L_p-NORM ESTIMATION TECHNIQUES APPLIED TO MULTIPLE EMITTER LOCATION

By

CHARLES WILLIAM KRIEL II

Bachelor of Science
Oklahoma State University
Stillwater, Oklahoma
1975

Master of Electrical Engineering
Oklahoma State University
Stillwater, Oklahoma
1975

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
May, 1988

L_p-NORM ESTIMATION TECHNIQUES APPLIED

TO MULTIPLE EMITTER LOCATION

Thesis Approved:

Thesis Adviser

James R. Rowland

David J. Soldan

C. M. Bacon

Jerry Tobin

Norman N. Durham

Dean of the Graduate College

1322612

PREFACE

An approach to L_p-norm based estimation of signal directions of arrival at an antenna array was developed. The Multiple Signal Classification (MUSIC) algorithm of R. O. Schmidt served as the base for this work. The approach is capable of resolving directions of arrival which MUSIC fails to isolate in the presence of impulsive noise. The approach is the first of its kind in explicitly considering values for p unequal to two.

Application of the L_p model to other beamforming algorithms is also discussed, and such application appears promising. Extension of the techniques described herein to other direction of arrival approaches is therefore encouraged.

I wish to express my sincere gratitude to all those who made it possible for me to conduct this research and prepare this dissertation. Although an exhaustive list would fill volumes, I can mention some of those most directly involved.

I am deeply indebted to my major adviser, Dr. Rao Yarlagadda, for his intelligent guidance, concern, and invaluable help. Dr. James Rowland deserves a special acknowledgement as well, both for serving on my committee and for providing the encouragement which convinced me to pursue a Ph.D. in the first place. No less thanks is due to the other members of my committee, Dr. Charles Bacon, Dr. Jerry Johnson, and Dr. Dave Soldan, for their patient assistance and advisement during this work. I would also like to thank all the other members of the Electrical Engineering faculty at Oklahoma State University, as

well as Dr. Craig Sims, for the encouragement and help they were always so willing to lend. Additionally, Mr. Jeffrey Speiser deserves mention for suggesting the topic of the research described herein, and for his help with the generalized singular value decomposition approaches mentioned in this dissertation.

My work would have been immeasurably more difficult had I not had the help of the Electrical Engineering and Graduate College staffs, particularly Mrs. Teresa Tanner, Mrs. Rea Maltzberger, and Mrs. Joyce Gazaway, to smooth the administrative path. I would also like to thank Mr. K. B. Williams for his generous assistance in the actual preparation of this document.

Special thanks is due the Office of Naval Research, which provided the funds for the research on which this dissertation is based. Also indispensable were the computing facilities of the Electrical and Computer Engineering Department of Oklahoma State University, and the help of Mr. Rod McAbee in the use of those facilities. Additionally, heavy use was made of the Virtual Array system for the VAX, which was created by Dr. Dwight Day, and I wish to acknowledge his help in my research efforts.

Finally, my wife, Pamela, my mother, father, sisters, and brother, and my wife's mother have my sincerest gratitude for their constant support, encouragement, patience, tolerance, and understanding.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS

$\mathbf{A}$	– matrix $\mathbf{A}$
$\mathbf{A}^T$	– transpose of $\mathbf{A}$
$\mathbf{A}^*$	– complex conjugate of $\mathbf{A}$
$\mathbf{A}^H$	– complex conjugate transpose of $\mathbf{A}$
tr{$\mathbf{A}$}	– trace of matrix $\mathbf{A}$
rank{$\mathbf{A}$}	– rank of matrix $\mathbf{A}$
$\lvert\mathbf{A}\rvert$	– determinant of matrix $\mathbf{A}$
[a,b,...,c]	– (column) vector with entries a, b, ..., c
diag(a,b,...,c)	– diagonal matrix with entries a, b, ..., c
$\underline{f}$	– (column) vector $\underline{f}$
$\underline{x}\cdot\underline{y}$	– inner ("dot") product of vectors $\underline{x}$ and $\underline{y}$; $\underline{x}\cdot\underline{y} = \underline{x}^H\underline{y}$
E{}	– statistical expectation

CHAPTER I

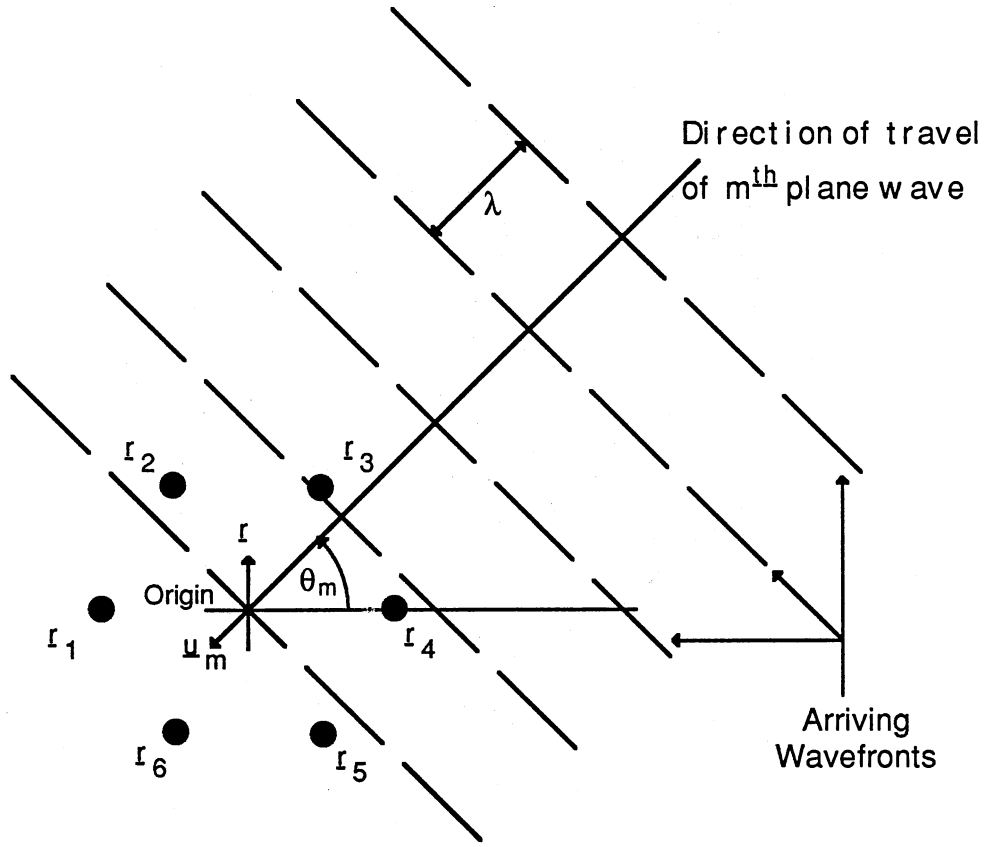
INTRODUCTION

The problem of identifying the directions of arrival of signals from multiple emitters illuminating an antenna array is addressed. Use of the L_p norm, $1 \leq p \leq 3$, in estimating the arrival directions is considered. It is assumed that the emitters are in the far field with respect to the physical size of the sensor array, and that the received signals are narrow band. A signal is classified as narrow band if its bandwidth is small compared to the reciprocal of the transmission time of a wavefront across the array. Since the signals received by the sensors are produced by the emitters rather than the antenna elements, the direction finding system is considered passive rather than active.

The problem of direction of arrival estimation has been a topic of con - tinuing interest to the engineering community for almost twenty years, having applications in areas as diverse as sonar, radar, radio-astronomy, seismology, and surveillance (Middleton, 1970) (Gallop, 1974) (Ouibrahim, 1986).

The Antenna Array Analysis Problem

Figure 1 shows a top view of an antenna array composed of N (six, in the case shown in the figure) receiving antennas, with M plane waves impinging on the array. Only one of the waves, the m^{th}, is shown for clarity of presentation. The system is to be used for azimuth-only direction of arrival determination (the antenna elements and wave propogation directions are coplanar). The anten - na elements are located at positions $\underline{r}_1, \underline{r}_2, \ldots, \underline{r}_N$, and the M incident waves are



λ = received carrier wavelength

$\underline{r}_n$ = position of the n^{th} antenna element, n = 1, 2, ..., N

N = number of elements in the antenna array

$\underline{r}$ = unit position vector (y direction)

$\underline{u}_m$ = unit vector in the look direction of the m^{th} plane wave, m = 1, 2, ..., M

M = number of plane waves incident on the array

θ_m = angle of arrival of the m^{th} plane wave

Figure 1. Geometry of an Antenna Array

impinging on the array at angles $\theta_1, \theta_2, \ldots, \theta_M$ from the abscissa of the (arbitrarily chosen) coordinate system. All M waves are assumed to have the same wavelength, λ (the narrow band assumption).

If the received waves are modeled as complex sinusoids, then the real part of the noise-free signal received by element n due to received wave m at time t is given by:

$$\mathrm{Re}\{s(n,m,t)\} = A_m \cos(2\pi f_c t + \psi_m(n) + \alpha_m) \qquad (1-1)$$

where:

$s(n,m,t)$ = received signal as a function of n, m, and t.

n = array element number; n = 1, 2, ..., N.

m = incident plane wave number; m = 1, 2, ..., M.

t = time.

A_m = (real) amplitude of the m^{th} plane wave.

f_c = carrier frequency.

$\psi_m(n)$ = phase shift due to position of the n^{th} antenna element.

α_m = phase of s(n,m,t) measured at the coordinate system origin.

The phase shift due to antenna element position is the projection of the distance vector from the origin to the element along the line of increasing phase of the incoming wave, and is expressed in Equation (1-2):

$$\psi_m(n) = 2\pi \frac{-\underline{r}_n \bullet \underline{u}_m}{\lambda} = -2\pi \frac{\underline{r}_n \bullet \underline{u}_m}{\lambda} \qquad (1-2)$$

where $\underline{u}_m$ is the unit vector in the look direction of the m^{th} plane wave, and $\underline{r}_n \bullet \underline{u}_m$ is the dot (inner) product of the vectors $\underline{r}_n$ and $\underline{u}_m$. The negative sign in (1-2) arises from the fact that the direction of increasing phase of the incoming wave is opposite to the direction of $\underline{u}_m$ shown in Figure 1. Incorporating the above expression into Equation (1-1), and expressing the result in phasor notation, gives:

$$s(n, m) = A_m e^{j\left(\frac{-2\pi \underline{r}_n \bullet \underline{u}_m}{\lambda} + \alpha_m\right)} \qquad (1-3)$$

The noise-free signal received at each antenna element is the sum of the contri - butions from the M received waves:

$$s(n) = \sum_{m=1}^{M} A_m e^{-j\left(\frac{2\pi}{\lambda} \underline{r}_n \bullet \underline{u}_m\right)} e^{j\alpha_m} \qquad (1-4)$$

$\underline{r}_n \bullet \underline{u}_m = |\underline{r}_n| \cos \delta_{nm}$; δ_{nm} = angle between $\underline{r}_n$ and $\underline{u}_m$

Rewriting Equation (1-4) in vector form gives:

$$\underline{s} = \mathbf{A}\underline{f} \qquad (1-5)$$

where:

$$\underline{s} \equiv \begin{bmatrix} s(1) \\ s(2) \\ \vdots \\ s(N) \end{bmatrix}; \mathbf{A} \equiv \begin{bmatrix} e^{-j\frac{2\pi}{\lambda} \underline{r}_1 \bullet \underline{u}_1} & e^{-j\frac{2\pi}{\lambda} \underline{r}_1 \bullet \underline{u}_2} & \cdots & e^{-j\frac{2\pi}{\lambda} \underline{r}_1 \bullet \underline{u}_M} \\ e^{-j\frac{2\pi}{\lambda} \underline{r}_2 \bullet \underline{u}_1} & e^{-j\frac{2\pi}{\lambda} \underline{r}_2 \bullet \underline{u}_2} & \cdots & e^{-j\frac{2\pi}{\lambda} \underline{r}_2 \bullet \underline{u}_M} \\ \vdots & \vdots & \ddots & \vdots \\ e^{-j\frac{2\pi}{\lambda} \underline{r}_N \bullet \underline{u}_1} & e^{-j\frac{2\pi}{\lambda} \underline{r}_N \bullet \underline{u}_2} & \cdots & e^{-j\frac{2\pi}{\lambda} \underline{r}_N \bullet \underline{u}_M} \end{bmatrix};$$

$$= \left[\underline{a}_1 \; \underline{a}_2 \cdots \underline{a}_M\right]$$

$$\underline{f} \equiv \begin{bmatrix} A_1 e^{j\alpha_1} \\ A_2 e^{j\alpha_2} \\ \vdots \\ A_M e^{j\alpha_M} \end{bmatrix}$$

The matrix **A** contains a complete description of the effects of antenna array geometry on the received signal vector $\underline{s}$. The vector $\underline{f}$ completely de - scribes the effects of the transmitted signal characteristics on the received sig - nal vector $\underline{s}$. The columns of **A**, $\underline{a}_m$, express the effect of array geometry on each

of the M individual incident waves. Plainly, the dot products $\underline{r}_n \cdot \underline{u}_m$ depend (albeit in a nonlinear fashion) on the incident wave arrival angles θ_m. Therefore, determining the entries in the vectors $\underline{a}_m$ is tantamount to determining the angles of arrival of the incident plane waves. The vectors $\underline{a}_m = \underline{a}(\theta_m)$ are called the *signal subspace* vectors or just the *signal* vectors.

Although the matrix **A** depends on θ_m, it of course also depends on antenna element range from the origin and on the received signal frequency. Additionally, although the relationship is not shown in Equation (1-5), the received signal vector is affected by other characteristics of the individual antennae (polarization, response pattern, etc.). Obviously, imperfect knowledge of these and other influences would result in errors in any attempt to determine emitter direction. Schmidt (1981) addresses the influence of polarization and other antenna characteristics on the matrix **A** (the "array manifold" of the antenna array). Additional research is currently in progress on the effects of variations in received frequencies and errors in sensor location information (Shaw and Kumaresan, 1987) (Lo and Marple, 1987). Although these and other related issues are certainly important concerns, the research addressed herein centers around the emitter location process itself. It will therefore be assumed that satisfactory descriptions are available of all such "extraneous" influences on the signal vector ($\underline{s}$).

The total signal from the antenna array is the sum of the received signal and noise. Representing the noise at the various array elements by a noise vector, $\underline{w}$, the total signal observed by the antenna array is given by $\underline{x}$, as follows:

$$\underline{w} \equiv \begin{bmatrix} w(1) \\ w(2) \\ \vdots \\ w(N) \end{bmatrix}$$

$$\underline{x} = \underline{s} + \underline{w}$$

$$\underline{x} = \mathbf{A}\underline{f} + \underline{w} \qquad (1-6)$$

It can be seen from the form of Equation (1-6) that the emitter location problem is in fact a problem of system identification. In an "ordinary" system identification problem, one is given or postulates a (typically) linear structure for the unknown system's dynamics. This takes the form of a vector equation such as:

$$\underline{x}_{n+1} = \Phi_n \underline{x}_n + \underline{b}_n \qquad (1-7)$$

where $\underline{x}_n$ is the system state, Φ_n is the state transition matrix, and $\underline{b}_n$ is the in - put vector, at time t_n. The system identification problem consists of the attempt to determine the entries in Φ_n (i.e. the system parameters) by suitable analysis of sets of known or measured system states ($\underline{x}_n$) and inputs ($\underline{b}_n$).

In comparing Equations (1-6) and (1-7), it is clear that the antenna array problem is posed in a form very like that of the "standard" linear system formula - tion, with the sensor output vector acting as a "system state" (at time t_{n+1}), the **A** matrix corresponding to the state transition matrix, and the noise vector corres - ponding to the system "input" vector. Thus, determination of wave arrival direc - tion (i.e. the columns of **A** in Equation (1-6)) using sensor output vector samples and known or assumed noise characteristics is completely analogous to the "normal" system identification problem, if the system to be identified is driven by a pure noise input. Indeed, the "output-input correlation" method of system identification is based on using a psuedo-random noise sequence as input to the system to be identified (Graupe, 1976) (Goodwin and Sin, 1984).

Thesis Summary

The remainder of this dissertation is organized as follows. Chapter II describes a number of historical approaches to the direction-of-arrival estimation problem. Techniques addressed are the Fourier method, the Maximum-Likelihood approach, the Adaptive Antenna method, and two methods based on linear prediction. The linear prediction approaches discussed are the Burg Technique and the Modified Forward-Backward Linear Predictor. The final traditional method addressed is the Multiple Signal Classification (MUSIC) algorithm. Chapter II closes with a discussion of L_p approaches to direction of arrival estimation.

The application of L_p-norm measures to the beamforming problem is discussed in detail in Chapter III. A development of the Iteratively Reweighted Least Squares (IRLS) algorithm is presented, followed by a discussion of methods for estimating the source covariance matrix. The approaches addressed are the enumeration method, the sum of powers method, and the L_1 selected rows method. Following this discussion, an expression for computing L_p estimates of directions of arrival (DOA) is proposed. Noise aspects are then addressed, followed by a discussion of adaptive approaches to the DOA problem.

Chapter IV presents the results of experiments conducted during this research, including DOA spectra for various values of p and the effects on DOA spectra of iterative DOA revision.

The conclusions drawn from the described research are discussed in Chapter V, which also includes a presentation of areas which show promise for further research in the L_p-DOA problem area.

CHAPTER II

PREVIOUS RESULTS

A wide variety of approaches to the emitter location problem has been employed in the past. Haykin (1985) provides an excellent summary of many of these techniques. The historical approaches discussed here are the Fourier method, the maximum-likelihood method, the adaptive antenna method, the Burg technique, the modified forward-backward linear predictor method, and the MUSIC algorithm.

Fourier Method

In the Fourier method, the "angle spectrum" of the received "spatial se - ries" (the entries in the received signal vector, taken as a sequence in space) is computed. The angle spectrum is defined in a manner similar to that in which the Fourier spectrum of a time series is defined, as follows:

$$X(\phi) \equiv \sum_{n=1}^{N} x(n)\, e^{jn\phi} \qquad (2-1)$$

$$\phi = \frac{2\pi d}{\lambda} \sin \theta$$

where x(n) is an output sample from antenna number n, ϕ is the "electrical phase angle" for the array, d is the separation between array elements, λ is re - ceived signal wavelength, and θ is angle of arrival. The separation d, above, refers to elements in a colinear array, and therefore the above expression for electrical phase angle applies to colinear arrays only. The square of the

magnitude of the angle spectrum is called the "periodogram".

The essence of the Fourier method is to detect the maxima in a plot of the periodogram as a function of the electrical phase angle ϕ. The angles ϕ at which the periodogram attains a maximum are related to wave arrival angles, as can be seen intuitively by comparing Equation (2-1) to Equation (1-4). In a sense, one can say that the effect of forming the angle spectrum is to "remove" phase shifts due to the angle ϕ, so that contributions to the periodogram from the separate antennas add "in phase" when ϕ corresponds to a wave arrival angle.

Some of the advantages of the Fourier approach are that it is simple in concept, and that it is non-parametric. By "non-parametric", it is meant that the method does not depend on the formulation of a model for the process generating the received data (apart from the model of the receiving array). On the other hand, it provides relatively poor resolution, even when only a single incident wave is present. This is due to a comparatively wide main lobe and the presence of side lobes in the periodogram of a received wave. The poor resolution of the Fourier approach makes it inappropriate for use in many situations, particularly when multiple closely-spaced emitters are present. A related disadvantage of the Fourier method is that resolution can be improved only by increasing the array aperture, which in turn requires increasing the physical size of the array.

Maximum-Likelihood Method

This method involves postulating a model of the conditional probability density function for the received antenna output samples (Böhme, 1983) (Van Trees, 1968). The probability density function used is expressed in Equation (2-2), below:

$$f(\underline{x}_c|\underline{q},\underline{\phi}) = Ke^{\left[\frac{-S(\underline{x}_c)}{2\sigma^2}\right]} \qquad (2-2)$$

$$\text{where} \quad S(\underline{x}_c) = \sum_n \left\{ x_c(n) - \sum_{m=1}^{2M} q_m s_m(n) \right\}^2$$

In Equation (2-2), f is a conditional probability density function, K is a constant, $\underline{x}_c$ is the "in-phase" component of the observed sample vector, and $\underline{q}$ is a vector of values dependent on the incident waves' amplitudes, phases, and angles of arrival. The vectors $\underline{\phi}$ and $\underline{s}_m$ are dependent only on the angles of arrival of the incident waves, and $s_m(n)$ is the $n^{\underline{th}}$ element of vector $\underline{s}_m$. The entries of $\underline{\phi}$ are the "electrical phase angles" of the various arriving waves, and are defined as in Equation (2-1). This implies that this development is valid for the colinear array case only. The vectors $\underline{s}_m$ are obtained from a Gram-Schmidt orthonormalization procedure (Haykin, 1983) applied to a matrix of functions of the elements of $\underline{\phi}$. The value σ^2 is the variance of the (assumed Gaussian) noise at each antenna element, M is the number of incident waves, n is the antenna element number, and m is the incident plane wave number. The "in-phase" component of the sample vector is the real part of the complex sinusoids being received.

The probability density function, $f(\underline{x}_c|\underline{q},\underline{\phi})$, is the probability of a particular value for the in-phase component of the observed sample vector, given selected values of the vectors $\underline{q}$ and $\underline{\phi}$. The "particular value", given in Equation (2-3), is the value which would result, in the absence of noise, from M plane waves as described by the selected $\underline{q}$ and $\underline{\phi}$ impinging on the array.

$$x_c(n) = \sum_{m=1}^{M} a_m(\cos \psi_m) \cos(n\phi_m) + \sum_{m=1}^{M} -a_m(\sin \psi_m) \sin(n\phi_m) \qquad (2-3)$$

where:

$x_c(n)$, m, and M are as defined above,

ϕ_m is the electrical phase angle of the m^{th} plane wave

(the m^{th} element of $\underline{\phi}$),

a_m is the complex amplitude of the m^{th} plane wave, and

ψ_m is the phase of the m^{th} plane wave.

In the maximum likelihood approach, a likelihood function $f(\underline{\phi}|\underline{x}_c)$ is formed from the probability density function given in Equation (2-2). This likeli - hood function is simply the probability that a selected output sample vector re - sults from a given electrical phase angle vector. To determine the actual electri - cal phase angle vector (and thus the actual angles of arrival of the incoming waves), the likelihood function is maximized. That is, initial values are guessed for ϕ_m, the likelihood function is evaluated based on the initial estimates, new estimates of ϕ_m are computed from the results, and the process is repeated until some iteration termination criterion is met.

The maximum likelihood approach shows resolution generally superior to that possible with the Fourier method, however, it suffers from several disad - vantages. One is that, in order to remove the effects on the likelihood function of the vector $\underline{a}$ (which embodies the transmitted waves' characteristics), it is neces - sary to assume a uniform joint probability density function for $\underline{\phi}$ and $\underline{a}$, which may not be justifiable in practice. Also, the development above applies to co - linear arrays only, and a maximum likelihood approach for arbitrary array

geometry is likely to be considerably more complicated. Another disadvantage is computational complexity, since both a Gram-Schmidt orthonormalization procedure and a multi-variate maximization procedure must be performed at each iteration step (Fletcher and Powell, 1963) (Bard, 1974) (Bandler, 1973). This disadvantage is less important than it once would have been, since very sophisticated digital processing capabilities are now relatively inexpensive to obtain; however, computational complexity is still a consideration in selection of an approach to be used. A final disadvantage is that the surfaces over which the multi-variate maximization process must be performed are often ill-conditioned, resulting in numerical problems and possible loss of resolution.

Adaptive Antenna Method

This method applies directly only to the case of a colinear antenna array in a symmetric multipath environment (Kesler, 1981). When "symmetric multi - path" transmission is present, the array is receiving a single plane wave from a direction θ from the normal to the array, along with a single reflection of the wave from a direction $-\theta$ from the normal to the array.

In the adaptive antenna approach, a beam forming network and an adap - tive canceller are used to minimize a weighted difference between a "reference beam" and two "auxiliary beams". The reference beam is produced by forming a weighted sum of the received antenna output samples, as shown in Equation (2-4):

$$X_R = \sum_{n=-\frac{N-1}{2}}^{\frac{N-1}{2}} b_n x_n \qquad (2-4)$$

In Equation (2-4), X_R is the output of the reference beam, N is the number of array elements, n is the number of the array element under consideration, $\{b_n\}$ is a set of real weights to be applied to the antenna outputs, and x_n is the output of the n^{th} array element. The array elements are numbered such that element number 0 is in the center of the (colinear) array. The set of weights b_n are constrained to be symmetric ($b_{-n} = b_n$), and a binomially weighted array has been found to yield good overall performance (Haykin and Kesler, 1983).

The two auxiliary beams, X_1 and X_2, are formed by summing weighted, phase-shifted samples from each of the array elements, as shown in Equation (2-5):

$$\left.\begin{aligned} X_1 &= \sum_{n=-\frac{N-1}{2}}^{\frac{N-1}{2}} b_n x_n e^{jn\phi} \\ X_2 &= \sum_{n=-\frac{N-1}{2}}^{\frac{N-1}{2}} b_n x_n e^{-jn\phi} \end{aligned}\right\} \qquad (2-5)$$

The factor ϕ, by which the samples are successively phase shifted in the auxiliary beams, is the electrical phase angle of the direction (relative to the normal to the array) of the first auxiliary beam.

In the adaptive antenna technique, the reference beam acts as a "desired response" for an adaptive controller. The controller adjusts a real weight, w, which is multiplied by the sum of the two auxiliary beams. The controller is constructed to minimize the mean square value of the "error", which is defined as the difference between the "desired response" (reference beam) and the sum of the auxiliary beams times the real weight, w, described above. The antenna is considered "nulled" when the mean square error is minimized. It can be shown

that, when the adaptive antenna is nulled, the weight w is a direct function of the angle of arrival of the primary wave, which can be estimated from a graph of weight versus arrival angle.

The advantages to this technique are that it is comparatively straightfor - ward in implementation and does not involve a large computational burden. Its primary disadvantage is that it is limited in application, being restricted to sym - metric multipath environments and colinear arrays only. Another disadvantage is that it exhibits poor resolution for angles of arrival near the normal to the ar - ray.

Linear Predictive Methods

In the linear prediction approach, the output, x_n, from antenna element n is "predicted" as a linear combination of the outputs from the T "previous" ele - ments in the array, as in Equation (2-6):

$$\hat{x}_n = \sum_{t=1}^{T} h_t x_{n-t}; \quad T < n \leq N \qquad (2-6)$$

where h_t are the "prediction filter coefficients" and T (the number of spatial sam - ples used in the "prediction") is called the "order" of the prediction filter.

The forward prediction error, or residual, is defined as the difference be - tween the observed (measured) output of the n^{th} antenna element and its pre - dicted value. The term "forward" is used because the "prediction" is made using samples that are considered to be "previous" (in space) to the n^{th} element. This is an arbitrary distinction, and a completely analogous "backward" prediction er - ror will be introduced later. Evaluating the difference between the actual and predicted outputs yields:

$$f(T,n) = \sum_{t=0}^{T} r_t x_{n-t}; \quad T < n \le N \qquad (2-7)$$

where:

$f(T,n)$ = forward prediction error using a filter of order T at antenna element n, and

$$r_t = \begin{cases} 1; & t = 0 \\ -h_t; & t = 1, 2, \ldots, T \end{cases} \qquad (2-8)$$

The spatial filter defined by the set of coefficients r_t is called a prediction error filter of order T. If the vectors $\underline{r}$ and $\underline{x}$ are defined as:

$$\underline{r} \equiv \begin{bmatrix} r_0 \\ r_1 \\ \vdots \\ r_T \end{bmatrix}; \qquad \underline{x}(n) \equiv \begin{bmatrix} x_n \\ x_{n-1} \\ \vdots \\ x_{n-t} \end{bmatrix} \qquad (2-9)$$

then the mean-square value of the forward prediction error is:

$$\varepsilon_f \equiv E\{ | f(T,n) |^2 \} = \underline{r}^H \mathbf{R} \underline{r} \qquad (2-10)$$

where $\mathbf{R} \equiv E\{ \underline{x}^*(n)\, \underline{x}^T(n) \}$, $E\{\}$ denotes the expectation operator, * denotes complex conjugate, ^{T} denotes transpose, and ^{H} denotes complex conjugate transpose. Defining a similar filter in the "backward" direction produces:

$$b(T,n) = \sum_{t=0}^{T} r_{bt} x_{n-t}; \quad T < n \le N \qquad (2-11)$$

$$r_{bt} = \begin{cases} -h_{bt}; & t = 0, 1, 2, \ldots, T-1 \\ 1; & t = T \end{cases} \qquad (2-12)$$

$$\underline{r}_b \equiv \begin{bmatrix} r_{b0} \\ r_{b1} \\ \vdots \\ r_{bT} \end{bmatrix} \qquad (2-13)$$

$$\varepsilon_b \equiv E\{ | b(T,n) |^2 \} = \underline{r}_b^H \mathbf{R} \underline{r}_b \qquad (2-14)$$

The backward prediction error filter defined by the vector $\underline{r}_b$ computes the difference between the observed value of x_{n-T} and its predicted value (since $r_{bT} = 1$).

It can be shown that when the average power of the prediction error is minimized, peaks will occur at points corresponding to the angles of arrival of the incident waves in the power spectrum of an all-pole filter constructed from the prediction coefficients. That is,

$$P(\phi) \equiv \frac{K}{\left| 1 + \sum_{t=1}^{T} r_t e^{-jt\phi} \right|^2}$$

where K is a constant and ϕ is the electrical phase angle (Equation (2-1)), exhibits peaks at values of ϕ corresponding to the directions of arrival of the incident waves. This expression uses only the forward error coefficients (r_t); it may be that a similar expression involving both the forward and backward coefficients would be of use as well.

The matrix **R** is Hermitian positive semidefinite for any vector $\underline{x}(n)$, and is usually positive definite. Additionally, when the input vector $\underline{x}(n)$ is "spatially stationary", **R** is Toeplitz as well. By "spatially stationary", it is meant that the statistics of the i^{th} output with respect to the j^{th} output depend only on the <u>difference</u> between i and j, not their actual values.

When **R** is both Hermitian and Toeplitz, the minimum mean square error for both the forward and backward filters is obtained by setting:

$$\underline{d} \equiv \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}; \quad \underline{d}_b \equiv \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \qquad (2-15)$$

$$\mathbf{R}\underline{r} = \varepsilon_{f,min}\underline{d} \qquad (2-16)$$

$$\mathbf{R}\underline{r}_b = \varepsilon_{b,min}\underline{d}_b \qquad (2-17)$$

$$r_{bt} = r^*_{T-t}; \qquad t = 0, 1, \ldots, T \qquad (2-18)$$

In this case, it is also true that $\varepsilon_{f,min} = \varepsilon_{b,min} = \varepsilon_{min}$.

Burg Technique

In the Burg technique, recursive relations are developed from Equations (2-15) through (2-18) for Toeplitz **R** to permit computation of the parameters of a prediction-error filter of order T+1 given the parameters of a filter of order T (Burg, 1967) (Burg, 1975). The resulting relationships are:

$$r_t(T+1) = r_t(T) + \Gamma_{T+1} r^*_{T+1-t}(T); \; t = 0, 1, \ldots, T \qquad (2-19)$$

$$\varepsilon_{min}(T+1) = (1-|\Gamma_{T+1}|^2)\varepsilon_{min}(T) \qquad (2-20)$$

where Γ_{T+1} is a coefficient to be computed at each increase in filter order. Equations (2-19) and (2-20) are known as the Levinson-Durbin recursion (Levinson, 1947) (Durbin, 1960). Prediction errors at each order can be evaluated using the prediction errors from the previous order by:

$$f(T,n) = f(T-1,n) + \Gamma_T b(T-1,n-1) \qquad (2-21)$$

$$b(T,n) = b(T-1,n-1) + \Gamma_T^* f(T-1,n) \qquad (2-22)$$

Equations (2-21) and (2-22) give rise to a lattice filter structure, permitting comparatively simple transition from one filter order to the next. The value of Γ_T is computed by minimizing the sum of the mean square values of the forward and backward prediction errors for a filter of order T. That is, the quantity $J(T) \equiv E\{ | f(T,n) |^2 + | b(T,n) |^2 \}$ is minimized. This produces:

$$\Gamma_{opt,T} = -\frac{2E\{f(T-1,n)b^*(T-1,n-1)\}}{E\{|f(T-1,n)|^2 + |b(T-1,n-1)|^2\}} \qquad (2-23)$$

Equation (2-23) is known as the Burg formula.

The Burg technique for emitter location starts with a filter of order 0, wherein $f(0,n) = b(0,n) = x_n$ for n from 1 to N and $\varepsilon_{min}(0)$ is the average of $|x_n|^2$. Filter order is then repeatedly incremented (computing $\underline{r}$ and $\underline{r}_b$ at each stage) until some desired threshold is reached in the mean-square error sum. The power spectrum of the all-pole filter constructed from the resulting prediction coefficients is then plotted, and the peaks in that spectrum are taken as the angles of arrival.

The advantages to the Burg technique are that it exhibits improved resolution over the Fourier method and is more generally applicable than the adaptive antenna approach. Its disadvantages are that most of the work with it has been applied to colinear antenna arrays only, and that the sample vector must be spatially stationary. The spatial stationarity requirement is violated in the case of arbitrarily coherent arriving waves, which makes the method unsuitable for a number of environments (e.g. specular multipath).

Modified Forward-Backward Linear Predictor

The (standard) forward-backward linear predictor (FBLP) approach is based on the same principles as the Burg technique (i.e. linear prediction) (Ulriych and Clayton, 1976) (Nuttal, 1976). However, the prediction error energy to be minimized in the FBLP approach is defined to be:

$$\varepsilon \equiv \sum_{n=T+1}^{N} \left(|f(T,n)|^2 + |b(T,n)|^2 \right) = \underline{r}^H \mathbf{C} \underline{r} \qquad (2-24)$$

where:

$$\mathbf{C} \equiv \sum_{n=T+1}^{N} \left[\underline{x}^*(n) \underline{x}^T(n) + \underline{x}_b(n) \underline{x}_b^H(n) \right] \qquad (2-25)$$

and the vectors $\underline{r}$ and $\underline{x}$ are defined in a manner analogous to that for the Burg technique. The matrix **C** in Equation (2-25) is, in general, non-Toeplitz. The standard FBLP approach exhibits improved performance over the Burg technique, however, its performance is limited by the noise contributions to any estimate of the matrix **C**.

In the modified forward-backward linear predictor (MFBLP) method, the estimate of **C** is replaced by an estimate **D**, which is of the same dimensions but of lower rank (Kumaresan and Tufts, 1981) (Tufts and Kumaresan, 1982) (Eckart and Young, 1931). The matrix estimate **D** is constructed using selected eigenvectors of a submatrix of the estimate of **C**. By discarding eigenvectors corresponding to "small" eigenvalues (which are introduced by observation noise), the selection process provides a means to effectively increase signal to noise ratio, permitting an increase in filter order and a corresponding increase in resolution.

The advantages of the modified forward-backward linear predictor approach are that it exhibits increased resolution over the Burg technique and that it works well even in the presence of arbitrarily coherent arriving waves (since it does not require that the covariance matrix be Toeplitz). Its primary disadvantage is that most of the work using it has been applied to colinear antenna arrays only.

The MUSIC Algorithm

The MUSIC (MUltiple SIgnal Classification) algorithm of R. O. Schmidt (1981) employs an eigenvector decomposition of the sample autocovariance matrix estimate to develop a function with peaks at the angles of arrival of the incident waves (Speiser, 1985). The technique is also applicable to estimation of the frequencies of sinusoids present in noise. Eigenvector techniques have

shown improved performance over spectral approaches to this problem, such as autoregressive or Prony (1795), particularly with lower signal to noise ratios and in the presence of closely-spaced narrow band spectral components (Hildebrand, 1956). Development of the MUSIC approach to emitter location is discussed in this section.

Recalling Equation (1-6), it can be seen that a sequence of samples $\underline{w}_S$ of the (time-varying) noise vector $\underline{w}(t)$ can be considered to be the output of a ran - dom process. Thus, samples $\underline{x}_S$ of the total signal vector $\underline{x}(t)$ can also be viewed as the output of a random process (actually the sum of a deterministic process and a purely random process). The covariance matrix associated with the total signal vector is (where E{} signifies expectation over s):

$$\mathbf{R}_x \equiv E\{\underline{x}\,\underline{x}^H\} = E\{(\mathbf{A}\underline{f} + \underline{w})(\mathbf{A}\underline{f} + \underline{w})^H\}$$

Assuming that the noise ($\underline{w}$) is zero mean and independent of the incom- ing signals ($\mathbf{A}\underline{f}$):

$$\mathbf{R}_x = \mathbf{A}E\{\underline{f}\,\underline{f}^H\}\mathbf{A}^H + E\{\underline{w}\,\underline{w}^H\} = \mathbf{R}_S + \mathbf{R}_W$$

where:

$$\mathbf{R}_S \equiv \mathbf{APA}^H;\ \mathbf{P} \equiv E\{\underline{f}\,\underline{f}^H\};\ \mathbf{R}_W \equiv E\{\underline{w}\,\underline{w}^H\}$$

Now $\mathbf{R}_x$, $\mathbf{R}_S$, and $\mathbf{R}_W$ are NxN matrices (where N is the number of anten - nas), and are non-negative definite based on their form. $\mathbf{P}$ is an MxM matrix (where M is the number of incident waves) of rank M, and is, in general, positive definite. $\mathbf{R}_W$ is also in general positive definite. The case where the number of arriving waves is less than the number of antennas (M < N) is of primary con - cern. Since $\mathbf{P}$ is of rank M and $\mathbf{A}$ is of rank N > M, $\mathbf{R}_S$ is of rank M. $\mathbf{R}_x$ and $\mathbf{R}_W$ are assumed to be full rank (rank N).

If $\mathbf{R}_W$ is rewritten as $\lambda\mathbf{R}_b$ for $tr\{\mathbf{R}_b\} = rank\{\mathbf{R}_b\} = N$ and some parameter λ, then $\mathbf{R}_x$ is expressed by:

$$\mathbf{R}_x = \mathbf{R}_s + \lambda\mathbf{R}_b \qquad (2-26)$$

$$\mathbf{R}_x - \lambda\mathbf{R}_b = \mathbf{R}_s$$

Since $\mathbf{R}_s$ is singular (being of dimension N and rank M < N):

$$|\mathbf{R}_s| = |\mathbf{R}_x - \lambda\mathbf{R}_b| = 0$$

Plainly, the above equation can be true only for λ a generalized eigenvalue of the matrix pair ($\mathbf{R}_x$, $\mathbf{R}_b$). That is, λ must be one of the solutions to Equation (2-27):

$$\mathbf{R}_x\underline{e}_n = \lambda_n\mathbf{R}_b\underline{e}_n; \quad n = 1, 2, \ldots, N \qquad (2-27)$$

It is clear from their definitions that $\mathbf{R}_s$, $\mathbf{R}_x$, and $\mathbf{R}_b$ are Hermitian. It can be shown that, for Hermitian $\mathbf{R}_x$ and $\mathbf{R}_b$, with either $\mathbf{R}_x$ or $\mathbf{R}_b$ or both positive definite, and any non-zero vector $\underline{z}$, there exists a non-singular transformation matrix $\mathbf{V}$ such that, when $\underline{r} = \mathbf{V}\underline{z}$:

$$\underline{r}^H\mathbf{R}_x\underline{r} = \sum_{n=1}^{N}\lambda_n|z_n|^2, \quad \text{and } \underline{r}^H\mathbf{R}_b\underline{r} = \sum_{n=1}^{N}|z_n|^2$$

where the λ_n satisfy Equation (2-27) (Hohn, 1964). A similar result holds for the "normal" eigenvalues of $\mathbf{R}_x$. That is, for any non-zero vector $\underline{y}$, there exists a non-singular transformation matrix $\mathbf{U}$ such that, when $\underline{r} = \mathbf{U}\underline{y}$:

$$\underline{r}^H\mathbf{R}_x\underline{r} = \sum_{n=1}^{N}\mu_n|y_n|^2$$

where the μ_n satisfy $|\mathbf{R}_x - \mu_n\mathbf{I}| = 0$. Further, since $\mathbf{R}_x$ is non-negative definite, $\mu_n \geq 0$, all n (> 0 for positive definite $\mathbf{R}_x$). Sylvester's Law of Inertia states that, regardless of the non-singular transformation used to reduce a Hermitian form to the form:

$$g_1|z_1|^2 + g_2|z_2|^2 + \ldots + g_p|z_p|^2 - g_{p+1}|z_{p+1}|^2 - \ldots - g_r|z_r|^2,$$

where the g_i's are all positive, the integers p (the "index" of the matrix) and r (the rank of the matrix) are unchanged. That is, p and r do not change as a result of

using a different non-singular transformation. Thus, since $\mu_n \geq 0$, it follows that $\lambda_n \geq 0$ for all n.

Recalling Equations (2-26) and (2-27), we have:

$$\mathbf{R}_s \underline{e}_n + \lambda \mathbf{R}_b \underline{e}_n = \mathbf{R}_x \underline{e}_n = \lambda_n \mathbf{R}_b \underline{e}_n$$

or:

$$\mathbf{R}_s \underline{e}_n = (\lambda_n - \lambda) \mathbf{R}_b \underline{e}_n$$

That is, the eigenvalues of the matrix $\mathbf{R}_s$ in the metric of $\mathbf{R}_b$ differ from the eigen - values of the matrix $\mathbf{R}_x$ in the metric of $\mathbf{R}_b$ (the λ_n) by λ (from Equation (2-26)) in every case. Since $\mathbf{R}_s$ is non-negative definite, the quantities $\lambda_n - \lambda$ must all be ≥ 0 (using an inertia argument similar to that given above). Therefore, λ in Equation (2-26) must be the minimum of the generalized eigenvalues. Equa - tion (2-26) can thus be rewritten as:

$$\mathbf{R}_x = \mathbf{R}_s + \lambda_{min} \mathbf{R}_b \qquad (2-28)$$

Now $\mathbf{R}_s$ is singular with a zero (normal) eigenvalue of multiplicity K $(= N - M)$. Therefore, the same argument which shows that $\lambda_n \geq 0$ implies that the minimum generalized eigenvalue of $\mathbf{R}_x$ in the metric of $\mathbf{R}_b$ (λ_{min}) must also have multiplicity K. That is, $\lambda_n - \lambda_{min}$ must be 0 for precisely K of the λ_n's. Since:

$$\mathbf{R}_s = \mathbf{A}\mathbf{P}\mathbf{A}^H$$

we have:

$$\mathbf{A}\mathbf{P}\mathbf{A}^H \underline{e}_n = (\lambda_n - \lambda_{min}) \mathbf{R}_b \underline{e}_n$$

Arranging the eigenvalues in descending order gives

$$\lambda_1 \geq \lambda_2 \geq \ldots > \lambda_{min,K} = \lambda_{min,K-1} = \ldots = \lambda_{min,1} = \lambda_{min}.$$

Thus $\mathbf{A}\mathbf{P}\mathbf{A}^H \underline{e}_n = \underline{0}$ for the K eigenvectors associated with λ_{min}, and therefore $\underline{e}_n^H \mathbf{A}\mathbf{P}\mathbf{A}^H \underline{e}_n = (\mathbf{A}^H \underline{e}_n)^H \mathbf{P} (\mathbf{A}^H \underline{e}_n) = 0$. Since $\mathbf{P}$ is positive definite, this in turn im - plies that:

$\mathbf{A}^H \underline{e}_n = \underline{0}$ for the K eigenvectors ($\underline{e}_n$) associated with λ_{min}.

The K eigenvectors associated with λ_{min} are called the "noise subspace" eigenvectors (or just the noise eigenvectors), and the remaining eigenvectors (associated with the non-minimum generalized eigenvalues) are called the "principal" eigenvectors (Marple, 1987). It has just been shown, then, that the noise eigenvectors are orthogonal to the signal vectors (the columns of **A**). That is:

$$\underline{e}^H\underline{a}(\theta) = \underline{a}^H(\theta)\underline{e} = 0; \quad \underline{e} = \underline{e}_{M+1}, \underline{e}_{M+2}, \ldots, \underline{e}_N; \quad \theta = \theta_1, \theta_2, \ldots, \theta_M$$

The MUSIC algorithm uses a direction estimator, $f(\theta)$, formed from the noise eigenvectors as follows:

$$\mathbf{E} \equiv [\, \underline{e}_{M+1} \mid \underline{e}_{M+2} \mid \cdots \mid \underline{e}_N \,]$$

$$g(\theta) = \underline{a}^H(\theta)\mathbf{E}\mathbf{E}^H\underline{a}(\theta)$$

$$f(\theta) = \frac{1}{g(\theta)} = \frac{1}{\underline{a}^H(\theta)\mathbf{E}\mathbf{E}^H\underline{a}(\theta)} \qquad (2-29)$$

Since:

$$g(\theta) = \sum_{n=M+1}^{N} \left[\underline{a}^H(\theta)\underline{e}_n\right]\left[\underline{e}_n^H\underline{a}(\theta)\right] = \sum_{n=M+1}^{N} \left[\underline{a}^H(\theta)\underline{e}_n\right]\left[\underline{a}^H(\theta)\underline{e}_n\right]^*$$

it is clear that (theoretically) $g(\theta) = 0$ and therefore $f(\theta)$ is infinite for values of θ corresponding to the directions of arrival of the M incident waves ($\theta = \theta_1, \theta_2, \ldots, \theta_M$). In practice, $f(\theta)$ as defined in Equation (2-29) exhibits sharp peaks at the arrival directions.

To summarize, the steps in the MUSIC algorithm for angle-of-arrival esti - mation are:

1) Collect sample vectors $\underline{x}$

2) Estimate $\mathbf{R}_x = E\{\underline{x}\,\underline{x}^H\}$ and $\mathbf{R}_w \equiv E\{\underline{w}\,\underline{w}^H\} = \lambda_{min}\mathbf{R}_b$

3) Solve the generalized eigenproblem: $\mathbf{R}_x\underline{e} = \lambda\mathbf{R}_b\underline{e}$

4) Form an estimate, M′, of the number of arriving plane waves, M, by subtracting the multiplicity of the minimum eigenvalue, λ_{min}, from the number of array elements, N

5) Form $\mathbf{E}$ from $\underline{e}_{M'+1}, \underline{e}_{M'+2}, \ldots, \underline{e}_N$

6) Evaluate $f(\theta) = \{\underline{a}^H(\theta)\mathbf{E}\mathbf{E}^H\underline{a}(\theta)\}^{-1}$, and identify M′ peaks at $\theta_1, \theta_2, \ldots, \theta_{M'}$.

7) Form an estimate, $\mathbf{A}'$, of $\mathbf{A}$ as follows:

$$\mathbf{A}' = [\ \underline{a}(\theta_1) \mid \underline{a}(\theta_2) \mid \ldots \mid \underline{a}(\theta_{M'})\ ]$$

8) Estimate the source covariance matrix:

$$\mathbf{P}' = (\mathbf{A}'^H\mathbf{A}')^{-1}\mathbf{A}'^H(\mathbf{R}_x - \lambda_{min}\mathbf{R}_b)\mathbf{A}'(\mathbf{A}'^H\mathbf{A}')^{-1}$$

The principal advantage of the MUSIC algorithm is that it is the most generally applicable high-resolution approach, applying as it does to completely arbitrary array geometry and functioning well in the presence of coherent arriving signals. Its disadvantages are that it requires a great deal of computation, and that its development is not straightforward (although its implementation is, at least conceptually).

For some array geometries, it is possible to drastically reduce the amount of computation required while retaining the high resolution of the MUSIC algorithm. The ESPRIT method of Paulraj, Roy, and Kailath can be applied to arrays consisting of matching element pairs, and does not require a search to determine directions of arrival (Speiser, 1987). ESPRIT is expected to perform as well as or better than MUSIC, its primary disadvantage being the loss of generality concerning array element placement.

L_p Methods

In the approaches described so far, estimates of directions of arrival are based on the L_2 norm; that is, the estimates produced are best in a "least squares" sense (although the quantities whose squares are minimized are different for each method). This dependence on least squares is explicit in some of the approaches (Burg, MFBLP, Maximum Likelihood), and only implicit in others (MUSIC, Fourier). When the noise present is Gaussian, and the system under study is linear, least squares estimates are also maximum-likelihood estimates. Thus, in the linear/Gaussian environment, there is little point in pursuing other than least squares approaches. However, the beamforming problem involves a highly nonlinear model (1-1). Also, it may not be reasonable to suppose that the noise to which the antenna array is subjected is invariably Gaussian. Geary (1947) states, "All texts should state: Normality is a myth, there has never been, and never will be, a normal distribution." Thus, least squares solutions may not be the ideal for the beamforming problem (at least not in all cases).

Yarlagadda, et al. (1985) point out that the L_p approach, where $1 \leq p < 2$, is less sensitive to aberrant noise than the L_2 approach. That is, minimizing a sum of error terms raised to some power (p) between 1 and 2 can produce a more robust estimator than one based on a minimum sum of squares. For example, since L_2 emphasizes all data equally, while L_1 is robust against outliers, L_1 generally produces superior results for impulsive noise. L_p measures can be used effectively to match the signal in a statistical sense. That is, if the density function has the form:

$$e^{-|r|^p}$$

i.e. a p-Gaussian function, then L_p methods give maximum likelihood estimates. For example, p = 1 corresponds to the Laplacian distribution and the L_1 measure gives maximum likelihood estimates for this case.

The L_p approach has been applied in a variety of areas including speech processing, seismic analysis, and geophysics (Denoel and Solvay, 1985) (Taylor, et. al., 1979) (Claerbout and Muir, 1973). On the other hand, very little work has been done in applying L_p methods to the direction of arrival problem. In fact, all the traditional approaches to the beamforming problem have been oriented toward obtaining L_2 solutions. The research described herein involves the use of the L_p, $1 \leq p \leq 3$, approach to estimation of directions of arrival and the source covariance matrix, **P**.

CHAPTER III

L_p-NORM ESTIMATION APPROACHES

The estimate of **P** given by step 8 in Chapter II, above, is a least-squares (L_2) estimate. The direction estimators described in Chapter II (including MU - SIC) are L_2 estimators as well. As discussed in Chapter II, the research ad - dressed herein applies the L_p norm to the direction of arrival problem.

Due to its ability to accommodate arbitrary array geometries, the MUSIC algorithm was chosen as the starting point for application of L_p norm based techniques. A number of L_p norm estimation algorithms exist (some of which are discussed in Chapter V), the most general being the Iteratively Reweighted Least Squares (IRLS) method.

Byrd and Pyne (1979) develop global convergence properties of the IRLS algorithm as applied to estimating a vector $\underline{f}$ given the linear model:

$$\underline{x} = \mathbf{A}\underline{f} + \underline{w};$$

where $\underline{x}$ is an N-element output vector, **A** is an NxM matrix of rank M, $\underline{f}$ is an M-element vector of unknown parameters, and $\underline{w}$ is an N-element zero-mean noise vector. The IRLS algorithm repeatedly evaluates:

$$\underline{f}_{t+1} = [\mathbf{A}^H\mathbf{W}(\underline{f}_t)\mathbf{A}]^{-1}\mathbf{A}^H\mathbf{W}(\underline{f}_t)\underline{x} \qquad (3-1)$$

until $\underline{f}$ converges. The weighting matrix **W** depends on the loss function to be minimized by the estimation. If **W** = **I**, Equation (3-1) clearly produces the L_2 es- timate of $\underline{f}$; i.e. the loss function minimized is the sum of the squares of the esti - mation residuals.

In the general form of the IRLS algorithm, **W** is a diagonal matrix with en - tries:

$$w_{nn} = w_n = w(r_n) = \psi(r_n)/r_n = \rho'(r_n)/r_n \qquad (3-2)$$

In Equation (3-2), $r_n = \underline{a}_n\underline{f} - x_n$; $\underline{a}_n$ is the $n^{\underline{th}}$ row of **A**; x_n is the $n^{\underline{th}}$ element of $\underline{x}$; and the objective function ρ is a positive, even, differentiable function. This defi - nition of **W** results in minimization of the loss function J, as defined below:

$$J = \sum_{n=1}^{N} \rho(\underline{a}_n\underline{f} - x_n) \qquad (3-3)$$

That iteration of Equation (3-1) using **W** as defined in Equation (3-2) mini - mizes J, above, is shown in Appendix A. Global convergence of the IRLS algo - rithm is guaranteed for objective functions ρ such that:

(1) $\rho(r)$ is a differentiable, symmetric, positive function nondecreasing in $|r|$,

(2) $\rho(r) \rightarrow \infty$ as $|r| \rightarrow \infty$,

(3) $w(r) = \rho'(r)/r$ is nonincreasing in $|r|$, and

(4) $w(r) = \rho'(r)/r$ is bounded for all r.

Application to the L_p minimization problem implies the use of the objec - tive function $\rho(r) = |r|^p$. This function, however, gives rise to unbounded w(r) as $r \rightarrow 0$ (violating condition 4). Huber and Dutter (1974) suggest the function:

$$w(r) = |r|^{p-2}; \quad |r| > \varepsilon \qquad (3-4)$$
$$= \varepsilon^{p-2}; \quad |r| \leq \varepsilon$$

for some small positive number ε. This function has been shown to give results which differ only negligibly from exact L_p solutions. Initialization of the IRLS al - gorithm is generally accomplished by starting with the L_2 estimate of $\underline{f}$. How - ever, since the loss function J, and not the initial estimate, determines the nature of the estimate produced, other initial values of $\underline{f}$ could reasonably be used.

The weight function (3-4) meets the convergence conditions for values of p between 1 and 2, thereby assuring global convergence of the IRLS algorithm to (approximate) L_p solutions for $1 \leq p \leq 2$. Additionally, local convergence of the IRLS algorithm using the function (3-4) is guaranteed for values of p up to 3 (Fletcher, et. al., 1971). Of course, since only local convergence is guaranteed for this case, the freedom of choice of initial $\underline{f}$ may be lost.

It may be that L_p estimates with $p > 2$ are of use in some situations; however, such estimates would emphasize outliers more than L_2 estimates, so that the resulting estimators are likely to lack robustness. For $p < 1$, the problem space is not a normed linear space (the triangle inequality does not apply), so that the analysis breaks down. So, the emphasis during this research has been on p values between 1 and 2, with some effort directed toward values between 2 and 3.

L_p Estimates of the Source Covariance Matrix

As mentioned in Chapter II, one objective of direction of arrival (DOA) estimation is to produce an estimate of the source covariance matrix, **P**. At least two distinct approaches to this problem are possible. The first involves estimating the signal vector samples and using those estimates to estimate **P**. The second approach is to attempt to estimate **P** directly from the problem parameters, bypassing estimation of the signal vector samples.

The second approach has the advantage of potentially requiring fewer computations, while the first approach leads to an algorithm for which Byrd and Pyne's (1979) analysis guarantees global convergence.

Enumeration Method

Given the model $\underline{x} = \mathbf{A}\underline{f} + \underline{w}$ from Chapter I, the L_p estimate of the signal vector sample ($\underline{f}_s$) which gives rise to the s^{th} array output sample ($\underline{x}_s$) is (from (3-1) above):

$$\underline{f}_s = [\mathbf{A}^H\mathbf{W}(\underline{f}_s)\mathbf{A}]^{-1}\mathbf{A}^H\mathbf{W}(\underline{f}_s)\underline{x}_s \qquad (3-5)$$

This expression is correct, of course, only when IRLS convergence has been achieved (the value of $\underline{f}_s$ has stopped changing).

The "enumeration" method uses the L_p estimates of all the signal vector samples to form an estimate of **P**. Recalling the definition of **P**, we may write:

$$\mathbf{P} \equiv E\{\underline{f}\,\underline{f}^H\}$$

$$\hat{\mathbf{P}} = \frac{1}{S}\sum_{s=1}^{S}\underline{f}_s\underline{f}_s^H \qquad (3-6)$$

where S total samples are used to form the estimate.

One potential disadvantage of this estimation approach is that, since neither the estimates $\underline{f}_s$ nor the summation in (3-6) involves noise terms directly, no use is made of the known or assumed noise characteristics. Of course, this may not always be a problem, since no information about the noise characteristics may be available in a particular application. On the other hand, the "sum of powers" method, below, does take advantage of any available knowledge of noise parameters.

Sum of Powers Method

The estimate (3-1) can be improved by removing the noise from the observation vector, if the noise is known, as follows:

$$\underline{f}_\infty = [\mathbf{A}^H\mathbf{W}_\infty\mathbf{A}]^{-1}\mathbf{A}^H\mathbf{W}_\infty(\underline{x} - \underline{w}) \qquad (3-7)$$

where the ∞ subscript denotes the values produced upon IRLS convergence. Substituting (3-7) into the definition of **P**, and assuming that **W** is Hermitian and will be evaluated in a way that does not depend on $\underline{f}$, one obtains:

$$\begin{aligned}\mathbf{P} &= E\{\, \underline{f}\,\underline{f}^H \,\} \\ &= E\{\, [\mathbf{A}^H\mathbf{W}_\infty\mathbf{A}]^{-1}\mathbf{A}^H\mathbf{W}_\infty(\underline{x}-\underline{w})(\underline{x}-\underline{w})^H\mathbf{W}_\infty\mathbf{A}[\mathbf{A}^H\mathbf{W}_\infty\mathbf{A}]^{-1} \,\} \\ &= [\mathbf{A}^H\mathbf{W}_\infty\mathbf{A}]^{-1}\mathbf{A}^H\mathbf{W}_\infty E\{\, (\underline{x}-\underline{w})(\underline{x}-\underline{w})^H \,\}\mathbf{W}_\infty\mathbf{A}[\mathbf{A}^H\mathbf{W}_\infty\mathbf{A}]^{-1} \\ \mathbf{P} &= [\mathbf{A}^H\mathbf{W}_\infty\mathbf{A}]^{-1}\mathbf{A}^H\mathbf{W}_\infty(\mathbf{R}_x-\lambda_{min}\mathbf{R}_b)\mathbf{W}_\infty\mathbf{A}[\mathbf{A}^H\mathbf{W}_\infty\mathbf{A}]^{-1} \qquad (3-8)\end{aligned}$$

Based on the form of (3-8), an iterative procedure for estimating **P** can be proposed that is similar to the iteration of (3-1):

$$\mathbf{P}_{t+1} = (\mathbf{A}^H\mathbf{W}_t\mathbf{A})^{-1}\mathbf{A}^H\mathbf{W}_t(\mathbf{R}_x-\lambda_{min}\mathbf{R}_b)\mathbf{W}_t\mathbf{A}(\mathbf{A}^H\mathbf{W}_t\mathbf{A})^{-1} \qquad (3-9)$$

where $\mathbf{W}_t$ is a weight matrix to be determined from the minimization process, and Equation (3-9) is iterated (starting with the L_2 solution) until some conver - gence criterion is satisfied.

For the above approach to be valid, the weight matrix $\mathbf{W}_t$ must be Hermi - tian and must not depend directly on the individual samples of the source signal vector, $\underline{f}_s$. On the other hand, it is clear that estimates of the individual source signal vector samples must be computed based in some way upon $\mathbf{W}_t$, since such estimates are the only source of residuals to be used in computing the weight matrix. Given these constraints, a reasonable approach is to make $\mathbf{W}_t$ a diagonal matrix based on a loss function formed from the sum (across samples) of estimation residuals raised to the p^{th} power:

$$r_n = \left\{ \sum_{s=1}^{S} \left| \underline{a}_n \underline{f}_s - x_{n,s} \right|^p \right\}^{\frac{1}{p}} \qquad (3-10)$$

$$\rho(r_n) = |\, r_n \,|^p$$

$$w_n(r_n) = \frac{\rho'(r_n)}{r_n} = p\frac{r_n^{p-1}}{r_n} = pr_n^{p-2}$$

$$= p\left\{\sqrt[p]{\sum_{s=1}^{S}\left|\underline{a}_n^{\underline{t}}\underline{f}_s - x_{n,s}\right|^p}\right\}^{p-2} \qquad (3-11)$$

where $x_{n,s}$ is the $n^{\underline{th}}$ element of the $s^{\underline{th}}$ array output sample. In practice, the multiplicative constant p can be ignored in evaluating w_n, since it is common to all elements of **W** and therefore cancels out in (3-1). Further, convergence is generally improved by normalizing the weights to a maximum value of 1. Additionally, since only the final value of **W** is needed in Equation (3-8), the iteration process need involve only equations (3-1) and (3-11), with the final (after convergence) value of **W** being used to form the L_p estimate of **P**.

The sum of powers method (Equations (3-10) and (3-11)) has the advantage that knowledge of noise characteristics is used in the **P** estimate (through the use of $\lambda_{min}\mathbf{R}_b$); however, it has the disadvantage (compared to the enumeration method) that the analysis of Appendix A no longer applies (since an ensemble of vectors, rather than a single vector, is being estimated). On the other hand, experimental results using the sum of powers method indicate that it can be expected to converge at least as quickly as the enumeration method. For example, in the experiments described in Chapter IV, the average iteration count for the enumeration method was approximately 34, while the average iteration count for the sum of powers method was approximately 24, for a value of p = 1.

L_1 Selected Rows Method

In a system of N equations with M unknowns, where M < N, L_1 solutions produced by the simplex algorithm in general satisfy M of the original equations

exactly. For the IRLS method, it is possible to produce a result which likewise satisfies M of the original equations exactly. Given this fact, it is reasonable to develop an approach which uses the exactly-sastified entries in (1-6) to form an estimate of **P** (for the L_1 case).

For example, the "sum of powers" approach described above produces a single N-element residuals vector ($\underline{r}$) as a result of the estimation process. Fur - ther, this vector is produced before an estimate of **P** is computed. An estimate of **P**, then, could be computed by solving the signal vector estimation problem for the M rows of **A** which correspond to the smallest M entries in the final residuals vector. This approach will, in most cases, yield an exact L_1 solution to the selected M rows (Scales and Treitel, 1987). The desired estimate of **P** would then be formed using the selected M rows of **A** and **W**, and the corresponding M^2 elements of **R**, which correspond to the smallest M entries in the final residuals vector, as shown in Figure 2. A similar approach could be applied to the "enumeration" estimation method by selecting entries in **A**, **W**, and **R** based, for instance, on the frequency (across the sample ensemble) with which a given set of M rows produces minimum residuals.

L_p Estimates of Directions of Arrival

As stated in Chapter II, MUSIC is a least squares, or L_2, estimator of di- rections of arrival. In order to take full advantage of the L_p norm, it is desirable to develop an L_p direction of arrival estimator based on a combination of the principals reflected in MUSIC and those embodied by the IRLS algorithms above. This provides an L_p solution to the complete DOA problem, rather than just an L_p-based modification to the last phase (estimation of **P**) of a solution based on L_2 principals (MUSIC).

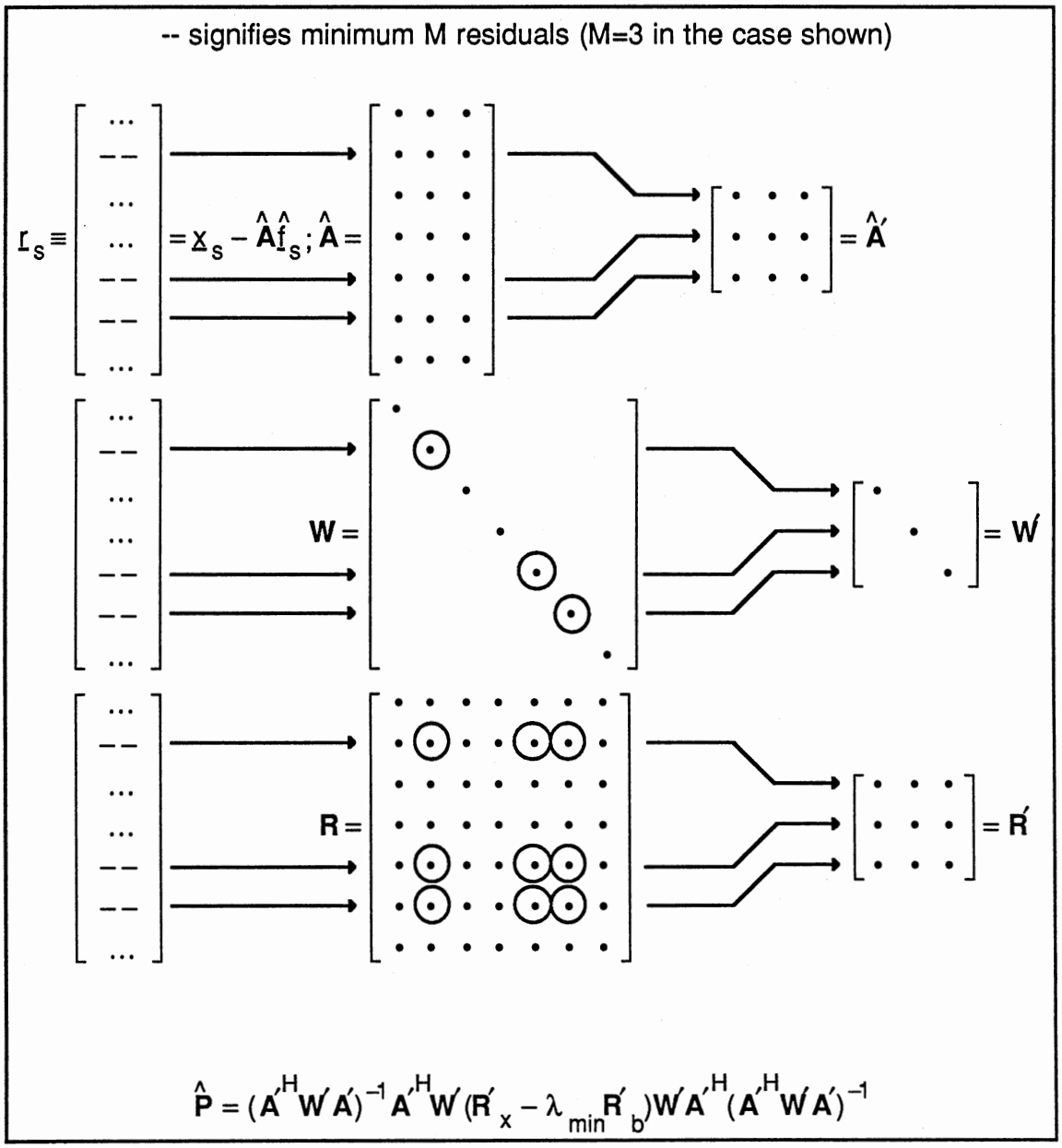


Figure 2. L_1 Selected Rows Method

The form of an L_p DOA estimator is suggested by the expression for MU - SIC's DOA estimator (Equation (2-29)). The MUSIC algorithm estimates wave directions of arrival by locating the maxima of the function:

$f(\theta) = \{\underline{a}^H(\theta)\mathbf{E}\mathbf{E}^H\underline{a}(\theta)\}^{-1}$, or, equivalently, the minima of the function:

$g(\theta) = \underline{a}^H(\theta)\mathbf{E}\mathbf{E}^H\underline{a}(\theta)$.

The expression for $g(\theta)$ can be rewritten in the following form:

$$g(\theta) = \sum_{n=M+1}^{N} \left[\underline{a}^H(\theta)\underline{e}_n\right]\left[\underline{e}_n^H\underline{a}(\theta)\right]$$

$$= \sum_{n=M+1}^{N} \left[\underline{a}^H(\theta)\underline{e}_n\right]\left[\underline{a}^H(\theta)\underline{e}_n\right]^*$$

$$= \sum_{n=M+1}^{N} \left|\underline{a}^H(\theta)\underline{e}_n\right|^2$$

Clearly, MUSIC DOA estimates are just those angles which minimize a sum of squares of magnitudes of the dot products shown above. Since this is the classic form of an L_2 estimate (minimization of a sum of magnitudes squared), the proposed L_p DOA estimates minimize a sum of magnitudes raised to the p^{th} power, as follows:

$$g_p(\theta) = \sum_{n=M+1}^{N} \left|\underline{a}^H(\theta)\underline{e}_n\right|^p \qquad (3-12)$$

Some observations about (3-12) are in order. First, the angles θ which minimize g_p are plainly L_p estimates of direction of arrival, in the sense that those angles minimize the p-norm of the difference between the array manifold ($\underline{a}(\theta)$) and the signal subspace ($\underline{e}_n$, n = 1, 2, ..., M). Put another way, since sub - space $\underline{e}_n$ (n = M+1, M+2, ..., N) forms a basis for the noise subspace, mini - mizing the sum of (3-12) minimizes (in the p-norm sense) the projection of the

vector $\underline{a}(\theta)$ onto the noise subspace. Further, since the noise subspace is the orthogonal complement of the signal subspace, the distance between the array manifold and the signal subspace is consequently minimized in the p-norm sense. That is, minimizing g_p selects the angles that best fit the observed data, in the p-norm sense, subject to the constraints of array geometry.

