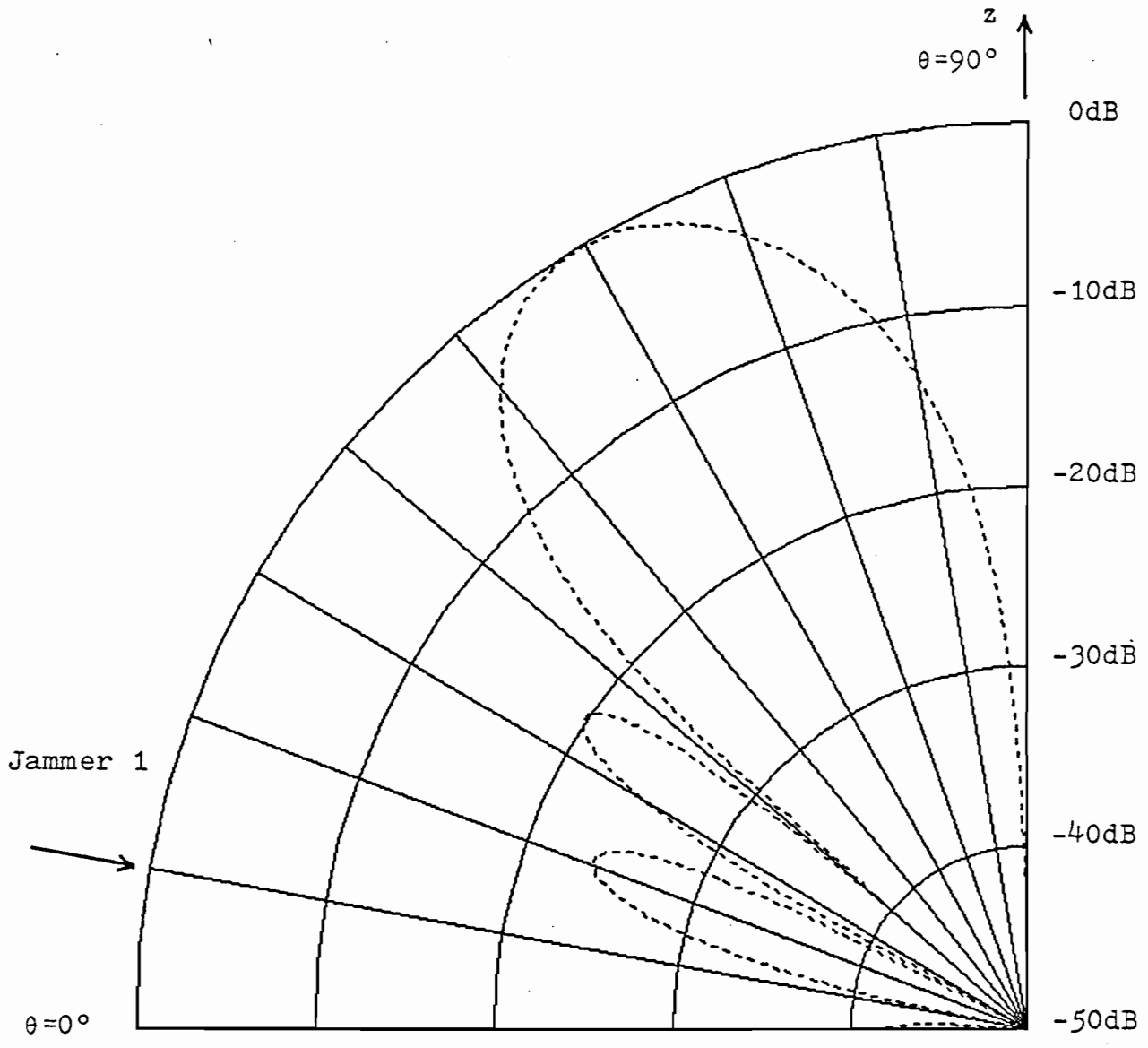


Figure T1.5 Constrained LMS-Adapted Pattern in Direction of Jammer 1

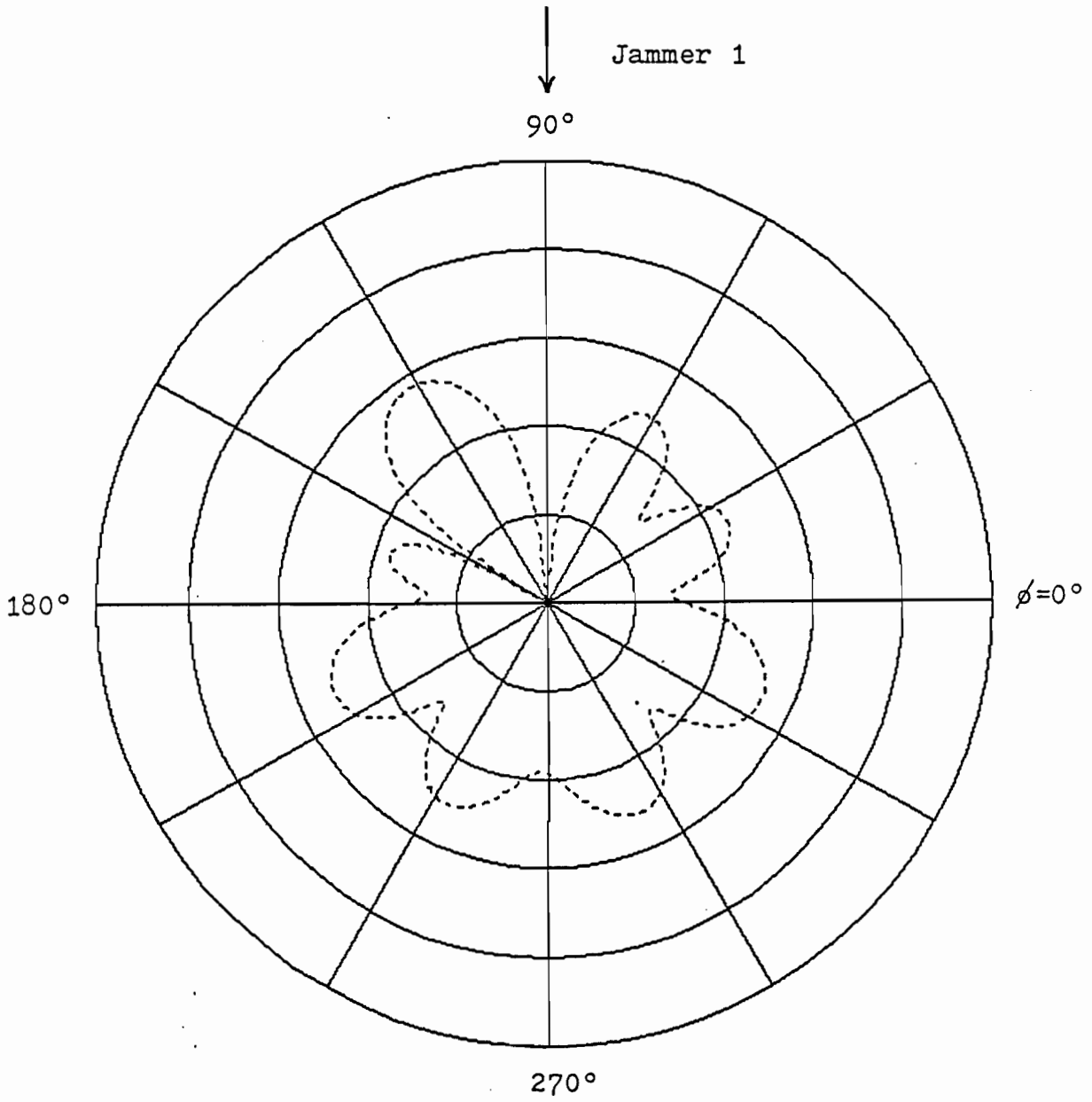


Figure T1.6 Constrained LMS-Adapted Pattern in direction of Jammer 1

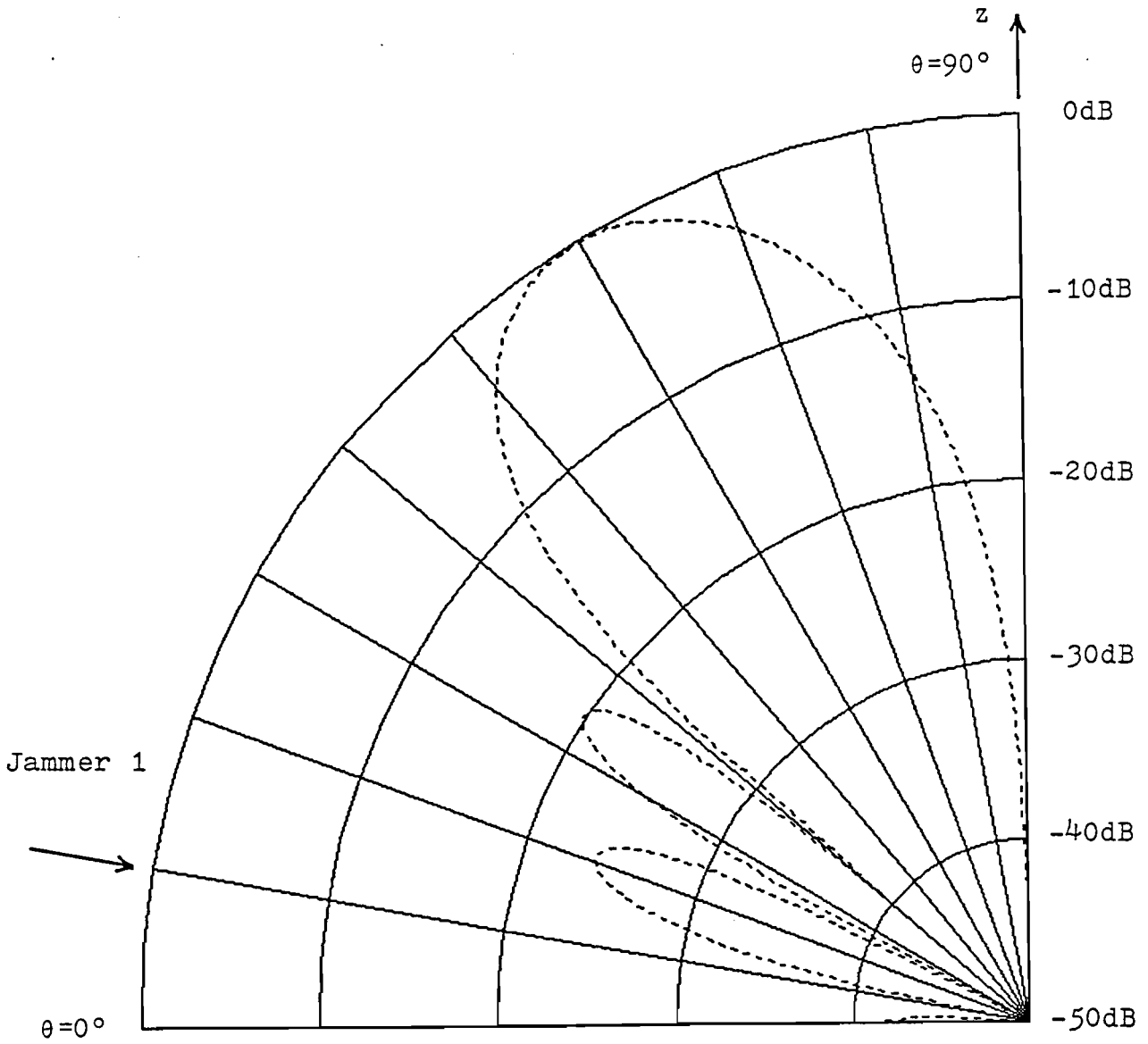


Figure T1.7 Update Covariance-Adapted Pattern in Direction of Jammer 1

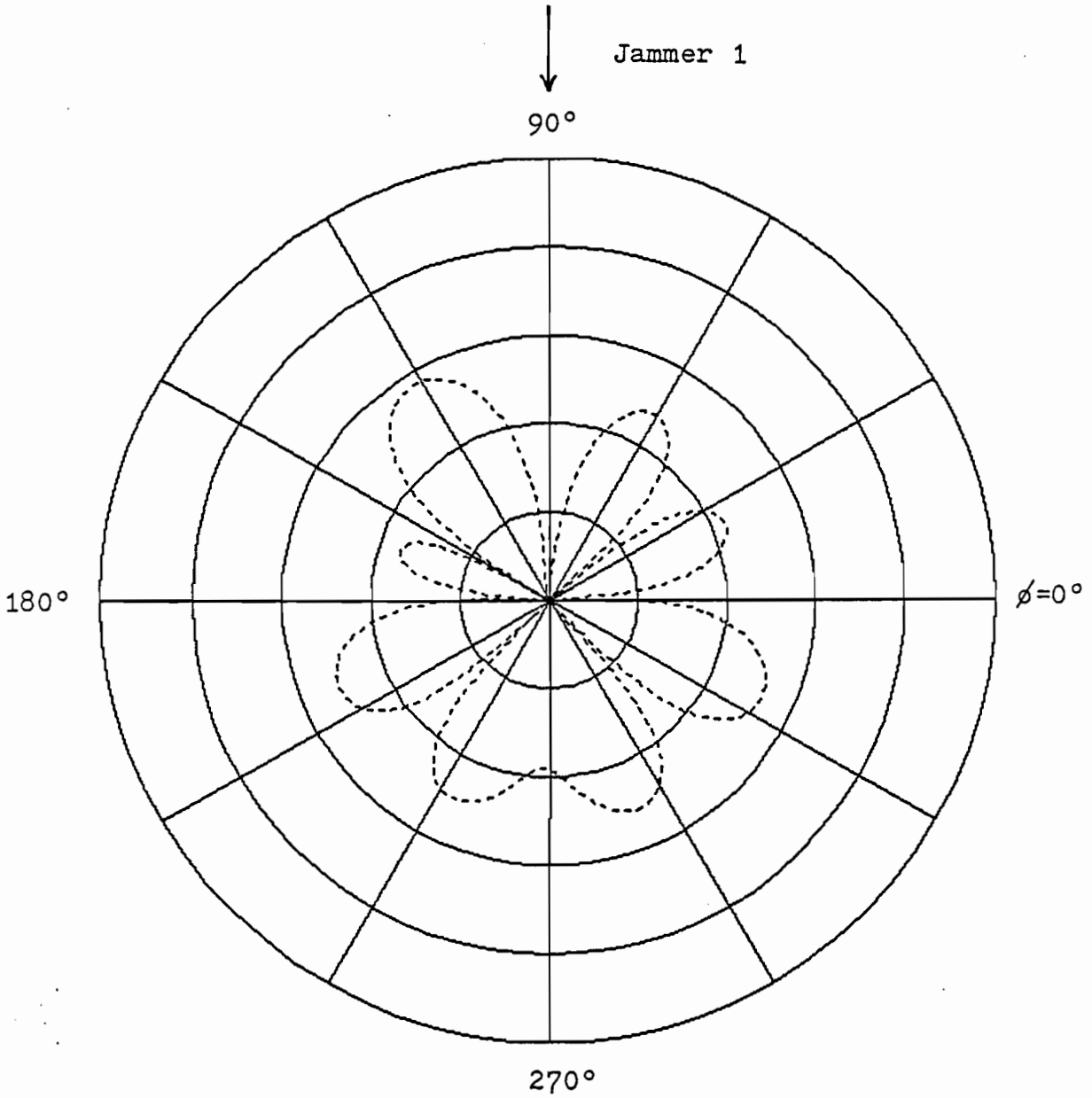
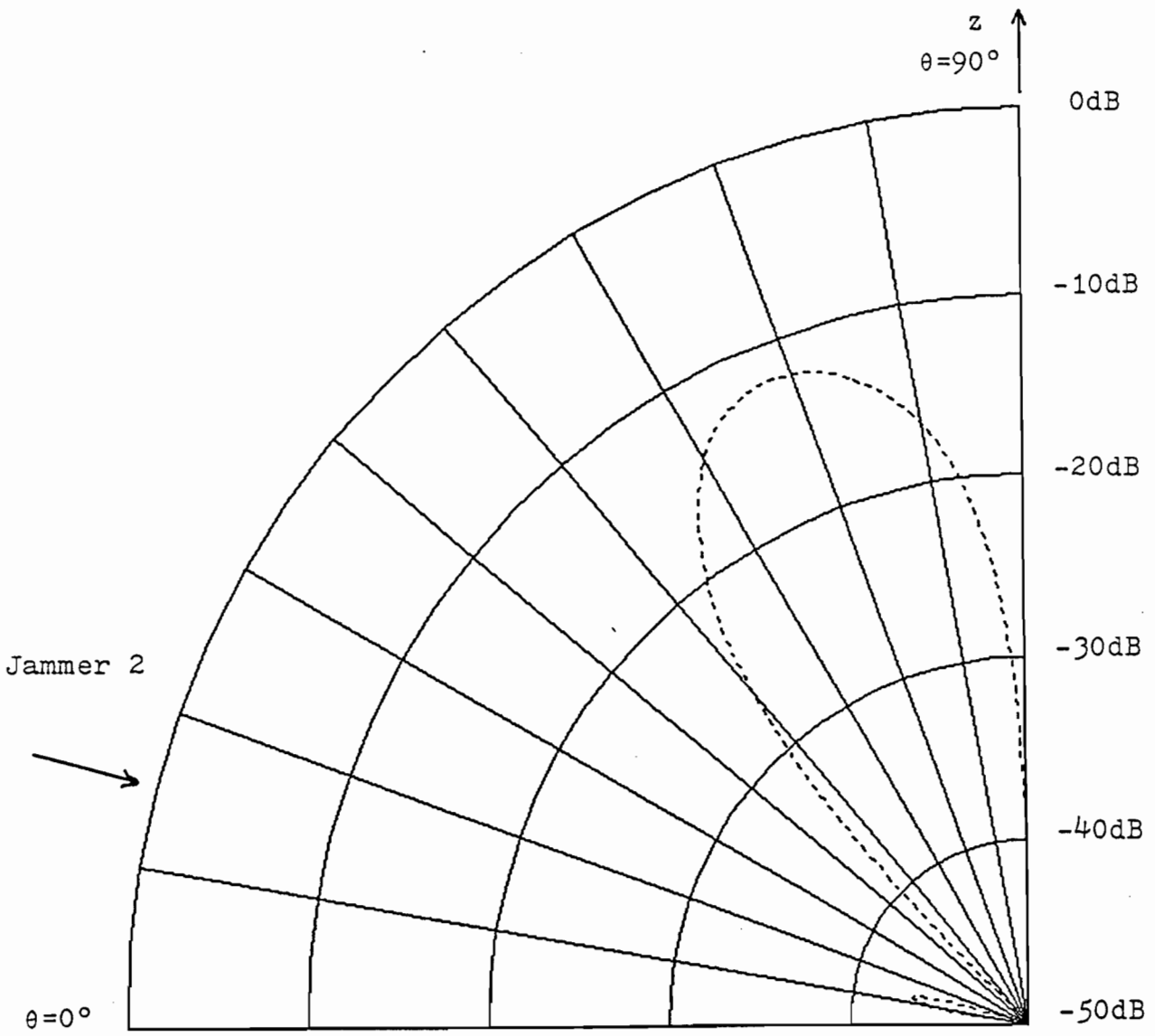


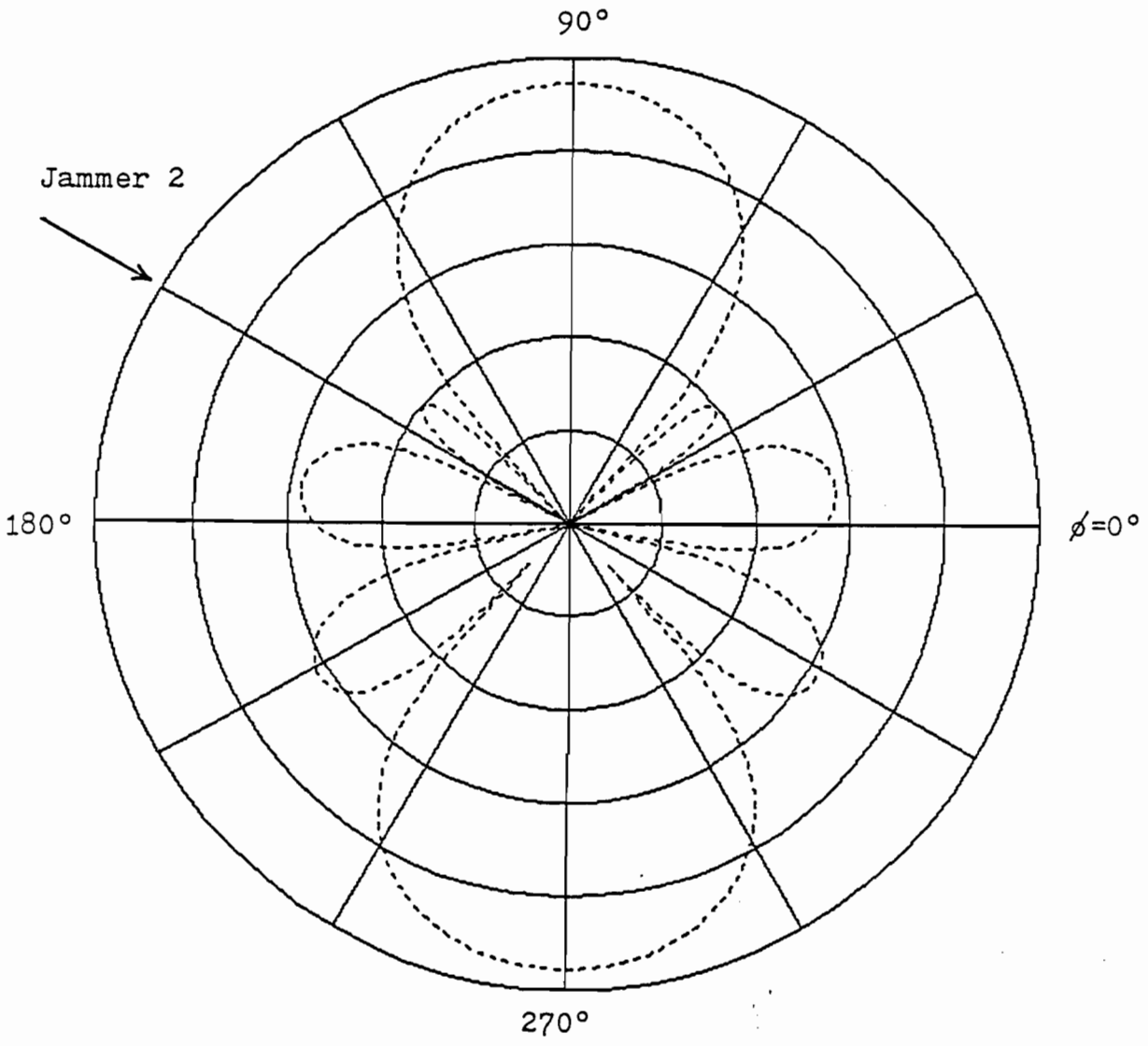
Figure T1.8 Update Covariance-Adapted Pattern in Direction of Jammer 1



Elevation plot at 150° azimuth

Arrow indicates arrival angle of Jammer 2 (θ=15°)

Figure T1.9 Unadapted Antenna Pattern in Direction of Jammer 2



Azimuthal plot at 15° elevation

Arrow indicates arrival angle of Jammer 2 (ø=150°)

Figure T1.10 Unadapted Antenna Pattern in Direction of Jammer 2

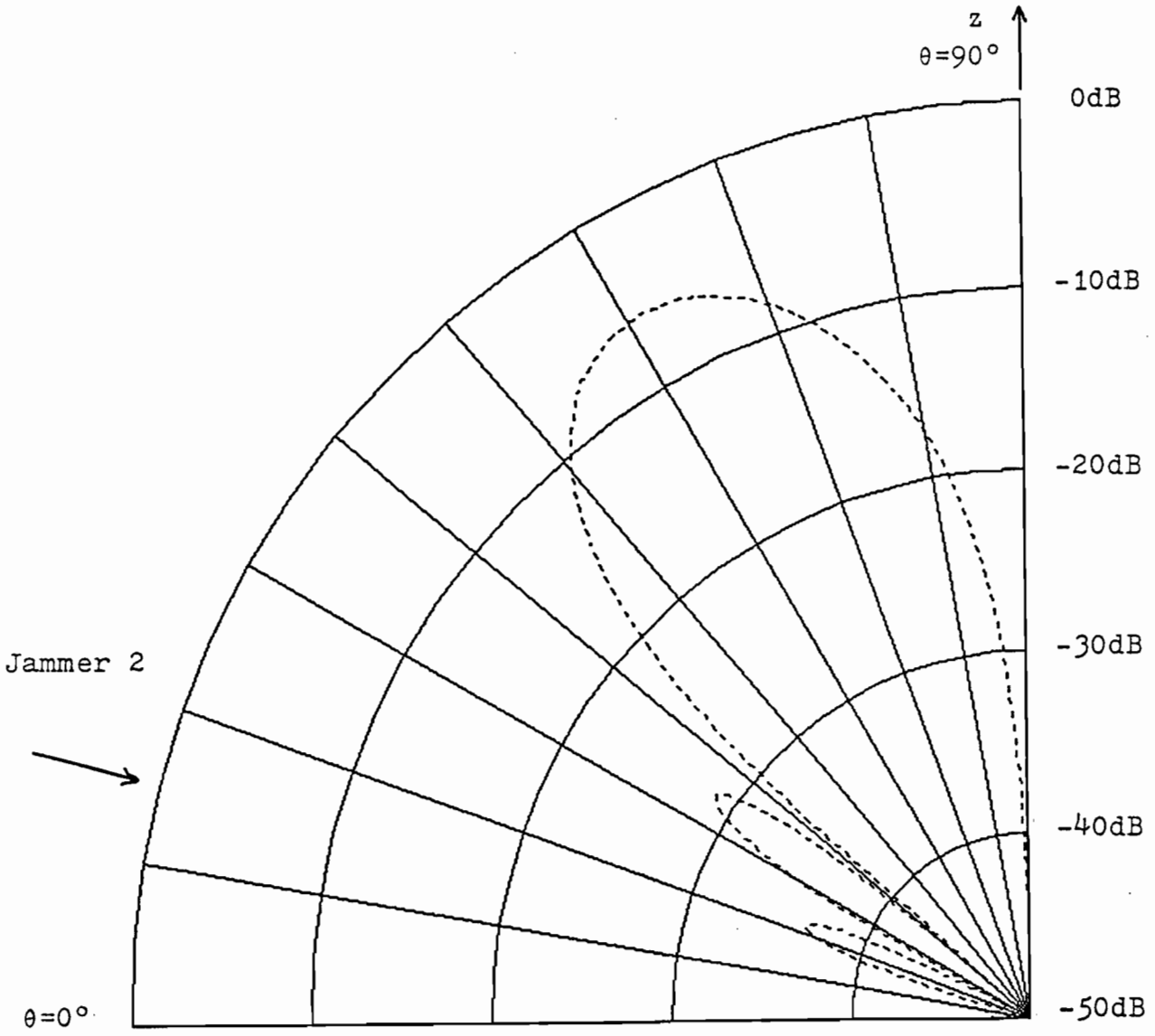


Figure T1.11 LMS-Adapted Pattern in Direction of Jammer 2

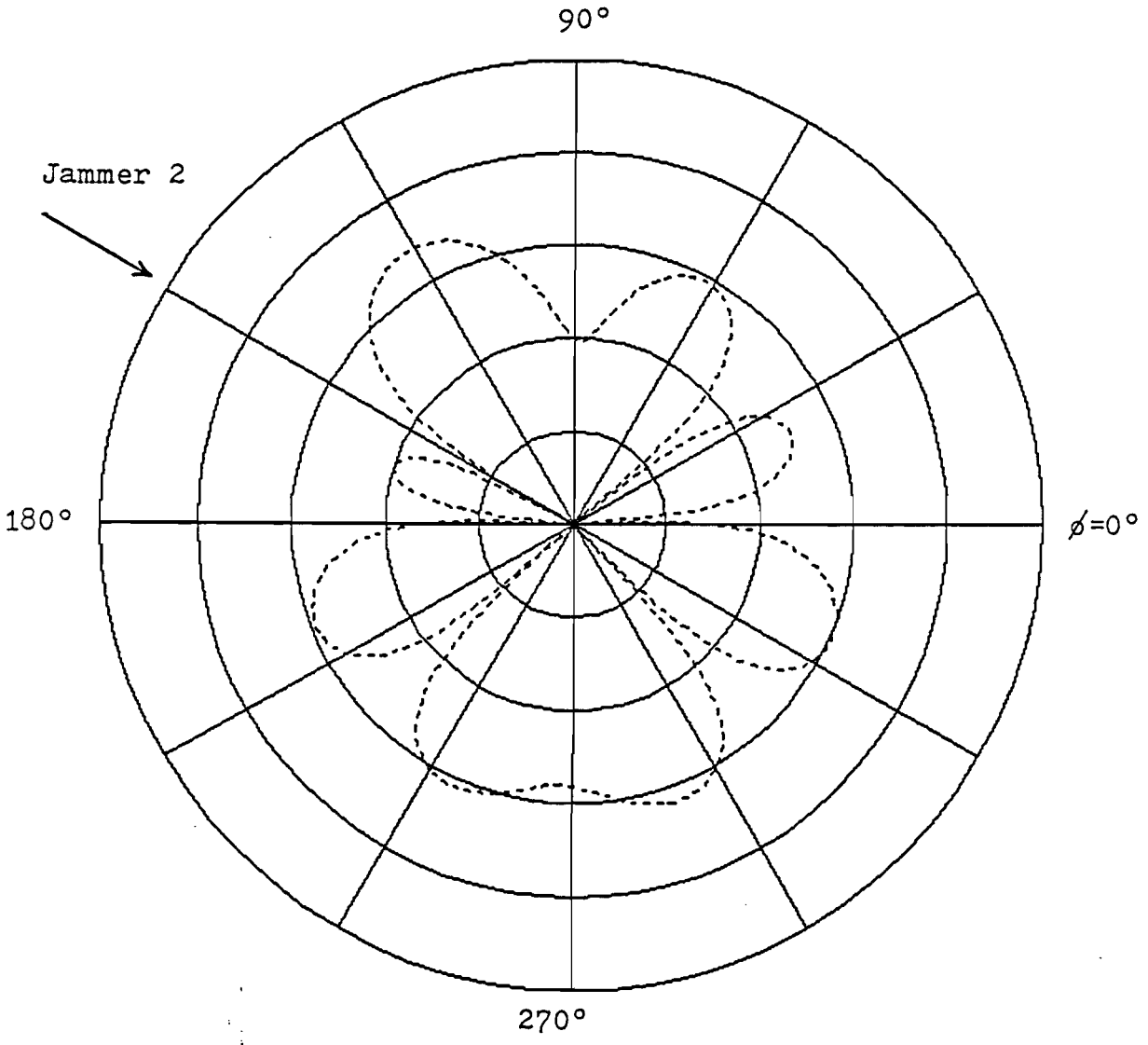


Figure T1.12 LMS-Adapted Pattern in Direction of Jammer 2

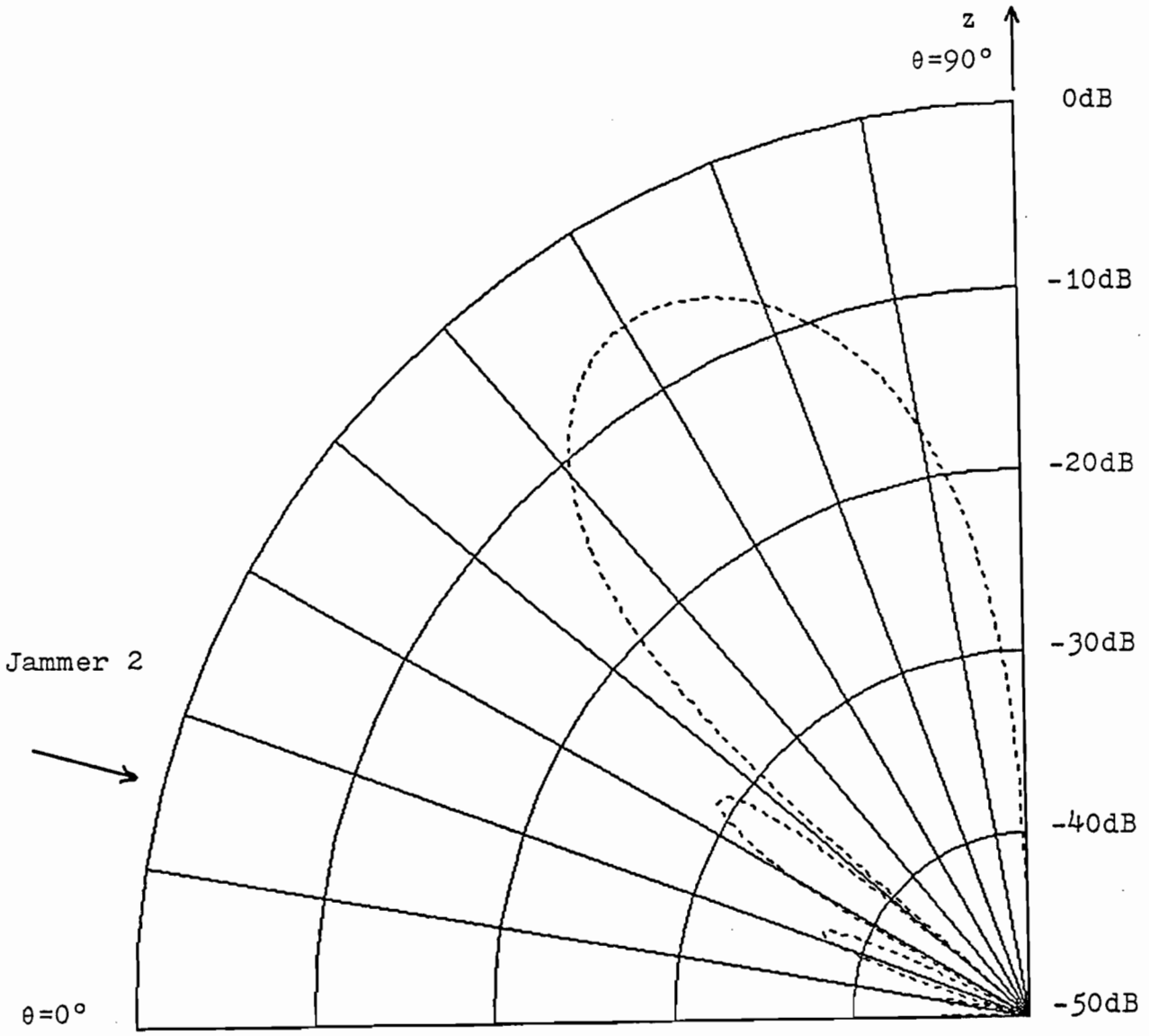


Figure T1.13 Constrained LMS-Adapted Pattern in Direction of Jammer 2

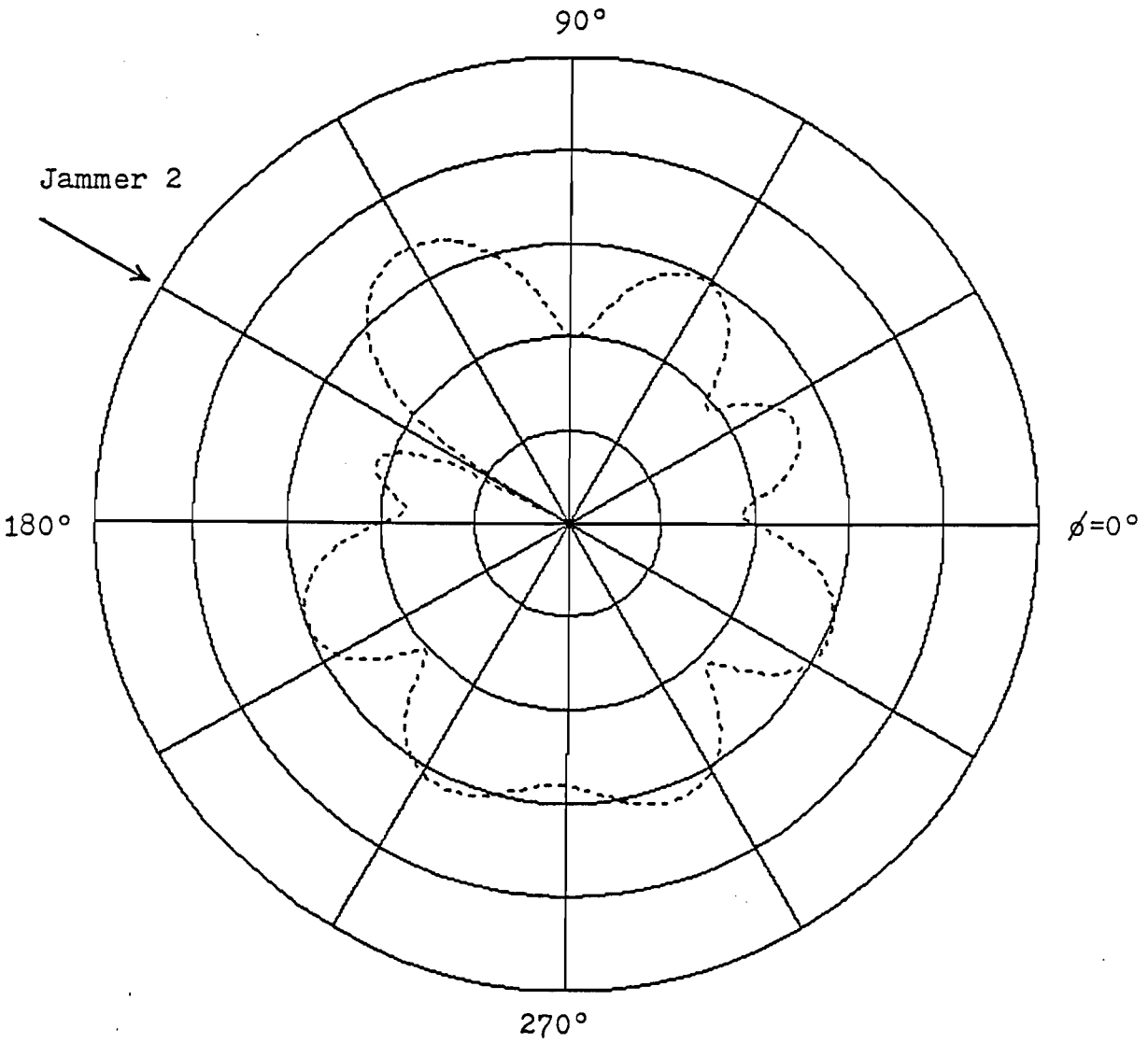


Figure T1.14 Constrained LMS-Adapted Pattern in Direction of Jammer 2

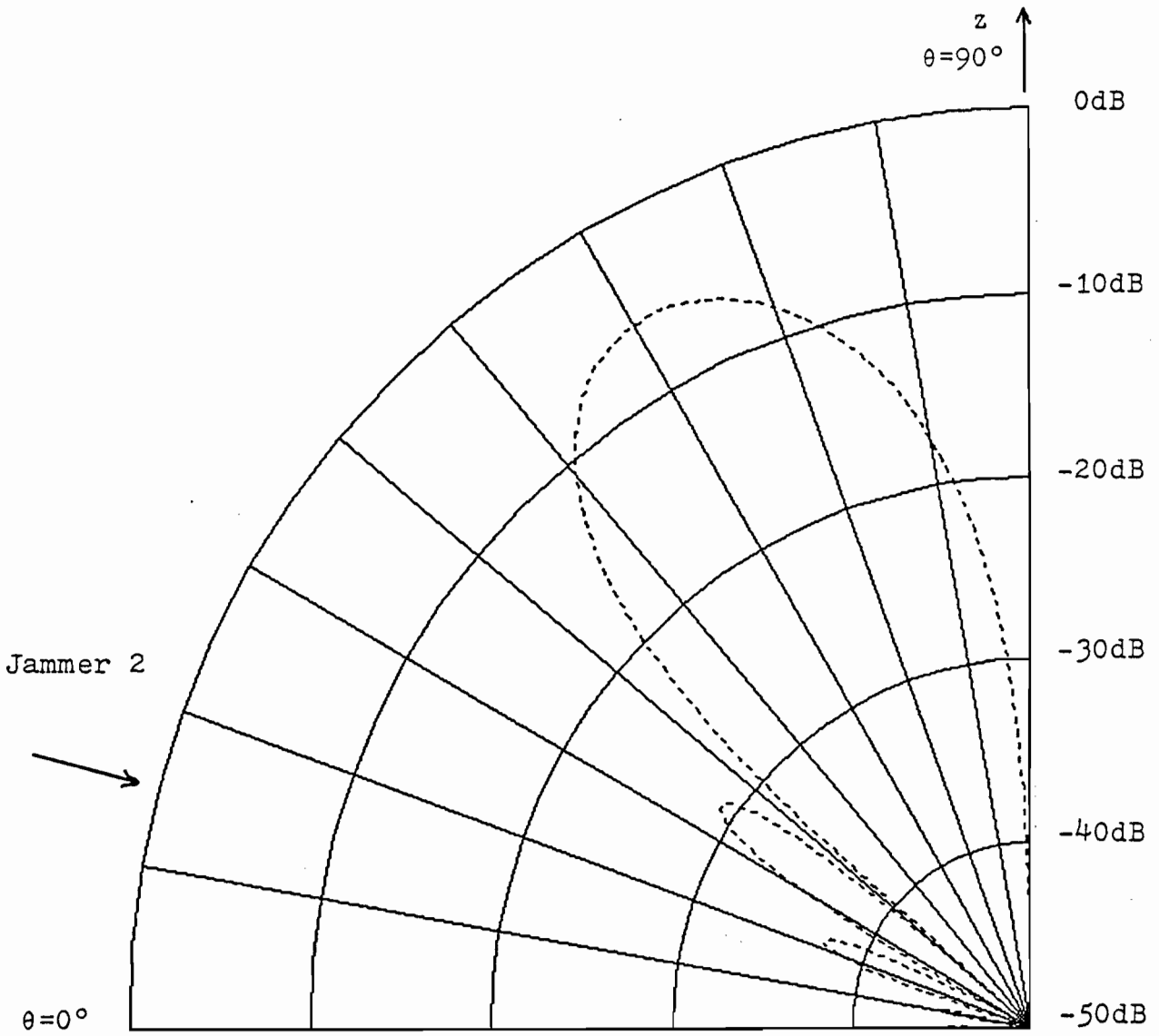


Figure T1.15 Update Covariance-Adapted Pattern
in Direction of Jammer 2

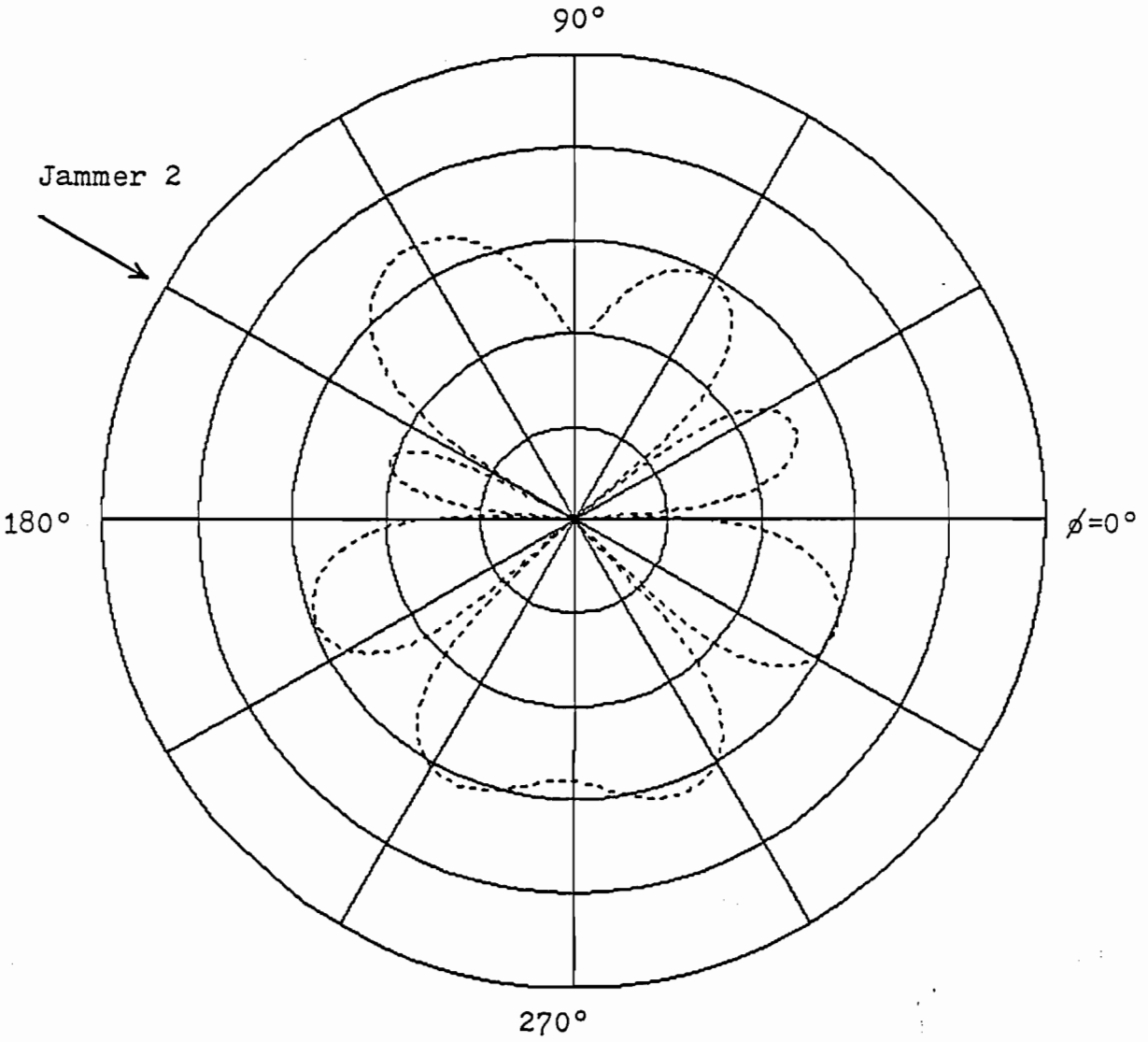


Figure T1.16 Update Covariance-Adapted Pattern
in Direction of Jammer 2

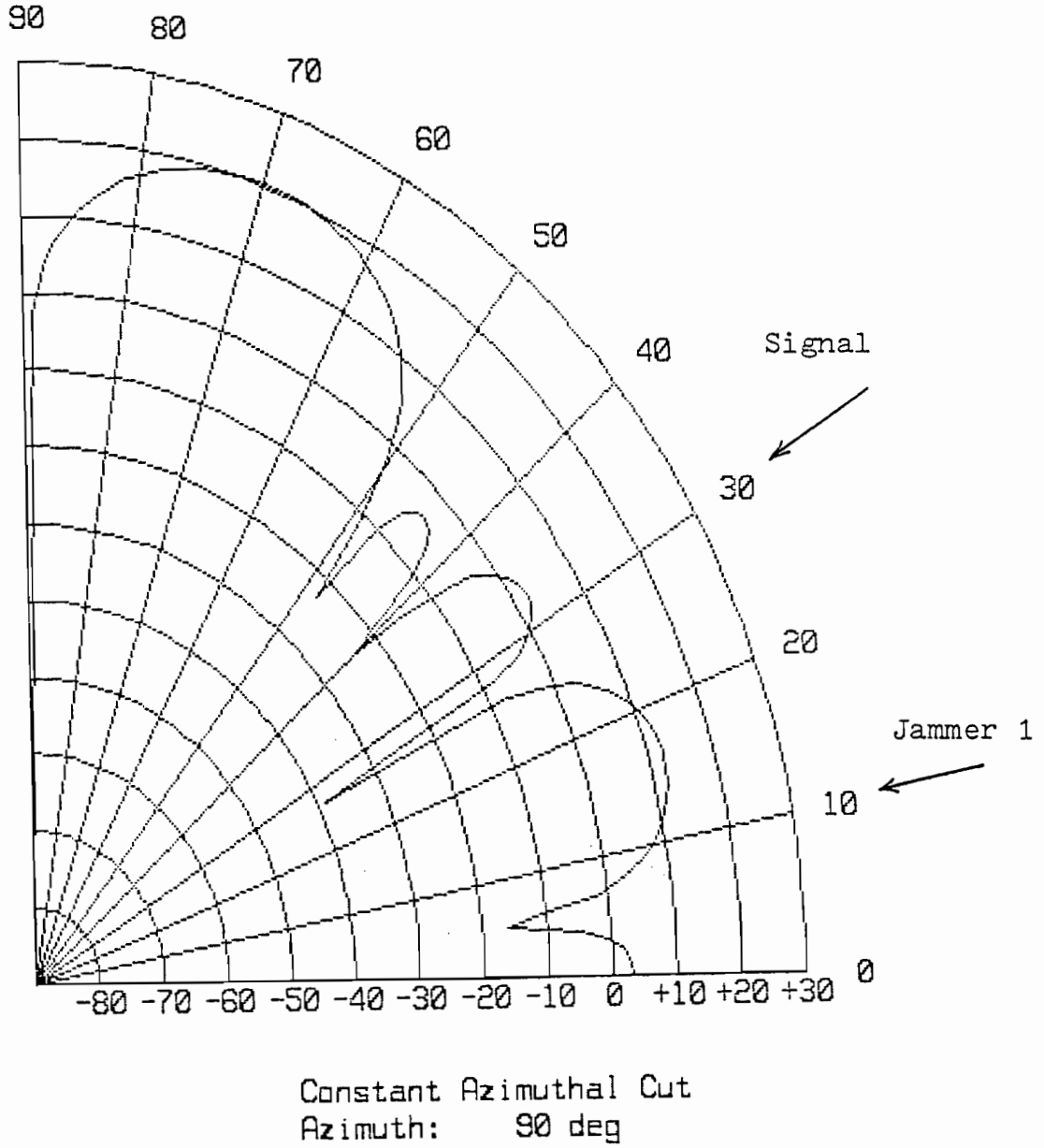


Figure T1.17 Unadapted Antenna Plot for Eigenvector Algorithm at $\phi = 90°$

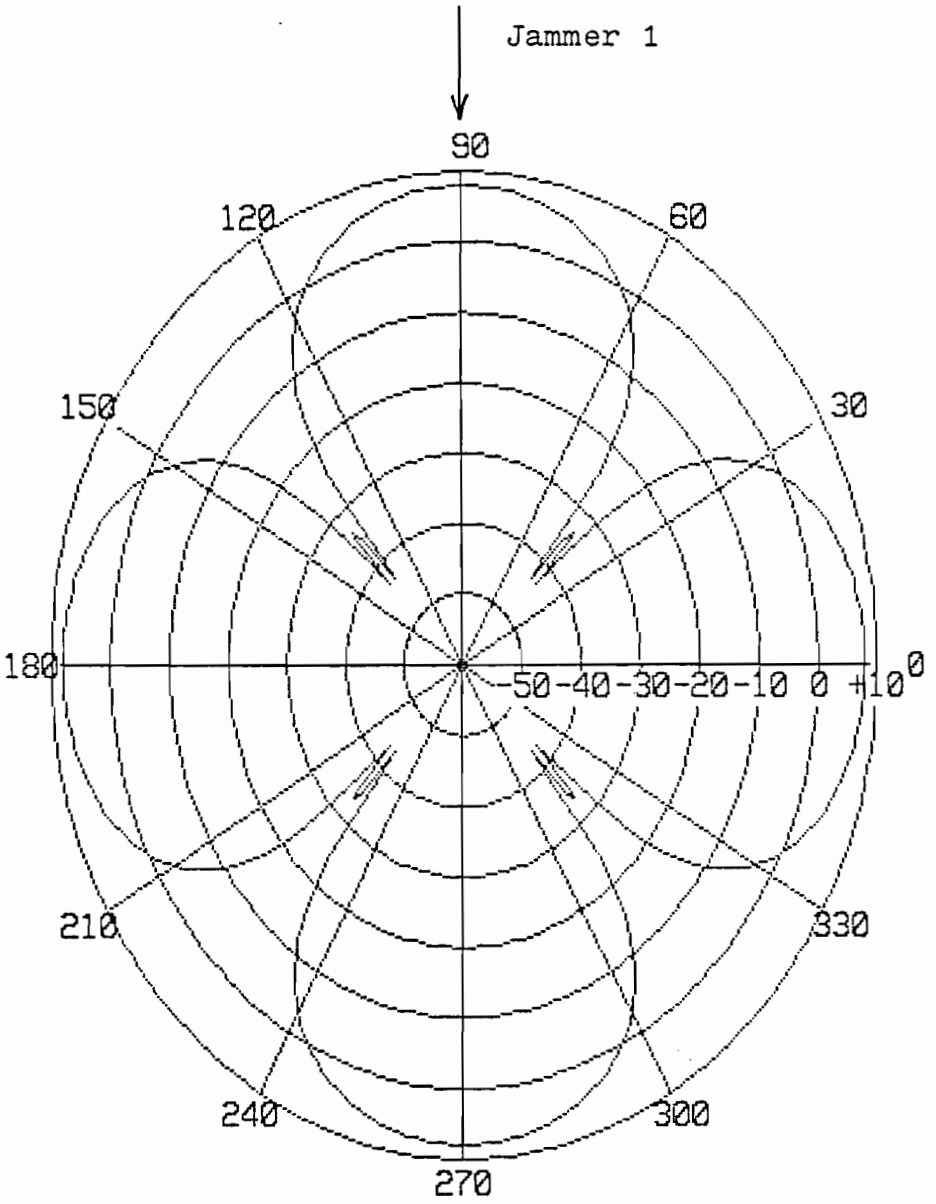


Figure T1.18 Unadapted Antenna Plot for Eigenvector Algorithm at $\theta = 10°$

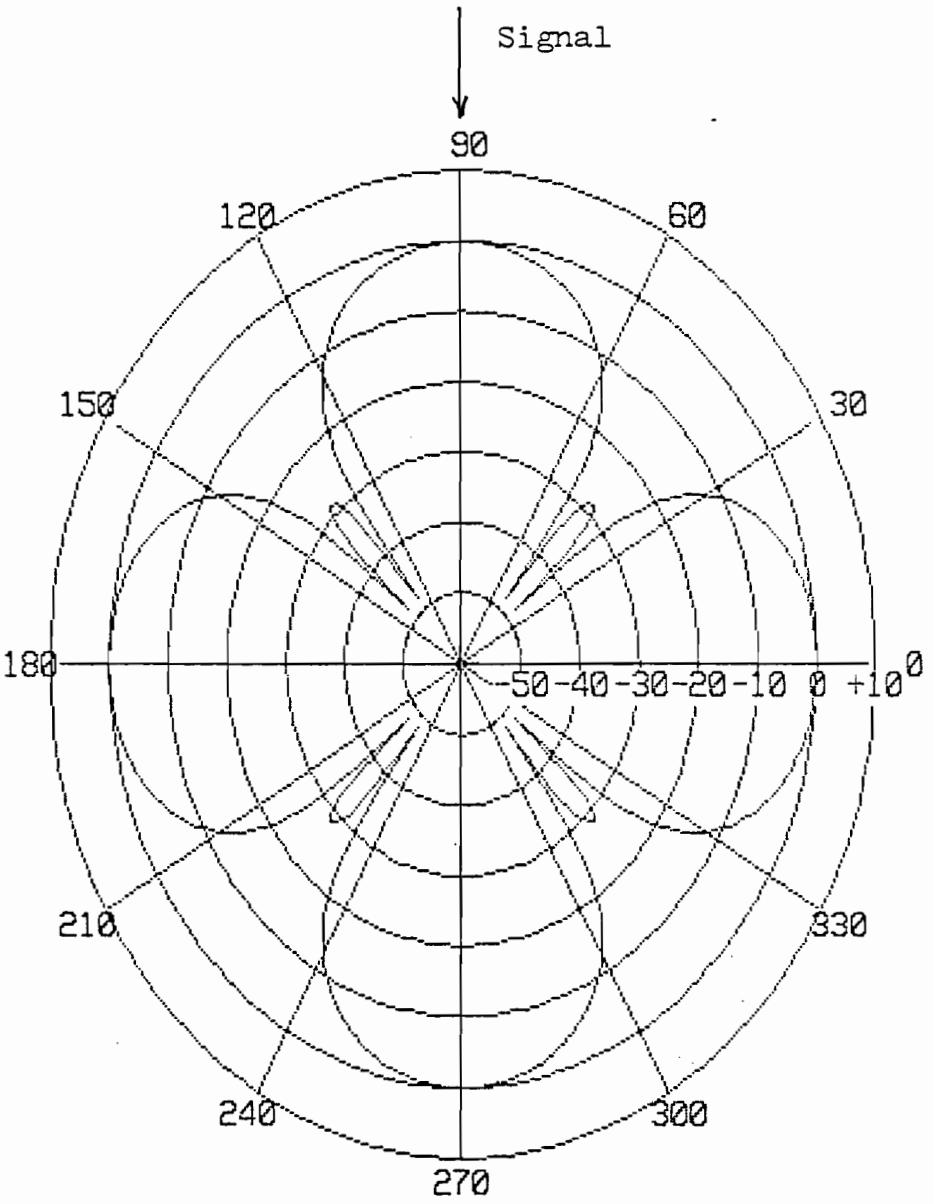


Figure T1.19 Unadapted Antenna Plot for Eigenvector Algorithm at $\theta = 30°$

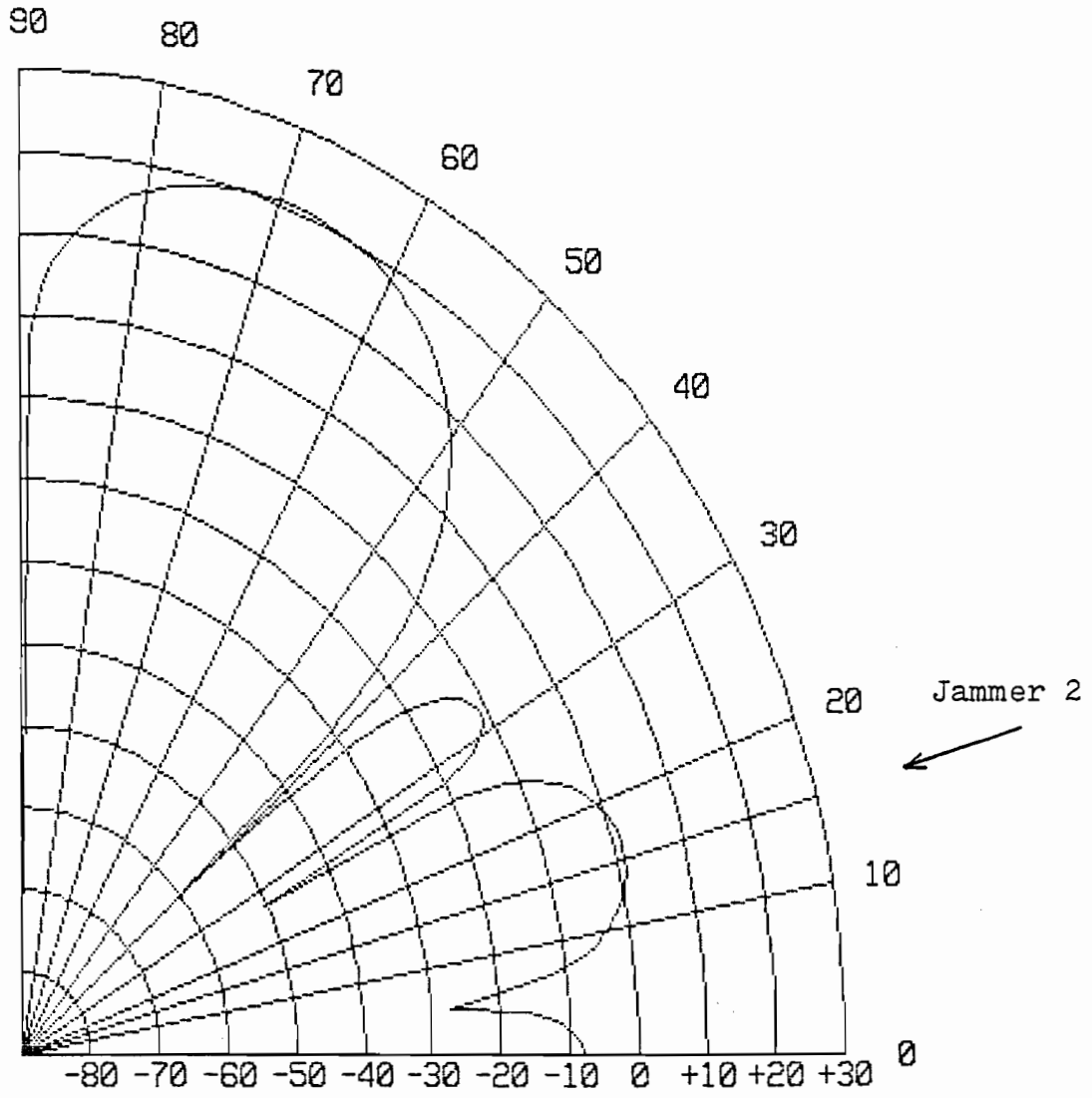


Figure T1.20 Unadapted Antenna Plot for Eigenvector Algorithm at ϕ = 150°

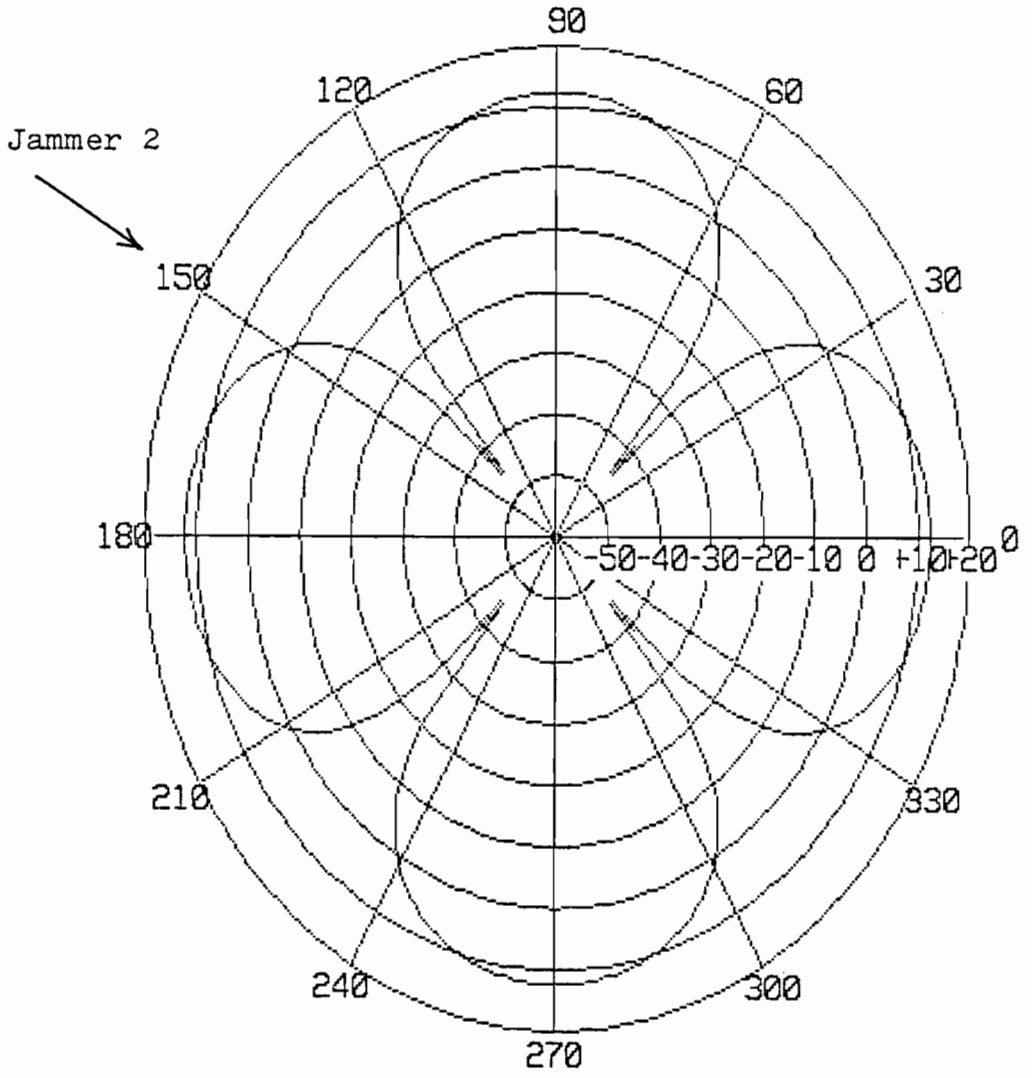
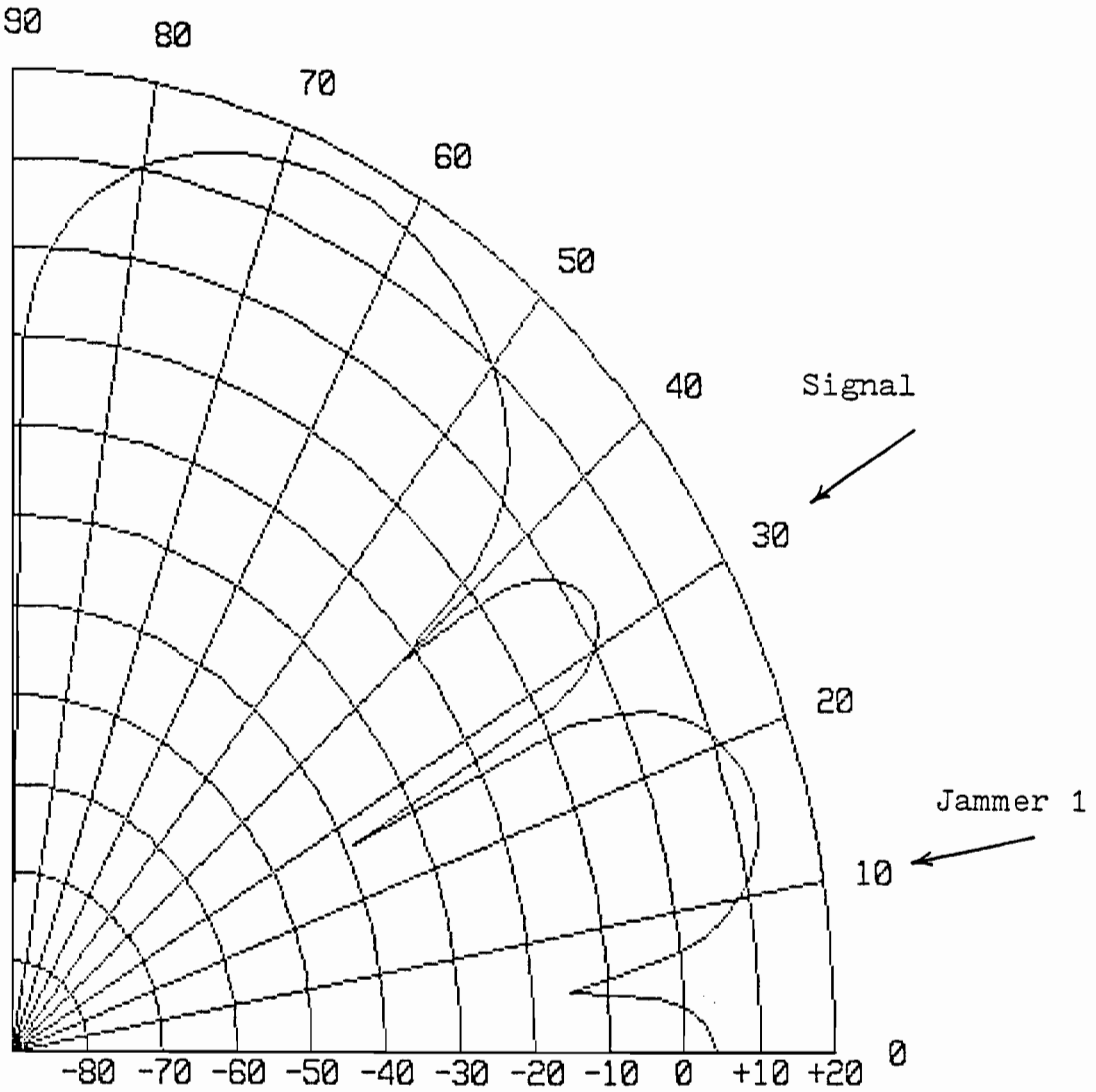


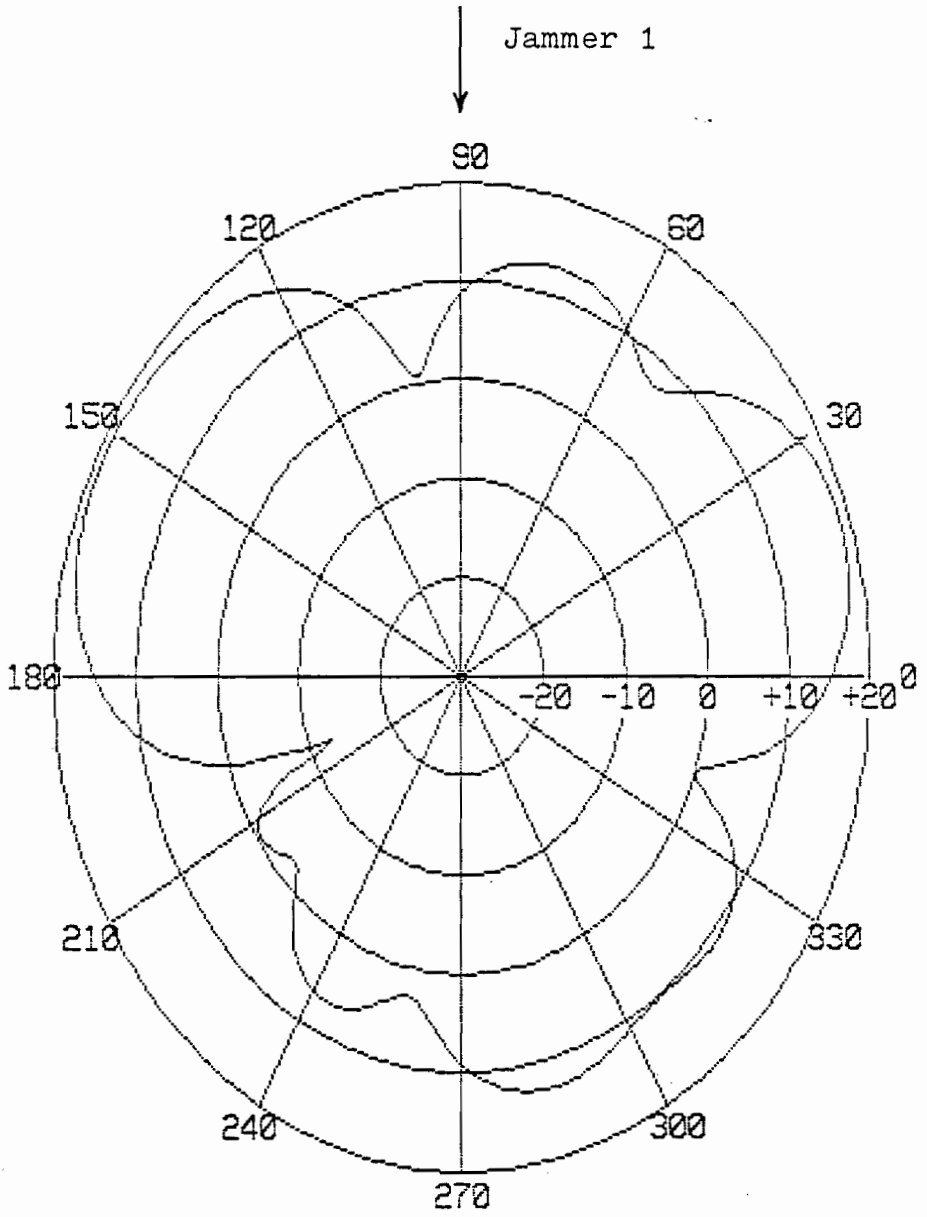
Figure T1.21 Unadapted Antenna Plot for Eigenvector Algorithm at $\theta = 15°$



Constant Azimuthal Cut
Azimuth: 90

(Largest Eigenvalue)

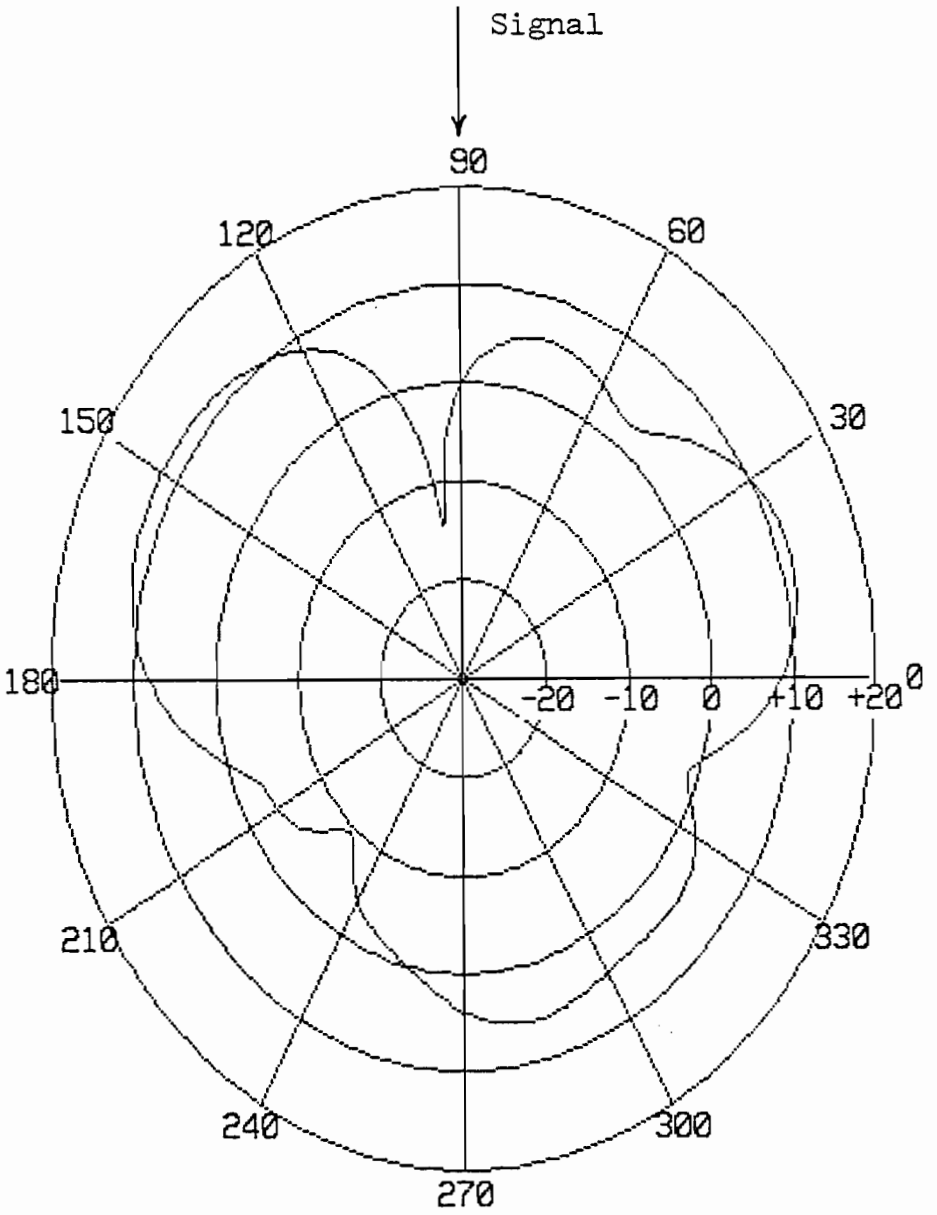
Figure T1.22 Eigenvector Sort-Adapted Antenna Plot at $\phi = 90°$



Constant Elevation Cut
Elevation: 10

(Largest Eigenvalue)

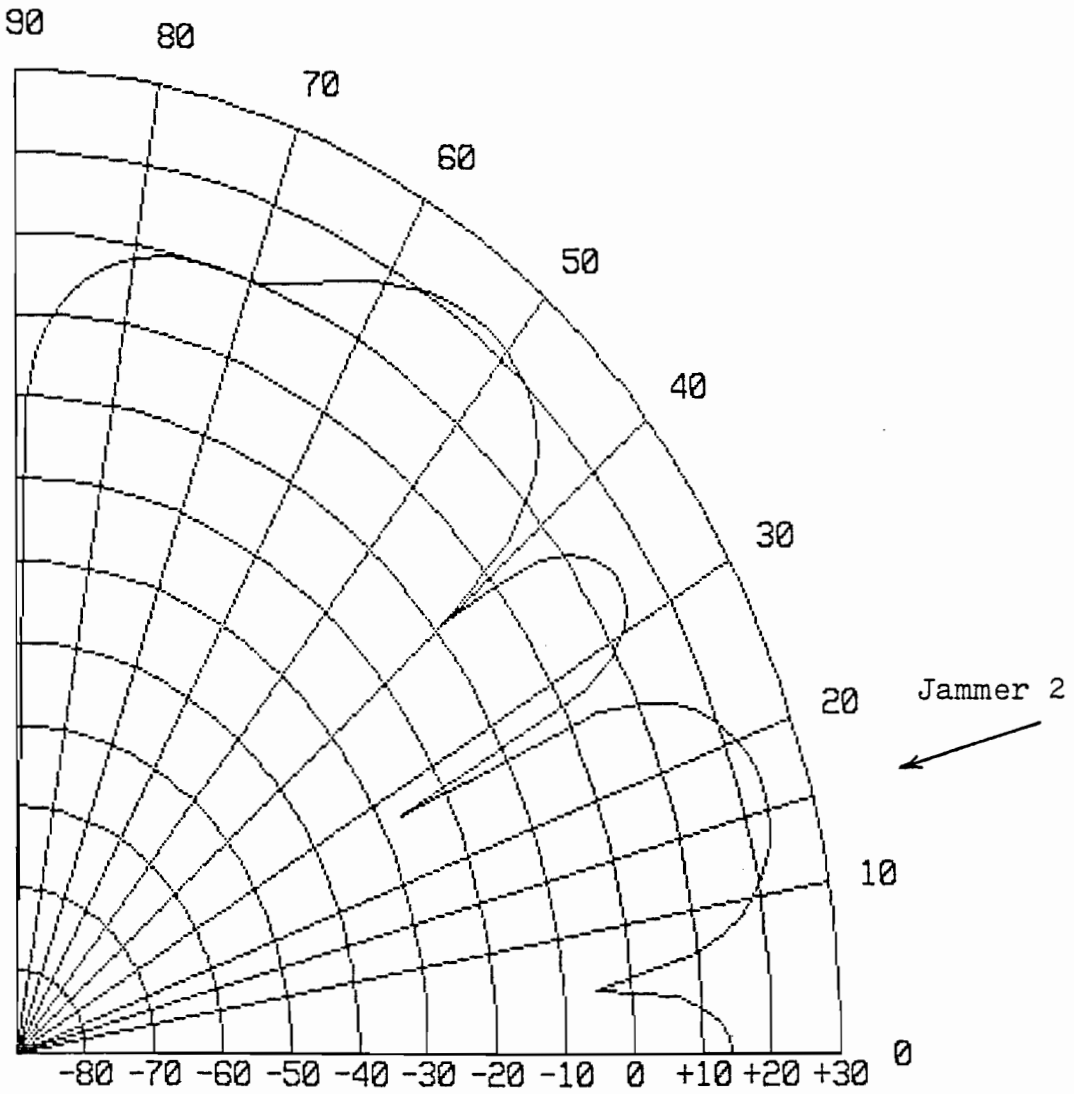
Figure T1.23 Eigenvector Sort-Adapted Antenna Plot at $\theta = 10°$



Constant Elevation Cut
Elevation: 30

(Largest Eigenvalue)

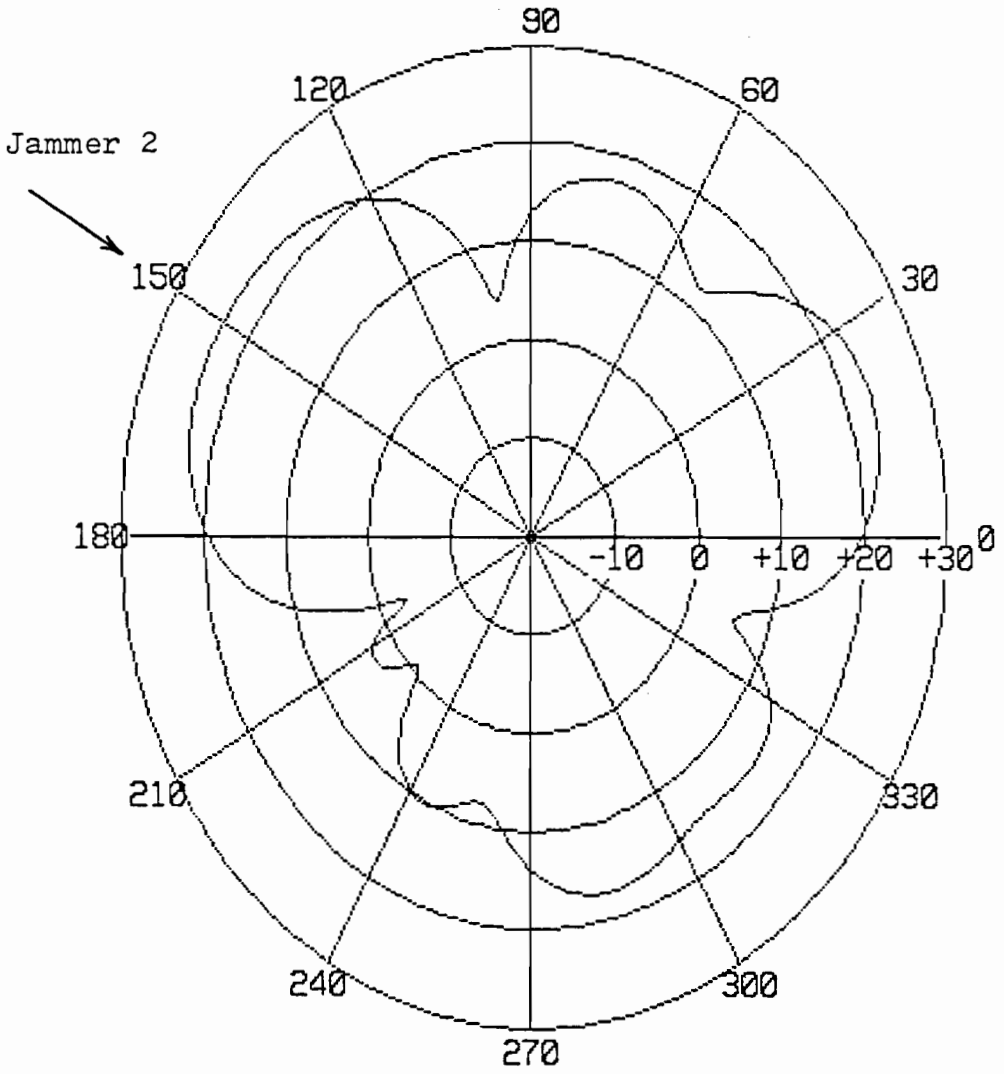
Figure T1.24 Eigenvector Sort-Adapted Antenna Plot at $\theta = 30°$



Constant Azimuthal Cut
Azimuth: 150

(Largest Eigenvalue)