Second, no iteration is required to determine solutions of (3-12), as is necessary in the case of estimating **P**. This implies that a **P** estimate based on L_p DOA estimates (which might be considered a form of L_p estimate of **P**) can be obtained with essentially no more computation than is required to obtain an L_2 estimate of **P** from the MUSIC DOA estimates. Also, since no convergence issues are involved in minimizing g_p, one could obtain L_p DOA estimates (and the associated L_p **P** estimate) for any value of $p > 0$ (although the interpretation for values of p outside the range $1 \leq p \leq 3$ might be open to question).

Third, since g_p provides a measure of the p-norm distance between the array manifold and the signal subspace, it gives at least a relative indication of the success of the estimation process. For example, if additional data is taken following an estimation step, and incorporation of the additional data results in lower values of g_p, it might reasonably be claimed that the estimates obtained using the additional data are in some sense "better" than the original estimates. Of course, such a qualitative discussion as that given requires considerable experimental or theoretical support if useful measures of goodness-of-fit are to be developed.

Finally, for the case of $N = M + 1$ (the number of arriving waves is one less than the number of antenna elements), the angles which minimize g_p are unchanged by varying p. This is because the sum in (3-12) consists of one term for this case, and, of course, the minimum of a single term occurs at the same point regardless of the exponent to which that term is raised (as long as the ex-

ponent is positive). This implies that, when estimation of the directions of arrival of one fewer waves than array elements is attempted (a condition which we will call "array saturation"), L_p DOA estimates are exactly the same as L_2 estimates, regardless of the (positive) value of p used. In this case the noise subspace consists of a line in N-space passing through the single noise eigenvector $\underline{e}_N$. Therefore, any local minimum of the projection of a vector (particularly the array manifold vector) onto the noise subspace (a line) can occur in only one direc - tion, regardless of the norm used. When array saturation occurs, therefore, L_2 DOA estimates are the only ones of interest (although L_p estimates of **P** may be of value). On the other hand, iterative DOA estimation (discussed later) may provide improved DOA resolution even in the presence of array saturation.

Noise Aspects

As mentioned in Chapter II, the L_2 norm is the generally accepted basis for estimation techniques, and it is the starting point for the development of L_p techniques. It is also the only approach necessary for linear systems in the presence of Gaussian noise. For the direction of arrival problem, however, the basic model, as previously stated, is not linear (although the model on which the eigenvector decomposition is based is, of course, linear).

In addition to the nonlinearity aspect of the problem, it is not reasonable to suppose that all noise to which an antenna array is subjected will necessarily be Gaussian (indeed, as discussed in Chapter II, it may be unreasonable to as - sume that Gaussian noise is present in <u>any</u> estimation environment). As an example, consider the case of an array in the presence of a jamming signal. It is highly unlikely that the jammer's modulation envelope resembles Gaussian noise, and yet it might be desirable to treat the jammer as a noise source rather than as an additional signal (to facilitate analysis of the other signals of interest,

for instance). If the jammer is in fact emitting periodic pulses of short duration and high intensity, the signal received by the array could resemble impulsive noise, rather than Gaussian noise.

Another example of impulsive noise would be bursts of static either received by the array or generated within the receivers themselves. Other noise distributions are also possible; for example, continuous static might give rise to a uniform noise distribution. Also, even if the underlying noise sources are Gaussian, the samples might not reflect this fact (due, for example, to small sample size).

Since non-Gaussian noise is a distinct possiblity in the DOA determination problem, the use of other than the L_2 norm seems a potential area of benefit. For example, the L_1 norm gives generally superior (to L_2) results in the presence of impulsive noise. Schroeder (1985) demonstrated that use of the L_p norm, $1 \leq p < 2$ permitted resolution of more closely spaced spectral components of a signal in non-Gaussian noise than did the L_2 norm. Figure 3 provides some illustrations of the resolution improvement (over L_2-norm performance) possible when the L_1 norm is used rather than the L_2 norm in the linear prediction approach to spectral estimation used by Schroeder.

The research described herein considered Gaussian, uniform, and impulsive noise distributions. Results of DOA estimation using the L_1 norm in the presence of impulsive noise were in general superior to results using the L_2 norm, as expected (Chapter IV).

Adaptive Methods

As mentioned above, the value of g_p yields some information concerning the degree of success of the DOA estimation process. Other measures of algorithm performance can also be computed. For example, Schmidt points out that

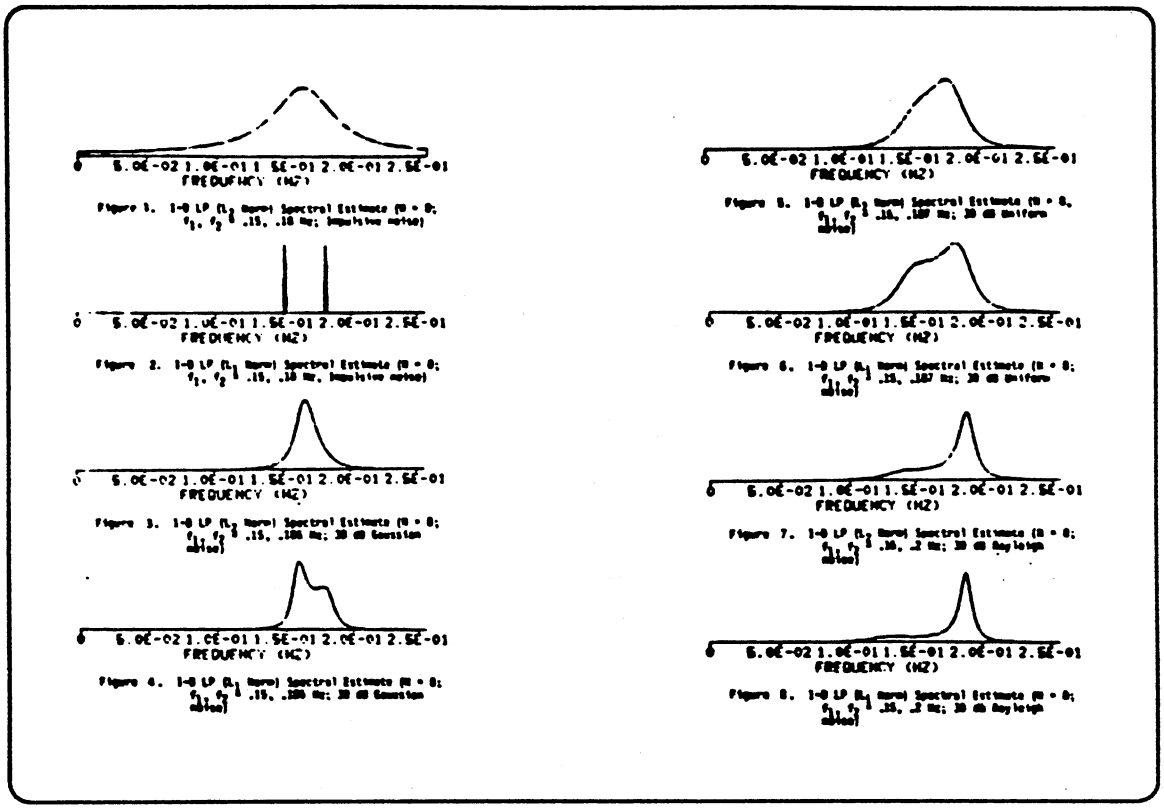


Figure 3. Resolution of Spectral Components in Noise (Schroeder, 1985)

it is possible for the MUSIC algorithm to indicate the presence of more arriving waves than can be resolved due to wave proximity. That is, if two waves are very closely spaced, it is possible for the MUSIC algorithm to be able to resolve the direction of arrival of only one of them, but still provide an indication that more are present.

Two estimates of the number of waves arriving at an antenna array are given in Equations (3-13) and (3-14). Schmidt (1981) suggests a χ^2-based likelihood ratio test for the number of arriving waves, as shown in Equation (3-13):

$$LRT(M) = S(N - M)\ln \frac{\lambda_a}{\lambda_g} \approx \chi^2_d \qquad (3-13)$$

$$d \equiv \frac{1}{2}[(N - M) - 1][(N - M) + 2]$$

where:

S = number of antenna array output samples taken.

N = number of antenna array elements.

M = number of arriving waves hypothesized.

λ_a = arithmetic mean of the smallest N – M generalized eigenvalues.

λ_g = geometric mean of the smallest N – M generalized eigenvalues.

An alternate estimate, which provides better results in some situations, is advanced by Marple (1987). In this approach, the number of arriving waves is estimated as the value which produces a minimum in the Akaike Information Criterion (AIC) as modified by Wax and Kailath:

$$AIC(M) = (N - M)\ln\left(\frac{\frac{1}{N-M}\sum_{m=M+1}^{N}\lambda_m}{\prod_{m=M+1}^{N}\lambda_m^{-(N-M)}}\right) + M(2N - M) \qquad (3-14)$$

where N and M are as defined for Equation (3-13), and λ_m is the m^{th} generalized eigenvalue (λ_1 being the largest and λ_N the smallest).

The above expressions provide estimates only - there is no guarantee that either expression will produce an accurate incoming-wave count in a given situation. The question of how many waves are being received at an antenna array (or how many spectral components are present in a noisy medium) is one of continuing interest. However, since the research described herein addresses the direction of arrival estimation process itself, the number of arriving waves is assumed to be known or estimated in advance.

Given a situation in which more impinging waves can be detected than can be located, it is important to investigate approaches which make use of per - formance measures of some sort to alter the behavior of the overall estimation process. Lansford (1988) investigated the use of kurtosis as a performance measure in the application of linear predictive models to the speech recognition problem. The nature of the measures used, as well as the specific modifications to be made to the algorithms, of necessity depend heavily on the application to be supported. Since each application involves its own engineering trade-offs, a modification that would constitute an improvement in performance in one appli - cation might actually be a degradation in another.

As an example, consider the jammer problem mentioned above. In one application, the DOA and signal characteristics of the jammer might be of in - terest, in which case the jammer should be considered an additional signal source. On the other hand, in a different application it might be desirable to treat the jammer as a noise source, in which case determination of its DOA and signal characteristics is unneccesary and might even interfere with problem solution.

Although any adaptive method is subject to application constraints, some can be discussed which would likely find use in a number of different scenarios. Such techniques as iterative revision of eigenvectors or DOA estimates fall into this category.

Iterative Direction of Arrival Revision

The DOA spectrum produced using (2-29) typically exhibits a number of peaks corresponding to estimated directions of arrival. In the case of multiple closely spaced arriving waves, fewer peaks than arriving waves may appear.

An example is shown in Chapter IV wherein two arriving waves at ±30° effectively suppress a third peak which should appear at 0°.

In such situations, an approach whereby the effects of previously located waves could be removed from the DOA spectrum might permit resolution of additional spectral peaks. Such a "spectral subtraction" can be accomplished by forming estimates of the array phase shift matrix (**A**) and the source signal co - variance matrix (**P**) for the wave(s) to be removed from the spectrum, subtracting the product $\mathbf{APA}^H$ from the original array output covariance matrix ($\mathbf{R}_x$), and forming a new eigenvalue problem from the remainder.

Unfortunately, the difference mentioned above is almost never positive semidefinite, which of course implies that it is not a valid covariance matrix. This can be explained by noting that, since only a subset of the arriving waves has been identified, the diagonal elements of the **P** estimate (which are estimates of arriving wave power) are very likely larger than the corresponding actual elements in **P**. Thus, the term $\mathbf{A'P'A'}^H$, where $\mathbf{A'}$ and $\mathbf{P'}$ are the estimates of **A** and **P**, respectively, tends to "overcompensate" for the waves for which the estimates were made, producing negative eigenvalues in the difference.

In order to eliminate the undesirable effect of producing negative eigen - values, some real fraction of the product estimate can be subtracted from the observation covariance matrix, as follows:

$$\mathbf{R}_x' = \mathbf{R}_x - \gamma\mathbf{A'P'A'}^H;\ 0 < \gamma \leq 1 \qquad (3-15)$$

Naturally, the question arises, "What value should be used for γ?" Since positive semidefiniteness is required of any covariance matrix (including $\mathbf{R}_x'$), the optimum γ is the maximum value for which the difference (3-15) remains positive semidefinite. In order to determine this optimum value in a theoretical sense, it is necessary to express all principal minors of the difference (3-15) in

terms of γ. That is, the formation and solution of a polynomial in γ of at least N^{th} order (where N is the number of antenna elements), for an arbitrary matrix pair $\mathbf{R}_x$, $\mathbf{A'P'A'}^H$, "suffices" to determine the optimum value of γ. Of course, this in - volves *at least* as much computation as did the original DOA estimation pro - cess itself!

This being the case, an alternative approach to determination of the opti - mum γ value is proposed. Although entirely empirical in nature, the alternative approach gives the optimum γ value to whatever precision is desired. The method implements a numerical binary search over the interval (0,1] for the largest acceptable value of γ. That is, a value of 1 is tried, followed by 0.5, fol - lowed by either 0.25 or 0.75, and so on until the desired resolution is achieved. The value of the next "guess" at each stage is determined based on whether positive semidefiniteness was obtained at the previous stage. This approach requires a positive semidefiniteness test at each step. Although this can repre - sent a considerable computational burden, the polynomial solution effort is avoided, which in itself is expected to reduce overall computational load. Ad - ditionally, the procedure can be stopped at any point, providing for an ad - justable processing time (at the expense of low γ resolution for short processing times). In practice, relatively little resolution for γ may be required, since DOA resolution can be expected to improve to some extent with <u>any</u> reduction in dominant spectrum peaks. The extent to which such peak reduction can be ex - pected to help depends strongly, of course, on the actual data involved. Several examples are provided in Chapter IV of the iterative DOA revision techniques discussed above.

A second concern in implementing the spectral subtraction described above is the danger of introducing spurious peaks. As an illustration of how this is possible, consider the operation of a band-pass filter with white noise applied

to its input. Since white noise contains energy at all frequencies, the filter will produce an output signal at (about) its center frequency, even though no signal was present at its input. A similar process is possible in digital signal proces - sing, where false periodicities can be established using successive smoothing and differencing operations. Yarlagadda and Hershey (1987) point out that "... the creation of a false periodicity in this manner is related to the Slutsky-Yule effect that predicts the period of the false oscillation based upon the smoothing and differencing operations."

Other Adaptive Approaches

Another iterative procedure suggested by the MUSIC eigenvalue prob - lem (Equation 2–27) involves the estimation of λ_{min} and $\mathbf{R}_b$. The usual esti - mates of these parameters are formed from noise samples (assuming they are available) as follows:

$$\mathbf{R}_w = \sum_{s=1}^{S} \underline{w}_s \underline{w}_s^H \qquad (3-16)$$

$$\lambda_{min} = tr\{ \mathbf{R}_w \} / N$$

$$\mathbf{R}_b = \mathbf{R}_w / \lambda_{min}$$

Obviously, the accuracy of the above values of λ_{min} and $\mathbf{R}_b$ depends on the number of noise samples available and how representative of the underly - ing random process the available noise samples are. The effect of inaccuracies in the estimates (3-16) is to produce an $\mathbf{R}_b$ value which, when used in (2–27), yields a valule for λ_{min} different from the above estimate. An adaptive approach which might improve the accuracy of the eventual eigenvalue problem solution is to compute the average of the λ_{min} estimate from (3-16) and the actual λ_{min} from (2–27), use the average to compute a new $\mathbf{R}_b$ estimate (using (3-16)),

solve (2–27) again, and so on until the λ_{min} estimate from (3-16) is acceptably close to the estimate from (2–27). This approach might be particularly useful in the case of small sample size, where an $\mathbf{R}_b$ estimate is desired with relatively few $\underline{w}_s$ samples.

In another adaptive procedure, iterative revision of the eigenvectors for DOA estimation is proposed (Fuhrmann, 1987). This approach employs an iterative singular value decomposition algorithm to periodically update esti - mates of the signal subspace vectors, and is most directly applicable to the case of time-varying directions of arrival. The subspace vector update is interleaved with both acquisition of new observation vector samples and DOA spectrum evaluation, the interleave ratios being determined by the application. The itera - tive DOA revision technique described above could be included directly into this procedure, allowing the decision concerning the number of waves to identify to be made dynamically.

CHAPTER IV

EXPERIMENTAL RESULTS

Several procedures described in previous chapters were implemented on a general purpose computer, and tested using simulated data sets. The approaches tested were the L_p DOA estimation algorithm, the iterative DOA revision approach, and the enumeration and sum of powers **P** estimation procedures.

To solve the generalized eigenvalue problem (2-27), a generalized singular value decomposition algorithm due to Van Loan was employed (Appendix B) (Van Loan, 1985). Further, due to the similarity of the form of the matrix term in (3-1) to the Moore-Penrose psuedo-inverse, a modified Moore-Penrose psuedo-inverse algorithm (Appendix C) was employed to compute L_p estimates of source signal vector samples ($\underline{f}_s$) (Graybill, 1969).

The program described in Appendix D was used to generate simulation data for arrays of arbitrary geometry illuminated by arbitrary wave patterns in the presence of either correlated or uncorrelated noise. Noise generators with fixed, uniform, Gaussian, and impulsive distributions were supported.

Effects of Noise Power on DOA Resolution

Figures 4 through 8 illustrate the effects of impulsive noise on the DOA spectra produced by the L_p MUSIC algorithm. Figure 4 shows the antenna array and impinging wave pattern simulated. Figures 5 through 8 show that, as

SNR is dropped progressively from 20db to 0db, DOA resolution fails for de - creasing values of p, until (Figure 8) p = 1 is the only value for which resolution is obtained. The impulsive noise was applied at an average of every 20 samples in the experiments shown in Figures 5 through 7, and 25 samples in Figure 8.

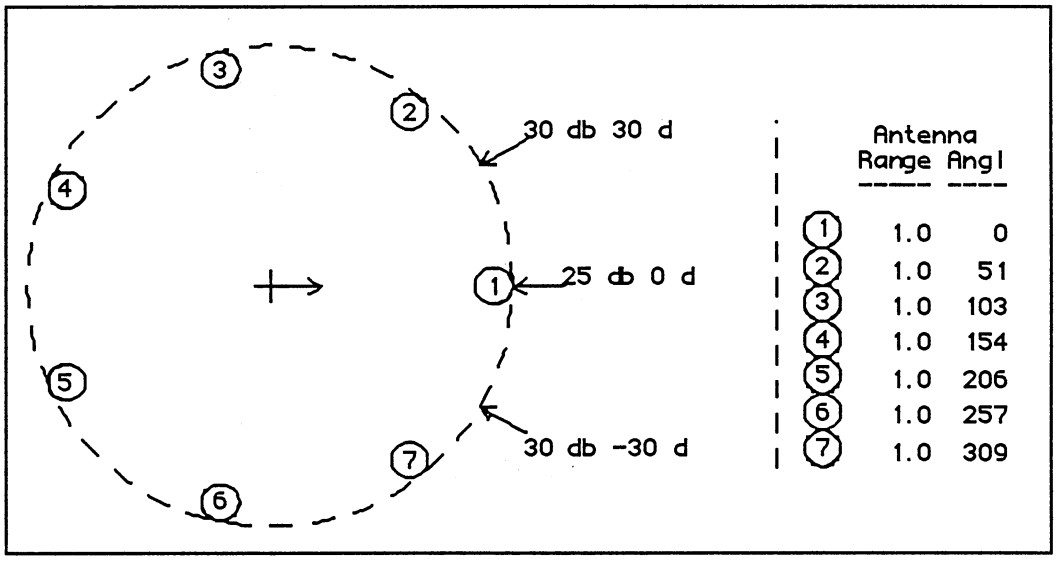


Figure 4. Antenna Array Geometry, Configuration #1

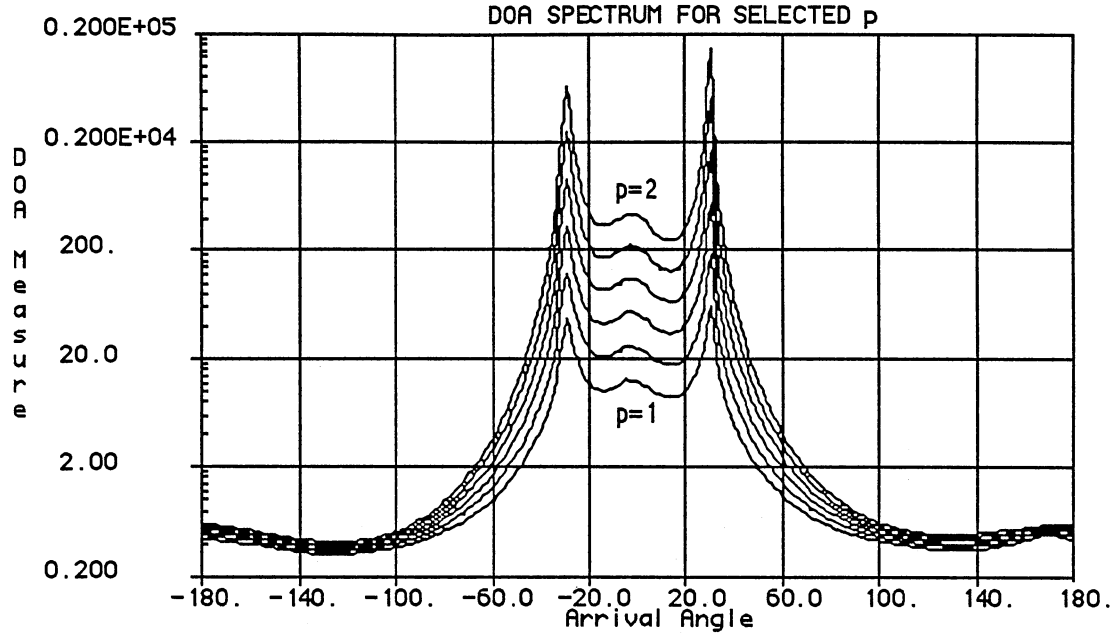


Figure 5. DOA Spectra, SNR = 20db, 20-Sample Interval

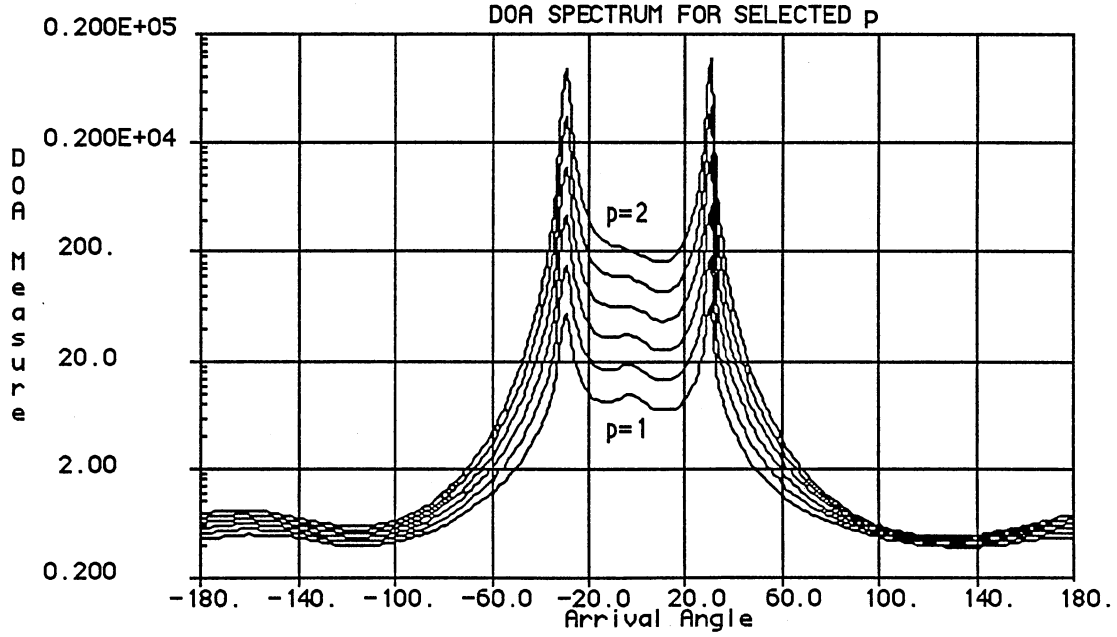


Figure 6. DOA Spectra, SNR = 15db, 20-Sample Interval

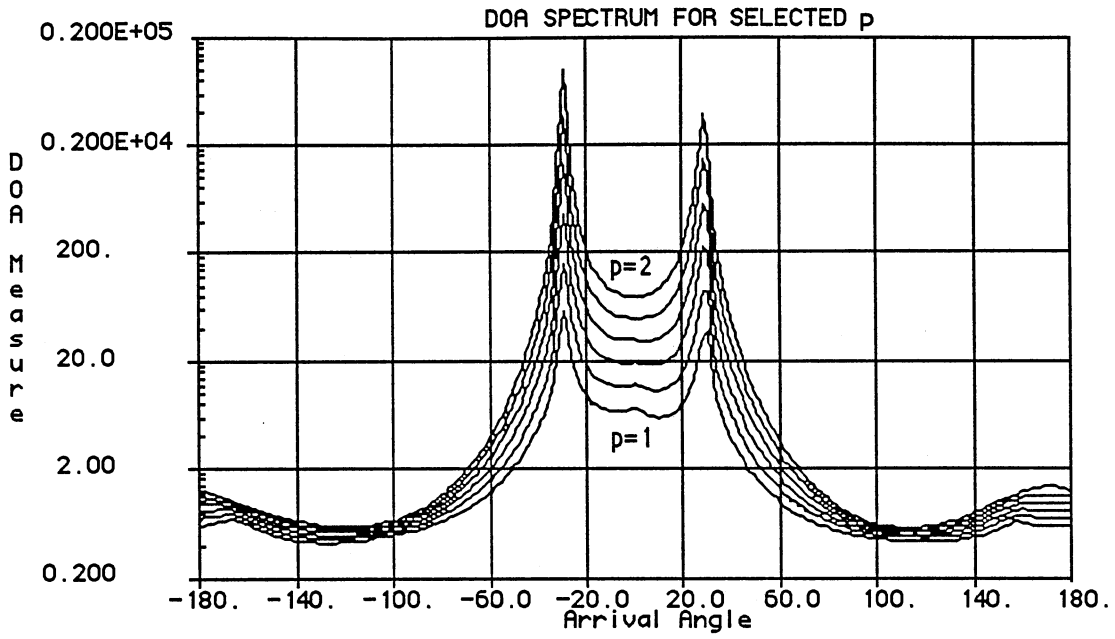


Figure 7. DOA Spectra, SNR = 10db, 20-Sample Interval

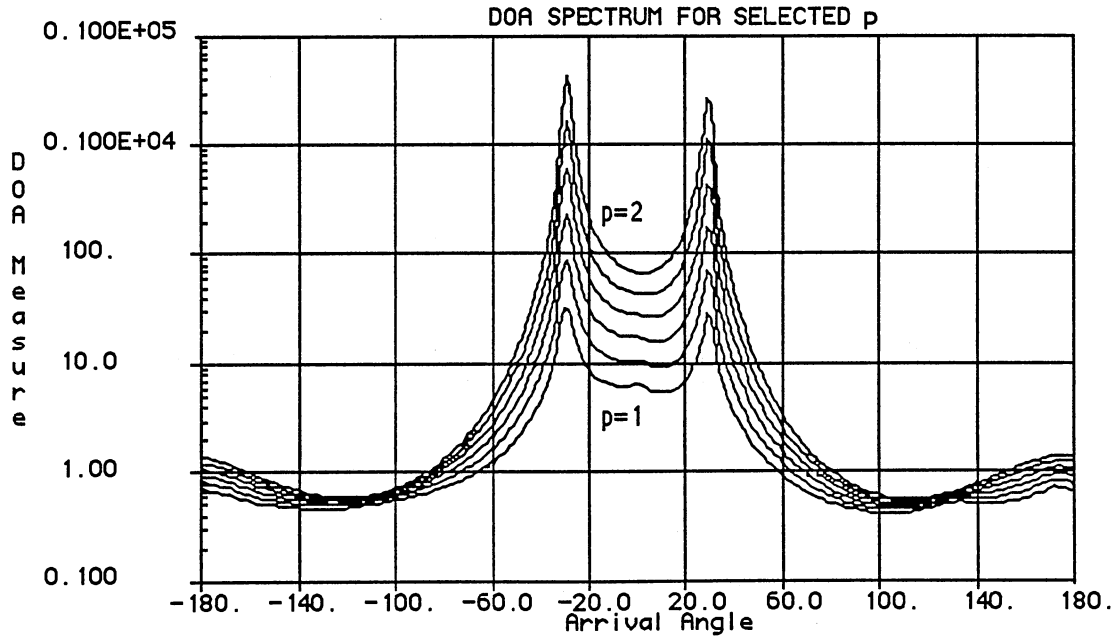


Figure 8. DOA Spectra, SNR = 0db, 25-Sample Interval

Figures 9 and 10 illustrate DOA spectra for values of p between 2 and 3. Noise strengths (relative to maximum incoming signal strength) and average impulse intervals were –20db each 20 samples for Figure 9 and 0db each 20 samples for Figure 10. Arrays as illustrated in Figure 4 were simulated in both cases.

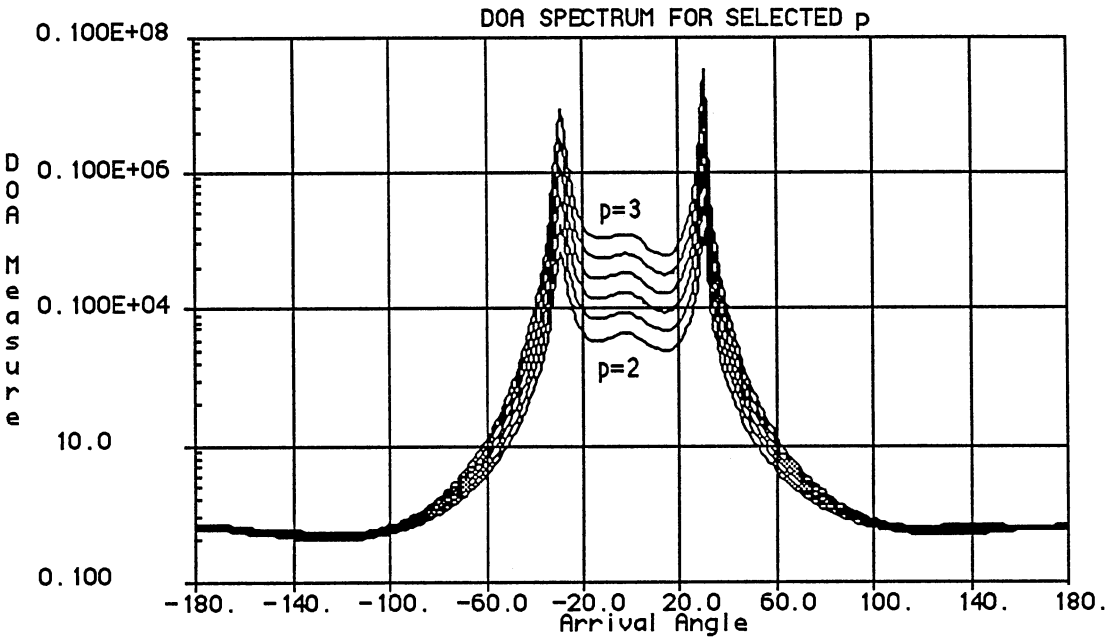


Figure 9. DOA Spectra, SNR = 20db, 20-Sample Interval

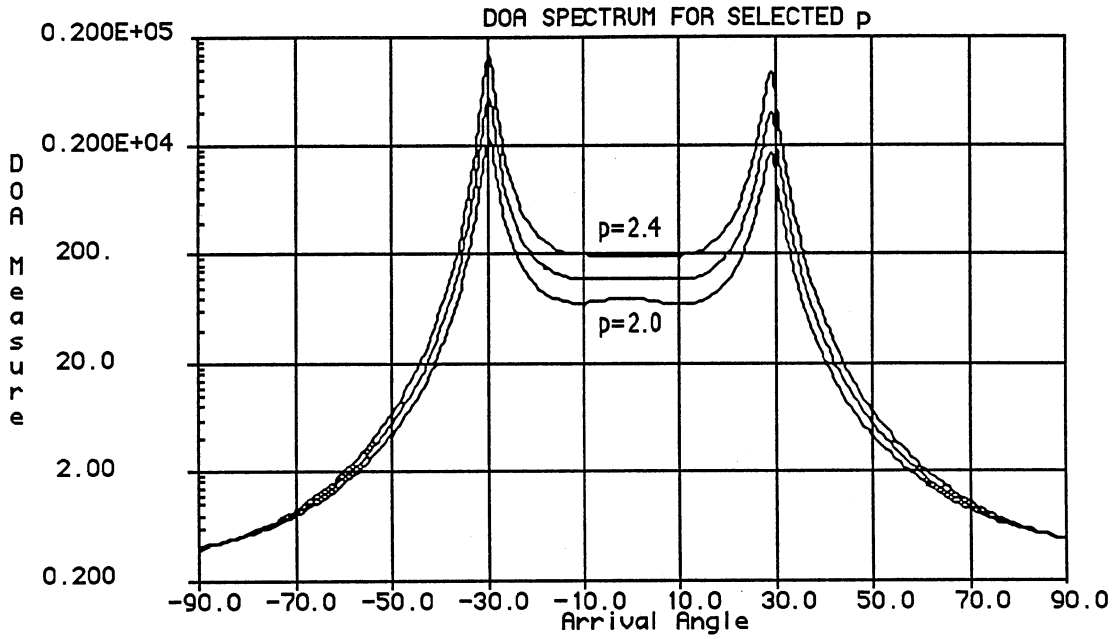


Figure 10. DOA Spectra, SNR = 0db, 20-Sample Interval

Figure 11 shows a second simulated array/wave combination, and Figures 12 through 14 illustrate a similar trend of progressive DOA resolution failure with decreasing SNR.

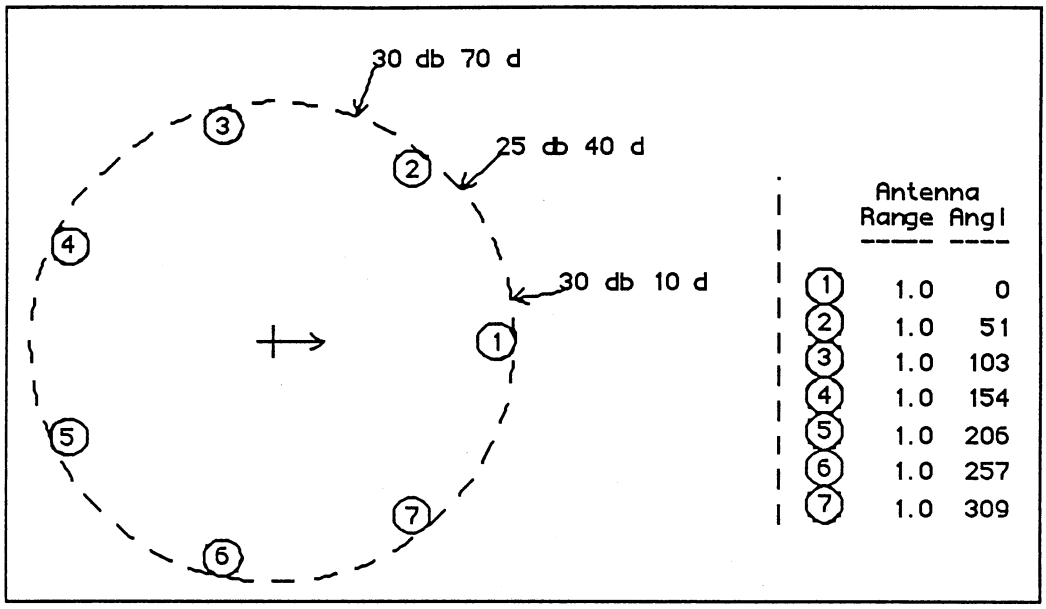


Figure 11. Antenna Array Geometry, Configuration #2

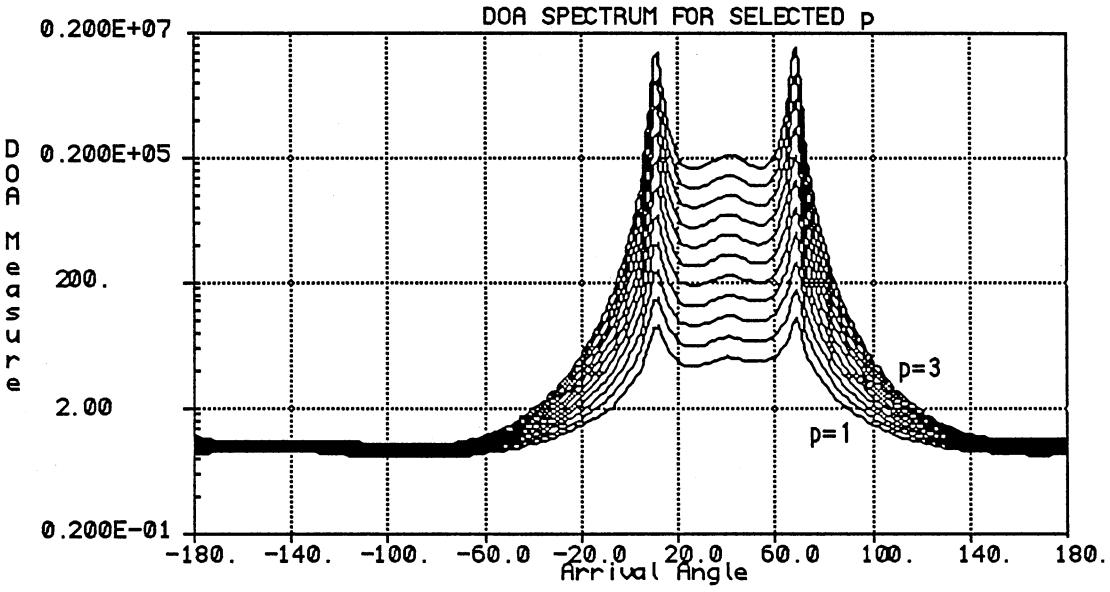


Figure 12. DOA Spectra, SNR = 30db, 50-Sample Interval

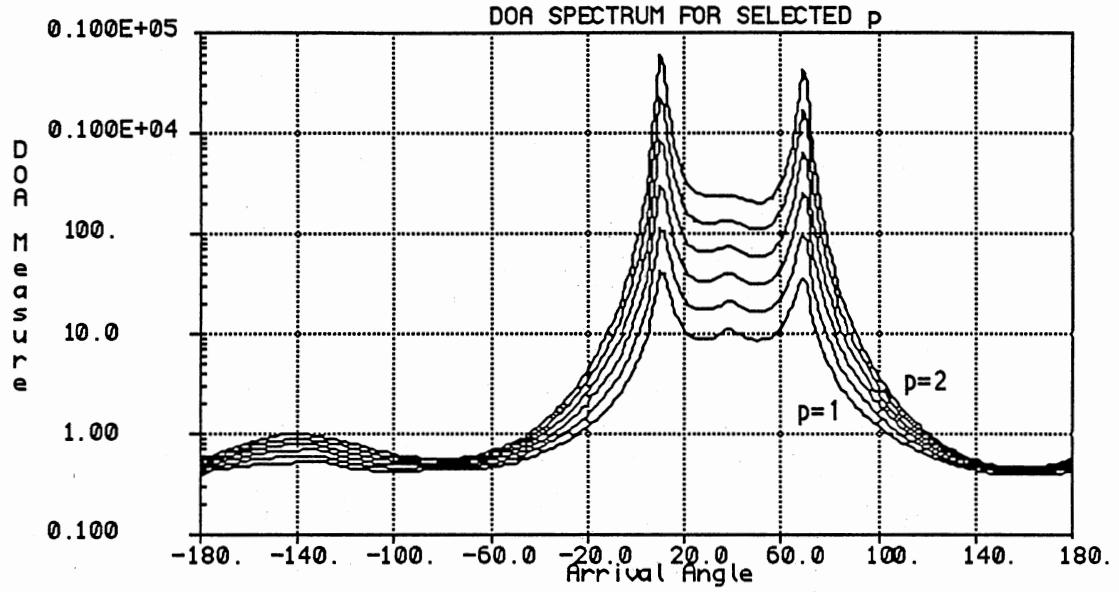


Figure 13. DOA Spectra, SNR = 20db, 50-Sample Interval

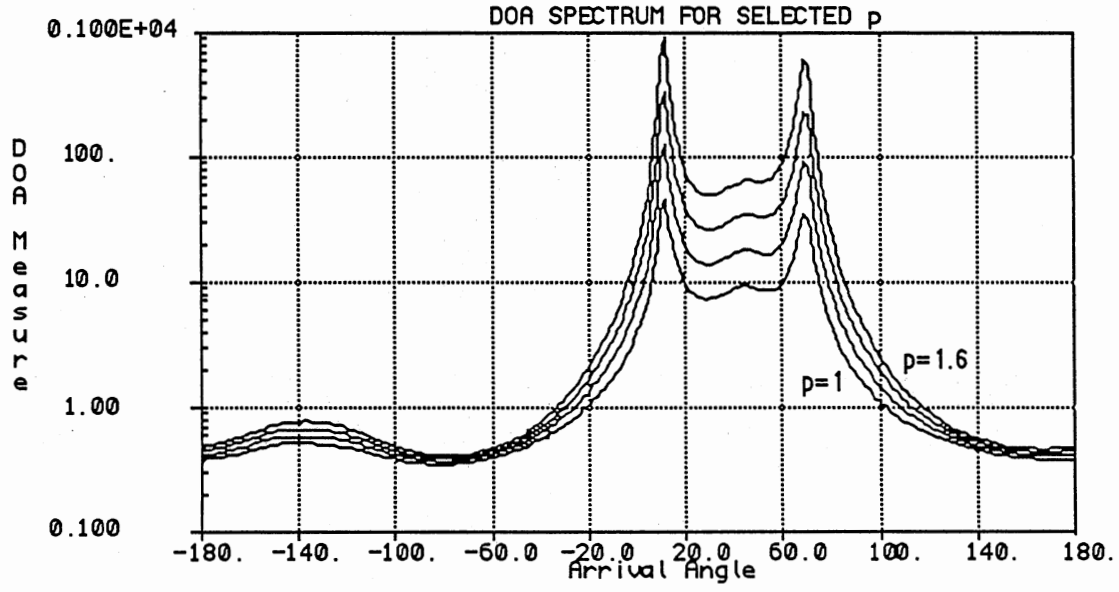


Figure 14. DOA Spectra, SNR = 10db, 50-Sample Interval

Effects of Noise Interval on DOA Resolution

Figures 15 through 18 illustrate that, as the average interval between impulsive noise bursts is increased, DOA resolution shows a general improvement. The arrays simulated were as illustrated in Figure 11.

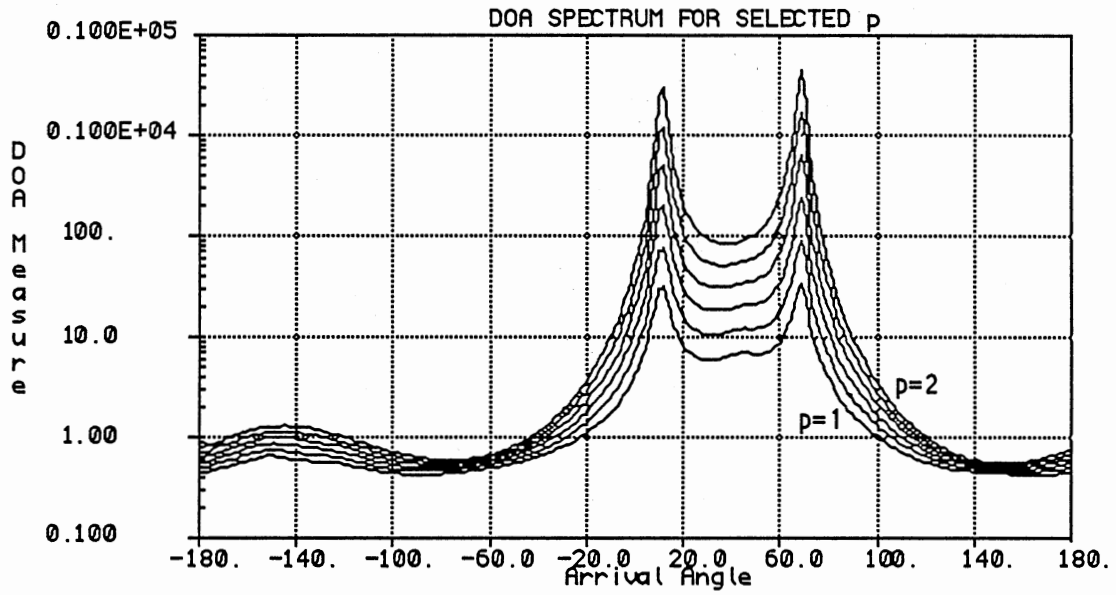


Figure 15. DOA Spectra, SNR = 20db, 40-Sample Interval

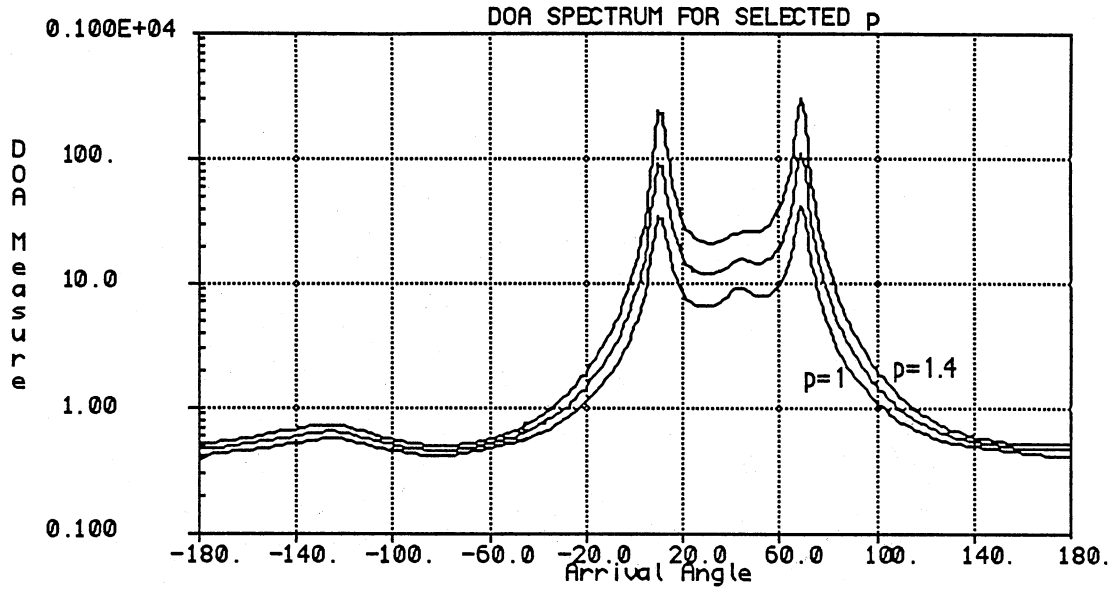


Figure 16. DOA Spectra, SNR = 20db, 45-Sample Interval

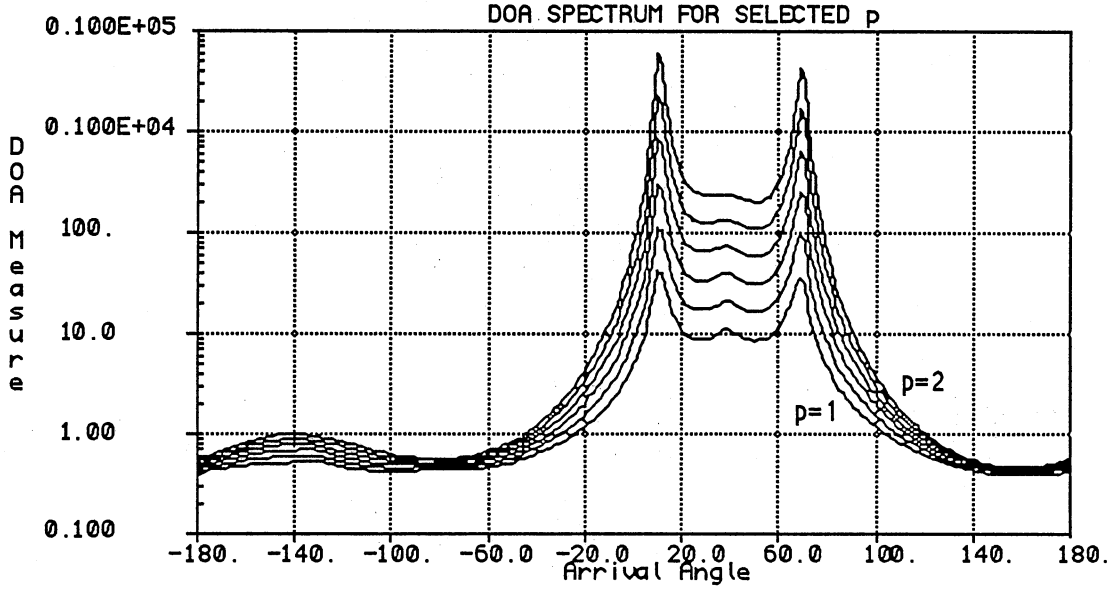


Figure 17. DOA Spectra, SNR = 20db, 50-Sample Interval

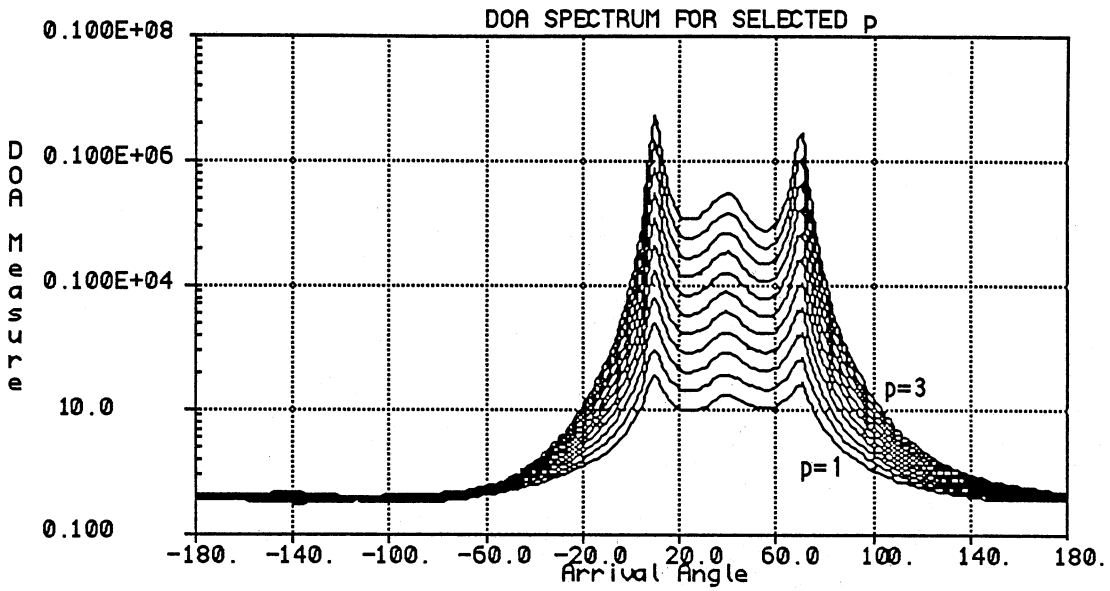


Figure 18. DOA Spectra, SNR = 20db, 55-Sample Interval

Iterative DOA Revision

Figures 19 - 22 provide an illustration of the application of the iterative DOA revision techniques discussed in Chapter III. Figure 19 shows the antenna array/arriving wave geometry simulated. The array was subjected to simulated impulsive noise at -20db (relative to maximum wave arrival strength) and a 40-sample average interval. As shown in Figure 20, resolution of the wave arriving from 35º is achieved only for values of $p < 2.0$. Figure 21 shows the DOA spectra which result from spectral subtraction of the effects of the wave arriving from 0º.

Figure 21 was produced as follows. The spectrum for $p = 2$ in Figure 20 (which does not resolve the wave from 35º) was used to estimate the angle of arrival of the wave from 0º. This DOA estimate was used to generate estimates of the **A** and **P** matrices corresponding to the identified arriving wave (from 0º).

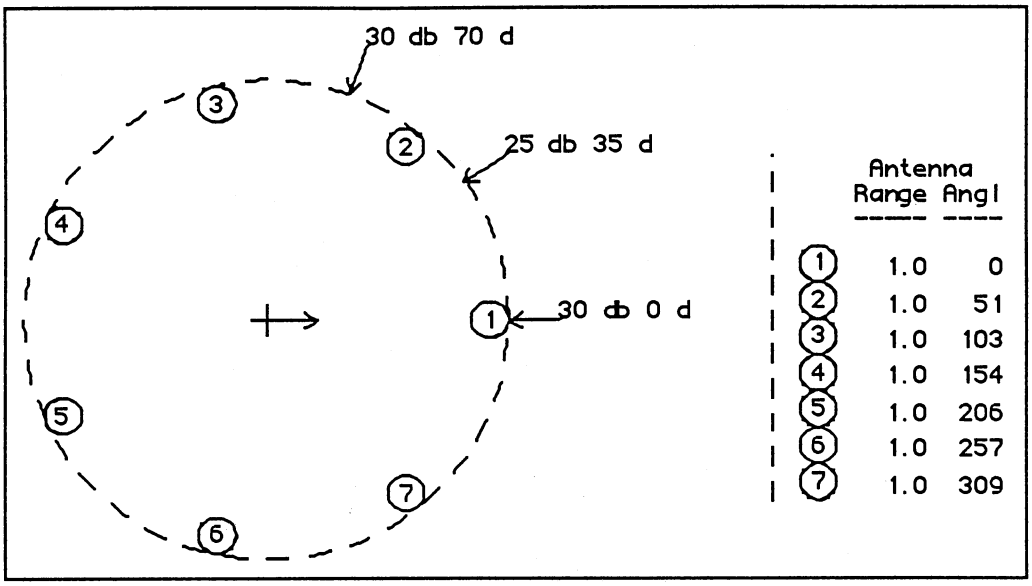


Figure 19. Antenna Array Geometry, Configuration #3

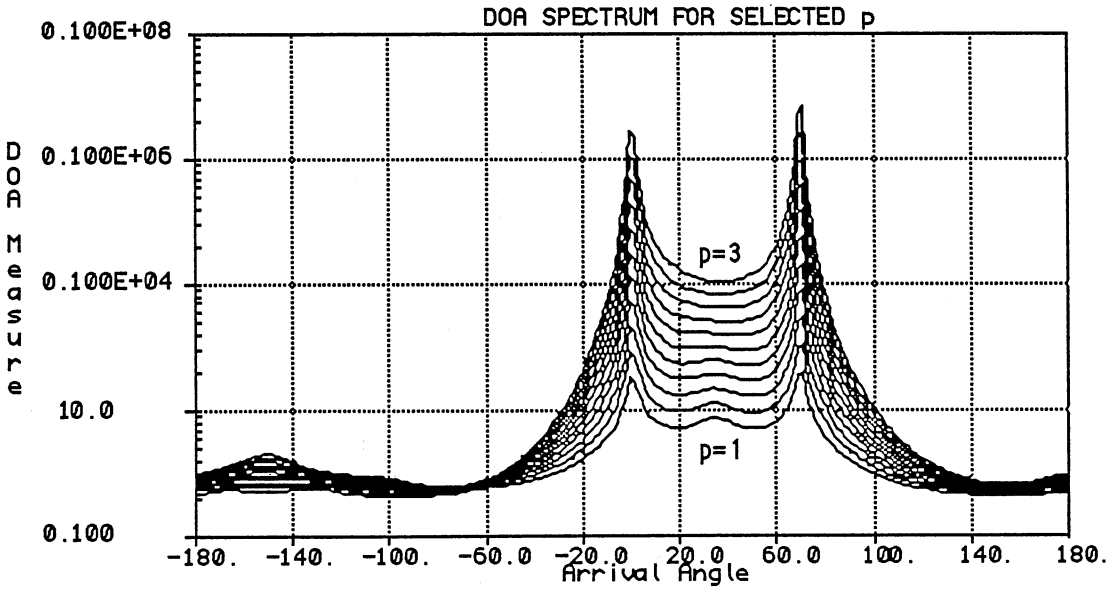


Figure 20. DOA Spectra, SNR = 20db, 40-Sample Interval

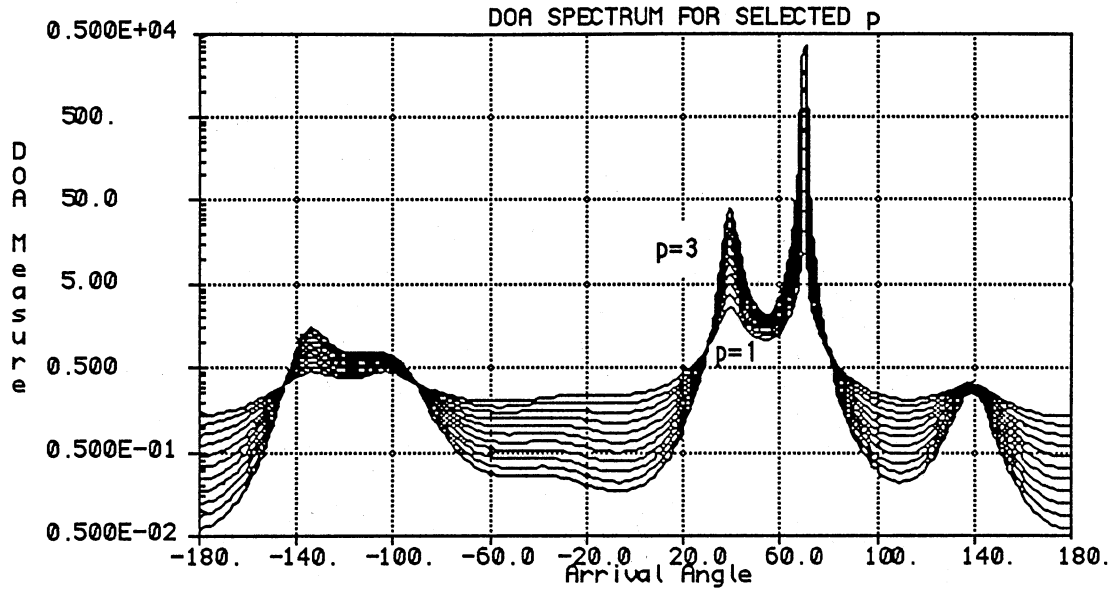


Figure 21. Configuration #3 Spectra Following Covariance Revision

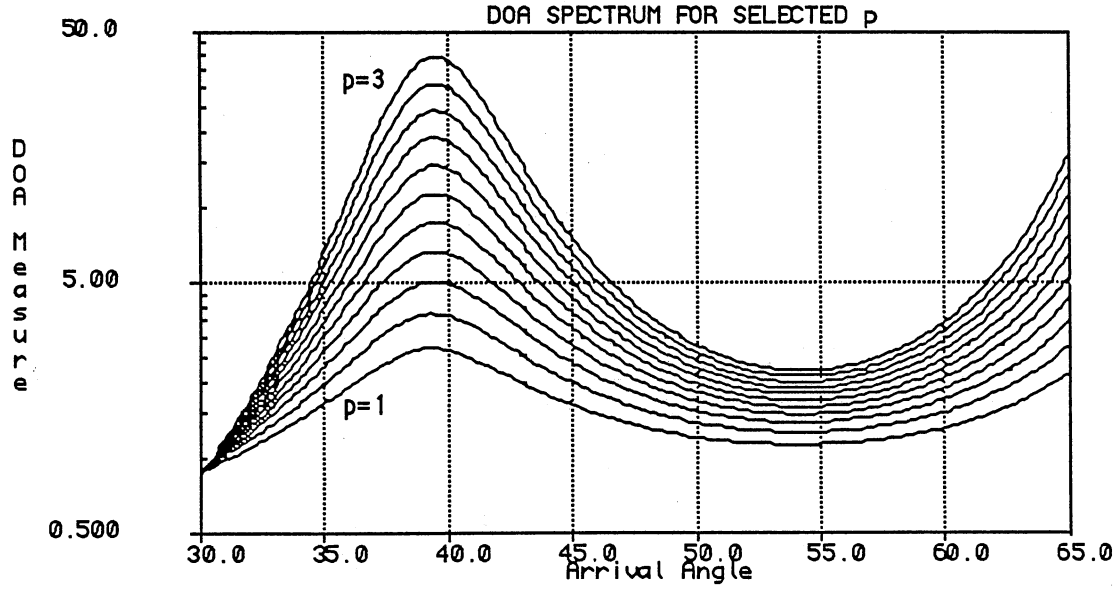


Figure 22. Configuration #3 Revised Spectra, 30° to 65°

These estimates were in turn used to revise the observation covariance matrix ($\mathbf{R}_x$), and a new eigenvalue problem involving the revised $\mathbf{R}_x$ was solved to produce the spectra in Figure 21. Figure 22 is a more detailed view of the region in Figure 21 from 30° to 65°.

Figures 21 and 22 illustrate both the benefits and some of the potential hazards of the iterative DOA revision technique. As can be seen from the figures, the previously unresolved wave from 35° is resolvable for $1 \leq p \leq 3$ following covariance revision. On the other hand, since covariance revision emphasizes all peaks in the DOA spectrum (except those being used in the revision process), the (less prominent) peaks at +140° and between –90° and –150°, which do not correspond to actual wave arrivals, are emphasized as well. So, as mentioned in Chapter III, when applying iterative DOA revision one must use caution to avoid the introduction of spurious wave arrival indications.

In producing the spectra shown in Figures 21 and 22, the optimum multiplier for the $\mathbf{APA}^H$ estimate (γ in Equation (3–15)) was resolved to 8 binary digits (bits). This implies that the value used was within 1% of the value which would have been available using infinite precision. As discussed in Chapter III, the use of a γ estimate with lower resolution should produce some DOA resolution improvement, while saving computation time. Figures 23 through 25 illustrate the effects on DOA spectra of using a γ estimate which is resolved to only 2 bits (within 50% of the value available using infinite precision).

Figure 23 shows the antenna array/arriving wave geometry simulated. The array was subjected to simulated impulsive noise at 0db (relative to maximum wave arrival strength) and a 35-sample (average) interval. Figure 24 shows that the two waves arriving from 35° and 70° produce only one spectrum peak, at about 65°. Figure 25 shows the result of revising the observation covariance matrix using estimated DOAs (from the p = 2 spectrum) for the waves

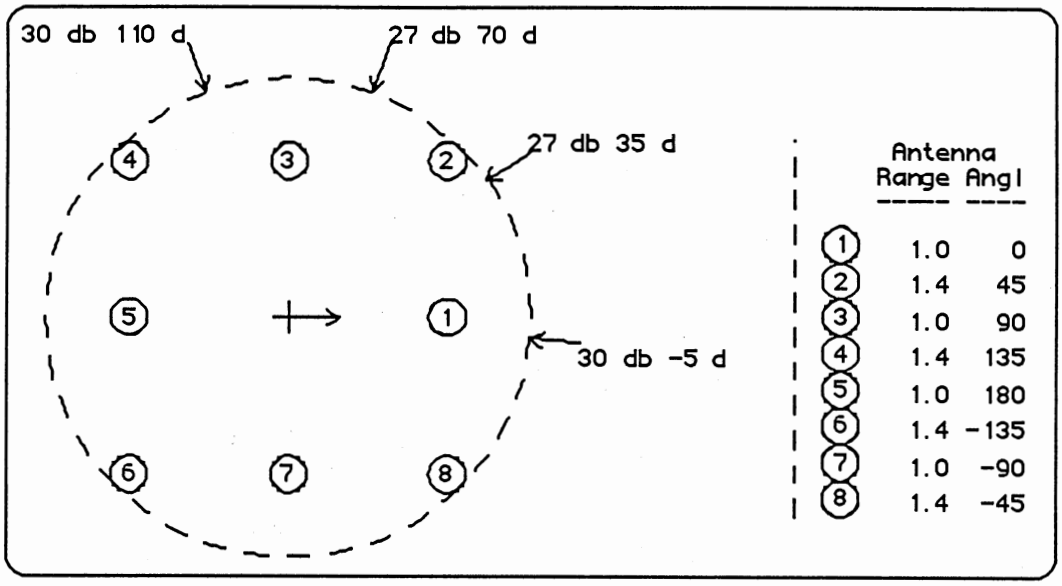


Figure 23. Antenna Array Geometry, Configuration #4

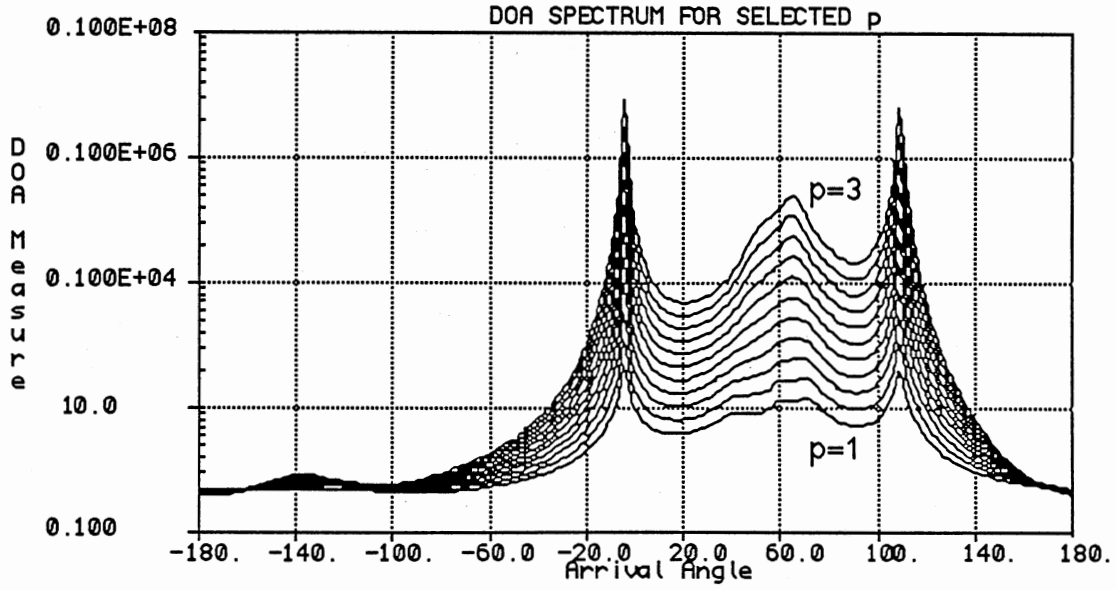


Figure 24. DOA Spectra, Configuration #4

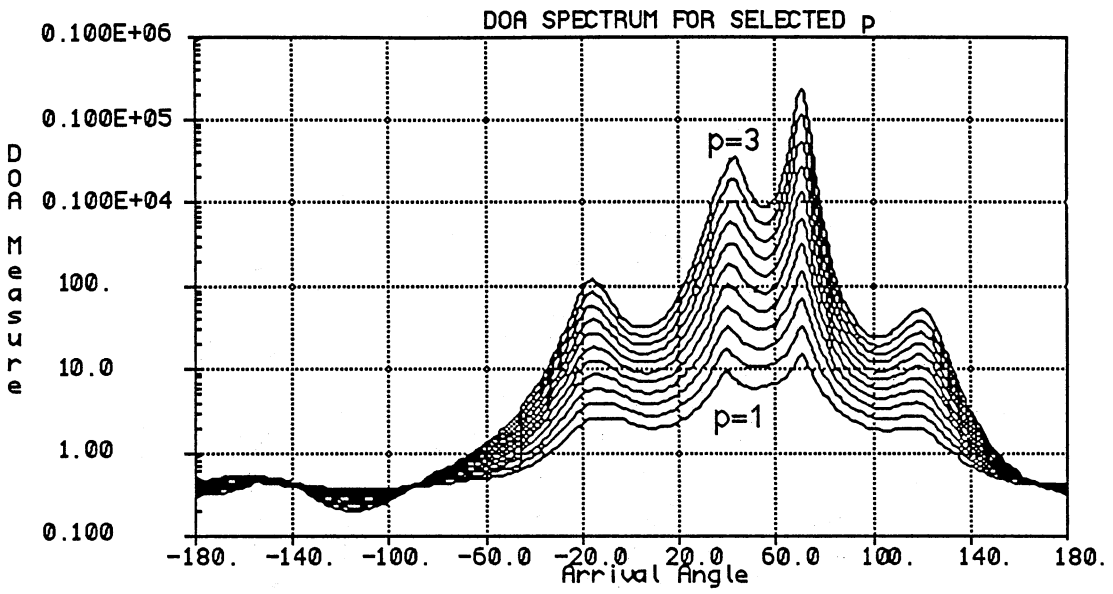


Figure 25. Configuration #4 Spectra Following Covariance Revision

arriving from –5º and +110º. The revision was made using a γ estimate which was correct to only 2 bits. As shown in Figure 25, DOA resolution for both in - terior waves is achieved following covariance revision for values of p over the entire range from 1 to 3.