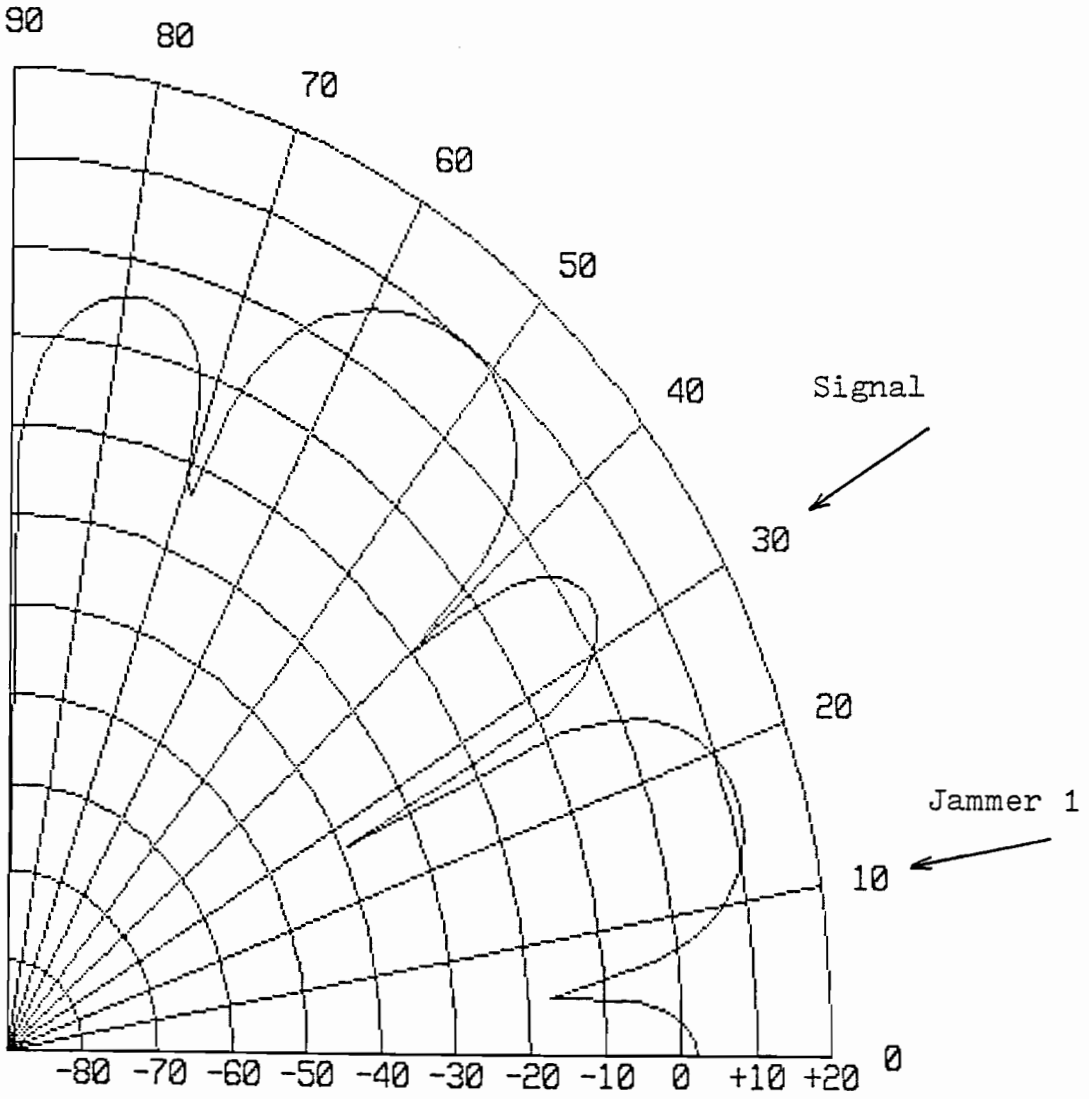
Figure T1.25 Eigenvector Sort-Adapted Antenna Plot at ϕ = 150°



Constant Elevation Cut
Elevation: 15

(Largest Eigenvalue)

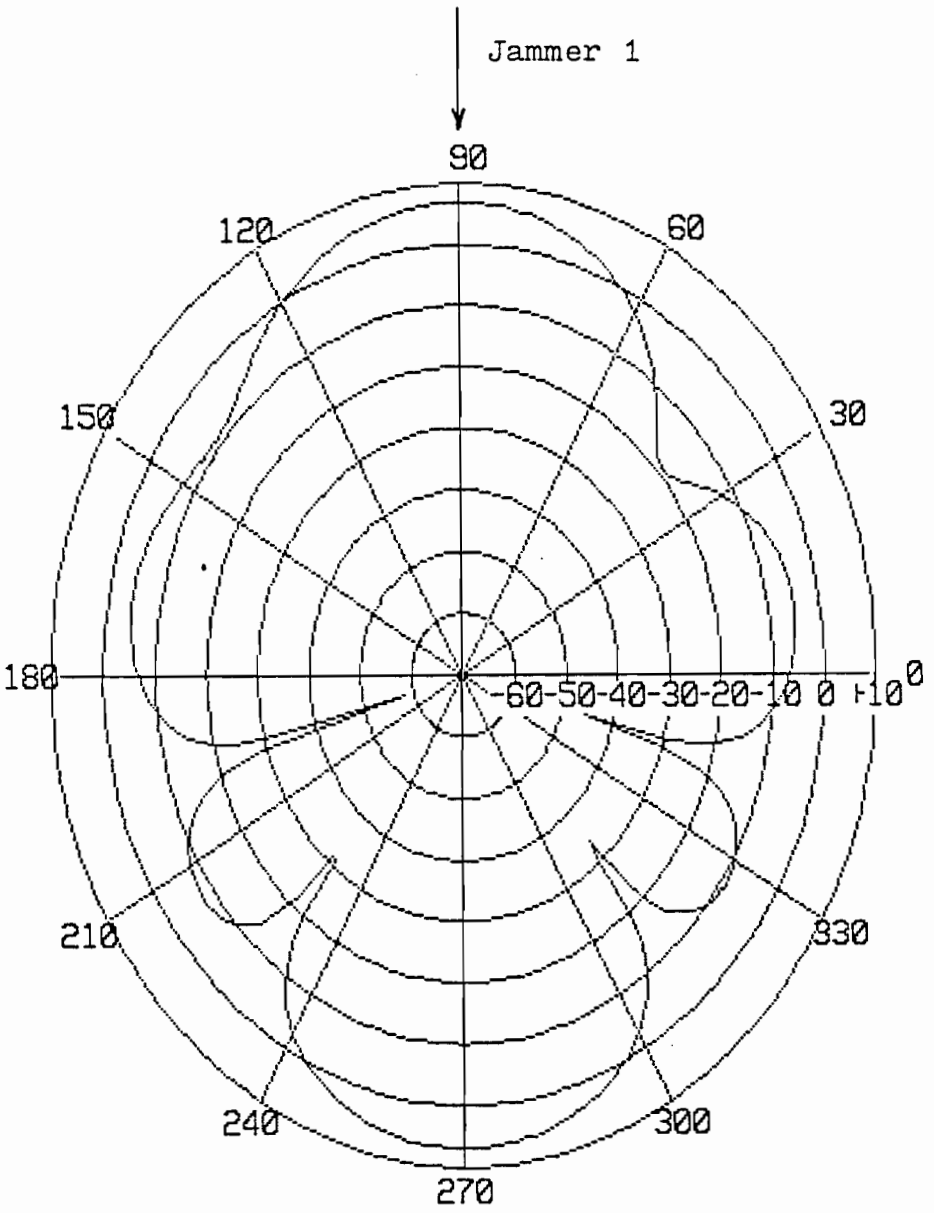
Figure T1.26 Eigenvector Sort-Adapted Antenna Plot at $\theta = 15°$



Constant Azimuthal Cut
Azimuth: 90

(Second Largest Eigenvalue)

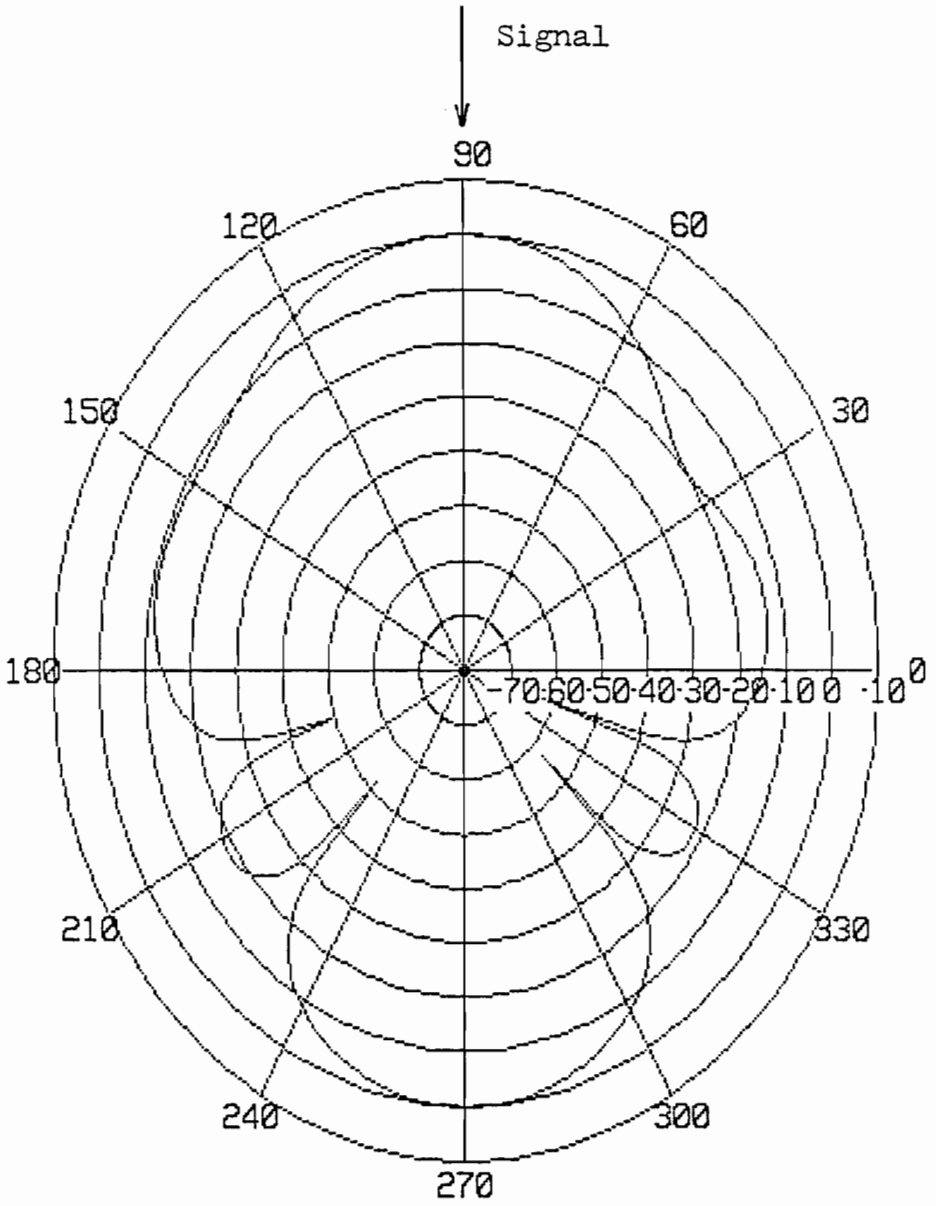
Figure T1.27 Eigenvector Sort-Adapted Antenna Plot at ϕ = 90°



Constant Elevation Cut
Elevation: 10

(Second Largest Eigenvalue)

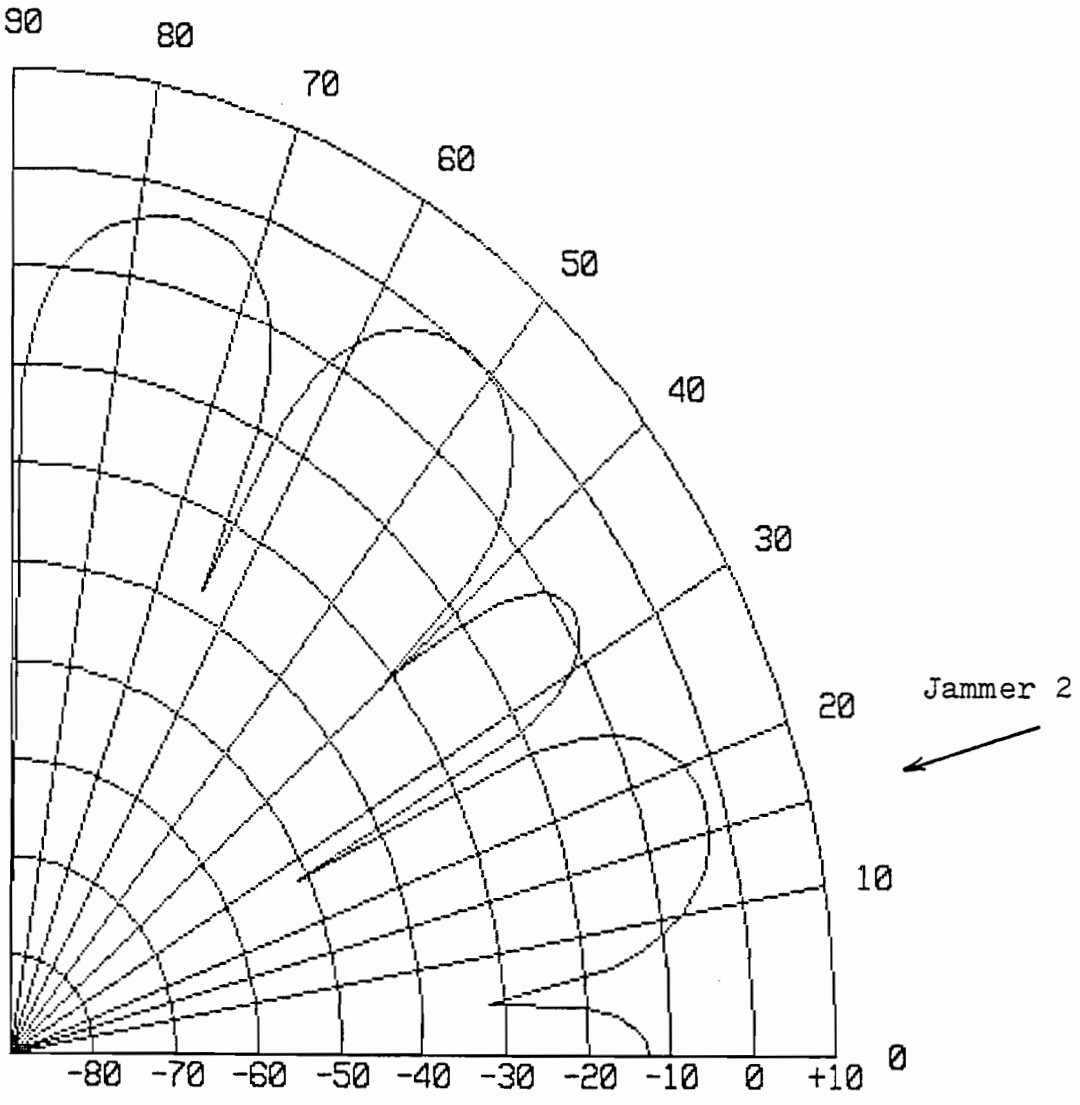
Figure T1.28 Eigenvector Sort-Adapted Antenna Plot at $\theta = 10°$



Constant Elevation Cut
Elevation: 30

(Second Largest Eigenvalue)

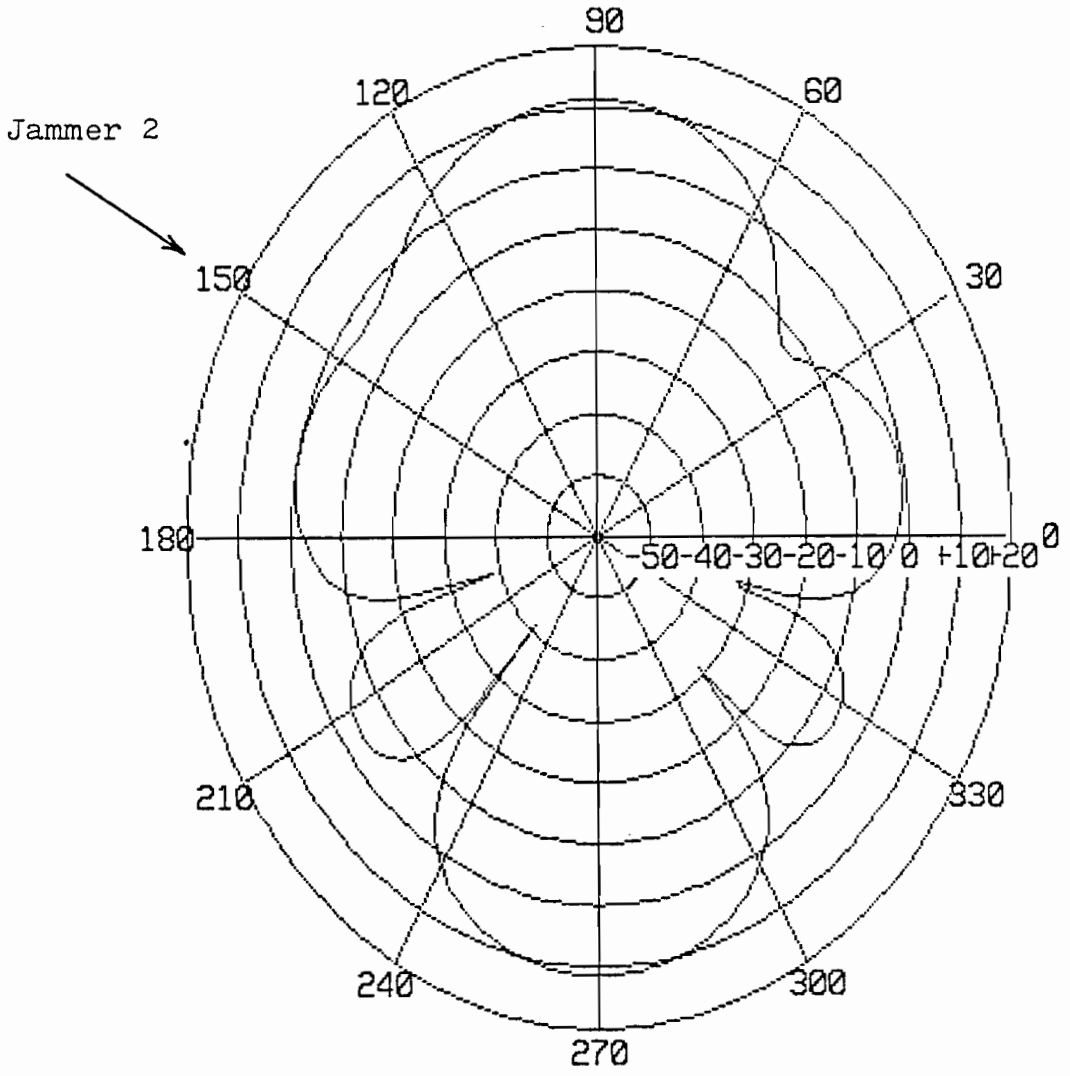
Figure T1.29 Eigenvector Sort-Adapted Antenna Plot at $\theta = 30°$



Constant Azimuthal Cut
Azimuth: 150

(Second Largest Eigenvalue)

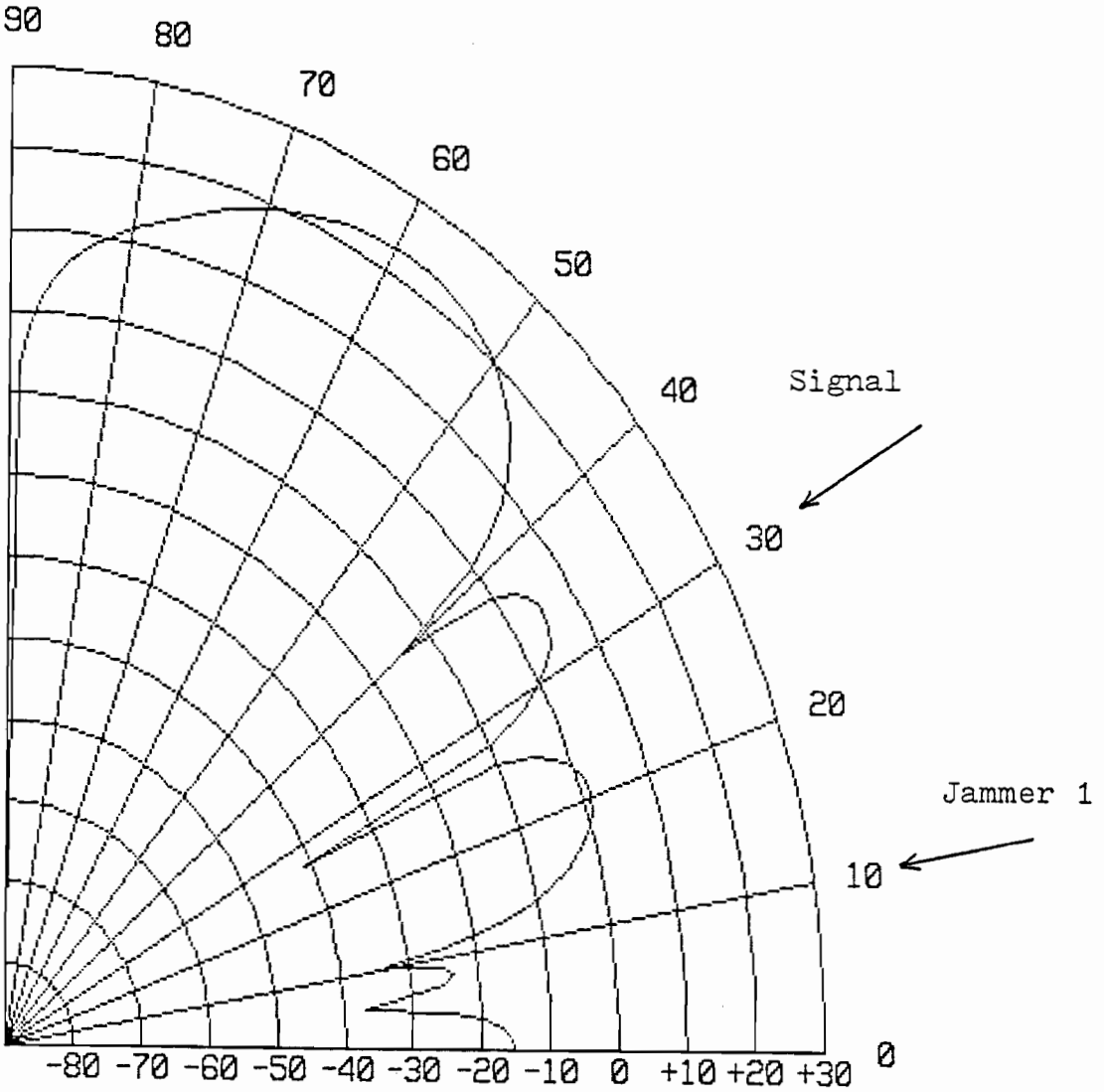
Figure T1.30 Eigenvector Sort-Adapted Antenna Plot at ϕ = 150°



Constant Elevation Cut
Elevation: 15

(Second Largest Eigenvalue)

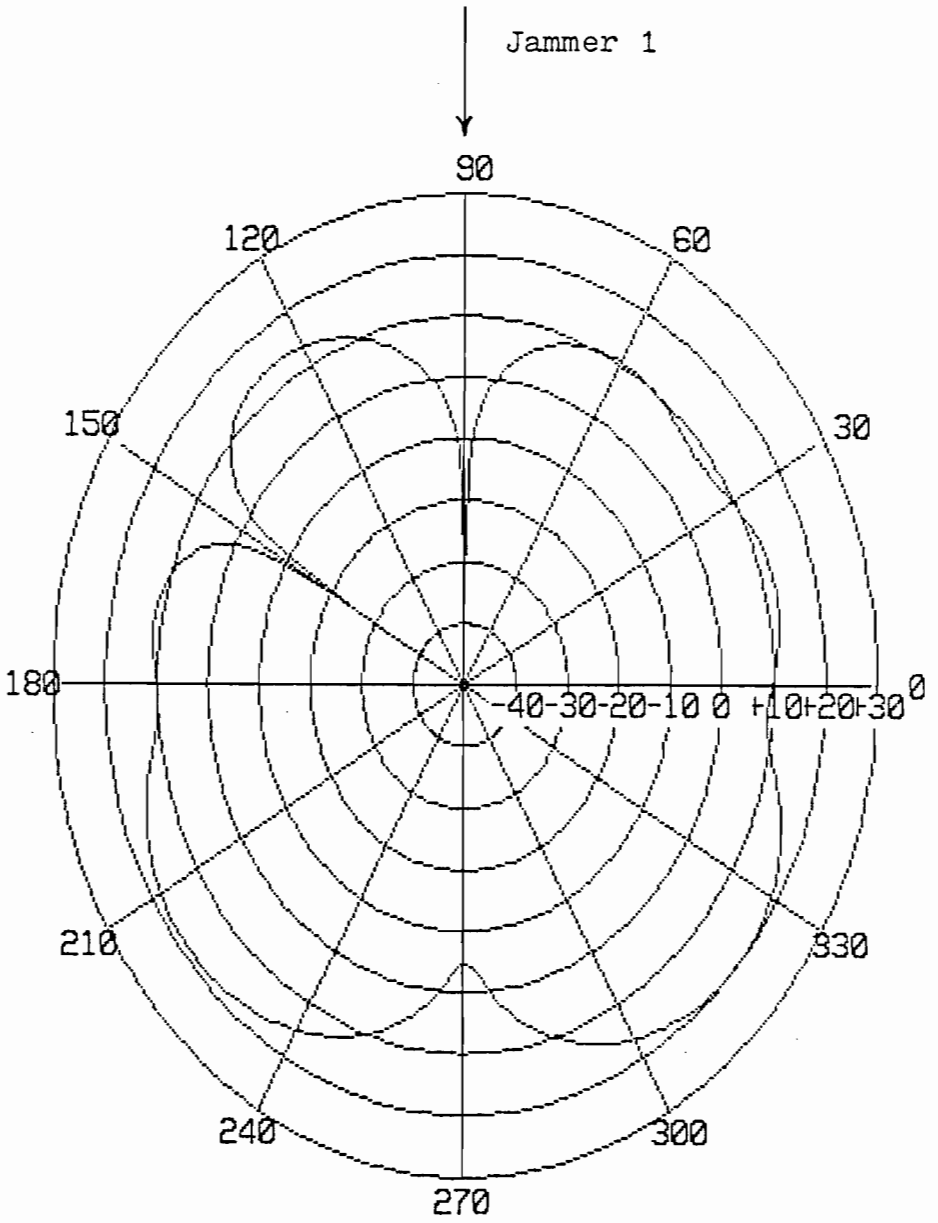
Figure T1.31 Eigenvector Sort-Adapted Antenna Plot at $\theta = 15°$



Constant Azimuthal Cut
Azimuth: 90

(Third Largest Eigenvalue)

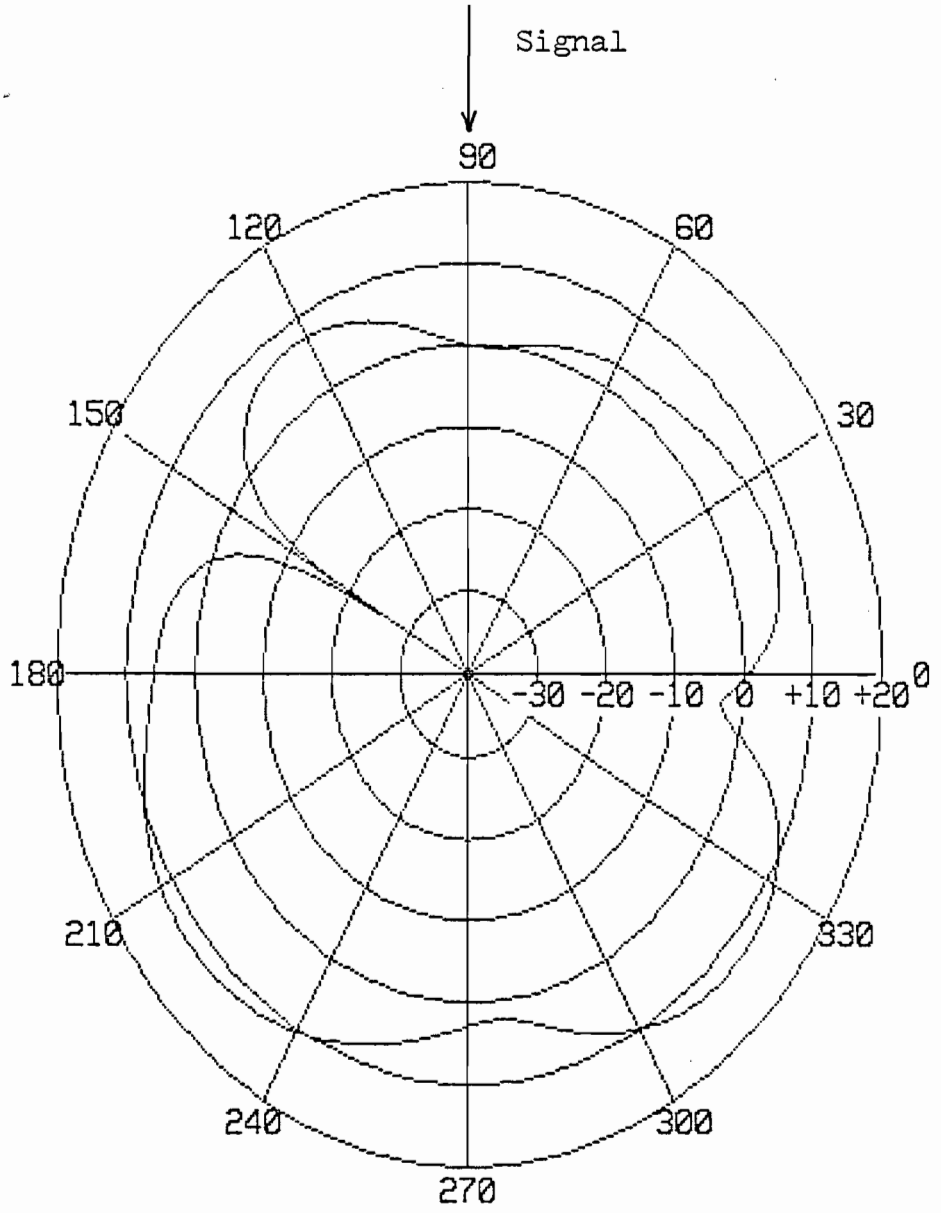
Figure T1.32 Eigenvector Sort-Adapted Antenna Plot at $\phi = 90°$



Constant Elevation Cut
Elevation: 10

(Third Largest Eigenvalue)

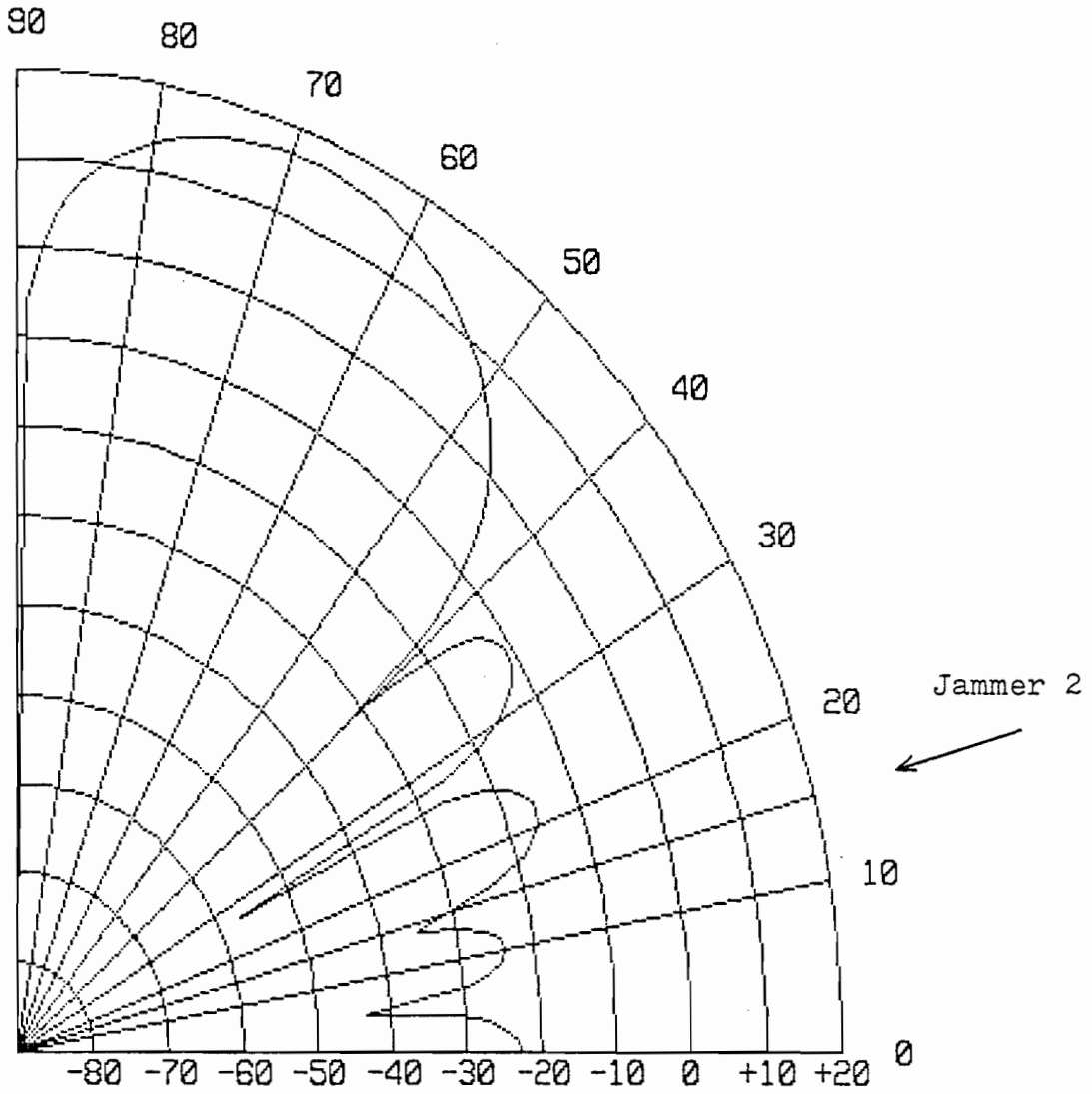
Figure T1.33 Eigenvector Sort-Adapted Antenna Plot at $\theta = 10°$



Constant Elevation Cut
Elevation: 30

(Third Largest Eigenvalue)

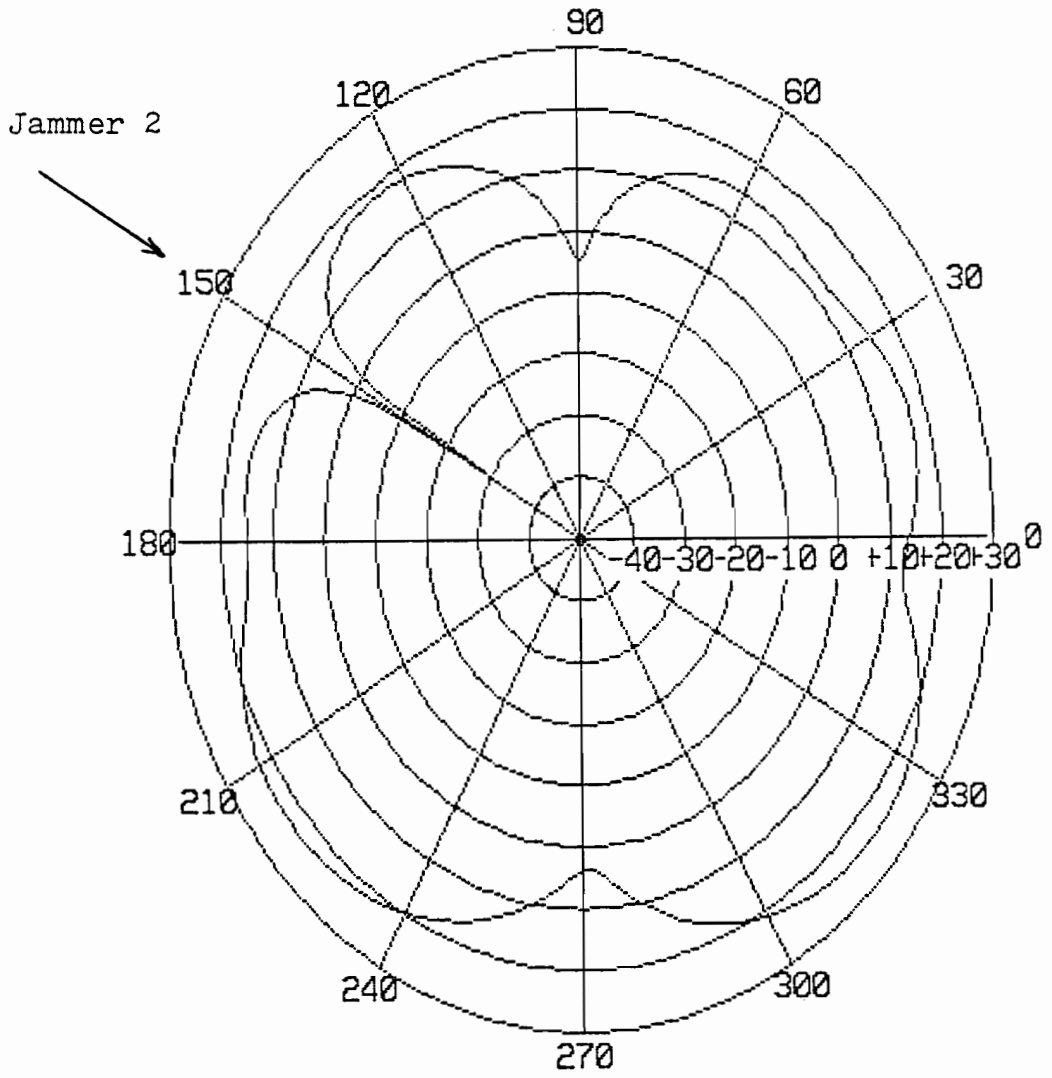
Figure T1.34 Eigenvector Sort-Adapted Antenna Plot at $\theta = 30°$



Constant Azimuthal Cut
Azimuth: 150

(Third Largest Eigenvalue)

Figure T1.35 Eigenvector Sort-Adapted Antenna Plot at ϕ = 150°



Constant Elevation Cut
Elevation: 15

(Third Largest Eigenvalue)

Figure T1.36 Eigenvector Sort-Adapted Antenna Plot at $\theta = 15°$

6.2 Test 2: Three Jammer Scenario

As a second test of the eigenvector sorting algorithm, another jammer is added to the environment at 170 degrees azimuth. In addition, the signal and jammers are all positioned at elevations of 30 degrees. The arrival angles and signal powers are shown in Table 6.2. Also, for this test, the number of samples averaged for the autocorrelation matrix is 48.

The focus of this test, and most of the remaining tests, is to determine the algorithm effectiveness in eliminating the jamming signals. This is, after all, the prime task of an adaptive algorithm in a jammed environment. Because of this, many more of the following plots are dedicated to the eigenvalue/eigenvector pair which produces the desired signal separation. In this test, since the desired signal is the smallest of the four, the eigenvector corresponding to the fourth largest eigenvalue produces the desired signal.

The first five plots (T2.1 - T2.5) show the antenna pattern variation, as a function of the eigenvectors, at the constant elevation of 30 degrees. These plots reveal that although moderate jammer separation is obtained using the first three eigenvectors, the desired signal is quite effectively separated using the fourth. The remaining plots (T2.6 - T2.13) indicate the antenna pattern variation as a function of elevation for the fourth largest eigenvalue. The plots are shown as alter-

SIGNAL	AZIMUTHAL ANGLE	ELEVATION ANGLE	POWER
COMMUNICATOR	60.0	30.0	0 dB
JAMMER 1	90.0	30.0	20 dB
JAMMER 2	150.0	30.0	20 dB
JAMMER 3	170.0	30.0	20 dB

Table 6.2 Signal Characteristics for Test 2

nating unadapted and adapted patterns in the directions of all incoming signals. The plots clearly show the nulls created in the antenna patterns at the jammer directions. In each case, the signal is given an advantage of at least 38 dB over the jamming signals.