The execution time required to obtain the value of γ used to produce Figures 21 and 22 (eight bit precision) was 1.3 CPU seconds on a VAX 11/750 processor. The eight-bit-precision value obtained was 0.82421875. To obtain a value precise to two bits ($\gamma = 0.75$) for the same problem required 0.92 CPU seconds, a 29% time savings. On the other hand, the time for computation of the **P** matrix estimate was 0.98 CPU seconds for p = 2, and 25.43 CPU seconds for p = 1. Thus, the time required to obtain even a comparatively high resolution estimate of γ can be small compared to the time required to estimate **P**. For the experiment shown in Figures 23 through 25, estimation of the **P** matrix (p = 2)

required 1.71 CPU seconds, and detemining γ (to two bits) required 1.22 CPU seconds.

Figure 26 shows values of γ as a function of the ratio between the maxi - mum diagonal element of the **P** estimate and the maximum diagonal element of $\mathbf{R}_x$. It is clear from the figure that, although there might be a tendency for γ to drop as the mentioned ratio increases, the actual functional relationship is extremely complicated.

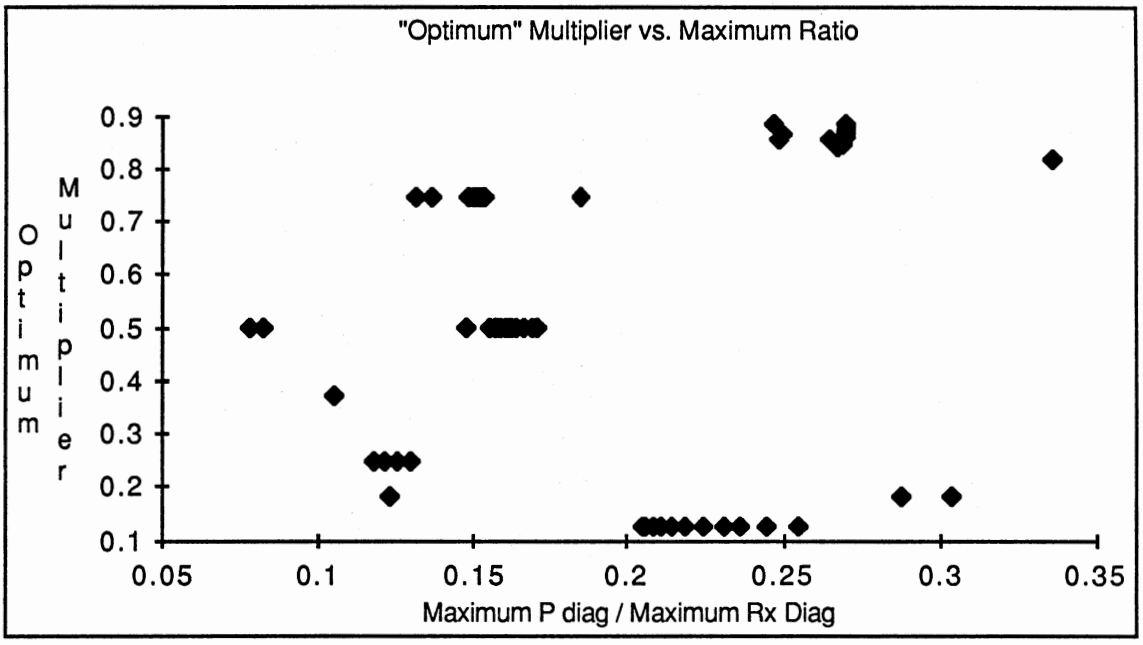


Figure 26. Optimum Multipliers

DOA Spectrum Peak Sharpness Measure

The L_p MUSIC algorithm is capable of determining directions of arrival of waves that are not resolved by the standard (L_2) MUSIC approach. When the

DOA spectrum for a particular value of p exhibits a peak that is not evident in the spectrum of another p value, it is clear that the former value of p is "better", at least as a detector of the wave that was missed by the latter p value. On the other hand, it is possible for a particular wave to produce peaks in the spectra for a variety of p values, in which case the decision of which p is "best" is some - what more complicated.

One basis for comparison of competing values of p can be developed from the degree of sharpness exhibited by the various peaks in the DOA spectra under consideration. The degree of sharpness of a particular peak can be ex - pressed as in Equation (4-1):

$$q(p) = \frac{1}{\Delta\theta} \log_{10} \sqrt[p]{f_p} \qquad (4-1)$$

where q(p) is the "sharpness" measure as a function of p, f_p is the magnitude ("height") of the peak being measured, and $\Delta\theta$ is the angular separation be - tween the two points in the spectrum (one on either side of the peak) at which the spectrum value is some fixed increment below the peak value. The quantity f_p is the reciprocal of g_p from Equation (3-12); i.e. f_p is the "DOA Measure" that is plotted to obtain a DOA spectrum.

Equation (4-1) incorporates all the visual aspects of a particular peak, and so can be expected to provide an objective measure of the "sharpness" of the peak (which, of course, is a subjective trait). For instance, higher peaks pro - duce larger values of q_p (assuming the other terms in Equation (4-1) don't change), since f_p is larger for higher peaks. Similarly, narrower peaks have larger q_p values, since $\Delta\theta$ is smaller for narrower peaks. Finally, taking the p^{th} root of the sum (of terms to the pth power) which comprises f_p removes the direct effect of p variations, which is to produce larger f_p values for larger p values.

Figures 27 and 28 provide examples of the variation of q_p with p. The value for the "fixed increment" in these figures is 1db; i.e. $\Delta\theta$ is measured be - tween the points at which the spectrum value is 1db below the peak value. The data used in generating Figures 27 and 28 was obtained from the experiment which produced Figures 19 through 22; with Figure 27 applicable before co - variance matrix revision and Figure 28 applicable after covariance matrix re - vision. Figure 27 was produced using the spectrum peaks at about 35° in Figure 20, for p values ranging from 1 to 2.2. As can be seen from Figure 27, the peak sharpness measure decreases as resolution is lost (as p rises from 1 to 2.2), so a p value of 1 or 1.2 would be preferred for use in further estimation steps. Additionally, actual DOA error, as shown in the figure, generally rises as p increases from 1 to 2.2 and resolution is lost. Therefore, in this case selection of the p which produces the largest sharpness measure results in one of the best available DOA estimates (although this is not guaranteed in general).

Figure 28 was produced using the spectrum peaks at about 35° in Figure 21. As stated above, Figure 21 results from covariance matrix revision of the experiment which produced Figure 20. In the case of Figure 28, p = 3 provides the greatest sharpness measure, so would be preferred as the p value for use in subsequent estimation. A review of Figure 22 shows that the peak produced for p = 3 does appear to be the sharpest of the spectrum peaks in the neighbor - hood of 35°. Although in this case the actual DOA error (shown in Figure 28) is larger for p = 3 than for p = 1, the difference in DOA error is very slight across the range of p values.

Two other points of interest are evident from a comparison of Figure 27 with Figure 28. First, all the sharpness measures plotted in the upper sharp - ness curve of Figure 28 (after covariance revision) are larger than any of the sharpness measures shown in Figure 27 (before covariance matrix revision).

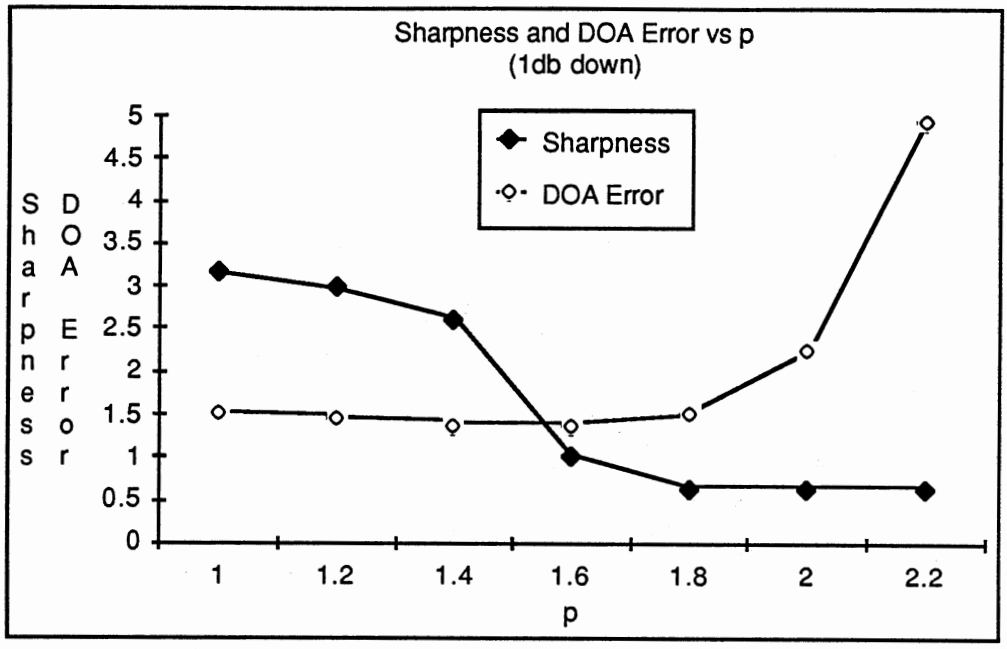


Figure 27. Sharpness Variation with p, Case 1

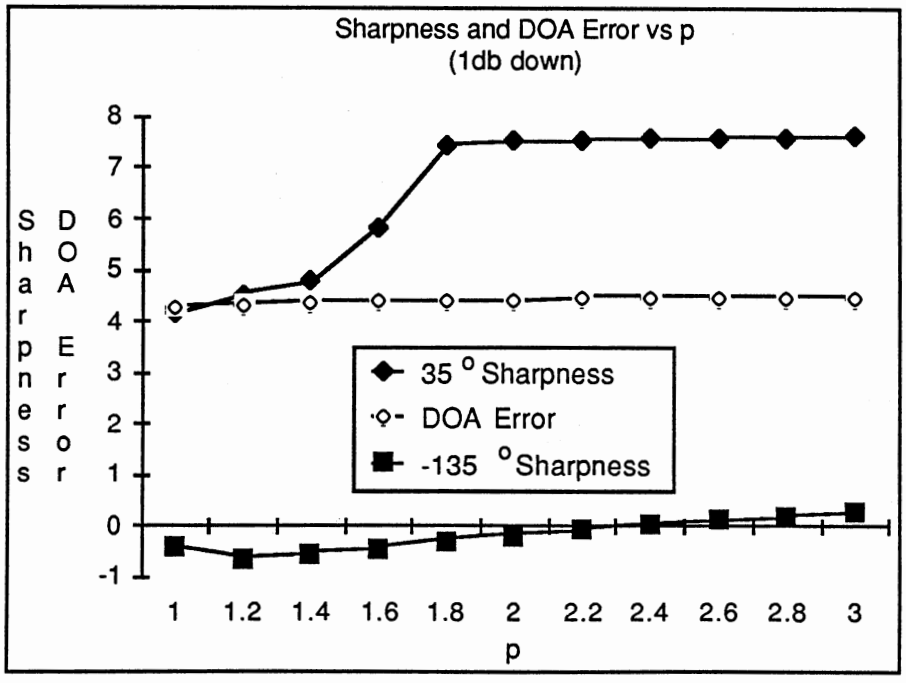


Figure 28. Sharpness Variation with p, Case 2

Thus, the sharpness measure accurately reflects the increase in peak sharp - ness produced by the iterative DOA revision approach.

The second point of interest is illustrated by comparing the upper sharp - ness curve with the lower sharpness curve in Figure 28. The upper curve is composed of the sharpness measures for the spectrum peaks at about 35º in Figure 22. The lower curve was produced for the spectrum peaks at about –135º, which do not correspond to an acutal arriving wave. As is clear from Figure 28, the spurious peaks at –135º exhibit much lower sharpness values (and thus are much milder peaks) than do those at about 35º, which correspond to actual arriving waves. So, in this case a comparison of sharpness values assists not only in selection of a value of p but also in selection of which spec - trum likely correspond to actual arriving waves.

Although the sharpest peak does not necessarily imply the most accurate DOA estimate (due to the complex interaction of the various influences on the DOA spectrum), q_p does provide a method for choosing one p value (or a col- lection of p values) over other values when problem constraints (e.g. available processing time) require such a choice. As mentioned above, q_p can also assist in selection of the DOA peaks which correspond to actual arriving waves, although again it is not guaranteed to do so in every case.

It is emphasized that Figures 27 and 28 are two examples out of many that may be produced. It is *not* true in general that p = 1 or p = 3 produces the best DOA estimates, nor that q_p is invariably highest for values of p which pro - duce best DOA estimates. Simulations conducted during this research reveal that values of p other than two can, in some situations, produce superior resolu - tion to that of standard (L_2) MUSIC, and that $p < 2$ is particularly useful in the presence of impulsive noise. On the other hand, the production of a DOA spec - trum is an extremely complicated procedure, and it would be simplistic to claim

that any particular value of p (or of any other parameter used in L_p MUSIC) is best for all applications or even all possible data sets from a selected application.

CHAPTER V

CONCLUSIONS AND FUTURE RESEARCH

Conclusions

Two extensions were proposed to the MUSIC direction finding algorithm. The first, use of the L_p norm in the DOA spectrum expression (3-12), can pro - vide improved DOA resolution in some situations, particularly where impulsive noise is present.

In the proposed L_p DOA approach, angles θ which minimize a sum (g_p) of terms to the p^{th} power are determined. These angles are L_p estimates of direction of arrival, in the sense that they minimize the p-norm of the difference between the array manifold ($\underline{a}(\theta)$) and the signal subspace ($\underline{e}_n$, n = 1, 2, ..., M). The angle determination is accomplished by minimizing (in the p-norm sense) the projection of the vector $\underline{a}(\theta)$ onto the noise subspace. Since the noise sub - space is the orthogonal complement of the signal subspace, this minimizes the distance between the array manifold and the signal subspace. That is, mini - mizing g_p selects the angles that best fit the observed data, in the p-norm sense, subject to the constraints of array geometry.

Additionally, no iteration is required to determine the L_p DOA estimates described above (as is necessary in the case of estimating **P**). This implies that a **P** estimate based on L_p DOA estimates (which can be considered an L_p esti - mate of **P**) can be obtained with essentially no more computation than is re - quired to obtain an L_2 estimate of **P** (from the MUSIC DOA estimates). Also,

since no convergence issues are involved in minimizing g_p, one could obtain L_p DOA estimates (and the associated L_p **P** estimate) for any value of $p > 0$ (al - though the interpretation for values of p outside the range $1 \leq p \leq 3$ might be open to question).

Further, since g_p provides a measure of the p-norm distance between the array manifold and the signal subspace, it gives at least a relative indication of the success of the estimation process. For example, if additional data is taken following an estimation step, and incorporation of the additional data results in lower values for g_p, it might be claimed that the estimates obtained using the additional data are in a sense "better" than the original estimates.

For the case of the number of arriving waves being one less than the number of antenna elements, the angles which minimize g_p are unchanged by varying p. This is because the sum to be minimized consists of one term for this case, and the minimum of a single term occurs at the same point regardless of the (positive) exponent to which that term is raised. Thus, when estimation of the directions of arrival of one fewer waves than array elements is attempted ("array saturation"), L_p DOA estimates are exactly the same as L_2 estimates, re - gardless of the (positive) value of p used. In this case the noise subspace con - sists of a line in N-space passing through the single noise eigenvector $\underline{e}_N$. Therefore, any local minimum of the projection of a vector (particularly the array manifold vector) onto the noise subspace (a line) can occur in only one direc - tion, regardless of the norm used.

It was also possible to improve MUSIC's DOA resolution capability using an iterative DOA revision technique. The approach is based on a spectral sub - traction of the effects of previously estimated directions of arrival from the L_p DOA spectrum. This technique was helpful whether the L_2 or the L_p norm was used, and provided the ability to resolve signals that are more closely spaced

than can be resolved with standard MUSIC. The technique involves empirical determination of a real multiplicative constant, and a numerical binary search procedure was proposed which permits determination of the constant's value to whatever resolution is desired. The search procedure allows for adjusting the resolution obtained based on application constraints such as computational burden.

Finally, an objective measure was developed of the sharpness of peaks in DOA spectra. This measure can be used to select which value or values of p are to be used during the estimation process, based on the variation in the sharpness measure over the range of p values.

Areas for Further Research

During the conduct of the research described herein, a number of issues arose which deserve further investigation. Two major areas for productive fu - ture research are alternatives to the algorithms used for DOA estimation, and implementation of these or other algorithms on multiple-processor computer systems.

Alternatives to MUSIC

The MUSIC algorithm was chosen as the basis for this research due to the fact that it is the most generally applicable of the high-resolution DOA esti- mation procedures. The L_p-norm DOA spectrum expression developed in Chapter III, however, could as easily have been developed for some other DOA estimation approach. For example, the sum-of-squares expressions used in the Maximum Likelihood, Burg, and Modified Forward-Backward Linear Predictor methods could be replaced by expressions involving sums of terms to the p^{th} power, with a possible improvement in DOA resolution similar to that shown by

the L_p MUSIC expression. Such revisions of existing techniques to accommodate L_p norm estimation could provide opportunities for improved performance with very little additional computational burden.

A related area of potential improvement involves the "array saturation" case described in Chapter III. In this situation, only one noise eigenvector is estimated, so that L_2 and L_p estimation produce the same results. The adaptive MUSIC method described in Chapter III, on the other hand, develops estimates of the signal subspace vectors rather than the noise subspace vectors. For the case of array saturation, then, it may be possible to apply the L_p norm to estimation of the signal subspace vectors, thereby possibly improving the resolution abilities of algorithms based on such estimates.

Use of Phase Information

The MUSIC algorithm employs the phase of the antenna array samples in the determination of directions of arrival. However, the IRLS procedure for estimating **P** deals only with the magnitudes of the estimation residuals, thereby effectively ignoring potentially useful phase information. Use might be made of residual phase information, perhaps in an iterative procedure which alternately minimizes residual magnitudes and residual phases, to improve the performance of the overall estimation approach.

Also, the IRLS algorithm is by no means limited to powers of residual magnitudes as cost functions. A more general criterion function, which could include phase information, may produce a superior estimation algorithm. For example, $\ln(A) = |A| + j \arg(A)$, where A is a complex number, embodies both magnitude and phase information. It may be advantagous to use, for instance, some function of $\ln(r_i)$, where r_i is an estimation residual, as the cost function for IRLS processing.

Alternatives to IRLS

The IRLS algorithm is extremely computationally intensive. This is be - cause a matrix psuedo-inverse is required at each IRLS iteration step (since the weight matrix changes at each step). An alternative approach, the Residual Steepest Descent (RSD) algorithm, can usually be employed to considerably reduce the computational burden. The RSD algorithm requires the matrix inver - sion only once, at the beginning of the estimation process. Although the RSD algorithm does not guarantee convergence to an exact L_p solution under all circumstances, it is accurate enough for use in most situations. Therefore, an RSD based approach might provide the advantages of L_p estimation with a re - duced computational load.

For the case of L_1 estimation, linear programming approaches are available; however, they are not very attractive (Lansford, 1985). Direct imple - mentation of the simplex linear programming solution is usually prohibitive in terms of memory required; however, Karmarkar's (1984) algorithm, among others, can be used to circumvent this difficulty to some extent. Use of such linear programming algorithms for L_1 estimation should therefore be investi - gated.

Adaptation to Multiple Processor Systems

The simulation program mentioned in Chapter III was developed for a general purpose, single-processor system. Adaptation for multiple-processor computers of the various algorithms discussed can be expected to provide the opportunity for improved performance. Parallel implementations of some of the procedures required by MUSIC and the other estimation routines already exist

(e.g. matrix multiplication and singular value decomposition). Of course, parallel implementation of a complete DOA estimation system requires more than merely "stringing together" the parallel versions of the various routines employed, and is a subject worthy of intense study.

As mentioned in Chapter III, Van Loan's generalized singular value de - composition was used to solve the generalized eigenvalue problem of Chapter II. Although a parallel version of Van Loan's algorithm may be possible, another algorithm, due to Paige (1986), is available for computing the general - ized singular value decomposition. A parallel version of this algorithm has al - ready been developed (Heath, et. al., 1986).

Finally, the iterative DOA revision process described in Chapter III, al - though not apparently a ready candidate for parallel processing, may be well suited to a Multiple Instruction stream Multiple Data stream (MIMD) architecture. Research currently in progress at Oklahoma State University is directed toward exploiting the exceptional power of this architecture, as applied to problems in - volving multiple similar but not identical processes.

REFERENCES

Bandler, J. W. "Computer-Aided Circuit Optimization,", Modern Filter Theory and Design, G. C. Temes and S. K. Mitra, ed., Wiley, New York (1973).

Bard, Y. Nonlinear Parameter Estimation, Academic Press, New York (1974).

Böhme, J. F. "On Parametric Methods for Array Processing," Proceedings of EUSIPCO, Erlangen, W. Germany (1983), 637-644.

Burg, J. P. "Maximum Entropy Spectral Analysis," Proc. 37th Mtg. Soc. Explor. Geophysicists (1967).

Burg, J. P. Maximum Entropy Spectral Analysis, Ph. D. Dissertation, Stanford University (1975).

Byrd, R. H., and D. A. Pyne. "Convergence of the Iteratively Reweighted Least Squares Algorithm for Robust Regression", Johns Hopkins University Technical Report No. 313 (June, 1979).

Claerbout, J. R., and F. Muir. "Robust Modeling with Erratic Data", Geophysics, Vol. 38, 1973.

Denoel, E. and Jean-Phillips Solvay. "Linear Prediction of Speech with a Least Absolute Error Criterion", IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-33 (1985).

Durbin, J., "The Fitting of Time-Series Models," Rev. Int. Stat. Inst., Vol. 28 (1960), 233-243.

Eckart, C., and G. Young, , "The Approximation of a Matrix by Another of Lower Rank," Psychometrika, Vol. 1 (1931), 211-218.

Fletcher, R., J. A. Grant, and M. D. Hebden. "The Calculation of Linear Best L_p Approximations", Computer Journal, 14 (1971).

Fletcher, R., and M. J. D. Powell. "A Rapidly Convergent Descent Method for Minimization," Comput. J., Vol. 6 (1963), 163-168.

Fuhrmann, D. R. "Adaptive Multiple Signal Classification (MUSIC)", SPIE Proceedings, Vol. 826 (August, 1987).

Gallop, M. A., and L. W. Nolte. "Bayesian Detection of Targets of Unknown Location," IEEE Trans. Aerospace and Electronic Systems, Vol. 10 (July, 1974), 429-435.

Geary, R. C. "Testing for Normality", Biometrika, Vol. 34 (1947), 209-242.

Goodwin, Graham C., and Kwai Sang Sin. Adaptive Filtering, Prediction, and Control, Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1984).

Graupe, Daniel. Identification of Systems, Robert E. Krieger Publishing Company, Huntington, New York (1976).

Graybill, Franklin A. Introduction to Matrices with Applications in Statistics, Wadsworth Publishing Company, Inc., Belmont, Calif. (1969).

Haykin, Simon. Communcation Systems, 2nd. ed., Wiley, New York (1983).

Haykin, Simon. "Radar Array Processing for Angle of Arrival Estimation", Array Signal Processing, S. Haykin, Editor, Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1985).

Haykin, Simon, and J. Kesler. "Adaptive Canceller for Elevation Angle Estimation in the Presence of Multipath," Proc. IEEE, Vol. 30, Part F (June, 1983), 303-308.

Heath, M. T., et. al. "Computing the Singular Value Decomposition of a Product of Two Matrices"", SIAM Journal of Scientific and Statistical Computing, Vol. 7, No. 4 (October, 1986).

Hildebrand, F. B. Introduction to Numerical Analysis, McGraw-Hill, New York (1956).

Hohn, Franz E. Elementary Matrix Algebra, Second Edition, The MacMillan Company, New York (1964).

Huber, P. J., and R. Dutter. "Numerical Solution of Robust Regression Problems", COMPSTAT 1974 Proceedings of the Symposium on Computational Statistics (G. Bruchmann, ed.) Wien: Physika Verlag.

Karmarkar, N. K. "A New Polynomial-Time Algorithm for Linear Programming", Combinatoria, Vol. 4 (1984), 373-395.

Kesler, J. Adaptive Interference Cancelling in Multiple Beam Antennas with Applications to Multipath, Ph.D. Dissertation, McMaster University, Hamilton, Ont. (March,1981).

Kumaresan, R., and D. W. Tufts. "Singular Value Decomposition and Spectral Analysis," Proc. First IEEE ASSP Workshop on Spectral Estimation, McMaster University, Hamilton, Ont. (August, 1981).

Lansford, J. L. L_p Models in Speech Coding and Markov Chains in Speech Recognition, Ph.D. Dissertation, Oklahoma State University (May, 1988).

Levinson, N. "The Wiener RMS (Root Mean Square) Error Criteria in Filter De - sign and Prediction," J. Math. Phys., Vol. 25 (1947), 261-278.

Lo, J. T., and S. L. Marple, Jr. "Eigenstructure Methods for Array Sensor Localization", Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (1987).

Marple, S. Lawrence, Jr. Digital Spectral Analysis with Applications, Prentice-Hall, Inc., Englewood Cliffs, N. J. (1987).

Middleton, D. "Multidimensional Detection and Extraction of Signals in Random Media," Proc. IEEE, Vol. 58, No. 5 (May, 1970), 696-706.

Nuttal, A. H. "Spectral Analysis of a Univariate Process with Bad Data Points via Maximum Entropy and Linear Predictive Techniques," Naval Under - water System Center (NUSC) Scientific and Engineering Studies, Spec - tral Estimation, NUSC, New London, Conn. (March, 1976).

Ouibrahim, H. A Generalized Approach to Direction Finding, Ph. D. Disserta - tion, Syracuse University (December, 1986).

Paige, C. C. "Computing the Generalized Singular Value Decomposition", SIAM Journal of Scientific and Statistical Computing, Vol. 7, No. 4 (October, 1986).

Prony, R. "Essai Expérimental et Analytique, etc.," L'Ecole Polytechnique, Paris, Vol. 1 (1795), 24-26.

Scales, J. A. and S. Treitel. IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-35, 581 - 582 (1987).

Schmidt, R. O. A Signal Subspace Approach to Multiple Emitter Location and Spectral Estimation, PhD. Dissertation, Stanford University (November, 1981).

Schroeder, J. E. Linear Predictive Spectral Analysis Via the L_p Norm, Ph.D. Dissertation, Oklahoma State University (December, 1985).

Shaw, A. K., and R. Kumaresan. "Estimation of Angles of Arrivals of Broadband Signals", Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (1987).

Speiser, J. M. "Progress in Eigenvector Beamforming," Real Time Signal Pro - cessing VIII, SPIE Proceedings, Vol. 564 (August, 1985).

Speiser, J. M. "Modern Matrix Computation Methods," Advanced, Real-Time, Parallel, Matrix Signal Processing, Evolving Technology Institute (October, 1985).

Speiser, J. M. "Some Observations Concerning the ESPRIT Direction Finding Method", SPIE Proceedings, Vol. 826 (August, 1987).

Taylor, H. L., S. C. Banks, and J. F. McCoy. "Deconvolution with the L_1 Norm", Geophysics, Vol. 44 (1979).

Tufts, D. W., and R. Kumaresan. "Estimation of Frequencies of Multiple Sinu - soids: Making Linear Prediction Perform Like Maximum Likelihood," Proc. IEEE, Vol. 70 (Sept, 1982), 975-989.

Ulriych, T. J., and R. W. Clayton. "Time Series Modelling and Maximum Entro - py," J. Phys. Earth Planet. Inter., Vol. 12 (1976), 188-200.

Van Loan, Charles. "Computing the CS and the Generalized Singular Value Decomposition", SIAM Journal of Numer. Math, Vol. 46 (1985).

Van Trees, H. L. Detection, Estimation, and Modulation Theory, Part I, Wiley, New York (1968).

Yarlagadda, R., J. B. Bednar, and T. L. Watt. "Fast Algorithms for l_p Deconvolu - tion", IEEE. Trans. Acoustics, Speech, and Signal Processing, Vol. ASSP-33, No. 1 (February, 1985).

Yarlagadda, R., and J. E. Hershey. "Signal Processing, General", Encyclopedia of Physical Science and Technology, Vol. 12, 626 - 646 (1987).

APPENDIX A

CONVERGENCE OF THE ITERATIVELY REWEIGHTED LEAST SQUARES (IRLS) ALGORITHM

The objective of the IRLS algorithm is to estimate a vector $\underline{f}$ given the linear model:

$$\underline{x} = \mathbf{A}\underline{f} + \underline{w};$$

where $\underline{x}$ is an N-element output vector, $\mathbf{A}$ is an NxM matrix of rank M, $\underline{f}$ is an M-element vector of unknown parameters, and $\underline{w}$ is an N-element zero-mean noise vector. The IRLS algorithm repeatedly evaluates:

$$\underline{f}_{t+1} = [\mathbf{A}^H\mathbf{W}(\underline{f}_t)\mathbf{A}]^{-1}\mathbf{A}^H\mathbf{W}(\underline{f}_t)\underline{x} \qquad (A-1)$$

until $\underline{f}$ converges. The weighting matrix $\mathbf{W}$ is a diagonal matrix with entries:

$$w_{nn} = w_n = w(r_n) = \psi(r_n)/r_n = \rho'(r_n)/r_n \qquad (A-2)$$

where $r_n = \underline{a}_n\underline{f} - x_n$; $\underline{a}_n$ is the $n^{\underline{th}}$ row of $\mathbf{A}$; x_n is the $n^{\underline{th}}$ element of $\underline{x}$; and the objective function ρ is a positive, even, differentiable function. This definition of $\mathbf{W}$ results in minimization of the loss function J, as defined below:

$$J = \sum_{n=1}^{N} \rho(\underline{a}_n\underline{f} - x_n) \qquad (A-3)$$

That iteration of Equation (A-1) using $\mathbf{W}$ as defined in Equation (A-2) minimizes J, above, can be seen as follows.

If $\rho(r)$ is nondecreasing in $|r|$, then for J to be minimized, it is necessary only that:

$$\frac{\partial J}{\partial f_m} = \sum_{n=1}^{N} \rho'(\underline{a}_n \underline{f} - x_n) a_{nm} = 0; \quad m = 1, 2, \ldots, M$$

where f_m is the m^{th} element of $\underline{f}$, and a_{nm} is the m^{th} element of the n^{th} row of **A**.

In vector form:

$$\mathbf{A}^T[\rho'(r_n)] = \underline{0} \qquad (A-4)$$

where $[\rho'(r_n)]$ is the N-element vector of derivatives evaluated at r_n; n = 1, 2, ..., N, and $\mathbf{A}^T[\rho'(r_n)]$ is an M-vector. Since $w(r_n) = \rho'(r_n)/r_n$:

$$\mathbf{W}(r_n)\,[r_n] = [\rho'(r_n)]$$

where $\mathbf{W}(r_n)$ is a diagonal matrix with elements $w(r_n)$ and $[r_n]$ is the N-element vector of residuals r_n. So, from Equation (A-4), we have:

$$\mathbf{A}^T\,\mathbf{W}(r_n)\,[r_n] = \underline{0} \qquad (A-5)$$

where $\mathbf{A}^T\,\mathbf{W}(r_n)\,[r_n]$ is an M-vector.

Now consider the change in $\underline{f}$ at iteration step t, given by the difference $\underline{f}_{t+1} - \underline{f}_t$. Using Equations (A-1) and (A-2) gives:

$$\begin{aligned}\underline{f}_{t+1} - \underline{f}_t &= \{\mathbf{A}^T\,\mathbf{W}(r_n)\,\mathbf{A}\}^{-1}\mathbf{A}^T\,\mathbf{W}(r_n)\,\underline{x} - \underline{f}_t \\ &= \{\mathbf{A}^T\,\mathbf{W}(r_n)\,\mathbf{A}\}^{-1}\mathbf{A}^T\,\mathbf{W}(r_n)\,\underline{x} - \{\mathbf{A}^T\,\mathbf{W}(r_n)\,\mathbf{A}\}^{-1}\{\mathbf{A}^T\,\mathbf{W}(r_n)\,\mathbf{A}\}\underline{f}_t \\ &= -\{\mathbf{A}^T\,\mathbf{W}(r_n)\,\mathbf{A}\}^{-1}\mathbf{A}^T\,\mathbf{W}(r_n)\,\{\mathbf{A}\underline{f}_t - \underline{x}\}\end{aligned}$$

Now when $\underline{f}$ converges, $\underline{f}_{t+1} - \underline{f}_t \rightarrow \underline{0}$. Thus, since $[r_n] = \mathbf{A}\underline{f}_t - \underline{x}$:

$$\underline{f}_{t+1} - \underline{f}_t = -\{\mathbf{A}^T\,\mathbf{W}(r_n)\,\mathbf{A}\}^{-1}\mathbf{A}^T\,\mathbf{W}(r_n)\,[r_n] \rightarrow \underline{0}$$

implies that $\mathbf{A}^T\,\mathbf{W}(r_n)\,[r_n] \rightarrow \underline{0}$, which, from Equation (A-5), means that J is mini - mized.

This means that use of Equation (A-1), with **W** as defined in Equation (A-2), minimizes the cost function given in Equation (A-3) if it (Equation (A-1)) con - verges. Global convergence of the algorithm is guaranteed for objective func - tions ρ such that:

(1) $\rho(r)$ is a differentiable, symmetric, positive function nondecreasing in $|r|$,

(2) $\rho(r) \rightarrow \infty$ as $|r| \rightarrow \infty$,

(3) $w(r) = \rho'(r)/r$ is nonincreasing in $|r|$, and

(4) $w(r) = \rho'(r)/r$ is bounded for all r.

For L_p estimation, the required objective function is:

$$\rho(r) = |r|^p; \ |r| > \varepsilon$$

$$= |r|^p; \ |r| \leq \varepsilon; \ \varepsilon \text{ some small positive number}$$

Since this function meets conditions (1) - (4) for values of p such that $1 \leq p \leq 2$, global convergence of the algorithm is guaranteed for p in this range. Additionally, local convergence of the algorithm is guaranteed for the above function for $2 < p \leq 3$. For $p > 3$, the algorithm diverges (Byrd and Pyne, 1979).

APPENDIX B

THE GENERALIZED SINGULAR VALUE DECOMPOSITION

The MUSIC algorithm requires the solution to a generalized eigenvalue problem involving two covariance matrices. That is, estimates are sought of the generalized eigenvalues and eigenvectors of the matrix pair ($\mathbf{R}_x$, $\mathbf{R}_b$), associated with a collection of S signal samples $\underline{x}_s$ and S noise samples $\underline{w}_s$. The signal and noise samples are generally "snapshots", or samples at various instants, from an antenna array. The matrices $\mathbf{R}_x$ and $\mathbf{R}_b$ are given by:

$$\mathbf{R}_x \equiv E\{ \underline{x}\, \underline{x}^H \} \qquad (B-1)$$

$$\lambda_{min}\mathbf{R}_b \equiv E\{ \underline{w}\, \underline{w}^H \} \qquad (B-2)$$

where λ_{min} normalizes $\mathbf{R}_b$ such that $tr\{ \mathbf{R}_b \} = N$, and $E\{\}$ denotes statistical expectation over s.

In practice, direct solution of a generalized eigenvalue problem involving matrices of the form $\mathbf{M}^H\mathbf{M}$ or $\mathbf{M}\mathbf{M}^H$ is not only computationally intensive but also can lead to numerical difficulties. This is due to the fact that the condition number of either matrix product is the square of the condition number of the original matrix ($\mathbf{M}$). It is therefore advantageous to devise a method whereby the generalized eigenvalue problem can be solved without requiring the actual formation of the matrix products listed. One approach to this problem is via a Generalized Singular Value Decomposition (GSVD) algorithm (Speiser, 1985).

A GSVD algorithm applied to the matrix pair ($\mathbf{A}$, $\mathbf{B}$) computes matrices $\mathbf{U}$, $\mathbf{Q}$, $\mathbf{Z}$, $\mathbf{D}_A$, and $\mathbf{D}_B$ such that:

$$\mathbf{U}^H\mathbf{A}\mathbf{Z} = \mathbf{D}_A \qquad (B-3)$$

$$\mathbf{Q}^H\mathbf{B}\mathbf{Z} = \mathbf{D}_B \qquad (B-4)$$

where:

U and **Q** are unitary ($\mathbf{U}^H\mathbf{U} = \mathbf{I}$, $\mathbf{Q}^H\mathbf{Q} = \mathbf{I}$),

Z is non-singular, and

$\mathbf{D}_A$ and $\mathbf{D}_B$ are real and diagonal.

If the conjugate transposes of Equations (B-3) and (B-4) are evaluated and postmultiplied by their original forms, the following are obtained:

$$\mathbf{Z}^H\mathbf{A}^H\mathbf{A}\mathbf{Z} = (\mathbf{D}_A)^2 \qquad (B-5)$$

$$\mathbf{Z}^H\mathbf{B}^H\mathbf{B}\mathbf{Z} = (\mathbf{D}_B)^2 \qquad (B-6)$$

Now, multiplying Equation (B-6) by λ_n and subtracting it from Equation (B-5) gives:

$$\mathbf{Z}^H(\mathbf{A}^H\mathbf{A} - \lambda_n\mathbf{B}^H\mathbf{B})\mathbf{Z} = (\mathbf{D}_A)^2 - \lambda_n(\mathbf{D}_B)^2 \qquad (B-7)$$

Evaluating Equation (B-7) with $\lambda_n = d_{A,n}^2 / d_{B,n}^2$, where $d_{A,n}$ and $d_{B,n}$ are the n^{th} elements of $\mathbf{D}_A$ and $\mathbf{D}_B$, respectively, produces a singular matrix on the right-hand side. This is true because the difference on the right-hand side of Equation (B-7) is diagonal, with a zero entry at the n^{th} (diagonal) element. On the left-hand side, since **Z** is non-singular, the expression in parenthesis must be singular. Thus:

$$|\mathbf{A}^H\mathbf{A} - \lambda_n\mathbf{B}^H\mathbf{B}| = 0 \qquad (B-8)$$

This can be true (for non-singular $\mathbf{A}^H\mathbf{A}$ and $\mathbf{B}^H\mathbf{B}$) only for λ_n a generalized eigenvalue of the matrix pair ($\mathbf{A}^H\mathbf{A}$, $\mathbf{B}^H\mathbf{B}$). Thus, the generalized eigenvalues of the matrix pair ($\mathbf{A}^H\mathbf{A}$, $\mathbf{B}^H\mathbf{B}$) are given by the ratios of the squares of the singular values $d_{A,n}$ and $d_{B,m}$, above.

Now consider the n^{th} diagonal entry in Equation (B-7):

$$\underline{z}_n^H(\mathbf{A}^H\mathbf{A} - \lambda_n\mathbf{B}^H\mathbf{B})\underline{z}_n = d_{A,n}^2 - \lambda_n d_{B,n}^2 \qquad (B-9)$$

If this equation is evaluated with $\lambda_n = d_{A,n}^2 / d_{B,n}^2$, the result is 0. This indicates that the generalized eigenvectors of the matrix pair ($\mathbf{A}^H\mathbf{A}$, $\mathbf{B}^H\mathbf{B}$) are the columns of **Z**.

Thus it is clear that an algorithm which computes a generalized SVD can also be used to produce the generalized eigenvalues and eigenvectors of a matrix pair of the form described above.

It is, in fact, not even necessary to deal with the matrices **A** and **B** directly, if this is not desired. Such a situation could occur, for instance, if the matrices of interest had a very large number of rows but relatively few columns. In such a case it would be preferable to deal with matrices whose size was strictly deter - mined by the number of columns, rather than the number of rows, in the mat - rices of interest.

Consider an S x N matrix **A** of rank N with $S \geq N$ (that is, the columns of **A** are linearly independent). A QR decomposition can be applied to such a matrix to produce:

$$\mathbf{A}_{S \times N} = \mathbf{Q}_{S \times N}\mathbf{T}_{N \times N} \qquad (B-10)$$

where the columns of **Q** are orthonormal and **T** is upper triangular and invert - ible. Evaluating $\mathbf{A}^H\mathbf{A}$ gives:

$$\mathbf{A}^H\mathbf{A} = \mathbf{T}^H\mathbf{Q}^H\mathbf{Q}\mathbf{T} = \mathbf{T}^H\mathbf{T} \qquad (B-11)$$

(since $\mathbf{Q}^H\mathbf{Q}$ is an N x N identity matrix). Thus, using **T** in the generalized singu - lar value decomposition is exactly equivalent to using the full **A** matrix (as far as concerns the generalized eigenvalue problem).

So it is evident that, except for an initial QR decomposition step, it is pos - sible to deal exclusively with comparitively small matrices and still solve the generalized eigenvalue problem for the matrix pair ($\mathbf{A}^H\mathbf{A}$, $\mathbf{B}^H\mathbf{B}$) without forming the actual matrix products.

To apply the above approach to the matrices used in the MUSIC algorithm, it is only necessary to form the matrices **X** and **W** from the signal and noise vector samples as follows:

$$\mathbf{X} \equiv \frac{1}{\sqrt{I}} \begin{bmatrix} \underline{x}_1^H \\ \underline{x}_2^H \\ \vdots \\ \underline{x}_I^H \end{bmatrix}; \mathbf{W} \equiv \frac{1}{\sqrt{I\lambda_{min}}} \begin{bmatrix} \underline{w}_1^H \\ \underline{w}_2^H \\ \vdots \\ \underline{w}_I^H \end{bmatrix} \qquad (B-12)$$

Application of a generalized SVD approach as described above (either with or without a preliminary QR decomposition step) determines the generalized eigenvalues and eigenvectors of the matrix pair ($\mathbf{X}^H\mathbf{X}$, $\mathbf{W}^H\mathbf{W}$). From Equation (B-12) it can be seen that:

$$\mathbf{X}^H\mathbf{X} = \frac{1}{S} \sum_{s=1}^{S} \underline{x}_s \underline{x}_s^H \qquad (B-13)$$

$$\lambda_{min}\mathbf{W}^H\mathbf{W} = \frac{1}{S} \sum_{s=1}^{S} \underline{w}_s \underline{w}_s^H \qquad (B-14)$$

That is, $\mathbf{X}^H\mathbf{X} \approx \mathbf{R}_x$ and $\mathbf{W}^H\mathbf{W} \approx \mathbf{R}_b$. Thus, the approximations desired can be obtained using a generalized SVD approach.

At least two generalized SVD algorithms are available, a comparatively older one due to Van Loan and a more recent one by Paige (Van Loan, 1985) (Paige, 1986). Van Loan's approach, when applied following a QR decomposition, has the possible advantage of producing an identity matrix for $\mathbf{D}_A$. However, it also involves an explicit matrix inversion, which could lead to numerical difficulty. Although the approach due to Paige is less straightforward, it does not involve any explicit matrix inversions, and so is likely to be the more robust algorithm. Also, a parallel implementation of Paige's algorithm has been de-

veloped by Heath, et. al. (1986). Paige's method uses a preliminary QR decom - position step.

Van Loan's approach was implemented in the simulation program de - scribed in Appendix D. It can be applied to the MUSIC algorithm's generalized eigenvalue problem using the following steps:

1) Perform QR decompositions of **X** and **W** (defined above):

   $\mathbf{X} = \mathbf{L}_X\mathbf{T}_X$; $\mathbf{L}_X^H\mathbf{L}_X = \mathbf{I}$; $\mathbf{T}_X$ upper triangular, invertible.

   $\mathbf{W} = \mathbf{L}_W\mathbf{T}_W$; $\mathbf{L}_W^H\mathbf{L}_W = \mathbf{I}$; $\mathbf{T}_W$ upper triangular, invertible.

2) Compute (normal) singular value decomposition of $\mathbf{T}_W\mathbf{T}_X^{-1}$:

   $\mathbf{T}_W\mathbf{T}_X^{-1} = \mathbf{Q}\mathbf{D}_B\mathbf{U}^H$; $\mathbf{Q}^H\mathbf{Q} = \mathbf{I}$; $\mathbf{U}^H\mathbf{U} = \mathbf{I}$; $\mathbf{D}_B$ real, diagonal.

3) Set $\mathbf{Z} = \mathbf{T}_X^{-1}\mathbf{U}$.

The matrices **U**, **Q**, and **Z** computed above perform the desired general - ized singular value decomposition, which can be seen as follows:

$\mathbf{U}^H\mathbf{T}_X\mathbf{Z} = \mathbf{U}^H\mathbf{T}_X\mathbf{T}_X^{-1}\mathbf{U} = \mathbf{U}^H\mathbf{U} = \mathbf{I}$ (which is real & diagonal).

$\mathbf{Q}^H\mathbf{T}_W\mathbf{Z} = \mathbf{Q}^H(\mathbf{T}_W\mathbf{T}_X^{-1})\mathbf{U} = \mathbf{Q}^H(\mathbf{Q}\mathbf{D}_B\mathbf{U}^H)\mathbf{U} = (\mathbf{Q}^H\mathbf{Q})\mathbf{D}_B(\mathbf{U}^H\mathbf{U}) = \mathbf{D}_B$

Thus, the computed **U**, **Q**, and **Z** comprise a generalized singular value decomposition of the triangular matrices $\mathbf{T}_X$ and $\mathbf{T}_W$. This, as shown above, suf - fices to determine the generalized eigenvalues and eigenvectors of the matrix product pair ($\mathbf{X}^H\mathbf{X}$, $\mathbf{W}^H\mathbf{W}$), which is the desired result.

APPENDIX C

THE MINIMUM NORM ESTIMATE OF A VECTOR

In applying the L_p norm to the estimation problems addressed in this dissertation, it becomes necessary in several situations to evaluate expressions of the form $(\mathbf{A}^H\mathbf{W}\mathbf{A})^{-1}\mathbf{A}^H\mathbf{W}$, where $\mathbf{W}$ is real, nonsingular, and diagonal. In order to avoid an explicit matrix inversion in the evaluation of the above expression, a procedure based on the Moore-Penrose psuedo-inverse is used. The approach proceeds as follows:

$$(\mathbf{A}^H\mathbf{W}\mathbf{A})^{-1}\mathbf{A}^H\mathbf{W} = \left(\mathbf{A}^H\mathbf{W}^{\frac{1}{2}}\mathbf{W}^{\frac{1}{2}}\mathbf{A}\right)^{-1}\mathbf{A}^H\mathbf{W}^{\frac{1}{2}}\mathbf{W}^{\frac{1}{2}}$$

and, since $\mathbf{W}$ is real and diagonal:

$$(\mathbf{A}^H\mathbf{W}\mathbf{A})^{-1}\mathbf{A}^H\mathbf{W} = \left(\left(\mathbf{W}^{\frac{1}{2}}\mathbf{A}\right)^H\mathbf{W}^{\frac{1}{2}}\mathbf{A}\right)^{-1}\left(\mathbf{W}^{\frac{1}{2}}\mathbf{A}\right)^H\mathbf{W}^{\frac{1}{2}}$$

Defining:

$$\mathbf{B} \equiv \left(\mathbf{W}^{\frac{1}{2}}\mathbf{A}\right)^H$$

gives:

$$(\mathbf{A}^H\mathbf{W}\mathbf{A})^{-1}\mathbf{A}^H\mathbf{W} = (\mathbf{B}^H\mathbf{B})^{-1}\mathbf{B}^H\mathbf{W}^{\frac{1}{2}}$$

Now $(\mathbf{B}^H\mathbf{B})^{-1}\mathbf{B}^H$ is the Moore-Penrose psuedo-inverse of $\mathbf{B}$, and computing the square root of $\mathbf{W}$ requires little more work than would computing $\mathbf{W}$ itself, since the elements of $\mathbf{W}$ are generally exponentials. This implies that, except

for the addition of one initial and one final diagonal matrix multiply at each itera - tion, the same operations can be used to form L_p estimates as are used to form L_2 estimates.

Actual evaluation of the Moore-Penrose psuedo-inverse of **B** is accomp - lished using an algorithm based on the singular value decomposition. Although proofs that such an algorithm produces minimum-norm vector estimates are fairly common for the real case, the more general case for complex vectors and matrices is covered only rarely (Graybill, 1969). Therefore, a proof for complex data is included here.

Problem Statement

It is desired to find the minimum (L_2) norm solution to the equation:

$$\mathbf{A}\underline{x} = \underline{b}$$

where **A** is N x M, $N \geq M$, of rank $r \leq M$. That is, the solution $\underline{x} = \underline{x}_0$ is sought such that:

$$(\mathbf{A}\underline{x} - \underline{b})^H(\mathbf{A}\underline{x} - \underline{b}) \geq (\mathbf{A}\underline{x}_0 - b)^H(\mathbf{A}\underline{x}_0 - \underline{b}), \text{ all } \underline{x} \text{ in } C^N; \text{ and}$$

$$\underline{x}^H\underline{x} > \underline{x}_0^H\underline{x}_0 \text{ for } \underline{x} \neq \underline{x}_0 \text{ and } (\mathbf{A}\underline{x} - \underline{b})^H(\mathbf{A}\underline{x} - \underline{b}) = (\mathbf{A}\underline{x}_0 - \underline{b})^H(\mathbf{A}\underline{x}_0 - \underline{b}).$$

Solution

The singular value decomposition of **A** yields $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^H$ with **U** N x N, $\mathbf{U}^H\mathbf{U} = \mathbf{U}\mathbf{U}^H = \mathbf{I}_{N \times N}$; **V** M x M, $\mathbf{V}^H\mathbf{V} = \mathbf{V}\mathbf{V}^H = \mathbf{I}_{M \times M}$; and Σ N x M, real, diagonal, as follows:

$$\begin{bmatrix} \sigma_1 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & \sigma_2 & \ddots & & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & & & \vdots \\ \vdots & & \ddots & \sigma_r & \ddots & & \vdots \\ \vdots & & & \ddots & 0 & \ddots & \vdots \\ \vdots & & & & \ddots & \ddots & 0 \\ \vdots & & & & & \ddots & 0 \\ \vdots & & & & & & 0 \\ \vdots & & & & & & \vdots \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \end{bmatrix}$$

The desired minimum norm solution is $\underline{x}_0 = \mathbf{A}^- \underline{b}$, where $\mathbf{A}^- = \mathbf{V}\mathrm{X}^T\mathbf{U}^H$, and the N x M matrix X is:

$$\begin{bmatrix} \sigma_1^{-1} & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & \sigma_2^{-1} & \ddots & & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & & & \vdots \\ \vdots & & \ddots & \sigma_r^{-1} & \ddots & & \vdots \\ \vdots & & & \ddots & 0 & \ddots & \vdots \\ \vdots & & & & \ddots & \ddots & 0 \\ \vdots & & & & & \ddots & 0 \\ \vdots & & & & & & 0 \\ \vdots & & & & & & \vdots \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \end{bmatrix}$$

Proof

First, the proposed $\mathbf{A}^-$ will be shown to have the following properties:

1) $\mathbf{AA}^-$ is Hermitian,

2) $\mathbf{A}^-\mathbf{A}$ is Hermitian,

3) $\mathbf{AA}^-\mathbf{A} = \mathbf{A}$

4) $\mathbf{A}^-\mathbf{AA}^- = \mathbf{A}^-$

To demonstrate the above properties, note that:

$\Sigma X^T = (\Sigma X^T)^T$ = N x N real diagonal matrix containing r ones:

$$\begin{bmatrix} 1_{(1)} & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & 1_{(2)} & \ddots & & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & & & \vdots \\ \vdots & & \ddots & 1_{(r)} & \ddots & & \vdots \\ \vdots & & & \ddots & 0 & \ddots & \vdots \\ \vdots & & & & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & 0_{(N)} \end{bmatrix}$$

$X^T\Sigma = (X^T\Sigma)^T$ = M x M real diagonal matrix containing r ones:

$$\begin{bmatrix} 1_{(1)} & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & 1_{(2)} & \ddots & & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & & & \vdots \\ \vdots & & \ddots & 1_{(r)} & \ddots & & \vdots \\ \vdots & & & \ddots & 0 & \ddots & \vdots \\ \vdots & & & & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & 0_{(M)} \end{bmatrix}$$

$\Sigma(X^T\Sigma) = \Sigma$; $(X^T\Sigma)X^T = X^T$.

To prove property (1):

$$\begin{aligned} [\mathbf{A}\mathbf{A}^-]^H &= [(\mathbf{U}\Sigma\mathbf{V}^H)(\mathbf{V}X^T\mathbf{U}^H)]^H \\ &= [\mathbf{U}\Sigma\mathbf{V}^H\mathbf{V}X^T\mathbf{U}^H]^H \\ &= [\mathbf{U}(\Sigma X^T)\mathbf{U}^H]^H \\ &= \mathbf{U}(\Sigma X^T)^T\mathbf{U}^H \\ &= \mathbf{A}\mathbf{A}^- \end{aligned}$$

To prove property (2):

$$\begin{aligned} [\mathbf{A}^-\mathbf{A}]^H &= [(\mathbf{V}X^T\mathbf{U}^H)(\mathbf{U}\Sigma\mathbf{V}^H)]^H \\ &= [\mathbf{V}X^T\mathbf{U}^H\mathbf{U}\Sigma\mathbf{V}^H]^H \\ &= [\mathbf{V}(X^T\Sigma)\mathbf{V}^H]^H \end{aligned}$$

$$= \mathbf{V}(\mathrm{X}^{\mathrm{T}}\Sigma)^{\mathrm{T}}\mathbf{V}^{\mathrm{H}}$$

$$= \mathbf{A}^{-}\mathbf{A}$$

To prove property (3):

$$\mathbf{A}\mathbf{A}^{-}\mathbf{A} = \mathbf{U}\Sigma\mathrm{X}^{\mathrm{T}}\mathbf{U}^{\mathrm{H}}\mathbf{U}\Sigma\mathbf{V}^{\mathrm{H}}$$

$$= \mathbf{U}\Sigma(\mathrm{X}^{\mathrm{T}}\Sigma)\mathbf{V}^{\mathrm{H}}$$

$$= \mathbf{U}\Sigma\mathbf{V}^{\mathrm{H}}$$

$$= \mathbf{A}$$

To prove property (4):

$$\mathbf{A}^{-}\mathbf{A}\mathbf{A}^{-} = \mathbf{V}\mathrm{X}^{\mathrm{T}}\Sigma\mathbf{V}^{\mathrm{H}}\mathbf{V}\mathrm{X}^{\mathrm{T}}\mathbf{U}^{\mathrm{H}}$$

$$= \mathbf{V}(\mathrm{X}^{\mathrm{T}}\Sigma)\mathrm{X}^{\mathrm{T}}\mathbf{U}^{\mathrm{H}}$$

$$= \mathbf{V}\mathrm{X}^{\mathrm{T}}\mathbf{U}^{\mathrm{H}}$$

$$= \mathbf{A}^{-}$$

Properties (1) - (4) can be used to prove that the proposed solution is the minimum norm solution as follows:

$$(\mathbf{A}\underline{x} - \underline{b})^{\mathrm{H}}(\mathbf{A}\underline{x} - \underline{b}) = (\mathbf{A}\underline{x} - \mathbf{A}\mathbf{A}^{-}\underline{b} + \mathbf{A}\mathbf{A}^{-}\underline{b} - \underline{b})^{\mathrm{H}}(\mathbf{A}\underline{x} - \mathbf{A}\mathbf{A}^{-}\underline{b} + \mathbf{A}\mathbf{A}^{-}\underline{b} - \underline{b})$$

$$= [\mathbf{A}(\underline{x} - \mathbf{A}^{-}\underline{b}) + (\mathbf{A}\mathbf{A}^{-} - \mathbf{I})\underline{b}]^{\mathrm{H}}[\mathbf{A}(\underline{x} - \mathbf{A}^{-}\underline{b}) + (\mathbf{A}\mathbf{A}^{-} - \mathbf{I})\underline{b}]$$

$$= [\mathbf{A}(\underline{x} - \mathbf{A}^{-}\underline{b})]^{\mathrm{H}}[\mathbf{A}(\underline{x} - \mathbf{A}^{-}\underline{b})] + [\mathbf{A}(\underline{x} - \mathbf{A}^{-}\underline{b})]^{\mathrm{H}}[(\mathbf{A}\mathbf{A}^{-} - \mathbf{I})\underline{b}]$$
$$+ [(\mathbf{A}\mathbf{A}^{-} - \mathbf{I})\underline{b}]^{\mathrm{H}}[\mathbf{A}(\underline{x} - \mathbf{A}^{-}\underline{b})] + [(\mathbf{A}\mathbf{A}^{-} - \mathbf{I})\underline{b}]^{\mathrm{H}}[(\mathbf{A}\mathbf{A}^{-} - \mathbf{I})\underline{b}]$$

$$= [\mathbf{A}(\underline{x} - \mathbf{A}^{-}\underline{b})]^{\mathrm{H}}[\mathbf{A}(\underline{x} - \mathbf{A}^{-}\underline{b})] + [(\mathbf{A}\mathbf{A}^{-} - \mathbf{I})\underline{b}]^{\mathrm{H}}[(\mathbf{A}\mathbf{A}^{-} - \mathbf{I})\underline{b}]$$
$$+ (\mathbf{A}\mathbf{A}^{-}\underline{b} - \mathbf{I}\,\underline{b})^{\mathrm{H}}(\mathbf{A}\underline{x} - \mathbf{A}\mathbf{A}^{-}\underline{b}) + (\mathbf{A}\underline{x} - \mathbf{A}\mathbf{A}^{-}\underline{b})^{\mathrm{H}}(\mathbf{A}\mathbf{A}^{-}\underline{b} - \mathbf{I}\,\underline{b})$$

$$= [\mathbf{A}(\underline{x} - \mathbf{A}^{-}\underline{b})]^{\mathrm{H}}[\mathbf{A}(\underline{x} - \mathbf{A}^{-}\underline{b})] + [(\mathbf{A}\mathbf{A}^{-} - \mathbf{I})\underline{b}]^{\mathrm{H}}[(\mathbf{A}\mathbf{A}^{-} - \mathbf{I})\underline{b}]$$
$$+ (\mathbf{A}\mathbf{A}^{-}\underline{b})^{\mathrm{H}}\mathbf{A}\underline{x} - (\mathbf{A}\mathbf{A}^{-}\underline{b})^{\mathrm{H}}\mathbf{A}\mathbf{A}^{-}\underline{b} - \underline{b}^{\mathrm{H}}\mathbf{A}\underline{x} + \underline{b}^{\mathrm{H}}\mathbf{A}\mathbf{A}^{-}\underline{b}$$
$$+ (\mathbf{A}\underline{x})^{\mathrm{H}}\mathbf{A}\mathbf{A}^{-}\underline{b} - (\mathbf{A}\underline{x})^{\mathrm{H}}\underline{b} - (\mathbf{A}\mathbf{A}^{-}\underline{b})^{\mathrm{H}}\mathbf{A}\mathbf{A}^{-}\underline{b} + (\mathbf{A}\mathbf{A}^{-}\underline{b})^{\mathrm{H}}\underline{b}$$

$$= [\mathbf{A}(\underline{x} - \mathbf{A}^{-}\underline{b})]^{\mathrm{H}}[\mathbf{A}(\underline{x} - \mathbf{A}^{-}\underline{b})] + [(\mathbf{A}\mathbf{A}^{-} - \mathbf{I})\underline{b}]^{\mathrm{H}}[(\mathbf{A}\mathbf{A}^{-} - \mathbf{I})\underline{b}]$$
$$+ \underline{b}^{\mathrm{H}}(\mathbf{A}\mathbf{A}^{-})^{\mathrm{H}}\mathbf{A}\underline{x} - \underline{b}^{\mathrm{H}}(\mathbf{A}\mathbf{A}^{-})^{\mathrm{H}}\mathbf{A}\mathbf{A}^{-}\underline{b} - \underline{b}^{\mathrm{H}}\mathbf{A}\underline{x} + \underline{b}^{\mathrm{H}}\mathbf{A}\mathbf{A}^{-}\underline{b}$$
$$+ \underline{x}^{\mathrm{H}}\mathbf{A}^{\mathrm{H}}\mathbf{A}\mathbf{A}^{-}\underline{b} - \underline{x}^{\mathrm{H}}\mathbf{A}^{\mathrm{H}}\underline{b} - \underline{b}^{\mathrm{H}}(\mathbf{A}\mathbf{A}^{-})^{\mathrm{H}}\mathbf{A}\mathbf{A}^{-}\underline{b} + \underline{b}^{\mathrm{H}}(\mathbf{A}\mathbf{A}^{-})^{\mathrm{H}}\underline{b}$$

$$= [\mathbf{A}(\underline{x} - \mathbf{A}^{-}\underline{b})]^{H}[\mathbf{A}(\underline{x} - \mathbf{A}^{-}\underline{b})] + [(\mathbf{A}\mathbf{A}^{-} - \mathbf{I})\underline{b}]^{H}[(\mathbf{A}\mathbf{A}^{-} - \mathbf{I})\underline{b}]$$
$$+ \underline{b}^{H}\mathbf{A}\mathbf{A}^{-}\mathbf{A}\underline{x} - \underline{b}^{H}(\mathbf{A}\mathbf{A}^{-}\mathbf{A})\mathbf{A}^{-}\underline{b} - \underline{b}^{H}\mathbf{A}\underline{x} + \underline{b}^{H}\mathbf{A}\mathbf{A}^{-}\underline{b}$$
$$+ \underline{x}^{H}\mathbf{A}^{H}(\mathbf{A}\mathbf{A}^{-})^{H}\underline{b} - \underline{x}^{H}\mathbf{A}^{H}\underline{b} - \underline{b}^{H}(\mathbf{A}\mathbf{A}^{-}\mathbf{A})\mathbf{A}^{-}\underline{b} + \underline{b}^{H}\mathbf{A}\mathbf{A}^{-}\underline{b}$$
$$= [\mathbf{A}(\underline{x} - \mathbf{A}^{-}\underline{b})]^{H}[\mathbf{A}(\underline{x} - \mathbf{A}^{-}\underline{b})] + [(\mathbf{A}\mathbf{A}^{-} - \mathbf{I})\underline{b}]^{H}[(\mathbf{A}\mathbf{A}^{-} - \mathbf{I})\underline{b}]$$
$$+ \underline{b}^{H}\mathbf{A}\underline{x} - \underline{b}^{H}\mathbf{A}\mathbf{A}^{-}\underline{b} - \underline{b}^{H}\mathbf{A}\underline{x} + \underline{b}^{H}\mathbf{A}\mathbf{A}^{-}\underline{b}$$
$$+ \underline{x}^{H}(\mathbf{A}\mathbf{A}^{-}\mathbf{A})^{H}\underline{b} - \underline{x}^{H}\mathbf{A}^{H}\underline{b} - \underline{b}^{H}\mathbf{A}\mathbf{A}^{-}\underline{b} + \underline{b}^{H}\mathbf{A}\mathbf{A}^{-}\underline{b}$$
$$= [\mathbf{A}(\underline{x} - \mathbf{A}^{-}\underline{b})]^{H}[\mathbf{A}(\underline{x} - \mathbf{A}^{-}\underline{b})] + [(\mathbf{A}\mathbf{A}^{-} - \mathbf{I})\underline{b}]^{H}[(\mathbf{A}\mathbf{A}^{-} - \mathbf{I})\underline{b}]$$
$$+ \underline{x}^{H}\mathbf{A}^{H}\underline{b} - \underline{x}^{H}\mathbf{A}^{H}\underline{b}$$

$$(\mathbf{A}\underline{x} - \underline{b})^{H}(\mathbf{A}\underline{x} - \underline{b}) = [\mathbf{A}(\underline{x} - \mathbf{A}^{-}\underline{b})]^{H}[\mathbf{A}(\underline{x} - \mathbf{A}^{-}\underline{b})] + [(\mathbf{A}\mathbf{A}^{-} - \mathbf{I})\underline{b}]^{H}[(\mathbf{A}\mathbf{A}^{-} - \mathbf{I})\underline{b}]$$

$$(\mathbf{A}\underline{x} - \underline{b})^{H}(\mathbf{A}\underline{x} - \underline{b}) \geq [(\mathbf{A}\mathbf{A}^{-} - \mathbf{I})\underline{b}]^{H}[(\mathbf{A}\mathbf{A}^{-} - \mathbf{I})\underline{b}] \qquad \text{(all } \underline{x} \text{ in } C^{N})$$

$$(\mathbf{A}\underline{x} - \underline{b})^{H}(\mathbf{A}\underline{x} - \underline{b}) \geq (\mathbf{A}\mathbf{A}^{-}\underline{b} - \underline{b})^{H}(\mathbf{A}\mathbf{A}^{-}\underline{b} - \underline{b})$$

For $\underline{x}_0 = \mathbf{A}^{-}\underline{b}$:

$$(\mathbf{A}\underline{x} - \underline{b})^{H}(\mathbf{A}\underline{x} - \underline{b}) \geq (\mathbf{A}\underline{x}_0 - \underline{b})^{H}(\mathbf{A}\underline{x}_0 - \underline{b}) \qquad \text{(all } \underline{x} \text{ in } C^{N})$$

Further:

$$(\mathbf{A}\underline{x} - \underline{b})^{H}(\mathbf{A}\underline{x} - \underline{b}) = (\mathbf{A}\underline{x}_0 - \underline{b})^{H}(\mathbf{A}\underline{x}_0 - \underline{b}) \qquad \text{(C - 1)}$$

iff $[\mathbf{A}(\underline{x} - \mathbf{A}^{-}\underline{b})]^{H}[\mathbf{A}(\underline{x} - \mathbf{A}^{-}\underline{b})] = 0$; i.e. $\mathbf{A}\underline{x} = \mathbf{A}\mathbf{A}^{-}\underline{b}$

Consider the product:

$$[\mathbf{A}^{-}\underline{b} + (\mathbf{I} - \mathbf{A}^{-}\mathbf{A})\underline{x}]^{H}[\mathbf{A}^{-}\underline{b} + (\mathbf{I} - \mathbf{A}^{-}\mathbf{A})\underline{x}]$$
$$= [\mathbf{A}^{-}\underline{b} + (\mathbf{I} - \mathbf{A}^{-}\mathbf{A})\underline{x}]^{H}\mathbf{A}^{-}\underline{b} + [\mathbf{A}^{-}\underline{b} + (\mathbf{I} - \mathbf{A}^{-}\mathbf{A})\underline{x}]^{H}(\mathbf{I} - \mathbf{A}^{-}\mathbf{A})\underline{x}$$
$$= (\mathbf{A}^{-}\underline{b})^{H}\mathbf{A}^{-}\underline{b} + [(\mathbf{I} - \mathbf{A}^{-}\mathbf{A})\underline{x}]^{H}\mathbf{A}^{-}\underline{b}$$
$$+ (\mathbf{A}^{-}\underline{b})^{H}(\mathbf{I} - \mathbf{A}^{-}\mathbf{A})\underline{x} + [(\mathbf{I} - \mathbf{A}^{-}\mathbf{A})\underline{x}]^{H}(\mathbf{I} - \mathbf{A}^{-}\mathbf{A})\underline{x}$$
$$= (\mathbf{A}^{-}\underline{b})^{H}(\mathbf{A}^{-}\underline{b}) + [(\mathbf{I} - \mathbf{A}^{-}\mathbf{A})\underline{x}]^{H}[(\mathbf{I} - \mathbf{A}^{-}\mathbf{A})\underline{x}]$$
$$+ (\mathbf{A}^{-}\underline{b})^{H}(\underline{x} - \mathbf{A}^{-}\mathbf{A}\underline{x}) + (\underline{x} - \mathbf{A}^{-}\mathbf{A}\underline{x})^{H}\mathbf{A}^{-}\underline{b}$$
$$= (\mathbf{A}^{-}\underline{b})^{H}(\mathbf{A}^{-}\underline{b}) + [(\mathbf{I} - \mathbf{A}^{-}\mathbf{A})\underline{x}]^{H}[(\mathbf{I} - \mathbf{A}^{-}\mathbf{A})\underline{x}]$$
$$+ \underline{b}^{H}(\mathbf{A}^{-})^{H}\underline{x} - \underline{b}^{H}(\mathbf{A}^{-})^{H}\mathbf{A}^{-}\mathbf{A}\underline{x} + \underline{x}^{H}\mathbf{A}^{-}\underline{b} - \underline{x}^{H}(\mathbf{A}^{-}\mathbf{A})^{H}\mathbf{A}^{-}\underline{b}$$
$$= (\mathbf{A}^{-}\underline{b})^{H}(\mathbf{A}^{-}\underline{b}) + [(\mathbf{I} - \mathbf{A}^{-}\mathbf{A})\underline{x}]^{H}[(\mathbf{I} - \mathbf{A}^{-}\mathbf{A})\underline{x}]$$
$$+ \underline{b}^{H}(\mathbf{A}^{-})^{H}\underline{x} - \underline{b}^{H}(\mathbf{A}^{-})^{H}(\mathbf{A}^{-}\mathbf{A})^{H}\underline{x} + \underline{x}^{H}\mathbf{A}^{-}\underline{b} - \underline{x}^{H}\mathbf{A}^{-}\underline{b}$$

$$= (\mathbf{A}^- \underline{b})^H(\mathbf{A}^- \underline{b}) + [(\mathbf{I} - \mathbf{A}^-\mathbf{A})\underline{x}]^H[(\mathbf{I} - \mathbf{A}^-\mathbf{A})\underline{x}]$$

$$+ \underline{b}^H(\mathbf{A}^-)^H\underline{x} - \underline{b}^H(\mathbf{A}^-\mathbf{A}\mathbf{A}^-)^H\underline{x}$$

$$= (\mathbf{A}^- \underline{b})^H(\mathbf{A}^- \underline{b}) + [(\mathbf{I} - \mathbf{A}^-\mathbf{A})\underline{x}]^H[(\mathbf{I} - \mathbf{A}^-\mathbf{A})\underline{x}]$$

$$+ \underline{b}^H(\mathbf{A}^-)^H\underline{x} - \underline{b}^H(\mathbf{A}^-)^H\underline{x}$$

$$[\mathbf{A}^- \underline{b} + (\mathbf{I} - \mathbf{A}^-\mathbf{A})\underline{x}]^H[\mathbf{A}^- \underline{b} + (\mathbf{I} - \mathbf{A}^-\mathbf{A})\underline{x}]$$

$$= (\mathbf{A}^- \underline{b})^H(\mathbf{A}^- \underline{b}) + [(\mathbf{I} - \mathbf{A}^-\mathbf{A})\underline{x}]^H[(\mathbf{I} - \mathbf{A}^-\mathbf{A})\underline{x}] \qquad \text{(all } \underline{x} \text{ in } C^N\text{)}$$

For $\mathbf{A}\mathbf{A}^- \underline{b} = \mathbf{A}\underline{x}$, we have $\mathbf{A}^- \underline{b} = \mathbf{A}^-\mathbf{A}\underline{x}$, so for equality (C - 1) to hold:

$$(\mathbf{A}^-\mathbf{A}\underline{x} + \underline{x} - \mathbf{A}^-\mathbf{A}\underline{x})^H(\mathbf{A}^-\mathbf{A}\underline{x} + \underline{x} - \mathbf{A}^-\mathbf{A}\underline{x}) = \underline{x}^H\underline{x}$$

$$= (\mathbf{A}^- \underline{b})^H(\mathbf{A}^- \underline{b}) + (\underline{x} - \mathbf{A}^- \underline{b})^H(\underline{x} - \mathbf{A}^- \underline{b})$$

So, for all $\underline{x}$ in C^N such that equality (C - 1) holds, $\underline{x}^H\underline{x} > \underline{x}_0^H\underline{x}_0$ if $\underline{x} \neq \underline{x}_0$.

Q.E.D.

APPENDIX D

SIMULATION PROGRAM DESCRIPTION

The experimental results given in Chapter IV were produced using a simulation program developed for the OSU ECEN VAX 11/750. This program permits direction of arrival (DOA) determination experiments to be performed using computer-generated data. The program was developed in FORTRAN-77, for the VMS operating system (Version 4.5), and consists of an interactive rou - tine and four batch routines. User interface functions are performed by the inter - active routine, while the four batch routines perform various estimation functions associated with DOA estimation.

The five routines communicate information concerning DOA estimation experiments in progress by way of two index files and multiple direct and se - quential files on disk. The MUSIC index file records information concerning in - dividual experiments, such as number of antenna elements, arriving waves, and noise generators to simulate, the number of samples to process, sample inter - val, IRLS parameters, and the range of p values to support. The "spawn" index file records the status of the various batch routines used to perform the esti- mation. The other files (sequential and direct) used for communication record information about each experiment in progress, such as antenna element posi - tions; wave strengths, angles of arrival and modulation envelope characteris - tics; noise generator strengths, distributions, and correlation coefficients; esti - mated directions of arrival; observation and noise sample vectors and

estimates; and computed values such as eigenvalues, eigenvectors, and co - variance matrices.

Experiments recorded in the MUSIC index file can be linked in a fashion which permits one experiment to produce a number of related experiments. In the comment lines contained in the routines, an experiment which produces another experiment is called a "parent" experiment, and an experiment pro - duced from another experiment is called a "descendant" experiment. This link - ing can encompass multiple levels of "descendancy", such that one experiment can produce a collection of descendant experiments, which in turn can each produce descendant experiments of their own, and so on. When an experiment gives rise to multiple levels of descendant experiments, the original experiment is called the "ancestor" experiment for each of the experiments in its de - scendancy chains.

The ancestor-descendant linkage mechanism is used to perform the iterative DOA revision described in Chapter III of this dissertation. This is ac - complished as follows. Once the generalized eigenvalue problem is solved for a particular experiment, the operator is given the opportunity to select estimated directions of arrival from the experiment's DOA spectra. Once estimated DOAs are determined, the operator can initiate processing of new experiments which use the original experiment's basic parameters (number of antenna elements, etc.), but whose generalized eigenvalue problems are adjusted using the esti - mated DOAs just obtained. Each new experiment so initiated is a descendant of the original experiment, and is linked to it in the manner described in the pre - vious paragraph. This process can be repeated for each of the descendant experiments if desired, and so the multiple levels of descendancy described above can be produced.

Routine DO_MUSIC

Program DO_MUSIC is designed to run in an interactive environment, and provides the operator interface to the L_p MUSIC algorithms. Lines 1 - 3117 contain comments and INCLUDEd declaration files. Setting the process name for coordination with the batch routines, obtaining logical unit numbers, opening the required index files, and other initialization are performed in lines 3118 - 3199. Lines 3200 - 3208 submit batch routines to perform sample generation, eigenproblem solution, or P matrix estimation as necessary. An index of active experiments is produced in lines 3209 - 3220. Lines 3221 - 3256 present the user with a menu of activities from which to select, and accept the operator's input selecting the activity to perform. Actions which can be performed are dis - cussed in the following paragraphs.

If the operator selects generation of an index of active experiments, lines 3257 - 3264 set an indicator that an index is desired, and return to accept fur - ther operator input.

Lines 3265 - 3594 process a user command to determine DOA estimates for an experiment. User selection of the experiment for which to plot DOA spectra is handled in lines 3265 - 3322. Lines 3323 - 3350 locate the selected experiment's ancestor, if necessary. Lines 3351 - 3408 allocate arrays required for storage of the information necessary to plot DOA spectra, and obtain the necessary information from the appropriate sequential data files. A direct-access scratch file for DOA accumulation is opened in lines 3409 - 3416. Actual DOA estimation is accomplished in lines 3417 - 3547. Lines 3420 - 3438 accept the number of noise eigenvectors to use and the range of p values for which to plot spectra. Spectra plots are produced by subprogram MUSIC_ - PLOT_SPECT called at line 3443, and peak location is accomplished by

subprogram PICK_MUSIC_PEAK invoked at line 3447. Lines 3453 - 3528 initiate descendant experiments as necessary using the estimated DOAs. Finally, lines 3529 - 3594 process operator selections for the disposition of the experiment for which DOAs were estimated, discard space allocated for DOA information storage, and handle any file I/O errors that may have occurred during the DOA estimation process.

Printing of a selected experiment's output file is accomplished in lines 3595 - 3624, if selected by the operator.

Processing of the operator's selection to estimate a source covariance matrix is accomplished in lines 3625 - 3819. Lines 3625 - 3762 accept the operator's selection of which experiment to process, and accumulate the DOA estimates associated with that experiment, its parent, its parent's parent, and so on up to the ancestor experiment. Lines 3763 - 3799 accept the user's choice of the estimation type to be used, and spawn a batch job (MUSIC_PEST) to evaluate the source covariance matrix estimate. File I/O errors are handled in lines 3800 - 3819.

An experiment selected by the operator can be marked for deletion (from the index of experiments) in lines 3820 - 3895, while lines 3896 - 3998 actually delete the index entry and data files for an experiment that has been previously marked for deletion.

Display of results for an experiment is accomplished in lines 3999 - 4579. Lines 3999 - 4130 accept the operator's choice of what results are to be dis - played, and the experiment for which to display the selected results. An array picture is generated in lines 4131 - 4199; lines 4200 - 4303 plot any combina - tion of array output, source signal vector, source vector estimate, or noise vector samples; lines 4304 - 4425 plot selected DOA spectra; and lines 4426 - 4579

plot an operator-selected combination of array output samples, estimated source vector samples, noise vector samples, and DOA spectra.

Lines 4580 - 4626 permit operator re-initializing of an experiment that was accidentally marked for deletion.

If the user wishes to re-run an experiment that was previously entered (creating a new "trial"), this is accomplished in lines 4627 - 4778.

A new experiment is activated by the operator in lines 4779 - 5119. Lines 4779 - 5017 assign default values for the experiment. Default values can be taken from an already existing experiment, or may be set to "standard" de - fault values, at the operator's selection. Operator adjustment of default values is accomplished in lines 5018 - 5039. The experimental parameters under opera - tor control are antenna array geometry, arriving wave characteristics, and noise attributes.

The antenna array to be simulated can consist of any number of antenna elements arranged in any desired locations about a common origin. No con - straints are placed on antenna locations; however, all simulated antenna ele - ments are identical and omnidirectional. Once the receiving array has been described, the operator can specify any number of arriving waves. Wave characteristics under operator control are angle of arrival (azimuth only), signal strength, carrier wavelength, and modulation characteristics. The program sup - ports both amplitude and phase modulation of the received signals, and permits operator selection of modulation envelope types, lengths, duty cycles, and relative phases.

The final simulation aspect under operator control is sample noise characteristics. Gaussian, uniform, and fixed noise generators of operator-selected intensities are available, the outputs from any number of which may be applied to any combination of antenna elements. The noise generators may be

used on either a continuous or intermittent basis, permitting simulation of noise or interference bursts impinging on the array. Two separate noise models are maintained by the program. One model is used in the estimation routines (e.g. DOA determination), and the other is used in actual sample generation. The two models may be identical or different, at the operator's option.

After operator adjustment of default experiment values, the new experiment is actually initiated in lines 5040 - 5107. Lines 5108 - 5119 handle file I/O errors that may occur during experiment generation.

An operator command to exit to the operating system is processed in lines 5120 - 5128, and lines 5129 - 5135 handle an invalid operator selection.

Lines 5136 - 5164 are used at various points in DO_MUSIC to update an experiment's index file entry.

File I/O errors not handled elswhere are processed in lines 5165 - 5260.

Several subprograms are included in the DO_MUSIC source file to accomplish various tasks needed by the operator interface. Function P_INDEX finds and displays records for selected experiments in the experiment index file; function NEXT_REC is used by P_INDEX to locate the next selected index record; batch jobs to be initiated are spawned by subroutine SUBMIT_JOB; and PRINT_EXPM initates printing of output files for selected experiments.

Routine SAMPL_MUSIC

Routine SAMPL_MUSIC, which was designed to run as a batch program, performs the sample generation function for experiments in progress. Lines 1 - 610 perform required initialization functions. Line 483 selects an experiment for which sample generation is required. Input of experiment definition data is accomplished in lines 611 - 660. Lines 661 - 743 compute the observation and noise vector samples, write the samples to the appropriate data files, and print

the problem definition information to the experiment's output file. Lines 744 - 812 compute covariance matrices and the minimum eigenvalue estimate, per - form QR decompositions of the array output and noise sample matrices, write the results of the QR decompositions to the appropriate data file, and print the covariance matrces to the experiment's output file. Lines 813 - 863 update the experiment's index file entry and initiate the eigenvector solution routine if necessary. Lines 864 - 875 terminate processing if no more experiments re - quire sample generation, and lines 876 - 912 process any file I/O errors.

Routine EIG_MUSIC

Routine EIG_MUSIC performs the eigenvector decomposition required by the L_p MUSIC approach. It is designed to run as a batch procedure. Lines 1 - 464 perform required initialization. Selection of an experiment for which to perform eigenvector decomposition, and input of the data necessary to accom - plish the decomposition, is performed in lines 465 - 565. Lines 566 - 597 solve the generalized eigenvalue problem, write the results to appropriate data files, perform a check of the results, and print various computed quantities to the ex - periment's output file. Lines 598 - 643 update the experiment's index file entry, and terminate processing if no more experiments require eigenvector decom - position. Finally, lines 644 - 708 process any file I/O errors which may have oc - curred.

Routine FIN_MUSIC

This routine, designed to operate as a batch routine, obtains estimates of source signal vector samples using observation vector samples and DOA esti - mates determined in DO_MUSIC. Initialization is performed in lines 1 - 482. Lines 483 - 734 select the experiment to be processed and obtain the data

necessary for the estimation process from the appropriate data files. Lines 735 - 746 evaluate the sharpness measures for the DOAs being used, and estimate the antenna array phase shift matrix from the estimated DOAs. Lines 747 - 788 estimate the source signal vector samples and source covariance matrix using the estimation type selected by the operator (in routine DO_MUSIC). Updating of the observation covariance matrix for iterative DOA revision is accomplished in lines 789 - 827. Finally, results are printed to the experiment's output file and written to the appropriate data files in lines 828 - 883. Lines 885 - 938 initiate the eigenvector decomposition routine, update the experiment's index file entry, and terminate processing if no more experiments require sample estimation. File I/O errors are handled in lines 939 - 985.

Routine MUSIC_PEST

This routine, which operates as a batch program, estimates the source covariance matrix for an experiment and evaluates the performance of the esti - mation process. Performance is evaluated by comparing the source covariance matrix estimate with the actual sample source covariance matrix. The actual sample source covariance matrix is computed by approximating the definition:

$$\mathbf{P} \approx \frac{1}{S} \sum_{s=1}^{S} \underline{f}_s \underline{f}_s^H$$

where:

S is the number of samples, and

$\underline{f}_s$ is the $s^{\underline{th}}$ sample of the source signal vector.

The performance comparison is made using an error measure produced from the Frobenius norm of the difference between the matrix estimate and the actual sample source covariance matrix. The error measure used is:

$$\varepsilon = \frac{\sum_j \sum_k \left| \hat{p}_{j,k} - p_{j,k} \right|^2}{\sum_j \sum_k \left| p_{j,k} \right|^2}$$

where:

$\hat{p}_{j,k}$ is the $(j, k)^{th}$ element of the covariance matrix estimate , and

$p_{j,k}$ is the $(j,k)^{th}$ element of the actual sample source covariance matrix.

Lines 1 - 467 perform initialization, and lines 468 - 494 obtain an imput file if any is available. If no input files are available, indicating that no experi - ments require performance evaluation, line 487 terminates execution. Lines 495 - 677 read the information required for the estimation process from the ap - propriate data files. Actual estimation and performance evaluation are con - ducted in lines 678 - 746. Results are written to the selected experiment's out - put file in lines 747 - 762. Finally, lines 764 - 770 delete the processed input file, and lines 771 - 824 process any errors that may have occurred during the estimation process.

Other Routines

Various subprograms are used by the above-described routines to ac - complish the estimation functions required in the L_p MUSIC process. The purposes of these subprograms are listed below.

BRACKET_DOA: Evaluates DOA peak sharpness measures.

C16_VACHOL: Performs Cholesky decomposition of updated $\mathbf{R}_x$.

C8_NOISE: Generates noise vector samples.

DCV_IRLS: Performs IRLS estimation of a COMPLEX*16 vector.

RESIDUALS: Used by DCV_IRLS to compute estimation residuals.

DC_MPPSINV: Evaluates the Moore-Penrose psuedo-inverse of a COM - PLEX*16 matrix using singular value decomposition.

DCSV_VSIGM1: Used by DC_MPPSINV to evaluate a matrix product.

DUMP_VA: Discards all storage allocated by the Virtual Array system (copyright 1987 by Dr. Dwight Day).

EIG_FROM_SV: Converts results of generalized singular value decom - position to generalized eigenvalue problem solution.

ENDSPAWN_IF: Terminates program execution if all criteria for doing so have been met.

EXIT_VA: Processes status conditions returned by Virtual Array rou - tines (copyright 1987 by Dr. Dwight Day).

FREE_VA: Releases specific portions of memory allocated by the Virtual Array system (copyright 1987 by Dr. Dwight Day).

GET_MUSIC_DEF: Obtains experiment definition values from the user.

GET_W_HALF: Evaluates the square root of a diagonal weight matrix.

MAKE_NOISE: Computes a COMPLEX*16 noise sample vector.

MODULATED: Returns the complex amplitude of the modulation envelope of an arriving wave at a specified sample instant.

MOD_VAL: Used by MODULATED to obtain amplitude.

MOD_PHASE: Used by MODULATED to obtain phase.

MUSIC_INIT: Computes complex amplitudes of arriving waves and initializes noise counters.

MUSIC_L2P: Computes an L_2 estimate of the **P** matrix.

MUSIC_LPP_EACH:
Computes L_p estimates of signal vector samples.

MUSIC_LPP_SPEIS:
Computes L_p estimate of **P** using sum-of-powers method.

MUSIC_LPW_YAR:

Computes square root of diagonal weight matrix for sum-of-powers method.

YAR_ESTS: Used by MUSIC_LPW_YAR to compute signal vector estimates.

YAR_RESIDS: Used by MUSIC_LPW_YAR to compute estimation residuals.

YAR_WEIGHTS: Used by MUSIC_LPW_YAR to compute weights.

MUSIC_PLOT_SPECT:

Plots DOA spectra for various p values.

VALUE_AT: Used by MUSIC_PLOT_SPECT to return a selected vector element.

MUSIC_POWER: Evaluates a DOA spectrum value.

MPWR_FL5, MPWR_DPDTRM, MPWR_FL1, MPWR_DPDSUM, MPWR_PWRTRM, MPWR_FL3, MPWR_PWRSUM:

Used by MUSIC_POWER to evaluate spectrum value and handle floating-point exceptions.

MUSIC_SPECTRUMX:

Evaluates abscissa values for a spectrum plot.

MUSIC_SPECTRUMY:

Evaluates ordinate values for a spectrum plot.

NOISE_COMBINE: Combines noise generator outputs to form antenna noise vector sample.

OBSVEC: Computes observation vector sample.

OPEN_VA: Dynamically allocates storage for FORTRAN programs; part of the Virtual Array system (copyright 1987 by Dr. Dwight Day).

PICK_MUSIC_PEAK: Determines the location of peaks in DOA spectra.

READ_ANTENNAE, READ_LP, READ_MMISC, READ_NOISE, READ_WAVES: Obtain experiment definition values from the user.

PR_NTYPE: Used by READ_NOISE to prompt for noise type.

PR_MTYPE: Used by READ_WAVES to prompt for modulation type.

SIGMAT_ESTM: Form estimate of array phase shift matrix.

SIGMAT_FORM: Form actual phase shift matrix.

SIGVAL: Compute phase shift vector for an incoming wave.

SIGVEC: Compute source signal vector at a selected sample instant.

SPAWN_IF: Initiates a selected batch job if it is not already active.

SPECT_PEAK: Determine the location of a peak in a DOA spectrum.

SQ_VLGSVD: Compute generalized singular value decomposition using Van Loan's method.

VA_DC_QR: Compute QR decomposition of a COMPLEX*16 matrix.

VA_DC_SQSVD: Compute singular value decomposition of a square COM - PLEX*16 matrix.

VA_DC_TRINV: Compute the inverse of an upper triangular COMPLEX*16 matrix.