Summary of Plots for Test 2

Fig. T2.1: Unadapted Antenna Plot at $\theta = 30°$

Fig. T2.2: Adapted Plot at $\theta = 30°$ (Largest Eigenvalue)

Fig. T2.3: Adapted Plot at $\theta = 30°$ (Second Largest Eigenvalue)

Fig. T2.4: Adapted Plot at $\theta = 30°$ (Third Largest Eigenvalue)

Fig. T2.5: Adapted Plot at $\theta = 30°$ (Fourth Largest Eigenvalue)

Fig. T2.6: Unadapted Antenna Plot at $\phi = 60°$

Fig. T2.7: Adapted Plot at $\phi = 60°$ (Fourth Largest Eigenvalue)

Fig. T2.8: Unadapted Antenna Plot at $\phi = 90°$

Fig. T2.9: Adapted Plot at $\phi = 90°$ (fourth Largest Eigenvalue)

Fig. T2.10: Unadapted Antenna Plot at $\phi = 150°$

Fig. T2.11: Adapted Plot at $\phi = 150°$ (Fourth Largest Eigenvalue)

Fig. T2.12: Unadapted Antenna Plot at $\phi = 170°$

Fig. T2.13: Adapted Plot at $\phi = 170°$ (Fourth Largest Eigenvalue)

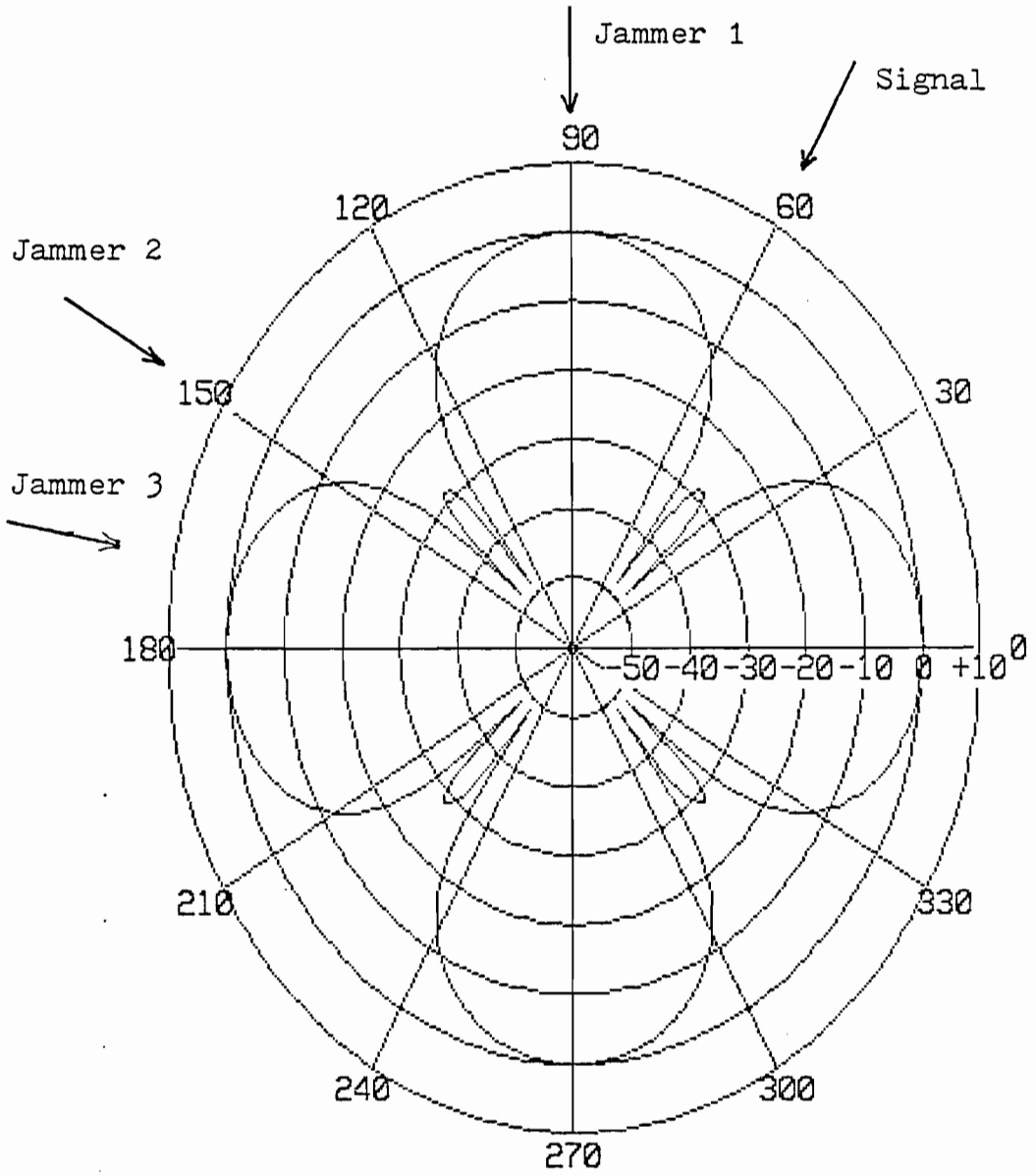
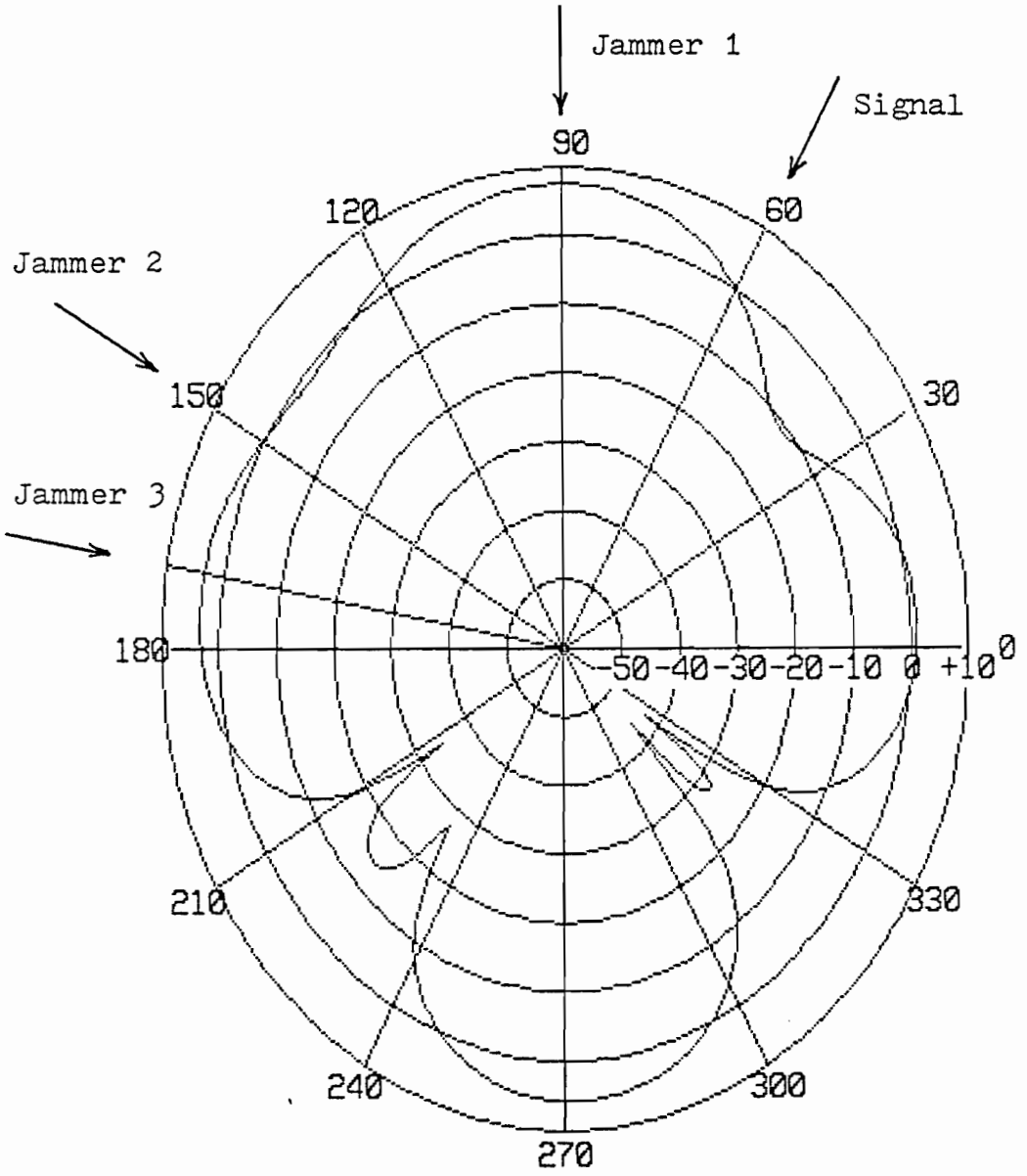


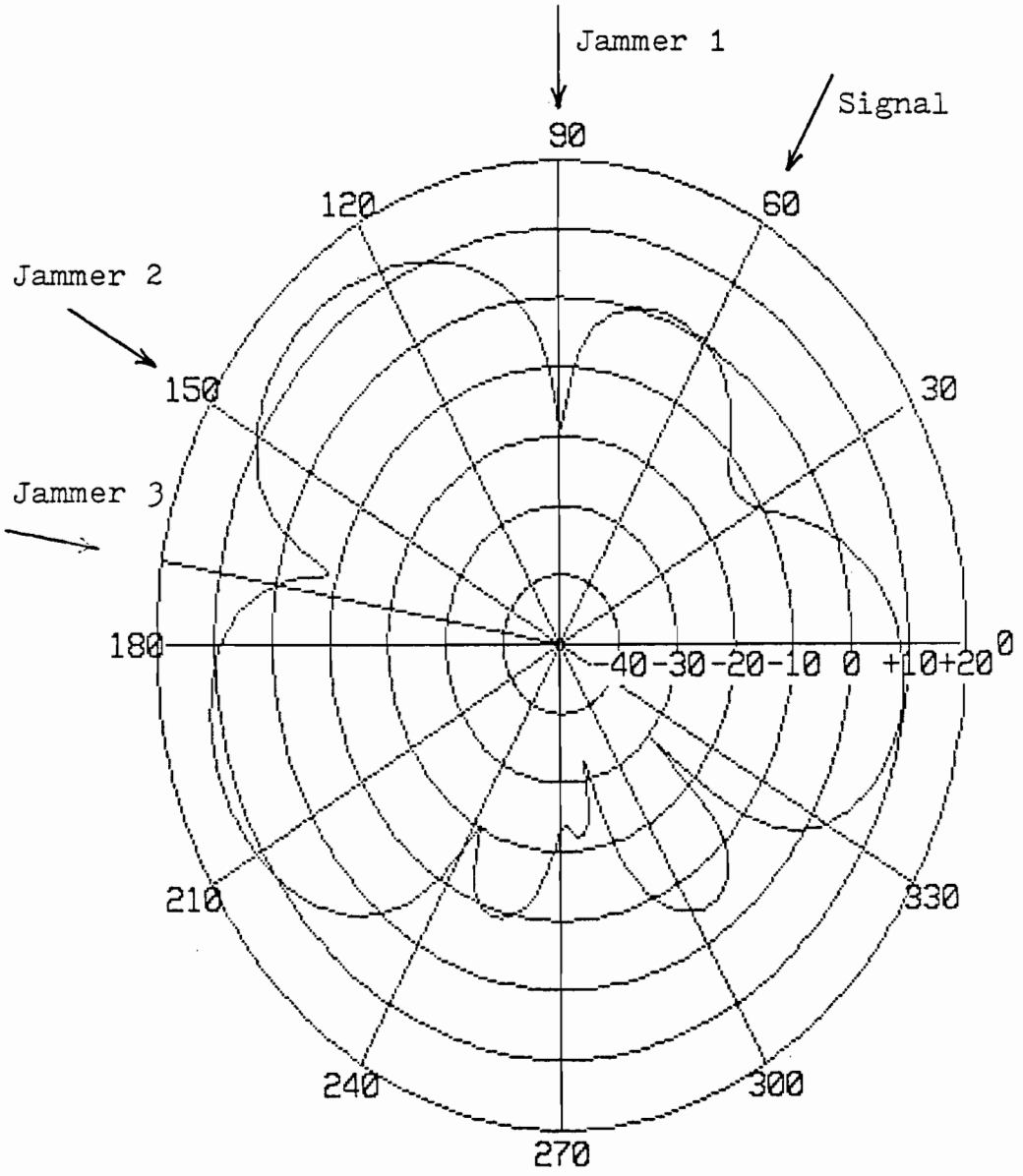
Figure T2.1 Unadapted Antenna Plot for Eigenvector Algorithm at $\theta = 30°$



Constant Elevation Cut
Elevation: 30

(Largest Eigenvalue)

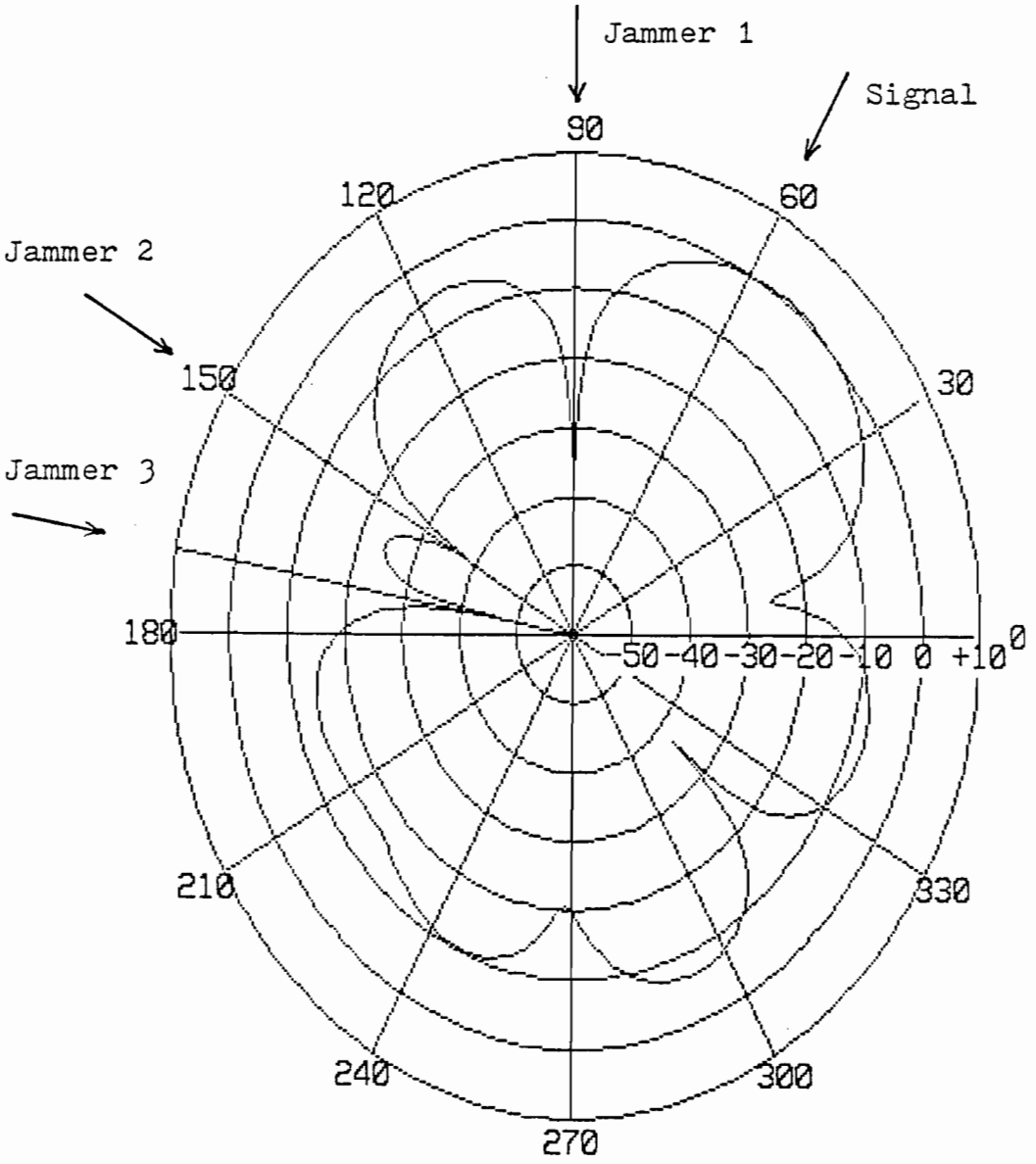
Figure T2.2 Eigenvector Sort-Adapted Antenna Plot at $\theta = 30°$



Constant Elevation Cut
Elevation: 30

(Third Largest Eigenvalue)

Figure T2.4 Eigenvector Sort-Adapted Antenna Plot at $\theta = 30°$



Constant Elevation Cut
Elevation: 30

(Fourth Largest Eigenvalue)

Figure T2.5 Eigenvector Sort-Adapted Antenna Plot at $\theta = 30°$

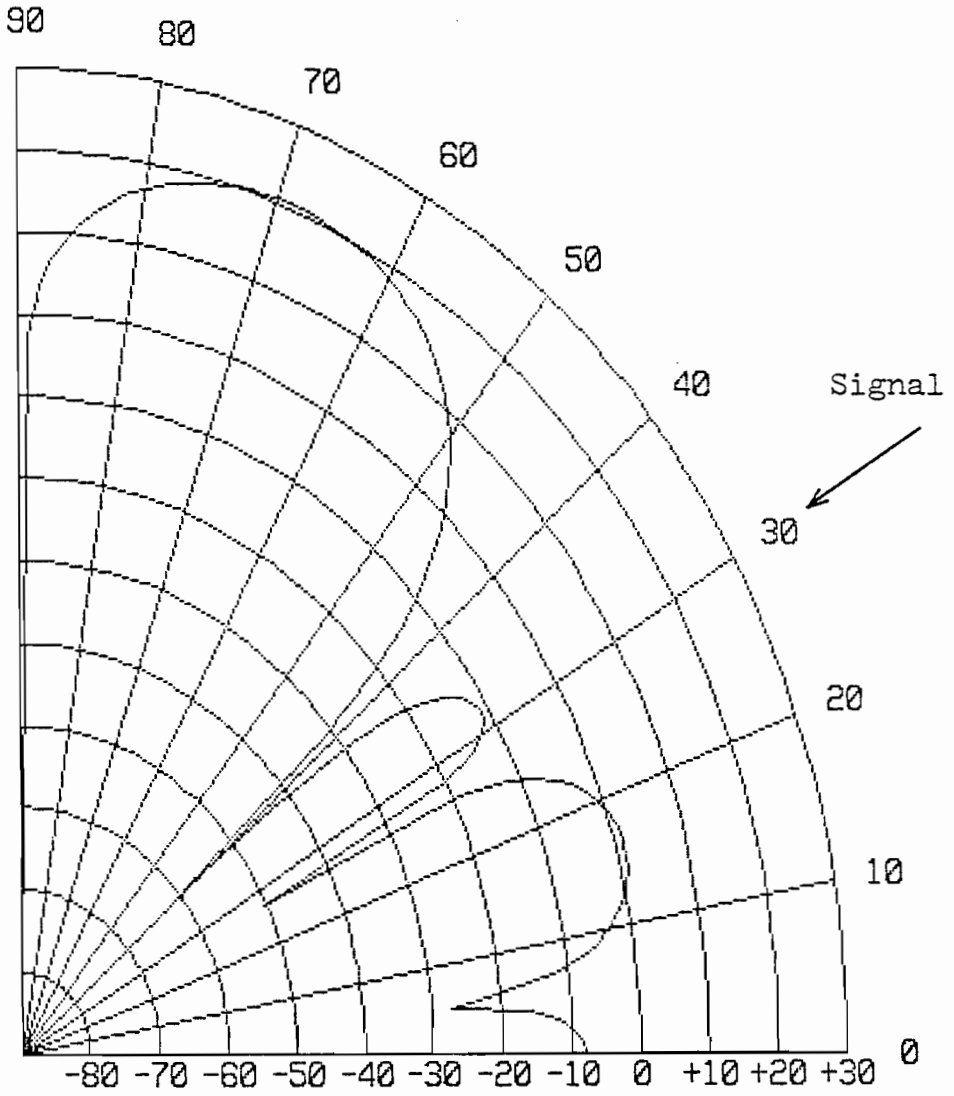
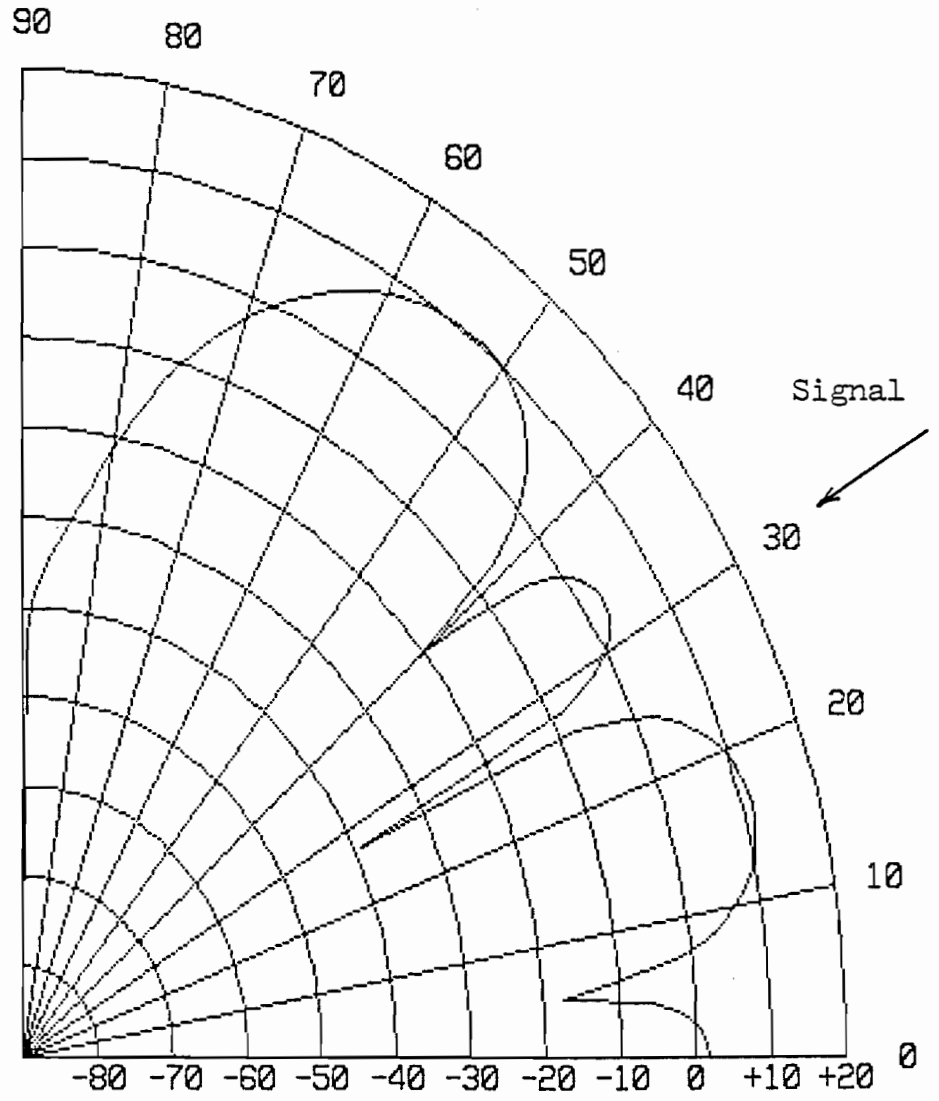


Figure T2.6 Unadapted Antenna Plot for Eigenvector Algorithm at ϕ = 60°



Constant Azimuthal Cut
Azimuth: 60

(Fourth Largest Eigenvalue)

Figure T2.7 Eigenvector Sort-Adapted Antenna Plot at ϕ = 60°

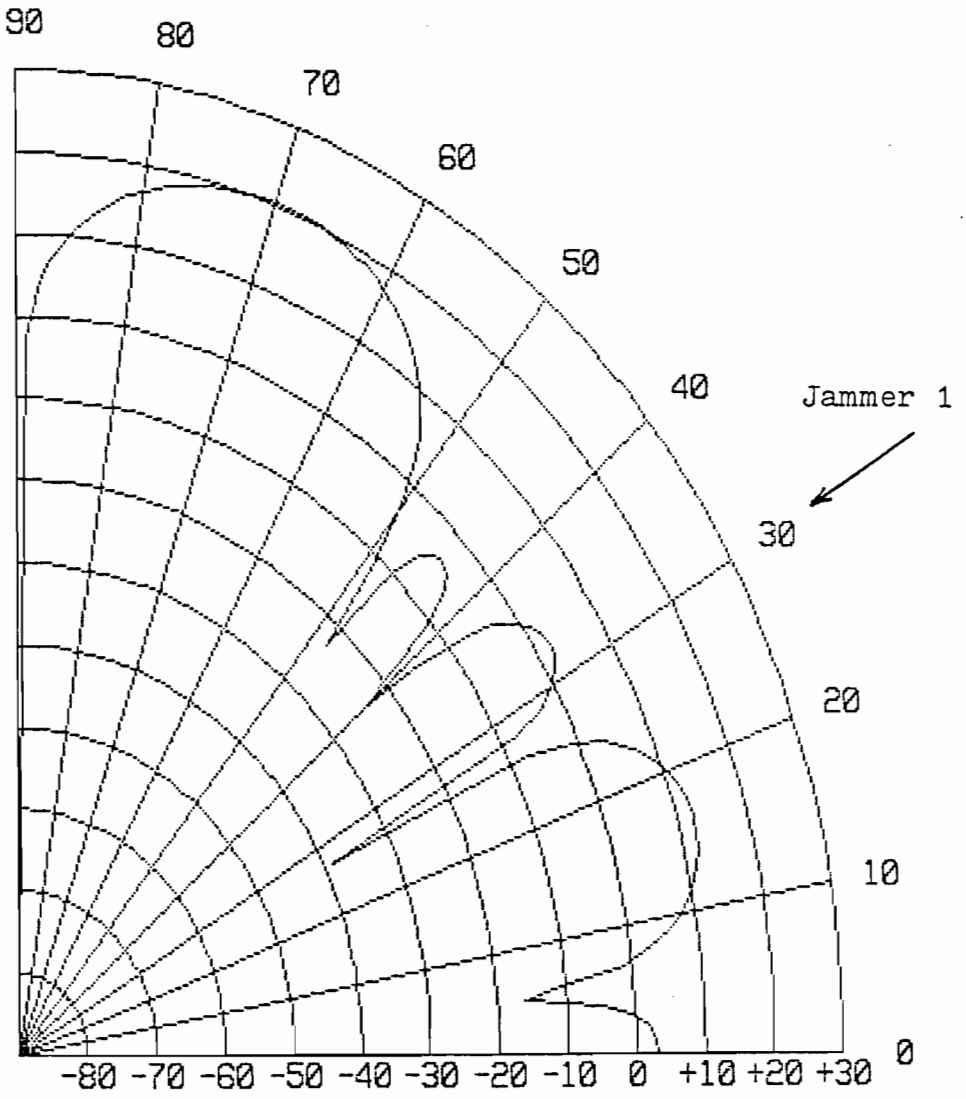
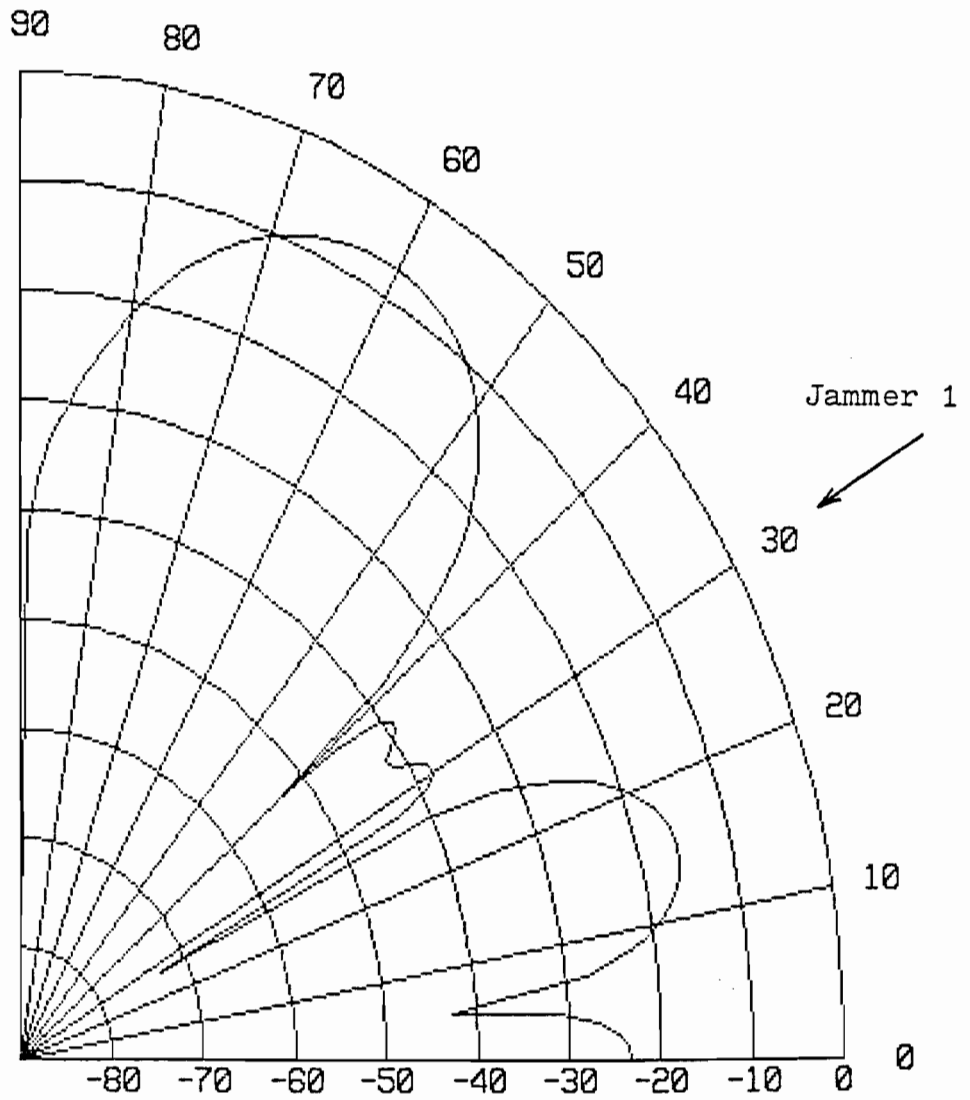


Figure T2.8 Unadapted Antenna Plot for Eigenvector Algorithm at ϕ = 90°



(Fourth Largest Eigenvalue)

Figure T2.9 Eigenvector Sort-Adapted Antenna Plot at $\phi = 90°$

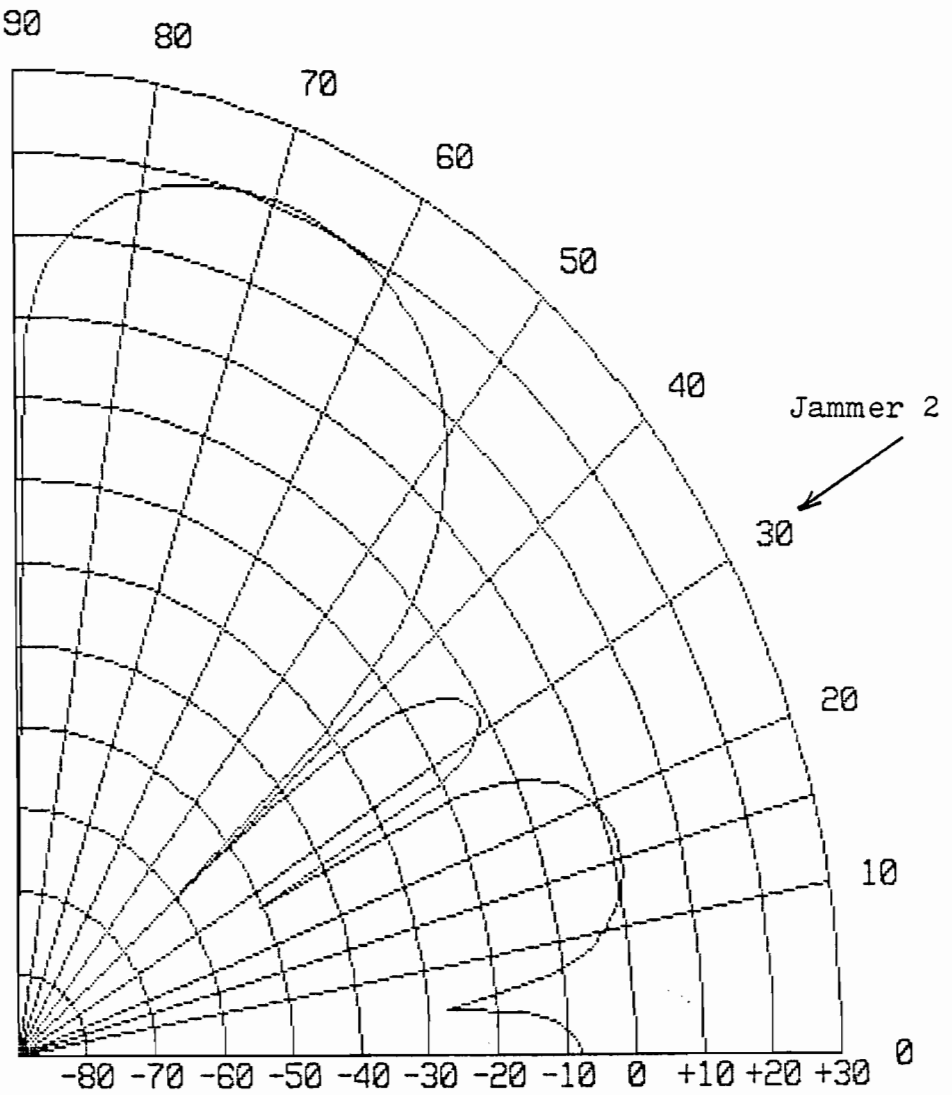
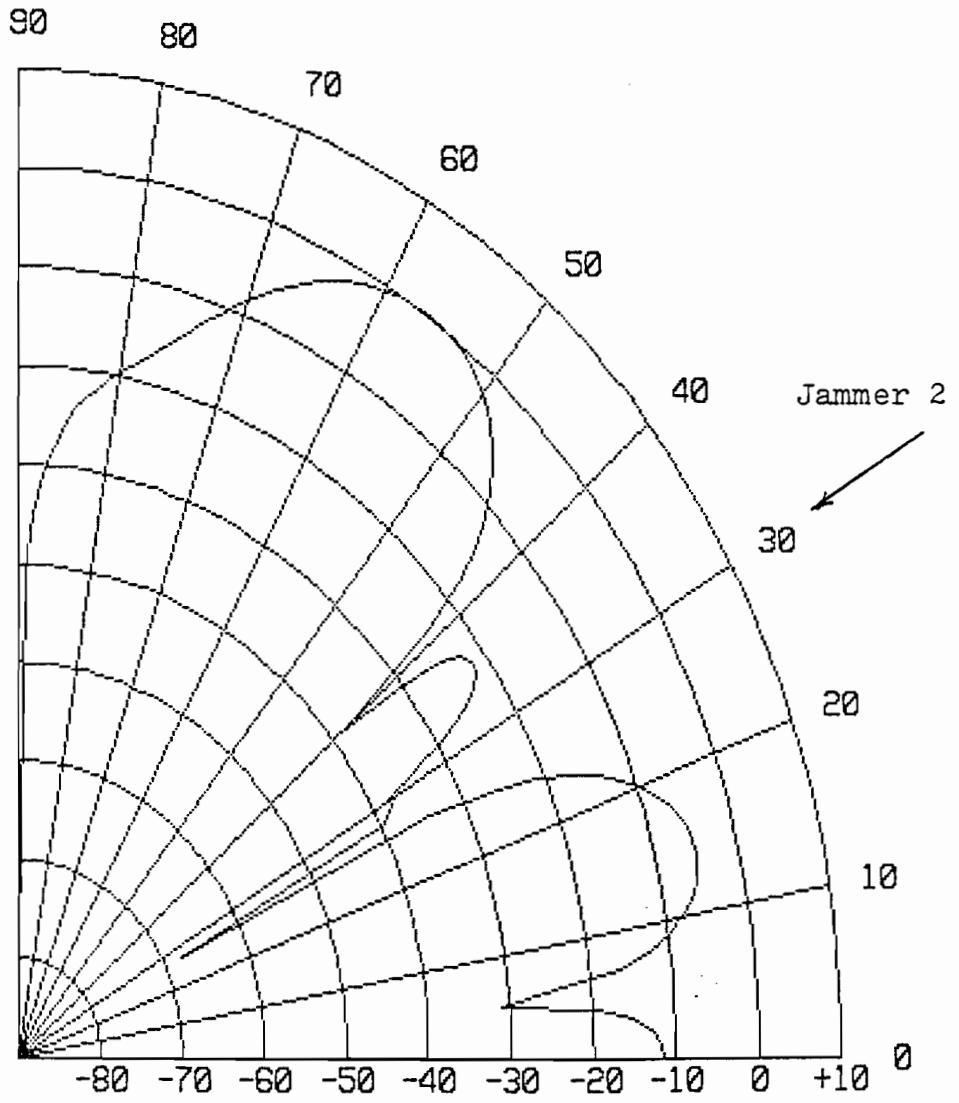


Figure T2.10 Unadapted Antenna Plot for Eigenvector Algorithm at ϕ = 150°



Constant Azimuthal Cut
Azimuth: 150

(Fourth Largest Eigenvalue)

Figure T2.11 Eigenvector Sort-Adapted Antenna Plot at ϕ = 150°

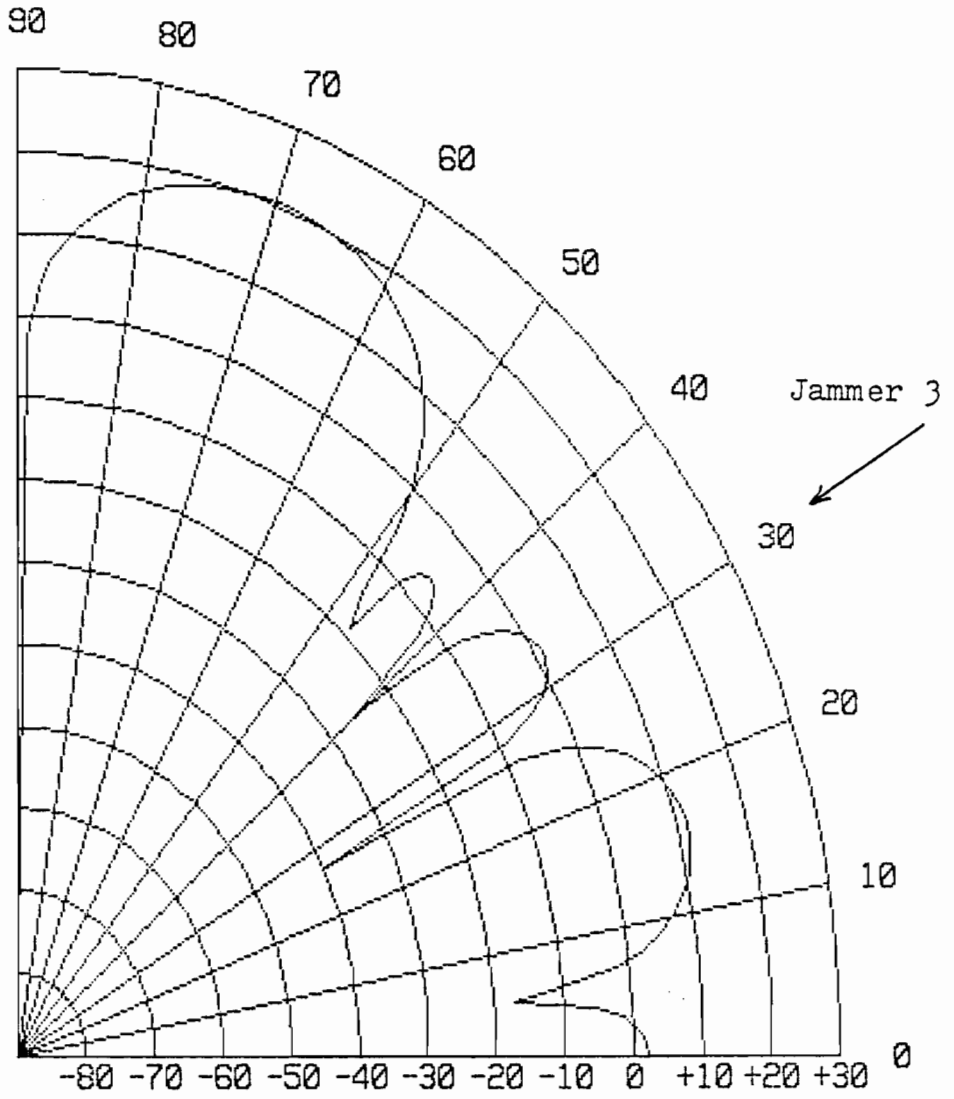
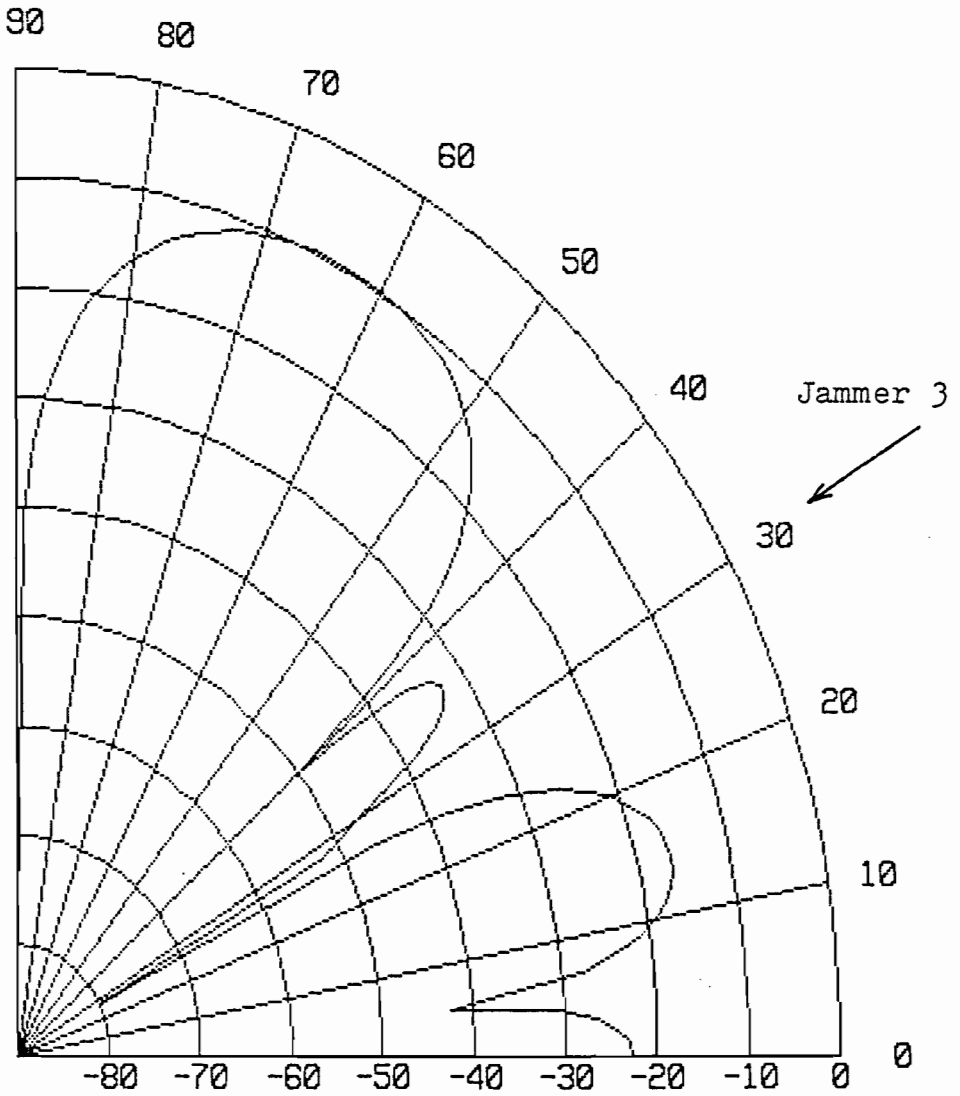


Figure T2.12 Unadapted Antenna Plot for Eigenvector Algorithm at ϕ = 170°



Constant Azimuthal Cut
Azimuth: 170

(Fourth Largest Eigenvalue)

Figure T2.13 Eigenvector Sort-Adapted Antenna Plot at ϕ = 170°

6.3 Test 3: Dependence of Pattern on Number of Averages

Since a crucial factor in the performance of the eigenvector sorting algorithm is the proper estimation of the autocorrelation matrix, it will be instructive to assess the pattern dependence on the number of averages performed. The autocorrelation matrix is the only external information used by the algorithm to evaluate the signal environment at the sensor array, and it is conceivable that the algorithm performance is proportional to the number of samples averaged.

To investigate this, the signals are arranged as shown in Table 6.3. Note that the signals are again arriving at 30 degrees elevation, and the azimuthal spread has been reduced to 30 degrees as well.

As mentioned at the beginning of this section, the number of samples used by the simulation, per bit of information, is 16. The autocorrelation matrix is produced by averaging over 2, 3, 4, and 6 bits of the information signal. This then corresponds to averaging intervals of 32, 48, 64, and 96 samples respectively. The results of these averages are shown as constant elevation plots at the arrival angle of 30 degrees. This, in effect, produces a scaled representation of the antenna array factor, since the contribution of the dipole element pattern is constant at a fixed elevation.

SIGNAL	AZIMUTHAL ANGLE	ELEVATION ANGLE	POWER
COMMUNICATOR	90.0	30.0	0 dB
JAMMER 1	60.0	30.0	0 dB
JAMMER 2	120.0	30.0	0 dB

Table 6.3 Signal Characteristics for Test 3

The patterns show that the nulling effect of the third eigenvector is apparent even at 32 samples, and is quite pronounced at 48. More samples beyond this, however, result in a negligible improvement in the nulling capability. On the other hand, the jammer separation achieved by the first and second eigenvalues steadily becomes more apparent. This test makes it clear that the algorithm characteristics are strongly dependent on the quality of the autocorrelation matrix estimate. Also, we see that once nulling has begun, very small returns are obtained from further averaging.

Summary of Plots for Test 3

Fig. T3.1: Unadapted Antenna Plot at $\theta = 30°$

Fig. T3.2: Adapted Plot at $\theta = 30°$
(Largest Eigenvalue)
(Number of Samples Averaged = 32)

Fig. T3.3: Adapted Plot at $\theta = 30°$
(Second Largest Eigenvalue)
(Number of Samples Averaged = 32)

Fig. T3.4: Adapted Plot at $\theta = 30°$
(Third Largest Eigenvalue)
(Number of Samples Averaged = 32)

Fig. T3.5: Adapted Plot at $\theta = 30°$
(Largest Eigenvalue)
(Number of Samples Averaged = 48)

Fig. T3.6: Adapted Plot at $\theta = 30°$
(Second Largest Eigenvalue)
(Number of Samples Averaged = 48)

Fig. T3.7: Adapted Plot at $\theta = 30°$
(Third Largest Eigenvalue)
(Number of Samples Averaged = 48)

Fig. T3.8: Adapted Plot at $\theta = 30°$
(Largest Eigenvalue)
(Number of Samples Averaged = 64)

Fig. T3.9: Adapted Plot at $\theta = 30°$
(Second Largest Eigenvalue)
(Number of Samples Averaged = 64)

Fig T3.10: Adapted Plot at $\theta = 30°$
(Third Largest Eigenvalue)
(Number of Samples Averaged = 64)

Fig. T3.11: Adapted Plot at $\theta = 30°$
(Largest Eigenvalue)
(Number of Samples Averaged = 96)

Fig. T3.12: Adapted Plot at $\theta = 30°$
(Second Largest Eigenvalue)
(Number of Samples Averaged = 96)

Fig. T3.13: Adapted Plot at $\theta = 30°$
(Third Largest Eigenvalue)
(Number of Samples Averaged = 96)

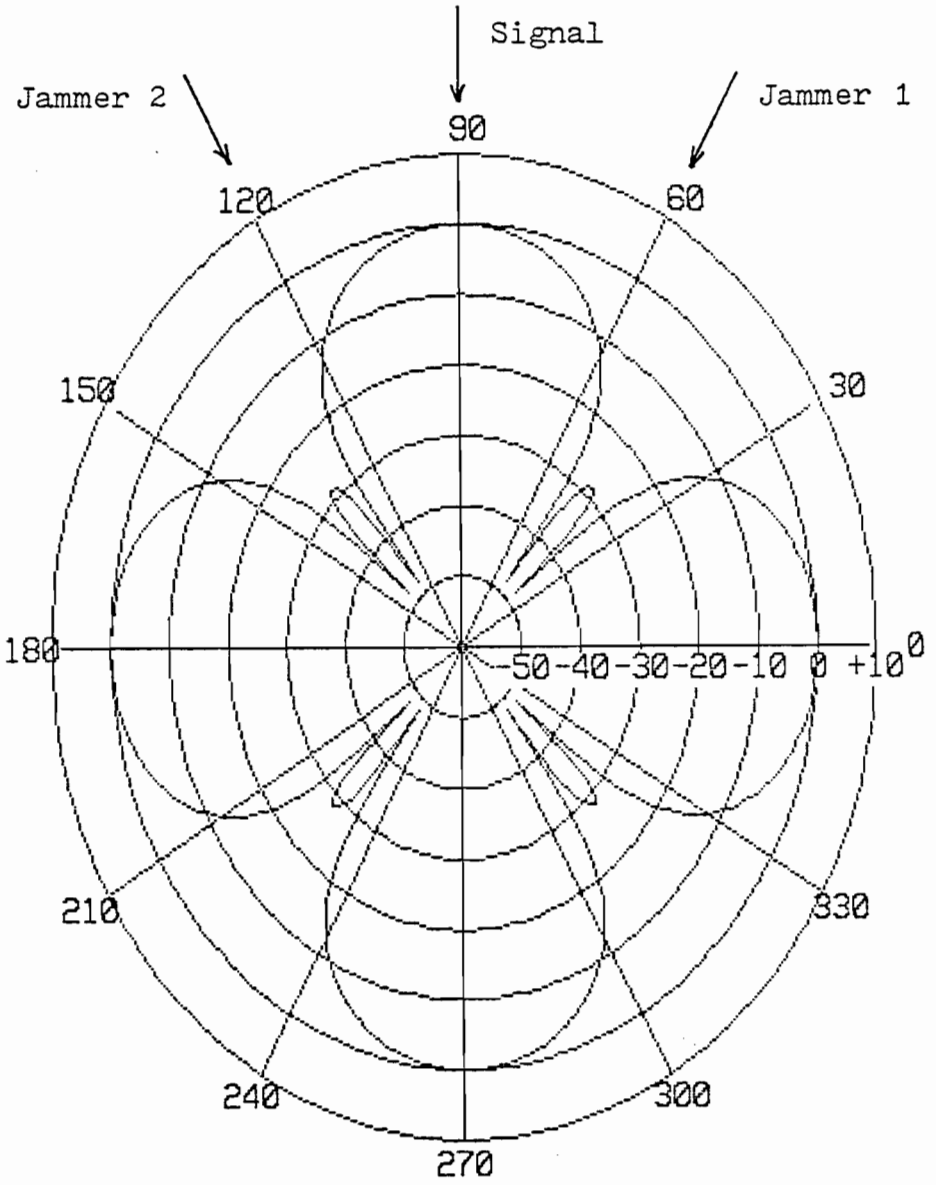
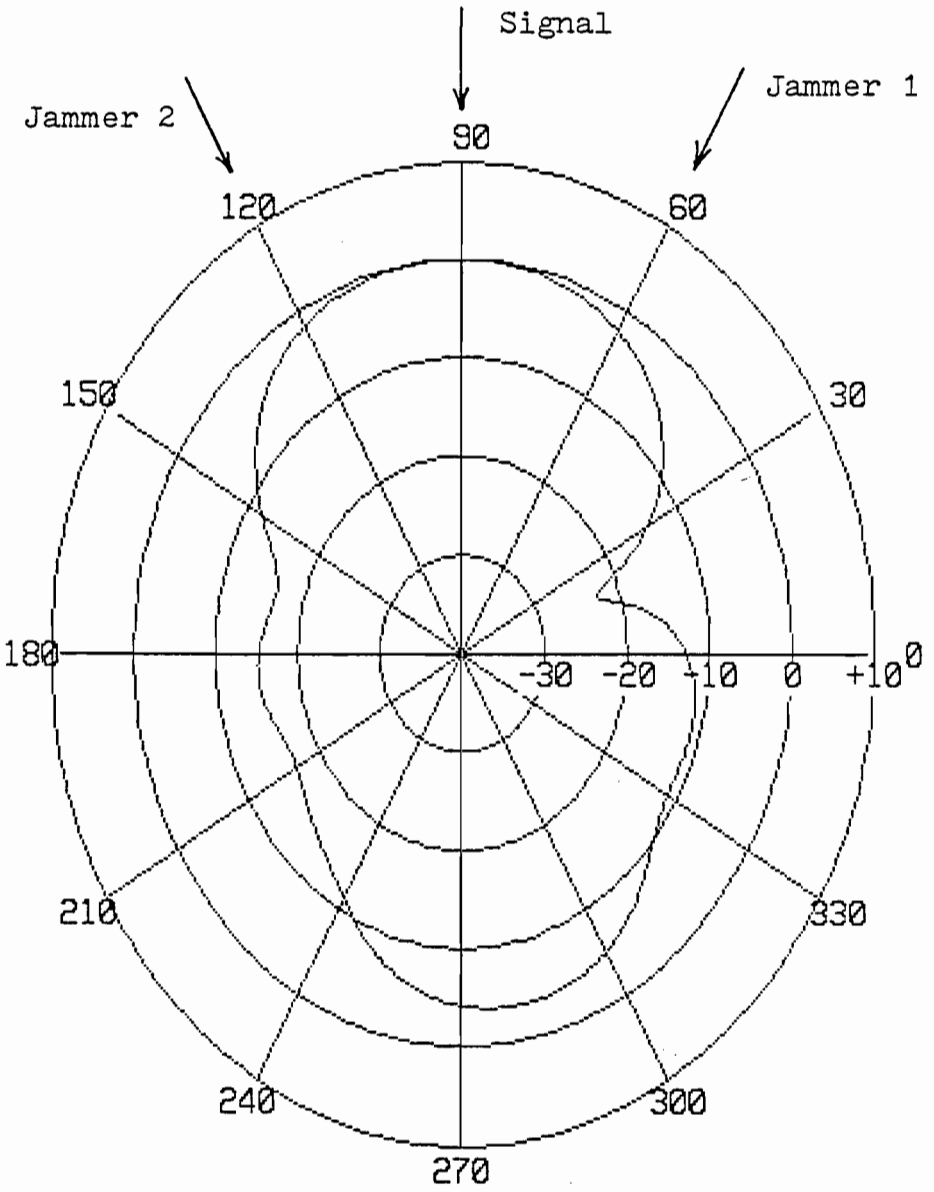


Figure T3.1 Unadapted Antenna Plot for Eigenvector Algorithm at $\theta = 30°$

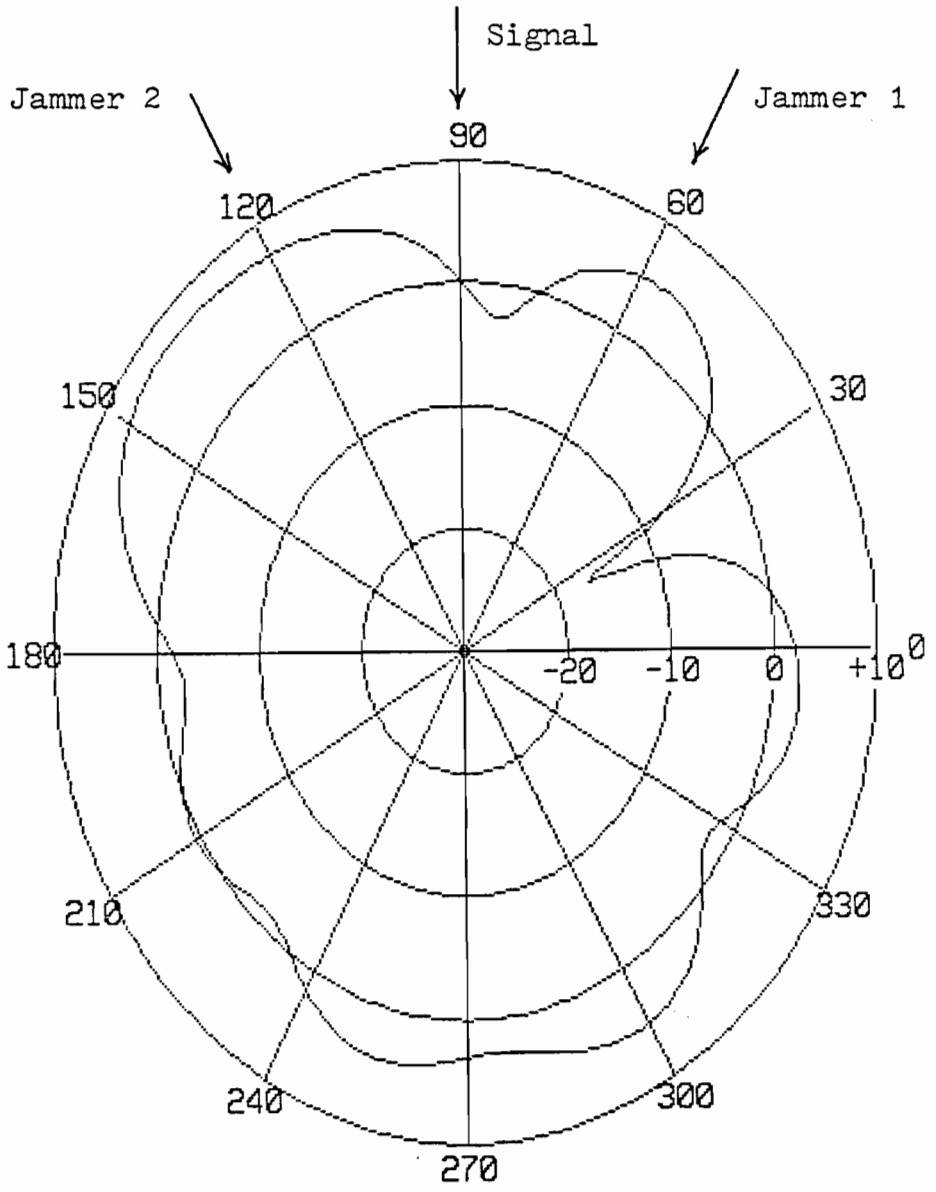


Constant Elevation Cut
Elevation: 30

(Largest Eigenvalue)

(Number of Samples Averaged = 32)

Figure T3.2 Eigenvector Sort-Adapted Antenna Plot at $\theta = 30°$

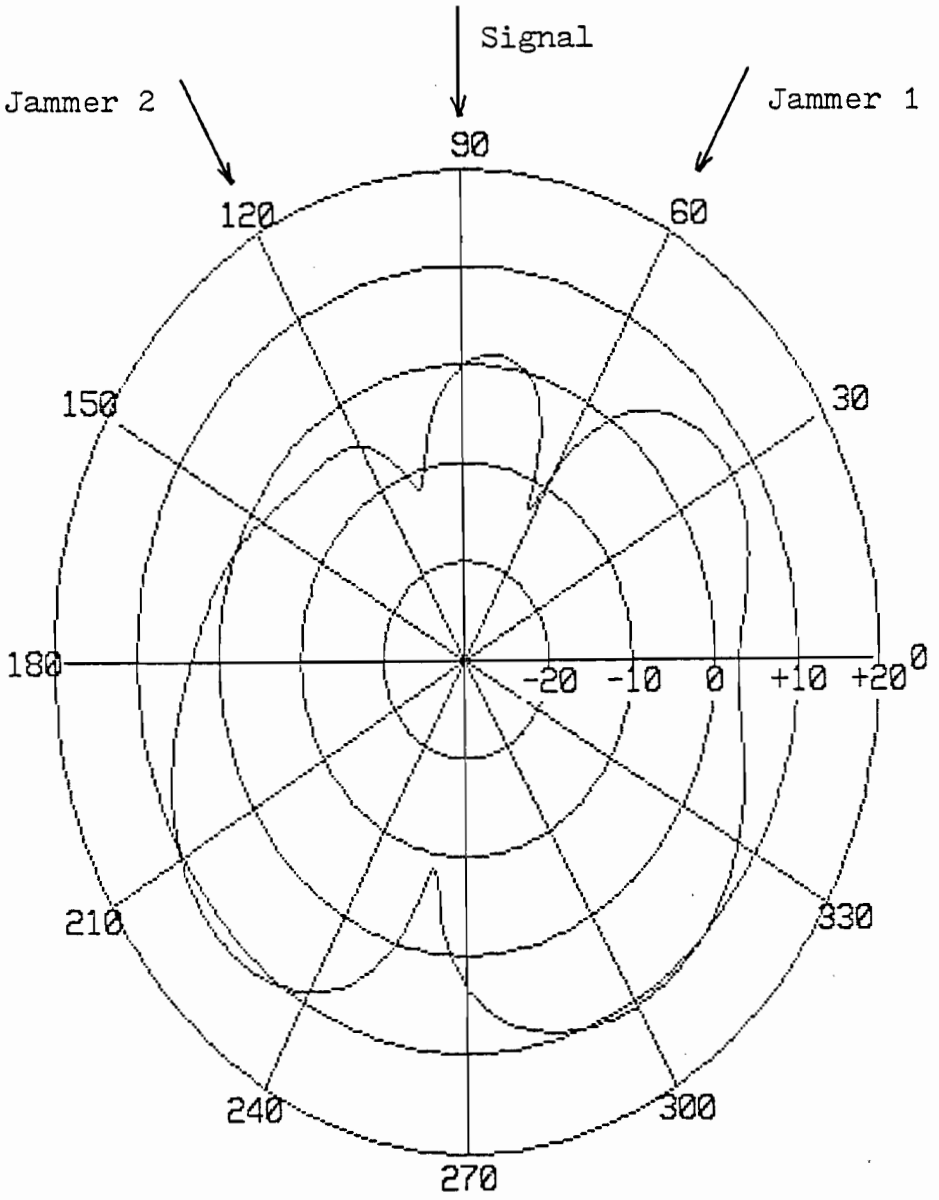


Constant Elevation Cut
Elevation: 30

(Second Largest Eigenvalue)

(Number of Samples Averaged = 32)

Figure T3.3 Eigenvector Sort-Adapted Antenna Plot at $\theta = 30°$

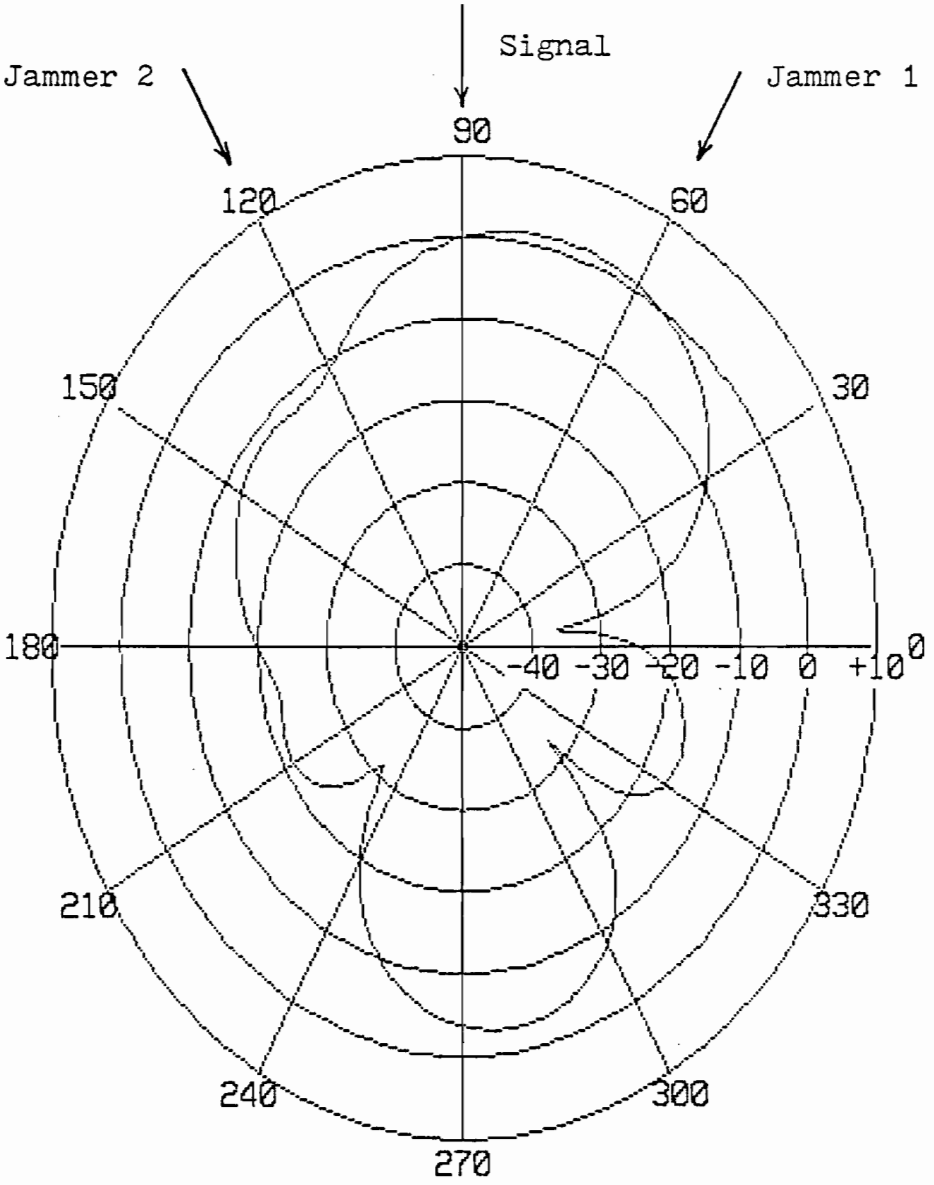


Constant Elevation Cut
Elevation: 30

(Third Largest Eigenvalue)

(Number of Samples Averaged = 32)

Figure T3.4 Eigenvector Sort-Adapted Antenna Plot at $\theta = 30°$

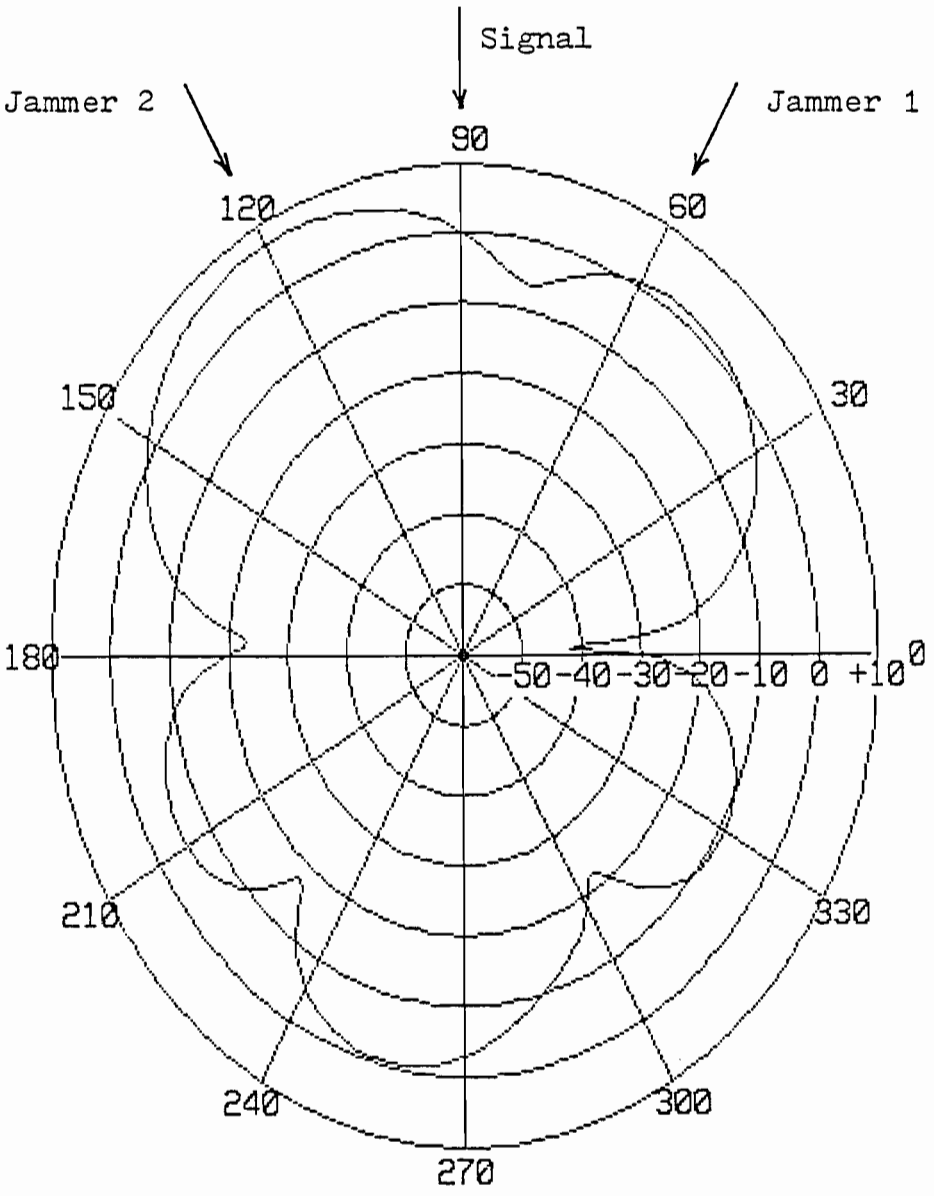


Constant Elevation Cut
Elevation: 30

(Largest Eigenvalue)

(Number of Samples Averaged = 48)

Figure T3.5 Eigenvector Sort-Adapted Antenna Plot at $\theta = 30°$

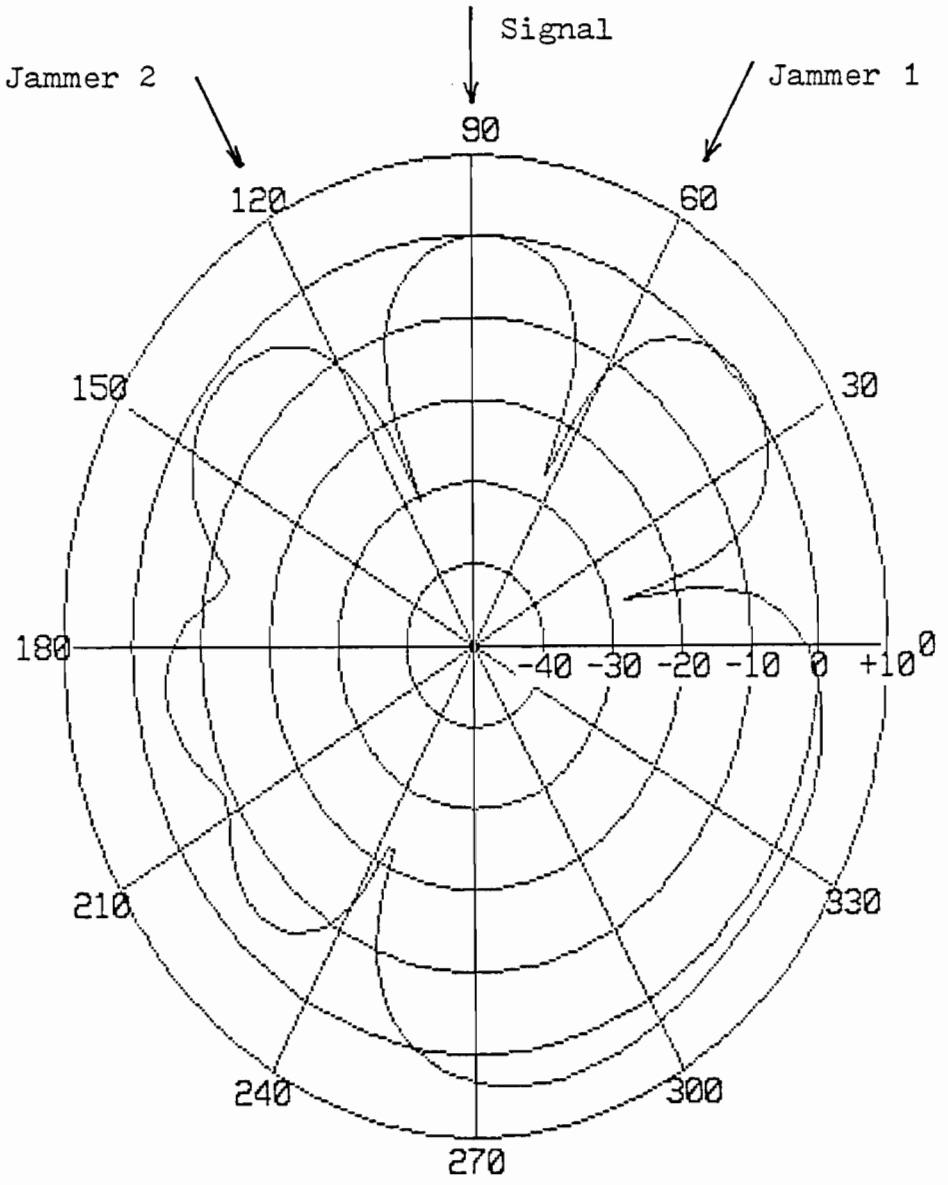


Constant Elevation Cut
Elevation: 30

(Second Largest Eigenvalue)

(Number of Samples Averaged = 48)

Figure T3.6 Eigenvector Sort-Adapted Antenna Plot at $\theta = 30°$

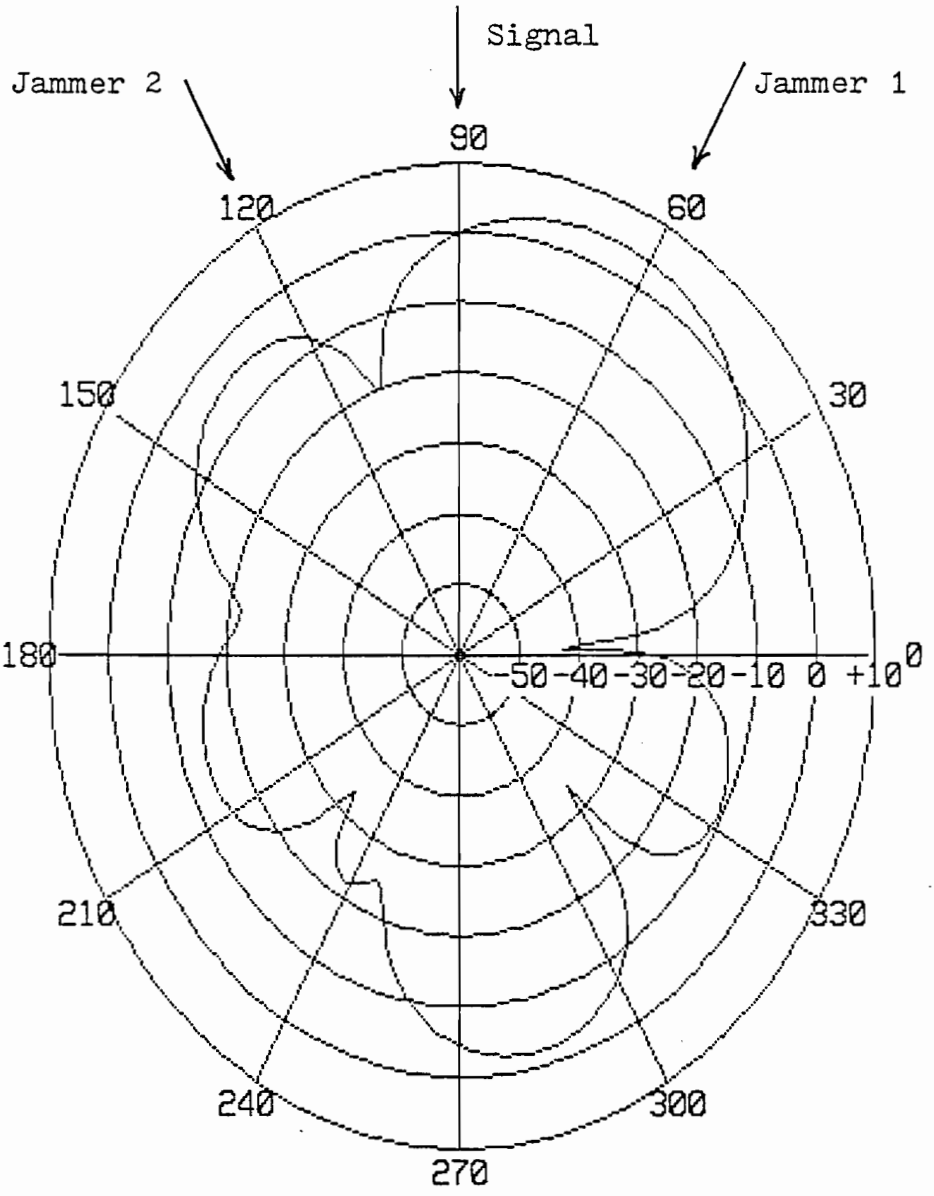


Constant Elevation Cut
Elevation: 30

(Third Largest Eigenvalue)

(Number of Samples Averaged = 48)

Figure T3.7 Eigenvector Sort-Adapted Antenna Plot at $\theta = 30°$

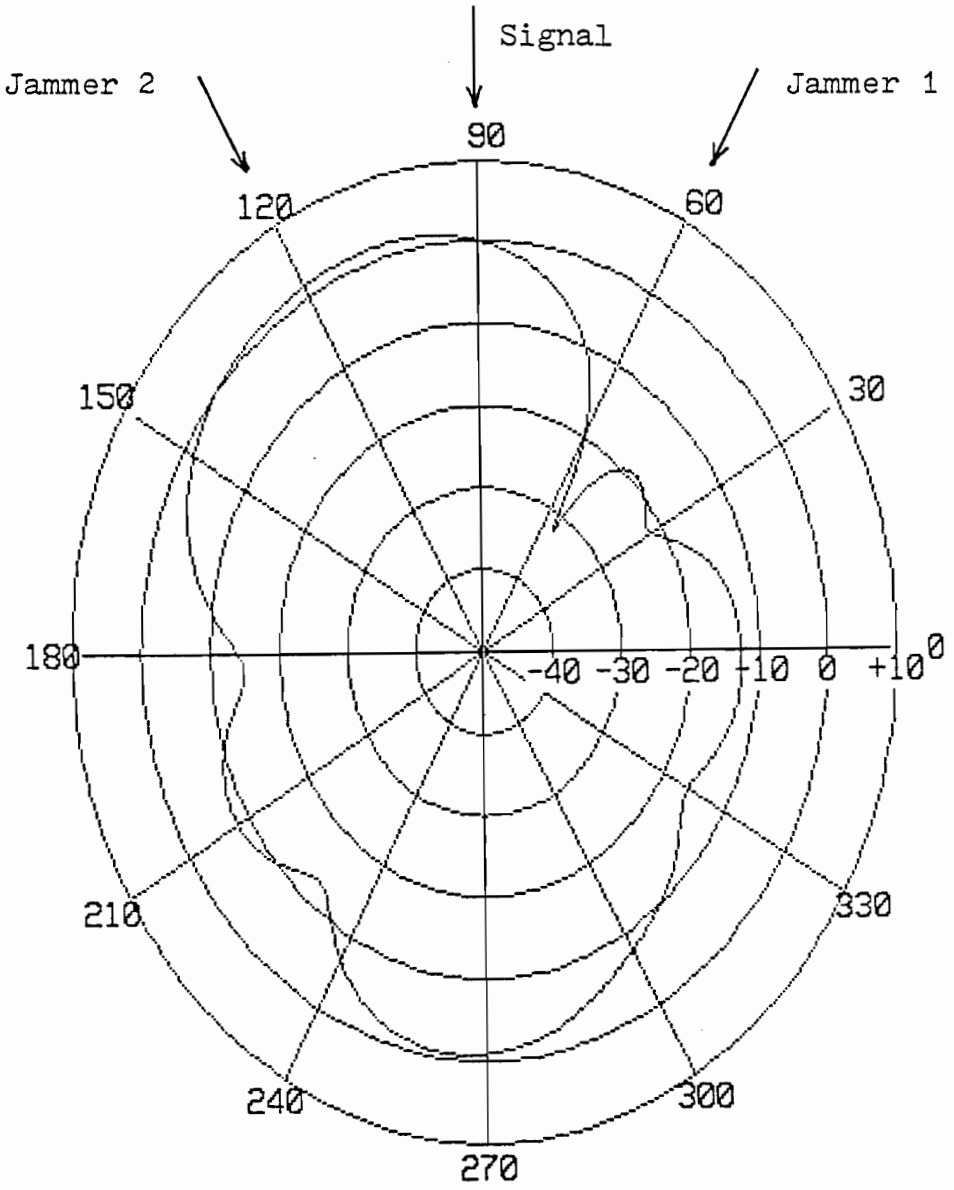


Constant Elevation Cut
Elevation: 30

(Largest Eigenvalue)

(Number of Samples Averaged = 64)

Figure T3.8 Eigenvector Sort-Adapted Antenna Plot at $\theta = 30°$

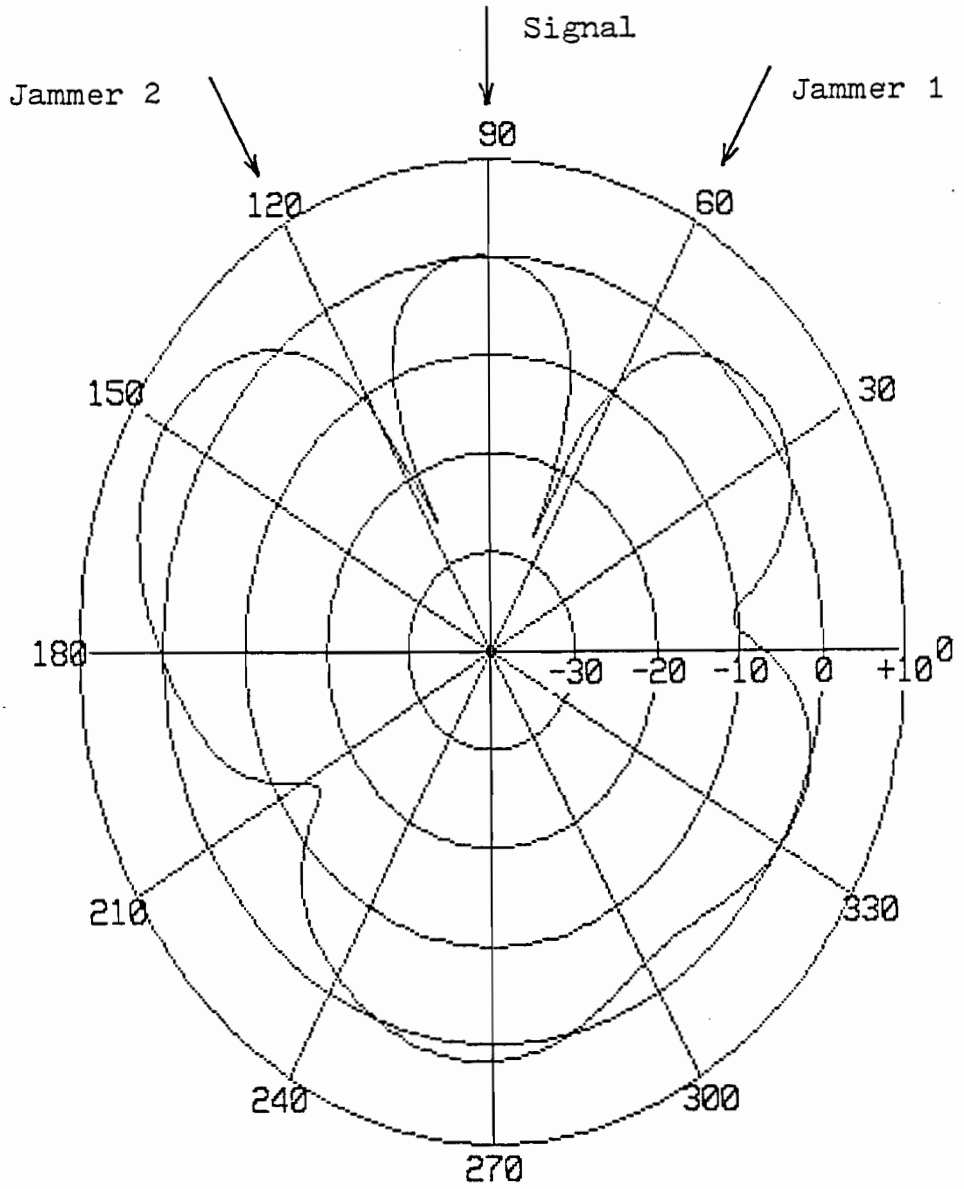


Constant Elevation Cut
Elevation: 30

(Second Largest Eigenvalue)

(Number of Samples Averaged = 64)

Figure T3.9 Eigenvector Sort-Adapted Antenna Plot at $\theta = 30°$

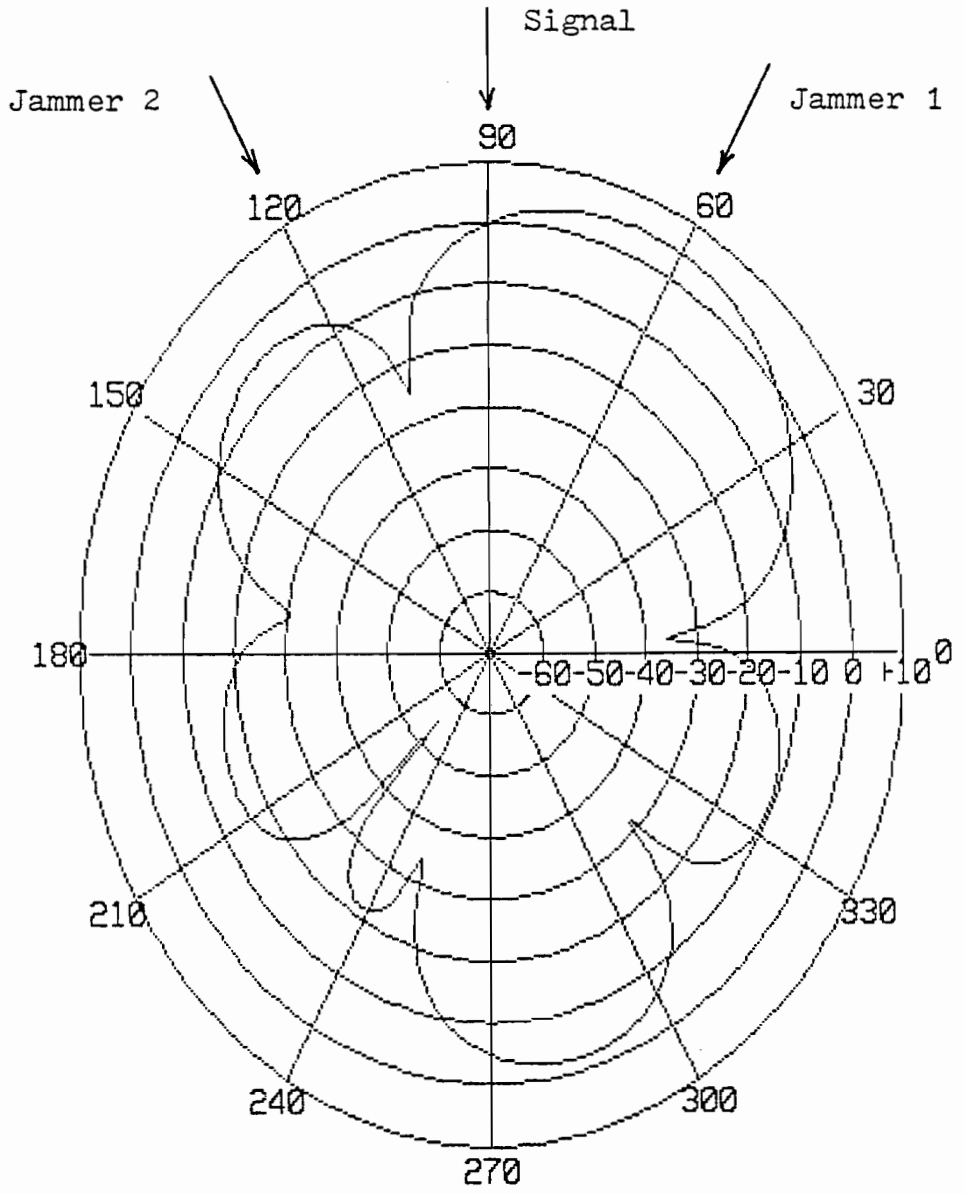


Constant Elevation Cut
Elevation: 30

(Third Largest Eigenvalue)