APPENDIX E

SIMULATION PROGRAM LISTING

This appendix provides computer listings of the routines which comprise the simulation program described in Appendix D. A scaled-down version of the simulation program listed herein is in development for use on personal computers.

```
c
c*******************************************************************************
c
c               This program segment defines various values for use in
c       accessing a MUSIC experiment index file.  A MUSIC experiment index file
c       has the following characteristics (note that the description below can
c       be inserted into an "open" statement if the 1st columns are cleared):
c
c       2       form='UNFORMATTED',recordtype='FIXED',recl=64,
c       3       organization='INDEXED',access='KEYED',
c       4       key=(1:4:INTEGER,5:5:CHARACTER,6:9:INTEGER),
c       5                                               dispose='KEEP',SHARED)
c
c       (The keyword SHARED may be replaced by READONLY).  A read / write from
c       / to this file takes the following form:
c
c               read (...)
c       1               mi_pkey, mi_stat, mi_park, mi_ierr, mi_eerr, mi_serr,
c       2               mi_dtim, mi_ants, mi_waves, mi_noises, mi_smp, mi_sint,
c       3               mi_begp, mi_endp, mi_delp, mi_peps, mi_norm, mi_wlen,
c       4               mi_tol, mi_which, mi_ofile, mi_anoises, mi_irlslim,
c       5               mi_zeigs, mi_doas, mi_estype, mi_parp
c
c               The fields in a MUSIC experiment index file record are defined
c       as follows:
c
c       Byte Positions                  Description
c       --------------  ------------------------------------------------------
c           1 -   4     mi_pkey:  Primary key value, INTEGER*4 (unique record
c                                 number).
c           5 -   5     mi_stat:  Experiment status, CHARACTER*1:
c                                 '-' = new experiment, no files created
c                                       (or input file creation
c                                       failed).
c                                 ' ' = new experiment, input file (but
c                                       no sample files) created.
c                                 'I' = initialized, sample files
c                                       created (ready for eigenvalue
c                                       problem computations).
c                                 'E' = eigenvalue problem solved (ready
c                                       for DOA selection).
c                                 'D' = DOA selections made (ready for Lp
c                                       estimation process) (Note that
c                                       this is the same status for
c                                       "descendant" experiments as ' '
c                                       is for "ancestor" experiments).
c                                 'R' = experiment complete; results
c                                       displayed (at least once).
c                                 'X' = experiment failed; computational
c                                       error encountered.
c           6 -   9     mi_park:  INTEGER*4 "mi_pkey" of the "parent"
c                                 experiment (the experiment which
c                                 produced the residuals which are used
c                                 as observations in this record's
c                                 experiment) (-1 for an "original" or
c                                 "ancestor" experiment (one with no
c                                 "parent")).
c          10 -  11     mi_ierr:  INTEGER*2 number of file read failures
c                                 encountered by the batch sample
c                                 generator job (each failure causes
```

```
c                                   status to change from ' ' to '-').
c          12 -  13     mi_eerr:  INTEGER*2 number of file read failures
c                                   encountered by the batch eigenvalue
c                                   problem solver job (each failure causes
c                                   status to change from 'I' to ' ').
c          14 -  15     mi_serr:  INTEGER*2 number of file read failures
c                                   encountered by the batch Lp estimation
c                                   job (each failure causes status to
c                                   change from 'D' to 'E').
c          16 -  34     mi_dtim:  CHARACTER*19 date / time of experiment.
c          35 -  38     mi_ants:  INTEGER*4 number of antenna elements in the
c                                   experiment.
c          39 -  42     mi_waves:  INTEGER*4 number of arriving waves in the
c                                   experiment.
c          43 -  46     mi_noises:  INTEGER*4 number of noise sources used in
c                                   the ESTIMATION noise model for the
c                                   experiment.
c          47 -  50     mi_smp:  INTEGER*4 number of samples in the experiment.
c          51 -  58     mi_sint:  REAL*8 sample interval for the experiment.
c          59 -  62     mi_begp:  REAL*4 starting value of p for Lp estimation.
c          63 -  66     mi_endp:  REAL*4 ending value of p for Lp estimation
c                                   (< mi_begp for no Lp estimation).
c          67 -  70     mi_delp:  REAL*4 p increment value for Lp estimation.
c          71 -  78     mi_peps:  REAL*8 minimum residual value for Lp
c                                   estimation.
c          79 -  86     mi_norm:  REAL*8 value to which to normalize Lp
c                                   residuals (0 for unnormalized).
c          87 -  94     mi_wlen:  REAL*8 carrier wavelength for the experiment.
c          95 - 102     mi_tol:  REAL*8 value to be taken for zero
c                                   ("tolerance").
c         103 - 106     mi_which:  INTEGER*4 selector which determines whether
c                                   estimators use actual (mi_which = 0) or
c                                   model (mi_which > 0) noise samples.
c         107 - 186     mi_ofile:  CHARACTER*80 name of the experiment's output
c                                   file.
c         187 - 190     mi_anoises:  INTEGER*4 number of noise sources used in
c                                   sample GENERATION.
c         191 - 194     mi_irlslim:  INTEGER*4 IRLS iteration limit.
c         195 - 198     mi_zeigs:  INTEGER*4 number of "noise" eigenvectors
c                                   used (in the "parent" experiment) to
c                                   determine the DOAs which give rise to
c                                   the residuals which serve as
c                                   observations in this record's
c                                   experiment (0 for "ancestor"
c                                   experiment).
c         199 - 202     mi_doas:  INTEGER*4 number of DOAs determined (in the
c                                   "parent" experiment), which are used to
c                                   produce the residuals which serve as
c                                   observations in this record's
c                                   experiment (0 for "ancestor"
c                                   experiment).
c         203 - 206     mi_estype:  INTEGER*4 value specifying the type of
c                                   estimation used to produce the
c                                   residuals which serve as observations
c                                   in this record's experiment (-1 for
c                                   "ancestor" experiment):
c                                           1:  Enumeration Method
c                                           2:  Sum-of-Powers Method
c                                           3:  Compensated Enumeration
c                                           4:  Sum-of-Powers Loss Method
```

```
c                                                5:  Compensated Sum-of-Powers
c                                                6:  Comp'd Sum-of-Powers Loss
c          207 - 210     mi_parp:  REAL*4 value of p used in the "parent"
c                                    experiment to produce the residuals
c                                    which serve as observations for this
c                                    record's experiment (0.00 for
c                                    "ancestor" experiment).
c
        integer*2 mi_ierr, mi_eerr, mi_serr
        integer*4 mi_pkey, mi_park, mi_ants, mi_waves, mi_noises, mi_smp
        integer*4 mi_which, mi_anoises, mi_irlslim, mi_zeigs, mi_doas
        integer*4 mi_estype
        real*4 mi_begp, mi_endp, mi_delp, mi_parp
        real*8 mi_sint, mi_peps, mi_norm, mi_wlen, mi_tol
        character*1 mi_stat
        character*19 mi_dtim
        character*80 mi_ofile
c
c*******************************************************************************
c
```

```
c
c               This program segment contains the declarations used by the
c       MUSIC / Lp main program.
c
        intrinsic date, time
        intrinsic index, char
        external init_va
        external lib$get_lun, lib$stop, lib$delete_file, lib$getjpi
        external lib$find_file, lib$find_file_end
        external cap, mreaddoas
        external open_va, exit_va, dump_va
        external init_music_def, get_music_def, avlbl_prikey
        external writearr, readarr, skiparr
        external read_antennae, read_waves, read_noise, read_mmisc, read_lp
        external dr_colfill
        external lib$wait
        external type_eigs
        external music_plot_inps, music_picture, music_plot_spect
        external music_plot_ests, music_plot_trial, collect_doas
        external pick_music_peak, prange
c
        logical writearr, readarr, skiparr
        logical music_plot_spect, music_plot_ests, music_plot_trial
        logical collect_doas
        integer*4 lib$get_lun, avlbl_prikey, lib$getjpi, type_eigs
        integer*4 lib$find_file, lib$find_file_end
        character*1 cap, pick_music_peak
c
        logical disp_index, havent_irec, pinfo_shown, oinst_shown, dummy
        logical ancestor
        integer*2 tempi
        integer*4 err_return    !  Statement # for return from data I/O error.
        integer*4 old_ants, old_waves, old_noises
        integer*4 old_anoises, old_which, old_smp
        integer*4 vstatus, status, pname_len
        integer*4 data_unit, index_unit, read_unit1, read_unit2, spawn_unit
        integer*4 data_unit2
        integer*4 ants_lin( 1 ), ants_sqr( 2 ), ants_by_2( 2 )
        integer*4 waves_lin( 1 ), waves_by_9( 2 )
        integer*4 waves_by_2( 2 )
        integer*4 nois_by_2( 2 ), nois_by_3( 2 )
        integer*4 ants_by_nois( 2 )
        integer*4 smp_by_ants( 2 ), ants_by_smp( 2 )
        integer*4 wavs_by_smp( 2 ), doas_lin( 1 )
        integer*4 anois_by_2( 2 ), anois_by_3( 2 ), ants_by_anois( 2 )
        integer*4 ant_add
        integer*4 wave_add, mtype_add
        integer*4 ndef_add, npars_add, correl_add
        integer*4 obsmat_add, noisemat_add
        integer*4 sigvm_add
        integer*4 clnmat_add
        integer*4 andef_add, anpars_add, acorrel_add
        integer*4 eig_add, eigv_add
        integer*4 pvm_add
        integer*4 resm_add
        integer*4 yvm_add, yresm_add, doa_add, exnm_add
        integer*4 i, count, fskip, iskip, num_zeigs, num_doas
        integer*4 num_acc, num_exps
        integer*4 vsel, elsel, selexp, selpar, itemp, estype/ 3 /
        real*4 r_answer, p
```

```
      character*1  psel, tch
      character*1  uparrow( 3 )/ 27, '[', 'A' /, spaces( 70 )/ 70*' ' /
      character*2  chsel
      character*5  ptstat( 8 ) / ' IEDR', 'IEDR/', 'IEDR/', 'IEDR/',
     1                           'IEDR/', 'IEDR/', 'ER///', 'ER///' /
      character*10 answer
      character*13 ptypes( 8 ) / 'array picture', 'signal-in-spc',
     1                           'clean observ.', 'noisy observ.',
     2                           'observ. noise', 'all input val',
     3                           'dir. of arriv', 'all trial val' /,
     4               p_etype( 6 ) / '[E]/P/N/L/S/C', 'E/[P]/N/L/S/C',
     5                           'E/P/[N]/L/S/C', 'E/P/N/[L]/S/C',
     6                           'E/P/N/L/[S]/C', 'E/P/N/L/S/[C]' /
      character*15 procname
      character*25 work_file
      character*80 msg, ti_ofile
c
```

```
                                                    18-Apr-1988 14:00:27     VAX
                                                    18-Apr-1988 15:15:49     [CHU
0001            program do_music        ! MAIN program for MUSIC / Lp exercises.
0002    c
0003    c*************************************************************************
0004    c
0005    c               This comment section describes the Virtual Array System vectors
0006    c       and matrices used by the MUSIC / Lp algorithm.  The name of each vector
0007    c       (matrix) address is the "name" of the vector (matrix) with the
0008    c       characters '_add' appended to it.  Thus, the vector ant starts at
0009    c       memory location ant_add, and so on.
0010    c               The Virtual Array System has been copyrighted by Dwight Day in
0011    c       1987.
0012    c
0013    c*************************************************************************
0014    c
0015    c       -----   ant:  ants x 2 REAL*8 matrix whose first column is the vector
0016    c                             of antenna ranges from the origin, and whose
0017    c                             second column is the vector of antenna angles
0018    c                             from the abscissa (RADIANS).
0019    c       -----   wave:  waves x 9 REAL*8 matrix whose columns are as follows:
0020    c                             1) vector of source signal strengths in db.
0021    c                             2) vector of source signal carrier starting
0022    c                                phases (RADIANS).
0023    c                             3) vector of actual angles of arrival of the
0024    c                                incoming waves.
0025    c                             4) vector of incoming wave modulation cycle
0026    c                                lengths, in carrier cycles.
0027    c                             5) vector of incoming wave modulation duty
0028    c                                cycles, as a fraction from 0 to
0029    c                                1.
0030    c                             6) vector of incoming wave modulation
0031    c                                (envelope) start phases
0032    c                                (carrier cycles).
0033    c                             7) vector of incoming wave amplitude modulation
0034    c                                ratios (maximum amplitude /
0035    c                                minimum amplitude).
0036    c                             8) vector of phase modulation total phase
0037    c                                shifts for the incoming waves
0038    c                                (RADIANS).
0039    c                             9) vector of carrier wavelengths of the
0040    c                                incoming waves.
0041    c       -----   ccamp:  waves-element COMPLEX*16 vector of source complex
0042    c                             amplitudes (in "rectangular" form).
0043    c       -----   mtype:  waves x 2 INTEGER*4 matrix whose first column is the
0044    c                             vector of amplitude modulation types for
0045    c                             incoming waves, and whose second column is the
0046    c                             vector of phase modulation types for the
0047    c                             incoming waves.
0048    c       -----   ndef:  noises x 2 INTEGER*4 matrix whose first column is the
0049    c                             vector of noise types, and whose second column
0050    c                             is the vector of noise counters (for
0051    c                             intermittent noise).  This matrix is used for
0052    c                             the ESTIMATION noise model.
0053    c       -----   andef:  anoises x 2 INTEGER*4 matrix whose first column is the
0054    c                             vector of noise types, and whose second column
0055    c                             is the vector of noise counters (for
0056    c                             intermittent noise).  This matrix is used for
0057    c                             ACTUAL SAMPLE GENERATION.
```

```
                                                    18-Apr-1988 14:00:27     \
                                                    13-Apr-1988 15:15:49     [

0058    c       -----   npars:  noises x 3 REAL*4 matrix of noise parameters (mean,
0059    c                       std. dev., ave. rep. rate).  This matrix is
0060    c                       used for the ESTIMATION noise model.
0061    c       -----   anpars:  anoises x 3 REAL*4 matrix of noise parameters (mean,
0062    c                       std. dev., ave. rep. rate).  This matrix is
0063    c                       used for ACTUAL SAMPLE GENERATION.
0064    c       -----   correl:  ants x noises REAL*8 matrix of noise multipliers for
0065    c                       the antenna array.  This matrix is used for the
0066    c                       ESTIMATION noise model.
0067    c       -----   acorrel:  ants x anoises REAL*8 matrix of noise multipliers for
0068    c                       the antenna array.  This matrix is used for
0069    c                       ACTUAL SAMPLE GENERATION.
0070    c       -----   obsmat:  smp x ants COMPLEX*16 matrix whose ROWS are the
0071    c                       samples of the antenna array output (including
0072    c                       noise).
0073    c       -----   noisemat:  smp x ants COMPLEX*16 matrix whose ROWS are the
0074    c                       noise samples used in the ESTIMATION process.
0075    c       -----   rb:  ants x ants noise covariance matrix, normalized such
0076    c                       that its trace is ants.  This matrix is
0077    c                       produced from noisemat; that is, the noise to
0078    c                       be used for ESTIMATION.
0079    c       -----   obs:  ants-element COMPLEX*16 vector sample of antenna array
0080    c                       output (including noise).
0081    c       -----   noise:  ants-element COMPLEX*16 vector noise sample.
0082    c       -----   robs:  ants x ants COMPLEX*16 observation (including noise)
0083    c                       covariance matrix.
0084    c       -----   rnoise:  ants x ants COMPLEX*16 noise covariance matrix (not
0085    c                       trace normalized).  This is the covariance
0086    c                       matrix of the noise produced from the
0087    c                       ESTIMATION noise model.
0088    c       -----   arnoise:  ants x ants COMPLEX*16 noise covariance matrix (not
0089    c                       trace normalized).  This is the covariance
0090    c                       matrix of the noise actually used in GENERATING
0091    c                       the antenna array output samples.
0092    c       -----   ants_di:  ants x ants COMPLEX*16 work matrix.
0093    c       -----   sigvm:  waves x smp COMPLEX*16 matrix whose COLUMNS are the
0094    c                       actual or estimated signal-in-space (source
0095    c                       amplitude & phase) vector samples.
0096    c       -----   clnmat:  ants x smp COMPLEX*16 matrix whose COLUMNS are the
0097    c                       samples of the antenna array output (without
0098    c                       noise).
0099    c       -----   sspc:  waves-element COMPLEX*16 signal-in-space (source
0100    c                       amplitude and phase) vector sample.
0101    c       -----   sgmt:  ants x waves COMPLEX*16 signal coefficient ("A") matrix
0102    c                       (actual).
0103    c       -----   cln:  ants-element COMPLEX*16 antenna array output sample
0104    c                       (without noise).
0105    c       -----   obst:  ants x ants COMPLEX*16 upper-triangular result of QR
0106    c                       decomposition of "obsmat".
0107    c       -----   noiset:  ants x ants COMPLEX*16 upper-triangular result of QR
0108    c                       decomposition of "noisemat".
0109    c       -----   eig:  ants-element REAL*8 vector of generalized eigenvalues.
0110    c       -----   eigv:  ants x ants COMPLEX*16 matrix whose COLUMNS are the
0111    c                       generalized eigenvectors corresponding to "eig"
0112    c                       entries.
0113    c       -----   amat:  ants x waves COMPLEX*16 estimate of the signal
0114    c                       coefficient ("A") matrix.
```

```
                                                                18-Apr-1988 14:00:27
                                                                18-Apr-1988 15:15:49

0115    c       -----   pmat:  waves x waves COMPLEX*16 source covariance matrix
0116    c                          (actual).
0117    c       -----   pl2:  waves x waves COMPLEX*16 least-squares estimate of the
0118    c                          source covariance matrix.
0119    c       -----   plp:  waves x waves COMPLEX*16 "Lp" estimate (using L2 "A") of
0120    c                          the source covariance matrix.
0121    c       -----   yarw:  ants-element REAL*8 vector of final weights produced
0122    c                          by the "Yarlagadda" P-matrix IRLS algorithm.
0123    c
0124    c*************************************************************************
0125    c
0126            implicit none
0127            external p_index, submit_job, print_expm
0128    c
0129            integer*4 p_index
0130    c
0131            include '6000$RT:[CHUCK.RESEARCH.FORTDIR]VATYPES.TXT'
0148            include '6000$RT:[CHUCK.RESEARCH.FORTDIR]VL_MUSIC_DECL.FOR'
0221            include '6000$RT:[CHUCK.RESEARCH.FORTDIR]MUSIC_MEASURES.FOR'
0310            include '6000$RT:[CHUCK.RESEARCH.FORTDIR]MUSIC_INDEX_DEF.TXT'
0451            include '6000$RT:[CHUCK.RESEARCH.FORTDIR]SPAWNED_DEF.TXT'
0494            include 'SYS$LIBRARY:FORIOSDEF'
0580            include '(system_symbols)'
0698            include '($SYSSRVNAM)'
3118    c
3119    c               Get current process name, and change process name to
3120    c       "Chuck MUSIC Drv".
3121    c
3122            status = lib$getjpi( JPI$_PRCNAM, , , , procname, pname_len )
3123            if (.not. status) call lib$stop( %val( status ) )
3124            status = sys$setprn( 'Chuck MUSIC Drv' )
3125            if (.not. status) call lib$stop( %val( status ) )
3126    c
3127    c               Get logical unit numbers and indicate that the program "just
3128    c       woke up".
3129    c
3130            status = lib$get_lun( data_unit )
3131            if (.not. status) call lib$stop( %val( status ) )
3132            status = lib$get_lun( data_unit2 )
3133            if (.not. status) call lib$stop( %val( status ) )
3134            status = lib$get_lun( index_unit )
3135            if (.not. status) call lib$stop( %val( status ) )
3136            status = lib$get_lun( read_unit1 )
3137            if (.not. status) call lib$stop( %val( status ) )
3138            status = lib$get_lun( read_unit2 )
3139            if (.not. status) call lib$stop( %val( status ) )
3140            status = lib$get_lun( spawn_unit )
3141            if (.not. status) call lib$stop( %val( status ) )
3142    c
3143            pinfo_shown = .false.
3144            oinst_shown = .false.
3145            disp_index  = .true.
3146    c
3147    c               Open the MUSIC index file.
3148    c
3149            open (unit=index_unit,file='6000$RT:[CHUCK.PARAMS]MUSIC_INDEX.DAT',
3150            1       form='UNFORMATTED',recordtype='FIXED',recl=64,
```

```
DO_MUSIC                                                    18-Apr-1988 14:00:27     VAX F
                                                            18-Apr-1988 15:15:49     [CHU

3151            2       organization='INDEXED',access='KEYED',status='OLD',
3152            3       key=(1:4:INTEGER,5:5:CHARACTER,6:9:INTEGER),
3153            4                                       dispose='KEEP',SHARED,err=30000)
3154            go to 30001
3155    c
3156    c               Error opening the MUSIC index file; create a new one.
3157    c
3158    30000   continue
3159            open (unit=index_unit,file='6000$RT:[CHUCK.PARAMS]MUSIC_INDEX.DAT',
3160            1       form='UNFORMATTED',recordtype='FIXED',recl=64,
3161            2       organization='INDEXED',access='KEYED',status='NEW',
3162            3       key=(1:4:INTEGER,5:5:CHARACTER,6:9:INTEGER),
3163            4                                                   dispose='KEEP',SHARED)
3164    c
3165    c               MUSIC index file open, open again on a separate LUNs for
3166    c       display routines.
3167    c
3168    30001   continue
3169            open (unit=read_unit1,file='6000$RT:[CHUCK.PARAMS]MUSIC_INDEX.DAT',
3170            1       form='UNFORMATTED',recordtype='FIXED',recl=64,
3171            2       organization='INDEXED',access='KEYED',status='OLD',
3172            3       key=(1:4:INTEGER,5:5:CHARACTER,6:9:INTEGER),
3173            4                                                   dispose='KEEP',SHARED)
3174            open (unit=read_unit2,file='6000$RT:[CHUCK.PARAMS]MUSIC_INDEX.DAT',
3175            1       form='UNFORMATTED',recordtype='FIXED',recl=64,
3176            2       organization='INDEXED',access='KEYED',status='OLD',
3177            3       key=(1:4:INTEGER,5:5:CHARACTER,6:9:INTEGER),
3178            4                                                   dispose='KEEP',SHARED)
3179    c
3180    c               Open the MUSIC spawned processes index file.
3181    c
3182            open (unit=spawn_unit,file='6000$RT:[CHUCK.PARAMS]SPAWNED.IDX',
3183            1       form='UNFORMATTED',recordtype='FIXED',recl=26,
3184            2       organization='INDEXED',access='KEYED',status='OLD',
3185            3       key=(1:4:INTEGER,5:8:INTEGER),dispose='KEEP',SHARED,err=30100)
3186            go to 20000
3187    c
3188    c               Error opening spawned-processes index file.  Give up.
3189    c
3190    30100     continue
3191              type *
3192              type *, 'Couldn''t open file 6000$RT:[CHUCK.PARAMS]SPAWNED.IDX.'
3193              type *, 'Have you run the right version of MAKE_SPAWN_IDX?'
3194              type *
3195              status = sys$setprn( procname( 1 : pname_len ) )
3196              if (.not. status) call lib$stop( %val( status ) )
3197              stop
3198    c
3199    20000   continue                !  Branch for not first time through program.
3200    c
3201    c               Submit any required batch jobs.
3202    c
3203            if ( p_index( index_unit, read_unit1, read_unit2, ' ', .false. )
3204            1                       .ge. 0 ) call submit_job( spawn_unit, 1 )
3205            if ( p_index( index_unit, read_unit1, read_unit2, 'I', .false. )
3206            1                       .ge. 0 ) call submit_job( spawn_unit, 2 )
3207            if ( p_index( index_unit, read_unit1, read_unit2, 'D', .false. )
```

```
DO_MUSIC                                                    18-Apr-1988 14:00:27     VA
                                                            13-Apr-1988 15:15:49     DC

3208            1                           .ge. 0 ) call submit_job( spawn_unit, 3 )
3209     c
3210     c                Display the MUSIC index file if selected.
3211     c
3212            if ( disp_index ) then
3213              disp_index = .false.
3214              mi_pkey = p_index( index_unit, read_unit1, read_unit2, '*', .true. )
3215              if ( mi_pkey .eq. -1 ) then
3216                type *
3217                type *, 'No active MUSIC experiments.'
3218                type *
3219              end if
3220            end if
3221     c
3222     c                Allow user to choose the activity he wishes to perform.
3223     c
3224            type *
3225            type *, 'Please tell me what you wish to do:'
3226            type *, '   N (or n) to initiate a new experiment.'
3227            if ( p_index( index_unit, read_unit1, read_unit2, '-X', .false. )
3228            1       .ge. 0 ) type *, '   I (or i) to re-start aborted or failed ',
3229            2                                                   'experiment.'
3230            if ( p_index( index_unit, read_unit1, read_unit2, 'ER', .false. )
3231            1                                                   .ge. 0 ) then
3232              type *, '   D (or d) to determine DOAs for active experiment.'
3233              type *, '   P (or p) to print output file for completed experiment.'
3234            end if
3235            if ( p_index( index_unit, read_unit1, read_unit2, 'IEDR', .false. )
3236            1       .ge. 0 ) type *, '   C (or c) to evaluate performance for ',
3237            2                                                 'completed experiment.'
3238            if ( p_index( index_unit, read_unit1, read_unit2, ' IEDR', .false. )
3239            1       .ge. 0 ) type *, '   L (or l) to display data for active ',
3240            2                                                   'experiment.'
3241            if ( p_index( index_unit, read_unit1, read_unit2, '-ER', .false. )
3242            1       .ge. 0 ) type *, '   M (or m) to mark active experiment for ',
3243            2                                                   'deletion.'
3244            if ( p_index( index_unit, read_unit1, read_unit2, '-RX', .false. )
3245            1       .ge. 0 ) type *, '   K (or k) to delete completed or failed ',
3246            2                                                   'experiment.'
3247            if ( p_index( index_unit, read_unit1, read_unit2, ' IER', .false. )
3248            1       .ge. 0 ) type *, '   R (or r) to re-run an initialized ',
3249            2                                       'experiment (new "trial").'
3250            type *, '   S (or s) to display current status of all experiments.'
3251            type *, '   X (or x) to exit to the operating system.'
3252     c
3253            type *
3254            type 90000,' What does the master command?  '
3255            accept 90010, answer
3256            psel = cap( answer( 1 : 1 ) )
3257     c                                                                              !
3258     c!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
3259     c                                                                              !
3260     c                Status of active experiments selected.                        !
3261     c                                                                              !
3262            if      ( psel .eq. 'S' ) then
3263              disp_index = .true.
3264              go to 20000
```

```
c                                                                       !
c!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
c                                                                       !
c               Determine DOAs of active experiments selected.          !
c                                                                       !
          else if ( psel .eq. 'D' ) then
            type *
            type *, '                             DOA SPECTRUM PEAK SELECTIONS'
            type *
            if ( .not. pinfo_shown ) then
60420         continue                                    !  Print plot info.
              pinfo_shown = .true.
              type *,'During plotting, the computer types an appropriate ',
     1                                            '"ready to plot" message,'
              type *,'then waits for a <CR> before proceeding with the plot.  ',
     1                                            'When a plot is'
              type *,'finished, the computer waits for another <CR> before ',
     1                                            'continuing with the next'
              type *,'program step.'
              type *
            end if
            type *, 'Enter experiment # for which to determine DOAs,'
            type *, '   <CR> to see a list of available experiments,'
            type 90000, '                 ? for information about plotting:  '
            accept 90010, answer
            if ( index( answer, '?' ) .gt. 0 ) go to 60420
            read (unit=answer,fmt=*,err=60430) mi_pkey
            if ( mi_pkey .ge. 0 ) go to 60410
c
60430       continue                                      !  Error in accepted #.
              mi_pkey
     1          = p_index( index_unit, read_unit1, read_unit2, 'ER', .true. )
              if ( mi_pkey .eq. -1 ) then                 !  None available.
60000           continue
                type *
                type *, char( 7 ), 'Can''t plot DOA spectrum.'
                type *
                go to 20000
              end if
              if ( mi_pkey .lt.  0 ) go to 20000          !  Operator cancelled.
c
60410       continue
              read (unit=index_unit,keyeq=mi_pkey,keyid=0,iostat=status)
     1                  mi_pkey, mi_stat, mi_park, mi_ierr, mi_eerr, mi_serr,
     2                  mi_dtim, mi_ants, mi_waves, mi_noises, mi_smp, mi_sint,
     3                  mi_begp, mi_endp, mi_delp, mi_peps, mi_norm, mi_wlen,
     4                  mi_tol, mi_which, mi_ofile, mi_anoises, mi_irlslim,
     5                  mi_zeigs, mi_doas, mi_estype, mi_parp
              unlock (unit=index_unit)
              if      ( status .eq. FOR$IOS_SPERECLOC ) then
                type *, 'Record #', mi_pkey, ' locked.  Waiting 1 sec.', uparrow
                call lib$wait( 1.0 )                      !  Wait 1 second.
                type *, spaces, uparrow
                go to 60410
              else if ( ( status .ne. 0 )
     1                  .or. ( index( 'ER', mi_stat ) .le. 0 ) ) then
                go to 60000
```

```
DO_MUSIC                                                    18-Apr-1988 14:00:27     VAX
                                                            18-Apr-1988 15:15:49     [CH

3322                end if
3323     c
3324     c                  Experiment for which to plot the DOA spectrum has primary key
3325     c          value mi_pkey, and it is a valid DOA spectrum candidate.  Determine the
3326     c          location of the antenna array description, according to whether it is
3327     c          a "descendant" or an "ancestor" experiment.
3328     c
3329                if ( mi_park .eq. -1 ) then                    !  "Ancestor".
3330                  selexp = mi_pkey
3331                else                                           !  "Descendant".
3332                  selexp = mi_park
3333     60440        continue
3334                  read (unit=index_unit,keyeq=selexp,keyid=0,iostat=status)
3335               1                                               selexp, tch, selpar
3336                  unlock (unit=index_unit)
3337                  if      ( status .eq. FOR$IOS_SPERECLOC ) then
3338                    type *, 'Record #', selexp, ' locked.  Waiting 1 sec.', uparrow
3339                    call lib$wait( 1.0 )                       !  Wait 1 second.
3340                    type *, spaces, uparrow
3341                    go to 60440
3342                  else if ( status .ne. 0 ) then               !  Error reading index.
3343                    mi_pkey = selexp
3344                    go to 60700
3345                  end if
3346                  if ( selpar .ne. -1 ) then
3347                    selexp = selpar
3348                    go to 60440
3349                  end if
3350                end if
3351     c
3352     c                  Now create the required Virtual Arrays.
3353     c
3354                call init_va
3355                ants_by_2(1) = mi_ants
3356                ants_by_2(2) = 2
3357                ants_lin(1)  = mi_ants
3358                ants_sqr(1)  = mi_ants
3359                ants_sqr(2)  = mi_ants
3360     c
3361                call open_va( ant_add, ants_by_2, 2, dr_type, vstatus, msg )
3362                call exit_va( vstatus, msg )
3363                call open_va( eig_add, ants_lin, 1, dr_type, vstatus, msg )
3364                call exit_va( vstatus, msg )
3365                call open_va( eigv_add, ants_sqr, 2, dc_type, vstatus, msg )
3366                call exit_va( vstatus, msg )
3367     c
3368     c                  Retrieve the antenna array description matrix and the
3369     c          eigenvalue problem solutions from the appropriate intermediate files.
3370     c
3371                assign 20000 to err_return
3372     c
3373                work_file = 'MUSIC_INPUT__________.DAT'
3374                write (unit=work_file( 12 : 21 ),fmt=90020,iostat=status,err=60510)
3375               1                                                                  selexp
3376     c
3377                open (unit=data_unit,file='6000$RT:[CHUCK.PARAMS]'//work_file,
3378               1        status='OLD',organization='SEQUENTIAL',access='SEQUENTIAL',
```

```
DO_MUSIC                                                    18-Apr-1988 14:00:27     V
                                                            18-Apr-1988 15:15:49     [

3379            2       form='UNFORMATTED',recordtype='FIXED',recl=64,dispose='KEEP',
3380            3                                   READONLY,iostat=status,err=60560)
3381              read (unit=data_unit,iostat=status,err=60500)
3382              if ( .not. (
3383            1       readarr( %val( ant_add ), dr_type, mi_ants, 2, data_unit, 32 )
3384            2                                                           ) ) go to 60500
3385              close (unit=data_unit)
3386    c
3387              work_file( 7 : 11 ) = 'EIGVS'
3388              write (unit=work_file( 12 : 21 ),fmt=90020,iostat=status,err=60510)
3389            1                                                               mi_pkey
3390              open (unit=data_unit,file='6000$RT:[CHUCK.PARAMS]'//work_file,
3391            1       status='OLD',organization='SEQUENTIAL',access='SEQUENTIAL',
3392            2       form='UNFORMATTED',recordtype='FIXED',recl=64,dispose='KEEP',
3393            3                                   READONLY,iostat=status,err=60560)
3394              do itemp = 1, 5
3395                if ( .not. ( skiparr( mi_ants, mi_ants, data_unit, 16 ) ) )
3396            1                                                       go to 60500
3397              end do
3398              if ( ( mi_park .eq. -1 ) .and. ( mi_which .gt. 0 ) ) then
3399                if ( .not. ( skiparr( mi_ants, mi_ants, data_unit, 16 ) ) )
3400            1                                                       go to 60500
3401              end if
3402              if ( .not. (
3403            1       readarr( %val( eig_add ), dr_type, mi_ants, 1, data_unit, 32 )
3404            2                                                           ) ) go to 60500
3405              if ( .not. (
3406            1       readarr( %val( eigv_add ), dc_type, mi_ants, mi_ants,
3407            2                                       data_unit, 16 ) ) ) go to 60500
3408              close (unit=data_unit)
3409    c
3410    c               Now open scratch file for DOA storage, plot DOA spectra and
3411    c       allow peak selection.
3412    c
3413              open (unit=data_unit,file='6000$RT:[CHUCK.PARAMS]SCRATCH.DAT',
3414            1               form='UNFORMATTED',recordtype='FIXED',recl=2,
3415            2               organization='RELATIVE',access='DIRECT',
3416            3               status='NEW',dispose='DELETE',iostat=status,err=60830)
3417    c
3418              psel = 'C'
3419              do while ( psel .eq. 'C' )    !  Until operator tires of DOA finding.
3420    c
3421    c               Accept number of "noise" eigenvectors to use.
3422    c
3423                num_zeigs = type_eigs( %val( eig_add ), mi_ants, mi_tol )
3424                if ( num_zeigs .lt. 0 ) then                !  For 0 eigenvalue.
3425                  close (unit=data_unit)
3426                  call dump_va( vstatus, msg )
3427                  call exit_va( vstatus, msg )
3428                  mi_stat = 'X'
3429                  go to 60600
3430                end if
3431    c
3432    c               Get range of p values to cover.
3433    c
3434                call prange( mi_begp, mi_endp, mi_delp, fskip, count, iskip )
3435                num_doas = 0
```

```
DO_MUSIC                                                    18-Apr-1988 14:00:27    VAX
                                                            18-Apr-1988 15:15:49    [CH

3436                write (unit=data_unit,rec=1,iostat=status,err=60820)
3437              1                                 num_doas  !  Initialize # DOAs found.
3438        c
3439        60450       continue                                !  Branch for replot selected.
3440        c
3441        c               Plot spectra and pick some peaks.
3442        c
3443                  if ( music_plot_spect( %val( ant_add ), mi_ants, mi_wlen,
3444              1                                 %val( eigv_add ), num_zeigs,
3445              2                                 mi_begp, mi_delp, fskip, count, iskip )
3446              3               ) go to 60800
3447                  psel = pick_music_peak( %val( ant_add ), mi_ants, mi_wlen,
3448              1                                 %val( eigv_add ), num_zeigs,
3449              2                                 data_unit, mi_begp, mi_delp,
3450              3                                                    fskip, count, iskip )
3451                  if ( psel .eq. 'X' ) go to 60800
3452                if ( psel .eq. 'R' ) go to 60450    !  Replot selected.
3453        c
3454        c               Peaks selected, initiate new experiments using peaks as DOAs if
3455        c       that is the operator's whim.
3456        c
3457        60480       continue
3458                    read (unit=data_unit,rec=1,iostat=status,err=60820) num_doas
3459                    if ( num_doas .gt. 0 ) then
3460                      type 90050, 'Do you wish to start', count,
3461              1                          ' new experiments with', num_doas,
3462              2                          ' DOAs each ([y]/n)?  '
3463                      accept 90010, tch
3464                      if ( cap( tch ) .ne. 'N' ) then
3465        c
3466        c               Select type of estimation for the new experiments.
3467        c
3468        60460           continue
3469                          type *, 'Use Enumeration, Sum-of-Powers, Compensated ',
3470              1                                                    'Enumeration,'
3471                          type *, 'Sum-of-Powers Loss, Compensated Sum-of-Powers,'
3472                          type 90000, ' or Compensated Sum-of-Powers Loss estimation ('
3473              1                                     // p_etype(estype) // ')?  '
3474                          read (unit=*,fmt=90010,err=60470) tch
3475                          itemp = index( 'EPNLSC', cap( tch ) )
3476                          if ( itemp .gt. 0 ) estype = itemp
3477        60470             continue
3478                        if ( estype .le. 0 ) go to 60460
3479        c
3480        c               Initiated the selected new experiments.
3481        c
3482                        doas_lin(1) = num_doas
3483                        call open_va( doa_add, doas_lin, 1, dr_type, vstatus, msg )
3484                        call exit_va( vstatus, msg )
3485        c
3486                        tch = 'D'                           !  Indicate estimates needed.
3487                        tempi = 0                           !  No file errors yet.
3488                        elsel = 2                           !  Start with 1st DOA.
3489                        ti_ofile = '6000$RT:[CHUCK.PARAMS]MUSIC_OUTPUT__________.DAT'
3490                        work_file = 'MUSIC_SAMPL__________.DAT'
3491                        do i = 1, count
3492                          selexp = avlbl_prikey( index_unit, 0 )
```

```
DO_MUSIC                                                    18-Apr-1988 14:00:27     VAX FC
                                                            18-Apr-1988 15:15:49     [CHUCK
3493                      if ( selexp .lt. 0 ) then
3494                        close (unit=data_unit)
3495                        go to 50000
3496                      end if
3497                      write (unit=ti_ofile( 35 : 44 ),fmt=90020,iostat=status,
3498           1                                                    err=60820) selexp
3499                      write (unit=work_file( 12 : 21 ),fmt=90020,iostat=status,
3500           1                                                    err=60820) selexp
3501     c
3502     c                Transfer the pertinent DOAs to the appropriate samples file.
3503     c
3504                        open (unit=data_unit2,
3505           1                  file='6000$RT:[CHUCK.PARAMS]'//work_file,status='NEW',
3506           2                  organization='SEQUENTIAL',access='SEQUENTIAL',
3507           3                  form='UNFORMATTED',recordtype='FIXED',recl=64,
3508           4                                  dispose='KEEP',iostat=status,err=60820)
3509                        call mreaddoas( data_unit, elsel, %val( doa_add ), num_doas )
3510                        if ( .not. (
3511           1                  writearr( %val( doa_add ), dr_type, num_doas, 1,
3512           2                                  data_unit2, 32 ) ) ) go to 60810
3513                        close (unit=data_unit2)
3514     c
3515     c                Record the existence of the new experiment.
3516     c
3517                        write (unit=index_unit,err=60820)
3518           1                  selexp, tch, mi_pkey, tempi, tempi, tempi, mi_dtim,
3519           2                  mi_ants, mi_waves, mi_noises, mi_smp, mi_sint, mi_begp,
3520           3                  mi_endp, mi_delp, mi_peps, mi_norm, mi_wlen, mi_tol,
3521           4                  mi_which, ti_ofile, mi_anoises, mi_irlslim, num_zeigs,
3522           5                  num_doas, estype, mi_begp +
3523           6                           ( fskip + ( i - 1 ) * ( iskip + 1 ) ) * mi_delp
3524                        unlock (unit=index_unit)
3525                      end do
3526                      call free_va( doa_add, vstatus, msg )
3527                      call exit_va( vstatus, msg )
3528                    end if
3529     c
3530     c                  All selected new experiments started.  Ask for further
3531     c       instructions.
3532     c
3533                    type *, 'Do you wish to start additional new experiments ',
3534           1                                                    'with current DOAs,'
3535                    type 90000, ' determine more DOAs, or stop DOA processing '
3536           1                                    // 'for this experiment (N/D/[X])?  '
3537                    read (unit=*,fmt=90010,err=60490) tch
3538                    tch = cap( tch )
3539                    if      ( tch .eq. 'D' ) then
3540                      psel = 'C'
3541                    else if ( tch .eq. 'N' ) then
3542                      go to 60480
3543                    else
3544                      psel = 'X'
3545                    end if
3546                  end if
3547                end do
3548     60490      continue                !  Finished DOA processing for this experiment.
3549     c
```

```
DO_MUSIC                                                    18-Apr-1988 14:00:27
                                                            13-Apr-1988 15:15:49

3550     c               Discard space used for Virtual Arrays, and update the
3551     c        experiment's index record if desired by the operator.
3552     c
3553               close (unit=data_unit)
3554               call dump_va( vstatus, msg )
3555               call exit_va( vstatus, msg )
3556     c
3557               type 90000, ' Do you wish to mark this experiment finished '
3558              1                                                    // '(Y/[N])?  '
3559               read (unit=*,fmt=90010,err=60495) tch
3560               if ( cap( tch ) .eq. 'Y' ) then
3561                 mi_stat = 'R'
3562                 go to 60600
3563               end if
3564     60495     continue
3565               go to 20000
3566     c
3567     c               Branch for operator cancel of DOA processing.
3568     c
3569     60800     continue
3570               close (unit=data_unit)
3571               call dump_va( vstatus, msg )
3572               call exit_va( vstatus, msg )
3573               go to 20000
3574     c
3575     c               Branch for file I/O error during DOA determination.
3576     c
3577     60810     continue
3578               close (unit=data_unit2)
3579     c
3580     60820     continue
3581               close (unit=data_unit)
3582     c
3583     60830     continue
3584               call dump_va( vstatus, msg )
3585               call exit_va( vstatus, msg )
3586     c
3587     60840     continue
3588               type *
3589               type *, char( 7 ), 'Error accessing temporary data file(s).'
3590               type 90030, char( 7 ) // 'Experiment #', mi_pkey, '.'
3591               type 90040, 'FORTRAN error code #', status, ' (hex).'
3592               type *, char( 7 ), 'File:  ', work_file
3593               type *
3594               go to 20000
3595     c                                                                              !
3596     c!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
3597     c                                                                              !
3598     c               Print output file selected.                                    !
3599     c                                                                              !
3600              else if ( psel .eq. 'P' ) then
3601                assign 20000 to err_return
3602     c
3603                type *, 'Enter number of the experiment for which to print ',
3604              1                                                  'the output file,'
3605                type 90000, '                          <CR> to see a list of '
3606              1                                      // 'printable experiments:  '
```

```
DO_MUSIC                                                    18-Apr-1988 14:00:27
                                                            13-Apr-1988 15:15:49

3607              accept 90010, answer
3608              read (unit=answer,fmt=*,err=62430) mi_pkey
3609              if ( mi_pkey .lt. 0 ) then
3610     62430      continue                                    !  Error in accepted #.
3611                mi_pkey
3612             1      = p_index( index_unit, read_unit1, read_unit2, 'ER', .true. )
3613                if ( mi_pkey .eq. -1 ) then          !  No printable experiments.
3614     62000        continue
3615                  type *
3616                  type *, char( 7 ), 'Can''t print.'
3617                  type *
3618                  go to 20000
3619                end if
3620                if ( mi_pkey .lt.  0 ) go to 20000          !  Operator cancelled.
3621              end if
3622     c
3623              call print_expm( mi_pkey )
3624              go to 20000
3625     c                                                                          !
3626     c!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
3627     c                                                                          !
3628     c                 Compute performance measures selected.                   !
3629     c                                                                          !
3630            else if ( psel .eq. 'C' ) then
3631              assign 20000 to err_return
3632     c
3633              type *, 'Enter number of the experiment for which to evaluate ',
3634             1                                                      'performance,'
3635              type 90000, '                        <CR> to see a list of applicable '
3636             1                                                   // 'experiments:  '
3637              accept 90010, answer
3638              read (unit=answer,fmt=*,err=63430) mi_pkey
3639              if ( mi_pkey .lt. 0 ) then
3640     63430      continue                                    !  Error in accepted #.
3641                mi_pkey
3642             1      = p_index( index_unit, read_unit1, read_unit2, 'IEDR', .true. )
3643                if ( mi_pkey .eq. -1 ) then          !  No valid experiments.
3644     63000        continue
3645                  type *
3646                  type *, char( 7 ), 'Can''t compute performance measures.'
3647                  type *
3648                  go to 20000
3649                end if
3650                if ( mi_pkey .lt.  0 ) go to 20000          !  Operator cancelled.
3651              end if
3652     c
3653     63410    continue
3654                read (unit=index_unit,keyeq=mi_pkey,keyid=0,iostat=status)
3655             1               mi_pkey, mi_stat, mi_park, mi_ierr, mi_eerr, mi_serr,
3656             2               mi_dtim, mi_ants, mi_waves, mi_noises, mi_smp, mi_sint,
3657             3               mi_begp, mi_endp, mi_delp, mi_peps, mi_norm, mi_wlen,
3658             4               mi_tol, mi_which, mi_ofile, mi_anoises, mi_irlslim,
3659             5               mi_zeigs, mi_doas, mi_estype, mi_parp
3660                unlock (unit=index_unit)
3661                if      ( status .eq. FOR$IOS_SPERECLOC ) then
3662                  type *, 'Record #', mi_pkey, ' locked.  Waiting 1 sec.', uparrow
3663                  call lib$wait( 1.0 )                      !  Wait 1 second.
```

```
DO_MUSIC                                                    18-Apr-1988 14:00:27     VAX
                                                            13-Apr-1988 15:15:49     [CHU
3664                  type *, spaces, uparrow
3665                  go to 63410
3666                else if ( ( status .ne. 0 )
3667               1                .or. ( mi_park .eq. -1 )
3668               2                .or. ( index( 'IEDR', mi_stat ) .le. 0 ) ) then
3669                  go to 63000
3670                end if
3671        c
3672        c                 Experiment for which to evaluation performance has primary key
3673        c           value mi_pkey, and it is a valid candidate.  Create required virtual
3674        c           arrays.
3675        c
3676                  call init_va
3677                  waves_lin(1) = mi_waves
3678        c
3679                  call open_va( wave_add, waves_lin, 1, dr_type, vstatus, msg )
3680                  call exit_va( vstatus, msg )
3681                  call open_va( exnm_add, waves_lin, 1, di_type, vstatus, msg )
3682                  call exit_va( vstatus, msg )
3683        c
3684        c                  Initialize counters, open work file, and collect DOA estimates.
3685        c
3686                  p         = mi_parp                     !  Value for P matrix estimate.
3687                  num_acc   = 0                           !  No collected estimates.
3688                  num_exps  = 0                           !  No experiments used yet.
3689                  selexp    = mi_pkey                     !  For first sample file sel.
3690                  itemp     = 0                           !  For find file call.
3691                  work_file = 'MUSIC_STDOA__________.DAT'
3692                  write (unit=work_file( 12 : 21 ),fmt=90020,iostat=status,err=63830)
3693               1                                                              selexp
3694                  if ( lib$find_file( '6000$RT:[CHUCK.PARAMS]'//work_file,
3695               1                      msg, itemp ) ) then                  !  File exists.
3696                    if ( .not. lib$find_file_end( itemp ) ) then             !  Error.
3697        63440         continue
3698                      call dump_va( vstatus, msg )
3699                      call exit_va( vstatus, msg )
3700                      go to 63000
3701                    end if
3702                    type *, char( 7 )
3703                    type *, 'Performance evaluation control file exists for this ',
3704               1                                                    'experiment.'
3705                    type *, 'This may indicate that performance evaluation is in ',
3706               1                                                    'progress.'
3707                    type 90000, ' Do you wish to delete the file (Y/[N])?  '
3708                    accept 90010, tch
3709                    if ( cap( tch ) .ne. 'Y' ) go to 63440
3710                    type *, char( 7 )
3711                    type 90000, ' Are you sure (Y/[N])?  '
3712                    if ( cap( tch ) .ne. 'Y' ) go to 63440
3713                    call lib$delete_file( '6000$RT:[CHUCK.PARAMS]' // work_file
3714               1                                                      // ';*' )
3715                  else
3716                    if ( .not. lib$find_file_end( itemp ) ) go to 63440 !  Error.
3717                  end if
3718                  open (unit=data_unit,file='6000$RT:[CHUCK.PARAMS]'//work_file,
3719               1        status='NEW',organization='SEQUENTIAL',access='SEQUENTIAL',
3720               2        form='UNFORMATTED',recordtype='FIXED',recl=8,dispose='KEEP',
```

```
3721            3                                       iostat=status,err=63830)
3722              work_file( 7 : 11 ) = 'SAMPL'
3723              do while ( mi_park .ge. 0 )           !  Collect DOA estimates.
3724                doas_lin(1) = mi_doas
3725                call open_va( doa_add, doas_lin, 1, dr_type, vstatus, msg )
3726                call exit_va( vstatus, msg )
3727                write (unit=work_file( 12 : 21 ),fmt=90020,iostat=status,err=63820)
3728            1                                                       selexp
3729    c
3730                open (unit=data_unit2,file='6000$RT:[CHUCK.PARAMS]'//work_file,
3731            1       status='OLD',organization='SEQUENTIAL',access='SEQUENTIAL',
3732            2       form='UNFORMATTED',recordtype='FIXED',recl=64,dispose='KEEP',
3733            3                                   READONLY,iostat=status,err=60550)
3734                if ( .not. ( readarr( %val( doa_add ), dr_type, mi_doas, 1,
3735            1                                   data_unit2, 82 ) ) ) go to 63810
3736                close (unit=data_unit2)
3737    c
3738                if ( .not. collect_doas( mi_waves, num_acc, num_exps,
3739            1                                   %val( wave_add ), %val( exnm_add ),
3740            2                                   data_unit, mi_doas, %val( doa_add ),
3741            3                                   selexp, mi_zeigs, mi_estype, mi_parp )
3742            4       ) go to 63820
3743                call free_va( doa_add, vstatus, msg )
3744                call exit_va( vstatus, msg )
3745    c                                           Go up one in ancestor chain.
3746    63420       continue
3747                read (unit=index_unit,keyeq=mi_park,keyid=0,iostat=status)
3748            1               selexp, mi_stat, mi_park, mi_ierr, mi_eerr, mi_serr,
3749            2               mi_dtim, mi_ants, mi_waves, mi_noises, mi_smp, mi_sint,
3750            3               mi_begp, mi_endp, mi_delp, mi_peps, mi_norm, mi_wlen,
3751            4               mi_tol, mi_which, mi_ofile, mi_anoises, mi_irlslim,
3752            5               mi_zeigs, mi_doas, mi_estype, mi_parp
3753                unlock (unit=index_unit)
3754                if      ( status .eq. FOR$IOS_SPERECLOC ) then
3755                  type *, 'Record #', mi_park, ' locked.  Waiting 1 sec.', uparrow
3756                  call lib$wait( 1.0 )                  !  Wait 1 second.
3757                  type *, spaces, uparrow
3758                  go to 63420
3759                else if ( status .ne. 0 ) then
3760                  go to 63820
3761                end if
3762              end do
3763    c
3764    c                DOA estimates for evaluation are all collected.  Identify
3765    c       observation and noise sample matrices, and spawn performance measure
3766    c       computation job.
3767    c
3768                if ( mi_waves .gt. num_acc ) then     !  Not enough estimates.
3769                  close (unit=data_unit)
3770                  call dump_va( vstatus, msg )
3771                  call exit_va( vstatus, msg )
3772                  type *
3773                  type *, 'Not all arriving waves have been identified.'
3774                  go to 63000
3775                end if
3776    c
3777    63450       continue
```

```
3778              type *, 'Use Enumeration, Sum-of-Powers, Compensated Enumeration,'
3779              type *, 'Sum-of-Powers Loss, Compensated Sum-of-Powers,'
3780              type 90000, ' or Compensated Sum-of-Powers Loss estimation ('
3781             1                                          // p_etype(estype) // ')?  '
3782              read (unit=*,fmt=90010,err=63460) tch
3783                itemp = index( 'EPNLSC', cap( tch ) )
3784                if ( itemp .gt. 0 ) estype = itemp
3785      63460   continue
3786              if ( estype .le. 0 ) go to 63450
3787      c
3788              write (unit=data_unit,err=63820)
3789             1                        mi_park, mi_pkey, selexp, num_exps, estype, p
3790              if ( .not. ( writearr( %val( wave_add ), dr_type,
3791             1                        mi_waves, 1, data_unit, 4 ) ) ) go to 63820
3792              if ( .not. ( writearr( %val( exnm_add ), di_type,
3793             1                        mi_waves, 1, data_unit, 8 ) ) ) go to 63820
3794              close (unit=data_unit)
3795      c
3796              call submit_job( spawn_unit, 4 )
3797              call dump_va( vstatus, msg )
3798              call exit_va( vstatus, msg )
3799              go to 20000
3800      c
3801      c               File I/O error - performance evaluation.
3802      c
3803      63810   continue
3804              close (unit=data_unit2)
3805      c
3806      63820   continue
3807              close (unit=data_unit)
3808      c
3809      63830   continue
3810              call dump_va( vstatus, msg )
3811              call exit_va( vstatus, msg )
3812      c
3813      63840   continue
3814              type *, char( 7 )
3815              type *, 'Performance evaluation:  Error accessing data files.'
3816              type *, 'Experiment #', mi_pkey, '.  Performance evaluation ',
3817             1                                                    'abandoned.'
3818              type *, char( 7 )
3819              go to 20000
3820      c                                                                      !
3821      c!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
3822      c                                                                      !
3823      c               Mark active experiment for deletion selected.          !
3824      c                                                                      !
3825             else if ( psel .eq. 'M' ) then
3826              assign 20000 to err_return
3827      c
3828              type *, 'Enter number of the experiment to mark for deletion,'
3829              type 90000, '          <CR> to see a list of deletable experiments:  '
3830              accept 90010, answer
3831              read (unit=answer,fmt=*,err=81430) mi_pkey
3832              if ( mi_pkey .lt. 0 ) then
3833      81430     continue                                          !  Error in accepted #.
3834                mi_pkey
```

```
DO_MUSIC                                                    18-Apr-1988 14:00:27     VAX F
                                                            18-Apr-1988 15:15:49     [CHU
3835            1       = p_index( index_unit, read_unit1, read_unit2, '-ER', .true. )
3836                  if ( mi_pkey .eq. -1 ) then          !  No deletable experiments.
3837    81000           continue
3838                    type *
3839                    type *, char( 7 ), 'Can''t mark for deletion.'
3840                    type *
3841                    go to 20000
3842                  end if
3843                  if ( mi_pkey .lt.  0 ) go to 20000          !  Operator cancelled.
3844                end if
3845    c
3846    81410       continue
3847                  read (unit=index_unit,keyeq=mi_pkey,keyid=0,iostat=status)
3848            1                mi_pkey, mi_stat, mi_park, mi_ierr, mi_eerr, mi_serr,
3849            2                mi_dtim, mi_ants, mi_waves, mi_noises, mi_smp, mi_sint,
3850            3                mi_begp, mi_endp, mi_delp, mi_peps, mi_norm, mi_wlen,
3851            4                mi_tol, mi_which, mi_ofile, mi_anoises, mi_irlslim,
3852            5                mi_zeigs, mi_doas, mi_estype, mi_parp
3853                  unlock (unit=index_unit)
3854                  if      ( status .eq. FOR$IOS_SPERECLOC ) then
3855                    type *, 'Record #', mi_pkey, ' locked.  Waiting 1 sec.', uparrow
3856                    call lib$wait( 1.0 )                      !  Wait 1 second.
3857                    type *, spaces, uparrow
3858                    go to 81410
3859                  else if ( ( status .ne. 0 )
3860            1                    .or. ( index( '-ER', mi_stat ) .le. 0 ) ) then
3861                    go to 81000
3862                  end if
3863    c
3864    c                   Experiment to be marked for deletion has primary key value
3865    c       mi_pkey, and it is a valid candidate.
3866    c
3867                  type *
3868                  type 90030, char( 7 ) // 'Mark experiment #', mi_pkey,
3869            1                                                  ' for DELETION?'
3870                  type 90000, ' (Y/[N])?  '
3871                  accept 90010, answer
3872                  psel = cap( answer( 1 : 1 ) )
3873    c
3874                  if ( psel .eq. 'Y' ) then
3875                    type 90000, ' ' // char( 7 ) // 'Are you sure (Y/[N])?  '
3876                    accept 90010, answer
3877                    psel = cap( answer( 1 : 1 ) )
3878                    if ( psel .ne. 'Y' ) go to 81520
3879    c
3880    c                   Operator has given assurance that he wants the selected
3881    c       experiment MARKED FOR DELETION.
3882    c
3883                    mi_stat = 'X'
3884                    type *
3885                    type 90030, char( 7 ) // 'Marking experiment #', mi_pkey,
3886            1                                                  ' for DELETION!'
3887                    type *
3888                    go to 60600
3889                  else                                  !  Operator changed his mind.
3890    81520           continue
3891                    type *
```

```
DO_MUSIC                                                    18-Apr-1988 14:00:27
                                                            13-Apr-1988 15:15:49

3892                type 90030, 'Experiment #', mi_pkey, ' not altered.'
3893                type *
3894              end if
3895              go to 20000
3896      c                                                                      !
3897      c!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
3898      c                                                                      !
3899      c                 Delete completed or failed experiment selected.      !
3900      c                                                                      !
3901            else if ( psel .eq. 'K' ) then
3902              assign 20000 to err_return
3903      c
3904              type *, 'Enter number of the experiment for which to DELETE files,'
3905              type 90000, '              <CR> to see a list of deletable '
3906             1                                                 // 'experiments:  '
3907              accept 90010, answer
3908              read (unit=answer,fmt=*,err=80430) mi_pkey
3909              if ( mi_pkey .lt. 0 ) then
3910      80430     continue                                        !  Error in accepted #.
3911                mi_pkey
3912             1      = p_index( index_unit, read_unit1, read_unit2, '-RX', .true. )
3913                if ( mi_pkey .eq. -1 ) then          !  No deletable experiments.
3914      80000       continue
3915                  type *
3916                  type *, char( 7 ), 'Can''t delete.'
3917                  type *
3918                  go to 20000
3919                end if
3920                if ( mi_pkey .lt.  0 ) go to 20000           !  Operator cancelled.
3921              end if
3922      c
3923      80410     continue
3924                read (unit=index_unit,keyeq=mi_pkey,keyid=0,iostat=status)
3925             1               mi_pkey, mi_stat, mi_park, mi_ierr, mi_eerr, mi_serr,
3926             2               mi_dtim, mi_ants, mi_waves, mi_noises, mi_smp, mi_sint,
3927             3               mi_begp, mi_endp, mi_delp, mi_peps, mi_norm, mi_wien,
3928             4               mi_tol, mi_which, mi_ofile, mi_anoises, mi_irlslim,
3929             5               mi_zeigs, mi_doas, mi_estype, mi_parp
3930                unlock (unit=index_unit)
3931                if      ( status .eq. FOR$IOS_SPERECLOC ) then
3932                  type *, 'Record #', mi_pkey, ' locked.  Waiting 1 sec.', uparrow
3933                  call lib$wait( 1.0 )                       !  Wait 1 second.
3934                  type *, spaces, uparrow
3935                  go to 80410
3936                else if ( ( status .ne. 0 )
3937             1               .or. ( index( '-RX', mi_stat ) .le. 0 ) ) then
3938                  go to 80000
3939                end if
3940      c
3941      c               Experiment to delete has primary key value mi_pkey, and it is a
3942      c       valid deletion candidate.
3943      c
3944                type *
3945                type *, char( 7 ), 'Delete ALL intermediate files ',
3946             1                                 'AND output file AND index entry ',
3947             2                                 'for experiment'
3948                type 90030, '#', mi_pkey, '?'
```

```
              type 90000, ' (Y/[N])?  '
              accept 90010, answer
              psel = cap( answer( 1 : 1 ) )
c
              if ( psel .eq. 'Y' ) then
                type 90000, ' ' // char( 7 ) // 'Are you sure (Y/[N])?  '
                accept 90010, answer
                psel = cap( answer( 1 : 1 ) )
                if ( psel .ne. 'Y' ) go to 80520
c
80530           continue
                read (unit=index_unit,key=mi_pkey,keyid=0,iostat=status)
                if       ( status .eq. FOR$IOS_SPERECLOC ) then
                  unlock (unit=index_unit)
                  type *, 'Index record locked by another program ...'
                  call lib$wait( 10.0 )                    !  Wait 10 seconds.
                  type *, 'Trying again.'
                  go to 80530
                else if ( status .ne. 0 ) then
                  unlock (unit=index_unit)
                  go to 60700
                end if
c
c                   Operator has given assurance that he wants all files associated
c           with the selected experiment DELETED.
c
                work_file = 'MUSIC_*__________.DAT'
                write (unit=work_file( 8 : 17 ),fmt=90020,err=80540) mi_pkey
                call lib$delete_file( '6000$RT:[CHUCK.PARAMS]' // work_file
     1                                                           // ';*' )
                delete (unit=index_unit,iostat=status,err=60700)
                unlock (unit=index_unit)
c
                type *
                type 90030, char( 7 ) // 'ALL files for experiment #', mi_pkey,
     1                                                          ' DELETED!'
                type *
              else
80520           continue
                type *
                type 90030, 'Experiment #', mi_pkey, ' not altered.'
                type *
              end if
              go to 20000
c
80540         continue
              type *
              type 90030, char( 7 ) // 'Unable to form data file name for '
     1                                      // 'experiment #', mi_pkey, '.'
              go to 80520
c                                                                              !
c!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
c                                                                              !
c                   Results display for active experiment selected.  Plots     !
c           available are:                                                     !
c                   1)      Picture of radar array with arriving waves.        !
c                   2)      Elements of the signal in space, observation, noise,  !
```

```
DO_MUSIC                                                    18-Apr-1988 14:00:27    VAX
                                                            18-Apr-1988 15:15:49    [CHU

4006    c                               Lp signal in space estimate, and residual      !
4007    c                               vectors.                                       !
4008    c               3)      MUSIC direction-of-arrival (DOA) spectra and           !
4009    c                               sharpness measures.                            !
4010    c                                                                              !
4011            else if ( psel .eq. 'L' ) then
4012              assign 61500 to err_return
4013    c
4014              vsel = 1
4015              do while ( vsel .ne. 9 )
4016                type *
4017                if ( oinst_shown ) then
4018                  if      ( p_index( index_unit, read_unit1, read_unit2,
4019             1                                          'ER', .false. ) .ge. 0 ) then
4020                    type *, 'c = P, S, C, O, N, D, I, A, X, or ?.'
4021                  else if ( p_index( index_unit, read_unit1, read_unit2,
4022             1                                          'ID', .false. ) .ge. 0 ) then
4023                    type *, 'c = P, S, C, O, N, I, X, or ?.'
4024                  else
4025                    type *, 'P, X, or ?.'
4026                  end if
4027                else
4028    10420           continue
4029                  oinst_shown = .true.
4030                  if ( p_index( index_unit, read_unit1, read_unit2,
4031             1                                          'IEDR', .false. ) .ge. 0 ) then
4032                    type *, 'Selections are of the form cnnn, where c is:'
4033                    type *, '     P(p):  array picture;'
4034                  else
4035                    type *, 'Enter P or p to plot array picture, ',
4036             1                                          'X or x to stop plotting.'
4037                    go to 61440
4038                  end if
4039                  if ( p_index( index_unit, read_unit1, read_unit2,
4040             1                                          'IEDR', .false. ) .ge. 0 ) then
4041                    type *, '     S(s):  signal-in-space (or estimate) entry;'
4042                    type *, '     C(c):  noise-free observation vector entry ',
4043             1                                              '("ancestors" only);'
4044                    type *, '     O(o):  noisy observation (or residual) entry;'
4045                    type *, '     N(n):  noise vector entry ("ancestors" only);'
4046                    type *, '     I(i):  all "input" vector entries (S, C, O, N);'
4047                  end if
4048                  if ( p_index( index_unit, read_unit1, read_unit2,
4049             1                                          'ER', .false. ) .ge. 0 ) then
4050                    type *, '     D(d):  direction of arrival spectra;'
4051                    type *, '     A(a):  all trial vector entries (O, N, D);'
4052                  end if
4053                  if ( p_index( index_unit, read_unit1, read_unit2,
4054             1                                          'IEDR', .false. ) .ge. 0 ) then
4055                    type *, '     X(x):  end pictures;'
4056                    type *, '     and nnn is an integer (<= 3 digits) choosing ',
4057             1                                          'the vector entry # to plot'
4058                    type *, '          (applies only to selections S, C, O, and N).'
4059                  else
4060                    type *, '     X(x):  end pictures.'
4061                  end if
4062    61440         continue
```

```
4063                  type *
4064                  type *, 'Note:  Entering ? repeats this explanation.'
4065                end if
4066     c
4067                type *
4068                type 90000,' Please select the activity to perform (cnnn):  '
4069                accept 90010, answer
4070                chsel = answer( 2 : 4 )
4071                vsel = index( 'PSCONIDAX', cap( answer( 1 : 1 ) ) )
4072                if ( vsel .le. 0 ) go to 10420
4073                if ( vsel .eq. 9 ) go to 61500
4074                type *, ' For what experiment # shall '
4075              1                                 // ptypes(vsel) // ' be computed?'
4076                type 90000, '            (<CR) for a list of applicable '
4077              1                                                 // 'experiments):  '
4078     c
4079                accept 90010, answer
4080                read (unit=answer,fmt=*,err=61430) mi_pkey
4081                if ( mi_pkey .lt. 0 ) then
4082     61430        continue                                     !  Error in accepted #.
4083                  mi_pkey = p_index( index_unit, read_unit1, read_unit2,
4084              1                                       ptstat(vsel), .true. )
4085                  if ( mi_pkey .eq. -1 ) then                 !  No outputable exp's.
4086     61010          continue
4087                    type *
4088                    type *, char( 7 ), 'Can''t plot.'
4089                    type *
4090                    go to  61500
4091                  end if
4092                  if ( mi_pkey .lt.  0 ) go to 61500          !  Operator cancelled.
4093                end if
4094     c
4095     61410      continue
4096                read (unit=index_unit,keyeq=mi_pkey,keyid=0,iostat=status)
4097              1             mi_pkey, mi_stat, mi_park, mi_ierr, mi_eerr, mi_serr,
4098              2             mi_dtim, mi_ants, mi_waves, mi_noises, mi_smp, mi_sint,
4099              3             mi_begp, mi_endp, mi_delp, mi_peps, mi_norm, mi_wlen,
4100              4             mi_tol, mi_which, mi_ofile, mi_anoises, mi_irlslim,
4101              5             mi_zeigs, mi_doas, mi_estype, mi_parp
4102                unlock (unit=index_unit)
4103                if      ( status .eq. FOR$IOS_SPERECLOC ) then
4104                  type *, 'Record #', mi_pkey, ' locked.  Waiting 1 sec.', uparrow
4105                  call lib$wait( 1.0 )                        !  Wait 1 second.
4106                  type *, spaces, uparrow
4107                  go to 61410
4108                else if ( ( status .ne. 0 )
4109              1             .or. ( index( ptstat(vsel), mi_stat ) .le. 0 ) ) then
4110                  go to 61010
4111                end if
4112     c
4113     c               Experiment for which to display results has primary key value
4114     c       mi_pkey, and it is a valid results display candidate.
4115     c
4116                if ( .not. pinfo_shown ) then
4117     61090        continue
4118                  pinfo_shown = .true.
4119                  type *
```

```
DO_MUSIC                                                    18-Apr-1988 14:00:27     V
                                                            18-Apr-1988 15:15:49     [

4120                  type *, '                                        PICTURES'
4121                  type *
4122                  type *,'During plotting, the computer types an appropriate ',
4123             1                                        '"ready to plot" message,'
4124                  type *,'then waits for a <CR> before proceeding with the ',
4125             1                                        'plot.  When a plot is'
4126                  type *,'finished, the computer waits for another <CR> before ',
4127             1                                        'continuing with the next'
4128                  type *,'program step.'
4129                  type *
4130                end if
4131     c
4132     c               Now plot the selected results.
4133     c
4134     c.....................Operator selected picture of antenna array.............
4135     c
4136                 if      ( vsel .eq. 1 ) then                !  Picture chosen.
4137     c
4138     c               Find antenna array description, which may be part of the
4139     c        selected experiment's "ancestor" experiment.
4140     c
4141                  if ( mi_park .eq. -1 ) then                !  "Ancestor".
4142                    selexp = mi_pkey
4143                  else                                       !  "Descendant".
4144                    selexp = mi_park
4145     61470          continue
4146                    read (unit=index_unit,keyeq=selexp,keyid=0,iostat=status)
4147             1                                              selexp, tch, selpar
4148                    unlock (unit=index_unit)
4149                    if      ( status .eq. FOR$IOS_SPERECLOC ) then
4150                      type *, 'Record #', selexp, ' locked.  Waiting 1 sec.',
4151             1                                                          uparrow
4152                      call lib$wait( 1.0 )                   !  Wait 1 second.
4153                      type *, spaces, uparrow
4154                      go to 61470
4155                    else if ( status .ne. 0 ) then           !  Error reading index.
4156                      mi_pkey = selexp
4157                      go to 60700
4158                    end if
4159                    if ( selpar .ne. -1 ) then
4160                      selexp = selpar
4161                      go to 61470
4162                    end if
4163                  end if
4164     c
4165     c               Now create the required Virtual Arrays.
4166     c
4167                  call init_va
4168                  ants_by_2(1)  = mi_ants
4169                  ants_by_2(2)  = 2
4170                  waves_by_9(1) = mi_waves
4171                  waves_by_9(2) = 9
4172     c
4173                  call open_va( ant_add, ants_by_2, 2, dr_type, vstatus, msg )
4174                  call exit_va( vstatus, msg )
4175                  call open_va( wave_add, waves_by_9, 2, dr_type, vstatus, msg )
4176                  call exit_va( vstatus, msg )
```

```
DO_MUSIC                                                    18-Apr-1988 14:00:27    VAX FC
                                                            18-Apr-1988 15:15:49    [CHUCK

4177     c
4178                   work_file = 'MUSIC_INPUT__________.DAT'
4179                   write (unit=work_file( 12 : 21 ),fmt=90020,iostat=status,
4180            1                                                 err=60510) selexp
4181                   open (unit=data_unit,file='6000$RT:[CHUCK.PARAMS]'//work_file,
4182            1              status='OLD',organization='SEQUENTIAL',
4183            2              access='SEQUENTIAL',form='UNFORMATTED',
4184            3              recordtype='FIXED',recl=64,dispose='KEEP',
4185            4                              READONLY,iostat=status,err=60560)
4186                   read (unit=data_unit,err=60500)
4187                   if ( .not. (
4188            1              readarr( %val( ant_add ), dr_type, mi_ants, 2,
4189            2                                        data_unit, 32 ) ) ) go to 60500
4190                   if ( .not. (
4191            1              readarr( %val( wave_add ), dr_type, mi_waves, 9,
4192            2                                        data_unit, 32 ) ) ) go to 60500
4193                   close (unit=data_unit)
4194     c
4195                   call music_picture( %val( ant_add ), mi_ants,
4196            1                                        %val( wave_add ), mi_waves )
4197     c
4198                   call dump_va( vstatus, msg )
4199                   call exit_va( vstatus, msg )
4200     c
4201     c.....................Operator selected plot(s) of input vector(s)...........
4202     c
4203                 else if ( vsel .le. 6 ) then
4204                   if ( vsel .ne. 6 ) then
4205                     elsel = -1
4206                     read(unit=chsel,fmt=*,err=61200) elsel
4207                   end if
4208     61200         continue
4209     c
4210                   ancestor = ( mi_park .eq. -1 )     !  True for "ancestor" exper.
4211                   if ( ancestor ) then               !  "Ancestor".
4212                     wavs_by_smp(1) = mi_waves
4213                     ants_by_smp(1) = mi_ants
4214                     ants_by_smp(2) = mi_smp
4215                   else                               !  "Descendant".
4216                     if ( ( vsel .eq. 3 ) .or. ( vsel .eq. 5 ) ) go to 61010
4217     c
4218     c              Get number of accumulated DOAs including this experiment.
4219     c
4220                     num_doas = mi_doas
4221                     selexp   = mi_park
4222     62470           continue
4223                     read (unit=index_unit,keyeq=selexp,keyid=0,iostat=status)
4224            1                selexp, tch, selpar, mi_ierr, mi_eerr, mi_serr,
4225            2                mi_dtim, mi_ants, mi_waves, mi_noises, mi_smp, mi_sint,
4226            3                mi_begp, mi_endp, mi_delp, mi_peps, mi_norm, mi_wlen,
4227            4                mi_tol, mi_which, mi_ofile, mi_anoises, mi_irlslim,
4228            5                mi_zeigs, itemp, mi_estype, mi_parp
4229                     unlock (unit=index_unit)
4230                     if       ( status .eq. FOR$IOS_SPERECLOC ) then
4231                       type *, 'Record #', selexp, ' locked.  Waiting 1 sec.',
4232            1                                                          uparrow
4233                         call lib$wait( 1.0 )                 !  Wait 1 second.
```

```
DO_MUSIC                                                    18-Apr-1988 14:00:27     VAX
                                                            13-Apr-1988 15:15:49     [CHU

4234                    type *, spaces, uparrow
4235                    go to 62470
4236                  else if ( status .ne. 0 ) then          !  Error reading index.
4237                    mi_pkey = selexp
4238                    go to 60700
4239                  end if
4240                  if ( selpar .ne. -1 ) then
4241                    num_doas = num_doas + itemp
4242                    selexp = selpar
4243                    go to 62470
4244                  end if
4245    c
4246                  wavs_by_smp(1) = num_doas
4247                end if
4248                wavs_by_smp(2) = mi_smp
4249                smp_by_ants(1) = mi_smp
4250                smp_by_ants(2) = mi_ants
4251    c
4252                call init_va
4253                call open_va( sigvm_add, wavs_by_smp, 2, dc_type, vstatus, msg )
4254                call exit_va( vstatus, msg )
4255                call open_va( obsmat_add, smp_by_ants, 2, dc_type, vstatus, msg )
4256                call exit_va( vstatus, msg )
4257                if ( ancestor ) then                    !  "Ancestor".
4258                  call open_va( clnmat_add, ants_by_smp, 2, dc_type, vstatus,
4259              1                                                              msg )
4260                  call exit_va( vstatus, msg )
4261                  call open_va( noisemat_add, smp_by_ants, 2, dc_type, vstatus,
4262              1                                                              msg )
4263                  call exit_va( vstatus, msg )
4264                end if
4265    c
4266                work_file = 'MUSIC_SAMPL__________.DAT'
4267                write (unit=work_file( 12 : 21 ),fmt=90020,iostat=status,
4268              1                                          err=60510) mi_pkey
4269                open (unit=data_unit,file='6000$RT:[CHUCK.PARAMS]'//work_file,
4270              1           status='OLD',organization='SEQUENTIAL',
4271              2           access='SEQUENTIAL',form='UNFORMATTED',
4272              3           recordtype='FIXED',recl=64,dispose='KEEP',
4273              4                       READONLY,iostat=status,err=60560)
4274                if ( .not. ancestor) then               !  "Descendant".
4275                  if ( .not. ( skiparr( mi_doas, 1,             !  Skip DOAs.
4276              1                                    data_unit, 32 ) ) ) go to 60500
4277                end if
4278                if ( .not. (
4279              1           readarr( %val( obsmat_add ), dc_type, mi_smp, mi_ants,
4280              2                                    data_unit, 16 ) ) ) go to 60500
4281                if ( ancestor ) then                    !  "Ancestor".
4282                  if ( .not. (
4283              1           readarr( %val( noisemat_add ), dc_type, mi_smp,
4284              2                   mi_ants, data_unit, 16 ) ) ) go to 60500
4285                end if
4286                if ( .not. (
4287              1           readarr( %val( sigvm_add ), dc_type, wavs_by_smp(1),
4288              2                               mi_smp, data_unit, 16 ) ) ) go to 60500
4289                if ( ancestor ) then                    !  "Ancestor".
4290                  if ( .not. (
```

```
DO_MUSIC                                                    18-Apr-1988 14:00:27    VAX F
                                                            13-Apr-1988 15:15:49    [CHUC

4291            1               readarr( %val( clnmat_add ), dc_type, mi_ants, mi_smp,
4292            2                                       data_unit, 16 ) ) ) go to 60500
4293                  end if
4294                  close (unit=data_unit)
4295    c
4296                  call music_plot_inps( vsel, elsel,
4297            1                           %val( sigvm_add ), %val( clnmat_add ),
4298            2                           %val( obsmat_add ), %val( noisemat_add ),
4299            3                           mi_which, mi_ants, wavs_by_smp(1), mi_smp,
4300            4                                                   ancestor )
4301    c
4302                  call dump_va( vstatus, msg )
4303                  call exit_va( vstatus, msg )
4304    c
4305    c.....Operator selected DOA spectrum plot(s).....
4306    c
4307                else if ( vsel .eq. 7 ) then
4308    c
4309    c               Find antenna array description, which may be part of the
4310    c       selected experiment's "ancestor" experiment.
4311    c
4312                  if ( mi_park .eq. -1 ) then               !  "Ancestor".
4313                    selexp = mi_pkey
4314                  else                                      !  "Descendant".
4315                    selexp = mi_park
4316    61450           continue
4317                    read (unit=index_unit,keyeq=selexp,keyid=0,iostat=status)
4318            1                                               selexp, tch, selpar
4319                    unlock (unit=index_unit)
4320                    if      ( status .eq. FOR$IOS_SPERECLOC ) then
4321                      type *, 'Record #', selexp, ' locked.  Waiting 1 sec.',
4322            1                                                       uparrow
4323                      call lib$wait( 1.0 )                  !  Wait 1 second.
4324                      type *, spaces, uparrow
4325                      go to 61450
4326                    else if ( status .ne. 0 ) then          !  Error reading index.
4327                      mi_pkey = selexp
4328                      go to 60700
4329                    end if
4330                    if ( selpar .ne. -1 ) then
4331                      selexp = selpar
4332                      go to 61450
4333                    end if
4334                  end if
4335    c
4336    c               Now open required Virtual Arrays.
4337    c
4338                  call init_va
4339                  ants_by_2(1) = mi_ants
4340                  ants_by_2(2) = 2
4341                  ants_sqr(1)  = mi_ants
4342                  ants_sqr(2)  = mi_ants
4343                  ants_lin(1)  = mi_ants
4344    c
4345                  call open_va( ant_add, ants_by_2, 2, dr_type, vstatus, msg )
4346                  call exit_va( vstatus, msg )
4347                  call open_va( eig_add, ants_lin, 1, dr_type, vstatus, msg )
```

```
DO_MUSIC                                                    18-Apr-1988 14:00:27     VA
                                                            18-Apr-1988 15:15:49     DC

4348                call exit_va( vstatus, msg )
4349                call open_va( eigv_add, ants_sqr, 2, dc_type, vstatus, msg )
4350                call exit_va( vstatus, msg )
4351     c
4352                work_file = 'MUSIC_INPUT__________.DAT'
4353                write (unit=work_file( 12 : 21 ),fmt=90020,iostat=status,
4354              1                                             err=60510) selexp
4355                open (unit=data_unit,file='6000$RT:[CHUCK.PARAMS]'//work_file,
4356              1           status='OLD',organization='SEQUENTIAL',
4357              2           access='SEQUENTIAL',form='UNFORMATTED',
4358              3           recordtype='FIXED',recl=64,dispose='KEEP',
4359              4                       READONLY,iostat=status,err=60560)
4360                read (unit=data_unit,err=60500)
4361                if ( .not. (
4362              1           readarr( %val( ant_add ), dr_type, mi_ants, 2,
4363              2                                   data_unit, 32 ) ) ) go to 60500
4364                close (unit=data_unit)
4365                work_file( 7 : 11 ) = 'EIGVS'
4366                write (unit=work_file( 12 : 21 ),fmt=90020,iostat=status,
4367              1                                             err=60510) mi_pkey
4368                open (unit=data_unit,file='6000$RT:[CHUCK.PARAMS]'//work_file,
4369              1           status='OLD',organization='SEQUENTIAL',
4370              2           access='SEQUENTIAL',form='UNFORMATTED',
4371              3           recordtype='FIXED',recl=64,dispose='KEEP',
4372              4                       READONLY,iostat=status,err=60560)
4373                do itemp = 1, 5
4374                  if ( .not. ( skiparr( mi_ants, mi_ants, data_unit, 16 ) ) )
4375              1                                                   go to 60500
4376                end do
4377                if ( ( mi_park .eq. -1 ) .and. ( mi_which .gt. 0 ) ) then
4378                  if ( .not. ( skiparr( mi_ants, mi_ants, data_unit, 16 ) ) )
4379              1                                                   go to 60500
4380                end if
4381                if ( .not. (
4382              1           readarr( %val( eig_add ), dr_type, mi_ants, 1,
4383              2                                   data_unit, 32 ) ) ) go to 60500
4384                if ( .not. (
4385              1           readarr( %val( eigv_add ), dc_type, mi_ants, mi_ants,
4386              2                                   data_unit, 16 ) ) ) go to 60500
4387                close (unit=data_unit)
4388     c
4389     c            Accept number of "noise" eigenvectors to use.
4390     c
4391                num_zeigs = type_eigs( %val( eig_add ), mi_ants, mi_tol )
4392                if ( num_zeigs .lt. 0 ) then
4393                  call dump_va( vstatus, msg )
4394                  call exit_va( vstatus, msg )
4395                  go to 61500
4396                end if
4397     c
4398     c            Open scratch file for DOA storage.
4399     c
4400                open (unit=data_unit,file='6000$RT:[CHUCK.PARAMS]SCRATCH.DAT',
4401              1           form='UNFORMATTED',recordtype='FIXED',recl=2,
4402              2           organization='RELATIVE',access='DIRECT',
4403              3           status='NEW',dispose='DELETE',iostat=status,err=60830)
4404     c
```

```
DO_MUSIC                                                    18-Apr-1988 14:00:27
                                                            18-Apr-1988 15:15:49

4405    c               Do plot(s) (finally!).
4406    c
4407                  psel = 'R'
4408                  do while ( psel .eq. 'R' )        !  Until operator tires.
4409                    call prange( mi_begp, mi_endp, mi_delp, fskip, count, iskip )
4410                    dummy = music_plot_spect( %val( ant_add ), mi_ants, mi_wlen,
4411             1                                %val( eigv_add ), num_zeigs,
4412             2                                mi_begp, mi_delp, fskip, count, iskip )
4413    c
4414                    num_doas = 0
4415                    write (unit=data_unit,rec=1,iostat=status,err=60820)
4416             1                                num_doas  !  Initialize # DOAs found.
4417                    psel = pick_music_peak( %val( ant_add ), mi_ants, mi_wlen,
4418             1                                %val( eigv_add ), num_zeigs,
4419             2                                data_unit, mi_begp, mi_delp,
4420             3                                                   fskip, count, iskip )
4421                  end do
4422    c
4423                  close (unit=data_unit)
4424                  call dump_va( vstatus, msg )
4425                  call exit_va( vstatus, msg )
4426    c
4427    c.....................Operator selected plots of "all" vectors for a trial...
4428    c
4429              else
4430    c
4431    c               Find antenna array description, which may be part of the
4432    c       selected experiment's "ancestor" experiment.
4433    c
4434                  ancestor = ( mi_park .eq. -1 )    !  True for "ancestor" exper.
4435                  if ( ancestor ) then                      !  "Ancestor".
4436                    selexp = mi_pkey
4437                  else                                      !  "Descendant".
4438                    selexp = mi_park
4439    61460           continue
4440                    read (unit=index_unit,keyeq=selexp,keyid=0,iostat=status)
4441             1                                              selexp, tch, selpar
4442                    unlock (unit=index_unit)
4443                    if      ( status .eq. FOR$IOS_SPERECLOC ) then
4444                      type *, 'Record #', selexp, ' locked.  Waiting 1 sec.',
4445             1                                                        uparrow
4446                      call lib$wait( 1.0 )                  !  Wait 1 second.
4447                      type *, spaces, uparrow
4448                      go to 61460
4449                    else if ( status .ne. 0 ) then          !  Error reading index.
4450                      mi_pkey = selexp
4451                      go to 60700
4452                    end if
4453                    if ( selpar .ne. -1 ) then
4454                      selexp = selpar
4455                      go to 61460
4456                    end if
4457                  end if
4458    c
4459    c               Now open required Virtual Arrays.
4460    c
4461                  call init_va
```

```
DO_MUSIC                                                    18-Apr-1988 14:00:27    VAX FO
                                                            18-Apr-1988 15:15:49    [CHUCK

4462                ants_lin(1)    = mi_ants
4463                ants_by_2(1)   = mi_ants
4464                ants_by_2(2)   = 2
4465                ants_sqr(1)    = mi_ants
4466                ants_sqr(2)    = mi_ants
4467                if ( ancestor ) then                !  "Ancestor".
4468                  ants_by_smp(1) = mi_ants
4469                  ants_by_smp(2) = mi_smp
4470                end if
4471                wavs_by_smp(2) = mi_smp
4472                smp_by_ants(1) = mi_smp
4473                smp_by_ants(2) = mi_ants
4474     c
4475                call open_va( ant_add, ants_by_2, 2, dr_type, vstatus, msg )
4476                call exit_va( vstatus, msg )
4477                call open_va( eig_add, ants_lin, 1, dr_type, vstatus, msg )
4478                call exit_va( vstatus, msg )
4479                call open_va( eigv_add, ants_sqr, 2, dc_type, vstatus, msg )
4480                call exit_va( vstatus, msg )
4481                call open_va( obsmat_add, smp_by_ants, 2, dc_type, vstatus, msg )
4482                call exit_va( vstatus, msg )
4483                if ( ancestor ) then                !  "Ancestor".
4484                  call open_va( noisemat_add, smp_by_ants, 2, dc_type, vstatus,
4485               1                                                              msg )
4486                  call exit_va( vstatus, msg )
4487                  call open_va( clnmat_add, ants_by_smp, 2, dc_type, vstatus,
4488               1                                                              msg )
4489                  call exit_va( vstatus, msg )
4490                end if
4491     c
4492                work_file = 'MUSIC_INPUT__________.DAT'
4493                write (unit=work_file( 12 : 21 ),fmt=90020,iostat=status,
4494               1                                            err=60510 ) selexp
4495                open (unit=data_unit,file='6000$RT:[CHUCK.PARAMS]'//work_file,
4496               1          status='OLD',organization='SEQUENTIAL',
4497               2          access='SEQUENTIAL',form='UNFORMATTED',
4498               3          recordtype='FIXED',recl=64,dispose='KEEP',
4499               4                    READONLY,iostat=status,err=60560)
4500                read (unit=data_unit,err=60500)
4501                if ( .not. (
4502               1          readarr( %val( ant_add ), dr_type, mi_ants, 2,
4503               2                                    data_unit, 32 ) ) ) go to 60500
4504                close (unit=data_unit)
4505                work_file( 7 : 11 ) = 'EIGVS'
4506                write (unit=work_file( 12 : 21 ),fmt=90020,iostat=status,
4507               1                                            err=60510 ) mi_pkey
4508                open (unit=data_unit,file='6000$RT:[CHUCK.PARAMS]'//work_file,
4509               1          status='OLD',organization='SEQUENTIAL',
4510               2          access='SEQUENTIAL',form='UNFORMATTED',
4511               3          recordtype='FIXED',recl=64,dispose='KEEP',
4512               4                    READONLY,iostat=status,err=60560)
4513                do itemp = 1, 5
4514                  if ( .not. ( skiparr( mi_ants, mi_ants, data_unit, 16 ) ) )
4515               1                                                  go to 60500
4516                end do
4517                if ( ( mi_park .eq. -1 ) .and. ( mi_which .gt. 0 ) ) then
4518                  if ( .not. ( skiparr( mi_ants, mi_ants, data_unit, 16 ) ) )
```

```
DO_MUSIC                                                    18-Apr-1988 14:00:27    VAX FC
                                                            18-Apr-1988 15:15:49    [CHUCK
4519            1                                                       go to 60500
4520                  end if
4521                  if ( .not. (
4522            1                readarr( %val( eig_add ), dr_type, mi_ants, 1,
4523            2                                         data_unit, 32 ) ) ) go to 60500
4524                  if ( .not. (
4525            1                readarr( %val( eigv_add ), dc_type, mi_ants, mi_ants,
4526            2                                         data_unit, 16 ) ) ) go to 60500
4527                  close (unit=data_unit)
4528                  work_file( 7 : 11 ) = 'SAMPL'
4529                  open (unit=data_unit,file='6000$RT:[CHUCK.PARAMS]'//work_file,
4530            1                status='OLD',organization='SEQUENTIAL',
4531            2                access='SEQUENTIAL',form='UNFORMATTED',
4532            3                recordtype='FIXED',recl=64,dispose='KEEP',
4533            4                                   READONLY,iostat=status,err=60560)
4534                  if ( .not. ancestor) then                 !  "Descendant".
4535                    if ( .not. ( skiparr( mi_doas, 1,               !  Skip DOAs.
4536            1                                         data_unit, 32 ) ) ) go to 60500
4537                  end if
4538                  if ( .not. (
4539            1                readarr( %val( obsmat_add ), dc_type, mi_smp, mi_ants,
4540            2                                         data_unit, 16 ) ) ) go to 60500
4541                  if ( ancestor ) then                      !  "Ancestor".
4542                    if ( .not. (
4543            1                readarr( %val( noisemat_add ), dc_type, mi_smp,
4544            2                         mi_ants, data_unit, 16 ) ) ) go to 60500
4545                    if ( .not. ( skiparr( mi_waves, mi_smp, !  Skip sigvm_add.
4546            1                                         data_unit, 16 ) ) ) go to 60500
4547                    if ( .not. (
4548            1                readarr( %val( cinmat_add ), dc_type, mi_ants, mi_smp,
4549            2                                         data_unit, 16 ) ) ) go to 60500
4550                  end if
4551                  close (unit=data_unit)
4552      c
4553      c             Accept number of "noise" eigenvectors to use.
4554      c
4555                  num_zeigs = type_eigs( %val( eig_add ), mi_ants, mi_tol )
4556                  if ( num_zeigs .lt. 0 ) then
4557                    call dump_va( vstatus, msg )
4558                    call exit_va( vstatus, msg )
4559                    go to 61500
4560                  end if
4561      c
4562      c             At last do plots.
4563      c
4564                  call music_plot_trial( %val( obsmat_add ), %val( noisemat_add ),
4565            1                            %val( cinmat_add ), mi_ants, mi_smp, mi_begp,
4566            2                            mi_endp, mi_delp, %val( ant_add ), mi_wien,
4567            3                            %val( eigv_add ), num_zeigs, mi_which,
4568            4                                                       ancestor )
4569      c
4570                  call dump_va( vstatus, msg )
4571                  call exit_va( vstatus, msg )
4572                end if
4573      61500     continue
4574              end do
4575      c
```

```
DO_MUSIC                                                    18-Apr-1988 14:00:27
                                                            18-Apr-1988 15:15:49

4576    c               Operator finished viewing results for the selected experiment
4577    c       return to ask the operator what to do next.
4578    c
4579              go to 20000
4580    c
4581    c!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
4582    c
4583    c               Input file re-creation selected.
4584    c
4585            else if ( psel .eq. 'I' ) then
4586              assign 20000 to err_return
4587    c
4588              type *, 'Please enter the number of the experiment to be restarted'
4589              type 90000, '              (<CR> for a list of applicable '
4590             1                                                  // 'experiments):  '
4591              accept 90010, answer
4592              read (unit=answer,fmt=*,err=70430) mi_pkey
4593              if ( mi_pkey .lt. 0 ) then
4594    70430       continue                                        !  Error in accepted
4595                mi_pkey = p_index( index_unit, read_unit1, read_unit2, '-X',
4596             1                                                          .true. )
4597                if ( mi_pkey .eq. -1 ) then          !  No restartable exps.
4598    70000         continue
4599                  type *
4600                  type *, char( 7 ), 'Can''t re-start.'
4601                  type *
4602                  go to 20000
4603                end if
4604                if ( mi_pkey .lt.  0 ) go to 20000  !  Operator cancelled.
4605              end if
4606    c
4607    70410     continue
4608              read (unit=index_unit,keyeq=mi_pkey,keyid=0,iostat=status)
4609             1               mi_pkey, mi_stat, mi_park, mi_ierr, mi_eerr, mi_serr,
4610             2               mi_dtim, mi_ants, mi_waves, mi_noises, mi_smp, mi_sir
4611             3               mi_begp, mi_endp, mi_delp, mi_peps, mi_norm, mi_wlen,
4612             4               mi_tol, mi_which, mi_ofile, mi_anoises, mi_irlslim,
4613             5               mi_zeigs, mi_doas, mi_estype, mi_parp
4614              unlock (unit=index_unit)
4615              if      ( status .eq. FOR$IOS_SPERECLOC ) then
4616                type *, 'Record #', mi_pkey, ' locked.  Waiting 1 sec.', uparrow
4617                call lib$wait( 1.0 )                        !  Wait 1 second.
4618                type *, spaces, uparrow
4619                go to 70410
4620              else if ( ( status .ne. 0 )
4621             1              .or. ( index( '-X', mi_stat ) .le. 0 ) ) then
4622                go to 70000
4623              end if
4624    c
4625              havent_irec = .false.
4626              go to 51200
4627    c
4628    c!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
4629    c
4630    c               New "trial" (rerun of initialized experiment) selected.  Note
4631    c       that each "trial" will result in a NEW output file with the name
4632    c       6000$RT:[CHUCK.PARAMS]MUSIC_OUTPUTnnnnnnnnnn.DAT, where nnnnnnnnnn is
```

```
DO_MUSIC                                                    18-Apr-1988 14:00:27     V
                                                            18-Apr-1988 15:15:49     D

4633     c       the (new) experiment number.  Note also that, since each new "trial"  !
4634     c       implies the generation of a new set of noise samples, this option     !
4635     c       makes sense only for "ancestor" experiments.                          !
4636     c                                                                             !
4637             else if ( psel .eq. 'R' ) then
4638               assign 20000 to err_return
4639     c
4640               type *, 'Please enter the number of the experiment to be rerun'
4641               type 90000, '            (<CR) for a list of applicable '
4642             1                                                  // 'experiments):  '
4643               accept 90010, answer
4644               read (unit=answer,fmt=*,err=51430) mi_pkey
4645               if ( mi_pkey .lt. 0 ) then
4646     51430       continue                                      !  Error in accepted #.
4647                 mi_pkey = p_index( index_unit, read_unit1, read_unit2, ' IER',
4648             1                                                          .true. )
4649                 if ( mi_pkey .eq. -1 ) then          !  No rerunnable experiments.
4650     51000         continue
4651                   type *
4652                   type *, char( 7 ), 'Can''t re-run.'
4653                   type *
4654                   go to 20000
4655                 end if
4656                 if ( mi_pkey .lt.  0 ) go to 20000  !  Operator cancelled.
4657               end if
4658     c
4659     51410     continue
4660               read (unit=index_unit,keyeq=mi_pkey,keyid=0,iostat=status)
4661             1               mi_pkey, mi_stat, mi_park, mi_ierr, mi_eerr, mi_serr,
4662             2               mi_dtim, mi_ants, mi_waves, mi_noises, mi_smp, mi_sint,
4663             3               mi_begp, mi_endp, mi_delp, mi_peps, mi_norm, mi_wien,
4664             4               mi_tol, mi_which, mi_ofile, mi_anoises, mi_irlslim,
4665             5               mi_zeigs, mi_doas, mi_estype, mi_parp
4666               unlock (unit=index_unit)
4667               if      ( status .eq. FOR$IOS_SPERECLOC ) then
4668                 type *, 'Record #', mi_pkey, ' locked.  Waiting 1 sec.', uparrow
4669                 call lib$wait( 1.0 )                        !  Wait 1 second.
4670                 type *, spaces, uparrow
4671                 go to 51410
4672               else if ( ( status .ne. 0 )
4673             1               .or. ( index( ' IER', mi_stat ) .le. 0 )
4674             2               .or. ( mi_park .ne. -1 )
4675             3               ) then
4676                 go to 51000
4677               end if
4678     c
4679     c              Experiment to rerun has primary key value mi_pkey, and it is a
4680     c       valid rerun candidate.  Now create required Virtual Arrays.
4681     c
4682               call init_va
4683               ants_by_2(1)    = mi_ants
4684               ants_by_2(2)    = 2
4685               waves_by_9(1)   = mi_waves
4686               waves_by_9(2)   = 9
4687               waves_by_2(1)   = mi_waves
4688               waves_by_2(2)   = 2
4689               nois_by_2(1)    = mi_noises
```

```
DO_MUSIC                                                    18-Apr-1988 14:00:27     VAX
                                                            18-Apr-1988 15:15:49     [CH
```

```
4690              nois_by_2(2)     = 2
4691              nois_by_3(1)     = mi_noises
4692              nois_by_3(2)     = 3
4693              ants_by_nois(1) = mi_ants
4694              ants_by_nois(2) = mi_noises
4695              if ( mi_which .gt. 0 ) then
4696                anois_by_2(1)     = mi_anoises
4697                anois_by_2(2)     = 2
4698                anois_by_3(1)     = mi_anoises
4699                anois_by_3(2)     = 3
4700                ants_by_anois(1) = mi_ants
4701                ants_by_anois(2) = mi_anoises
4702              end if
4703     c
4704              call open_va( ant_add, ants_by_2, 2, dr_type, vstatus, msg )
4705              call exit_va( vstatus, msg )
4706              call open_va( wave_add, waves_by_9, 2, dr_type, vstatus, msg )
4707              call exit_va( vstatus, msg )
4708              call open_va( mtype_add, waves_by_2, 2, di_type, vstatus, msg )
4709              call exit_va( vstatus, msg )
4710              call open_va( ndef_add, nois_by_2, 2, di_type, vstatus, msg )
4711              call exit_va( vstatus, msg )
4712              call open_va( npars_add, nois_by_3, 2, sr_type, vstatus, msg )
4713              call exit_va( vstatus, msg )
4714              call open_va( correl_add, ants_by_nois, 2, dr_type, vstatus, msg )
4715              call exit_va( vstatus, msg )
4716              if ( mi_which .gt. 0 ) then
4717                call open_va( andef_add, anois_by_2, 2, di_type, vstatus, msg )
4718                call exit_va( vstatus, msg )
4719                call open_va( anpars_add, anois_by_3, 2, sr_type, vstatus, msg )
4720                call exit_va( vstatus, msg )
4721                call open_va( acorrel_add, ants_by_anois, 2, dr_type, vstatus,
4722            1                                                              msg )
4723                call exit_va( vstatus, msg )
4724              end if
4725     c
4726     c                 Get problem definition from input file, initialize the noise
4727     c           counters, and form the array phase shift matrix.
4728     c
4729              work_file = 'MUSIC_INPUT__________.DAT'
4730              write (unit=work_file( 12 : 21 ),fmt=90020,iostat=status,
4731            1                                                 err=60510) mi_pkey
4732     c
4733              open (unit=data_unit,file='6000$RT:[CHUCK.PARAMS]'//work_file,
4734            1       status='OLD',organization='SEQUENTIAL',access='SEQUENTIAL',
4735            2       form='UNFORMATTED',recordtype='FIXED',recl=64,dispose='KEEP',
4736            3                                READONLY,iostat=status,err=60560)
4737              read (unit=data_unit,err=60500)
4738              if ( .not. (
4739            1       readarr( %val( ant_add ), dr_type, mi_ants, 2, data_unit, 32 )
4740            2                                                          ) ) go to 60500
4741              if ( .not. (
4742            1       readarr( %val( wave_add ), dr_type, mi_waves, 9, data_unit,
4743            2                                                              32 )
4744            3                                                        ) ) go to 60500
4745              if ( .not. (
4746            1       readarr( %val( mtype_add ), di_type, mi_waves, 2, data_unit,
```

```
     2                                                  64 )
     3                                              ) ) go to 60500
        if ( .not. (
     1        readarr( %val( ndef_add ), di_type, mi_noises, 2, data_unit,
     2                                                  64 )
     3                                              ) ) go to 60500
        if ( .not. (
     1        readarr( %val( npars_add ), sr_type, mi_noises, 3, data_unit,
     2                                                  64 )
     3                                              ) ) go to 60500
        if ( .not. (
     1        readarr( %val( correl_add ), dr_type, mi_ants, mi_noises,
     2                                          data_unit, 32 )
     3                                              ) ) go to 60500
        if ( mi_which .gt. 0 ) then
          if ( .not. (
     1        readarr( %val( andef_add ), di_type, mi_anoises, 2,
     2                                          data_unit, 64 )
     3                                              ) ) go to 60500
          if ( .not. (
     1        readarr( %val( anpars_add ), sr_type, mi_anoises, 3,
     2                                          data_unit, 64 )
     3                                              ) ) go to 60500
          if ( .not. (
     1        readarr( %val( acorrel_add ), dr_type, mi_ants, mi_anoises,
     2                                          data_unit, 32 )
     3                                              ) ) go to 60500
        end if
        close (unit=data_unit)
c
        havent_irec = .true.
        go to 51210
c                                                                        !
c!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
c                                                                        !
c             New experiment selected.                                   !
c                                                                        !
      else if ( psel .eq. 'N' ) then
        assign 20000 to err_return
        havent_irec = .true.
c
51200   continue                 !  Branch to restart a failed experiment.
        type *
        type *, '   Take defaults from which experiment number?'
        if ( havent_irec ) then
          type *, '                 (-1 for "standard" defaults)'
        else
          type *, '                        (-1 for current values)'
        end if
        type 90000, ' (<CR> to see a list of available experiments):  '
        accept 90010, answer
        read (unit=answer,fmt=*,err=51100) selexp
        go to 51110
c
51100     continue                                   !  Error in accepted #.
          selexp = p_index( index_unit, read_unit1, read_unit2,
     1                                     ' IEDR', .true. )
```

```
DO_MUSIC                                                    18-Apr-1988 14:00:27     VAX
                                                            13-Apr-1988 15:15:49     [CHL
4804                  if ( selexp .lt. -1 ) go to 20000           !  Operator cancelled.
4805      c
4806      51110     continue
4807                if ( selexp .ge. 0 ) then                     !  Defaults specified.
4808      51120       continue
4809                  read (unit=index_unit,keyeq=selexp,keyid=0,iostat=status)
4810                1               selexp, mi_stat, mi_park, mi_ierr, mi_eerr, mi_serr,
4811                2               mi_dtim, mi_ants, mi_waves, mi_noises, mi_smp, mi_sint,
4812                3               mi_begp, mi_endp, mi_delp, mi_peps, mi_norm, mi_wlen,
4813                4               mi_tol, mi_which, mi_ofile, mi_anoises, mi_irlslim,
4814                5               mi_zeigs, mi_doas, mi_estype, mi_parp
4815                  unlock (unit=index_unit)
4816                  if      ( status .eq. FOR$IOS_SPERECLOC ) then
4817                    type *, 'Record #', selexp, ' locked.  Waiting 1 sec.', uparrow
4818                    call lib$wait( 1.0 )                      !  Wait 1 second.
4819                    type *, spaces, uparrow
4820                    go to 51120
4821                  else if ( ( status .ne. 0 )
4822                1                 .or. ( index( ' IEDR', mi_stat ) .le. 0 ) ) then
4823      51150       continue
4824                  type *
4825                  type *, char( 7 ), 'Can''t use that experiment for defaults.'
4826                  type *, 'Setting "standard" defaults.'
4827                  type *
4828                  selexp = -1
4829                  go to 51130
4830                end if
4831      c
4832      c               Experiment from which to obtain default values has primary key
4833      c         value selexp, and it is a valid defaults candidate.  Replace it with
4834      c         its "ancestor" if it is a "descendant" experiment.
4835      c
4836                  do while ( mi_park .ne. -1 )                !  "Descendant".
4837                    selexp = mi_park
4838      51140         continue
4839                    read (unit=index_unit,keyeq=selexp,keyid=0,iostat=status)
4840                1               selexp, mi_stat, mi_park, mi_ierr, mi_eerr, mi_serr,
4841                2               mi_dtim, mi_ants, mi_waves, mi_noises, mi_smp, mi_sint,
4842                3               mi_begp, mi_endp, mi_delp, mi_peps, mi_norm, mi_wlen,
4843                4               mi_tol, mi_which, mi_ofile, mi_anoises, mi_irlslim,
4844                5               mi_zeigs, mi_doas, mi_estype, mi_parp
4845                    unlock (unit=index_unit)
4846                    if      ( status .eq. FOR$IOS_SPERECLOC ) then
4847                      type *, 'Record #', selexp, ' locked.  Waiting 1 sec.', uparrow
4848                      call lib$wait( 1.0 )                    !  Wait 1 second.
4849                      type *, spaces, uparrow
4850                      go to 51140
4851                    else if ( status .ne. 0 ) then            !  Error reading index.
4852                      go to 51150
4853                    end if
4854                  end do
4855      c
4856                else if ( havent_irec ) then                  !  No defaults spec'd.
4857                  mi_park = -1
4858                  mi_ants = 5
4859                  mi_waves = 3
4860                  mi_noises = 5
```

```
DO_MUSIC                                                    18-Apr-1988 14:00:27    VAX
                                                            18-Apr-1988 15:15:49    [CH
```

```
4861                mi_smp = 101
4862                mi_sint = .25
4863                mi_begp = 1.5
4864                mi_endp = 2.0
4865                mi_delp = 0.5
4866                mi_peps = 0.01
4867                mi_norm = 1.0
4868                mi_wlen = 3.0
4869                mi_tol = 1.0e-10
4870                mi_which = 1
4871                mi_anoises = 6
4872                mi_irlslim = 100
4873                mi_zeigs = 0
4874                mi_doas = 0
4875                mi_estype = 0
4876                mi_parp = 0.0
4877              end if
4878     c
4879     51130    continue
4880              call date( mi_dtim(  1:9  ) )
4881              mi_dtim( 10:11 ) = '  '
4882              call time( mi_dtim( 12:19 ) )
4883     c
4884              old_ants = mi_ants
4885              old_waves = mi_waves
4886              old_noises = mi_noises
4887              old_anoises = mi_anoises
4888              old_which = mi_which
4889              old_smp = mi_smp
4890              call get_music_def( mi_ants, mi_waves, mi_noises,
4891             1                                          mi_anoises, mi_which, mi_smp )
4892     c
4893     c                   Now create required Virtual Arrays.
4894     c
4895              call init_va
4896              ants_by_2(1)    = mi_ants
4897              ants_by_2(2)    = 2
4898              waves_by_9(1)   = mi_waves
4899              waves_by_9(2)   = 9
4900              waves_by_2(1)   = mi_waves
4901              waves_by_2(2)   = 2
4902              nois_by_2(1)    = mi_noises
4903              nois_by_2(2)    = 2
4904              nois_by_3(1)    = mi_noises
4905              nois_by_3(2)    = 3
4906              ants_by_nois(1) = mi_ants
4907              ants_by_nois(2) = mi_noises
4908              if ( mi_which .gt. 0 ) then
4909                anois_by_2(1)    = mi_anoises
4910                anois_by_2(2)    = 2
4911                anois_by_3(1)    = mi_anoises
4912                anois_by_3(2)    = 3
4913                ants_by_anois(1) = mi_ants
4914                ants_by_anois(2) = mi_anoises
4915              end if
4916     c
4917              call open_va( ant_add, ants_by_2, 2, dr_type, vstatus, msg )
```

```
4918              call exit_va( vstatus, msg )
4919              call open_va( wave_add, waves_by_9, 2, dr_type, vstatus, msg )
4920              call exit_va( vstatus, msg )
4921              call open_va( mtype_add, waves_by_2, 2, di_type, vstatus, msg )
4922              call exit_va( vstatus, msg )
4923              call open_va( ndef_add, nois_by_2, 2, di_type, vstatus, msg )
4924              call exit_va( vstatus, msg )
4925              call open_va( npars_add, nois_by_3, 2, sr_type, vstatus, msg )
4926              call exit_va( vstatus, msg )
4927              call open_va( correl_add, ants_by_nois, 2, dr_type, vstatus, msg )
4928              call exit_va( vstatus, msg )
4929              if ( mi_which .gt. 0 ) then
4930                call open_va( andef_add, anois_by_2, 2, di_type, vstatus, msg )
4931                call exit_va( vstatus, msg )
4932                call open_va( anpars_add, anois_by_3, 2, sr_type, vstatus, msg )
4933                call exit_va( vstatus, msg )
4934                call open_va( acorrel_add, ants_by_anois, 2, dr_type, vstatus,
4935             1                                                              msg )
4936                call exit_va( vstatus, msg )
4937              end if
4938     c
4939     c                 Get antenna positions, wave characteristics, noise definitions,
4940     c           sample interval, tolerance, and parameters for Lp estimate calculation.
4941     c           First get values from selected default experiment, if any.
4942     c
4943              if ( selexp .ge. 0 ) then              !  If default was selected.
4944                work_file = 'MUSIC_INPUT__________.DAT'
4945                write (unit=work_file( 12 : 21 ),fmt=90020,iostat=status,
4946             1                                                  err=51310) selexp
4947                open (unit=data_unit,file='6000$RT:[CHUCK.PARAMS]'//work_file,
4948             1       status='OLD',organization='SEQUENTIAL',access='SEQUENTIAL',
4949             2       form='UNFORMATTED',recordtype='FIXED',recl=64,dispose='KEEP',
4950             3                              READONLY,iostat=status,err=51310)
4951                read (unit=data_unit,err=51300)
4952                if ( old_ants .eq. mi_ants ) then
4953                  if ( .not. ( readarr( %val( ant_add ), dr_type,
4954             1                       mi_ants, 2, data_unit, 32 ) ) ) go to 51300
4955                else
4956                  if ( .not. ( skiparr( old_ants, 2, data_unit, 32 ) )
4957             1                                                      ) go to 51300
4958                end if
4959                if ( old_waves .eq. mi_waves ) then
4960                  if ( .not. ( readarr( %val( wave_add ), dr_type,
4961             1                       mi_waves, 9, data_unit, 32 ) ) ) go to 51300
4962                  if ( .not. ( readarr( %val( mtype_add ), di_type,
4963             1                       mi_waves, 2, data_unit, 64 ) ) ) go to 51300
4964                else
4965                  if ( .not. ( skiparr( old_waves, 9, data_unit, 32 ) )
4966             1                                                      ) go to 51300
4967                  if ( .not. ( skiparr( old_waves, 2, data_unit, 64 ) )
4968             1                                                      ) go to 51300
4969                end if
4970                if ( old_noises .eq. mi_noises ) then
4971                  if ( .not. ( readarr( %val( ndef_add ), di_type,
4972             1                       mi_noises, 2, data_unit, 64 ) ) ) go to 51300
4973                  if ( .not. ( readarr( %val( npars_add ), sr_type,
4974             1                       mi_noises, 3, data_unit, 64 ) ) ) go to 51300
```

```
DO_MUSIC                                                    18-Apr-1988 14:00:27    VA>
                                                            18-Apr-1988 15:15:49    [CH
4975                  if ( old_ants .eq. mi_ants ) then
4976                    if ( .not. ( readarr( %val( correl_add ), dr_type, mi_ants,
4977               1                       mi_noises, data_unit, 32 ) ) ) go to 51300
4978                  else
4979                    if ( .not. ( skiparr( old_ants, old_noises, data_unit, 64 ) )
4980               1                                                        ) go to 51300
4981                  end if
4982                else
4983                  if ( .not. ( skiparr( mi_noises, 2, data_unit, 64 ) )
4984               1                                                        ) go to 51300
4985                  if ( .not. ( skiparr( mi_noises, 3, data_unit, 64 ) )
4986               1                                                        ) go to 51300
4987                  if ( .not. ( readarr( %val( correl_add ), dr_type, mi_ants,
4988               1                       mi_noises, data_unit, 32 ) ) ) go to 51300
4989                end if
4990                if ( ( mi_which .gt. 0 )
4991               1                .and. ( old_which .gt. 0 )
4992               2                .and. ( old_anoises .eq. mi_anoises ) ) then
4993                  if ( .not. ( readarr( %val( andef_add ), di_type,
4994               1                       mi_anoises, 2, data_unit, 64 ) ) ) go to 51300
4995                  if ( .not. ( readarr( %val( anpars_add ), sr_type,
4996               1                       mi_anoises, 3, data_unit, 64 ) ) ) go to 51300
4997                  if ( old_ants .eq. mi_ants ) then
4998                    if ( .not. ( readarr( %val( acorrel_add ), dr_type, mi_ants,
4999               1                       mi_anoises, data_unit, 32 ) ) ) go to 51300
5000                  end if
5001                end if
5002                close (unit=data_unit)
5003              end if
5004              go to 51320
5005    c
5006    c               Error reading defaults branch - abandon defaults.
5007    c
5008    51300     continue
5009              close (unit=data_unit)
5010    c
5011    51310     continue
5012              selexp = -1
5013              type *
5014              type *, 'Error accessing defaults definition file.'
5015              type *, 'Setting "standard" defaults.'
5016              type *
5017    c
5018    51320     continue
5019              call read_antennae( %val( ant_add ), mi_ants, old_ants, selexp )
5020              call read_waves( mi_waves, old_waves, selexp, %val( wave_add ),
5021             1                                              %val( mtype_add ) )
5022              if ( mi_which .gt. 0 ) then                   !  Get actual noise?
5023                call read_noise( selexp, mi_ants, old_ants, mi_anoises,
5024             1                   old_anoises, %val( andef_add ), %val( anpars_add ),
5025             2                                %val( acorrel_add ), -1, old_which )
5026              end if
5027              call read_noise( selexp, mi_ants, old_ants, mi_noises,
5028             1                 old_noises, %val( ndef_add ), %val( npars_add ),
5029             2                           %val( correl_add ), mi_which, old_which )
5030              if ( havent_irec ) then                       !  New, not recreate.
5031                call read_mmisc( selexp, mi_wlen, mi_sint, mi_tol )
```

```
DO_MUSIC                                                    18-Apr-1988 14:00:27
                                                            13-Apr-1988 15:15:49

5032                call read_lp( selexp, mi_begp, mi_endp, mi_delp,
5033               1                                        mi_peps, mi_norm, mi_irlslim )
5034                end if
5035        c
5036        c               Convert input scalar wavelength to vector of (identical)
5037        c       wavelengths.
5038        c
5039                call dr_colfill( %val( wave_add ), mi_waves, 9, mi_wlen, 9 )
5040        c
5041        c               Obtain an (unused) experiment number, create a (hopefully)
5042        c       unique output file name, and enter the initial experiment description
5043        c       in the index.
5044        c
5045        51210   continue                !  Branch to create new trial or experiment.
5046                if ( havent_irec ) then                         !  New, not recreate.
5047                  mi_pkey = avlbl_prikey( index_unit, 0 )
5048                  if ( mi_pkey .lt. 0 ) go to 50000
5049        c
5050                  mi_stat = '-'
5051                  mi_ierr = 0
5052                  mi_eerr = 0
5053                  mi_serr = 0
5054        c
5055                  mi_ofile = '6000$RT:[CHUCK.PARAMS]MUSIC_OUTPUT__________.DAT'
5056                  write (unit=mi_ofile( 35 : 44 ),fmt=90020,err=60510) mi_pkey
5057        c
5058                  write (unit=index_unit,err=50000)
5059               1                  mi_pkey, mi_stat, mi_park, mi_ierr, mi_eerr, mi_serr,
5060               2                  mi_dtim, mi_ants, mi_waves, mi_noises, mi_smp, mi_sint,
5061               3                  mi_begp, mi_endp, mi_delp, mi_peps, mi_norm, mi_wlen,
5062               4                  mi_tol, mi_which, mi_ofile, mi_anoises, mi_irlslim,
5063               5                  mi_zeigs, mi_doas, mi_estype, mi_parp
5064                  unlock (unit=index_unit)
5065                end if
5066        c
5067                work_file = 'MUSIC_INPUT__________.DAT'
5068                write (unit=work_file( 12 : 21 ),fmt=90020,iostat=status,
5069               1                                                err=60510) mi_pkey
5070        c
5071                call lib$delete_file( '6000$RT:[CHUCK.PARAMS]' // work_file // ';*' )
5072                open (unit=data_unit,file='6000$RT:[CHUCK.PARAMS]'//work_file,
5073               1       status='NEW',organization='SEQUENTIAL',access='SEQUENTIAL',
5074               2       form='UNFORMATTED',recordtype='FIXED',recl=64,dispose='KEEP',
5075               3                                        iostat=status,err=60510)
5076                write (unit=data_unit,err=60500)
5077               1                  mi_pkey, mi_stat, mi_park, mi_ierr, mi_eerr, mi_serr,
5078               2                  mi_dtim, mi_ants, mi_waves, mi_noises, mi_smp, mi_sint,
5079               3                  mi_begp, mi_endp, mi_delp, mi_peps, mi_norm, mi_wlen,
5080               4                  mi_tol, mi_which, mi_ofile, mi_anoises, mi_irlslim,
5081               5                  mi_zeigs, mi_doas, mi_estype, mi_parp
5082                if ( .not. ( writearr( %val( ant_add ), dr_type,
5083               1                        mi_ants, 2, data_unit, 32 ) ) ) go to 60500
5084                if ( .not. ( writearr( %val( wave_add ), dr_type,
5085               1                        mi_waves, 9, data_unit, 32 ) ) ) go to 60500
5086                if ( .not. ( writearr( %val( mtype_add ), di_type,
5087               1                        mi_waves, 2, data_unit, 64 ) ) ) go to 60500
5088                if ( .not. ( writearr( %val( ndef_add ), di_type,
```

```
             1                      mi_noises, 2, data_unit, 64 ) ) ) go to 60500
               if ( .not. ( writearr( %val( npars_add ), sr_type,
             1                      mi_noises, 3, data_unit, 64 ) ) ) go to 60500
               if ( .not. ( writearr( %val( correl_add ), dr_type, mi_ants,
             1                      mi_noises, data_unit, 32 ) ) ) go to 60500
               if ( mi_which .gt. 0 ) then
                 if ( .not. ( writearr( %val( andef_add ), di_type,
             1                      mi_anoises, 2, data_unit, 64 ) ) ) go to 60500
                 if ( .not. ( writearr( %val( anpars_add ), sr_type,
             2                      mi_anoises, 3, data_unit, 64 ) ) ) go to 60500
                 if ( .not. ( writearr( %val( acorrel_add ), dr_type, mi_ants,
             1                      mi_anoises, data_unit, 32 ) ) ) go to 60500
               end if
               close (unit=data_unit)
c
c                    Update index file entry to reflect creation of the input file.
c
               mi_stat = ' '
               go to 60600
c
c                    Special file I/O error path for creating a new experiment.
c
      50000    continue
                 call dump_va( vstatus, msg )
                 call exit_va( vstatus, msg )
                 type *
                 type *, 'Error initializing experiment; new trial / experiment ',
             1                                                      'not created.'
                 type *, '(And you can''t do a thing about it!)'
                 type *
                 go to 20000
c
c!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
c                                                                               !
c                    Exit to operating system selected.                         !
c                                                                               !
             else if ( psel .eq. 'X' ) then
               status = sys$setprn( procname( 1 : pname_len ) )
               if (.not. status) call lib$stop( %val( status ) )
               stop
c                                                                               !
c!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
c                                                                               !
c                    No legitimate selection made.                              !
c                                                                               !
             end if
             go to 20000
c                                                                               !
c!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
c*******************************************************************************
c
c                    Branch to rewrite the currently selected experiment's index
c            record with a revised status (mi_stat set to desired new value).  If
c            the rewrite is successful, this program segment returns to statement
c            number 20000, otherwise index-record I/O error processing is initiated.
c
      60600  continue
```

```
DO_MUSIC                                                    18-Apr-1988 14:00:27     \
                                                            18-Apr-1988 15:15:49     !

5146            read (unit=index_unit,key=mi_pkey,keyid=0,iostat=status)
5147            if      ( status .eq. FOR$IOS_SPERECLOC ) then
5148              unlock (unit=index_unit)
5149              type *, 'Index record locked by another program ...'
5150              call lib$wait( 10.0 )                 !  Wait 10 seconds.
5151              type *, 'Trying again.'
5152              go to 60600
5153            else if ( status .ne. 0 ) then
5154              unlock (unit=index_unit)
5155              go to 60700
5156            end if
5157            rewrite (unit=index_unit,iostat=status,err=60700)
5158            1               mi_pkey, mi_stat, mi_park, mi_ierr, mi_eerr, mi_serr,
5159            2               mi_dtim, mi_ants, mi_waves, mi_noises, mi_smp, mi_sint,
5160            3               mi_begp, mi_endp, mi_delp, mi_peps, mi_norm, mi_wien,
5161            4               mi_tol, mi_which, mi_ofile, mi_anoises, mi_irlslim,
5162            5               mi_zeigs, mi_doas, mi_estype, mi_parp
5163            unlock (unit=index_unit)
5164            go to 20000
5165    c
5166    c*******************************************************************************
5167    c
5168    c               File error processing branches.
5169    c
5170    c
5171    c               Error opening intermediate data file - could be locked.
5172    c
5173    60550   continue
5174            close (unit=data_unit)
5175    c
5176    60560   continue
5177            call dump_va( vstatus, msg )
5178            call exit_va( vstatus, msg )
5179    c
5180    60570   continue
5181            type *
5182            type *, char( 7 ), 'Can''t open required data file.'
5183            type *, 'File may be in use by another program.'
5184            type *
5185            go to err_return
5186    c
5187    c               Error reading an open intermediate data file - must be some
5188    c       problem with the file itself.
5189    c
5190    60500   continue
5191            close (unit=data_unit)
5192    c
5193    60510   continue
5194            call dump_va( vstatus, msg )
5195            call exit_va( vstatus, msg )
5196    c
5197    60540   continue
5198            type *
5199            type *, char( 7 ), 'Error finding or accessing data file(s).'
5200            type 90030, char( 7 ) // 'Experiment #', mi_pkey, '.'
5201            type 90040, 'FORTRAN error code #', status, ' (hex).'
5202            type *, char( 7 ), 'File:  ', work_file
```

```
DO_MUSIC                                                    18-Apr-1988 14:00:27    VAX
                                                            18-Apr-1988 15:15:49    [CH
5203            type *
5204            type 90000, ' Mark ALL intermediate files for that experiment '
5205            1                                           // 'for DELETION (Y/[N])?  '
5206            accept 90010, answer
5207            psel = cap( answer( 1 : 1 ) )
5208    c
5209            if ( psel .eq. 'Y' ) then
5210              type 90000, ' ' // char( 7 ) // 'Are you sure (Y/[N])?  '
5211              accept 90010, answer
5212              psel = cap( answer( 1 : 1 ) )
5213              if ( psel .ne. 'Y' ) go to 60520
5214    c
5215    60530     continue
5216              read (unit=index_unit,key=mi_pkey,keyid=0,iostat=status)
5217              if      ( status .eq. FOR$IOS_SPERECLOC ) then
5218                unlock (unit=index_unit)
5219                type *, 'Index record locked by another program ...'
5220                call lib$wait( 10.0 )                   !  Wait 10 seconds.
5221                type *, 'Trying again.'
5222                go to 60530
5223              else if ( status .ne. 0 ) then
5224                unlock (unit=index_unit)
5225                go to 60700
5226              end if
5227              mi_stat = 'X'
5228              rewrite (unit=index_unit,iostat=status,err=60700)
5229            1                 mi_pkey, mi_stat, mi_park, mi_ierr, mi_eerr, mi_serr,
5230            2                 mi_dtim, mi_ants, mi_waves, mi_noises, mi_smp, mi_sint,
5231            3                 mi_begp, mi_endp, mi_delp, mi_peps, mi_norm, mi_wlen,
5232            4                 mi_tol, mi_which, mi_ofile, mi_anoises, mi_irlslim,
5233            5                 mi_zeigs, mi_doas, mi_estype, mi_parp
5234              unlock (unit=index_unit)
5235    c
5236              type *
5237              type 90030, char( 7 ) // 'Experiment #', mi_pkey, ' marked for '
5238            1                                                   // 'DELETEION!'
5239              type *
5240            else
5241    60520     continue
5242              type *
5243              type 90030, 'Index record for experiment #', mi_pkey, ' not changed.'
5244              type *
5245            end if
5246            go to err_return
5247    c
5248    c               Failed access to index record - not a lot we can do about it.
5249    c
5250    60700   continue
5251            type *
5252            type *, char( 7 ), 'Read or write failure accessing index record!'
5253            type 90030, char( 7 ) // 'Experiment #', mi_pkey, '.'
5254            type 90040, 'FORTRAN error code #', status, ' (hex).'
5255            type *, char( 7 ), 'Status of that index record is uncertain.'
5256            type *
5257            go to 20000
5258    c
5259    c*******************************************************************************
```

```
DO_MUSIC                                                    18-Apr-1988 14:00:27    VA
                                                            18-Apr-1988 15:15:49    DC

5260     c
5261     90000    format( $a )
5262     90010    format( a10 )
5263     90020    format( i10.10 )
5264     90030    format( 1x, a, i10.10, a )
5265     90040    format( 1x, a, z8.8, a )
5266     90050    format( $, 1x, 2( a, i6 ), a )
5267     c
5268              end


PROGRAM SECTIONS

    Name                                Bytes   Attributes

  0 $CODE                               25984   PIC CON REL LCL   SHR   EXE   RD NOWRT LC
  1 $PDATA                               5492   PIC CON REL LCL   SHR NOEXE   RD NOWRT LC
  2 $LOCAL                               6308   PIC CON REL LCL NOSHR NOEXE   RD   WRT QL
  3 MUSIC_MEASURES                        120   PIC OVR REL GBL   SHR NOEXE   RD   WRT LC

    Total Space Allocated               37904


ENTRY POINTS

    Address  Type  Name

  0-00000000       DO_MUSIC


VARIABLES

    Address  Type  Name              Address  Type  Name              Address  Type  Name

  2-00000388  I*4  ACORREL_ADD    2-00000318  L*4  ANCESTOR       2-00000380  I*4  ANDEF
  2-000001A0  CHAR ANSWER         2-00000358  I*4  ANT_ADD        2-0000019D  CHAR CHSEL
  2-0000036C  I*4  CORREL_ADD     2-000003A0  I*4  COUNT          3-0000000C  I*4  CYCLE
  2-00000354  I*4  DATA_UNIT2     2-00000308  L*4  DISP_INDEX     2-00000394  I*4  DOA_A
  2-00000390  I*4  EIGV_ADD       2-0000038C  I*4  EIG_ADD       2-000003C0  I*4  ELSEL
  2-000003D0  I*4  ESTYPE         2-00000398  I*4  EXNM_ADD      2-000003A4  I*4  FSKIP
  2-0000039C  I*4  I              2-00000344  I*4  INDEX_UNIT    2-000003A8  I*4  ISKIP
  2-000003F4  I*4  MI_ANOISES     2-000003E0  I*4  MI_ANTS       2-00000408  R*4  MI_BE
  2-00000400  I*4  MI_DOAS        2-00000273  CHAR MI_DTIM       2-000002D8  I*2  MI_EE
  2-00000404  I*4  MI_ESTYPE      2-000002D6  I*2  MI_IERR       2-000003F8  I*4  MI_IR
  2-000002F0  R*8  MI_NORM        2-00000286  CHAR MI_OFILE      2-000003DC  I*4  MI_PA
  2-000002E8  R*8  MI_PEPS        2-000003D8  I*4  MI_PKEY       2-000002DA  I*2  MI_SE
  2-000003EC  I*4  MI_SMP         2-00000272  CHAR MI_STAT       2-00000300  R*8  MI_TC
  2-000003F0  I*4  MI_WHICH       2-000002F8  R*8  MI_WLEN       2-000003FC  I*4  MI_ZE
  2-00000360  I*4  MTYPE_ADD      2-00000364  I*4  NDEF_ADD      2-00000374  I*4  NOISE
  3-00000068  I*4  NON_TRI_CYC    2-00000368  I*4  NPARS_ADD     2-000003B4  I*4  NUM_A
  2-000003B8  I*4  NUM_EXPS       2-000003AC  I*4  NUM_ZEIGS     2-00000370  I*4  OBSMA
  2-0000032C  I*4  OLD_ANOISES    2-00000320  I*4  OLD_ANTS      2-00000328  I*4  OLD_N
  2-00000324  I*4  OLD_WAVES      2-00000330  I*4  OLD_WHICH     2-000003D4  R*4  P
  2-0000033C  I*4  PNAME_LEN      2-000001AA  CHAR PROCNAME      2-0000019B  CHAR PSEL
  3-0000006C  R*8  QPSI_LMARG     3-00000074  I*4  QPSI_L_ZCYC   2-00000348  I*4  READ_
```

18-Apr-1988 14:00
13-Apr-1988 15:15

```
0001    c
0002            integer*4 function p_index( index_unit, read_unit1, read_unit2,
0003            1                                                   keyrange, show )
0004    c
0005    c=================================================================
0006    c
0007    c               This function finds and (optionally) displays records fro
0008    c       MUSIC experiments index file open on logical unit numbers index_u
0009    c       read_unit1, and read_unit2 (ALL LUNs attached to the SAME file).
0010    c       parameters index_unit, read_unit1, and read_unit2 are INTEGER*4,
0011    c       keyrange is CHARACTER*(*), and show is LOGICAL.
0012    c               If show is .true., the records to be displayed are select
0013    c       keyrange as follows:  If keyrange is '*', all records from the se
0014    c       MUSIC index file are displayed.  Otherwise, the records whose
0015    c       "statuses" (first alternate key field) match the characters in th
0016    c       string keyrange are displayed, in the order of the characters in
0017    c       keyrange.  For example, if keyrange is 'SR', all 'S' records are
0018    c       displayed, followed by all 'R' records.
0019    c               If show is .true., the function returns the number the op
0020    c       enters in response to a MUSIC_INDEX_DISPLAY routine prompt, or -1
0021    c       matching records exist (no records at all if keyrange = '*'), or
0022    c       one or more records are displayed but the operator makes no recor
0023    c       selection.
0024    c               If show is .false., the function simply tests for the pre
0025    c       of matching records (any records if keyrange = '*') in the select
0026    c       MUSIC index file.  If any matching records are found, the functio
0027    c       returns 0, otherwise it returns -1.
0028    c               The three LUNs index_unit, read_unit1, and read_unit2 are
0029    c       handled as follows:  If keyrange = '*', read_unit1 and read_unit2
0030    c       used to read the file in SECOND ALTERNATE / PRIMARY key HEIRARCHY
0031    c       order; otherwise read_unit1 is used to read the file in FIRST ALT
0032    c       key ASCENDING order.  The LUN index_unit is used by routine
0033    c       MUSIC_INDEX_DISPLAY (if show is .true.), to read the file in PRIM
0034    c       key order only.
0035    c
0036    c=================================================================
0037    c
0038            implicit none
0039            intrinsic len, index, char
0040            external next_rec, music_index_display
0041    c
0042            logical next_rec
0043            integer*4 music_index_display
0044    c
0045            logical show
0046            integer*4 index_unit, read_unit1, read_unit2
0047            character*(*) keyrange
0048    c
0049            logical notall, some_found, tlog
0050            integer*4 ikey, status, slen, chno, iskey
0051            character*1 ch
0052    c
0053            some_found = .false.                    !  No records found yet.
0054            tlog = next_rec( read_unit1, read_unit2, ikey, char( 0 ), iskey )
0055            notall = index( keyrange, '*' ) .le. 0  !  Not all statuses selec
0056            if ( notall ) then
0057              chno = 1
```

```
P_INDEX                                                  18-Apr-1988 14:00:27    VAX
                                                         13-Apr-1988 15:15:49    [CHU
0058              slen = len( keyrange )
0059            else
0060              ch = '*'
0061            end if
0062      c
0063      10000 continue
0064              if ( notall ) ch = keyrange( chno : chno )
0065      c
0066      10010   continue
0067              if ( .not. next_rec( read_unit1, read_unit2, ikey, ch, iskey ) )
0068             1                                                    go to 10500
0069      c
0070      10100   continue
0071              if ( show ) then
0072      10130     continue
0073                p_index    = music_index_display( index_unit, ikey, .true. )
0074                some_found = .true.
0075                if      ( p_index .eq. -1 ) then            !  Display not full.
0076      10110       continue
0077                  if ( .not. next_rec( read_unit1, read_unit2, ikey, ch, iskey ) )
0078             1                                                    go to 10200
0079                  go to 10130
0080                else if ( p_index .eq. -2 ) then            !  User wants next pg.
0081      10120       continue
0082                  if ( .not. next_rec( read_unit1, read_unit2, ikey, ch, iskey ) )
0083             1                                                    go to 10300
0084                  go to 10130
0085                else                  !  User entered a number (since <CR> on non-
0086      c                                  existent record is handled by status
0087      c                                  checks).
0088      c
0089                  tlog = next_rec( read_unit1, read_unit2, ikey, char( 0 ), iskey )
0090                  return
0091                end if
0092              else                    !  Find only, a matching record found.
0093                p_index = 0
0094                tlog = next_rec( read_unit1, read_unit2, ikey, char( 0 ), iskey )
0095                return
0096              end if
0097      c
0098      10200   continue
0099      c
0100      c               End of selected index records, ( maximum records currently
0101      c       displayed.
0102      c
0103                if ( notall .and. ( chno .lt. slen ) ) then
0104      c
0105      c               Multiple secondary key values specified, and values remain to
0106      c       be processed.
0107      c
0108                  chno = chno + 1
0109                  go to 10000
0110                end if
0111      c
0112      c               All secondary key values processed, or primary key search was
0113      c       performed, and display was selected.
0114      c
```

```
0115                p_index = music_index_display( index_unit, -1, .true. )
0116                if ( p_index .lt. 0 ) p_index = -2  !  Indicate operator cancelled.
0117                tlog = next_rec( read_unit1, read_unit2, ikey, char( 0 ), iskey )
0118                return
0119    c
0120    10300     continue
0121    c
0122    c                   End of selected index records, with maximum currently
0123    c           displayed, operator has solicited the next page, and display was
0124    c           selected.
0125    c
0126                    if ( notall .and. ( chno .lt. slen ) ) then
0127    c
0128    c                   Multiple secondary key values specified, and values remain to
0129    c           be processed.
0130    c
0131                      chno = chno + 1
0132                      go to 10000
0133                    end if
0134    c
0135    c                   All secondary key values processed, or primary key search was
0136    c           performed, and display was selected.
0137    c
0138                    p_index = -2                        !  Indicate operator cancelled.
0139                    tlog = next_rec( read_unit1, read_unit2, ikey, char( 0 ), iskey )
0140                    return
0141    c
0142    c                   Error on read of initial record with the current selected key.
0143    c
0144    10500     continue
0145                    if ( notall .and. ( chno .lt. slen ) ) then
0146    c
0147    c                   Multiple secondary key values specified, and values remain to
0148    c           be processed.
0149    c
0150                      chno = chno + 1
0151                      go to 10000
0152                    end if
0153    c
0154    c                   All secondary key values processed, or primary key search was
0155    c           performed.
0156    c
0157                    if ( some_found .and. show ) then
0158                      p_index = music_index_display( index_unit, -1, .true. )
0159                      if( p_index .lt. 0 ) p_index = -2 !  Indicate operator cancelled.
0160                    else
0161                      p_index = -1                      !  Indicate nothing available.
0162                    end if
0163                    tlog = next_rec( read_unit1, read_unit2, ikey, char( 0 ), iskey )
0164                    return
0165    c
0166            end
```

```
0001    c
0002            logical function next_rec( unit1, unit2, pkey, skey1, skey2 )
0003    c
0004    c=======================================================================
0005    c
0006    c                       This function determines the key values for the next MUSIC
0007    c       index record meeting a search criterion.  The MUSIC index file to be
0008    c       searched is assumed to be open on Logical Unit Numbers unit1 and unit2
0009    c       (BOTH LUNs open on the SAME file).
0010    c                       The search criterion to be used depends on the CHARACTER*1
0011    c       value skey1 on entry.  If skey1 is not '*', records whose first
0012    c       alternate key values ("statuses") are equal to skey1 are sought.  If
0013    c       skey1 = '*', records are sought in "heirarchical" order.  When
0014    c       "heirarchical" order is used, an "ancestor" record is returned first,
0015    c       followed by its "descendant" records, followed by the next "ancestor"
0016    c       record, and so on.  "Descendant" records are presented in a similar
0017    c       order; i.e. each record is followed directly by its "descendant"
0018    c       records, followed by the next record which is at the same level of
0019    c       "descendancy" as the first "descendant" record.
0020    c                       Since this routine is designed to be called repeatedly to
0021    c       perform a particular search, a reset MUST BE PERFORMED when the desired
0022    c       search has been completed.  A reset of this routine is accomplished by
0023    c       invoking the function with skey1 set to ASCII NULL (ASCII value of 0).
0024    c                       The parameters unit1, unit2, pkey, and skey2 are INTEGER*4,
0025    c       while skey1 is CHARACTER*1.  The primary key value of the next
0026    c       applicable record (if any) is returned in pkey, and its second
0027    c       alternate key value (its "parent" primary key value) is returned in
0028    c       skey2.  NOTE that the parameter skey1 is NOT written (value NOT
0029    c       changed).
0030    c                       The function return value is .true. if a matching record was
0031    c       found, and .false. if not.  It is also set to .true. when a reset is
0032    c       performed.
0033    c
0034    c=======================================================================
0035    c
0036            implicit none
0037            intrinsic ichar
0038            external lib$get_lun, lib$stop, lib$free_lun, lib$wait
0039            include 'SYS$LIBRARY:FORIOSDEF'
0125    c
0126            integer*4 lib$get_lun
0127    c
0128            integer*4 unit1, unit2, pkey, skey2
0129            character*1 skey1
0130    c
0131            integer*4 sort_unit, status, sort_open/ .false. /, stage/ -1 /
0132            integer*4 old_sk1/ -1 /, rd_ptr, ls_ptr, ins_ptr, shf_ptr
0133            integer*4 tpkey, tskey, parkey
0134            character*1 tch
0135            character*1  uparrow( 3 )/ 27, '[', 'A' /, spaces( 70 )/ 70*' ' /
0136    c
0137            save sort_unit, sort_open, stage, old_sk1, rd_ptr, ls_ptr, ins_ptr
0138    c
0139            if ( ichar( skey1 ) .eq. 0 ) then       !  Reset to be performed.
0140              if ( sort_open ) then
0141                close (unit=sort_unit)
0142                call lib$free_lun( sort_unit )
```

```
NEXT_REC                                                    18-Apr-1988 14:00:27    VAX FO
                                                            18-Apr-1988 15:15:49    [CHUCK

0143                sort_open = .false.
0144              end if
0145              old_sk1 = -1
0146              stage = -1
0147              next_rec = .true.
0148              return                                  !  Reset completed.
0149            end if
0150      c
0151            if ( stage .eq. -1 ) then
0152              if ( ( skey1 .eq. '*' ) .and. ( .not. sort_open ) ) then
0153                status = lib$get_lun( sort_unit )
0154                if (.not. status) call lib$stop( %val( status ) )
0155                open (unit=sort_unit,file='6000$RT:[CHUCK.PARAMS]SORT_FILE.DAT',
0156           1         form='UNFORMATTED',recordtype='FIXED',recl=2,
0157           2         organization='RELATIVE',access='DIRECT',status='UNKNOWN',
0158           3                                                 dispose='DELETE')
0159                sort_open = .true.
0160              end if
0161              stage = 0
0162            end if
0163      c
0164            if ( ( skey1 .ne. '*' ) .and. ( ichar( skey1 ) .ne. old_sk1 ) ) then
0165              old_sk1 = ichar( skey1 )
0166              stage = 0
0167            end if
0168      c
0169            if ( stage .eq. 0 ) then
0170      10000   continue
0171              if ( skey1 .eq. '*' ) then
0172                read (unit=unit1,keyeq=-1,keyid=2,iostat=status) pkey, tch, skey2
0173              else
0174                read (unit=unit1,keyeq=skey1,keyid=1,iostat=status)
0175           1                                                 pkey, tch, skey2
0176              end if
0177              unlock (unit=unit1)
0178              if      ( status .eq. FOR$IOS_SPERECLOC ) then        !  Record lock.
0179                type *, 'Record, status ''', skey1, ''' locked.  Waiting 1 sec.',
0180           1                                                                 uparrow
0181                call lib$wait( 1.0 )
0182                type *, spaces, uparrow
0183                go to 10000
0184              else if ( status .ne. 0 ) then                !  Error initial read.
0185                if ( ( skey1 .eq. '*' ) .and. ( sort_open ) ) then
0186                  close (unit=sort_unit)
0187                  call lib$free_lun( sort_unit )
0188                  sort_open = .false.
0189                  stage = -1
0190                end if
0191                next_rec = .false.
0192              else                                          !  Success first read.
0193                if ( skey1 .eq. '*' ) then
0194                  rd_ptr = 1
0195                  ls_ptr = 1
0196                  write (unit=sort_unit,rec=rd_ptr) pkey
0197                end if
0198                stage  = 1
0199                next_rec = .true.
```

```
NEXT_REC                                                    18-Apr-1988 14:00:27
                                                            13-Apr-1988 15:15:49

0200              end if
0201              return
0202            end if
0203      c
0204            if ( stage .eq. 2 ) then
0205              read (unit=sort_unit,rec=rd_ptr) parkey
0206      10100   continue                                          !  "Do while records".
0207                read (unit=unit2,iostat=status) pkey, tch, skey2
0208                unlock (unit=unit2)
0209                if      ( status .eq. FOR$IOS_SPERECLOC ) then      !  Record lock.
0210                  type *, 'Index record locked.  Waiting 1 sec.', uparrow
0211                  call lib$wait( 1.0 )
0212                  type *, spaces, uparrow
0213                  go to 10100
0214                else if ( ( status .eq. 0 ) .and. ( skey2 .eq. parkey ) ) then
0215                  shf_ptr = ls_ptr
0216                  ls_ptr  = ls_ptr + 1
0217                  do while ( shf_ptr .ge. ins_ptr )
0218                    read (unit=sort_unit,rec=shf_ptr) tpkey, tskey
0219                    write (unit=sort_unit,rec=shf_ptr+1) tpkey, tskey
0220                    shf_ptr = shf_ptr - 1
0221                  end do
0222                  write (unit=sort_unit,rec=ins_ptr) pkey, skey2
0223                  ins_ptr = ins_ptr + 1
0224                  go to 10100
0225                end if
0226              rd_ptr = rd_ptr + 1
0227              stage = 1
0228            end if
0229      c
0230            if ( stage .eq. 1 ) then
0231              if ( skey1 .eq. '*' ) read (unit=sort_unit,rec=rd_ptr) tpkey
0232      10200   continue
0233              if ( skey1 .eq. '*' ) then
0234                read (unit=unit2,keyeq=tpkey,keyid=2,iostat=status)
0235            1                                                 pkey, tch, skey2
0236                unlock (unit=unit2)
0237              else
0238                read (unit=unit1,iostat=status) pkey, tch, skey2
0239                unlock (unit=unit1)
0240              end if
0241              if      ( status .eq. FOR$IOS_SPERECLOC ) then      !  Record lock.
0242                type *, 'Index record locked.  Waiting 1 sec.', uparrow
0243                call lib$wait( 1.0 )
0244                type *, spaces, uparrow
0245                go to 10200
0246              else if ( status .eq. 0 ) then
0247                if ( skey1 .eq. '*' ) then
0248                  ins_ptr = rd_ptr + 1
0249                  shf_ptr = ls_ptr
0250                  ls_ptr  = ls_ptr + 1
0251                  do while ( shf_ptr .ge. ins_ptr )
0252                    read (unit=sort_unit,rec=shf_ptr) tpkey, tskey
0253                    write (unit=sort_unit,rec=shf_ptr+1) tpkey, tskey
0254                    shf_ptr = shf_ptr - 1
0255                  end do
0256                  write (unit=sort_unit,rec=ins_ptr) pkey
```

```
NEXT_REC                                                    18-Apr-1988 14:00:27     VA
                                                            13-Apr-1988 15:15:49     [C
0257                    ins_ptr = ins_ptr + 1
0258                    stage = 2
0259                    next_rec = .true.
0260                  else
0261                    next_rec = tch .eq. skey1
0262                  end if
0263                else
0264                  if ( skey1 .eq. '*' ) then
0265                    if ( rd_ptr .lt. ls_ptr ) then
0266                      rd_ptr = rd_ptr + 1
0267                      read (unit=sort_unit,rec=rd_ptr) pkey, skey2
0268                      next_rec = .true.
0269                    else
0270     10300            continue
0271                      read (unit=unit1,iostat=status) pkey, tch, skey2
0272                      unlock (unit=unit1)
0273                      if      ( status .eq. FOR$IOS_SPERECLOC ) then  !  Record lock.
0274                        type *, 'Index record locked.  Waiting 1 sec.', uparrow
0275                        call lib$wait( 1.0 )
0276                        type *, spaces, uparrow
0277                        go to 10300
0278                      else if ( ( status .eq. 0 ) .and. ( skey2 .eq. -1 ) ) then
0279                        rd_ptr = 1
0280                        ls_ptr = 1
0281                        write (unit=sort_unit,rec=rd_ptr) pkey
0282                        next_rec = .true.
0283                      else
0284                        stage = -1
0285                        next_rec = .false.
0286                      end if
0287                    end if
0288                  else
0289                    next_rec = .false.
0290                  end if
0291                end if
0292                return
0293              end if
0294     c
0295              end
```

```
                                                    18-Apr-1988 14:00:27     VA
                                                    18-Apr-1988 15:15:49     [C
0001    c
0002            subroutine submit_job( log_unit, prog_num )
0003    c
0004    c=======================================================================
0005    c
0006    c                This subroutine uses function SPAWN_IF to submit, if necessary,
0007    c       the batch job listed in the record whose primary key is prog_num in the
0008    c       index-organization file open on logical unit number log_unit.  The
0009    c       parameters log_unit and prog_num are INTEGER*4.
0010    c
0011    c=======================================================================
0012    c
0013            implicit none
0014            external lib$wait, spawn_if
0015            include '(system_symbols)'
0133            include '6000$RT:[CHUCK.RESEARCH.FORTDIR]SPAWNED_DEF.TXT'
0176            include 'SYS$LIBRARY:FORIOSDEF'
0262    c
0263            integer*4 spawn_if
0264    c
0265            integer*4 log_unit, prog_num
0266    c
0267            integer*4 status, ios
0268            character*1  uparrow( 3 )/ 27, '[', 'A' /, spaces( 70 )/ 70*' ' /
0269    c
0270            status = spawn_if( log_unit, prog_num )
0271            if ( status .eq. 0 ) return
0272    c
0273    10000   continue
0274            read (unit=log_unit,keyeq=prog_num,keyid=0,iostat=ios)
0275            1                                       sp_num, sp_stat, sp_file
0276            unlock (unit=log_unit)
0277            if      ( ios .eq. FOR$IOS_SPERECLOC ) then             !  Record lock.
0278              type *, 'Spawn record #', prog_num, ' locked.  Waiting 1 sec.',
0279            1                                                          uparrow
0280              call lib$wait( 1.0 )                                  !  Wait 1 sec.
0281              type *, spaces, uparrow
0282              go to 10000
0283            else if ( ios .ne. 0 ) then                             !  Error read.
0284              type *
0285              type *, 'Error reading spawned-processes index file, code ', ios, '.'
0286              type *, 'Couldn''t read record with primary key value ', prog_num,
0287            1                                                          '.'
0288              type *
0289              return
0290            end if
0291    c
0292            type *
0293            type *, 'Couldn''t execute file:'
0294            type *, sp_file
0295            type *
0296            if      ( status .eq. 1 ) then
0297              type *, 'Error reading spawned-processes index file.'
0298            else if ( status .eq. 2 ) then
0299              type *, 'Error spawning subprocess.'
0300            else if ( status .eq. 3 ) then
0301              type *, 'Error rewriting spawned-processes index file.'
```

```
SUBMIT_JOB                                                    18-Apr-1988 14:00:27     VAX F
                                                              18-Apr-1988 15:15:49     [CHU

0302              else
0303                type *, 'Impossible error, number ', status, '.'
0304              end if
0305     c
0306              return
0307              end


PROGRAM SECTIONS

    Name                                    Bytes   Attributes

  0 $CODE                                     600   PIC CON REL LCL   SHR   EXE   RD NOWRT LONG
  1 $PDATA                                    297   PIC CON REL LCL   SHR NOEXE   RD NOWRT LONG
  2 $LOCAL                                    312   PIC CON REL LCL NOSHR NOEXE   RD   WRT LONG

    Total Space Allocated                    1209


ENTRY POINTS

    Address  Type  Name

  0-00000000       SUBMIT_JOB


VARIABLES

    Address  Type  Name             Address  Type  Name             Address  Type  Name

      **     I*4   IOS           AP-00000004@ I*4   LOG_UNIT      AP-00000008@ I*4   PROG_NUM
  2-0000009C I*4   SP_NUM             **      CHAR  SP_PROC        2-000000A0  I*4   SP_STAT


ARRAYS

    Address  Type  Name               Bytes  Dimensions

  2-00000003  CHAR  SPACES              70  (70)
  2-00000000  CHAR  UPARROW              3  (3)


LABELS

    Address   Label

  0-00000028  10000
```

```
                                                    18-Apr-1988 14:00:27     V
                                                    13-Apr-1988 15:15:49     [

0001    c
0002            subroutine print_expm( exp_num )
0003    c
0004    c=======================================================================
0005    c
0006    c                This subroutine spawns a subprocess which prints the output
0007    c       file for the MUSIC experiment selected by the INTEGER*4 value exp_num.
0008    c
0009    c=======================================================================
0010    c
0011            implicit none
0012            external lib$find_file, lib$find_file_end, lib$spawn
0013    c
0014            integer*4 lib$find_file, lib$find_file_end, lib$spawn
0015    c
0016            integer*4 exp_num
0017    c
0018            integer*4 status, scratch
0019            character*10  expm_sel
0020            character*255 scr_str
0021    c
0022            scratch = 0
0023            write (unit=expm_sel,fmt=90000) exp_num
0024            status = lib$find_file( '6000$RT:[CHUCK.PARAMS]MUSIC_OUTPUT'
0025            1                       // expm_sel // '.DAT', scr_str, scratch )
0026            if ( status ) then
0027              status = lib$spawn( '$ print 6000$RT:[CHUCK.PARAMS]MUSIC_OUTPUT'
0028            1                                                 // expm_sel // '.DAT' )
0029              if ( .not. status ) go to 10000
0030            else
0031    10000     continue
0032              type *
0033              type *, char( 7 ), 'Can''t print an output file for experiment #',
0034            1                                                   expm_sel, '.'
0035              type *, '(Output file in use by another program?)'
0036              type *
0037            end if
0038    c
0039            if ( .not. lib$find_file_end( scratch ) ) then
0040              type *
0041              type *, char( 7 ), 'Error on call to LIB$FIND_FILE_END.'
0042              type *
0043            end if
0044    c
0045            return
0046    c
0047    90000   format( i10.10 )
0048    c
0049            end
```

18-Apr-1988 14:06:48
2-Apr-1988 14:48:17

```
0001            program sampl_music
0002      c
0003      c:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
0004      c
0005      c                 This program lurks in the SYS$SLOWBATCH queue (usually), and
0006      c       pounces on any record with a status of ' ' in MUSIC experiment index
0007      c       file 6000$RT:[CHUCK.PARAMS]MUSIC_INDEX.DAT.  When it finds such a
0008      c       record, it reads the information necessary to generate sample sets fror
0009      c       the appropriate data files, and generates samples for the experiment
0010      c       specified in the selected index file record.  The program then writes
0011      c       its results to output files (selected by the experiment number), and
0012      c       updates the MUSIC experiment index record to a status of 'I'.  The
0013      c       above sequence is repeated until no more ' ' status records are found
0014      c       AND no process named "Chuck MUSIC Drv" can be found on the VAX.
0015      c                 The above description applies exactly to error-free operation
0016      c       only.  If a computational error occurs during sample generation, or a
0017      c       file error occurs during writing of the solutions, the program changes
0018      c       the index record's status to 'X' instead of 'I'.  If a file error
0019      c       occurs during reading of the input data, the program increments the
0020      c       value mi_ierr (fourth field in the index file record). If the mi_ierr
0021      c       value then exceeds 10, the index record's status is changed to 'X'
0022      c       rather than 'I'.  If the mi_ierr value does not exceed 10, the index
0023      c       record's status is changed to '-' rather than 'I' (and the index
0024      c       record's mi_ierr value is updated).
0025      c                 NOTE that, since only "ancestor" experiments are ever given the
0026      c       status ' ', only "ancestor" experiments will ever be processed by this
0027      c       program.
0028      c
0029      c:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
0030      c
0031            implicit none
0032            intrinsic dsqrt, dble
0033            external lib$get_lun, lib$stop, lib$wait, lib$delete_file
0034            external init_va, open_va, exit_va, dump_va
0035            external readarr, writearr
0036            external music_init, sigmat_form, music_inp_rep, sigvec, obsvec
0037            external c16_vec_ins, c16_col_ins, dc_matcopy, c16_mat_dr_leftmul
0038            external sqdc_matclr, dc_diad, sqdc_matadd, c16_sel_mdiff
0039            external c16_vec_normsq, sqdc_mat_intdiv, dc_mathermmpy
0040            external c16_mat_dr_leftdiv, music_out_rep, dc_matconjg, va_dc_qr
0041            external endspawn_if, spawn_if
0042      c
0043            real*8 c16_vec_normsq
0044      c
0045            include '6000$RT:[CHUCK.RESEARCH.FORTDIR]VATYPES.TXT'
0057            include '6000$RT:[CHUCK.RESEARCH.FORTDIR]MUSIC_MEASURES.FOR'
0146            include '6000$RT:[CHUCK.RESEARCH.FORTDIR]MUSIC_INDEX_DEF.TXT'
0287            include '6000$RT:[CHUCK.RESEARCH.FORTDIR]SPAWNED_DEF.TXT'
0330            include 'SYS$LIBRARY:FORIOSDEF'
0416      c
0417            logical readarr, writearr, endspawn_if
0418            integer*4 lib$get_lun, spawn_if
0419            integer*4 my_prog, next_prog
0420            parameter (my_prog = 1)
0421            parameter (next_prog = 2)
0422      c
0423            integer*4 status, index_unit, data_unit, spawn_unit
```

```
SAMPL_MUSIC                                                    18-Apr-1988 14:06:48     VAX
                                                               2-Apr-1988 14:48:17      [CH
0424            integer*4 waited, wait_lim/ 4 /, vstatus
0425            integer*4 ants_by_2( 2 ), waves_by_9( 2 ), waves_by_2( 2 )
0426            integer*4 nois_by_2( 2 ), nois_by_3( 2 ), ants_by_nois( 2 )
0427            integer*4 waves_lin( 1 ), ants_by_wvs( 2 ), ants_lin( 1 )
0428            integer*4 smp_by_ants( 2 ), wavs_by_smp( 2 ), ants_by_smp( 2 )
0429            integer*4 anois_by_2( 2 ), anois_by_3( 2 ), ants_by_anois( 2 )
0430            integer*4 ants_sqr( 2 ), robs_add, rb_add, ants_di_add, rnoise_add
0431            integer*4 arnoise_add, anoise_add, obst_add, noiset_add
0432            integer*4 ant_add, wave_add, mtype_add, ndef_add, npars_add, correl_add
0433            integer*4 ccamp_add, sgmt_add, sspc_add, cln_add, obs_add, noise_add
0434            integer*4 obsmat_add, noisemat_add, sigvm_add, clnmat_add
0435            integer*4 andef_add, anpars_add, acorrel_add
0436            integer*4 i
0437            real*8 sum_normsq, lambda_min
0438            character*25 work_file
0439            character*80 msg
0440    c
0441    c               Get logical unit numbers
0442    c
0443            status = lib$get_lun( index_unit )
0444            if (.not. status) call lib$stop( %val( status ) )
0445            status = lib$get_lun( data_unit )
0446            if (.not. status) call lib$stop( %val( status ) )
0447            status = lib$get_lun( spawn_unit )
0448            if (.not. status) call lib$stop( %val( status ) )
0449    c
0450    c               Open the MUSIC index file.
0451    c
0452    10000   continue
0453            open (unit=index_unit,file='6000$RT:[CHUCK.PARAMS]MUSIC_INDEX.DAT',
0454            1       form='UNFORMATTED',recordtype='FIXED',recl=64,
0455            2       organization='INDEXED',access='KEYED',status='OLD',
0456            3       key=(1:4:INTEGER,5:5:CHARACTER),dispose='KEEP',SHARED,
0457            4                                                       err=10100)
0458            go to 10110
0459    c
0460    c               Error opening the MUSIC index file or MUSIC spawned-processes
0461    c       index file; abandon hope.
0462    c
0463    10100     continue
0464              type *
0465              type *, 'Error opening MUSIC index file or spawned-processes file.'
0466              go to 60620
0467    c
0468    c               MUSIC index file successfully opened.  Open MUSIC spawned-
0469    c       processes index file.
0470    c
0471    10110   continue
0472            open (unit=spawn_unit,file='6000$RT:[CHUCK.PARAMS]SPAWNED.IDX',
0473            1       form='UNFORMATTED',recordtype='FIXED',recl=26,
0474            2       organization='INDEXED',access='KEYED',status='OLD',
0475            3       key=(1:4:INTEGER,5:8:INTEGER),dispose='KEEP',SHARED,err=10100)
0476    c
0477    c               MUSIC index file and MUSIC spawned-processes index file
0478    c       successfully opened.  Start lurking, waiting to pounce on unsuspecting
0479    c       status ' ' records.
0480    c
```

```
0481            waited = 0
0482    10120   continue
0483              read (unit=index_unit,keyeq=' ',keyid=1,iostat=status,err=20100)
0484            1               mi_pkey, mi_stat, mi_park, mi_ierr, mi_eerr, mi_serr,
0485            2               mi_dtim, mi_ants, mi_waves, mi_noises, mi_smp, mi_sint,
0486            3               mi_begp, mi_endp, mi_delp, mi_peps, mi_norm, mi_wlen,
0487            4               mi_tol, mi_which, mi_ofile, mi_anoises, mi_irlslim,
0488            5               mi_zeigs, mi_doas, mi_estype, mi_parp
0489              unlock (unit=index_unit)
0490              waited = 0
0491              go to 20110
0492    c
0493    c               Error reading status ' ' records.  Could be that the first such
0494    c       record is locked, no such records exist, or some other error.
0495    c
0496    20100     continue
0497                unlock (unit=index_unit)
0498                if      ( status .eq. FOR$IOS_SPERECLOC ) then      !  Locked.
0499                  type *, 'Record locked, status '' ''.  Waiting 5 min.'
0500                  waited = 0
0501    20120         continue                              !  Branch to wait then read.
0502                  call lib$wait( 300.0 )                !  Hibernate for 5 minutes.
0503                  go to 10120
0504                else if ( status .eq. FOR$IOS_ATTACCNON ) then      ! No record.
0505                  go to 70000
0506                else                                              ! Some other error.
0507                  type *
0508                  type *, 'Failed reading status '' '' index records.'
0509                  call lib$stop( %val( status ) )
0510                end if
0511    c
0512    c               Successfully read a status ' ' record into the batch of
0513    c       variables listed in the read statement above.  Commence sample
0514    c       generation.
0515    c               Start by beating the Virtual Array system (copyright 1987 by
0516    c       Dwight Day) to death.
0517    c
0518    20110     continue
0519    c
0520              call init_va
0521              ants_by_2(1)    = mi_ants
0522              ants_by_2(2)    = 2
0523              waves_by_9(1)   = mi_waves
0524              waves_by_9(2)   = 9
0525              waves_by_2(1)   = mi_waves
0526              waves_by_2(2)   = 2
0527              waves_lin(1)    = mi_waves
0528              ants_by_wvs(1)  = mi_ants
0529              ants_by_wvs(2)  = mi_waves
0530              ants_lin(1)     = mi_ants
0531              ants_sqr(1)     = mi_ants
0532              ants_sqr(2)     = mi_ants
0533              smp_by_ants(1)  = mi_smp
0534              smp_by_ants(2)  = mi_ants
0535              wavs_by_smp(1)  = mi_waves
0536              wavs_by_smp(2)  = mi_smp
0537              ants_by_smp(1)  = mi_ants
```

```
0538              ants_by_smp(2)  = mi_smp
0539              nois_by_2(1)    = mi_noises
0540              nois_by_2(2)    = 2
0541              nois_by_3(1)    = mi_noises
0542              nois_by_3(2)    = 3
0543              ants_by_nois(1) = mi_ants
0544              ants_by_nois(2) = mi_noises
0545              if ( mi_which .gt. 0 ) then
0546                anois_by_2(1)    = mi_anoises
0547                anois_by_2(2)    = 2
0548                anois_by_3(1)    = mi_anoises
0549                anois_by_3(2)    = 3
0550                ants_by_anois(1) = mi_ants
0551                ants_by_anois(2) = mi_anoises
0552              end if
0553     c
0554              call open_va( ants_di_add, ants_sqr, 2, dc_type, vstatus, msg )
0555              call exit_va( vstatus, msg )
0556              call open_va( ant_add, ants_by_2, 2, dr_type, vstatus, msg )
0557              call exit_va( vstatus, msg )
0558              call open_va( wave_add, waves_by_9, 2, dr_type, vstatus, msg )
0559              call exit_va( vstatus, msg )
0560              call open_va( mtype_add, waves_by_2, 2, di_type, vstatus, msg )
0561              call exit_va( vstatus, msg )
0562              call open_va( ndef_add, nois_by_2, 2, di_type, vstatus, msg )
0563              call exit_va( vstatus, msg )
0564              call open_va( npars_add, nois_by_3, 2, sr_type, vstatus, msg )
0565              call exit_va( vstatus, msg )
0566              call open_va( correl_add, ants_by_nois, 2, dr_type, vstatus, msg )
0567              call exit_va( vstatus, msg )
0568              if ( mi_which .gt. 0 ) then
0569                call open_va( andef_add, anois_by_2, 2, di_type, vstatus, msg )
0570                call exit_va( vstatus, msg )
0571                call open_va( anpars_add, anois_by_3, 2, sr_type, vstatus, msg )
0572                call exit_va( vstatus, msg )
0573                call open_va( acorrel_add, ants_by_anois, 2,
0574             1                                          dr_type, vstatus, msg )
0575                call exit_va( vstatus, msg )
0576                call open_va( anoise_add, ants_lin, 1, dc_type, vstatus, msg )
0577                call exit_va( vstatus, msg )
0578                call open_va( arnoise_add, ants_sqr, 2, dc_type, vstatus, msg )
0579                call exit_va( vstatus, msg )
0580              end if
0581              call open_va( ccamp_add, waves_lin, 1, dc_type, vstatus, msg )
0582              call exit_va( vstatus, msg )
0583              call open_va( sgmt_add, ants_by_wvs, 2, dc_type, vstatus, msg )
0584              call exit_va( vstatus, msg )
0585              call open_va( sspc_add, waves_lin, 1, dc_type, vstatus, msg )
0586              call exit_va( vstatus, msg )
0587              call open_va( cln_add, ants_lin, 1, dc_type, vstatus, msg )
0588              call exit_va( vstatus, msg )
0589              call open_va( obs_add, ants_lin, 1, dc_type, vstatus, msg )
0590              call exit_va( vstatus, msg )
0591              call open_va( robs_add, ants_sqr, 2, dc_type, vstatus, msg )
0592              call exit_va( vstatus, msg )
0593              call open_va( obst_add, ants_sqr, 2, dc_type, vstatus, msg )
0594              call exit_va( vstatus, msg )
```

```
SAMPL_MUSIC                                                    18-Apr-1988 14:06:48
                                                               2-Apr-1988 14:48:17

0595              call open_va( noise_add, ants_lin, 1, dc_type, vstatus, msg )
0596              call exit_va( vstatus, msg )
0597              call open_va( rnoise_add, ants_sqr, 2, dc_type, vstatus, msg )
0598              call exit_va( vstatus, msg )
0599              call open_va( rb_add, ants_sqr, 2, dc_type, vstatus, msg )
0600              call exit_va( vstatus, msg )
0601              call open_va( noiset_add, ants_sqr, 2, dc_type, vstatus, msg )
0602              call exit_va( vstatus, msg )
0603              call open_va( obsmat_add, smp_by_ants, 2, dc_type, vstatus, msg )
0604              call exit_va( vstatus, msg )
0605              call open_va( noisemat_add, smp_by_ants, 2, dc_type, vstatus, msg )
0606              call exit_va( vstatus, msg )
0607              call open_va( sigvm_add, wavs_by_smp, 2, dc_type, vstatus, msg )
0608              call exit_va( vstatus, msg )
0609              call open_va( clnmat_add, ants_by_smp, 2, dc_type, vstatus, msg )
0610              call exit_va( vstatus, msg )
0611     c
0612     c                 Get problem definition from input file, initialize the noise
0613     c        counters, and form the array phase shift matrix.
0614     c
0615              work_file = 'MUSIC_INPUT__________.DAT'
0616              write (unit=work_file( 12 : 21 ),fmt=90020,err=60510) mi_pkey
0617     c
0618              open (unit=data_unit,file='6000$RT:[CHUCK.PARAMS]'//work_file,
0619             1       status='OLD',organization='SEQUENTIAL',access='SEQUENTIAL',
0620             2       form='UNFORMATTED',recordtype='FIXED',recl=64,dispose='KEEP',
0621             3                                                  READONLY,err=60510)
0622              read (unit=data_unit,err=60500)
0623              if ( .not. (
0624             1       readarr( %val( ant_add ), dr_type, mi_ants, 2, data_unit, 32 )
0625             2                                                          ) ) go to 60500
0626              if ( .not. (
0627             1       readarr( %val( wave_add ), dr_type, mi_waves, 9, data_unit,
0628             2                                                              32 )
0629             3                                                          ) ) go to 60500
0630              if ( .not. (
0631             1       readarr( %val( mtype_add ), di_type, mi_waves, 2, data_unit,
0632             2                                                              64 )
0633             3                                                          ) ) go to 60500
0634              if ( .not. (
0635             1       readarr( %val( ndef_add ), di_type, mi_noises, 2, data_unit,
0636             2                                                              64 )
0637             3                                                          ) ) go to 60500
0638              if ( .not. (
0639             1       readarr( %val( npars_add ), sr_type, mi_noises, 3, data_unit,
0640             2                                                              64 )
0641             3                                                          ) ) go to 60500
0642              if ( .not. (
0643             1       readarr( %val( correl_add ), dr_type, mi_ants, mi_noises,
0644             2                                                  data_unit, 32 )
0645             3                                                          ) ) go to 60500
0646              if ( mi_which .gt. 0 ) then
0647                if ( .not. (
0648             1       readarr( %val( andef_add ), di_type, mi_anoises, 2,
0649             2                                                  data_unit, 64 )
0650             3                                                          ) ) go to 60500
0651                if ( .not. (
```

```
SAMPL_MUSIC                                                    18-Apr-1988 14:06:48
                                                               2-Apr-1988 14:48:17

0652            1       readarr( %val( anpars_add ), sr_type, mi_anoises, 3,
0653            2                                                 data_unit, 64 )
0654            3                                                         ) ) go to 60500
0655                  if ( .not. (
0656            1       readarr( %val( acorrel_add ), dr_type, mi_ants, mi_anoises,
0657            2                                                 data_unit, 32 )
0658            3                                                         ) ) go to 60500
0659                  end if
0660                  close (unit=data_unit)
0661    c
0662                  call music_init( mi_waves, %val( wave_add ), %val( ccamp_add ),
0663            1                     mi_noises, %val( ndef_add ), %val( npars_add ),
0664            2                     mi_anoises, %val( andef_add ), %val( anpars_add ),
0665            3                                                       mi_which )
0666                  call sigmat_form( %val( ant_add ), mi_ants,
0667            1                           %val( wave_add ), mi_waves, %val( sgmt_add ) )
0668    c
0669    c             Print out input parameters (converted as appropriate).
0670    c
0671                  call music_inp_rep( data_unit, mi_pkey, mi_ofile, mi_dtim,
0672            1       %val( ant_add ), mi_ants, %val( wave_add ), %val( mtype_add ),
0673            2       mi_waves, %val( ndef_add ), %val( npars_add ),
0674            3       %val( correl_add ), mi_noises, mi_which, %val( andef_add ),
0675            4       %val( anpars_add ), %val( acorrel_add ), mi_anoises, mi_smp,
0676            5       mi_sint, mi_begp, mi_endp, mi_delp, mi_peps, mi_norm, mi_tol,
0677            6                                                       mi_irlslim )
0678    c
0679    c             Collect noise and observation sample vectors into their
0680    c       respective matrices, and accumulate the actual noise covariance
0681    c       matrix if necessary.
0682    c
0683                  sum_normsq = 0.0d0
0684                  if ( mi_which .gt. 0 )
0685            1                   call sqdc_matclr( %val( arnoise_add ), mi_ants )
0686    c
0687                  do i = 1, mi_smp
0688                    call sigvec( mi_waves, %val( ccamp_add ), %val( wave_add ),
0689            1                   %val( mtype_add ), i, mi_sint, %val( sspc_add ) )
0690                    call obsvec( %val( sgmt_add ), mi_ants, mi_waves,
0691            1                   %val( ndef_add ), %val( npars_add ),
0692            2                   %val( correl_add ), mi_noises, %val( andef_add ),
0693            3                   %val( anpars_add ), %val( acorrel_add ), mi_anoises,
0694            4                   %val( sspc_add ), %val( cln_add ), %val( obs_add ),
0695            5                                     %val( noise_add ), mi_which )
0696                    call c16_vec_ins( %val( obs_add), mi_ants, %val( obsmat_add ),
0697            1                                                  mi_smp, mi_ants, i )
0698                    call c16_vec_ins( %val( noise_add ), mi_ants,
0699            1                         %val( noisemat_add ), mi_smp, mi_ants, i )
0700                    call c16_col_ins( %val( sspc_add ), mi_waves, %val( sigvm_add ),
0701            1                                                  mi_waves, mi_smp, i )
0702                    call c16_col_ins( %val( cln_add ), mi_ants, %val( clnmat_add ),
0703            1                                                  mi_ants, mi_smp, i )
0704    c
0705                    if ( mi_which .gt. 0 ) then
0706                      call c16_sel_mdiff( %val( obs_add ), 1, 1, %val( cln_add ),
0707            1                                    1, 1, mi_ants, %val( anoise_add ) )
0708                      call dc_diad( %val( anoise_add ), mi_ants,
```

```
SAMPL_MUSIC                                                    18-Apr-1988 14:06:48
                                                               2-Apr-1988 14:48:17

0709            1                                                  %val( ants_di_add ) )
0710                  call sqdc_matadd( %val( arnoise_add ), %val( ants_di_add ),
0711            1                                                     mi_ants )
0712                  end if
0713    c
0714                  sum_normsq = sum_normsq
0715            1                          + c16_vec_normsq( %val( noise_add ), mi_ants
0716                end do
0717    c
0718    c                   Write the generated vector samples to the "samples" file for
0719    c             this experiment.
0720    c
0721                  work_file( 7 : 11 ) = 'SAMPL'
0722                  call lib$delete_file( '6000$RT:[CHUCK.PARAMS]' // work_file // ';*'
0723                  open (unit=data_unit,file='6000$RT:[CHUCK.PARAMS]'//work_file,
0724            1           status='NEW',organization='SEQUENTIAL',access='SEQUENTIAL',
0725            2           form='UNFORMATTED',recordtype='FIXED',recl=64,dispose='KEEP',
0726            3                                                     err=60530)
0727                  if ( .not. (
0728            1           writearr( %val( obsmat_add ), dc_type, mi_smp, mi_ants,
0729            2                                                     data_unit, 1€
0730            3                                                     ) ) go to 60!
0731                  if ( .not. (
0732            1           writearr( %val( noisemat_add ), dc_type, mi_smp, mi_ants,
0733            2                                                     data_unit, 1€
0734            3                                                     ) ) go to 60!
0735                  if ( .not. (
0736            1           writearr( %val( sigvm_add ), dc_type, mi_waves, mi_smp,
0737            2                                                     data_unit, 1€
0738            3                                                     ) ) go to 60!
0739                  if ( .not. (
0740            1           writearr( %val( cinmat_add ), dc_type, mi_ants, mi_smp,
0741            2                                                     data_unit, 1€
0742            3                                                     ) ) go to 60!
0743                  close (unit=data_unit)
0744    c
0745    c                   Compute minimum generalized eigenvalue estimate and covarianc
0746    c             matrices.
0747    c
0748                  lambda_min = sum_normsq / ( mi_ants * mi_smp )
0749                  if ( mi_which .gt. 0 )
0750            1           call sqdc_mat_intdiv( %val( arnoise_add ), mi_ants, mi_smp )
0751    c
0752    c                   Form normalized complex conjugates of observation and noise
0753    c             sample matrices, then perform QR decomposition of normalized complex
0754    c             conjugates.
0755    c
0756                  call dc_matconjg( %val( obsmat_add ), mi_smp, mi_ants )
0757                  call c16_mat_dr_leftdiv( %val( obsmat_add ), mi_smp, mi_ants,
0758            1                                    dsqrt( dble( mi_smp ) ), mi_ants )
0759                  call va_dc_qr( %val( obsmat_add ), mi_smp, mi_ants,
0760            1                                                  %val( obst_add ) )
0761    c
0762                  call dc_matconjg( %val( noisemat_add ), mi_smp, mi_ants )
0763                  call c16_mat_dr_leftdiv( %val( noisemat_add ), mi_smp, mi_ants,
0764            1                              dsqrt( sum_normsq / mi_ants ), mi_ants )
0765                  call va_dc_qr( %val( noisemat_add ), mi_smp, mi_ants,
```

```
SAMPL_MUSIC                                                    18-Apr-1988 14:06:48
                                                               2-Apr-1988 14:48:17

0766            1                                                %val( noiset_add ) :
0767      c
0768      c                 Compute observation, normalized (model) noise, and (model)
0769      c         noise covariance matrices.
0770      c
0771                call dc_mathermmpy( %val( obst_add ), mi_ants, mi_ants,
0772            1                       %val( obst_add ), mi_ants, %val( robs_add )
0773                call dc_mathermmpy( %val( noiset_add ), mi_ants, mi_ants,
0774            1                       %val( noiset_add ), mi_ants, %val( rb_add )
0775                call dc_matcopy( %val( rb_add ), mi_ants, mi_ants,
0776            1                                                %val( rnoise_add )
0777                call c16_mat_dr_leftmul( %val( rnoise_add ), mi_ants, mi_ants,
0778            1                                                lambda_min, mi_ants
0779      c
0780      c                 Print out accumulated observation and noise covariance
0781      c         matrices.
0782      c
0783                call music_out_rep( data_unit, mi_pkey, mi_park, mi_parp, -2,
0784            1                       mi_which, mi_ofile, %val( robs_add ),
0785            2                       %val( rnoise_add ), %val( arnoise_add ),
0786            3                                  mi_ants, lambda_min, mi_dtir
0787      c
0788      c                 Create the experiment's "eigenvectors" file, and write
0789      c         covariance matrices and triangular matrices from QR decompositions
0790      c         it.
0791      c
0792                work_file( 7 : 11 ) = 'EIGVS'
0793                call lib$delete_file( '6000$RT:[CHUCK.PARAMS]' // work_file // ';'
0794                open (unit=data_unit,file='6000$RT:[CHUCK.PARAMS]'//work_file,
0795            1         status='NEW',organization='SEQUENTIAL',access='SEQUENTIAL',
0796            2         form='UNFORMATTED',recordtype='FIXED',recl=64,dispose='KEEP'
0797            3                                                  err=60530)
0798                if ( .not. ( writearr( %val( robs_add ), dc_type, mi_ants, mi_ants
0799            1                                          data_unit, 16 ) ) ) go to 60
0800                if ( .not. ( writearr( %val( rnoise_add ), dc_type, mi_ants,
0801            1                       mi_ants, data_unit, 16 ) ) ) go to 60520
0802                if ( mi_which .gt. 0 ) then
0803                  if ( .not. ( writearr( %val( arnoise_add ), dc_type, mi_ants,
0804            1                       mi_ants, data_unit, 16 ) ) ) go to 60520
0805                end if
0806                if ( .not. ( writearr( %val( rb_add ), dc_type, mi_ants, mi_ants,
0807            1                                          data_unit, 16 ) ) ) go to 60
0808                if ( .not. ( writearr( %val( obst_add ), dc_type, mi_ants, mi_ants
0809            1                                          data_unit, 16 ) ) ) go to 60
0810                if ( .not. ( writearr( %val( noiset_add ), dc_type, mi_ants,
0811            1                       mi_ants, data_unit, 16 ) ) ) go to 60520
0812                close (unit=data_unit)
0813      c
0814      c                 Update index file entry to reflect creation of the "samples"
0815      c         and "eigenvectors" files, discard memory used for Virtual Arrays, ar
0816      c         exit if there is no more work to do.
0817      c
0818                call dump_va( vstatus, msg )
0819                call exit_va( vstatus, msg )
0820                mi_stat = '1'
0821      c
0822                status = spawn_if( spawn_unit, next_prog )     !  Start eigenvalues
```

```
SAMPL_MUSIC                                                    18-Apr-1988 14:06:48
                                                               2-Apr-1988 14:48:17

0823              if ( status .ne. 0 ) then                    !  Spawn failed.
0824                type *
0825                type *, 'Error initiating eigenvalue solver.'
0826                if      ( status .eq. 1 ) then
0827                  type *, 'Read error on spawned-processes index file.'
0828                else if ( status .eq. 2 ) then
0829                  type *, 'Error spawning subprocess.'
0830                else if ( status .eq. 3 ) then
0831                  type *, 'Error rewriting spawned-processes index file.'
0832                else
0833                  type *, 'Impossible error, code ', status, '.'
0834                end if
0835    60620       continue                                     !  Error exit.
0836                type *, 'Sample generation abandoned.'
0837                type *
0838                stop
0839              end if
0840    c
0841    60600     continue
0842              read (unit=index_unit,key=mi_pkey,keyid=0,iostat=status)
0843              if      ( status .eq. FOR$IOS_SPERECLOC ) then
0844                type *, 'Record to rewrite locked, #', mi_pkey,
0845             1                                               '.  Waiting 10 sec.'
0846                unlock (unit=index_unit)
0847                call lib$wait( 10.0 )                        !  Wait 10 seconds.
0848                go to 60600
0849              else if ( status .ne. 0 ) then                 !  Error reading index.
0850    60610       continue                              !  Index access error branch.
0851                unlock (unit=index_unit)
0852                type *
0853                type *, 'Error accessing MUSIC experiment index, code ', status
0854                go to 60620
0855              end if
0856              rewrite (unit=index_unit,iostat=status,err=60610)
0857             1                mi_pkey, mi_stat, mi_park, mi_ierr, mi_eerr, mi_serr,
0858             2                mi_dtim, mi_ants, mi_waves, mi_noises, mi_smp, mi_sint,
0859             3                mi_begp, mi_endp, mi_delp, mi_peps, mi_norm, mi_wlen,
0860             4                mi_tol, mi_which, mi_ofile, mi_anoises, mi_irlslim,
0861             5                mi_zeigs, mi_doas, mi_estype, mi_parp
0862              unlock (unit=index_unit)
0863             go to 10120
0864    c
0865    c                 No more status ' ' records available.
0866    c
0867    70000   continue
0868             if ( waited .ge. wait_lim ) then             !  Already waited 20 minutes.
0869               waited = 0
0870               if ( endspawn_if( spawn_unit, my_prog ) ) stop
0871             else
0872               waited = waited + 1
0873               call lib$wait( 300.0 )                     !  Wait 5 minutes & try again.
0874             end if
0875             go to 10120
0876    c
0877    c*******************************************************************************
0878    c
0879    c                 File error processing branches.
```

```
0880    c
0881    c*******************************************************************************
0882    c
0883    c               Computation error, or error opening or writing samples output
0884    c       or eigenvectors output file.
0885    c
0886    60520   continue
0887            close (unit=data_unit)
0888    c
0889    60530   continue
0890            call dump_va( vstatus, msg )
0891            call exit_va( vstatus, msg )
0892    c
0893    60540   continue
0894            mi_stat = 'X'
0895            go to 60600
0896    c
0897    c               Error reading input data file.
0898    c
0899    60500   continue
0900            close (unit=data_unit)
0901    c
0902    60510   continue
0903            call dump_va( vstatus, msg )
0904            call exit_va( vstatus, msg )
0905    c
0906            if ( mi_ierr .ge. 10 ) then
0907              mi_stat = 'X'
0908            else
0909              mi_ierr = mi_ierr + 1
0910              mi_stat = '-'
0911            end if
0912            go to 60600
0913    c
0914    90020   format( i10.10 )
0915    c
0916            end
```