(Number of Samples Averaged = 64)

Figure T3.10 Eigenvector Sort-Adapted Antenna Plot at $\theta = 30°$

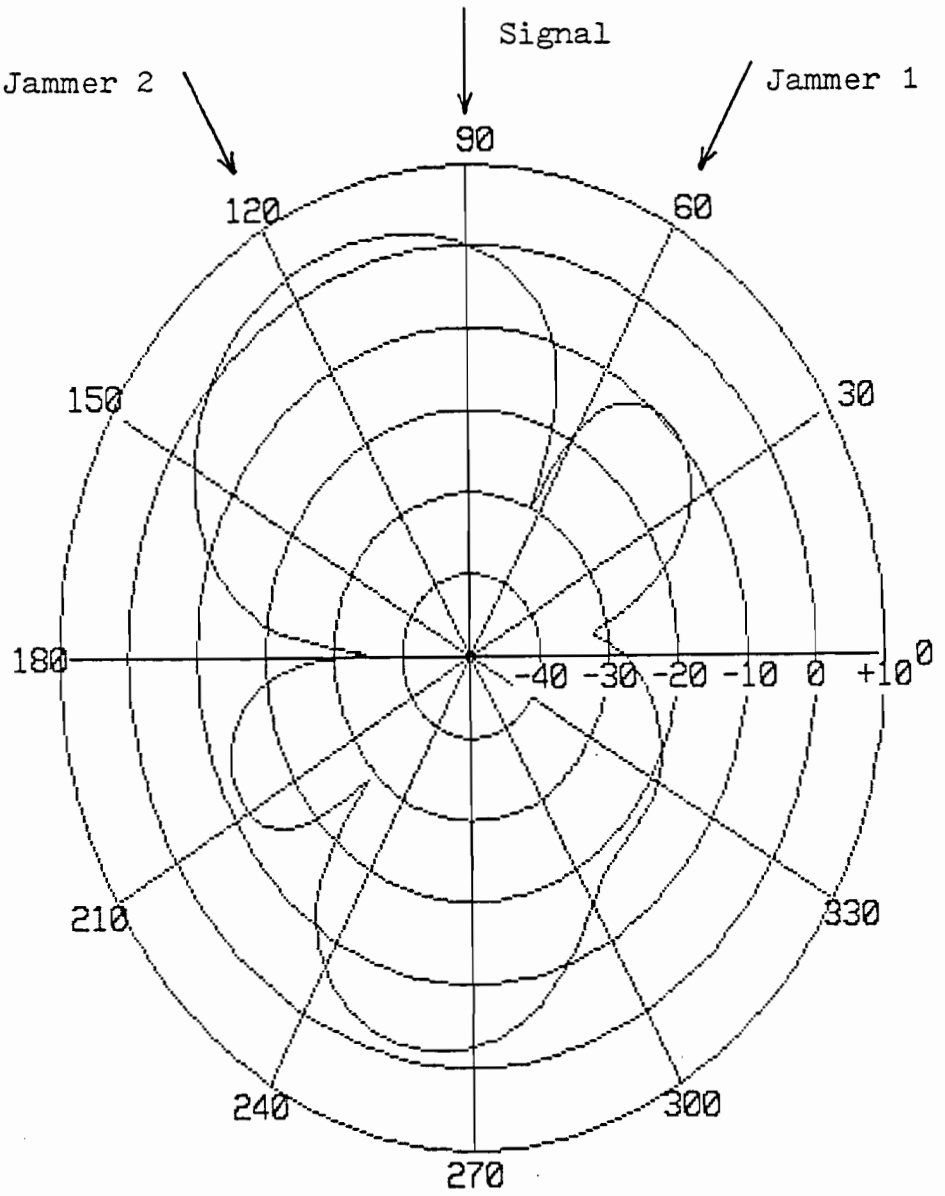


Constant Elevation Cut
Elevation: 30

(Largest Eigenvalue)

(Number of Samples Averaged = 96)

Figure T3.11 Eigenvector Sort-Adapted Antenna Plot at $\theta = 30°$

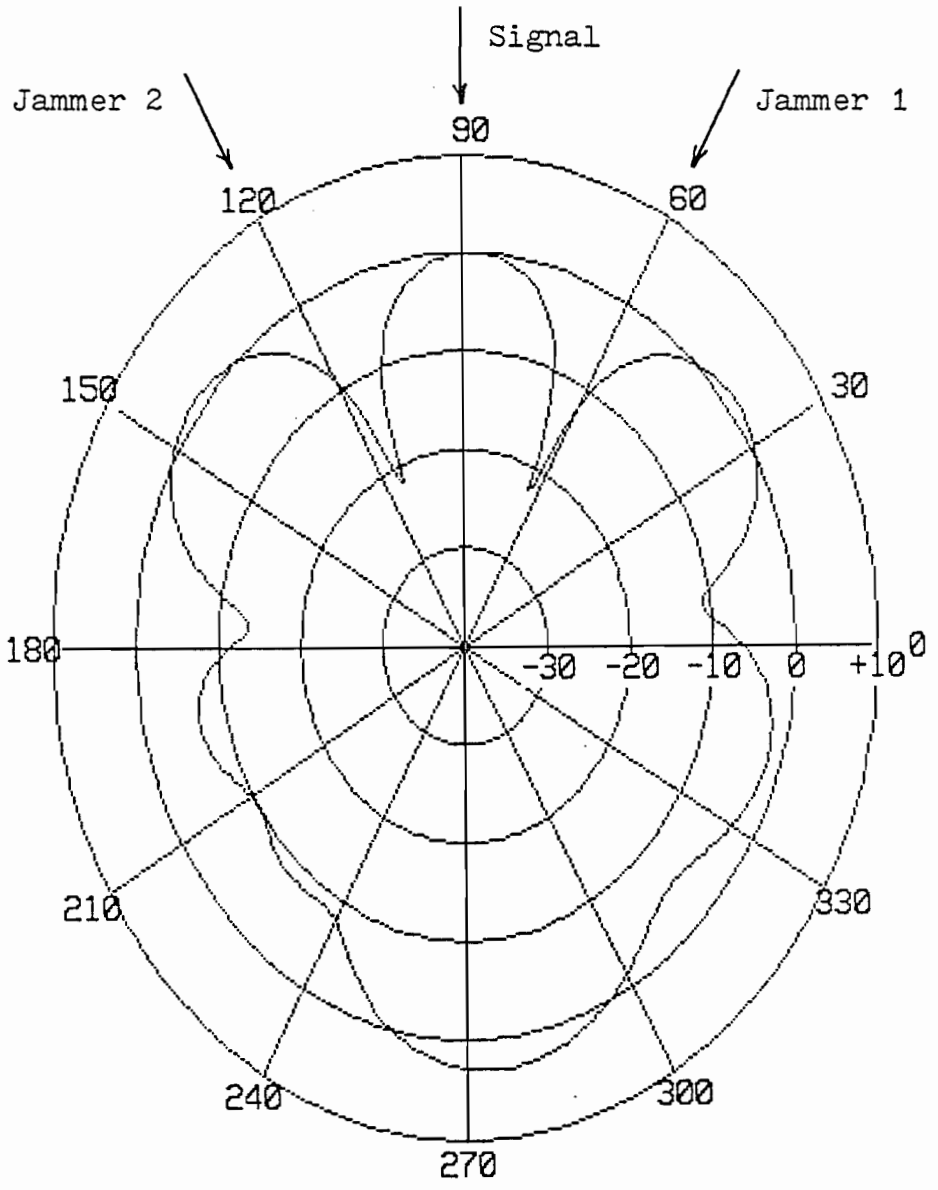


Constant Elevation Cut
Elevation: 30

(Second Largest Eigenvalue)

(Number of Samples Averaged = 96)

Figure T3.12 Eigenvector Sort-Adapted Antenna Plot at $\theta = 30°$



Constant Elevation Cut
Elevation: 30

(Third Largest Eigenvalue)

(Number of Samples Averaged = 96)

Figure T3.13 Eigenvector Sort-Adapted Antenna Plot at $\theta = 30°$

6.4 Test 4: Signal Spacing vs. Number of Averages Required

The performance of adaptive arrays are necessarily dependent on the angular separation of the signals arriving at its sensors. If a jammer is directly aligned with the desired signal, obviously there is no way to apply spatial discrimination to separate the signals. To test the effects of signal separation on the eigenvector sorting algorithm, a series of three sub-tests were conducted.

The first scenario involves signals separated by 60 degrees as shown in the top portion of table 6.4. The signals are moved into closer proximity in the remaining sub-tests to 30, and finally to 15 degrees. These scenarios are also shown in table 6.4. To assess the effect of these changes in signal spacing, the number of samples needed (for an estimation of the autocorrelation matrix) to cause desired signal separation is determined. These are again in multiples of 16, so, in effect, the number of information bits is determined.

It has been noticed that the nulling effect usually occurs quite abruptly (from one bit to another) and the plots represent the state of the system at the first occurrence of the effects. At 60 degree signal spacing, represented by Figures T4.1 - T4.10, the desired communicator is separated within 32 samples. We do notice that even at this large signal spacing, there is again a relatively poor jammer separation. Figures

SIGNAL	AZIMUTHAL ANGLE	ELEVATION ANGLE	POWER
COMMUNICATOR	30.0	30.0	0 dB
JAMMER 1	90.0	30.0	20 dB
JAMMER 2	150.0	30.0	20 dB

Number of Samples Averaged = 32

SIGNAL	AZIMUTHAL ANGLE	ELEVATION ANGLE	POWER
COMMUNICATOR	60.0	30.0	0 dB
JAMMER 1	90.0	30.0	20 dB
JAMMER 2	120.0	30.0	20 dB

Number of Samples Averaged = 48

SIGNAL	AZIMUTHAL ANGLE	ELEVATION ANGLE	POWER
COMMUNICATOR	75.0	30.0	0 dB
JAMMER 1	90.0	30.0	20 dB
JAMMER 2	105.0	30.0	20 dB

Number of Samples Averaged = 96

Table 6.4 Signal and Averaging Characteristics for Test 4

T4.11 - T4.20 show the antenna patterns at the signal spacing of 30 degrees. Again the signal is given an advantage over the jammers of approximately 40 dB, but it is also true that the jammer separation has become worse than the previous test even though the number of averaged samples has increased to 48.

Finally, the last ten plots (T4.21 - T4.30) deal with the signal separation of 15 degrees. The number of samples required to form the autocorrelation matrix such that nulling behavior begins is 96. In this case it can be seen that, again, relatively poor jammer separation is achieved using the largest two eigenvalues. Also, it is apparent that, using the third largest eigenvalue, the desired signal is chosen but the nulls are not as deep as in the previous cases.

We see from this test that the number of samples required does indeed increase as the signals are moved into closer proximity to one another. It is also apparent that jammer separation comes about much more slowly in the cases of crowded signals. This is to be expected since, as mentioned before, if the signals arrived at the array from virtually the same direction, it would be impossible to separate the signals at all.

Summary of Plots for Test 4

(Signal Spacing = 60)
(Number of Samples Averaged = 32)

Fig. T4.1: Unadapted Antenna Plot at $\theta = 30°$

Fig. T4.2: Adapted Plot at $\theta = 30°$ (Largest Eigenvalue)

Fig. T4.3: Adapted Plot at $\theta = 30°$ (Second Largest Eigenvalue)

Fig. T4.4: Adapted Plot at $\theta = 30°$ (Third Largest Eigenvalue)

Fig. T4.5: Unadapted Antenna Plot at $\phi = 30°$

Fig. T4.6: Adapted Plot at $\phi = 30°$ (Third Largest Eigenvalue)

Fig. T4.7: Unadapted Antenna Plot at $\phi = 90°$

Fig. T4.8: Adapted Plot at $\phi = 90°$ (Third Largest Eigenvalue)

Fig. T4.9: Unadapted Antenna Plot at $\phi = 150°$

Fig. T4.10: Adapted Plot at $\phi = 150°$ (Third Largest Eigenvalue)

(Signal Spacing = 30)
(Number of Samples Averaged = 48)

Fig. T4.11: Unadapted Antenna Plot at $\theta = 30°$

Fig. T4.12: Adapted Plot at $\theta = 30°$ (Largest Eigenvalue)

Fig. T4.13: Adapted Plot at $\theta = 30°$ (Second Largest Eigenvalue)

Fig. T4.14: Adapted Plot at $\theta = 30°$ (Third Largest Eigenvalue)

Fig. T4.15: Unadapted Antenna Plot at $\phi = 60°$

Fig. T4.16: Adapted Plot at $\phi = 60°$ (Third Largest Eigenvalue)

Fig. T4.17: Unadapted Antenna Plot at $\phi = 90°$

Fig. T4.18: Adapted Plot at $\phi = 90°$ (Third Largest Eigenvalue)

Fig. T4.19: Unadapted Antenna Plot at $\phi = 120°$

Fig. T4.20: Adapted Plot at $\phi = 120°$ (Third Largest Eigenvalue)

(Signal Spacing = 15)
(Number of Samples Averaged = 96)

Fig. T4.21: Unadapted Antenna Plot at $\theta = 30°$

Fig. T4.22: Adapted Plot at $\theta = 30°$ (Largest Eigenvalue)

Fig. T4.23: Adapted Plot at $\theta = 30°$ (Second Largest Eigenvalue)

Fig. T4.24: Adapted Plot at $\theta = 30°$ (Third Largest Eigenvalue)

Fig. T4.25: Unadapted Antenna Plot at $\phi = 75°$

Fig. T4.26: Adapted Plot at $\phi = 75°$ (Third Largest Eigenvalue)

Fig. T4.27: Unadapted Antenna Plot at $\phi = 90°$

Fig. T4.28: Adapted Plot at $\phi = 90°$ (Third Largest Eigenvalue)

Fig. T4.29: Unadapted Antenna Plot at $\phi = 105°$

Fig. T4.30: Adapted Plot at $\phi = 105°$ (Third Largest Eigenvalue)

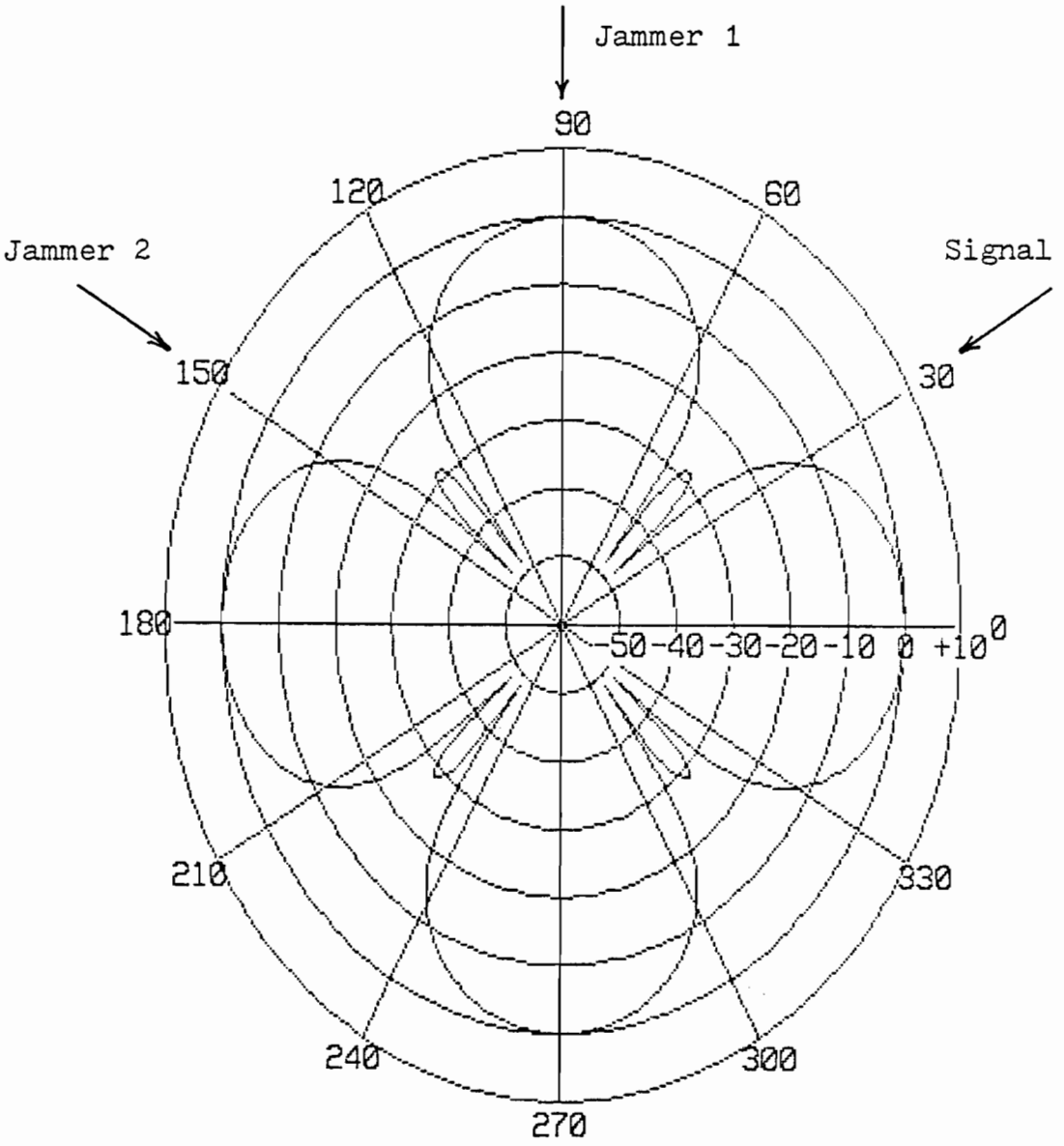
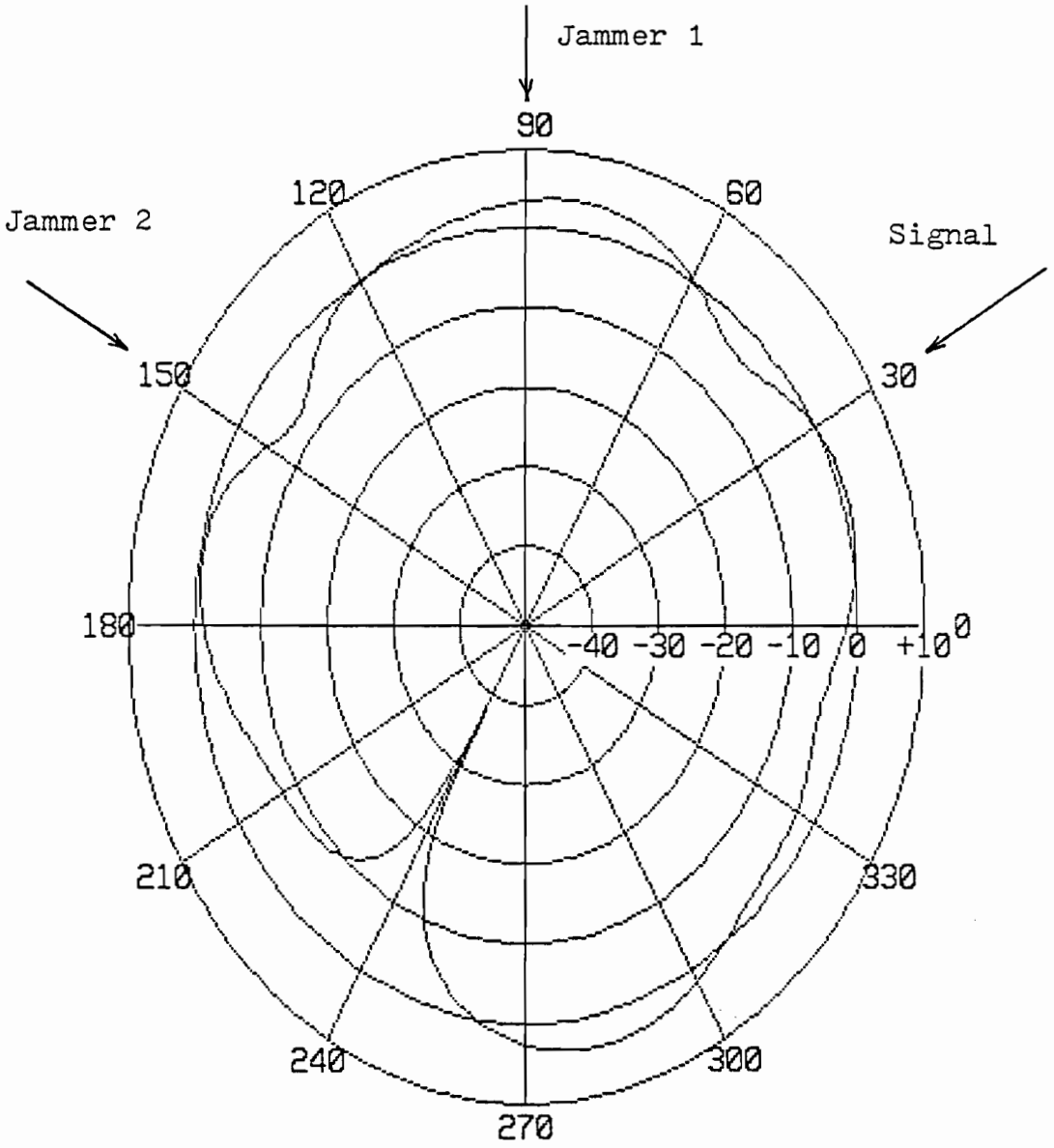


Figure T4.1 Unadapted Antenna Plot for Eigenvector Algorithm at $\theta = 30°$

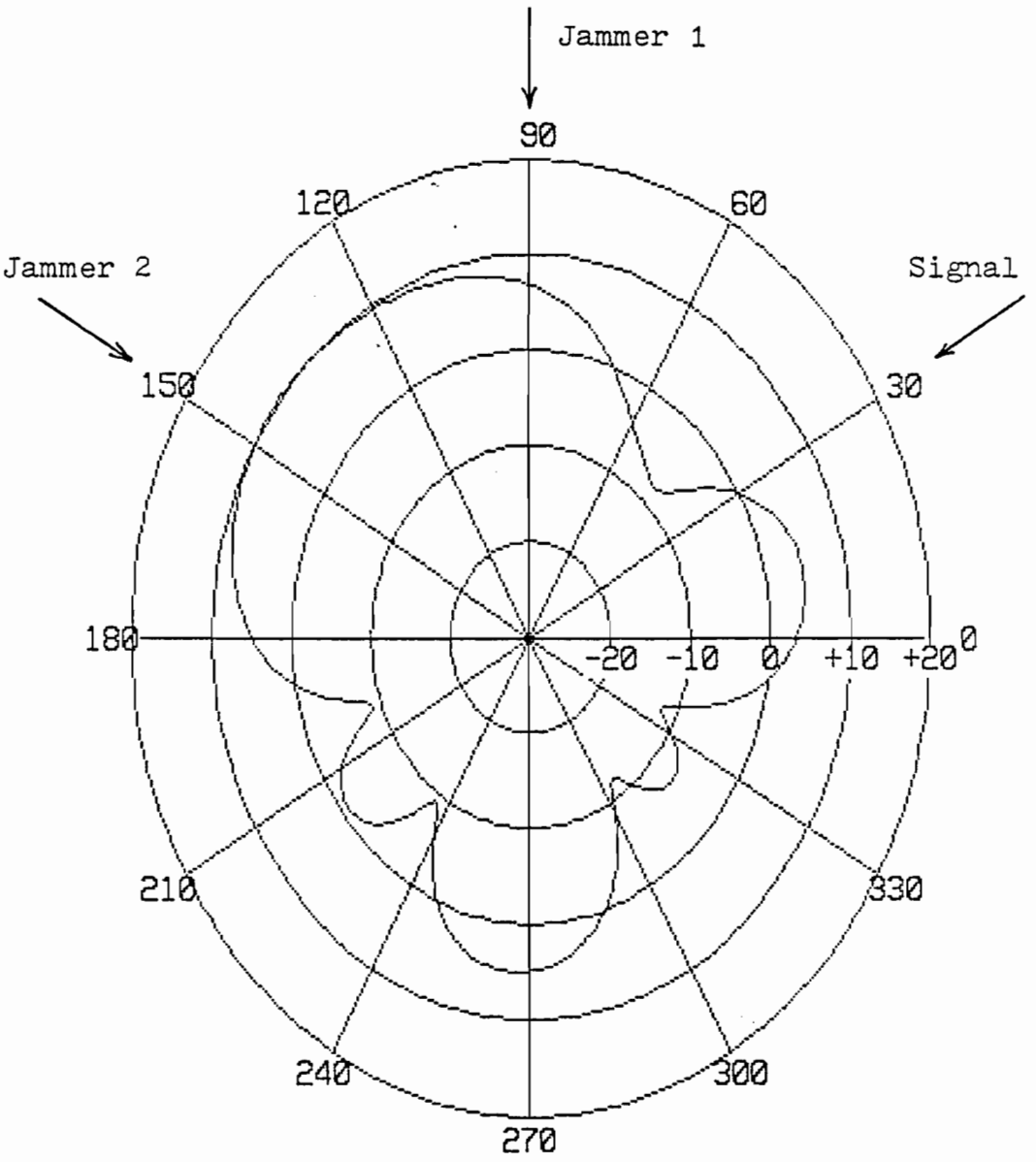


Constant Elevation Cut
Elevation: 30

(Largest Eigenvalue)

(Number of Samples Averaged = 32)

Figure T4.2 Eigenvector Sort-Adapted Antenna Plot at $\theta = 30°$

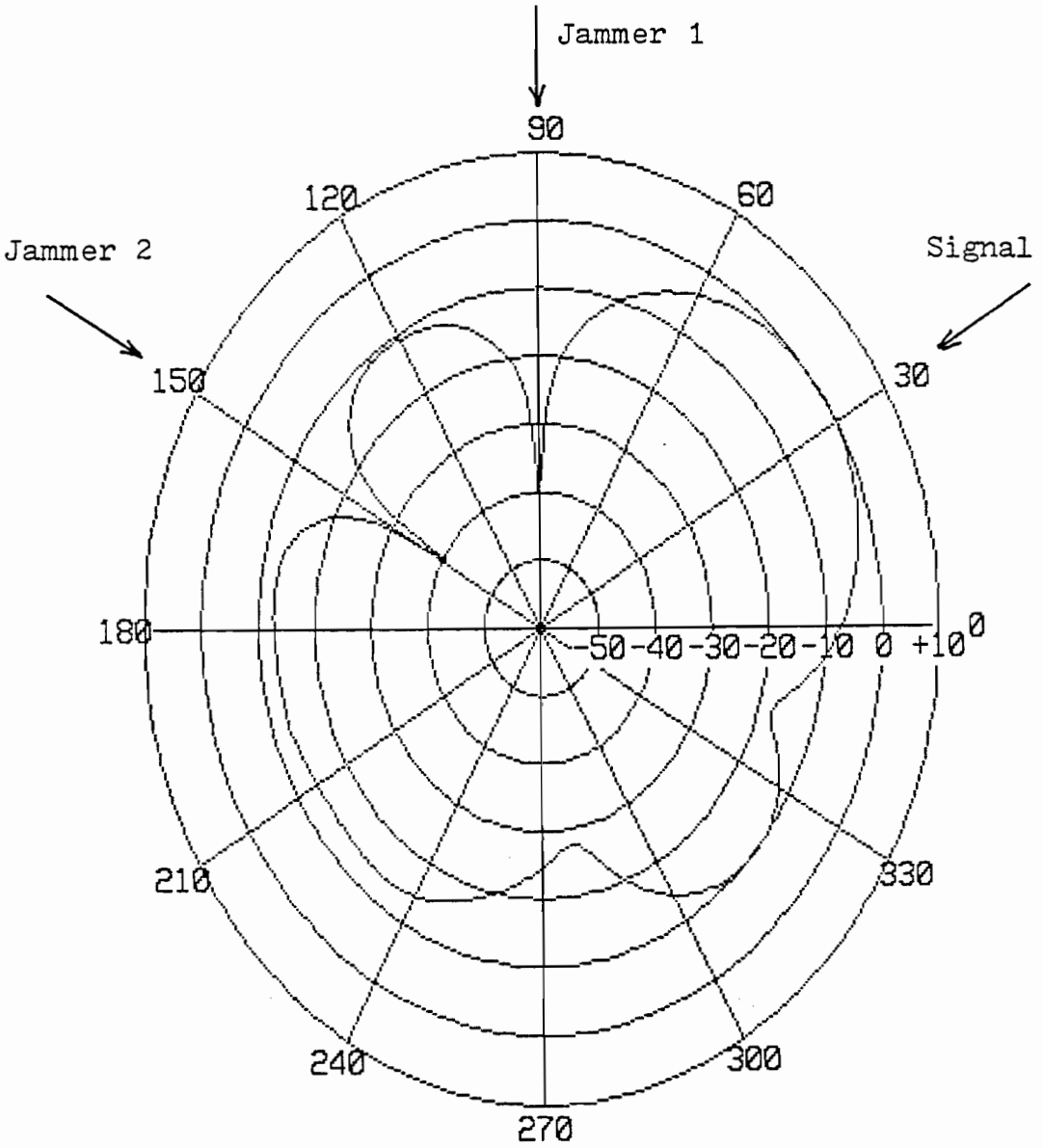


Constant Elevation Cut
Elevation: 30

(Second Largest Eigenvalue)

(Number of Samples Averaged = 32)

Figure T4.3 Eigenvector Sort-Adapted Antenna Plot at $\theta = 30°$



Constant Elevation Cut
Elevation: 30

(Third Largest Eigenvalue)

(Number of Samples Averaged = 32)

Figure T4.4 Eigenvector Sort-Adapted Antenna Plot at $\theta = 30°$

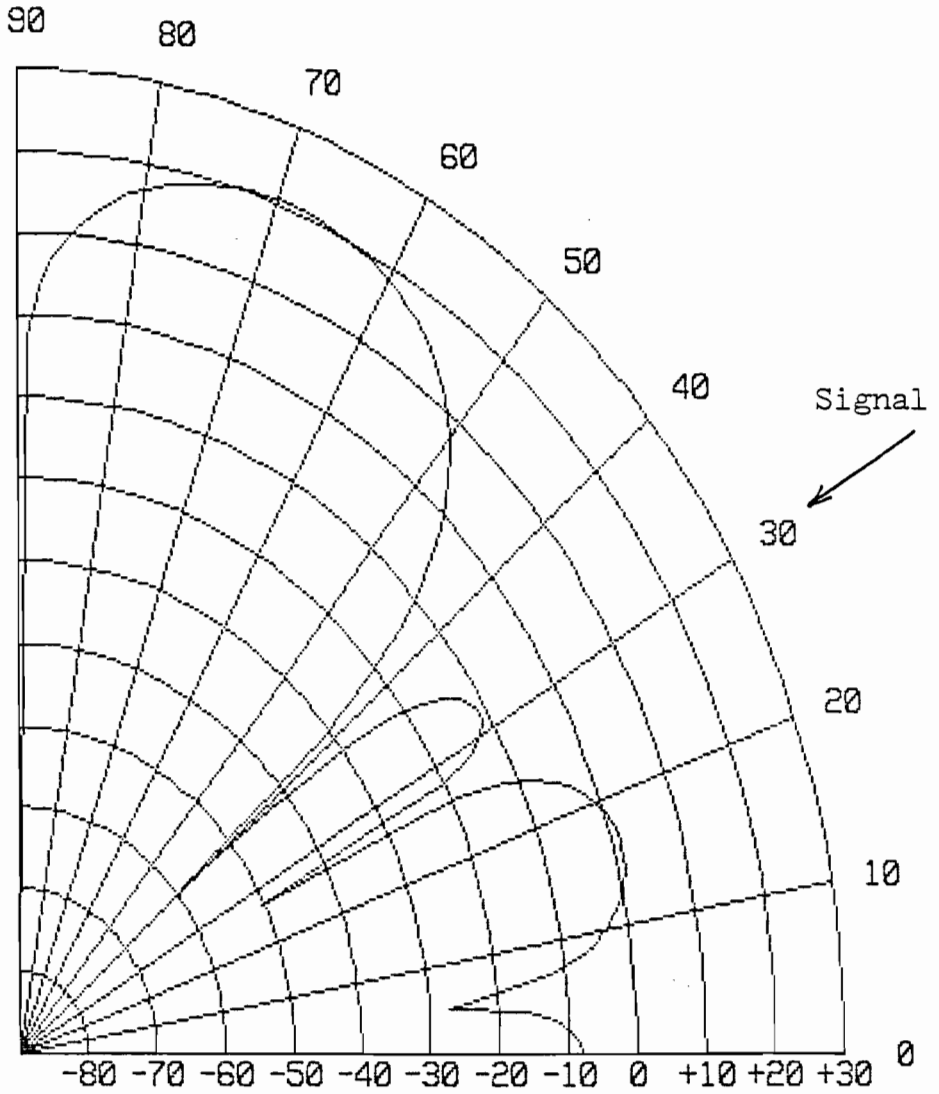
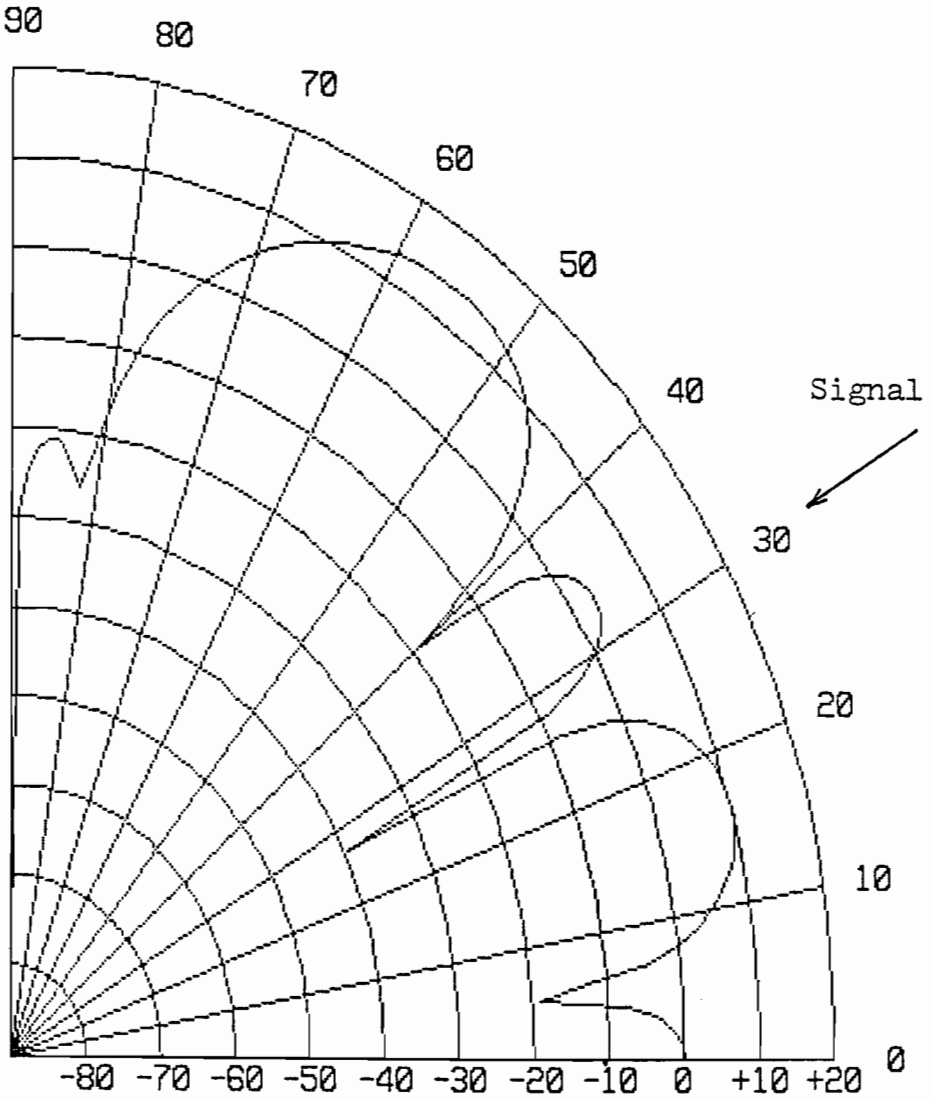


Figure T4.5 Unadapted Antenna Plot for Eigenvector Algorithm at ϕ = 30°



Constant Azimuthal Cut
Azimuth: 30

(Third Largest Eigenvalue)

(Number of Samples Averaged = 32)

Figure T4.6 Eigenvector Sort-Adapted Antenna Plot at $\phi = 90°$

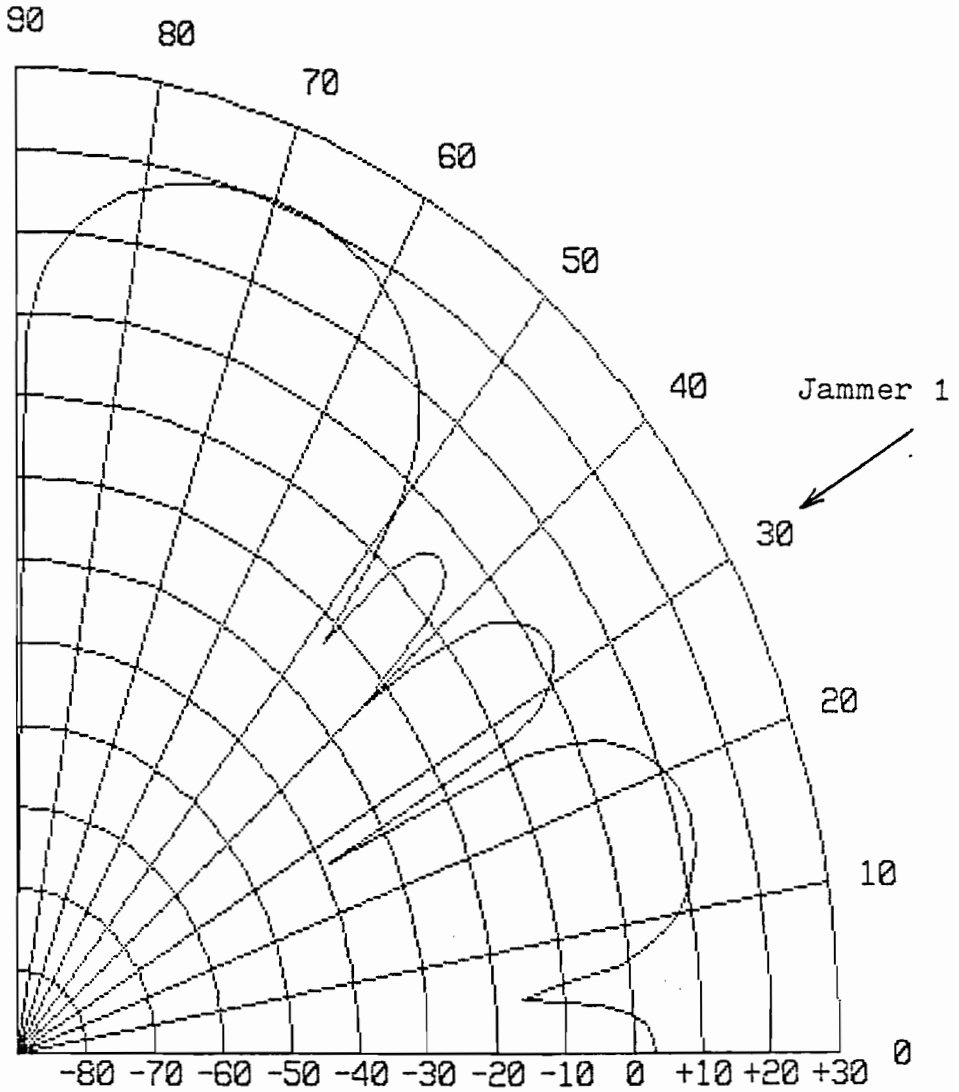
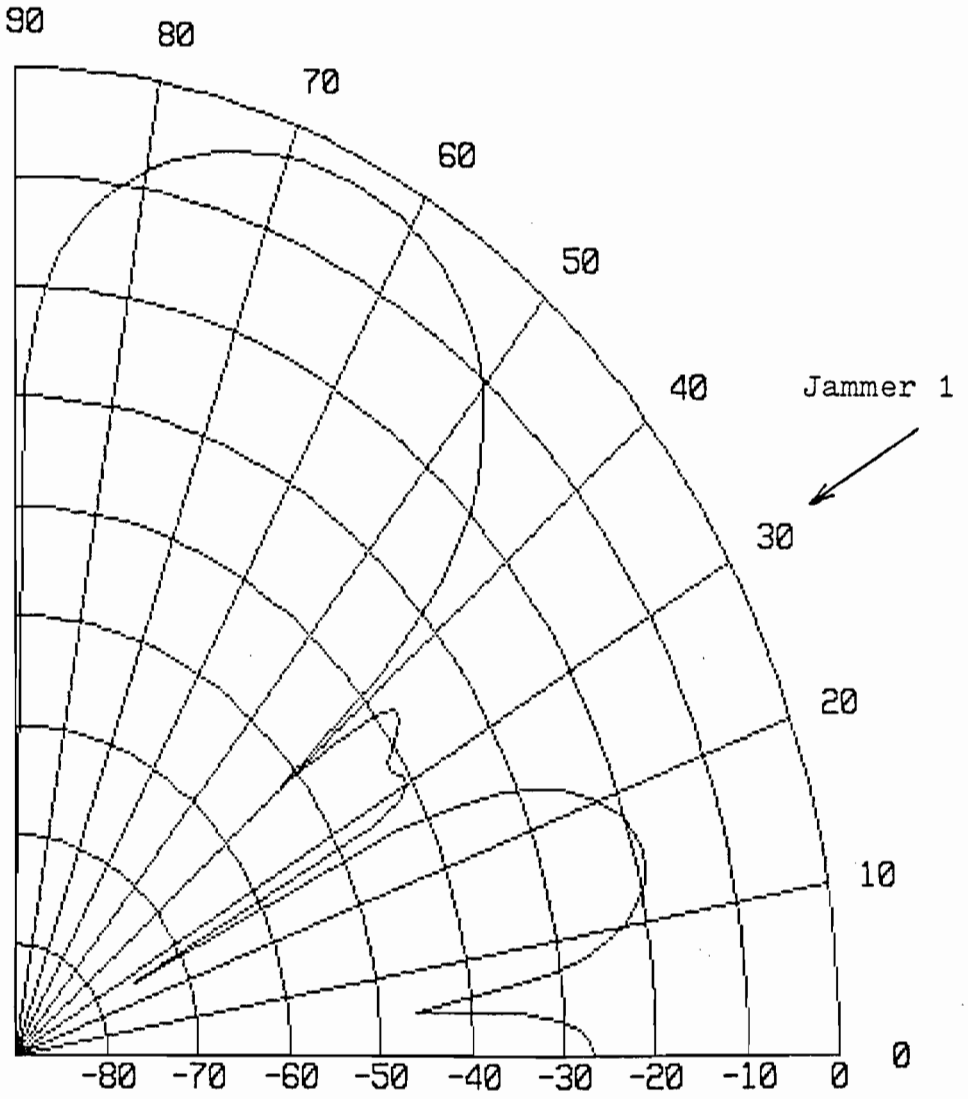