18-Apr-1988 14:09:24
2-Apr-1988 14:41:20

```
0001            program eig_music
0002    c
0003    c:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
0004    c
0005    c                   This program lurks in the SYS$SLOWBATCH queue (usually), and
0006    c           pounces on any record with a status of 'I' in MUSIC experiment index
0007    c           file 6000$RT:[CHUCK.PARAMS]MUSIC_INDEX.DAT.  When it finds such a
0008    c           record, it reads the information necessary to solve the generalized
0009    c           eigenvalue problem from the appropriate data files, and proceeds to
0010    c           solve the problem for the experiment specified in the selected index
0011    c           file record.  The program then writes its results to an eigenvalue dat
0012    c           output file (selected by the experiment number), and updates the MUSI(
0013    c           experiment index record to a status of 'E'.  The above sequence is
0014    c           repeated until no more status 'I' records are found AND no process
0015    c           named "Chuck MUSIC Drv" can be found on the VAX.
0016    c                   The above description applies exactly to error-free operation
0017    c           only.  If a computational error occurs during eigenvalue problem
0018    c           solution, or a file error occurs during writing of the solutions, the
0019    c           program changes the index record's status to 'X' instead of 'E'.  If a
0020    c           file error occurs during reading of the input data, the program
0021    c           increments the value mi_eerr (fifth field in the index file record).
0022    c           If the mi_eerr value then exceeds 10, the index record's status is
0023    c           changed to 'X' rather than 'E'.  If the mi_eerr value does not exceed
0024    c           10, the index record's status is changed to ' ' (for "ancestor"
0025    c           experiments) or 'D' (for "descendant" experiments) rather than 'E' (ar
0026    c           the index record's mi_eerr value is updated).  Finally, the program
0027    c           changes the index record's status to 'i' if a data file can't be opene
0028    c           (the assumption being that the file is in use by another program).
0029    c           Status 'i' records are processed after all status 'I' record operatior
0030    c           have been completed.
0031    c
0032    c:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
0033    c
0034            implicit none
0035            external lib$get_lun, lib$stop, lib$wait
0036            external open_va, exit_va, dump_va
0037            external readarr, writearr, skiparr
0038            external sq_vlgsvd, sort_eig, music_eig_rep, vl_check
0039            external endspawn_if
0040    c
0041            include '6000$RT:[CHUCK.RESEARCH.FORTDIR]VATYPES.TXT'
0053            include '6000$RT:[CHUCK.RESEARCH.FORTDIR]MUSIC_MEASURES.FOR'
0142            include '6000$RT:[CHUCK.RESEARCH.FORTDIR]MUSIC_INDEX_DEF.TXT'
0283            include '6000$RT:[CHUCK.RESEARCH.FORTDIR]SPAWNED_DEF.TXT'
0326            include 'SYS$LIBRARY:FORIOSDEF'
0412    c
0413            integer*4 my_prog
0414            parameter (my_prog = 2)
0415    c
0416            logical readarr, writearr, skiparr, endspawn_if
0417            integer*4 lib$get_lun
0418    c
0419            logical ancestor
0420            integer*4 status, index_unit, data_unit, spawn_unit
0421            integer*4 waited, wait_lim/ 4 /, vstatus
0422            integer*4 ants_sqr( 2 ), ants_lin( 1 )
0423            integer*4 robs_add, rb_add, obst_add, noiset_add, eig_add, eigv_add
```

```
EIG_MUSIC                                                    18-Apr-1988 14:08:24
                                                              2-Apr-1988 14:41:20

0424            integer*4 panic
0425            real*8 condit
0426            character*1  sch_stat
0427            character*25 work_file
0428            character*80 msg
0429    c
0430    c               Get logical unit numbers
0431    c
0432            status = lib$get_lun( index_unit )
0433            if (.not. status) call lib$stop( %val( status ) )
0434            status = lib$get_lun( data_unit )
0435            if (.not. status) call lib$stop( %val( status ) )
0436            status = lib$get_lun( spawn_unit )
0437            if (.not. status) call lib$stop( %val( status ) )
0438    c
0439    c               Open the MUSIC index file.
0440    c
0441    10000   continue
0442            open (unit=index_unit,file='6000$RT:[CHUCK.PARAMS]MUSIC_INDEX.DAT',
0443            1       form='UNFORMATTED',recordtype='FIXED',recl=64,
0444            2       organization='INDEXED',access='KEYED',status='OLD',
0445            3       key=(1:4:INTEGER,5:5:CHARACTER,6:9:INTEGER),
0446            4                                   dispose='KEEP',SHARED,err=10100)
0447            go to 10110
0448    c
0449    c               Error opening the MUSIC index file or MUSIC spawned-processes
0450    c       index file; abandon hope.
0451    c
0452    10100     continue
0453              type *
0454              type *, 'Error opening MUSIC index file or spawned-processes file.'
0455              go to 60620
0456    c
0457    c               MUSIC index file successfully opened.  Open MUSIC spawned-
0458    c       processes index file.
0459    c
0460    10110   continue
0461            open (unit=spawn_unit,file='6000$RT:[CHUCK.PARAMS]SPAWNED.IDX',
0462            1       form='UNFORMATTED',recordtype='FIXED',recl=26,
0463            2       organization='INDEXED',access='KEYED',status='OLD',
0464            3       key=(1:4:INTEGER,5:8:INTEGER),dispose='KEEP',SHARED,err=10100)
0465    c
0466    c               MUSIC index file and MUSIC spawned-processes index file
0467    c       successfully opened.  Start lurking, waiting to pounce on unsuspecting
0468    c       status 'I' records.
0469    c
0470            waited   = 0
0471            sch_stat = 'I'
0472    10120   continue
0473              read (unit=index_unit,keyeq=sch_stat,keyid=1,iostat=status,err=20100)
0474            1               mi_pkey, mi_stat, mi_park, mi_ierr, mi_eerr, mi_serr,
0475            2               mi_dtim, mi_ants, mi_waves, mi_noises, mi_smp, mi_sint,
0476            3               mi_begp, mi_endp, mi_delp, mi_peps, mi_norm, mi_wlen,
0477            4               mi_tol, mi_which, mi_ofile, mi_anoises, mi_irlslim,
0478            5               mi_zeigs, mi_doas, mi_estype, mi_parp
0479              unlock (unit=index_unit)
0480              sch_stat = 'I'
```

```
0481                waited   = 0
0482                go to 21110
0483      c
0484      c                   Error reading status 'I' records.  Could be that the first such
0485      c         record is locked, no such records exist, or some other error.
0486      c
0487      20100     continue
0488                unlock (unit=index_unit)
0489                if      ( status .eq. FOR$IOS_SPERECLOC ) then      !  Locked.
0490                  waited = 0
0491                  type *, 'Locked record, status ', sch_stat, '.  Waiting 5 min.'
0492      20120       continue                                !  Branch to wait then read.
0493                  call lib$wait( 300.0 )                  !  Hibernate for 5 minutes.
0494                  go to 10120
0495                else if ( status .eq. FOR$IOS_ATTACCNON ) then      ! No record.
0496                  if ( sch_stat .eq. 'I' ) then                     ! May be 'i's.
0497                    sch_stat = 'i'
0498                    go to 10120
0499                  else                                    ! Neither 'I's nor 'i's.
0500                    go to 70000
0501                  end if
0502                else                                              ! Some other error.
0503                  type *
0504                  type *, 'Failed reading status ''I'' index records.'
0505                  call lib$stop( %val( status ) )
0506                end if
0507      c
0508      c                   Successfully read a status 'I' record into the batch of
0509      c         variables listed in the read statement above.  Commence eigenvalue
0510      c         problem solution.
0511      c                   Start by beating the Virtual Array system (copyright 1987 by
0512      c         Dwight Day) to death.
0513      c
0514      21110     continue
0515                ancestor = ( mi_park .eq. -1 )
0516      c.
0517                call init_va
0518                ants_lin(1)     = mi_ants
0519                ants_sqr(1)     = mi_ants
0520                ants_sqr(2)     = mi_ants
0521      c
0522                call open_va( robs_add, ants_sqr, 2, dc_type, vstatus, msg )
0523                call exit_va( vstatus, msg )
0524                call open_va( rb_add, ants_sqr, 2, dc_type, vstatus, msg )
0525                call exit_va( vstatus, msg )
0526                call open_va( obst_add, ants_sqr, 2, dc_type, vstatus, msg )
0527                call exit_va( vstatus, msg )
0528                call open_va( noiset_add, ants_sqr, 2, dc_type, vstatus, msg )
0529                call exit_va( vstatus, msg )
0530                call open_va( eig_add, ants_lin, 1, dr_type, vstatus, msg )
0531                call exit_va( vstatus, msg )
0532                call open_va( eigv_add, ants_sqr, 2, dc_type, vstatus, msg )
0533                call exit_va( vstatus, msg )
0534      c
0535      c                   Get covariance matrices and their QR decompositions from the
0536      c         experiment's "eigenvectors" file, LEAVING THE FILE OPEN for subsequent
0537      c         write of eigenproblem solutions.
```

```
EIG_MUSIC                                                    18-Apr-1988 14:08:24
                                                             2-Apr-1988 14:41:20

0538     c
0539               work_file = 'MUSIC_EIGVS__________.DAT'
0540               write (unit=work_file( 12 : 21 ),fmt=90020,err=62510) mi_pkey
0541     c
0542               open (unit=data_unit,file='6000$RT:[CHUCK.PARAMS]'//work_file,
0543              1       status='OLD',organization='SEQUENTIAL',access='SEQUENTIAL',
0544              2       form='UNFORMATTED',recordtype='FIXED',recl=64,dispose='KEEP',
0545              3                                       iostat=status,err=61510)
0546               if ( .not. ( ancestor ) ) then                          !  Descendant.
0547                 if ( .not. ( skiparr( mi_ants, mi_ants,
0548              1                                       data_unit, 16 ) ) ) go to 6050
0549               end if
0550               if ( .not. ( readarr( %val( robs_add ), dc_type, mi_ants, mi_ants,
0551              1                                       data_unit, 16 ) ) ) go to 6050
0552               if ( ancestor ) then                                    !  Ancestor.
0553                 if ( .not. ( skiparr( mi_ants, mi_ants,
0554              1                                       data_unit, 16 ) ) ) go to 6050
0555                 if ( mi_which .gt. 0 ) then
0556                   if ( .not. ( skiparr( mi_ants, mi_ants,
0557              1                                       data_unit, 16 ) ) ) go to 6050
0558                 end if
0559               end if
0560               if ( .not. ( readarr( %val( rb_add ), dc_type, mi_ants, mi_ants,
0561              1                                       data_unit, 16 ) ) ) go to 6050
0562               if ( .not. ( readarr( %val( obst_add ), dc_type, mi_ants, mi_ants,
0563              1                                       data_unit, 16 ) ) ) go to 6050
0564               if ( .not. ( readarr( %val( noiset_add ), dc_type, mi_ants,
0565              1                        mi_ants, data_unit, 16 ) ) ) go to 60500
0566     c
0567     c                    Solve the generalized eigenvalue / eigenvector problem for the
0568     c         observation covariance and normalized noise covariance.  Solution of
0569     c         the eigenvalue / eigenvector problem is via generalized singular value
0570     c         decomposition of the normalized observation and noise vector matrices.
0571     c
0572               call sq_vlgsvd( %val( obst_add ), %val( noiset_add ), mi_ants,
0573              1                        mi_tol, %val( eig_add ), %val( eigv_add ),
0574              2                                                 condit, panic
0575               call sort_eig( %val( eig_add ), %val( eigv_add ), mi_ants )
0576     c
0577     c                    Write results to eigenvalue problem output file, and close the
0578     c         file.
0579     c
0580               if ( .not. ( writearr( %val( eig_add ), dr_type, mi_ants, 1,
0581              1                                       data_unit, 32 ) ) ) go to 6052
0582               if ( .not. ( writearr( %val( eigv_add ), dc_type, mi_ants, mi_ants,
0583              1                                       data_unit, 16 ) ) ) go to 6052
0584               close (unit=data_unit)
0585     c
0586     c                    Print results, check for errors in Van Loan algorithm, and
0587     c         discard space used by Virtual Arrays.
0588     c
0589               call music_eig_rep( data_unit, mi_pkey, mi_park, mi_parp,
0590              1                        mi_ofile, %val( eig_add ), %val( eigv_add ),
0591              2                                                 mi_ants, mi_dtim )
0592               call vl_check( data_unit, mi_pkey, mi_park, mi_parp, mi_ofile,
0593              1                        %val( robs_add ), %val( rb_add ), mi_ants,
0594              2                        %val( eig_add ), %val( eigv_add ),
```

```
EIG_MUSIC                                                    18-Apr-1988 14:08:
                                                              2-Apr-1988 14:41:

0595            3                                       condit, panic, mi_dtim )
0596              call dump_va( vstatus, msg )
0597              call exit_va( vstatus, msg )
0598    c
0599              if ( panic .ne. 0 ) go to 60530       !  Mark for delete if fail
0600    c
0601    c               Update the experiment's index record, and exit if there is
0602    c       more work to do.
0603    c
0604              mi_stat = 'E'
0605    60600     continue
0606              read (unit=index_unit,key=mi_pkey,keyid=0,iostat=status)
0607              if      ( status .eq. FOR$IOS_SPERECLOC ) then
0608                type *, 'Locked record to rewrite, #', mi_pkey,
0609            1                                                 '.  Waiting 10 sec
0610                unlock (unit=index_unit)
0611                call lib$wait( 10.0 )                       !  Wait 10 seconds
0612                go to 60600
0613              else if ( status .ne. 0 ) then                !  Error reading i
0614    60610       continue                                !  Index access error brar
0615                unlock (unit=index_unit)
0616                type *
0617                type *, 'Error accessing MUSIC experiment index, code ', statu
0618    60620       continue                                    !  Error exit.
0619                type *, 'Eigenvector problem processing abandoned.'
0620                type *
0621                stop
0622              end if
0623              rewrite (unit=index_unit,iostat=status,err=60610)
0624            1               mi_pkey, mi_stat, mi_park, mi_ierr, mi_eerr, mi_se
0625            2               mi_dtim, mi_ants, mi_waves, mi_noises, mi_smp, mi_
0626            3               mi_begp, mi_endp, mi_delp, mi_peps, mi_norm, mi_wl
0627            4               mi_tol, mi_which, mi_ofile, mi_anoises, mi_irslim
0628            5               mi_zeigs, mi_doas, mi_estype, mi_parp
0629              unlock (unit=index_unit)
0630            go to 10120
0631    c
0632    c               No more status 'I' or 'i' records available.
0633    c
0634    70000   continue
0635            sch_stat = 'I'                              !  Revert to main search.
0636            if ( waited .ge. wait_lim ) then            !  Already waited 20 minut
0637              waited = 0
0638              if ( endspawn_if( spawn_unit, my_prog ) ) stop
0639            else
0640              waited = waited + 1
0641              call lib$wait( 300.0 )                    !  Wait 5 minutes & try ag
0642            end if
0643            go to 10120
0644    c
0645    c*******************************************************************************
0646    c
0647    c               File error processing branches.
0648    c
0649    c*******************************************************************************
0650    c
0651    c               Computation error, or error writing eigenvalue solutions f
```

```
EIG_MUSIC                                                    18-Apr-1988 14:08:24
                                                              2-Apr-1988 14:41:20

0652     c
0653     60520   continue
0654             close (unit=data_unit)
0655     c
0656     60530   continue
0657             call dump_va( vstatus, msg )
0658             call exit_va( vstatus, msg )
0659     c
0660     60540   continue
0661             mi_stat = 'X'
0662             go to 60600
0663     c
0664     c               Error reading input data file.
0665     c
0666     60500   continue
0667             close (unit=data_unit)
0668     c
0669     60510   continue
0670             call dump_va( vstatus, msg )
0671             call exit_va( vstatus, msg )
0672     c
0673             if ( mi_eerr .ge. 10 ) then
0674               mi_stat = 'X'
0675             else
0676               mi_eerr = mi_eerr + 1
0677               if ( ancestor ) then
0678                 mi_stat = ' '
0679               else
0680                 mi_stat = 'D'
0681               end if
0682             end if
0683             go to 60600
0684     c
0685     c               Error opening input data file - could be in use by another
0686     c       program.
0687     c
0688     61510   continue
0689               call dump_va( vstatus, msg )
0690               call exit_va( vstatus, msg )
0691     c
0692               if ( status .eq. FOR$IOS_OPEFAI ) then        !  Probably locked.
0693                 mi_stat = 'i'
0694               else                                          !  Some other error.
0695                 type *
0696                 type *, 'Failed opening input data file ', work_file, ', error #',
0697             1                                                        status, '.'
0698                 type *
0699                 mi_stat = 'X'                               !  Mark for deletion.
0700               end if
0701               go to 60600
0702     c
0703     c               Error performing internal write - abandon efforts.
0704     c
0705     62510     continue
0706               type *
0707               type *, 'Failed performing internal write.'
0708               go to 60620
```

```
EIG_MUSIC                                                    18-Apr-1988 14:08:24
                                                              2-Apr-1988 14:41:20

0709     c
0710     90020    format( i10.10 )
0711     c
0712              end


PROGRAM SECTIONS

     Name                                  Bytes   Attributes

  0 $CODE                                  2165   PIC CON REL LCL    SHR    EXE    RD NOWRT
  1 $PDATA                                  500   PIC CON REL LCL    SHR NOEXE    RD NOWRT
  2 $LOCAL                                 1296   PIC CON REL LCL NOSHR NOEXE    RD   WRT
  3 MUSIC_MEASURES                          120   PIC OVR REL GBL    SHR NOEXE    RD   WRT

     Total Space Allocated                 4081

ENTRY POINTS

     Address  Type  Name

  0-00000000       EIG_MUSIC

VARIABLES

     Address  Type  Name              Address  Type  Name              Address  Type  Na

        **    L*4   ANCESTOR      2-00000108  R*8   CONDIT        3-0000000C  I*4   CY
  2-00000178  I*4   EIGV_ADD      2-00000174  I*4   EIG_ADD       2-00000154  I*4   IN
  2-00000118  I*4   MI_ANTS       2-00000140  R*4   MI_BEGP       2-00000148  R*4   MI
  2-0000000D  CHAR  MI_DTIM       2-000000DC  I*2   MI_EERR       2-00000144  R*4   MI
  2-000000DA  I*2   MI_IERR       2-00000130  I*4   MI_IRLSLIM    2-00000120  I*4   MI
  2-00000020  CHAR  MI_OFILE      2-00000114  I*4   MI_PARK       2-0000014C  R*4   MI
  2-00000110  I*4   MI_PKEY       2-000000DE  I*2   MI_SERR       2-000000E0  R*8   MI
  2-0000000C  CHAR  MI_STAT       2-00000100  R*8   MI_TOL        2-0000011C  I*4   MI
  2-000000F8  R*8   MI_WLEN       2-00000134  I*4   MI_ZEIGS      2-0000008A  CHAR  MS
  3-00000060  R*8   NON_TRI       3-00000068  I*4   NON_TRI_CYC   2-0000016C  I*4   OB
  3-0000006C  R*8   QPSI_LMARG    3-00000074  I*4   QPSI_L_ZCYC   2-00000168  I*4   RB
  2-00000070  CHAR  SCH_STAT      3-00000008  I*4   SCORR         2-0000015C  I*4   SP
     **       I*4   SP_NUM           **       CHAR  SP_PROC          **       I*4   SP
  3-00000010  R*8   SV_IMERR      3-00000000  R*8   TOL           3-00000018  R*8   UP
  3-00000020  R*8   UPHI_SMARG    3-00000034  I*4   UPHI_S_ZCYC   3-00000028  R*8   U_
  3-0000003C  R*8   VPSI_LMARG    3-00000054  I*4   VPSI_L_ZCYC   3-00000044  R*8   VP
  2-00000160  I*4   VSTATUS       3-0000004C  R*8   V_UNIERR      3-0000005C  I*4   V_
     **       I*4   WAIT_LIM      2-00000071  CHAR  WORK_FILE
```

18-Apr-1988 14:09:42
16-Apr-1988 10:13:54

```
0001            program fin_music
0002    c
0003    c:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
0004    c
0005    c                   This program lurks in the SYS$SLOWBATCH queue (usually), and
0006    c       pounces on any record with a status of 'D' in MUSIC experiment index
0007    c       file 6000$RT:[CHUCK.PARAMS]MUSIC_INDEX.DAT.  When it finds such a
0008    c       record, it reads the information necessary to solve the Lp estimation
0009    c       problem from the appropriate data files, and proceeds to generate the
0010    c       required Lp estimates for the experiment specified in the selected
0011    c       index file record.  The program then writes its results to an estimates
0012    c       data file (selected by the experiment number), and updates the MUSIC
0013    c       experiment index record to a status of 'I'.  The above sequence is
0014    c       repeated until no more status 'D' records are found AND no process
0015    c       named "Chuck MUSIC Drv" can be found on the VAX.
0016    c                   The above description applies exactly to error-free operation
0017    c       only.  If a computational error occurs during the estimation process,
0018    c       or a file error occurs during reading of input data or writing of the
0019    c       solutions, the program changes the index record's status to 'X' instead
0020    c       of 'I'.  Finally, the program changes the index record's status to
0021    c       'd' if a data file can't be opened (the assumption being that the file
0022    c       is in use by another program).  Status 'd' records are processed after
0023    c       all status 'D' record operations have been completed.
0024    c
0025    c:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
0026    c
0027            implicit none
0028            external lib$get_lun, lib$stop, lib$wait, lib$delete_file
0029            external lib$init_timer, lib$stat_timer
0030            external init_va, open_va, free_va, exit_va, dump_va
0031            external readarr, writearr, skiparr
0032            external c16_sel_vdiff, sigmat_estm, music_lpp_each
0033            external music_lpw_var, music_est_rep, endspawn_if, spawn_if
0034            external accum_doas, mreaddoas, dc_matmpyherm
0035            external sqdc_mat_intdiv, music_lpp_speis, music_out_rep
0036            external dc_matconjg, dc_mathermmpy, dc_matmpy, c16_vachol, dc_matcopy
0037            external bracket_doas, c16_mat_dr_leftmul, dc_enfherm
0038    c
0039            include '6000$RT:[CHUCK.RESEARCH.FORTDIR]VATYPES.TXT'
0051            include '6000$RT:[CHUCK.RESEARCH.FORTDIR]MUSIC_MEASURES.FOR'
0140            include '6000$RT:[CHUCK.RESEARCH.FORTDIR]MUSIC_INDEX_DEF.TXT'
0281            include '6000$RT:[CHUCK.RESEARCH.FORTDIR]SPAWNED_DEF.TXT'
0324            include 'SYS$LIBRARY:FORIOSDEF'
0410    c
0411            integer*4 my_prog, next_prog
0412            parameter (my_prog = 3)
0413            parameter (next_prog = 2)
0414    c
0415            logical readarr, writearr, skiparr, endspawn_if
0416            integer*4 lib$init_timer, lib$stat_timer, lib$get_lun, spawn_if
0417    c
0418            logical last, have
0419            integer*4 status, index_unit, data_unit2, data_unit, spawn_unit
0420            integer*4 waited, wait_lim/ 4 /, vstatus
0421            integer*4 tstatus, ptics, gtics
0422            integer*4 mnitlim, atitlim
0423            integer*4 ants_by_2( 2 ), ants_by_wvs( 2 ), waves_lin( 1 )
```

```
FIN_MUSIC                                                    18-Apr-1988 14:09:42
                                                             16-Apr-1988 10:13:54

0424            integer*4 smp_by_ants( 2 ), wavs_by_smp( 2 ), ants_lin( 1 )
0425            integer*4 wavs_sqr( 2 ), ants_sqr( 2 )
0426            integer*4 ant_add, doa_add, amat_add
0427            integer*4 obsmat_add, noisemat_add, pvm_add, resm_add, rescj_add
0428            integer*4 yarw_add, plp_add, robs_add, rnoise_add, rb_add
0429            integer*4 obst_add, noiset_add, rescov_add, ap_add, apah_add
0430            integer*4 eigv_add, doaperf_add
0431            integer*4 ai_pkey, rec_num
0432            integer*4 iters, robscop_add, apahcop_add
0433            real*8 min_del, condit, down
0434            real*8 smul, amlim, amul, omul
0435            character*1  sch_stat
0436            character*25 work_file
0437            character*80 msg
0438    c
0439            integer*2 ti_ierr, ti_eerr, ti_serr
0440            integer*4 ti_pkey, ti_park, ti_ants, ti_waves, ti_noises, ti_smp
0441            integer*4 ti_which, ti_anoises, ti_irlslim, ti_zeigs, ti_doas
0442            integer*4 ti_estype
0443            real*4 ti_begp, ti_endp, ti_delp, ti_parp
0444            real*8 ti_sint, ti_peps, ti_norm, ti_wlen, ti_tol
0445            character*1 ti_stat
0446            character*19 ti_dtim
0447            character*80 ti_ofile
0448    c
0449    c               Get logical unit numbers
0450    c
0451            status = lib$get_lun( index_unit )
0452            if (.not. status) call lib$stop( %val( status ) )
0453            status = lib$get_lun( data_unit )
0454            if (.not. status) call lib$stop( %val( status ) )
0455            status = lib$get_lun( data_unit2 )
0456            if (.not. status) call lib$stop( %val( status ) )
0457            status = lib$get_lun( spawn_unit )
0458            if (.not. status) call lib$stop( %val( status ) )
0459    c
0460    c               Open the MUSIC index file.
0461    c
0462    10000   continue
0463            open (unit=index_unit,file='6000$RT:[CHUCK.PARAMS]MUSIC_INDEX.DAT',
0464           1        form='UNFORMATTED',recordtype='FIXED',recl=64,
0465           2        organization='INDEXED',access='KEYED',status='OLD',
0466           3        key=(1:4:INTEGER,5:5:CHARACTER,6:9:INTEGER),dispose='KEEP',
0467           4                                                SHARED,err=10100)
0468            go to 10110
0469    c
0470    c               Error opening the MUSIC index file or MUSIC spawned-processe
0471    c       index file; abandon hope.
0472    c
0473    10100     continue
0474              type *
0475              type *, 'Error opening MUSIC index file or spawned-processes file.
0476              go to 60620
0477    c
0478    c               MUSIC index file successfully opened.  Open MUSIC spawned-
0479    c       processes index file.
0480    c
```

```
FIN_MUSIC                                                    18-Apr-1988 14:09:42     V
                                                             16-Apr-1988 10:13:54     [

0481     10110   continue
0482             open (unit=spawn_unit,file='6000$RT:[CHUCK.PARAMS]SPAWNED.IDX',
0483             1       form='UNFORMATTED',recordtype='FIXED',recl=26,
0484             2       organization='INDEXED',access='KEYED',status='OLD',
0485             3       key=(1:4:INTEGER,5:8:INTEGER),dispose='KEEP',SHARED,err=10100)
0486     c
0487     c               MUSIC index file and MUSIC spawned-processes index file
0488     c       successfully opened.  Start lurking, waiting to pounce on unsuspecting
0489     c       status 'D' records.
0490     c
0491             waited   = 0
0492             sch_stat = 'D'
0493     10120   continue
0494               read (unit=index_unit,keyeq=sch_stat,keyid=1,iostat=status,err=20100)
0495             1               mi_pkey, mi_stat, mi_park, mi_ierr, mi_eerr, mi_serr,
0496             2               mi_dtim, mi_ants, mi_waves, mi_noises, mi_smp, mi_sint,
0497             3               mi_begp, mi_endp, mi_delp, mi_peps, mi_norm, mi_wlen,
0498             4               mi_tol, mi_which, mi_ofile, mi_anoises, mi_irlslim,
0499             5               mi_zeigs, mi_doas, mi_estype, mi_parp
0500               unlock (unit=index_unit)
0501               sch_stat = 'D'
0502               waited   = 0
0503               go to 20110
0504     c
0505     c               Error reading status 'D' records.  Could be that the first such
0506     c       record is locked, no such records exist, or some other error.
0507     c
0508     20100     continue
0509                 unlock (unit=index_unit)
0510                 if      ( status .eq. FOR$IOS_SPERECLOC ) then      !  Locked.
0511                   type *, 'Record locked, status ', sch_stat, '.  Waiting 5 min.'
0512                   waited = 0
0513     20120         continue                              !  Branch to wait then read.
0514                   call lib$wait( 300.0 )                !  Hibernate for 5 minutes.
0515                   go to 10120
0516                 else if ( status .eq. FOR$IOS_ATTACCNON ) then      ! No record.
0517                   if ( sch_stat .eq. 'D' ) then                     ! May be 'd's.
0518                     sch_stat = 'd'
0519                     go to 10120
0520                   else                                  ! Neither 'D's nor 'd's.
0521                     go to 70000
0522                   end if
0523                 else                                            ! Some other error.
0524     60830         continue
0525                   type *
0526                   type *, 'Failed reading index records or accessing scratch file.'
0527                   call lib$stop( %val( status ) )
0528                 end if
0529     20110     continue
0530     c
0531     c               Successfully read a status 'D' (or 'd') record into the batch
0532     c       of variables listed in the read statement above.  Now determine the
0533     c       current experiment's ancestor's key number.  Also collect DOA estimates
0534     c       to scratch disk file.
0535     c
0536               call init_va
0537               open (unit=data_unit2,file='6000$RT:[CHUCK.PARAMS]FMSCRATCH.DAT',
```

```
FIN_MUSIC                                                    18-Apr-1988 14:09:42    VAX F(
                                                             16-Apr-1988 10:13:54    [CHUCI

0538            1               form='UNFORMATTED',recordtype='FIXED',recl=2,
0539            2               organization='RELATIVE',access='DIRECT',
0540            3               status='NEW',dispose='DELETE',iostat=status,err=60830)
0541              ti_doas = 0
0542              write (unit=data_unit2,rec=1,iostat=status,err=60830) ti_doas
0543              ai_pkey   = mi_pkey
0544              ti_doas   = mi_doas
0545              ti_park   = mi_park
0546              work_file = 'MUSIC_SAMPL__________.DAT'
0547    c
0548              do while ( ti_park .ne. -1 )  !  Until we find the "ancestor".
0549                write (unit=work_file( 12 : 21 ),fmt=90020,err=60510) ai_pkey
0550                open (unit=data_unit,file='6000$RT:[CHUCK.PARAMS]'//work_file,
0551            1       status='OLD',organization='SEQUENTIAL',access='SEQUENTIAL',
0552            2       form='UNFORMATTED',recordtype='FIXED',recl=64,dispose='KEEP',
0553            3                                  iostat=status,READONLY,err=61500)
0554    c
0555                waves_lin(1) = ti_doas
0556                call open_va( doa_add, waves_lin, 1, dr_type, vstatus, msg )
0557                call exit_va( vstatus, msg )
0558                if ( .not. ( readarr( %val( doa_add ), dr_type, ti_doas, 1,
0559            1                                          data_unit, 32 ) ) ) go to 60500
0560                close (unit=data_unit)
0561    c
0562                call accum_doas( ti_doas, %val( doa_add ), data_unit2 )
0563                call free_va( doa_add, vstatus, msg )
0564                call exit_va( vstatus, msg )
0565    c
0566                ai_pkey  = ti_park
0567    21120       continue
0568                read (unit=index_unit,keyeq=ai_pkey,keyid=0,iostat=status,
0569            1                                                   err=21100)
0570            2               ai_pkey, ti_stat, ti_park, ti_ierr, ti_eerr, ti_serr,
0571            3               ti_dtim, ti_ants, ti_waves, ti_noises, ti_smp, ti_sint,
0572            4               ti_begp, ti_endp, ti_delp, ti_peps, ti_norm, ti_wlen,
0573            5               ti_tol, ti_which, ti_ofile, ti_anoises, ti_irlslim,
0574            6               ti_zeigs, ti_doas, ti_estype, ti_parp
0575                unlock (unit=index_unit)
0576                go to 21110
0577    c
0578    c               Error reading status 'D' record ancestor chain.  Could be that
0579    c       a record is locked, or some other error.
0580    c
0581    21100       continue
0582                  unlock (unit=index_unit)
0583                  if      ( status .eq. FOR$IOS_SPERECLOC ) then    !  Locked.
0584                    type *, 'Record #', ai_pkey, ' locked.  Waiting 5 min.'
0585    21130           continue                            !  Branch to wait then read.
0586                    call lib$wait( 300.0 )              !  Hibernate for 5 minutes.
0587                    go to 21120
0588                  else                                          ! Some other error.
0589                    go to 60830                                 ! Abandon program.
0590                  end if
0591    c
0592    21110       continue                                !  Successful read.
0593              end do                            !  End of loop to find "ancestor".
0594    c
```

```
0595     c                Get accumulated DOA estimates from scratch file, then close
0596     c        (and, as a result, delete) the file.
0597     c
0598               read (unit=data_unit2,rec=1) waves_lin(1)
0599               call open_va( doa_add, waves_lin, 1, dr_type, vstatus, msg )
0600               call exit_va( vstatus, msg )
0601               call open_va( doaperf_add, waves_lin, 1, dr_type, vstatus, msg )
0602               call exit_va( vstatus, msg )
0603               rec_num = 2
0604               call mreaddoas( data_unit2, rec_num, %val( doa_add ), waves_lin(1) )
0605               close (unit=data_unit2)
0606     c
0607     c                Commence estimation process.
0608     c
0609               wavs_sqr(1)    = waves_lin(1)
0610               wavs_sqr(2)    = waves_lin(1)
0611               ants_lin(1)    = mi_ants
0612               ants_sqr(1)    = mi_ants
0613               ants_sqr(2)    = mi_ants
0614               ants_by_wvs(1) = mi_ants
0615               ants_by_wvs(2) = waves_lin(1)
0616               ants_by_2(1)   = mi_ants
0617               ants_by_2(2)   = 2
0618               wavs_by_smp(1) = waves_lin(1)
0619               wavs_by_smp(2) = mi_smp
0620               smp_by_ants(1) = mi_smp
0621               smp_by_ants(2) = mi_ants
0622     c
0623               call open_va( ant_add, ants_by_2, 2, dr_type, vstatus, msg )
0624               call exit_va( vstatus, msg )
0625               call open_va( amat_add, ants_by_wvs, 2, dc_type, vstatus, msg )
0626               call exit_va( vstatus, msg )
0627               call open_va( obsmat_add, smp_by_ants, 2, dc_type, vstatus, msg )
0628               call exit_va( vstatus, msg )
0629               call open_va( pvm_add, wavs_by_smp, 2, dc_type, vstatus, msg )
0630               call exit_va( vstatus, msg )
0631               call open_va( resm_add, smp_by_ants, 2, dc_type, vstatus, msg )
0632               call exit_va( vstatus, msg )
0633               call open_va( resj_add, smp_by_ants, 2, dc_type, vstatus, msg )
0634               call exit_va( vstatus, msg )
0635               call open_va( plp_add, wavs_sqr, 2, dc_type, vstatus, msg )
0636               call exit_va( vstatus, msg )
0637               call open_va( robs_add, ants_sqr, 2, dc_type, vstatus, msg )
0638               call exit_va( vstatus, msg )
0639               call open_va( rescov_add, ants_sqr, 2, dc_type, vstatus, msg )
0640               call exit_va( vstatus, msg )
0641               call open_va( rb_add, ants_sqr, 2, dc_type, vstatus, msg )
0642               call exit_va( vstatus, msg )
0643               call open_va( obst_add, ants_sqr, 2, dc_type, vstatus, msg )
0644               call exit_va( vstatus, msg )
0645               call open_va( noiset_add, ants_sqr, 2, dc_type, vstatus, msg )
0646               call exit_va( vstatus, msg )
0647               call open_va( eigv_add, ants_sqr, 2, dc_type, vstatus, msg )
0648               call exit_va( vstatus, msg )
0649               call open_va( ap_add, ants_by_wvs, 2, dc_type, vstatus, msg )
0650               call exit_va( vstatus, msg )
0651               call open_va( apah_add, ants_sqr, 2, dc_type, vstatus, msg )
```

```
FIN_MUSIC                                                    18-Apr-1988 14:09:42     VAX
                                                             16-Apr-1988 10:13:54     [CH

0652              call exit_va( vstatus, msg )
0653              call open_va( robscop_add, ants_sqr, 2, dc_type, vstatus, msg )
0654              call exit_va( vstatus, msg )
0655              call open_va( apahcop_add, ants_sqr, 2, dc_type, vstatus, msg )
0656              call exit_va( vstatus, msg )
0657              if ( ( mi_estype .ne. 1 ) .and. ( mi_estype .ne. 3 ) ) then   ! SPwr.
0658                call open_va( yarw_add, ants_lin, 1, dr_type, vstatus, msg )
0659                call exit_va( vstatus, msg )
0660                call open_va( rnoise_add, ants_sqr, 2, dc_type, vstatus, msg )
0661                call exit_va( vstatus, msg )
0662              end if
0663              if ( ( mi_estype .eq. 3 )
0664             1       .or. ( mi_estype .eq. 5 )
0665             2       .or. ( mi_estype .eq. 6 ) ) then         !  Compensated estim.
0666                call open_va( noisemat_add, smp_by_ants, 2, dc_type, vstatus, msg )
0667                call exit_va( vstatus, msg )
0668              end if
0669        c
0670        c                   Get observation (and, for "compensated" estimation types,
0671        c         noise) samples from the "ancestor" experiment's "samples" file.
0672        c
0673              write (unit=work_file( 12 : 21 ),fmt=90020,err=60530) ai_pkey
0674              open (unit=data_unit,file='6000$RT:[CHUCK.PARAMS]'//work_file,
0675             1       status='OLD',organization='SEQUENTIAL',access='SEQUENTIAL',
0676             2       form='UNFORMATTED',recordtype='FIXED',recl=64,dispose='KEEP',
0677             3                                  iostat=status,READONLY,err=61510)
0678              if ( .not. ( readarr( %val( obsmat_add ), dc_type, mi_smp, mi_ants,
0679             1                                        data_unit, 16 ) ) ) go to 60520
0680              if ( ( mi_estype .eq. 3 )
0681             1       .or. ( mi_estype .eq. 5 )
0682             2       .or. ( mi_estype .eq. 6 ) ) then
0683                 if ( .not. ( readarr( %val( noisemat_add ), dc_type, mi_smp,
0684             1                             mi_ants, data_unit, 16 ) ) ) go to 60520
0685                call c16_sel_vdiff( %val( obsmat_add ), 1,
0686             1                             %val( noisemat_add ), 1, mi_smp * mi_ants,
0687             2                                                  %val( obsmat_add ) )
0688              end if
0689              close (unit=data_unit)
0690        c
0691        c                   Get antenna array definition from the "ancestor" experiment's
0692        c         "input" file.
0693        c
0694              work_file( 7 : 11 ) = 'INPUT'
0695              open (unit=data_unit,file='6000$RT:[CHUCK.PARAMS]'//work_file,
0696             1       status='OLD',organization='SEQUENTIAL',access='SEQUENTIAL',
0697             2       form='UNFORMATTED',recordtype='FIXED',recl=64,dispose='KEEP',
0698             3                                  iostat=status,READONLY,err=61510)
0699              read (unit=data_unit,err=60520)
0700              if ( .not. ( readarr( %val( ant_add ), dr_type, mi_ants, 2,
0701             1                                        data_unit, 32 ) ) ) go to 60520
0702              close (unit=data_unit)
0703        c
0704        c                   Get observation, noise (for "sum of power" estimation types),
0705        c         and normalized noise covariance matrices, and QR decomposition result
0706        c         matrices from the "ancestor" experiment's "eigenvectors" file.
0707        c
0708        c                   Also get eigenvectors.
```

```
FIN_MUSIC                                                  18-Apr-1988 14:09:42    VAX
                                                           16-Apr-1988 10:13:54    [CHU

0709     c
0710               work_file( 7 : 11 ) = 'EIGVS'
0711               open (unit=data_unit,file='6000$RT:[CHUCK.PARAMS]'//work_file,
0712              1       status='OLD',organization='SEQUENTIAL',access='SEQUENTIAL',
0713              2       form='UNFORMATTED',recordtype='FIXED',recl=64,dispose='KEEP',
0714              3                                iostat=status,READONLY,err=61510)
0715               if ( .not. ( readarr( %val( robs_add ), dc_type, mi_ants, mi_ants,
0716              1                                        data_unit, 16 ) ) ) go to 60520
0717               if ( ( mi_estype .ne. 1 ) .and. ( mi_estype .ne. 3 ) ) then
0718                 if ( .not. ( readarr( %val( rnoise_add ), dc_type, mi_ants,
0719              1                           mi_ants, data_unit, 16 ) ) ) go to 60520
0720               else
0721                 if ( .not. ( skiparr( mi_ants, mi_ants, data_unit, 16 ) ) )
0722              1                                                        go to 60520
0723               end if
0724               if ( ti_which .gt. 0 ) then
0725                 if ( .not. ( skiparr( mi_ants, mi_ants, data_unit, 16 ) ) )
0726              1                                                        go to 60520
0727               end if
0728               if ( .not. ( readarr( %val( rb_add ), dc_type, mi_ants, mi_ants,
0729              1                                        data_unit, 16 ) ) ) go to 60520
0730               if ( .not. ( skiparr( mi_ants, mi_ants, data_unit, 16 ) ) )
0731              1                                                        go to 60520
0732               if ( .not. ( readarr( %val( noiset_add ), dc_type, mi_ants, mi_ants,
0733              1                                        data_unit, 16 ) ) ) go to 60520
0734               if ( .not. ( skiparr( mi_ants, 1, data_unit, 32 ) ) ) go to 60520
0735               if ( .not. ( readarr( %val( eigv_add ), dc_type, mi_ants, mi_ants,
0736              1                                        data_unit, 16 ) ) ) go to 60520
0737               close (unit=data_unit)
0738     c
0739     c                 Get DOA estimate "peakedness" measures.
0740     c
0741               down = 1.0d0
0742               call bracket_doas( %val( doa_add ), waves_lin(1), %val( ant_add ),
0743              1                           mi_ants, mi_wlen, %val( eigv_add ), mi_zeigs,
0744              2                                  mi_parp, %val( doaperf_add ), down )
0745     c
0746     c                 Form estimate of A matrix.
0747     c
0748               call sigmat_estm( %val( ant_add ), mi_ants, %val( doa_add ),
0749              1                           waves_lin(1), mi_wlen, %val( amat_add ) )
0750     c
0751     c                 Form estimates of signal and residual vectors and P matrix as
0752     c            selected by the estimation type specified for the experiment.
0753     c
0754               tstatus = lib$init_timer()
0755               if ( .not. tstatus ) call lib$stop( %val( tstatus ) )
0756               if       ( ( mi_estype .eq. 1 )                          !  Enumeration.
0757              1               .or. ( mi_estype .eq. 3 ) ) then          !  Comp'd enum.
0758                 call music_lpp_each( %val( obsmat_add ), mi_smp, mi_ants,
0759              1                           %val( amat_add ), waves_lin(1), mi_parp,
0760              2                           mi_peps, mi_norm, mi_tol, %val( pvm_add ),
0761              3                           %val( resm_add ), mi_irlslim, mnitlim,
0762              4                                                atitlim, min_del )
0763                 call dc_matmpyherm( %val( pvm_add ), waves_lin(1), mi_smp,
0764              1                           %val( pvm_add ), waves_lin(1),
0765              2                                                %val( plp_add ) )
```

```
0766                call sqdc_mat_intdiv( %val( plp_add ), waves_lin(1), mi_smp )
0767              else if ( ( mi_estype .eq. 2 )                          !  Sum powers.
0768             1              .or. ( mi_estype .eq. 5 ) ) then         !  C'd sum pwr.
0769                call music_lpw_yar( %val( obsmat_add ), mi_smp, mi_ants,
0770             1                      %val( amat_add ), waves_lin(1), mi_parp,
0771             2                      mi_peps, mi_norm, mi_tol, %val( yarw_add ),
0772             3                      %val( pvm_add ), %val( resm_add ),
0773             4                        mi_irlslim, mnitlim, min_del, 1 )
0774                call music_lpp_speis( %val( amat_add ), mi_ants, waves_lin(1),
0775             1                      %val( yarw_add ), %val( robs_add ),
0776             2                      %val( rnoise_add ), mi_tol,
0777             3                                  %val( plp_add ), status )
0778                if ( status .ne. 0 ) mnitlim = - mnitlim - 1
0779              else if ( ( mi_estype .eq. 4 )                          !  S. pwr loss.
0780             1              .or. ( mi_estype .eq. 6 ) ) then         !  C. SPw loss.
0781                call music_lpw_yar( %val( obsmat_add ), mi_smp, mi_ants,
0782             1                      %val( amat_add ), waves_lin(1), mi_parp,
0783             2                      mi_peps, mi_norm, mi_tol, %val( yarw_add ),
0784             3                      %val( pvm_add ), %val( resm_add ),
0785             4                        mi_irlslim, mnitlim, min_del, 2 )
0786                call music_lpp_speis( %val( amat_add ), mi_ants, waves_lin(1),
0787             1                      %val( yarw_add ), %val( robs_add ),
0788             2                      %val( rnoise_add ), mi_tol,
0789             3                                  %val( plp_add ), status )
0790                if ( status .ne. 0 ) mnitlim = - mnitlim - 1
0791              else                                !  Invalid method.
0792                go to 60530                       !  Mark for deletion.
0793              end if
0794     c
0795     c                 Compute residuals covariance matrix, updated observation
0796     c          covariance matrix, and Cholesky decomposition of updated covariance
0797     c          matrix.
0798     c
0799                tstatus = lib$stat_timer( 2, ptics )
0800                if ( .not. tstatus ) call lib$stop( %val( tstatus ) )
0801                tstatus = lib$init_timer()
0802                if ( .not. tstatus ) call lib$stop( %val( tstatus ) )
0803     c
0804                call dc_matcopy( %val( resm_add ), mi_smp, mi_ants,
0805             1                                          %val( rescj_add ) )
0806                call dc_matconjg( %val( rescj_add ), mi_smp, mi_ants )
0807                call dc_mathermmpy( %val( rescj_add ), mi_smp, mi_ants,
0808             1                    %val( rescj_add ), mi_ants, %val( rescov_add ) )
0809                call sqdc_mat_intdiv( %val( rescov_add ), mi_ants, mi_smp )
0810
0811                call dc_matmpy( %val( amat_add ), mi_ants, waves_lin(1),
0812             1                      %val( plp_add ), waves_lin(1), %val( ap_add ) )
0813                call dc_matmpyherm( %val( ap_add ), mi_ants, waves_lin(1),
0814             1                      %val( amat_add ), mi_ants, %val( apah_add ) )
0815     c
0816     c                 Determine "almost optimum" multiplier for A P A-conj-transp.
0817     c
0818                iters = 0
0819                smul = 1.0d0
0820                have = .false.
0821                last = .false.
0822     40000      continue
```

```
FIN_MUSIC                                                    18-Apr-1988 14:09:42    VAX FOI
                                                             16-Apr-1988 10:13:54    ECHUCK

0823                call dc_matcopy( %val( robs_add ), mi_ants, mi_ants,
0824               1                                          %val( robscop_add ) )
0825                call dc_matcopy( %val( apah_add ), mi_ants, mi_ants,
0826               1                                          %val( apahcop_add ) )
0827                call c16_mat_dr_leftmul( %val( apahcop_add ), mi_ants,
0828               1                                      mi_ants, smul, mi_ants )
0829                call c16_sel_vdiff( %val( robscop_add ), 1, %val( apahcop_add ),
0830               1                          1, mi_ants * mi_ants, %val( robscop_add ) )
0831                call dc_enfherm( %val( robscop_add ), mi_ants, mi_tol )
0832                call c16_vachol( %val( robscop_add ), mi_ants,
0833               1                                %val( obst_add ), condit, status )
0834                if ( last ) go to 40010                 !  Just re-did last working.
0835                if ( have ) then                        !  Have a working multiplier.
0836                  if ( status .eq. 0 ) then             !  Cholesky succeeded.
0837                    omul = smul
0838                  else                                  !  Cholesky failed.
0839                    smul = omul
0840                  end if
0841                  if ( amul .ge. amlim ) then           !  Insufficient resolution.
0842                    amul = amul * 0.5d0                 !  Up the resolution some.
0843                    smul = smul + amul                  !  Try slightly larger mult.
0844                  else                                  !  Sufficient resolution.
0845                    last = .true.                       !  Set last time through.
0846                  end if
0847                else                                    !  Don't have working yet.
0848                  if ( status .eq. 0 ) then             !  Cholesky succeeded.
0849                    amlim = smul * 0.0078125d0          !  Set resolution limit.
0850                    amul = smul * 0.5d0                 !  Set first mult. add term.
0851                    omul = smul                         !  Save working multiplier.
0852                    smul = smul + amul                  !  Try slightly larger mult.
0853                    have = .true.                       !  Indicate we have working m.
0854                  else                                  !  Cholesky failed.
0855                    iters = iters + 1                   !  Indicate another pass.
0856                    if ( iters .gt. 25 ) then           !  Tried all we care to.
0857                      smul = 0.0d0                      !  Don't subtract any.
0858                      last = .true.                     !  Indicate last attempt.
0859                    else                                !  Haven't tried too many.
0860                      smul = smul * 0.5d0               !  Try half of previous mult.
0861                    end if
0862                  end if
0863                end if
0864                go to 40000                             !  Try again.
0865     40010      continue
0866     c
0867                tstatus = lib$stat_timer( 2, gtics )
0868                if ( .not. tstatus ) call lib$stop( %val( tstatus ) )
0869                call dc_matcopy( %val( robscop_add ), mi_ants, mi_ants,
0870               1                                          %val( robs_add ) )
0871     c
0872     c                  Print estimation results, and cancel further processing if a
0873     c          computation failure has occurred.
0874     c
0875                call music_est_rep( data_unit, mi_ofile, mi_dtim, mi_pkey,
0876               1                    mi_park, %val( doa_add ), %val( doaperf_add ),
0877               2                    %val( pip_add ), waves_lin(1), smul, down,
0878               3                    %val( amat_add ), mi_ants, mi_parp, mi_estype,
0879               4                    mi_irlslim, mnitlim, atitlim, min_del, condit,
```

```
FIN_MUSIC                                                  18-Apr-1988 14:09:42    VAX Fi
                                                           16-Apr-1988 10:13:54    [CHUC
0880           5                        status, ptics / 100.0d0, gtics / 100.0d0 )
0881             call music_out_rep( data_unit, mi_pkey, mi_park, mi_parp, ai_pkey,
0882           1                                 mi_which, mi_ofile, %val( robs_add ),
0883           2                                 0, 0, mi_ants, 0.0d0, mi_dtim )
0884             if ( ( status .ne. 0 ) .or. ( mnitlim .lt. 0 ) ) go to 60530
0885     c
0886     c               Write signal and residual vector estimates to output file.
0887     c
0888             work_file( 7 : 11 ) = 'SAMPL'
0889             write (unit=work_file( 12 : 21 ),fmt=90020,err=60530) mi_pkey
0890             open (unit=data_unit,file='6000$RT:[CHUCK.PARAMS]'//work_file,
0891           1                 status='OLD',organization='SEQUENTIAL',
0892           2                 access='SEQUENTIAL',form='UNFORMATTED',
0893           3                 recordtype='FIXED',recl=64,dispose='KEEP',err=60530)
0894             if ( .not. ( skiparr( mi_doas, 1, data_unit, 32 ) ) ) go to 60520
0895             if ( .not. ( writearr( %val( resm_add ), dc_type, mi_smp,
0896           1                        mi_ants, data_unit, 16 ) ) ) go to 60520
0897             if ( .not. ( writearr( %val( pvm_add ), dc_type, waves_lin(1),
0898           1                        mi_smp, data_unit, 16 ) ) ) go to 60520
0899             close (unit=data_unit)
0900     c
0901     c               Create eigenvectors file for the current experiment, and write
0902     c       the covariance and factored covariance matrices to the file.
0903     c
0904             work_file( 7 : 11 ) = 'EIGVS'
0905             call lib$delete_file(
0906           1                 '6000$RT:[CHUCK.PARAMS]' // work_file // ';*' )
0907             open (unit=data_unit,file='6000$RT:[CHUCK.PARAMS]'//work_file,
0908           1       status='NEW',organization='SEQUENTIAL',access='SEQUENTIAL',
0909           2       form='UNFORMATTED',recordtype='FIXED',recl=64,dispose='KEEP',
0910           3                                                       err=60530)
0911             if ( .not. ( writearr( %val( rescov_add ), dc_type, mi_ants,
0912           1                        mi_ants, data_unit, 16 ) ) ) go to 60520
0913             if ( .not. ( writearr( %val( robs_add ), dc_type, mi_ants,
0914           1                        mi_ants, data_unit, 16 ) ) ) go to 60520
0915             if ( .not. ( writearr( %val( rb_add ), dc_type, mi_ants,
0916           1                        mi_ants, data_unit, 16 ) ) ) go to 60520
0917             if ( .not. ( writearr( %val( obst_add ), dc_type, mi_ants,
0918           1                        mi_ants, data_unit, 16 ) ) ) go to 60520
0919             if ( .not. ( writearr( %val( noiset_add ), dc_type, mi_ants,
0920           1                        mi_ants, data_unit, 16 ) ) ) go to 60520
0921             close (unit=data_unit)
0922     c
0923     c               Discard space used by Virtual Arrays, update the experiment's
0924     c       index record, and exit if there is no more work to do.
0925     c
0926             call dump_va( vstatus, msg )
0927             call exit_va( vstatus, msg )
0928             mi_stat = 'I'
0929     c
0930             status = spawn_if( spawn_unit, next_prog )     !  Start eigenvalues.
0931             if ( status .ne. 0 ) then                      !  Spawn failed.
0932               type *
0933               type *, 'Error initiating eigenvalue solver.'
0934               if      ( status .eq. 1 ) then
0935                 type *, 'Read error on spawned-processes index file.'
0936               else if ( status .eq. 2 ) then
```

```
FIN_MUSIC                                                    18-Apr-1988 14:09:42    V
                                                             16-Apr-1988 10:13:54    [

0937                  type *, 'Error spawning subprocess.'
0938                else if ( status .eq. 3 ) then
0939                  type *, 'Error rewriting spawned-processes index file.'
0940                else
0941                  type *, 'Impossible error, code ', status, '.'
0942                end if
0943                go to 60620
0944              end if
0945      c
0946      60600     continue
0947                read (unit=index_unit,key=mi_pkey,keyid=0,iostat=status)
0948                if      ( status .eq. FOR$IOS_SPERECLOC ) then
0949                  type *, 'Locked record to rewrite, #', mi_pkey,
0950             1                                                 '.  Waiting 10 sec.'
0951                  unlock (unit=index_unit)
0952                  call lib$wait( 10.0 )                        !  Wait 10 seconds.
0953                  go to 60600
0954                else if ( status .ne. 0 ) then                 !  Error reading index.
0955      60610       continue                                  !  Index access error branch.
0956                  unlock (unit=index_unit)
0957                  type *
0958                  type *, 'Error accessing MUSIC experiment index, code ', status
0959      60620       continue                                     !  Error exit.
0960                  type *, 'Lp estimation abandoned.'
0961                  type *
0962                  stop
0963                end if
0964                rewrite (unit=index_unit,iostat=status,err=60610)
0965             1                  mi_pkey, mi_stat, mi_park, mi_ierr, mi_eerr, mi_serr,
0966             2                  mi_dtim, mi_ants, mi_waves, mi_noises, mi_smp, mi_sint,
0967             3                  mi_begp, mi_endp, mi_delp, mi_peps, mi_norm, mi_wlen,
0968             4                  mi_tol, mi_which, mi_ofile, mi_anoises, mi_irlslim,
0969             5                  mi_zeigs, mi_doas, mi_estype, mi_parp
0970                unlock (unit=index_unit)
0971              go to 10120
0972      c
0973      c                 No more status 'D' records available.
0974      c
0975      70000   continue
0976              if ( waited .ge. wait_lim ) then           !  Already waited 20 minutes.
0977                waited = 0
0978                if ( endspawn_if( spawn_unit, my_prog ) ) stop
0979              else
0980                waited = waited + 1
0981                call lib$wait( 300.0 )                   !  Wait 5 minutes & try again.
0982              end if
0983              go to 10120
0984      c
0985      c*******************************************************************************
0986      c
0987      c                 File error processing branches.
0988      c
0989      c*******************************************************************************
0990      c
0991      c                 Computation or file access error.
0992      c
0993      60500   continue
```

```
0994            close (unit=data_unit)
0995    c
0996    60510   continue
0997            close (unit=data_unit2)
0998            go to 60530
0999    c
1000    60520   continue
1001            close (unit=data_unit)
1002    c
1003    60530   continue
1004            call dump_va( vstatus, msg )
1005            call exit_va( vstatus, msg )
1006    c
1007    60540   continue
1008            mi_stat = 'X'
1009            go to 60600
1010    c
1011    c               Error opening input data file - could be in use by another
1012    c       program.
1013    c
1014    61500   continue
1015              close (unit=data_unit2)
1016    c
1017    61510   continue
1018              call dump_va( vstatus, msg )
1019              call exit_va( vstatus, msg )
1020    c
1021              if ( status .eq. FOR$IOS_OPEFAI ) then        !  Probably locked.
1022                mi_stat = 'd'
1023              else                                          !  Some other error.
1024                type *
1025                type *, 'Failed opening input data file ', work_file, ', error #',
1026             1                                                    status, '.'
1027                type *
1028                mi_stat = 'X'                               !  Mark for deletion.
1029              end if
1030              go to 60600
1031    c
1032    90020   format( i10.10 )
1033    c
1034            end
```

```
                                                    18-Apr-1988 14:10:36     VA
                                                     2-Apr-1988 14:46:06     DC

0001            program music_pest
0002    c
0003    c:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
0004    c
0005    c                   This program lurks in the SYS$SLOWBATCH queue (usually), and
0006    c       pounces on any file matching the specification
0007    c       6000$RT:[CHUCK.PARAMS]MUSIC_STDOA%%%%%%%%%%.DAT.  When it finds such a
0008    c       file, it reads the information necessary to solve the Lp estimation
0009    c       problem from the appropriate data files, and proceeds to generate the
0010    c       required Lp estimates of the P matrix for the experiment which
0011    c       created the file.  The program then writes its results to the
0012    c       appropriate output file (selected by the experiment number), and
0013    c       deletes the input file.  The above sequence is repeated until no more
0014    c       input files are found AND no process named "Chuck MUSIC Drv" can be
0015    c       found on the VAX.
0016    c
0017    c:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
0018    c
0019            implicit none
0020            external lib$get_lun, lib$stop, lib$wait
0021            external lib$init_timer, lib$stat_timer
0022            external lib$find_file, lib$find_file_end, lib$delete_file
0023            external init_va, open_va, exit_va, dump_va
0024            external endspawn_if, readarr, skiparr
0025            external c16_sel_vdiff, sigmat_estm, dc_matmpyherm, sqdc_mat_intdiv
0026            external music_l2p, music_lpp_each, music_lpw_yar, music_lpp_speis
0027            external music_p_rep
0028    c
0029            logical endspawn_if, readarr, skiparr
0030            integer*4 lib$init_timer, lib$stat_timer
0031            integer*4 lib$get_lun, lib$find_file, lib$find_file_end
0032    c
0033            include '6000$RT:[CHUCK.RESEARCH.FORTDIR]VATYPES.TXT'
0045            include '6000$RT:[CHUCK.RESEARCH.FORTDIR]MUSIC_MEASURES.FOR'
0134            include '6000$RT:[CHUCK.RESEARCH.FORTDIR]MUSIC_INDEX_DEF.TXT'
0275            include '6000$RT:[CHUCK.RESEARCH.FORTDIR]SPAWNED_DEF.TXT'
0318            include 'SYS$LIBRARY:FORIOSDEF'
0404    c
0405            integer*4 my_prog
0406            parameter (my_prog = 4)
0407    c
0408            logical found
0409            integer*4 index_unit, data_unit1, data_unit2, spawn_unit
0410            integer*4 status, vstatus, l2status, findadd, waits, waits_lim/ 4 /
0411            integer*4 fails, fails_lim/ 100 /
0412            integer*4 parent, experiment, ancestor, ants, waves, exps, estim
0413            integer*4 waves_lin( 1 ), doa_add, expnum_add
0414            integer*4 waves_by_9( 2 ), waves_sqr( 2 ), ants_lin( 1 )
0415            integer*4 ants_by_wvs( 2 ), ants_sqr( 2 ), ants_by_2( 2 )
0416            integer*4 wavs_by_smp( 2 ), smp_by_ants( 2 )
0417            integer*4 wave_add, ant_add, amat_add, obsmat_add, noisemat_add
0418            integer*4 pvm_add, resm_add, pl2_add, plp_add, pmat_add, sigvm_add
0419            integer*4 robs_add, rnoise_add, yarw_add
0420            integer*4 mnitlim, atitlim, tstatus, l2tics, lptics
0421            real*4 p
0422            real*8 min_del
0423            character*25 work_file
```

```
MUSIC_PEST                                                    18-Apr-1988 14:10:36     VAX
                                                               2-Apr-1988 14:46:06     [CHU

0424            character*80 msg
0425            character*255 found_file
0426    c
0427    c               Get logical unit numbers
0428    c
0429            status = lib$get_lun( index_unit )
0430            if (.not. status) call lib$stop( %val( status ) )
0431            status = lib$get_lun( data_unit1 )
0432            if (.not. status) call lib$stop( %val( status ) )
0433            status = lib$get_lun( data_unit2 )
0434            if (.not. status) call lib$stop( %val( status ) )
0435            status = lib$get_lun( spawn_unit )
0436            if (.not. status) call lib$stop( %val( status ) )
0437    c
0438    c               Open the MUSIC index file.
0439    c
0440    10000   continue
0441            open (unit=index_unit,file='6000$RT:[CHUCK.PARAMS]MUSIC_INDEX.DAT',
0442            1       form='UNFORMATTED',recordtype='FIXED',recl=64,
0443            2       organization='INDEXED',access='KEYED',status='OLD',
0444            3       key=(1:4:INTEGER,5:5:CHARACTER,6:9:INTEGER),dispose='KEEP',
0445            4                                                   SHARED,err=10100)
0446            go to 10110
0447    c
0448    c               Error opening the MUSIC index file or MUSIC spawned-processes
0449    c       index file; abandon hope.
0450    c
0451    10100     continue
0452              type *
0453              type *, 'Error opening MUSIC index file or spawned-processes file.'
0454    c
0455    60620     continue                                  !  Error exit.
0456              type *, 'Lp estimation abandoned.'
0457              type *
0458              stop
0459    c
0460    c               MUSIC index file successfully opened.  Open MUSIC spawned-
0461    c       processes index file.
0462    c
0463    10110   continue
0464            open (unit=spawn_unit,file='6000$RT:[CHUCK.PARAMS]SPAWNED.IDX',
0465            1       form='UNFORMATTED',recordtype='FIXED',recl=26,
0466            2       organization='INDEXED',access='KEYED',status='OLD',
0467            3       key=(1:4:INTEGER,5:8:INTEGER),dispose='KEEP',SHARED,err=10100)
0468    c
0469    c               MUSIC index file and MUSIC spawned-processes index file
0470    c       successfully opened.  Start lurking, waiting to pounce on unsuspecting
0471    c       input data files.
0472    c
0473            waits = 0
0474            fails = 0
0475    20000   continue
0476            findadd = 0
0477            found   = .false.
0478    21000   continue
0479            status = lib$find_file(
0480            1               '6000$RT:[CHUCK.PARAMS]MUSIC_STDOA%%%%%%%%%%.DAT;*',
```

```
MUSIC_PEST                                                    18-Apr-1988 14:10:36     VAX I
                                                               2-Apr-1988 14:46:06     [CHU

0481            2                                                      found_file, findadd )
0482            if ( .not. status ) then                    !  No input files.
0483              status = lib$find_file_end( findadd ) !  Complete file search.
0484              if ( .not. found ) then                   !  Found no files before.
0485                if ( waits .ge. waits_lim ) then        !  Already waited 20 minutes.
0486                  waits = 0
0487                  if ( endspawn_if( spawn_unit, my_prog ) ) stop
0488                else
0489                  waits = waits + 1
0490                  call lib$wait( 300.0 )                !  Wait 5 minutes & try again.
0491                end if
0492              end if
0493              go to 20000                               !  Start new file search.
0494            end if
0495    c
0496    c               Found an input file.  Get experiment number, ancestor number,
0497    c       number of experiments in ancestor chain, estimation type to use, and
0498    c       value of p to use for estimation.
0499    c
0500            found = .true.
0501            waits = 0
0502    23000   continue
0503            open (unit=data_unit1,file=found_file,status='OLD',
0504            1       organization='SEQUENTIAL',access='SEQUENTIAL',
0505            2       form='UNFORMATTED',recordtype='FIXED',recl=8,dispose='KEEP',
0506            3                                  READONLY,iostat=status,err=80000)
0507            go to 80010
0508    c
0509    80000   continue
0510              if ( waits .ge. waits_lim ) then      !  Already tried for 20 min.
0511                type *
0512                type *, 'Failed opening input file:'
0513                type *, found_file
0514                type *, waits_lim, ' times in a row.'
0515                go to 60620
0516              else                                  !  Haven't tried for 20 min.
0517                call lib$wait( 300.0 )              !  Wait 5 minutes & try again.
0518                go to 23000
0519              end if
0520    c
0521    80010   continue
0522            call init_va                        !  Prepare for VA operations.
0523            waits = 0
0524            parent = 1
0525            do while ( parent .ge. 0 )              !  Find definitions record.
0526              read (unit=data_unit1,err=50000)
0527            1                       parent, experiment, ancestor, exps, estim, p
0528            end do
0529    c
0530    c               Collect experiment definition from ancestor's index record.
0531    c
0532    22000   continue
0533            read (unit=index_unit,keyeq=ancestor,keyid=0,iostat=status)
0534            1               mi_pkey, mi_stat, mi_park, mi_ierr, mi_eerr, mi_serr,
0535            2               mi_dtim, mi_ants, mi_waves, mi_noises, mi_smp, mi_sint,
0536            3               mi_begp, mi_endp, mi_delp, mi_peps, mi_norm, mi_wlen,
0537            4               mi_tol, mi_which, mi_ofile, mi_anoises, mi_irlslim,
```

```
0538            5               mi_zeigs, mi_doas, mi_estype, mi_parp
0539            unlock (unit=index_unit)
0540            if      ( status .eq. FOR$IOS_SPERECLOC ) then  !  Record locked.
0541              type *, 'Record #', ancestor, ' locked.  Waiting 5 min.'
0542              call lib$wait( 300.0 )                        !  Wait 5 minutes.
0543              go to 22000
0544            else if ( status .ne. 0 ) then                  !  Error reading index.
0545              close (unit=index_unit)
0546              close (unit=data_unit1)
0547              type *
0548              type *, 'Error reading index file, experiment #', ancestor, '.'
0549              go to 60620
0550            end if
0551    c
0552    c               Ancestor experiment definition read into wad of variables
0553    c       listed above.  Open virtual arrays for DOA estimates and corresponding
0554    c       experiment numbers, and get DOA estimate and experiment number arrays.
0555    c
0556            waves_lin(1) = mi_waves
0557    c
0558            call open_va( doa_add, waves_lin, 1, dr_type, vstatus, msg )
0559            call exit_va( vstatus, msg )
0560            call open_va( expnum_add, waves_lin, 1, di_type, vstatus, msg )
0561            call exit_va( vstatus, msg )
0562    c
0563            if ( .not. ( readarr( %val( doa_add ), dr_type, mi_waves, 1,
0564            1                                       data_unit1, 4 ) ) ) go to 50000
0565            if ( .not. ( readarr( %val( expnum_add ), di_type, mi_waves, 1,
0566            1                                       data_unit1, 8 ) ) ) go to 50000
0567            close (unit=data_unit1)
0568    c
0569    c               Now collect antenna array description, wave descriptions,
0570    c       observation samples, noise samples, and covariance matrices as
0571    c       required.  Finally, form estimate of P matrix as selected by estimation
0572    c       type.
0573    c
0574            waves_by_9(1)   = mi_waves
0575            waves_by_9(2)   = 9
0576            waves_sqr(1)    = mi_waves
0577            waves_sqr(2)    = mi_waves
0578            ants_lin(1)     = mi_ants
0579            ants_by_wvs(1) = mi_ants
0580            ants_by_wvs(2) = mi_waves
0581            ants_sqr(1)     = mi_ants
0582            ants_sqr(2)     = mi_ants
0583            ants_by_2(1)    = mi_ants
0584            ants_by_2(2)    = 2
0585            wavs_by_smp(1) = mi_waves
0586            wavs_by_smp(2) = mi_smp
0587            smp_by_ants(1) = mi_smp
0588            smp_by_ants(2) = mi_ants
0589    c
0590            call open_va( wave_add, waves_by_9, 2, dr_type, vstatus, msg )
0591            call exit_va( vstatus, msg )
0592            call open_va( ant_add, ants_by_2, 2, dr_type, vstatus, msg )
0593            call exit_va( vstatus, msg )
0594            call open_va( amat_add, ants_by_wvs, 2, dc_type, vstatus, msg )
```

```
MUSIC_PEST                                                  18-Apr-1988 14:10:36     VAX
                                                             2-Apr-1988 14:46:06     [CH

0595            call exit_va( vstatus, msg )
0596            call open_va( obsmat_add, smp_by_ants, 2, dc_type, vstatus, msg )
0597            call exit_va( vstatus, msg )
0598            call open_va( noisemat_add, smp_by_ants, 2, dc_type, vstatus, msg )
0599            call exit_va( vstatus, msg )
0600            call open_va( pvm_add, wavs_by_smp, 2, dc_type, vstatus, msg )
0601            call exit_va( vstatus, msg )
0602            call open_va( resm_add, smp_by_ants, 2, dc_type, vstatus, msg )
0603            call exit_va( vstatus, msg )
0604            call open_va( pi2_add, waves_sqr, 2, dc_type, vstatus, msg )
0605            call exit_va( vstatus, msg )
0606            call open_va( pip_add, waves_sqr, 2, dc_type, vstatus, msg )
0607            call exit_va( vstatus, msg )
0608            call open_va( pmat_add, waves_sqr, 2, dc_type, vstatus, msg )
0609            call exit_va( vstatus, msg )
0610            call open_va( sigvm_add, wavs_by_smp, 2, dc_type, vstatus, msg )
0611            call exit_va( vstatus, msg )
0612            call open_va( robs_add, ants_sqr, 2, dc_type, vstatus, msg )
0613            call exit_va( vstatus, msg )
0614            call open_va( rnoise_add, ants_sqr, 2, dc_type, vstatus, msg )
0615            call exit_va( vstatus, msg )
0616            if ( ( estim .ne. 1 ) .and. ( estim .ne. 3 ) ) then     !  Spwr. type.
0617              call open_va( yarw_add, ants_lin, 1, dr_type, vstatus, msg )
0618              call exit_va( vstatus, msg )
0619            end if
0620    c
0621    c                Get observation, noise, and signal vector samples from the
0622    c       ancestor experiment's "samples" file.
0623    c
0624            work_file = 'MUSIC_SAMPL__________.DAT'
0625            write (unit=work_file( 12 : 21 ),fmt=90020,err=50200) ancestor
0626            open (unit=data_unit1,file='6000$RT:[CHUCK.PARAMS]'//work_file,
0627            1       status='OLD',organization='SEQUENTIAL',access='SEQUENTIAL',
0628            2       form='UNFORMATTED',recordtype='FIXED',recl=64,dispose='KEEP',
0629            3                                 iostat=status,READONLY,err=50010)
0630            if ( .not. ( readarr( %val( obsmat_add ), dc_type, mi_smp, mi_ants,
0631            1                                 data_unit1, 16 ) ) ) go to 50100
0632            if ( .not. ( readarr( %val( noisemat_add ), dc_type, mi_smp, mi_ants,
0633            1                                 data_unit1, 16 ) ) ) go to 50100
0634            if ( .not. ( readarr( %val( sigvm_add ), dc_type, mi_waves, mi_smp,
0635            1                                 data_unit1, 16 ) ) ) go to 50100
0636            close (unit=data_unit1)
0637    c
0638    c                Get antenna array and wave definitions from the ancestor
0639    c       experiment's "input" file.
0640    c
0641            work_file( 7 : 11 ) = 'INPUT'
0642            open (unit=data_unit1,file='6000$RT:[CHUCK.PARAMS]'//work_file,
0643            1       status='OLD',organization='SEQUENTIAL',access='SEQUENTIAL',
0644            2       form='UNFORMATTED',recordtype='FIXED',recl=64,dispose='KEEP',
0645            3                                 iostat=status,READONLY,err=50010)
0646            read (unit=data_unit1,err=50100)
0647            if ( .not. ( readarr( %val( ant_add ), dr_type, mi_ants, 2,
0648            1                                 data_unit1, 32 ) ) ) go to 50100
0649            if ( .not. ( readarr( %val( wave_add ), dr_type, mi_waves, 9,
0650            1                                 data_unit1, 32 ) ) ) go to 50100
0651            close (unit=data_unit1)
```

```
MUSIC_PEST                                                    18-Apr-1988 14:10:36     \
                                                               2-Apr-1988 14:46:06     [

0652     c
0653     c               Get observation and noise covariance matrices from the ancestor
0654     c       experiment's "eigenvalues" file.
0655     c
0656             work_file( 7 : 11 ) = 'EIGVS'
0657             open (unit=data_unit1,file='6000$RT:[CHUCK.PARAMS]'//work_file,
0658             1       status='OLD',organization='SEQUENTIAL',access='SEQUENTIAL',
0659             2       form='UNFORMATTED',recordtype='FIXED',recl=64,dispose='KEEP',
0660             3                               iostat=status,READONLY,err=50010)
0661             if ( .not. ( readarr( %val( robs_add ), dc_type, mi_ants, mi_ants,
0662             1                               data_unit1, 16 ) ) ) go to 50100
0663             if ( .not. ( readarr( %val( rnoise_add ), dc_type, mi_ants, mi_ants,
0664             1                               data_unit1, 16 ) ) ) go to 50100
0665             close (unit=data_unit1)
0666     c
0667     c               For compensated estimation types, subtract the noise samples
0668     c       from the observation samples (replacing the observation samples with
0669     c       the differences).
0670     c
0671             if ( ( estim .eq. 3 )
0672             1       .or. ( estim .eq. 5 )
0673             2       .or. ( estim .eq. 6 ) ) then
0674               call c16_sel_vdiff( %val( obsmat_add ), 1,
0675             1                     %val( noisemat_add ), 1, mi_smp * mi_ants,
0676             2                                           %val( obsmat_add ) )
0677             end if
0678     c
0679     c               Form estimate of A matrix.
0680     c
0681             call sigmat_estm( %val( ant_add ), mi_ants, %val( doa_add ), mi_waves,
0682             1                                   mi_wlen, %val( amat_add ) )
0683     c
0684     c               Compute actual sample P matrix from original signal vectors,
0685     c       and L2 P matrix estimate.
0686     c
0687             tstatus = lib$init_timer()
0688             if ( .not. tstatus ) call lib$stop( %val( tstatus ) )
0689     c
0690             call dc_matmpyherm( %val( sigvm_add ), mi_waves, mi_smp,
0691             1                     %val( sigvm_add ), mi_waves, %val( pmat_add ) )
0692             call sqdc_mat_intdiv( %val( pmat_add ), mi_waves, mi_smp )
0693             call music_l2p( %val( amat_add ), mi_ants, mi_waves,
0694             1                     %val( robs_add ), %val( rnoise_add ),
0695             2                           mi_tol, %val( pl2_add ), l2status )
0696     c
0697             tstatus = lib$stat_timer( 2, l2tics )
0698             if ( .not. tstatus ) call lib$stop( %val( tstatus ) )
0699     c
0700     c               Form estimates of signal and residual vectors and P matrix as
0701     c       selected by the estimation type specified for the experiment.
0702     c
0703             tstatus = lib$init_timer()
0704             if ( .not. tstatus ) call lib$stop( %val( tstatus ) )
0705     c
0706             if ( l2status .eq. 0 ) then     !  Go on only if L2 estimate succeeded.
0707               if      ( ( estim .eq. 1 )                    !  Enumeration.
0708             1               .or. ( estim .eq. 3 ) ) then    !  Compensated enum.
```

```
MUSIC_PEST                                                    18-Apr-1988 14:10:36     VA
                                                               2-Apr-1988 14:46:06     [C
0709                  call music_lpp_each( %val( obsmat_add ), mi_smp, mi_ants,
0710                1                      %val( amat_add ), mi_waves, p, mi_peps,
0711                2                      mi_norm, mi_tol, %val( pvm_add ),
0712                3                      %val( resm_add ), mi_irlslim, mnitlim,
0713                4                                                atitlim, min_del )
0714                  else if ( ( estim .eq. 2 )                     !  Sum of powers.
0715                1               .or. ( estim .eq. 5 ) ) then     !  Compensated sum pwr.
0716                  call music_lpw_yar( %val( obsmat_add ), mi_smp, mi_ants,
0717                1                     %val( amat_add ), mi_waves, p, mi_peps,
0718                2                     mi_norm, mi_tol, %val( yarw_add ),
0719                3                     %val( pvm_add ), %val( resm_add ),
0720                4                       mi_irlslim, mnitlim, min_del, 1 )
0721                  else if ( ( estim .eq. 4 )                     !  Sum of powers loss.
0722                1               .or. ( estim .eq. 6 ) ) then     !  Comp'd sum pwr loss.
0723                  call music_lpw_yar( %val( obsmat_add ), mi_smp, mi_ants,
0724                1                     %val( amat_add ), mi_waves, p, mi_peps,
0725                2                     mi_norm, mi_tol, %val( yarw_add ),
0726                3                     %val( pvm_add ), %val( resm_add ),
0727                4                       mi_irlslim, mnitlim, min_del, 2 )
0728                  else                                           !  Invalid method.
0729                    go to 50300
0730                  end if
0731                  if ( ( estim .eq. 1 ) .or. ( estim .eq. 3 ) ) then     !  Enum. type.
0732                    call dc_matmpyherm( %val( pvm_add ), mi_waves, mi_smp,
0733                1                       %val( pvm_add ), mi_waves, %val( plp_add ) )
0734                    call sqdc_mat_intdiv( %val( plp_add ), mi_waves, mi_smp )
0735                  else                                           !  Sum of powers type.
0736                    call music_lpp_speis( %val( amat_add ), mi_ants, mi_waves,
0737                1                         %val( yarw_add ), %val( robs_add ),
0738                2                         %val( rnoise_add ), mi_tol,
0739                3                                          %val( plp_add ), status )
0740                    if ( ( status .ne. 0 ) .and. ( mnitlim .gt. 0 ) )
0741                1                                             mnitlim = - mnitlim - 1
0742                  end if
0743                end if
0744          c
0745                tstatus = lib$stat_timer( 2, lptics )
0746                if ( .not. tstatus ) call lib$stop( %val( tstatus ) )
0747          c
0748          c                 Finally report results (even if L2 estimate failed).
0749          c
0750                open (unit=data_unit1,file=found_file,status='OLD',
0751                1       organization='SEQUENTIAL',access='DIRECT',form='UNFORMATTED',
0752                2       recordtype='FIXED',recl=8,dispose='KEEP',READONLY,
0753                3                                                 iostat=status)
0754                call music_p_rep( l2status, status+1, %val( doa_add ),
0755                1                 %val( expnum_add ), data_unit1, data_unit2,
0756                2                 mi_ofile, mi_dtim, experiment, ancestor, exps,
0757                3                 %val( pmat_add ), %val( pl2_add ), mi_waves, p,
0758                4                 mi_irlslim, mnitlim, atitlim, %val( plp_add ),
0759                5                 l2tics / 100.0d0, lptics / 100.0d0, estim, min_del )
0760                close (unit=data_unit1)
0761                call dump_va( vstatus, msg )
0762                call exit_va( vstatus, msg )
0763          c
0764          c                 Succeeded processing input file.  Clear error indicator,
0765          c       delete used input file, and continue search for more input files.
```

```
0766    c
0767    50130   continue                                ! Failed reading data file.
0768            fails = 0
0769            call lib$delete_file( found_file )
0770            go to 21000
0771    c
0772    c****************************** Error branches. ******************************
0773    c
0774    c               Error opening data file.  It may be in use by another program.
0775    c
0776    50000   continue                                ! Data unit 1 & VA open.
0777            close (unit=data_unit1)
0778    c
0779    50010   continue                                ! VA open.
0780            call dump_va( vstatus, msg )
0781            call exit_va( vstatus, msg )
0782    c
0783    50020   continue                                ! Nothing open.
0784            if ( fails .ge. fails_lim ) then        ! Failed too many times.
0785              type *
0786              type *, 'Failed opening input data file(s) ', fails_lim, ' times.'
0787              go to 60620
0788            else                                    ! Haven't failed too many.
0789              fails = fails + 1
0790              go to 21000
0791            end if
0792    c
0793    c               Error reading opened intermediate data file.
0794    c
0795    50100   continue
0796            close (unit=data_unit1)
0797            call dump_va( vstatus, msg )
0798            call exit_va( vstatus, msg )
0799    c
0800            type *
0801            type *, 'Failure reading data file:  ', work_file
0802    50110   continue
0803            type *, 'while processing input file:'
0804            type *, found_file
0805            type *
0806            type *, 'Deleting the input file.'
0807            go to 50130
0808    c
0809    c               Failure performing internal write.
0810    c
0811    50200   continue
0812            type *
0813            type *, 'Failed performing internal write.'
0814            go to 60620
0815    c
0816    c               Invalid estimation method.
0817    c
0818    50300   continue
0819            call dump_va( vstatus, msg )
0820            call exit_va( vstatus, msg )
0821    c
0822            type *
```

```
MUSIC_PEST                                                    18-Apr-1988 14:10:36     V
                                                               2-Apr-1988 14:46:06     [

0823              type *, 'Invalid estimation method encountered'
0824              go to 50110
0825     c
0826     90020    format( i10.10 )
0827     c
0828              end


PROGRAM SECTIONS

    Name                                  Bytes   Attributes

  0 $CODE                                  3041   PIC CON REL LCL   SHR   EXE   RD NOWRT L
  1 $PDATA                                  616   PIC CON REL LCL   SHR NOEXE   RD NOWRT L
  2 $LOCAL                                 2660   PIC CON REL LCL NOSHR NOEXE   RD   WRT
  3 MUSIC_MEASURES                          120   PIC OVR REL GBL   SHR NOEXE   RD   WRT L

    Total Space Allocated                  6437


ENTRY POINTS

    Address  Type  Name

  0-00000000       MUSIC_PEST


VARIABLES

    Address  Type  Name              Address  Type  Name              Address  Type  Name

  2-000002D4  I*4  AMAT_ADD        2-00000288  I*4  ANCESTOR            **    I*4  ANT
  2-00000308  I*4  ATITLIM         3-0000000C  I*4  CYCLES           2-00000290  I*4  DAT
  2-000002C4  I*4  DOA_ADD         2-000002C0  I*4  ESTIM           2-000002B4  I*4  EXP
  2-000002BC  I*4  EXPS               **       I*4  FAILS              **       I*4  FAI
  2-00000288  L*4  FOUND           2-0000010D  CHAR FOUND_FILE       2-0000028C  I*4  IND
  2-0000030C  I*4  L2TICS          2-00000310  I*4  LPTICS           2-00000240  R*8  MIN
  2-00000250  I*4  MI_ANTS         2-00000278  R*4  MI_BEGP          2-00000280  R*4  MI_
  2-00000041  CHAR MI_DTIM         2-0000020E  I*2  MI_EERR          2-0000027C  R*4  MI_
  2-0000020C  I*2  MI_IERR         2-00000268  I*4  MI_IRLSLIM       2-00000258  I*4  MI_
  2-00000054  CHAR MI_OFILE        2-0000024C  I*4  MI_PARK          2-00000284  R*4  MI_
  2-00000248  I*4  MI_PKEY         2-00000210  I*2  MI_SERR          2-00000218  R*8  MI_
  2-00000040  CHAR MI_STAT         2-00000238  R*8  MI_TOL           2-00000254  I*4  MI_
  2-00000230  R*8  MI_WLEN         2-0000026C  I*4  MI_ZEIGS         2-00000304  I*4  MNI
  2-000002DC  I*4  NOISEMAT_ADD    3-00000060  R*8  NON_TRI          3-00000068  I*4  NON
  2-00000314  R*4  P               2-000002B0  I*4  PARENT           2-000002E8  I*4  PL2
  2-000002F0  I*4  PMAT_ADD        2-000002E0  I*4  PVM_ADD          3-0000006C  R*8  QPS
  2-000002E4  I*4  RESM_ADD        2-000002FC  I*4  RNOISE_ADD       2-000002F8  I*4  ROB
  2-000002F4  I*4  SIGVM_ADD       2-00000298  I*4  SPAWN_UNIT          **       CHAR SP_
     **       CHAR SP_PROC            **       I*4  SP_STAT         2-0000029C  I*4  STA
  3-00000000  R*8  TOL                 **       I*4  TSTATUS         3-00000018  R*8  UPH
  3-00000020  R*8  UPHI_SMARG      3-00000034  I*4  UPHI_S_ZCYC      3-00000028  R*8  U_U
  3-0000003C  R*8  VPSI_LMARG      3-00000054  I*4  VPSI_L_ZCYC      3-00000044  R*8  VPS
  2-000002A0  I*4  VSTATUS         3-0000004C  R*8  V_UNIERR         3-0000005C  I*4  V_U
     **       I*4  WAITS_LIM          **       I*4  WAVES           2-000002CC  I*4  WAV
```

```
                                                    18-Apr-1988 14:12:16     VAX
                                                    14-Apr-1988 08:40:56     BRA

0001            subroutine bracket_doa( doa, apars, num_ants, wavelen, eigvec,
0002            1                       num_zeigs, p, down, pwr, lower, upper, perf )
0003    c
0004    c-------------------------------------------------------------------------------
0005    c
0006    c               This routine computes the sharpness measure for a peak in a
0007    c       MUSIC "power" spectrum.
0008    c
0009    c       Parameters:
0010    c       -----------
0011    c               num_ants:   INTEGER*4 number of antenna elements.
0012    c               num_zeigs:   INTEGER*4 number of eigenvectors to use in
0013    c                                   computing the MUSIC power spectrum.
0014    c               p:  REAL*4 value of the exponent to which to raise the terms in
0015    c                                   the MUSIC power spectrum expression
0016    c                                   ( 1 <= p <= 3 ).
0017    c               doa:  REAL*8 direction of arrival estimate (RADIANS) of the
0018    c                                   peak for which the sharpness measure is
0019    c                                   to be computed.  This need not actually
0020    c                                   identify a peak location; the routine
0021    c                                   locates the lower and upper angles
0022    c                                   which produce MUSIC spectrum values
0023    c                                   that are (down)db down from the value
0024    c                                   at angle doa.
0025    c               apars:  num_ants row REAL*8 array of antenna element
0026    c                                   parameters.  The first column of apars
0027    c                                   contains element ranges from the origin
0028    c                                   and the second column contains element
0029    c                                   angles (in RADIANS) from the abscissa.
0030    c               wavelen:  REAL*8 carrier wavelength to be used in computing the
0031    c                                   MUSIC power spectrum.  This must be in
0032    c                                   the same units as the antenna distances
0033    c                                   from the origin (apars(*,1)).
0034    c               down:  REAL*8 number of db down from the MUSIC spectrum peak
0035    c                                   value at which the lower and upper
0036    c                                   "bracket" angles are to be identified.
0037    c               eigvec:  num_ants x num_ants COMPLEX*16 matrix whose first
0038    c                                   num_zeigs columns are the "noise"
0039    c                                   eigenvectors to be used for computing
0040    c                                   MUSIC power spectrum values.
0041    c               pwr:  REAL*8 output value of the MUSIC spectrum at angle doa.
0042    c               lower:  REAL*8 output value of the largest angle (RADIANS) less
0043    c                                   than doa (but >= - pi) at which the
0044    c                                   MUSIC spectrum value is (down)db down
0045    c                                   from pwr.
0046    c               upper:  REAL*8 output value of the smallest angle (RADIANS)
0047    c                                   greater than doa (but <= pi) at which
0048    c                                   the MUSIC spectrum value is (down)db
0049    c                                   down from pwr.
0050    c               perf:  REAL*8 output value of the sharpness measure for the
0051    c                                   peak at doa.
0052    c
0053    c-------------------------------------------------------------------------------
0054    c
0055            implicit none
0056            intrinsic dlog10
0057            external open_va, exit_va, free_va
```

```
BRACKET_DOA                                                    18-Apr-1988 14:12:16    VAX FO
                                                               14-Apr-1988 08:40:56    BRACKE

0058            external music_power, sigval
0059            include '6000$RT:[CHUCK.RESEARCH.FORTDIRIVATYPES.TXT'
0071            include '(real_constants)'
0122      c
0123            real*8 doadel
0124            parameter ( doadel = rad_p_deg8 )
0125      c
0126            real*8 music_power
0127      c
0128            integer*4 num_ants, num_zeigs
0129            real*4 p
0130            real*8 doa, apars( num_ants, 1 ), pwr, lower
0131            real*8 wavelen, upper, perf, down
0132            complex*16 eigvec( num_ants, num_ants )
0133      c
0134            integer*4 steerv_add, linear(1), status
0135            real*8 dpwr
0136            character*80 msg
0137      c
0138            linear(1) = num_ants
0139            call open_va( steerv_add, linear, 1, dc_type, status, msg )
0140            call exit_va( status, msg )
0141      c
0142            call sigval( apars, num_ants, doa, wavelen, %val( steerv_add ) )
0143            pwr = music_power( eigvec, %val( steerv_add ), num_ants, num_zeigs, p )
0144            dpwr = pwr * ( 10.0d0 ** ( - down / 10.0d0 ) )
0145      c
0146            lower = doa - doadel
0147            do while ( lower .gt. - pi8 )
0148              call sigval( apars, num_ants, lower, wavelen, %val( steerv_add ) )
0149              if ( music_power( eigvec, %val( steerv_add ), num_ants,
0150            1                                                    num_zeigs, p )
0151            2       .le. dpwr ) go to 10000
0152              lower = lower - doadel
0153            end do
0154      10000 continue
0155      c
0156            upper = doa + doadel
0157            do while ( upper .lt. pi8 )
0158              call sigval( apars, num_ants, upper, wavelen, %val( steerv_add ) )
0159              if ( music_power( eigvec, %val( steerv_add ), num_ants,
0160            1                                                    num_zeigs, p )
0161            2       .le. dpwr ) go to 10010
0162              upper = upper + doadel
0163            end do
0164      10010 continue
0165      c
0166            if ( upper .gt. lower ) then
0167              perf = dlog10( pwr ) / ( p * ( upper - lower ) )
0168            else
0169              perf = -1.0d0
0170            end if
0171      c
0172            call free_va( steerv_add, status, msg )
0173            call exit_va( status, msg )
0174      c
0175            return
```

```
                                                    18-Apr-1988 14:12:53     V
                                                    28-Mar-1988 20:34:40     C

0001            subroutine c16_vachol( inmat, size, outmat, condit, status )
0002    c
0003    c-------------------------------------------------------------------------------
0004    c
0005    c                       This subroutine uses LINPAC routine ZPOCO, and routines from
0006    c       the Virtual Array system (copyright 1987 by Dwight Day), to perform a
0007    c       Cholesky decomposition of the COMPLEX*16 size x size input matrix
0008    c       inmat.  The Cholesky factor is returned in the size x size COMPLEX*16
0009    c       output matrix outmat.  If the factorization succeeded (inmat is
0010    c       positive definite), the INTEGER*4 parameter status is set to 0 and the
0011    c       REAL*8 parameter condit is set to an estimate of the inverse of the
0012    c       condition number of inmat.  If factorization failed, status is set to
0013    c       a value > 0 (size of the leading submatrix of inmat that is not
0014    c       positive definite), and condit is not changed.
0015    c                       The Cholesky decomposition of inmat produces an upper
0016    c       triangular matrix outmat such that:
0017    c                               inmat = outmat-conjugate-transpose * outmat.
0018    c                       Since inmat is Hermitian, the strictly lower triangle of inmat
0019    c       is NEVER REFERENCED.  The strictly lower triangle of outmat is SET TO
0020    c       ZERO.  The input matrix inmat is NOT CHANGED by this routine.
0021    c
0022    c-------------------------------------------------------------------------------
0023    c
0024    c       implicit none
0025            external zpoco, open_va, exit_va, free_va
0026            include '6000$RT:[CHUCK.RESEARCH.FORTDIR]VATYPES.TXT'
0038    c
0039            integer*4 size, status
0040            real*8 condit
0041            complex*16 inmat( size, size ), outmat( size, size )
0042    c
0043            integer*4 i, j
0044            integer*4 work_add, size_lin( 1 ), vstatus
0045            character*80 msg
0046    c
0047    c                       Copy input upper triangle to output upper triangle, since
0048    c       ZPOCO puts its output in its input.
0049    c
0050            do i = 1, size
0051              do j = 1, i - 1
0052                outmat(i,j) = ( 0.0d0, 0.0d0 )
0053              end do
0054              do j = i, size
0055                outmat(i,j) = inmat(i,j)
0056              end do
0057            end do
0058    c
0059    c                       Make a work vector for ZPOCO.
0060    c
0061            size_lin(1) = size
0062            call open_va( work_add, size_lin, 1, dc_type, vstatus, msg )
0063            call exit_va( vstatus, msg )
0064    c
0065    c                       Now let ZPOCO do the real work.
0066    c
0067            call zpoco( outmat, size, size, condit, %val( work_add ), status )
0068    c
```

```
                                                    18-Apr-1988 14:12:53     V
                                                    28-Mar-1988 20:34:40     C

0001            subroutine c16_vachol( inmat, size, outmat, condit, status )
0002    c
0003    c-------------------------------------------------------------------------------
0004    c
0005    c                       This subroutine uses LINPAC routine ZPOCO, and routines from
0006    c       the Virtual Array system (copyright 1987 by Dwight Day), to perform a
0007    c       Cholesky decomposition of the COMPLEX*16 size x size input matrix
0008    c       inmat.  The Cholesky factor is returned in the size x size COMPLEX*16
0009    c       output matrix outmat.  If the factorization succeeded (inmat is
0010    c       positive definite), the INTEGER*4 parameter status is set to 0 and the
0011    c       REAL*8 parameter condit is set to an estimate of the inverse of the
0012    c       condition number of inmat.  If factorization failed, status is set to
0013    c       a value > 0 (size of the leading submatrix of inmat that is not
0014    c       positive definite), and condit is not changed.
0015    c                       The Cholesky decomposition of inmat produces an upper
0016    c       triangular matrix outmat such that:
0017    c                               inmat = outmat-conjugate-transpose * outmat.
0018    c                       Since inmat is Hermitian, the strictly lower triangle of inmat
0019    c       is NEVER REFERENCED.  The strictly lower triangle of outmat is SET TO
0020    c       ZERO.  The input matrix inmat is NOT CHANGED by this routine.
0021    c
0022    c-------------------------------------------------------------------------------
0023    c
0024    c       implicit none
0025            external zpoco, open_va, exit_va, free_va
0026            include '6000$RT:[CHUCK.RESEARCH.FORTDIR]VATYPES.TXT'
0038    c
0039            integer*4 size, status
0040            real*8 condit
0041            complex*16 inmat( size, size ), outmat( size, size )
0042    c
0043            integer*4 i, j
0044            integer*4 work_add, size_lin( 1 ), vstatus
0045            character*80 msg
0046    c
0047    c                       Copy input upper triangle to output upper triangle, since
0048    c       ZPOCO puts its output in its input.
0049    c
0050            do i = 1, size
0051              do j = 1, i - 1
0052                outmat(i,j) = ( 0.0d0, 0.0d0 )
0053              end do
0054              do j = i, size
0055                outmat(i,j) = inmat(i,j)
0056              end do
0057            end do
0058    c
0059    c                       Make a work vector for ZPOCO.
0060    c
0061            size_lin(1) = size
0062            call open_va( work_add, size_lin, 1, dc_type, vstatus, msg )
0063            call exit_va( vstatus, msg )
0064    c
0065    c                       Now let ZPOCO do the real work.
0066    c
0067            call zpoco( outmat, size, size, condit, %val( work_add ), status )
0068    c
```