Figure T4.7 Unadapted Antenna Plot for Eigenvector Algorithm at ϕ = 90°



Constant Azimuthal Cut
Azimuth: 90

(Third Largest Eigenvalue)

(Number of Samples Averaged = 32)

Figure T4.8 Eigenvector Sort-Adapted Antenna Plot at $\phi = 90°$

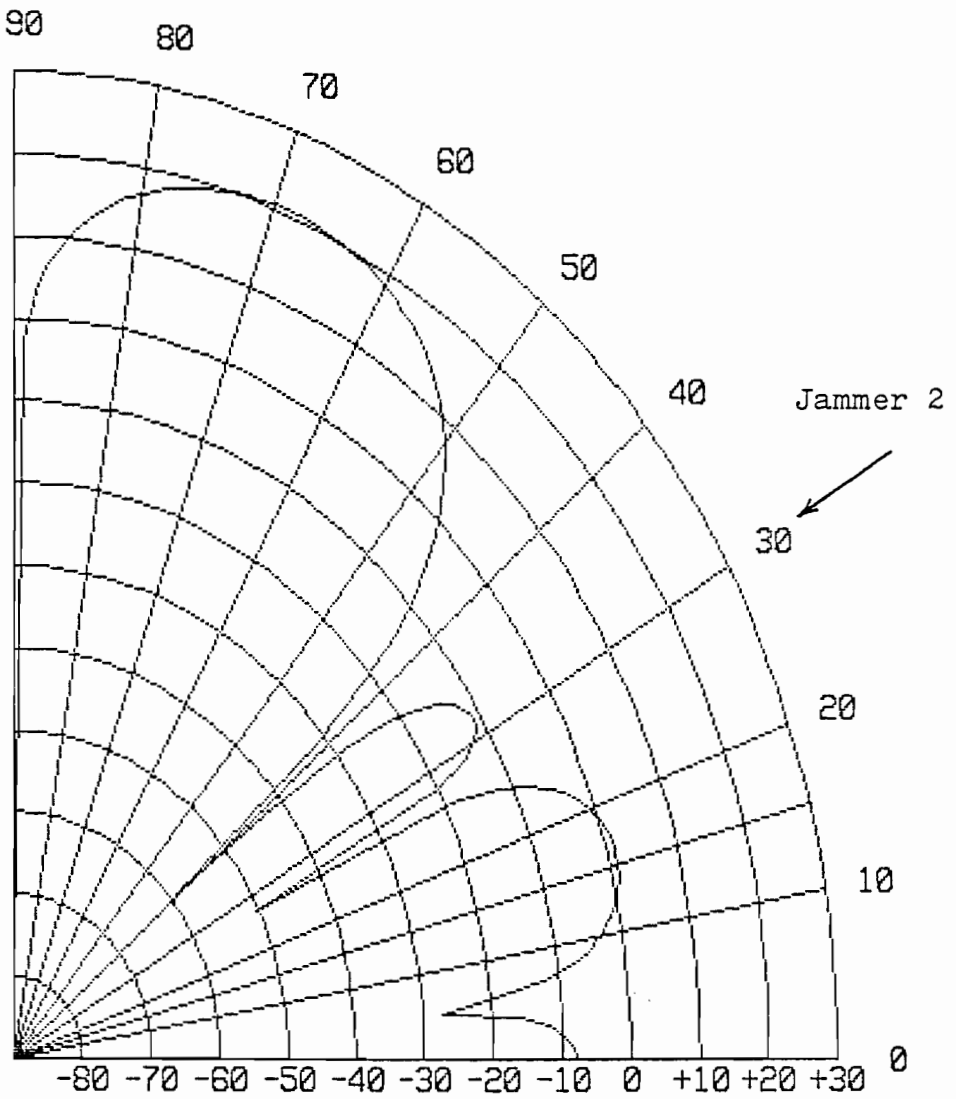
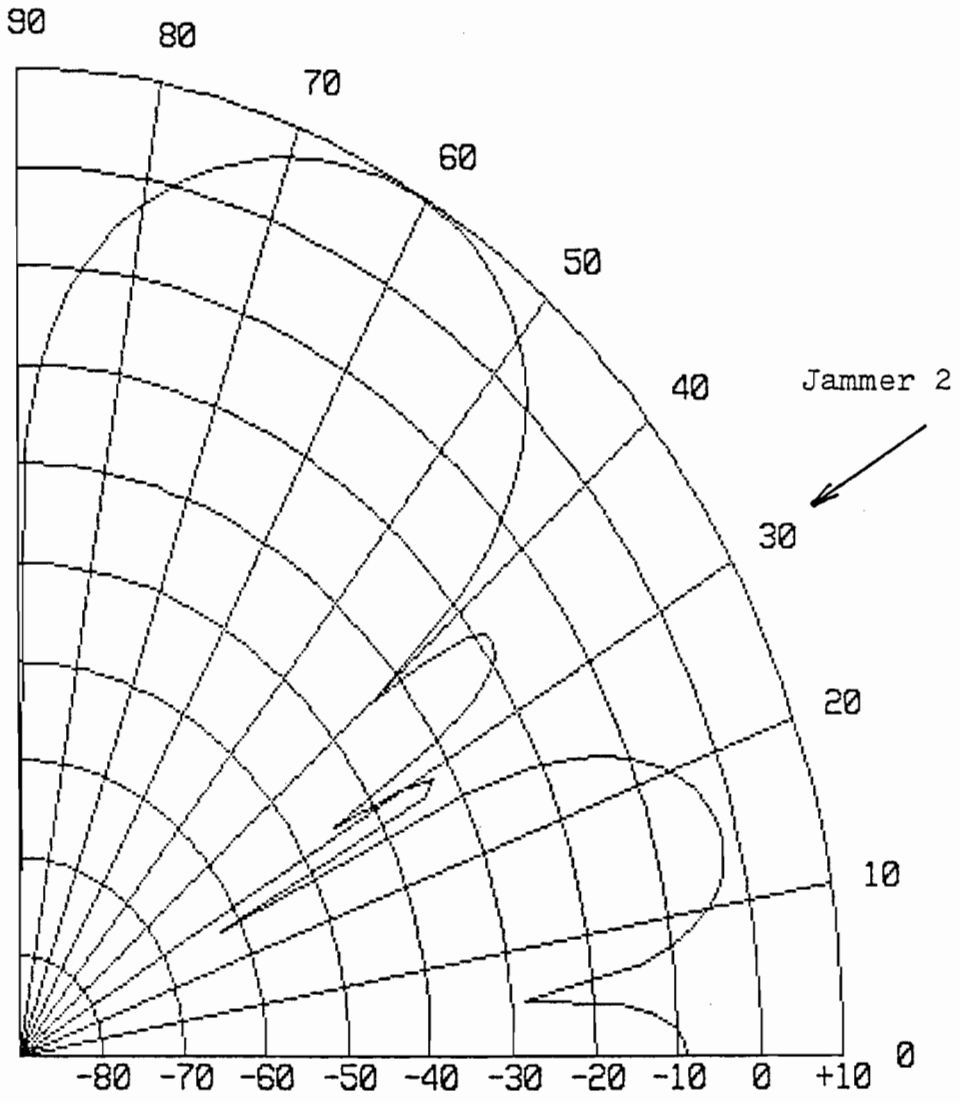


Figure T4.9 Unadapted Antenna Plot for Eigenvector Algorithm at ϕ = 150°



Constant Azimuthal Cut
Azimuth: 150

(Third Largest Eigenvalue)

(Number of Samples Averaged = 32)

Figure T4.10 Eigenvector Sort-Adapted Antenna Plot at ϕ = 150°

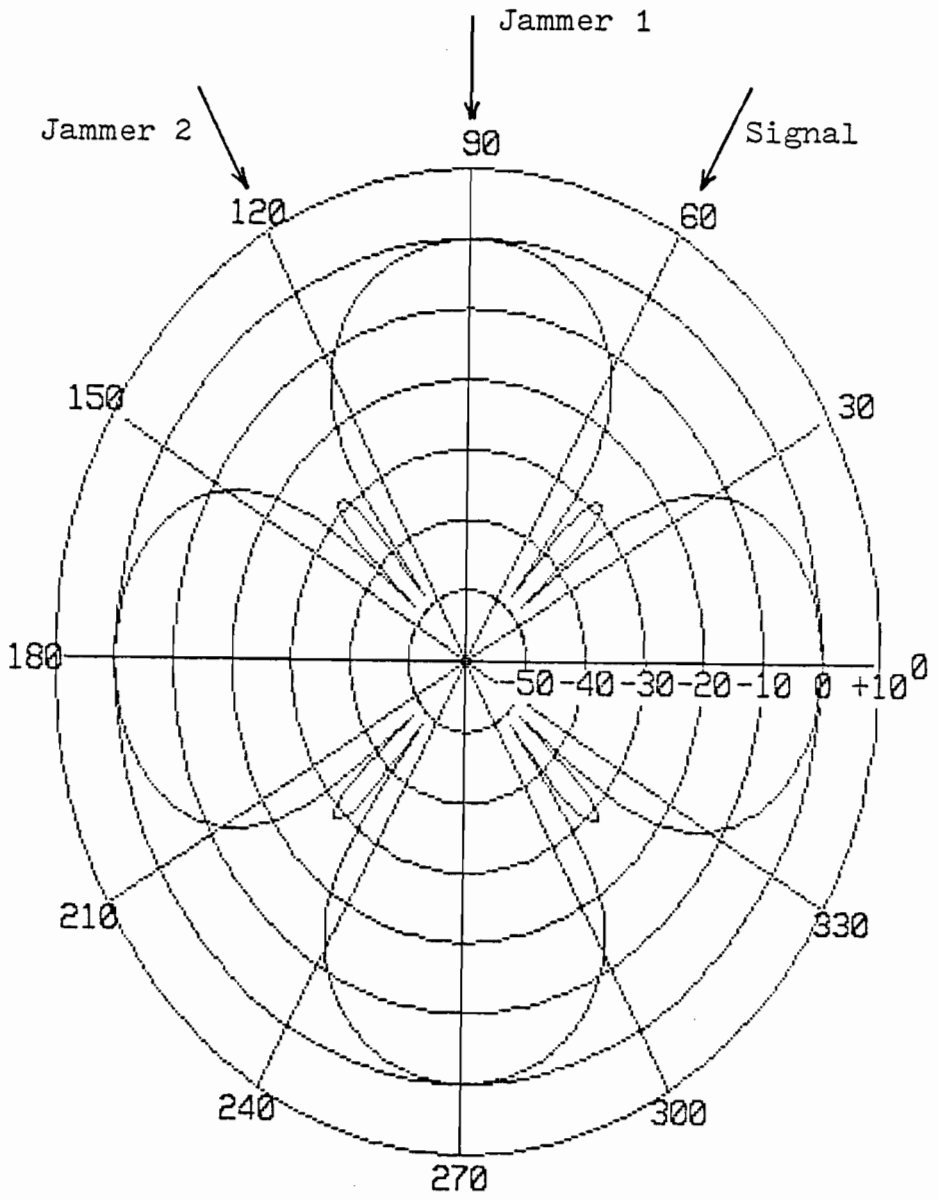
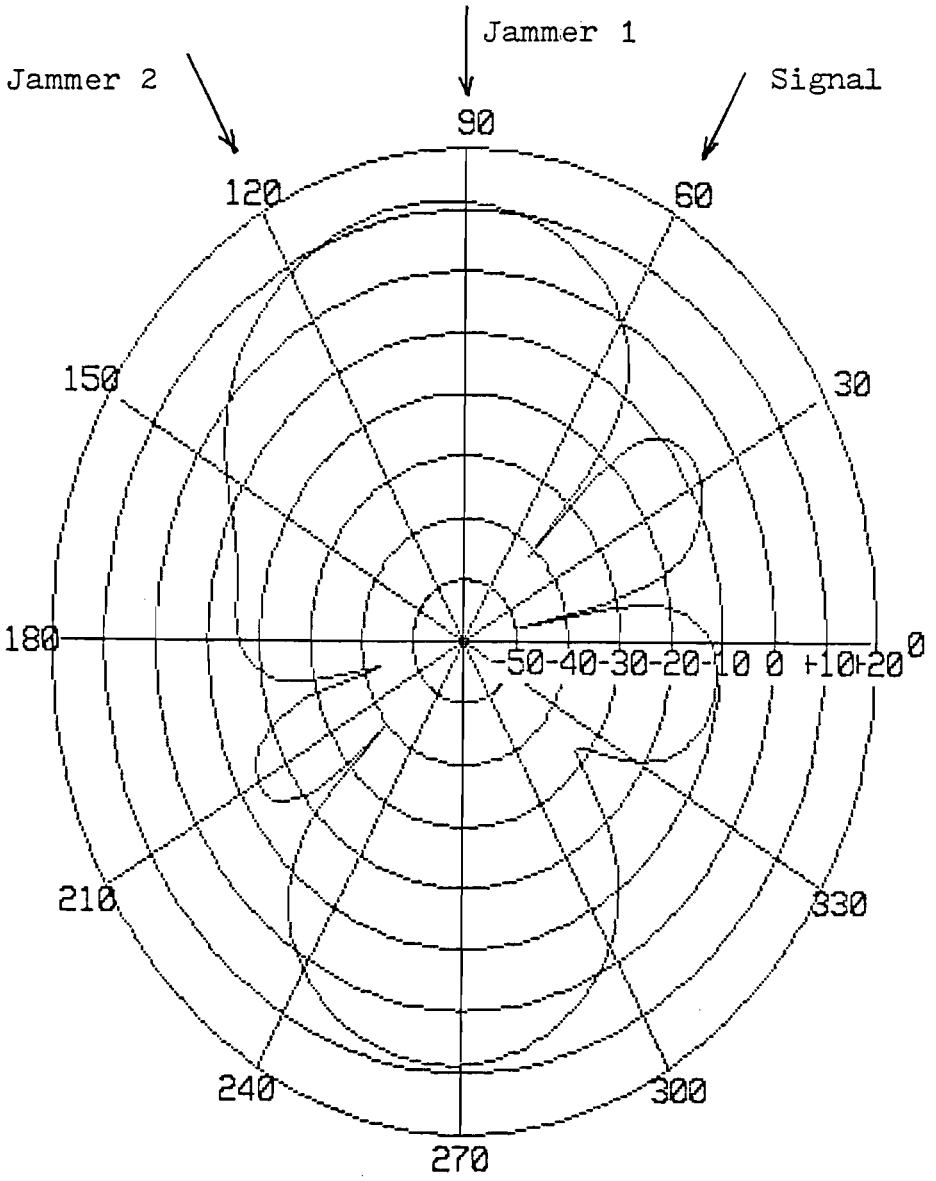


Figure T4.11 Unadapted Antenna Plot for Eigenvector Algorithm at $\theta = 30°$

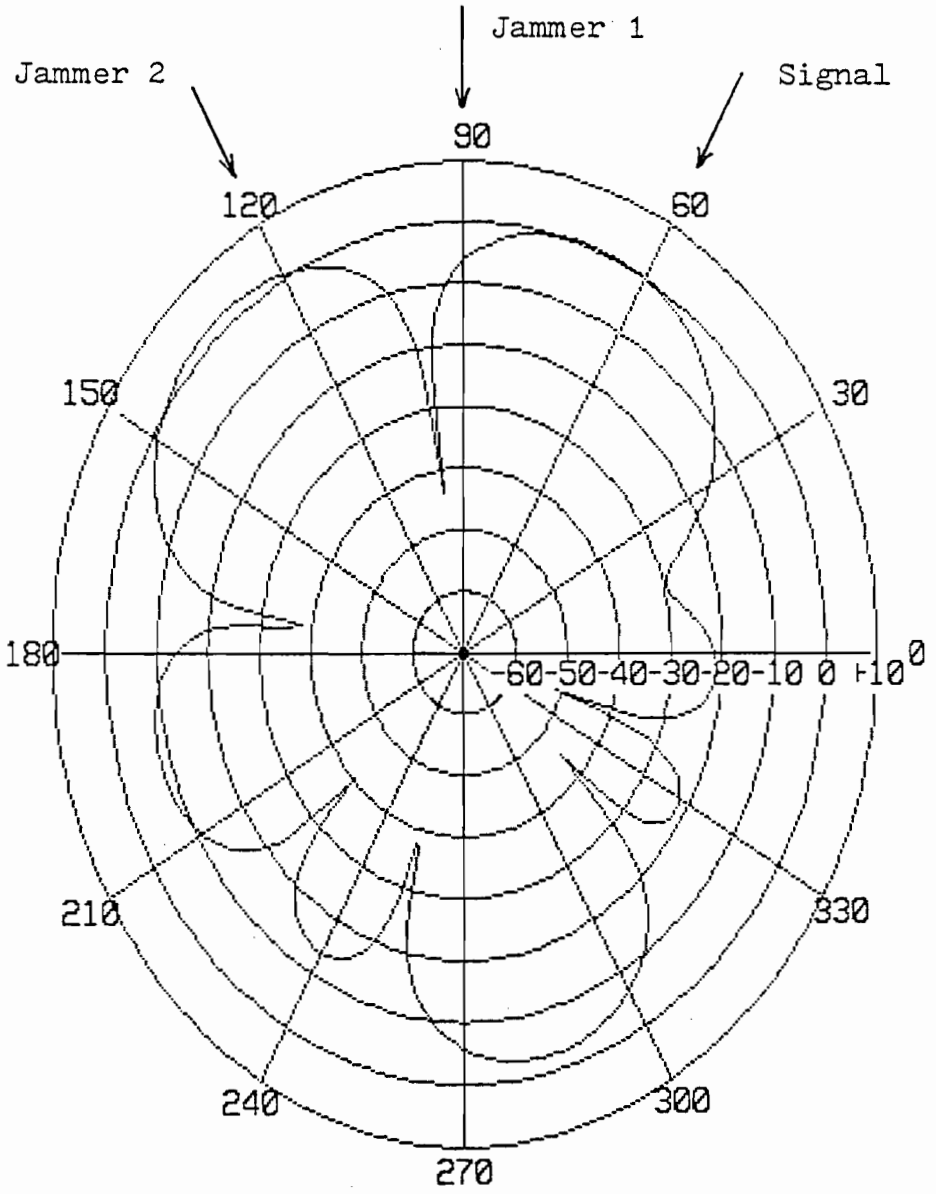


Constant Elevation Cut
Elevation: 30

(Largest Eigenvalue)

(Number of Samples Averaged = 48)

Figure T4.12 Eigenvector Sort-Adapted Antenna Plot at $\theta = 30°$

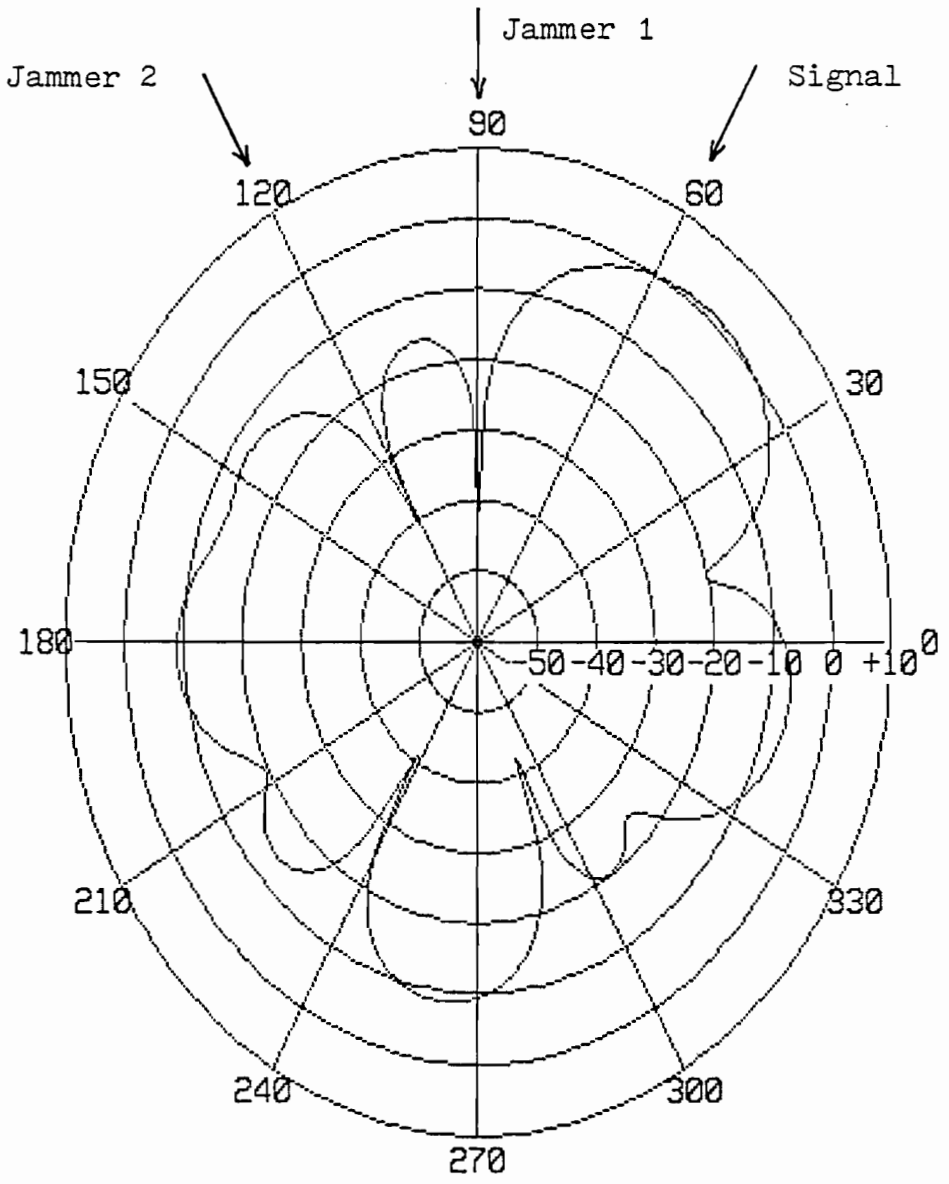


Constant Elevation Cut
Elevation: 30

(Second Largest Eigenvalue)

(Number of Samples Averaged = 48)

Figure T4.13 Eigenvector Sort-Adapted Antenna Plot at $\theta = 30°$



Constant Elevation Cut
Elevation: 30

(Third Largest Eigenvalue)

(Number of Samples Averaged = 48)

Figure T4.14 Eigenvector Sort-Adapted Antenna Plot at $\theta = 30°$

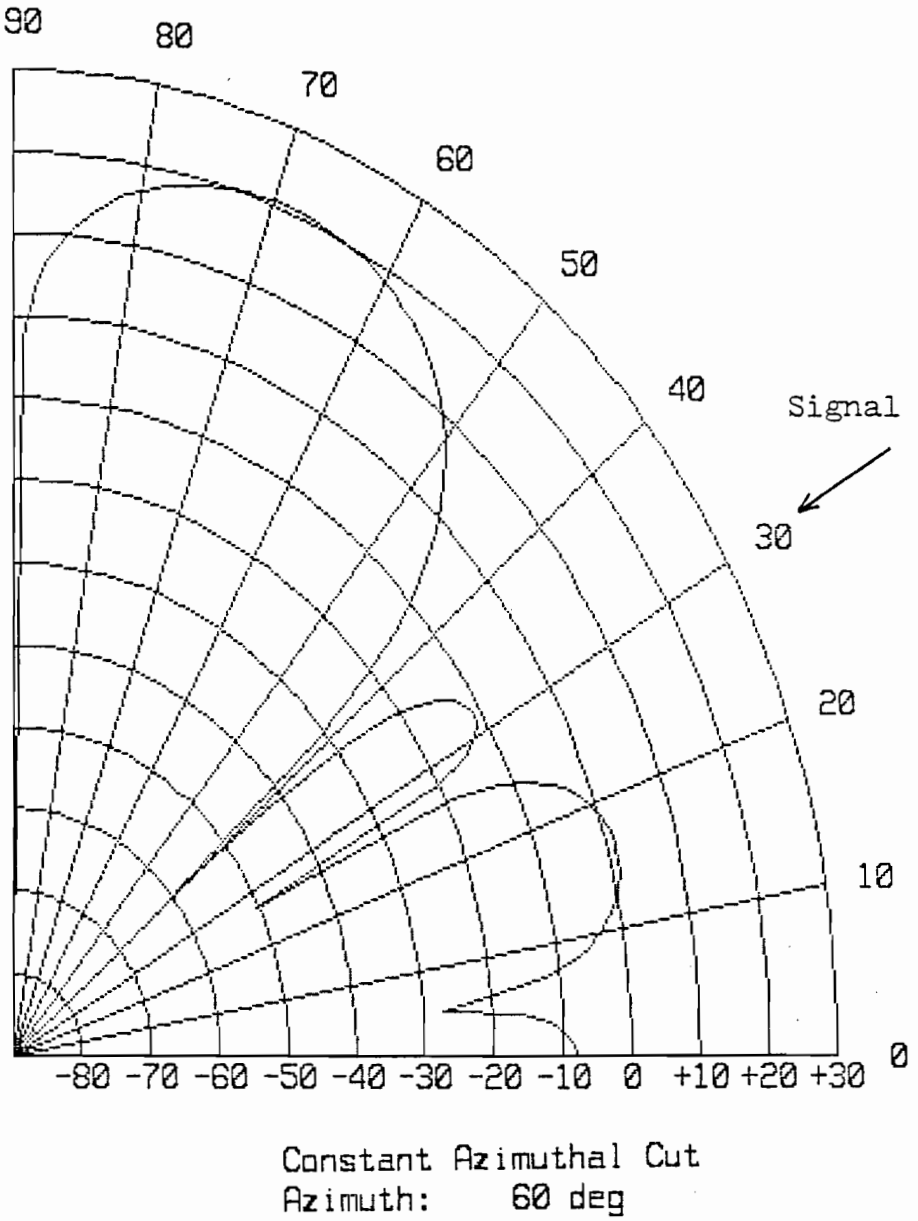
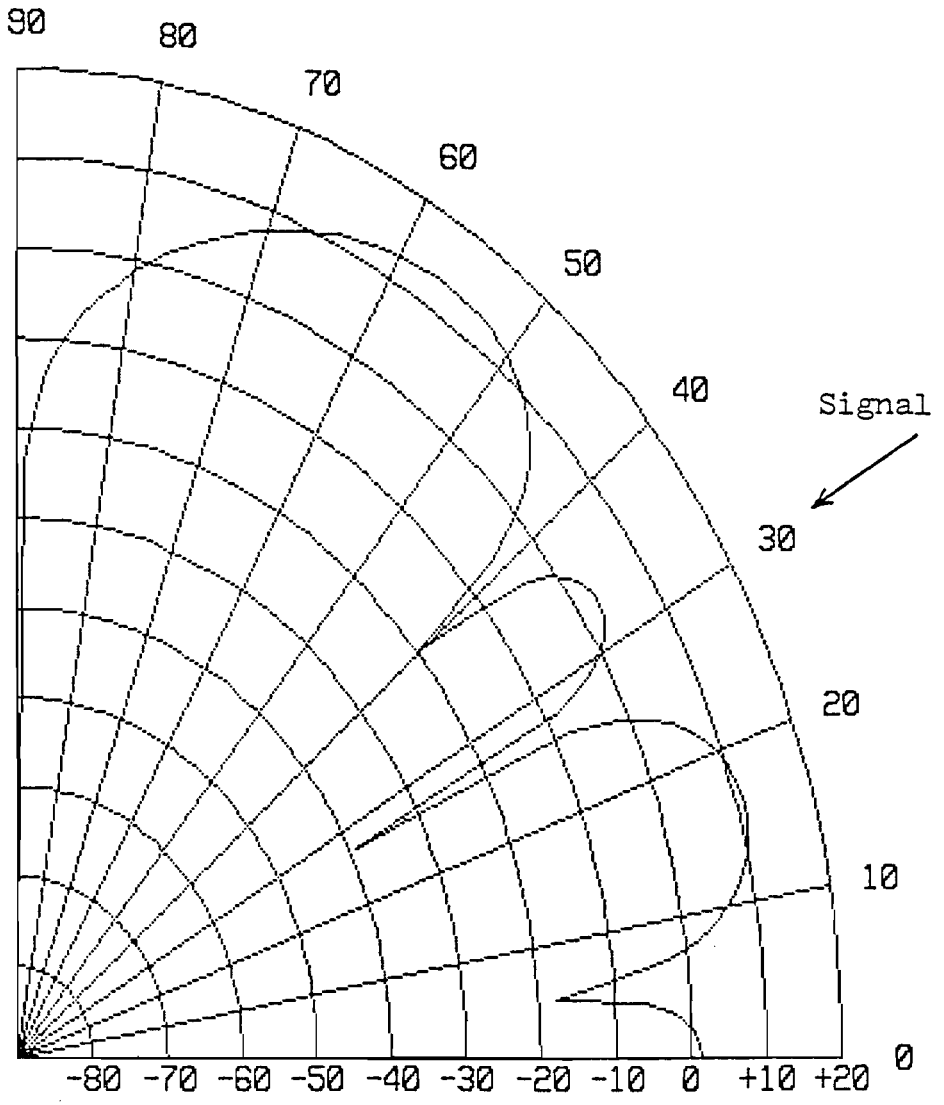


Figure T4.15 Unadapted Antenna Plot for Eigenvector Algorithm at ϕ = 60°



Constant Azimuthal Cut
Azimuth: 60

(Third Largest Eigenvalue)

(Number of Samples Averaged = 48)

Figure T4.16 Eigenvector Sort-Adapted Antenna Plot at $\phi = 60°$

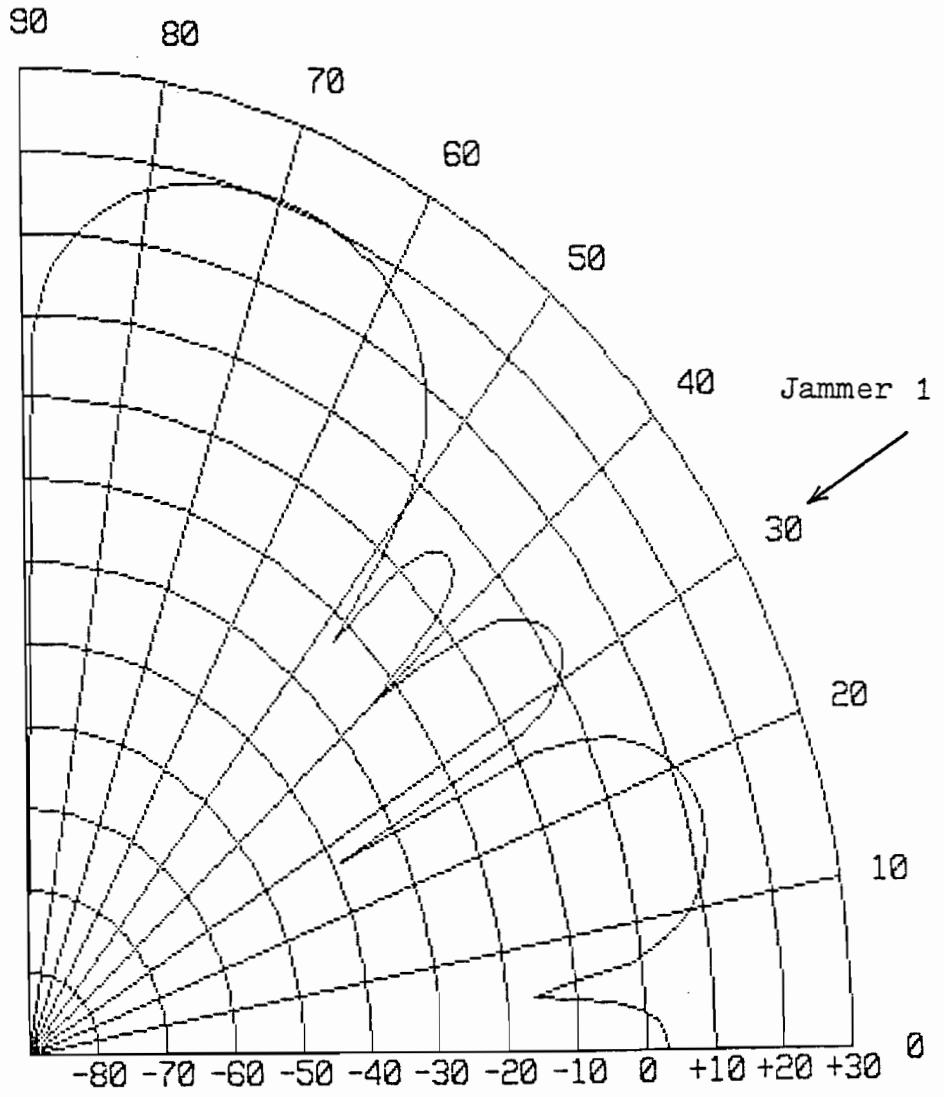
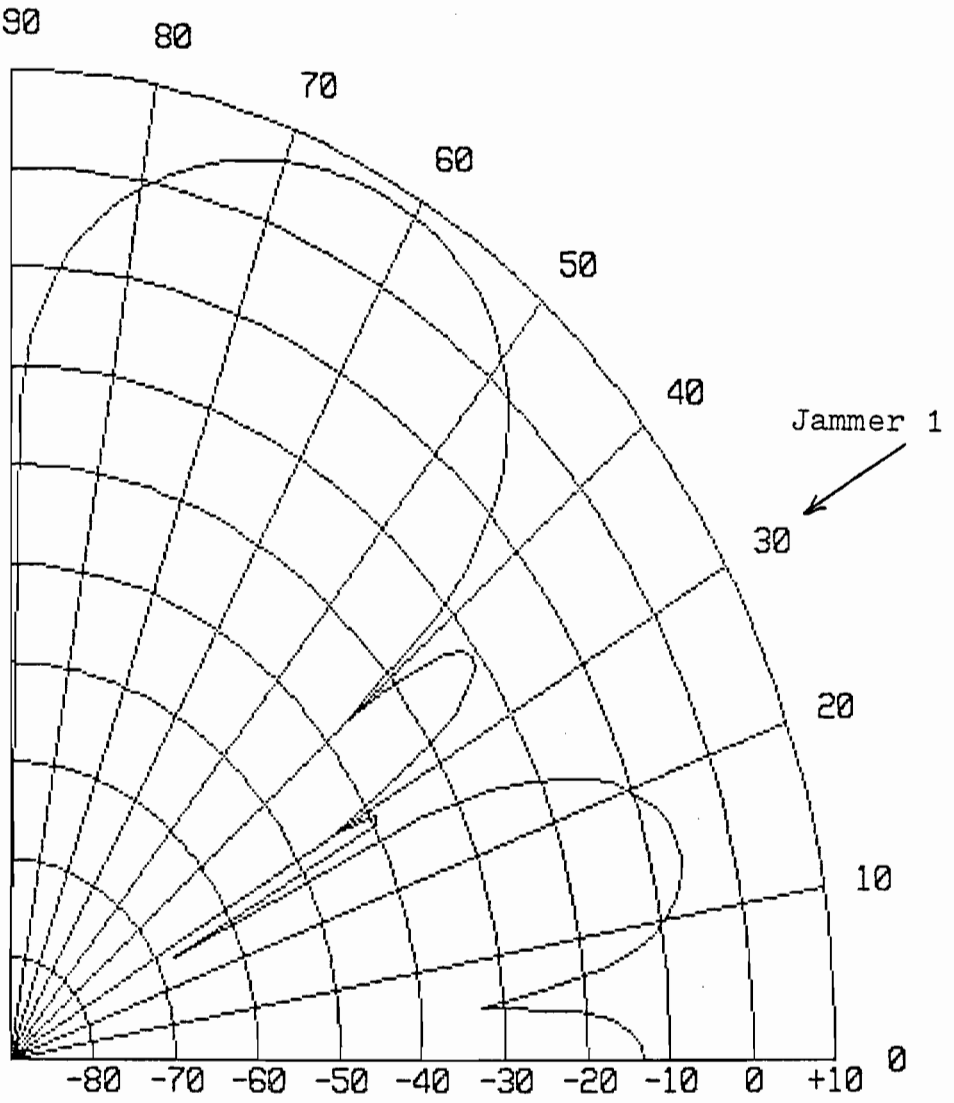


Figure T4.17 Unadapted Antenna Plot for Eigenvector Algorithm at ϕ = 90°



Constant Azimuthal Cut
Azimuth: 90

(Third Largest Eigenvalue)

(Number of Samples Averaged = 48)

Figure T4.18 Eigenvector Sort-Adapted Antenna Plot at $\phi = 90°$

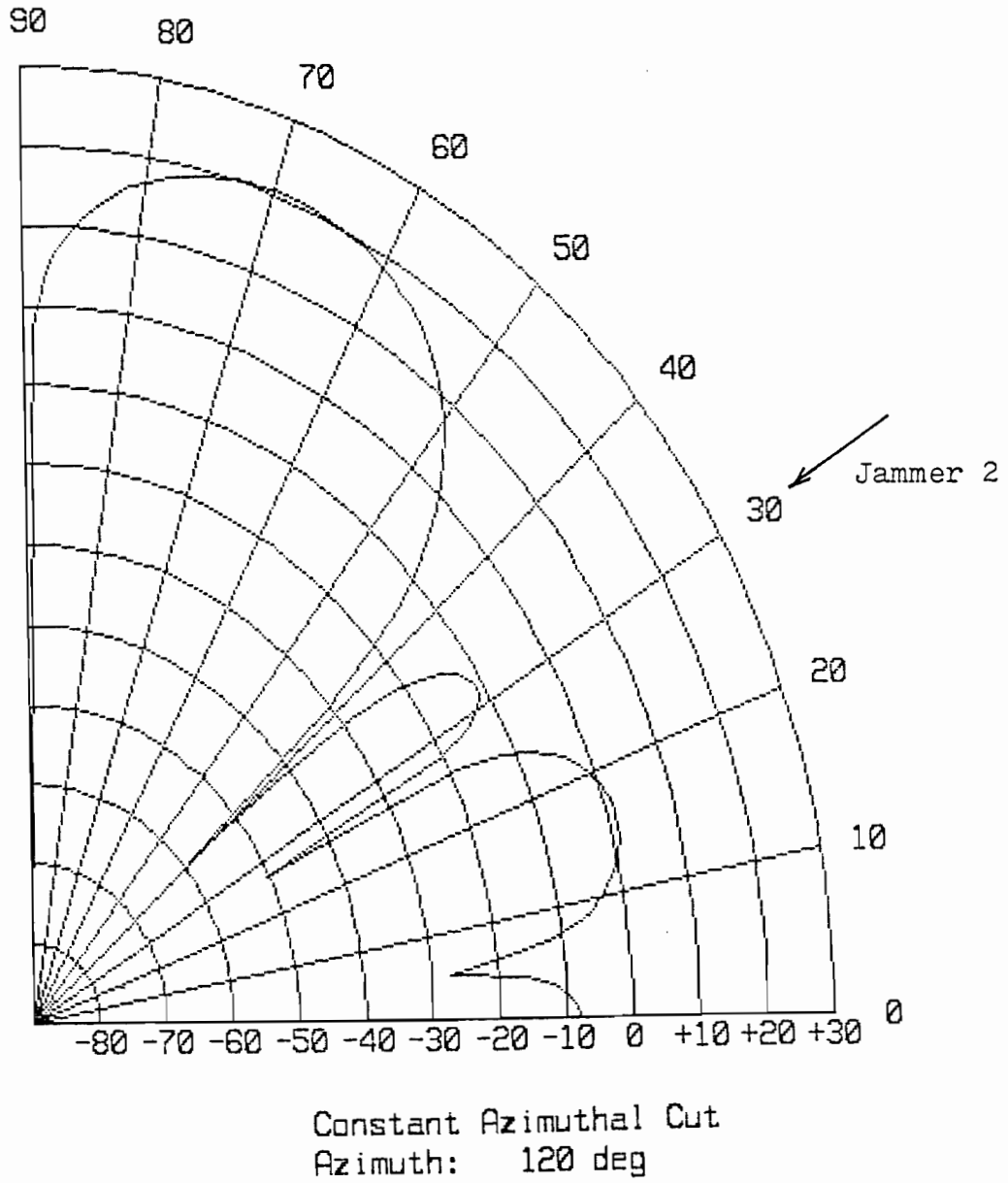
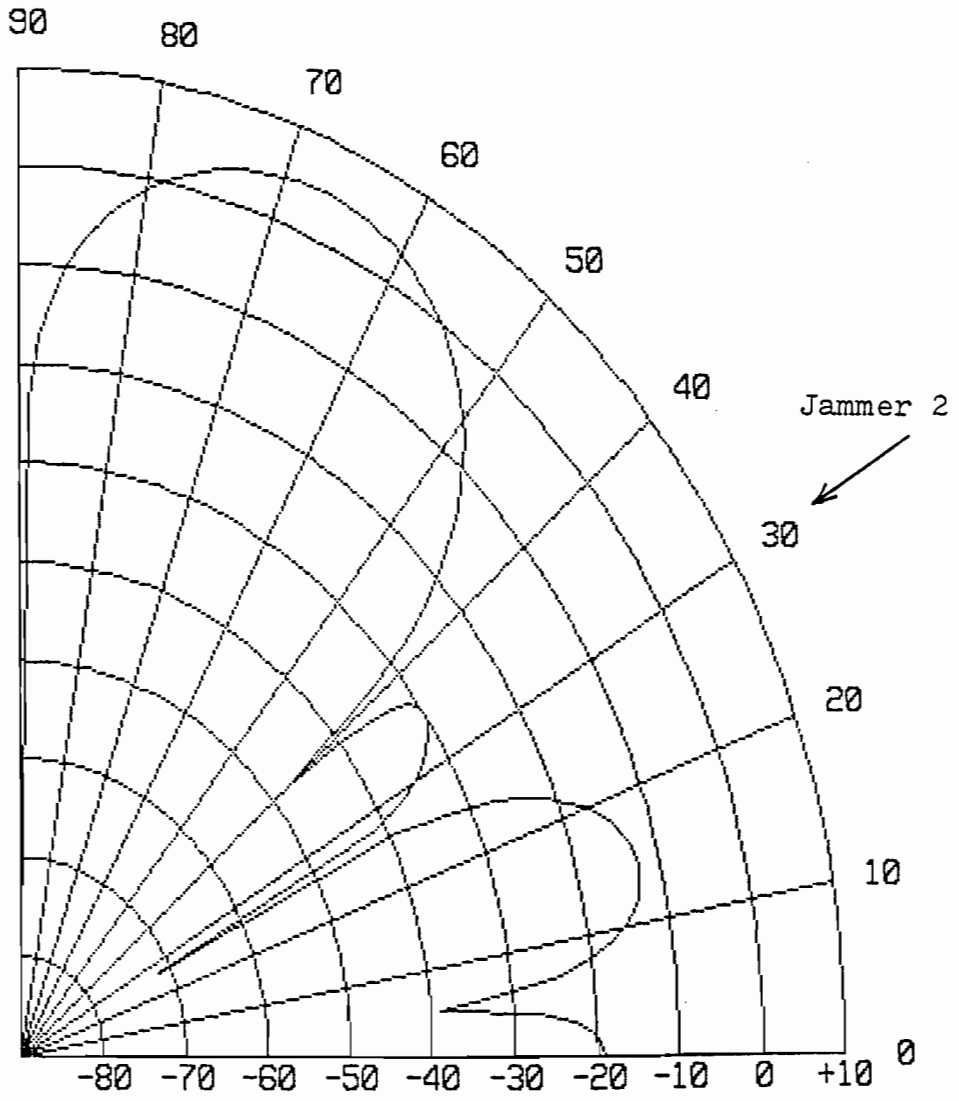


Figure T4.19 Unadapted Antenna Plot for Eigenvector Algorithm at ϕ = 120°



Constant Azimuthal Cut
Azimuth: 120

(Third Largest Eigenvalue)

(Number of Samples Averaged = 48)

Figure T4.20 Eigenvector Sort-Adapted Antenna Plot at ϕ = 120°

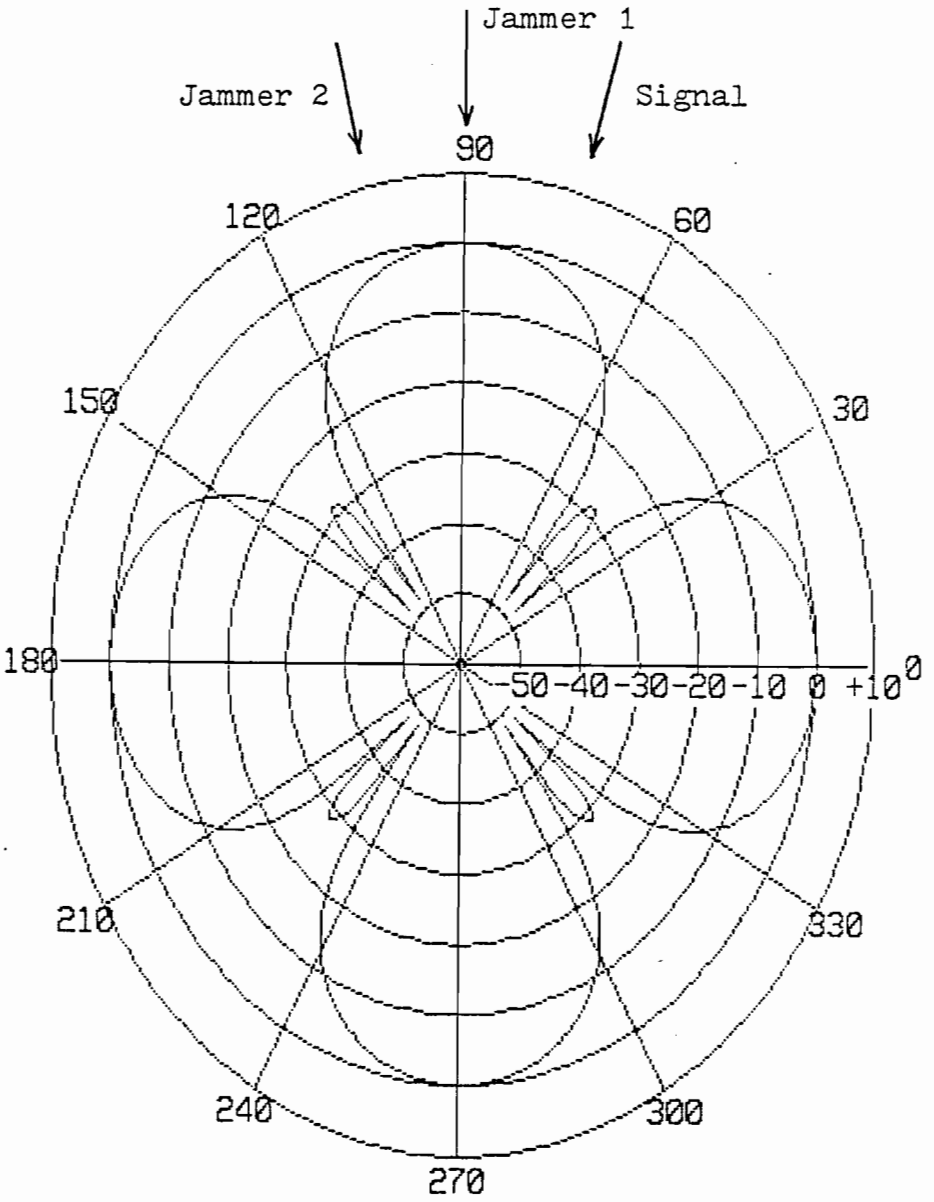
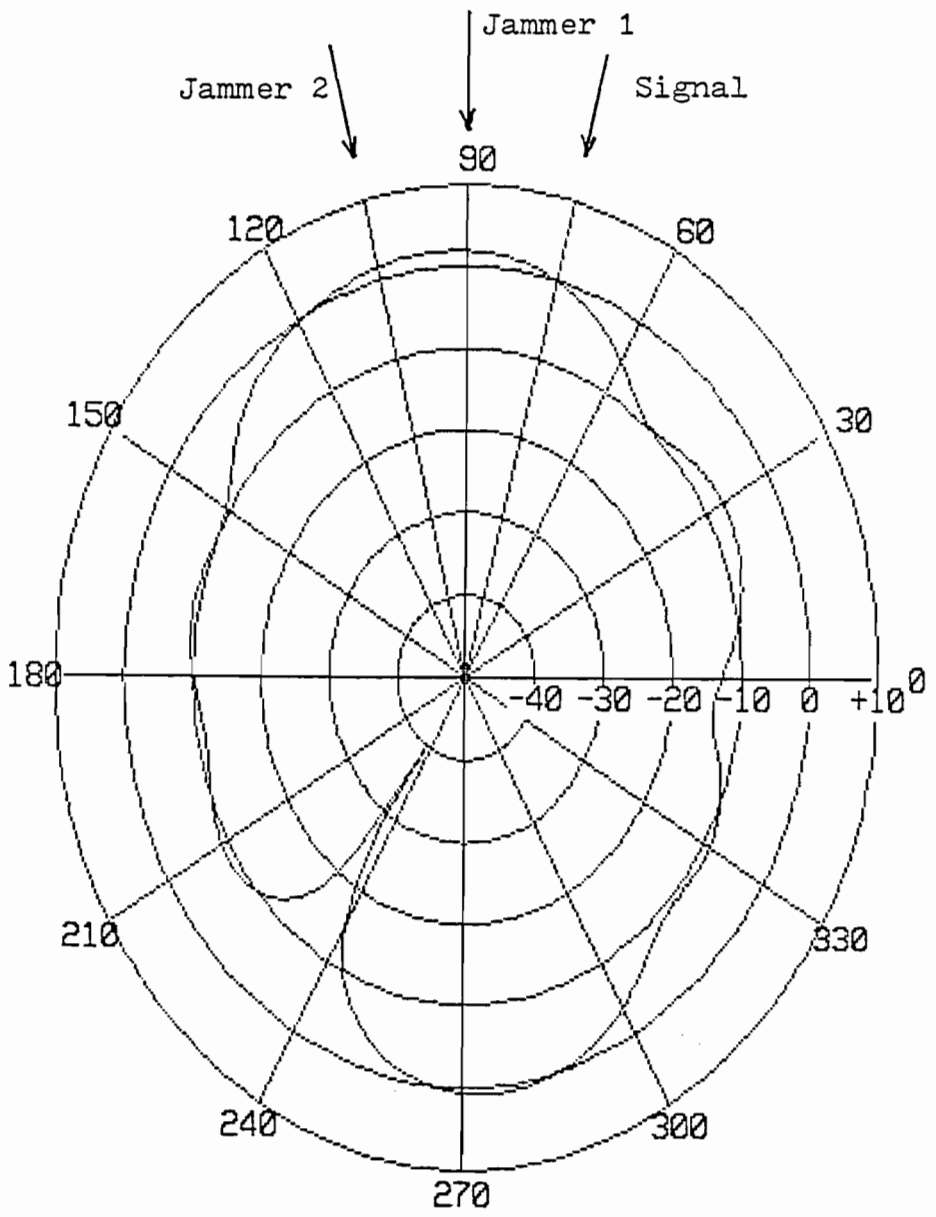


Figure T4.21 Unadapted Antenna Plot for Eigenvector Algorithm at $\theta = 30°$

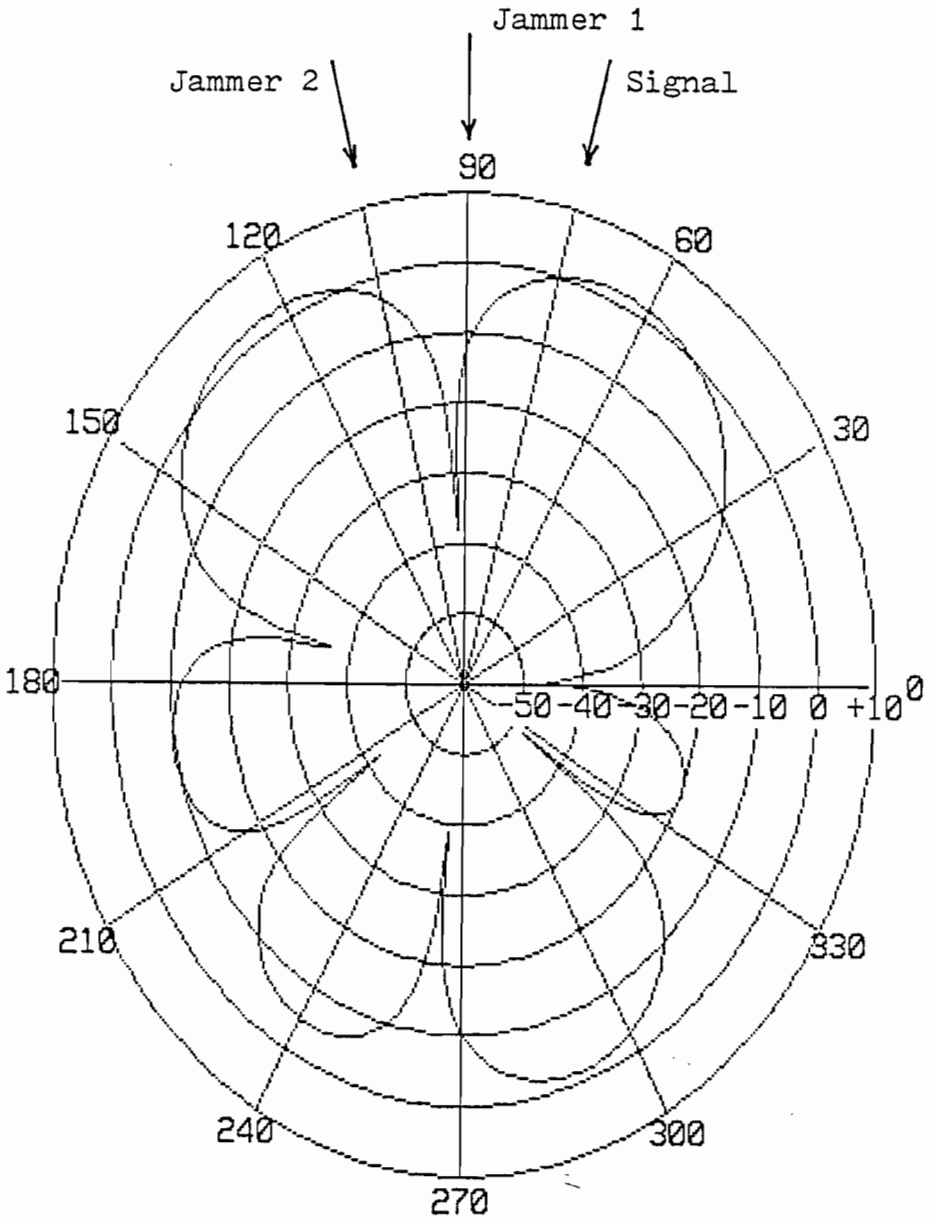


Constant Elevation Cut
Elevation: 30

(Largest Eigenvalue)

(Number of Samples Averaged = 96)

Figure T4.22 Eigenvector Sort-Adapted Antenna Plot at $\theta = 30°$

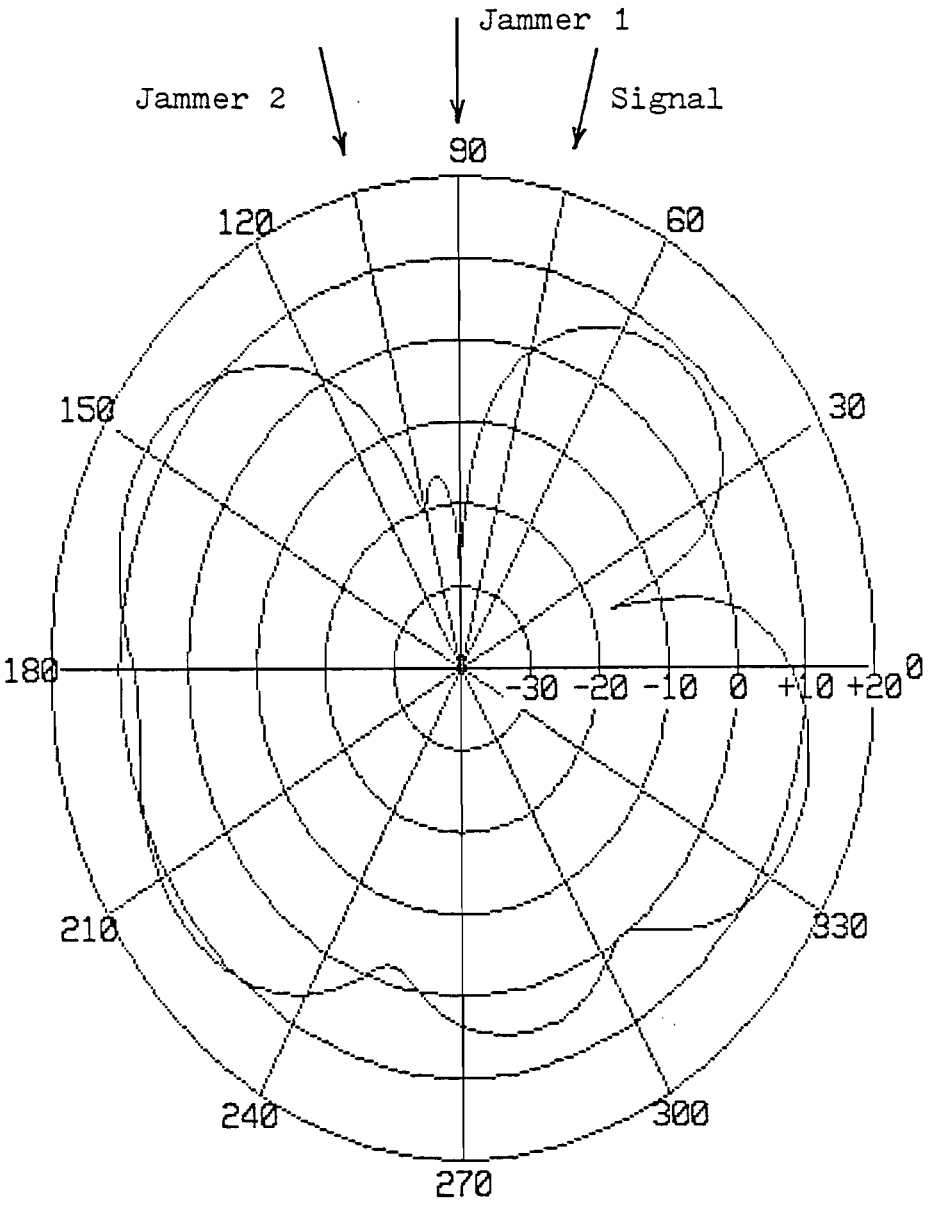


Constant Elevation Cut
Elevation: 30

(Second Largest Eigenvalue)

(Number of Samples Averaged = 96)

Figure T4.23 Eigenvector Sort-Adapted Antenna Plot at $\theta = 30°$



Constant Elevation Cut
Elevation: 30

(Third Largest Eigenvalue)

(Number of Samples Averaged = 96)

Figure T4.24 Eigenvector Sort-Adapted Antenna Plot at $\theta = 30°$

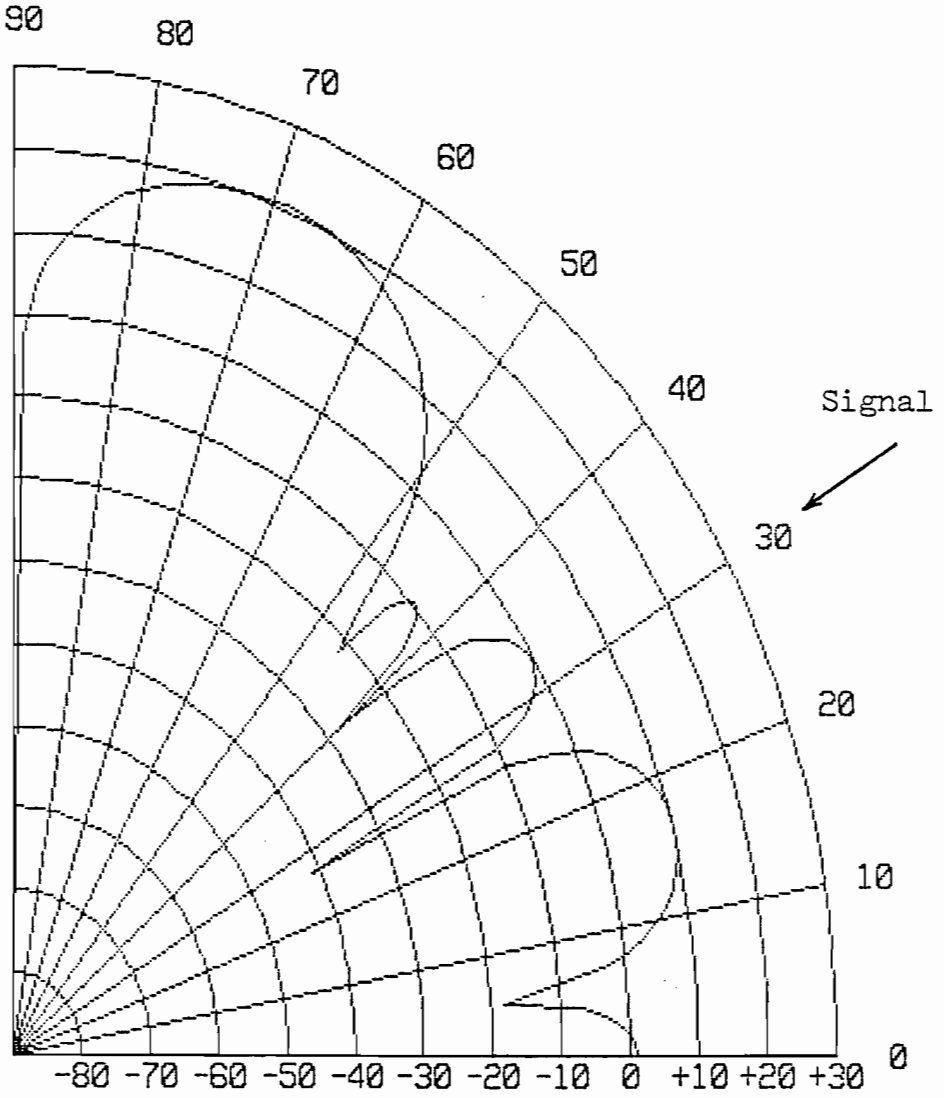
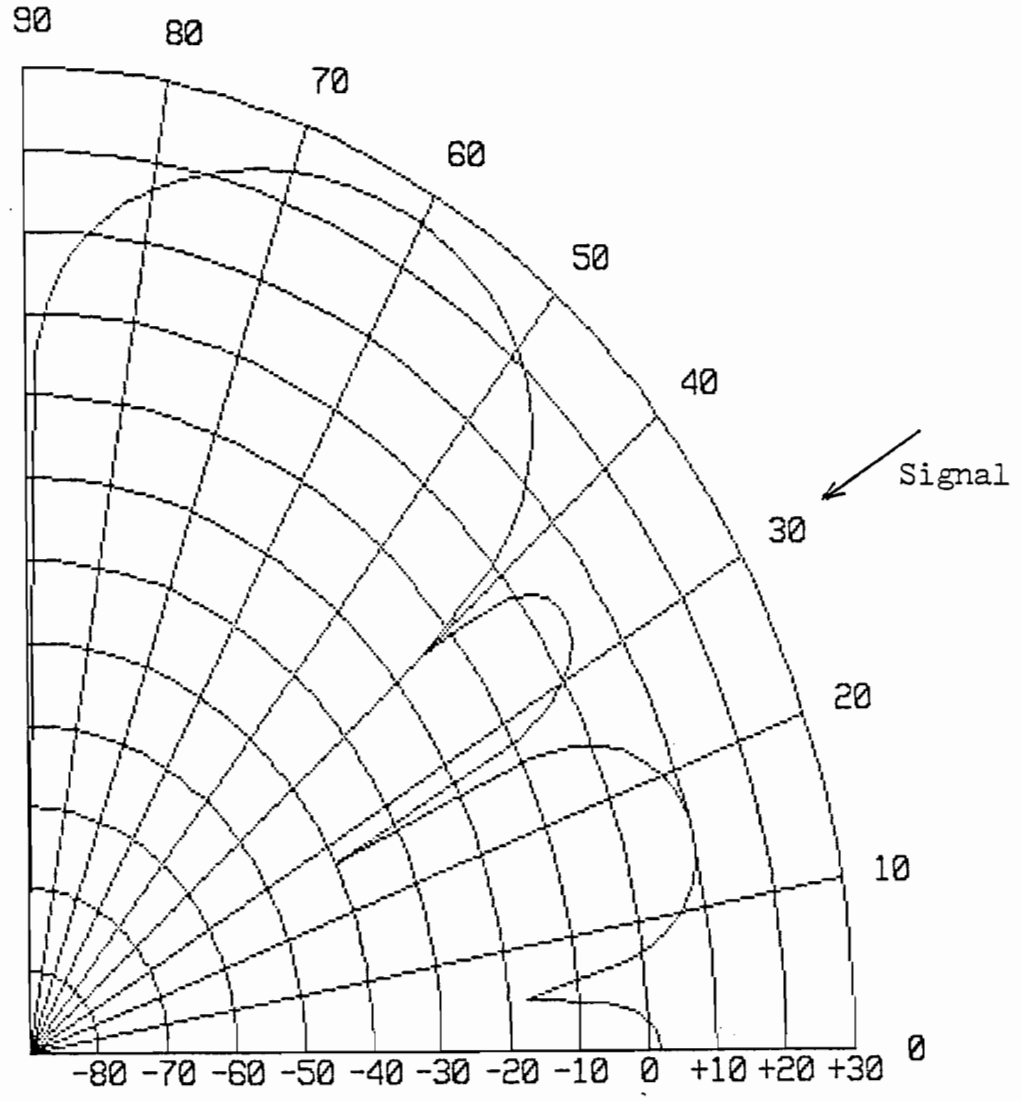


Figure T4.25 Unadapted Antenna Plot for Eigenvector Algorithm at ϕ = 75°



Constant Azimuthal Cut
Azimuth: 75

(Third Largest Eigenvalue)

(Number of Samples Averaged = 96)

Figure T4.26 Eigenvector Sort-Adapted Antenna Plot at $\phi = 75°$

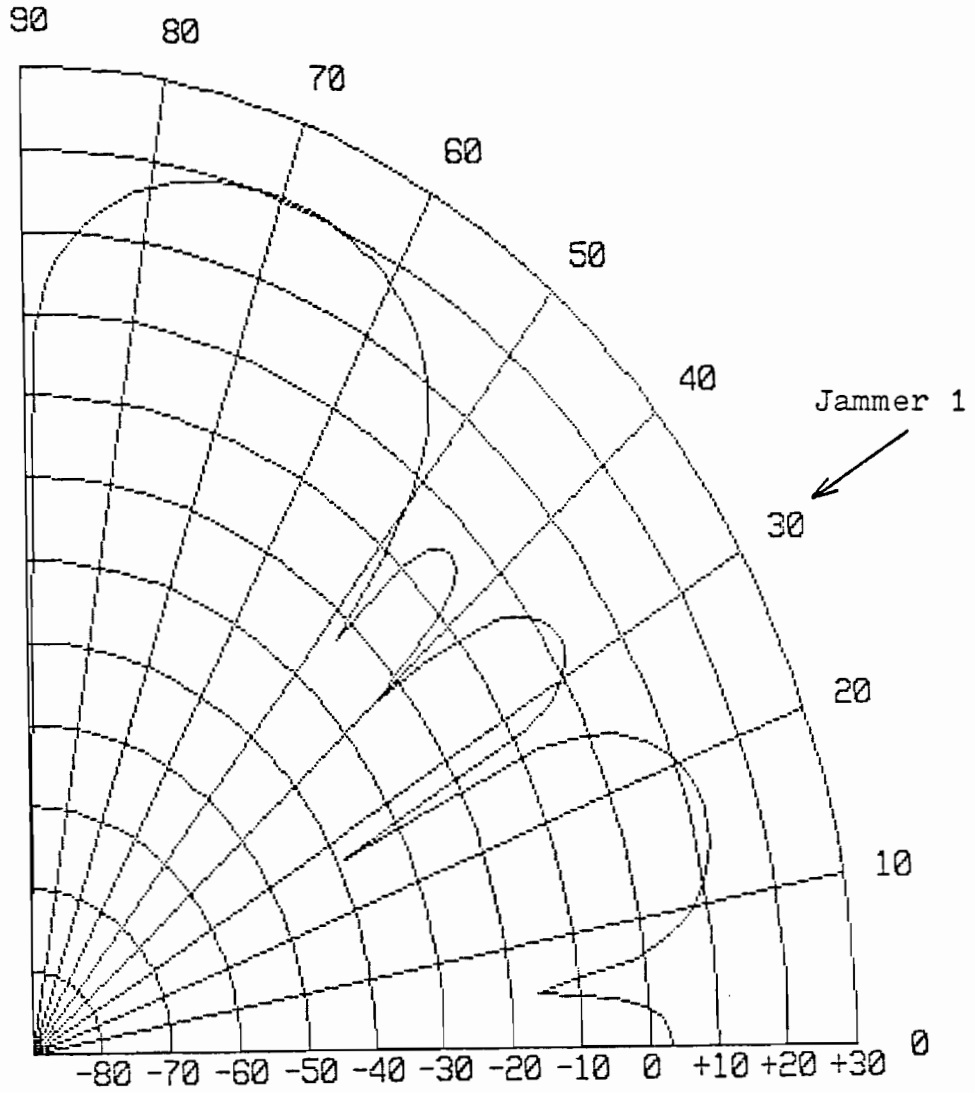
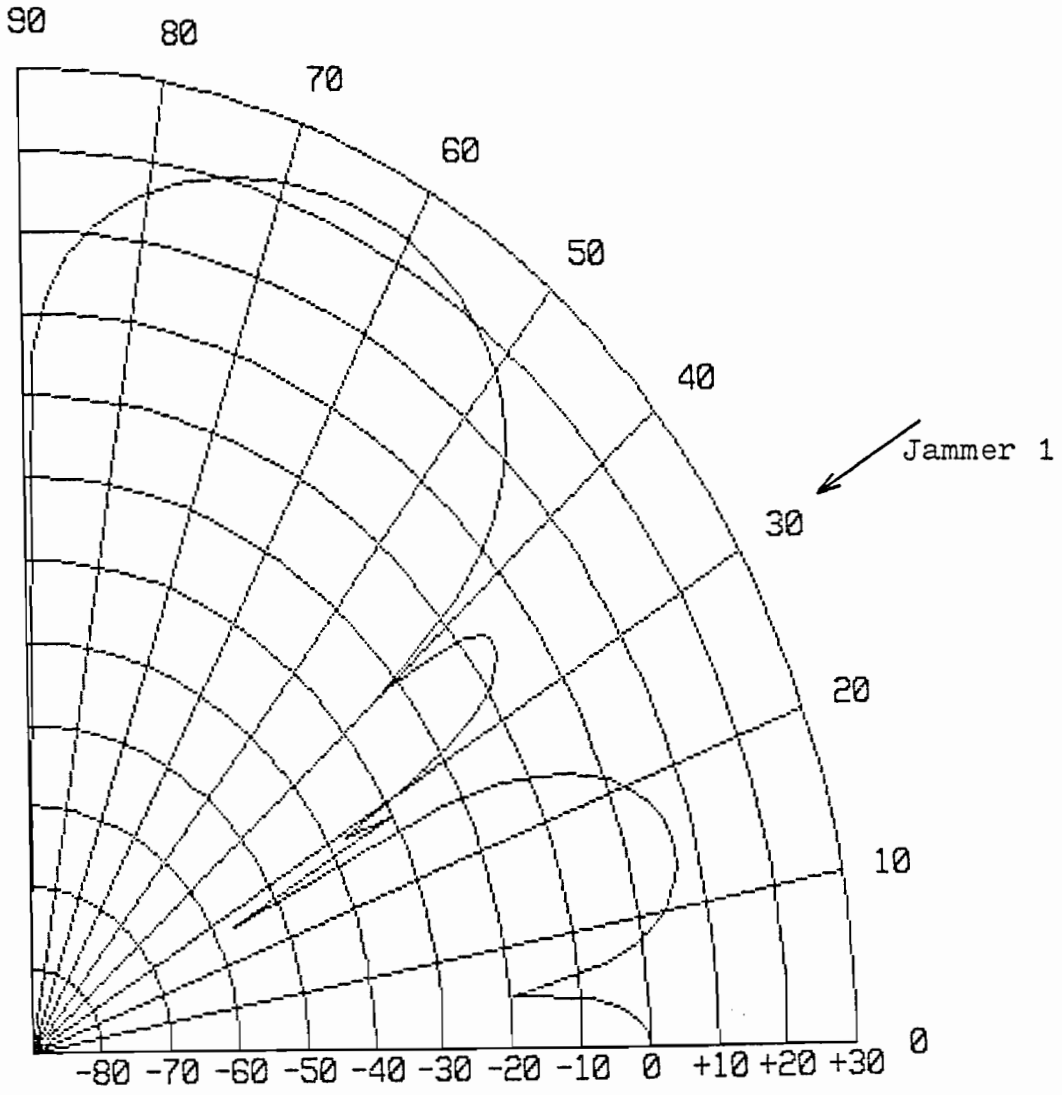


Figure T4.27 Unadapted Antenna Plot for Eigenvector Algorithm at ϕ = 90°



Constant Azimuthal Cut
Azimuth: 90

(Third Largest Eigenvalue)

(Number of Samples Averaged = 96)

Figure T4.28 Eigenvector Sort-Adapted Antenna Plot at $\phi = 90°$

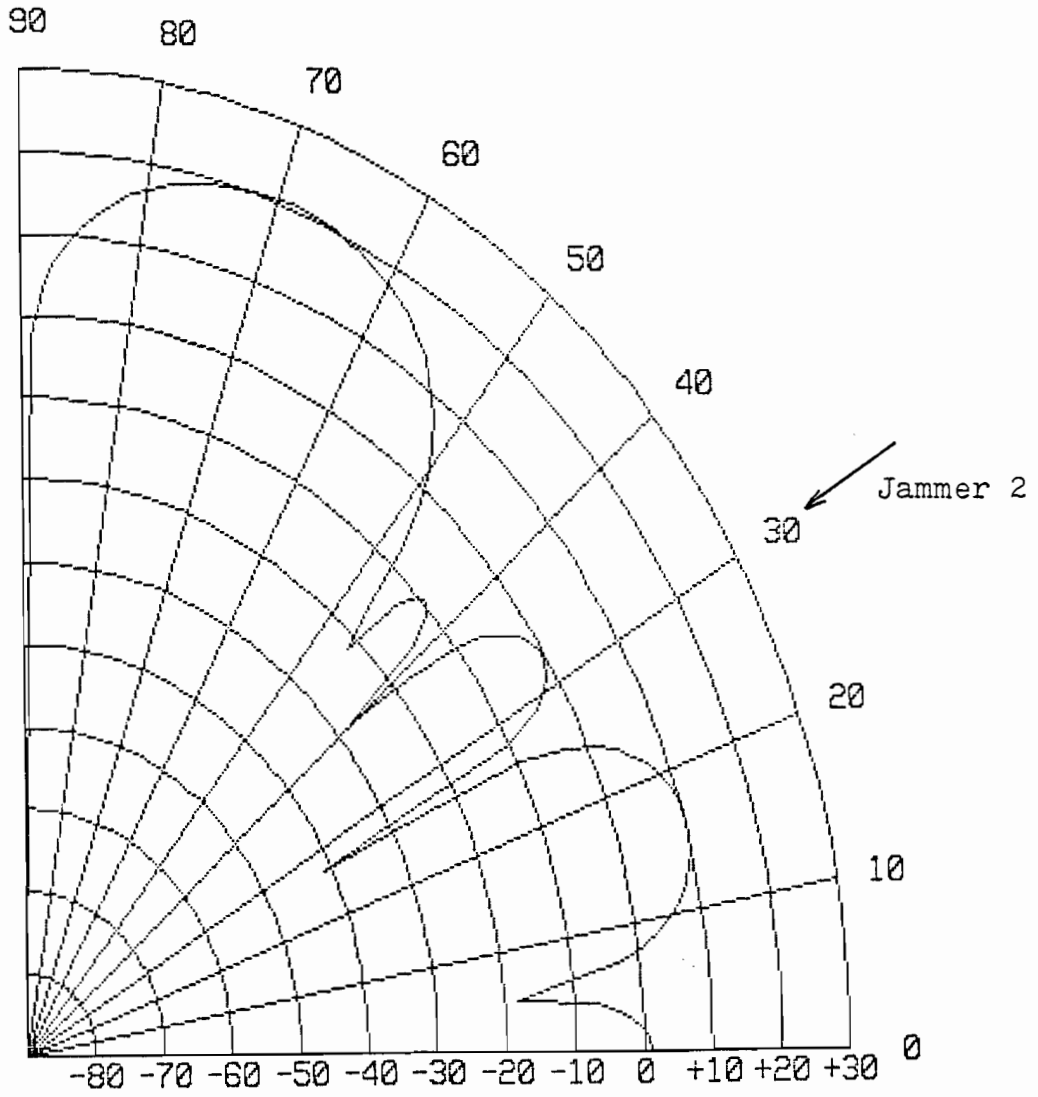
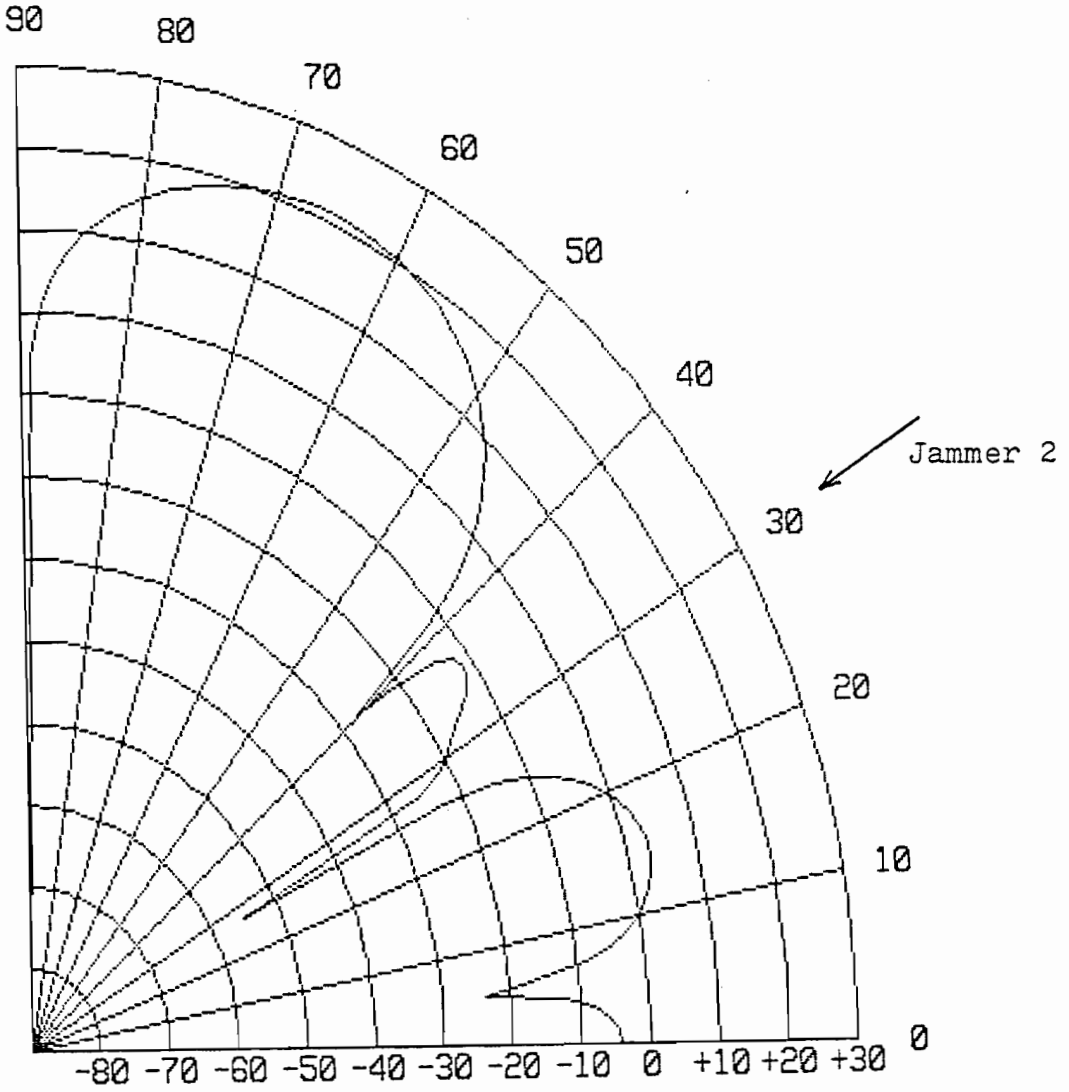


Figure T4.29 Unadapted Antenna Plot for Eigenvector Algorithm at ϕ = 105°



Constant Azimuthal Cut
Azimuth: 105

(Third Largest Eigenvalue)

(Number of Samples Averaged = 96)

Figure T4.30 Eigenvector Sort-Adapted Antenna Plot at ϕ = 105°

6.5 Conclusions and Recommendations

In all of the tests which were conducted, the eigenvector sorting algorithm seems to perform relatively well in the elimination of the hostile jamming signals. The number of samples required to construct an acceptable autocorrelation matrix is usually under 100, which corresponds to approximately six bits of information at 16 samples per bit. This is quite acceptable in the case of, for example, the employment of PN sequences which allow adaptation before the actual information is transmitted. By use of a 31 bit sequence, it will almost always be possible to adapt within this amount of time. Concern must be given, however, to the complexity of the operations undertaken at each step. In the case of the eigenvector sort algorithm, the problem consists of both the formation of the autocorrelation matrix, as well as the intensive numerical operations required to formulate the eigenvectors of this matrix. As is usually the case, algorithms which adapt in a relatively small number of samples also entail a complex series of computations for the weight updates. If this complexity can be tolerated, however, the algorithm seems to be an acceptable choice for the operation in a slowly varying HF environment.

The tests indicate that the algorithm is able to separate signals which are spaced relatively close in arrival angle, but it is also shown that there is a definite dependence of the number of averages required with respect to the signal spacing.

Due to this, it would be inadvisable to rely on the eigenvector sorting algorithm for extremely closely spaced signals. This situation is likely to cause adaptation at a moderate to slow rate.

Finally it should be realized that the use of the eigenvector algorithm requires a sufficient number of samples to formulate the autocorrelation matrix. This is definitely a critical portion of the procedure for acceptable results. As indicated in test 3, premature computation of the eigenvector weights may result in degradation of the algorithm performance. This is not detrimental from the standpoint of hardware complexity, since the formulation of the autocorrelation matrix point estimates is a relatively easy task. It is true, however, that the system must wait for the samples, which in certain cases could require an extended amount of time. In general, however, for moderately spaced signals in the HF environment, the algorithm performance is quite acceptable.

REFERENCES

[1] D. Haegert, "A Study of Adaptive Algorithms for HF Antenna Arrays," M.S. project report, Dept. of Electrical and Computer Engineering, 1986.

[2] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, "Numerical Recipes - The Art of Scientific Computing," Cambridge University Press, 1986.

[3] G. Dahlquist and A. Bjorck, "Numerical Methods," Prentice-Hall, Englewood Cliffs, New Jersey, 1974.

[4] F.S. Acton, "Numerical Methods That Work," Harper & Row, New York, 1970.

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

[6] B. Widrow, P. Mantley, L. Griffiths, and B. Goode, "Adaptive Antenna Systems," Proceedings of the IEEE, vol. 55, Dec. 1967.

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

[8] R.T. Compton Jr., "An Adaptive Array in a Spread Spectrum Communication System," Proceedings of the IEEE, vol. 66, March 1978.

[9] O.L. Frost III, "An Algorithm for Linearly Constrained Adaptive Array Processing," Proceedings of the IEEE, vol. 60, Aug. 1972.

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

[11] Personal Correspondence, Peter J. Ritchie, RADC/DCCD

[12] R. Moats, "Development of a PC Workstation for Performance Evaluation of Adaptive Arrays in an HF Communication System," M.S. project report, Dept. of Electrical and Computer Engineering, 1987.