18-Apr-1988 14:13:22 VAX
24-Sep-1987 11:45:08 C8_

```
0001            subroutine c8_noise( noise, size, defs, pars )
0002      c
0003      c-------------------------------------------------------------------------------
0004      c
0005      c                This subroutine computes a COMPLEX*8 noise "sources" vector.
0006      c
0007      c       Parameters:
0008      c       -----------
0009      c                noise = COMPLEX*8 size-element output noise vector.
0010      c                size = INTEGER*4 number of noise sources in use.
0011      c                defs = INTEGER*4 size x 2 matrix whose first column is the
0012      c                                 vector of noise source types (see subroutine
0013      c                                 READ_NOISE), and whose second column is the
0014      c                                 vector of counts for intermittent noise
0015      c                                 sources.
0016      c                pars = REAL*4 size x 3 matrix of noise source parameters (see
0017      c                                 subroutine READ_NOISE).
0018      c
0019      c-------------------------------------------------------------------------------
0020      c
0021            implicit none
0022            intrinsic cmplx, jishft, jiand
0023            external gauss, uni
0024      c
0025            real*4 uni
0026      c
0027            integer*4 size, defs( size, 1 )
0028            real*4 pars( size, 1 )
0029            complex*8 noise( size )
0030      c
0031            integer*4 i, it1
0032            real*4 rnoise, inoise
0033      c
0034            do i = 1, size
0035              it1 = jishft( defs(i,1), -16 )
0036              if ( ( ( it1 .ne. 0) .and. ( defs(i,2) .eq. 1 ) ) .or.
0037           1                                             ( it1 .eq. 0 ) ) then
0038                 it1 = jiand( defs(i,1), 65536 )
0039                 if      ( it1 .eq. 1 ) then
0040                   noise(i) = cmplx( pars(i,1), pars(i,2) )
0041                 else if ( it1 .eq. 2 ) then
0042                   noise(i) = cmplx( pars(i,1) + pars(i,2) * ( uni( 10571 ) - 0.5 ),
0043           1                         pars(i,1) + pars(i,2) * ( uni( 10571 ) - 0.5 )
0044           2                         )
0045                 else
0046                   call gauss( rnoise, inoise, pars(i,2), pars(i,1), 0 )
0047                   noise(i) = cmplx( rnoise, inoise )
0048                 end if
0049               else
0050                 noise(i) = ( 0.0, 0.0 )
0051               end if
0052            end do
0053      c
0054            return
0055            end
```

```
                                                    18-Apr-1988 14:13:58     V
                                                    16-Mar-1988 09:30:14     D
0001            subroutine dcv_irls( a, n, m, x, p, eps, norm,
0002            1                                        tol, itlim, f, resid, del )
0003      c
0004      c-------------------------------------------------------------------------------
0005      c
0006      c               This subroutine forms an approximate Lp estimate of a vector
0007      c       f which satisfies the equation x = a f + noise.  The IRLS algorithm
0008      c       is used to determine the f estimate.
0009      c
0010      c       Parameters:
0011      c       -----------
0012      c               a:  n x m COMPLEX*16 coefficient matrix.
0013      c               n:  INTEGER*4 number of rows in the coefficient matrix (usually
0014      c                                   the number of observations obtained).
0015      c               m:  INTEGER*4 number of elements in the vector to be estimated
0016      c                                   (problem order).
0017      c               x:  n-element COMPLEX*16 observation vector.
0018      c               p:  REAL*4 power of error magnitudes to be minized (an Lp
0019      c                                   estimate is computed).
0020      c               eps:  REAL*8 minimum residual magnitude to be used (see
0021      c                                   subroutine get_w).
0022      c               norm:  REAL*8 value to which to normalize the SQUARE ROOT OF
0023      c                                   the maximum weight.  If norm <= 0.0d0,
0024      c                                   no normalization occurs.  Otherwise,
0025      c                                   all weight SQUARE ROOT values (w(i))
0026      c                                   are divided by a value which results in
0027      c                                   a maximum weight SQUARE ROOT of norm.
0028      c               tol:  REAL*8 value to be taken for zero.  The routine exits
0029      c                                   when the SQUARE of the norm of the
0030      c                                   difference between successive f
0031      c                                   approximations drops below tol.  NOTE
0032      c                                   that tol expresses a FRACTIONAL
0033      c                                   tolerance, that is, || difference || /
0034      c                                   || sum || <= tol is the exit criterion.
0035      c               itlim:  INTEGER*4 number of iterations permitted.  This
0036      c                                   varialble is COUNTED DOWN at each
0037      c                                   iteration, and the routine exits if it
0038      c                                   reaches zero (regardless of
0039      c                                   convergence).  Also, the variable is
0040      c                                   SET to the negative of its current
0041      c                                   value minus one and the routine exits
0042      c                                   IMMEDIATELY if the routine DC_MPPSINV
0043      c                                   fails ("info", below, returned with a
0044      c                                   non-zero value).
0045      c               f:  m-element COMPLEX*16 computed estimate vector.
0046      c               resid:  n-element vector of residuals from the last computed
0047      c                                   estimate.  resid = x - a f.
0048      c               del:  REAL*8 exit value of the square of the norm of the
0049      c                                   difference between successive estimates
0050      c                                   of f.
0051      c
0052      c               NOTE that an L2 estimate of f is produced if this routine is
0053      c       called with itlim = 0 (regardless of the passed value of p).
0054      c
0055      c-------------------------------------------------------------------------------
0056      c
0057            implicit none
```

```
0058            external open_va, free_va, exit_va
0059            external dc_mppsinv, dc_matvmpy, residuals, dc_veccopy, get_w_half
0060            external drdi_dc_mmpy, drdi_dc_vmpy, dc_vdel_nmsq
0061      c
0062            real*8 dc_vdel_nmsq
0063      c
0064            integer*4 n, m, itlim
0065            real*4 p
0066            real*8 eps, norm, tol, del
0067            complex*16 a( n, m ), x( n ), f( m ), resid( n )
0068      c
0069            integer*4 prevf_add, mlin( 1 ), dc_type/7/, status
0070            integer*4 w_add, nlin( 1 ), dr_type/5/
0071            integer*4 psinv_add, m_by_n( 2 )
0072            integer*4 wx_add, wa_add, n_by_m( 2 )
0073            integer*4 info
0074            character*80 msg
0075      c
0076            mlin(1)   = m
0077            nlin(1)   = n
0078            m_by_n(1) = m
0079            m_by_n(2) = n
0080            n_by_m(1) = n
0081            n_by_m(2) = m
0082      c
0083      c             Get space for various working arrays.
0084      c
0085            call open_va( prevf_add, mlin, 1, dc_type, status, msg )
0086            call exit_va( status, msg )
0087            call open_va( w_add, nlin, 1, dr_type, status, msg )
0088            call exit_va( status, msg )
0089            call open_va( psinv_add, m_by_n, 2, dc_type, status, msg )
0090            call exit_va( status, msg )
0091            call open_va( wx_add, nlin, 1, dc_type, status, msg )
0092            call exit_va( status, msg )
0093            call open_va( wa_add, n_by_m, 2, dc_type, status, msg )
0094            call exit_va( status, msg )
0095      c
0096      c             Compute L2 estimate of f, and residuals therefor.
0097      c
0098            del = tol + .5
0099            call dc_mppsinv( a, n, m, tol, %val( psinv_add ), info )
0100            if ( info .ne. 0 ) then                 !  Error in psuedo-inverse.
0101              itlim = - itlim - 1
0102              go to 10000
0103            end if
0104            call dc_matvmpy( %val( psinv_add ), m, n, x, f )
0105            call residuals( x, n, a, m, f, resid )
0106      c
0107      c             Exit if L2 solution desired (itlim = 0); otherwise start
0108      c       iterating to Lp solution.
0109      c
0110            do while ( ( del .gt. tol ) .and. ( itlim .gt. 0 ) )
0111              call dc_veccopy( f, m, %val( prevf_add ) )
0112      c
0113              call get_w_half( resid, n, p, eps, norm, %val( w_add ) )
0114                                                    !  Note using SQRT( weights ).
```

```
0115              call drdi_dc_mmpy( %val( w_add ), n, a, m, %val( wa_add ) )
0116              call dc_mppsinv( %val( wa_add ), n, m, tol, %val( psinv_add ), info )
0117              if ( info .ne. 0 ) then                !  Error in psuedo-inverse.
0118                itlim = - itlim - 1
0119                go to 10000
0120              end if
0121              call drdi_dc_vmpy( %val( w_add ), n, x, %val( wx_add ) )
0122              call dc_matvmpy( %val( psinv_add ), m, n, %val( wx_add ), f )
0123              call residuals( x, n, a, m, f, resid )
0124     c
0125              itlim = itlim - 1
0126              del = dc_vdel_nmsq( f, %val( prevf_add ), m, tol )
0127            end do
0128     c
0129     c                Dump the working array space.
0130     c
0131     10000  continue
0132            call free_va( prevf_add, status, msg )
0133            call exit_va( status, msg )
0134            call free_va( w_add, status, msg )
0135            call exit_va( status, msg )
0136            call free_va( psinv_add, status, msg )
0137            call exit_va( status, msg )
0138            call free_va( wx_add, status, msg )
0139            call exit_va( status, msg )
0140            call free_va( wa_add, status, msg )
0141            call exit_va( status, msg )
0142     c
0143            return
0144            end
```

```
                                                    18-Apr-1988 14:13:58     VA)
                                                    16-Mar-1988 09:30:14     DC\
0001    c
0002            subroutine residuals( x, n, a, m, f, resid )
0003    c
0004    c-------------------------------------------------------------------------------
0005    c
0006    c               This subroutine calculates the n-element COMPLEX*16 vector
0007    c       (resid) of residuals for an estimate of f.  x is n-element COMPLEX*16,
0008    c       a is n x m COMPLEX*16, n and m are INTEGER*4, and f is m-element
0009    c       COMPLEX*16.
0010    c               The formula used is resid = x - a f.
0011    c
0012    c-------------------------------------------------------------------------------
0013    c
0014            implicit none
0015            external dc_matvmpy
0016    c
0017            integer*4 n, m
0018            complex*16 x( n ), a( n, m ), f( m ), resid( n )
0019    c
0020            integer*4 i
0021    c
0022            call dc_matvmpy( a, n, m, f, resid )
0023            do i = 1, n
0024              resid(i) = x(i) - resid(i)
0025            end do
0026    c
0027            return
0028            end
```

18-Apr-1988 14:15:16 VA)
17-Nov-1987 08:45:17 DC_

```
0001            subroutine dc_mppsinv( a, n, m, fuzz, psinv, info )
0002      c
0003      c-------------------------------------------------------------------------------
0004      c
0005      c               This subroutine forms the Moore-Penrose psuedo-inverse of a
0006      c       COMPLEX*16 matrix.  A Singular Value Decomposition (SVD) algorithm
0007      c       is used, so that no explicit inversion is required; thus the matrix to
0008      c       be "inverted" need not be of full rank (or, for that matter, square).
0009      c       The Moore-Penrose psuedo-inverse of a is computed as:
0010      c       psinv = ( a_conjugate_transpose * a )-inverse * a_conjugate_transpose.
0011      c
0012      c       Parameters:
0013      c       -----------
0014      c               a:   n x m COMPLEX*16 matrix for which the psuedo-inverse is to
0015      c                                     be computed.
0016      c               n, m:   INTEGER*4 number of rows and columns, respectively, in
0017      c                                     the a matrix.
0018      c               fuzz:   REAL*8 value to be taken as zero ("tolerance") (used
0019      c                                     to determine when a singular value is
0020      c                                     "almost" zero).
0021      c               psinv:   m x n COMPLEX*16 computed psuedo-inverse.
0022      c               info:   INTEGER*4 status indicator (output); 0 for successful
0023      c                                     psuedo-inversion, non-0 on failure of
0024      c                                     singular value decomposition.
0025      c
0026      c-------------------------------------------------------------------------------
0027      c
0028            implicit none
0029            intrinsic jmin0
0030            external open_va, free_va, exit_va
0031            external zsvdc
0032            external dc_matcopy, dc_matmpyherm, dcsv_vsigm1
0033      c
0034            integer*4 n, m, info
0035            real*8 fuzz
0036            complex*16 a( n, m ), psinv( m, n )
0037      c
0038            integer*4 dc_type/7/, status
0039            integer*4 n_lin( 1 ), n_sqr( 2 ), m_lin( 1 ), m_sqr( 2 )
0040            integer*4 n_by_m( 2 ), m_by_n( 2 ), slen( 1 )
0041            integer*4 aa_add, s_add, e_add, u_add, v_add, wk1_add, wk2_add
0042            character*80 msg
0043      c
0044            n_lin(1)  = n
0045            n_sqr(1)  = n
0046            n_sqr(2)  = n
0047            m_lin(1)  = m
0048            m_sqr(1)  = m
0049            m_sqr(2)  = m
0050            n_by_m(1) = n
0051            n_by_m(2) = m
0052            m_by_n(1) = m
0053            m_by_n(2) = n
0054            slen(1)   = jmin0( n + 1, m )
0055      c
0056            call open_va( aa_add, n_by_m, 2, dc_type, status, msg )
0057            call exit_va( status, msg )
```

DC_MPPSINV 18-Apr-1988 14:15:16
17-Nov-1987 08:45:17

```
0058            call open_va( s_add, slen, 1, dc_type, status, msg )
0059            call exit_va( status, msg )
0060            call open_va( e_add, m_lin, 1, dc_type, status, msg )
0061            call exit_va( status, msg )
0062            call open_va( u_add, n_sqr, 2, dc_type, status, msg )
0063            call exit_va( status, msg )
0064            call open_va( v_add, m_sqr, 2, dc_type, status, msg )
0065            call exit_va( status, msg )
0066            call open_va( wk1_add, n_lin, 1, dc_type, status, msg )
0067            call exit_va( status, msg )
0068            call open_va( wk2_add, m_by_n, 2, dc_type, status, msg )
0069            call exit_va( status, msg )
0070    c
0071            call dc_matcopy( a, n, m, %val( aa_add ) )
0072            call zsvdc( %val( aa_add ), n, n, m, %val( s_add ), %val( e_add ),
0073            1                                  %val( u_add ), n, %val( v_add ), m,
0074            2                                         %val( wk1_add ), 11, info )
0075            if ( info .ne. 0 ) go to 10000
0076    c
0077            call dcsv_vsigm1( %val( v_add ), m, %val( s_add ), n, fuzz,
0078            1                                                  %val( wk2_add ) )
0079            call dc_matmpyherm( %val( wk2_add ), m, n, %val( u_add ), n, psinv )
0080    c
0081    10000   continue
0082            call free_va( aa_add, status, msg )
0083            call exit_va( status, msg )
0084            call free_va( s_add, status, msg )
0085            call exit_va( status, msg )
0086            call free_va( e_add, status, msg )
0087            call exit_va( status, msg )
0088            call free_va( u_add, status, msg )
0089            call exit_va( status, msg )
0090            call free_va( v_add, status, msg )
0091            call exit_va( status, msg )
0092            call free_va( wk1_add, status, msg )
0093            call exit_va( status, msg )
0094            call free_va( wk2_add, status, msg )
0095            call exit_va( status, msg )
0096    c
0097            return
0098            end
```

```
c
        subroutine dcsv_vsigm1( v, m, sigma, n, zero, product )
c
c<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
c
c               This subroutine forms the product v * sigma1.  v is an m x m
c       COMPLEX*16 matrix which is generally produced by a single value
c       decomposition algorithm (v is usually the left singular vectors).  m
c       is INTEGER*4.  sigma is a min( m, n ) - element COMPLEX*16 vector,
c       also generally the result of an SVD (sigma is usually the singular
c       values).  n is INTEGER*4.  zero is the REAL*8 value to be taken for
c       zero ("tolerance).  The result of the routine is returned in the m x n
c       COMPLEX*16 matrix product.
c               The product v * sigma1 is evaluated as follows:
c                       product = v * < r(i) >
c       where < * > is an m x n matrix with (main) diagonal entries *,
c             r(i) = 1 / sigma(i) if | sigma(i) | > (passed) zero,
c                  = 0                otherwise.
c
c<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
c
        implicit none
        intrinsic cdabs, jmin0
c
        integer*4 m, n
        real*8 zero
        complex*16 v( m, m ), sigma( 1 ), product( m, n )
c
        integer*4 num_sig, i, j
c
        num_sig = jmin0( m, n )
c
        do i = 1, num_sig
          if ( cdabs( sigma(i) ) .gt. zero ) then
            do j = 1, m
              product(j,i) = v(j,i) / sigma(i)
            end do
          else
            do j = 1, m
              product(j,i) = ( 0.0d0, 0.0d0 )
            end do
          end if
        end do
        do i = num_sig + 1, n
          do j = 1, m
            product(j,i) = ( 0.0d0, 0.0d0 )
          end do
        end do
c
        return
        end
```

18-Apr-1988 14:17:03
22-Jun-1987 20:01:56

```
0001            subroutine eig_from_sv( svb, n, fuzz, eig )
0002      c
0003      c-------------------------------------------------------------------------------
0004      c
0005      c               This subroutine completes the operations started by subroutine
0006      c       sq_vlgsvd, producing the generalized eigenvalues for the matrix pair
0007      c       (a, b).
0008      c
0009      c       Parameters:
0010      c       -----------
0011      c               svb:  n-element COMPLEX*16 vector of singular values of
0012      c                             b * a_inv (all entries should be real).
0013      c               n:   INTEGER*4 number of elements in svb and eig.
0014      c               fuzz:  REAL*8 value to be taken for zero ("tolerance").
0015      c               eig:  n-element REAL*8 result vector of generalized
0016      c                             eigenvalues.
0017      c
0018      c-------------------------------------------------------------------------------
0019      c
0020            implicit none
0021            intrinsic dreal
0022      c
0023            integer*4 n
0024            real*8 fuzz, eig(n)
0025            complex*16 svb(n)
0026      c
0027            integer*4 i
0028            real*8 svsqb
0029      c
0030      c               Compute generalized eigenvalues.
0031      c
0032            do i = 1, n
0033              svsqb = dreal( svb(i) )
0034              svsqb = svsqb * svsqb
0035              if ( svsqb .le. fuzz ) then
0036                eig(i) = 0.0d0
0037              else
0038                eig(i) = 1.0d0 / svsqb
0039              end if
0040            end do
0041      c
0042            return
0043            end
```

```
                                                    18-Apr-1988 14:17:39     VA
                                                     2-Apr-1988 16:47:00     EN
0001            logical function endspawn_if( log_unit, prog_num )
0002    c
0003    c-------------------------------------------------------------------------------
0004    c
0005    c                       This function reads (and locks) the record with primary key
0006    c       value prog_num in the indexed-organization file open on logical unit
0007    c       number log_unit.  If the specified record is locked when the read
0008    c       attempt is made, the function exits returning a value of .false..
0009    c                       If the desired record has been read and locked, the function
0010    c       checks that the process named in the record is not present on the VAX,
0011    c       and if it is present exits returning the value .false..  If the
0012    c       process is not present, the function sets the "spawned" indication in
0013    c       the record to 0, rewrites (and unlocks) the record, and returns .true..
0014    c                       If an error occurs during any of the system calls, the function
0015    c       prints an error message and returns .true..
0016    c                       The FORTRAN program MAKE_SPAWN_IDX can be used to create a file
0017    c       of the required format.  The text file
0018    c       6000$RT:[CHUCK.RESEARCH.FORTDIR]SPAWNED_DEF.TXT contains the
0019    c       definitions required to access such a file.  The parameters log_unit
0020    c       and prog_num are INTEGER*4.
0021    c
0022    c-------------------------------------------------------------------------------
0023    c
0024            implicit none
0025            external lib$getjpi, lib$wait
0026            include '(system_symbols)'
0144            include 'SYS$LIBRARY:FORIOSDEF'
0230            include '6000$RT:[CHUCK.RESEARCH.FORTDIR]SPAWNED_DEF.TXT'
0273    c
0274            integer*4 lib$getjpi
0275    c
0276            integer*4 log_unit, prog_num
0277    c
0278            integer*4 status, out_int
0279            character*30 err_type
0280    c
0281            read (unit=log_unit,keyeq=prog_num,keyid=0,iostat=status)
0282            1                                sp_num, sp_stat, sp_file, sp_proc
0283            if      ( status .eq. FOR$IOS_SPERECLOC ) then  !  Record lock.
0284              unlock (unit=log_unit)
0285              endspawn_if = .false.
0286              return
0287            else if ( status .ne. 0 ) then                  !  Read error.
0288              type *
0289              type *, 'Error during read, code:  ', status
0290              type *
0291              unlock (unit=log_unit)
0292              endspawn_if = .true.
0293              return
0294            end if
0295    c
0296            status = lib$getjpi( JPI$_PID, , sp_proc, out_int )
0297            if      ( ( status .eq. SS$_NORMAL )
0298            1               .or. ( status .eq. SS$_SUSPENDED ) ) then
0299              unlock (unit=log_unit)
0300              endspawn_if = .false.
0301              return
```

```
ENDSPAWN_IF                                                    18-Apr-1988 14:17:39
                                                               2-Apr-1988 16:47:00

0302            else if ( status .eq. SS$_NONEXPR ) then
0303              go to 10000
0304            else if ( status .eq. SS$_BADPARAM ) then
0305              unlock (unit=log_unit)
0306              err_type = 'Bad parameter.'
0307            else if ( status .eq. SS$_ACCVIO ) then
0308              unlock (unit=log_unit)
0309              err_type = 'Access violation.'
0310            else if ( status .eq. SS$_IVLOGNAM ) then
0311              unlock (unit=log_unit)
0312              err_type = 'Invalid process name length.'
0313            else if ( status .eq. SS$_NOMOREPROC ) then
0314              unlock (unit=log_unit)
0315              err_type = 'No "more" processes.'
0316            else if ( status .eq. SS$_NOPRIV ) then
0317              unlock (unit=log_unit)
0318              err_type = 'No privilege.'
0319            else
0320              unlock (unit=log_unit)
0321              err_type = 'Code:            hex.'
0322              write (unit=err_type( 8 : 15 ),fmt=90000) status
0323            end if
0324    c
0325            type *
0326            type *, 'Failed checking "', sp_proc, '":  ', err_type
0327            type *
0328            endspawn_if = .true.
0329            return
0330    c
0331    10000   continue
0332            sp_stat = 0
0333            rewrite (unit=log_unit,err=20000,iostat=status)
0334            1                                sp_num, sp_stat, sp_file, sp_proc
0335            unlock (unit=log_unit)
0336    c
0337            endspawn_if = .true.
0338            return
0339    c
0340    20000   continue                                          !  Rewrite error.
0341            unlock (unit=log_unit)
0342            type *
0343            type *, 'Error during rewrite, code:  ', status
0344            type *
0345            endspawn_if = .true.
0346            return
0347    c
0348    90000   format( z8.8 )
0349    c
0350            end
```

```
                                                                18-Apr-1988 14:18:06
                                                                25-Feb-1988 18:45:56

0001            subroutine get_music_def( num_ants, num_waves, noises, anoises,
0002            1                                                  which, num_smp )
0003      c
0004      c-------------------------------------------------------------------------------
0005      c
0006      c               This subroutine sets the number of antennae, waves, samples,
0007      c       and noise generators to use for the MUSIC / Lp algorithm based on
0008      c       on operator input.  The default values are assumed to be the values o
0009      c       entry to the routine.
0010      c
0011      c       Parameters:
0012      c       -----------
0013      c               num_ants:  INTEGER*4 number of antenna array elements.
0014      c               num_waves:  INTEGER*4 number of arriving waves.
0015      c               noises:  INTEGER*4 number of (nominally) independent noise
0016      c                                       sources used in the ESTIMATION model.
0017      c               anoises:  INTEGER*4 number of (nominally) independent noise
0018      c                                       sources used in sample GENERATION.
0019      c               which:  INTEGER*4 selector for using "model" (which > 0) or
0020      c                                       "actual" (which = 0) noise for
0021      c                                       estimation.
0022      c               num_smp:  INTEGER*4 number of samples to process (in the
0023      c                                       current experiment).
0024      c
0025      c-------------------------------------------------------------------------------
0026      c
0027            implicit none
0028            intrinsic index
0029            external caps, cap, nonblank
0030      c
0031            logical nonblank
0032            character*1 cap
0033      c
0034            integer*4 num_ants, num_waves, noises, anoises, which
0035            integer*4 num_smp
0036      c
0037            integer*4 linenum, it1, it2
0038            character*5 ianswer
0039      c
0040      10010 continue
0041              assign 10010 to linenum
0042              type 90010, ' Number of antennae (', num_ants, ')?  '
0043              accept 90020, ianswer
0044              if ( nonblank( ianswer ) ) then
0045                read (unit=ianswer,fmt=*,err=20000) it1
0046                if ( it1 .le. 0 ) go to 20050
0047                num_ants = it1
0048              end if
0049      c
0050      10020 continue
0051              assign 10020 to linenum
0052              type 90010, ' Number of arriving waves (', num_waves, ')?  '
0053              accept 90020, ianswer
0054              if ( nonblank( ianswer ) ) then
0055                read (unit=ianswer,fmt=*,err=20000) it1
0056                if ( it1 .le. 0 ) go to 20050
0057                num_waves = it1
```

```
GET_MUSIC_DEF                                                    18-Apr-1988 14:18:06    VAX
                                                                 25-Feb-1988 18:45:56    GET

0058                end if
0059        c
0060        10025   continue
0061                assign 10025 to linenum
0062                type 90010, ' Number of noise generators for sample GENERATION (',
0063               1                                                    anoises, ')?  '
0064                accept 90020, ianswer
0065                if ( nonblank( ianswer ) ) then
0066                  read (unit=ianswer,fmt=*,err=20000) it1
0067                  if ( it1 .le. 0 ) go to 20050
0068                  anoises = it1
0069                end if
0070        c
0071        10120   continue
0072                if ( which .gt. 0 ) then
0073                  ianswer = 'A/[M]'
0074                else
0075                  ianswer = '[A]/M'
0076                end if
0077                type 90030, ' Use actual or (different) model noise samples '
0078               1                          // 'for ESTIMATION (' // ianswer // ')?  '
0079                accept 90020, ianswer
0080                if ( nonblank( ianswer ) ) then
0081                  call caps( ianswer )
0082                  it1 = index( ianswer, 'A' )
0083                  it2 = index( ianswer, 'M' )
0084                  if ( ( it1 .eq. 0 ) .and. ( it2 .eq. 0 ) ) go to 10120
0085                  if ( it1 .gt. it2 ) then
0086                    which = 0
0087                  else
0088                    which = 1
0089                  end if
0090                end if
0091        c
0092                if ( which .gt. 0 ) then
0093        10125     continue
0094                  assign 10125 to linenum
0095                  type 90010, ' Number of noise generators for ESTIMATION '
0096                 1                                  // 'noise model (', noises, ')?  '
0097                  accept 90020, ianswer
0098                  if ( nonblank( ianswer ) ) then
0099                    read (unit=ianswer,fmt=*,err=20000) it1
0100                    if ( it1 .le. 0 ) go to 20050
0101                    noises = it1
0102                  end if
0103                else
0104                  noises = anoises
0105                end if
0106        c
0107        10030   continue
0108                assign 10030 to linenum
0109                type 90010, ' Number of samples per trial to process (', num_smp,
0110               1                                                  ')?  '
0111                accept 90020, ianswer
0112                if ( nonblank( ianswer ) ) then
0113                  read (unit=ianswer,fmt=*,err=20000) it1
0114                  if ( it1 .le. 0 ) go to 20050
```

```
GET_MUSIC_DEF                                                  18-Apr-1988 14:18:06
                                                               25-Feb-1988 18:45:56

0115                  num_smp = it1
0116                end if
0117     c
0118               type *
0119               type 90030, ' Above values OK ([Y]/N)?  '
0120               accept 90020, ianswer
0121               if ( cap( ianswer( 1 : 1 ) ) .eq. 'N' ) go to 10010
0122     c
0123               return
0124     c
0125     20000     continue
0126               type *, char( 7 ), 'Error on input.  Please re-enter.'
0127               go to linenum
0128     c
0129     20050     continue
0130               type *, char( 7 ), 'Must be > 0.  Please re-enter.'
0131               go to linenum
0132     c
0133     90010     format ( $ a, i5, a )
0134     90020     format ( a5 )
0135     90030     format ( $a )
0136     c
0137               end


PROGRAM SECTIONS

    Name                                    Bytes   Attributes

  0 $CODE                                    1196   PIC CON REL LCL   SHR   EXE   RD NOWR
  1 $PDATA                                    380   PIC CON REL LCL   SHR NOEXE   RD NOWR
  2 $LOCAL                                    184   PIC CON REL LCL NOSHR NOEXE   RD   WR

    Total Space Allocated                    1760


ENTRY POINTS

    Address  Type  Name

  0-00000000       GET_MUSIC_DEF


VARIABLES

    Address  Type  Name              Address  Type  Name              Address  Type  N

AP-00000010@ I*4  ANOISES          2-00000000  CHAR  IANSWER         2-00000008  I*4   I
     **      I*4  LINENUM         AP-0000000C@ I*4  NOISES        AP-00000004@ I*4  N
AP-00000008@ I*4  NUM_WAVES       AP-00000014@ I*4  WHICH
```

```
                                                        18-Apr-1988 14:18:26
                                                        25-Aug-1987 14:27:25

0001            subroutine get_w_half( resid, size, p, eps, norm, w )
0002    c
0003    c-------------------------------------------------------------------------------
0004    c
0005    c                   This subroutine computes the SQUARE ROOT OF the weighting
0006    c       "matrix" (W) for the IRLS algorithm.  W is a diagonal matrix, so the
0007    c       result is returned in the VECTOR w.
0008    c
0009    c       Parameters:
0010    c       -----------
0011    c               resid:  size-element COMPLEX*16 vector of residuals for which
0012    c                                       the weight matrix is to be computed.
0013    c               size:  INTEGER*4 number of residual elements (problem order).
0014    c               p:  REAL*4 value of power of residuals to minimize.  That is
0015    c                                       an Lp minimization is to be performed
0016    c               eps:  REAL*8 value of minimum residual magnitude to be used
0017    c                                       (see below).
0018    c               norm:  REAL*8 value to which to normalize the SQUARE ROOT OF
0019    c                                       the maximum weight.  If norm <= 0.0d0
0020    c                                       no normalization occurs.  Otherwise,
0021    c                                       all weight SQUARE ROOT values (w(i))
0022    c                                       are divided by a value which results
0023    c                                       a maximum weight SQUARE ROOT of norm.
0024    c               w:  size-element REAL*8 vector of the SQUARE ROOTS OF the
0025    c                                       computed weights.
0026    c
0027    c       Operation:
0028    c       ----------
0029    c       The elements of w are computed as follows:
0030    c               ! resid(i) ! ^ ( ( p - 2.0 ) / 2.0 )    if ! resid(i) ! > eps
0031    c                    ! eps ! ^ ( ( p - 2.0 ) / 2.0 )    otherwise.
0032    c       If norm > 0.0d0, w(i) is then divided by norm / ( max w(i) ).
0033    c
0034    c-------------------------------------------------------------------------------
0035    c
0036            implicit none
0037            intrinsic cdabs
0038    c
0039            integer*4 size
0040            real*4 p
0041            real*8 eps, norm, w( size )
0042            complex*16 resid( size )
0043    c
0044            integer*4 i
0045            real*4 pm2
0046            real*8 epm2, wmult
0047    c
0048            pm2   = 0.5 * p - 1.0
0049            epm2  = eps ** pm2
0050            wmult = -1.0d0
0051    c
0052            do i = 1, size
0053              w(i) = cdabs( resid(i) )
0054              if ( w(i) .lt. eps ) then
0055                w(i) = epm2
0056              else
0057                w(i) = w(i) ** pm2
```

```
GET_W_HALF                                                    18-Apr-1988 14:18:26
                                                              25-Aug-1987 14:27:25

0058                end if
0059                if ( norm .gt. 0.0d0 ) then
0060                  if ( w(i) .gt. wmult ) wmult = w(i)
0061                end if
0062              end do
0063     c
0064              if ( norm .gt. 0.0d0 ) then
0065                wmult = norm / wmult
0066                do i = 1, size
0067                  w(i) = w(i) * wmult
0068                end do
0069              end if
0070     c
0071              return
0072              end


PROGRAM SECTIONS

    Name                                    Bytes   Attributes

  0 $CODE                                     228   PIC CON REL LCL   SHR   EXE   RD NOW
  2 $LOCAL                                     96   PIC CON REL LCL NOSHR NOEXE   RD   W

    Total Space Allocated                     324


ENTRY POINTS

    Address  Type  Name

  0-00000000       GET_W_HALF


VARIABLES

    Address  Type  Name             Address  Type  Name             Address  Type

      **      R*8  EPM2          AP-00000010@ R*8  EPS                **      I*4
 AP-0000000C@ R*4  P                  **      R*4  PM2           AP-00000008@ I*4


ARRAYS

    Address  Type  Name               Bytes  Dimensions

 AP-00000004@ C*16 RESID               **   (*)
 AP-00000018@ R*8  W                   **   (*)
```

```
                                                        18-Apr-1988 14:19:09
                                                        19-Dec-1987 09:14:31

0001            subroutine make_noise( out, n, defs, pars, corr, size )
0002    c
0003    c-------------------------------------------------------------------------------
0004    c
0005    c               This subroutine computes a COMPLEX*16 noise vector.
0006    c
0007    c       Parameters:
0008    c       -----------
0009    c               out = COMPLEX*16 n-element output noise vector.
0010    c               n = INTEGER*4 number of elements in the output vector.
0011    c               defs = INTEGER*4 size x 2 matrix whose first column is the
0012    c                               vector of noise source types (see subroutine
0013    c                               READ_NOISE), and whose second column is the
0014    c                               vector of counters for intermittent and burst
0015    c                               noise.
0016    c               pars = REAL*4 size x 3 matrix of noise source parameters (see
0017    c                               subroutine READ_NOISE).
0018    c               corr = REAL*8 n x size matrix of noise source multipliers for
0019    c                               each output element ("correlation" matrix).
0020    c               size = INTEGER*4 number of noise sources in use.
0021    c
0022    c               NOTE:  Intermittent and burst noise sources are controlled by
0023    c       the parameter defs, above.  If the defs(*,2) entry for an intermitten
0024    c       or burst source is 1, the source is activated for the current output
0025    c       vector.  All intermittent and burst source counts are decremented aft
0026    c       output generation, and sources whose counts reach zero are then
0027    c       supplied with new counts (based on the approriate entries in pars).
0028    c
0029    c-------------------------------------------------------------------------------
0030    c
0031            implicit none
0032            intrinsic jishft
0033            external open_va, free_va, exit_va
0034            external c8_noise, noise_combine, uni
0035            include '6000$RT:[CHUCK.RESEARCH.FORTDIR]VATYPES.TXT'
0047    c
0048            real*4 uni
0049    c
0050            integer*4 n, size, defs( size, 1 )
0051            real*4 pars( size, 1 )
0052            real*8 corr( n, size )
0053            complex*16 out( n )
0054    c
0055            integer*4 linear( 1 ), va_status, source_noise_add, i
0056            character*80 va_msg
0057    c
0058            linear(1) = size
0059    c
0060    c               Get space for source noise vector, then generate size COMPLEX
0061    c       noise values in the vector.
0062    c
0063            call open_va( source_noise_add, linear, 1, sc_type, va_status, va_msg
0064            call exit_va( va_status, va_msg )
0065            call c8_noise( %val( source_noise_add ), size, defs, pars )
0066    c
0067    c               Now multiply the result noise source vector by corr (see
0068    c       subroutine NOISE_COMBINE), and discard the space used by the source
```

```
MAKE_NOISE                                                    19-Apr-1988 14:1
                                                              19-Dec-1987 09:1

0069     c        noise vector.
0070     c
0071              call noise_combine( corr, n, size, %val( source_noise_add ), def
0072              1
0073              call free_va( source_noise_add, va_status, va_msg )
0074              call exit_va( va_status, va_msg )
0075     c
0076     c                Finally, re-trigger intermittent and burst noise sources
0077     c        appropriate.
0078     c
0079              do i = 1, size
0080                if ( jishft( defs(i,1), -16 ) .ne. 0 ) then
0081                  defs(i,2) = defs(i,2) - 1
0082                  if ( defs(i,2) .le. 0 ) then
0083                    defs(i,2) = pars(i,3) * ( 0.5 + uni( 10571 ) ) + 0.5
0084                  end if
0085                end if
0086              end do
0087     c
0088              return
0089              end.


PROGRAM SECTIONS

    Name                                Bytes   Attributes

  0 $CODE                                 316   PIC CON REL LCL   SHR   EXE   R
  1 $PDATA                                 12   PIC CON REL LCL   SHR NOEXE   R
  2 $LOCAL                                412   PIC CON REL LCL NOSHR NOEXE   R

    Total Space Allocated                 740


ENTRY POINTS

    Address  Type  Name

  0-00000000       MAKE_NOISE


VARIABLES

    Address  Type  Name                                          Address  T

       **      I*4  I                                        AP-00000008@
 AP-00000018@  I*4  SIZE                                      2-00000058
  2-00000004   CHAR VA_MSG                                   2-00000054
```

```
                                                                18-Apr-1988 14:20:47
                                                                 9-Sep-1987 17:14:46

0001            complex*16 function modulated( wave, num_waves, carr_camp, wpars,
0002            1                                               mtypes, samp_num, samp_int )
0003      c
0004      c-------------------------------------------------------------------------------
0005      c
0006      c                 This function returns the complex amplitude of wave number
0007      c       wave based on the wave's modulation characteristics and carrier
0008      c       amplitude and phase.  That is, this routine returns the wave-th elem
0009      c       of the "signal-in-space" vector at sample number samp_num.
0010      c       .
0011      c       Parameters:
0012      c       -----------
0013      c                 wave:  INTEGER*4 number of the wave for which to compute the
0014      c                                  complex amplitude.
0015      c                 num_waves:  INTEGER*4 number of waves present.
0016      c                 carr_camp:  num_waves-element COMPLEX*16 vector of carrier
0017      c                                  complex amplitudes (from SNRs and "starting"
0018      c                                  phases).
0019      c                 wpars = num_waves x 9 REAL*8 matrix whose columns include:
0020      c                                  4) vector of modulation cycle lengths, in
0021      c                                        number of carrier cycles per modulat
0022      c                                        cycle.
0023      c                                  5) vector of modulation duty cycles (expressc
0024      c                                        as fractions of modulation cycle
0025      c                                        lengths, 0.0 < mod_duty(i) <= 1.0)
0026      c                                        (apply only to ramp and rectangular
0027      c                                        modulation).
0028      c                                  6) vector of "starting" modulation phases, i
0029      c                                        number of carrier cycles.
0030      c                                  7) vector of amplitude modulation ratios
0031      c                                        (maximum / minimum).
0032      c                                  8) vector of phase modulation total phase
0033      c                                        shifts (in RADIANS).
0034      c                 mtypes = num_waves x 2 INTEGER*4 matrix whose first column i
0035      c                                  the vector of amplitude modulation types and
0036      c                                  whose second column is the vector of phase
0037      c                                  modulation types (see below).
0038      c                 samp_num:  INTEGER*4 number of the sample for which the comp
0039      c                                  amplitude is to be computed.
0040      c                 samp_int:  REAL*8 number of carrier cycles between samples.
0041      c
0042      c                 Modulation types:  1 = sine, 2 = triangle, 3 = increasing ra
0043      c       4 = decreasing ramp, 5 = rectangular.
0044      c
0045      c-------------------------------------------------------------------------------
0046      c
0047            implicit none
0048            intrinsic dcmplx, dcos, dsin
0049            external mod_val, mod_phase
0050      c
0051            real*8 mod_val, mod_phase
0052      c
0053            integer*4 wave, num_waves, samp_num, mtypes( num_waves, 1 )
0054            real*8 wpars( num_waves, 1 ), samp_int
0055            complex*16 carr_camp( num_waves )
0056      c
0057            real*8 temp, mod_phs
```

```
MODULATED                                                    18-Apr-1988 14:20:47     V
                                                              9-Sep-1987 17:14:46     M

0058     c
0059               mod_phs = mod_phase( wave, num_waves, wpars, samp_num, samp_int )
0060     c
0061               temp = mod_val( mtypes(wave,1), mod_phs, wpars(wave,5) )
0062               modulated = carr_camp(wave) *
0063               1                          ( temp + ( ( 1.0d0 - temp ) / wpars(wave,7) ) )
0064     c
0065               temp = 2.0d0 * wpars(wave,8) *
0066               1        ( mod_val( mtypes(wave,2), mod_phs, wpars(wave,5) ) - 0.5d0 )
0067               modulated = modulated * dcmplx( dcos( temp ), dsin( temp ) )
0068     c
0069               return
0070               end


PROGRAM SECTIONS

    Name                                  Bytes   Attributes

  0 $CODE                                   308   PIC CON REL LCL   SHR   EXE   RD NOWRT L
  2 $LOCAL                                  192   PIC CON REL LCL NOSHR NOEXE   RD   WRT Q

    Total Space Allocated                   500


ENTRY POINTS

    Address  Type  Name

  0-00000000  C*16 MODULATED


VARIABLES

    Address  Type  Name             Address  Type  Name             Address  Type  Name

  2-00000000  R*8  MOD_PHS       AP-0000000C@ I*4  NUM_WAVES     AP-00000020@ R*8  SAMP
     **       R*8  TEMP          AP-00000008@ I*4  WAVE


ARRAYS

    Address  Type  Name               Bytes  Dimensions

 AP-00000010@ C*16 CARR_CAMP            **  (*)
 AP-00000018@ I*4  MTYPES               **  (*, 1)
 AP-00000014@ R*8  WPARS                **  (*, 1)
```

18-Apr-1988 14:20:4
9-Sep-1987 17:14:4

```
c
        real*8 function mod_val( type, phs, duty )
c
c<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
c
c               This function returns the modulation value selected by the
c       passed modulation type, current phase, and duty cycle.  The value
c       returned is between 0.0 and 1.0, and is the fraction of modulation
c       be applied.
c
c       Parameters:
c       -----------
c               type:  INTEGER*4 value selecting the type of modulation bei
c                               applied (see below).
c               phs:  REAL*8 value specifying the current modulation phase,
c                               expressed as a fraction of the modulation
c                               cycle (phs = 0.0 => start of modulation cyc
c                               phs = 1.0 => end of modulation cycle).
c               duty:  REAL*8 value specifying the duty cycle for ramp and
c                               rectangular modulation (ignored for other
c                               modulation types).  Expressed as a fraction
c                               the modulation cycle (see "phs").
c
c               Modulation types:  1 = sine, 2 = triangle, 3 = increasing r
c       4 = decreasing ramp, 5 = rectangular.
c
c<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
c
        implicit none
        intrinsic dsin
        include '(real_constants)'
c
        integer*4 type
        real*8 phs, duty
c
        integer*4 Smod/1/, Tmod/2/, Imod/3/, Dmod/4/, Rmod/5/
c
        if ( type .eq. Smod ) then                      ! Sine.
          mod_val = 0.5d0 + 0.5d0 * dsin( phs * pi8t2 )
c
        else if ( type .eq. Tmod ) then                 ! Triangle.
          if ( phs .lt. 0.25d0 ) then
            mod_val = 0.5d0 + 2.0d0 * phs
          else if ( phs .lt. 0.75d0 ) then
            mod_val = 1.5d0 - 2.0d0 * phs
          else
            mod_val = -1.5d0 + 2.0d0 * phs
          end if
c
        else if ( type .eq. Imod ) then                 ! Increasing ramp.
          if ( phs .le. duty ) then
            mod_val = phs / duty
          else
            mod_val = 0.0d0
          end if
c
        else if ( type .eq. Dmod ) then                 ! Decreasing ramp.
```

```
MOD_VAL                                                    18-Apr-1988 14:20:47
                                                           9-Sep-1987 17:14:46

0108                if ( phs .le. duty ) then
0109                  mod_val = 1.0d0 - ( phs / duty )
0110                else
0111                  mod_val = 0.0d0
0112                end if
0113     c
0114              else                                     ! Rectangular.
0115                if ( phs .le. duty ) then
0116                  mod_val = 1.0d0
0117                else
0118                  mod_val = 0.0d0
0119                end if
0120     c
0121              end if
0122              return
0123     c
0124              end


PROGRAM SECTIONS

     Name                                  Bytes   Attributes

   0 $CODE                                   204   PIC CON REL LCL   SHR   EXE   RD NOWR
   2 $LOCAL                                   24   PIC CON REL LCL NOSHR NOEXE   RD   WR

     Total Space Allocated                   228


ENTRY POINTS

     Address  Type  Name

   0-00000000  R*8  MOD_VAL


VARIABLES

     Address  Type  Name              Address  Type  Name              Address  Type  N

   2-00000014  I*4  DMOD           AP-0000000C@ R*8  DUTY           2-00000010  I*4  I
      **       I*4  RMOD            2-00000008  I*4  SMOD           2-0000000C  I*4  T


FUNCTIONS AND SUBROUTINES REFERENCED

  Type  Name

   R*8  MTH$DSIN
```

```
c
        real*8 function mod_phase( wave, num_waves, wpars, sample, samp_intvl )
c
c<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
c
c               This function returns the modulation phase of wave number wave
c       at sample number sample, as a fraction of modulation cycle length
c       (0.0 <= mod_phase < 1.0).
c
c       Parameters:
c       -----------
c               wave:  INTEGER*4 number of the wave for which to evaluate the
c                               modulation phase.
c               num_waves:  INTEGER*4 number of incident waves.
c               wpars:  num_waves x 9 REAL*8 matrix whose fourth column is the
c                               vector of modulation cycle lengths, in number
c                               of carrier cycles per modulation cycle, and
c                               whose sixth column is the vector of "starting"
c                               modulation phases, in number of carrier cycles.
c               sample:  INTEGER*4 number of the sample for which modulation
c                               phase is to be computed.
c               samp_intvl:  REAL*8 number of carrier cycles between samples.
c
c<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
c
        implicit none
        intrinsic dmod
        include '(real_constants)'
c
        integer*4 wave, num_waves, sample
        real*8 wpars(num_waves,1), samp_intvl
c
        real*8 carr_cycnm
c
        carr_cycnm = ( sample - 1 ) * samp_intvl + wpars(wave,6)
        mod_phase = dmod( carr_cycnm, wpars(wave,4) ) / wpars(wave,4)
c
        return
        end
```

```
                                                                18-Apr-1988 14:21:33
                                                                18-Dec-1987 17:42:03

0001            subroutine music_init( waves, wpars, ccamp,
0002            1                                 noises, ndefs, npars,
0003            2                                 anoises, andefs, anpars, which )
0004      c
0005      c-------------------------------------------------------------------------------
0006      c
0007      c               This subroutine computes the complex amplitudes of the carriers
0008      c       of the waves incident on an antenna array, and initializes counters for
0009      c       use by intermittent noise generators.
0010      c
0011      c       Parameters:
0012      c       -----------
0013      c               waves:   INTEGER*4 number of waves incident on the array.
0014      c               wpars:   waves x 9 REAL*8 matrix whose first column is the
0015      c                        vector of incident wave strengths, in db
0016      c                        (referred to an arbitrary unit amplitude), and
0017      c                        whose second column is the vector of carrier
0018      c                        starting phases, in RADIANS.
0019      c               ccamp:   waves-element COMPLEX*16 output vector of carrier
0020      c                        complex amplitudes (from SNRs and "starting"
0021      c                        phases).  Expressed in "rectangular"
0022      c                        coordinates (i.e., A * exp( j * phi ) ).
0023      c              noises:   INTEGER*4 number of noise generators in use in the
0024      c                        ESTIMATION noise model.
0025      c               ndefs:   noises x 2 INTEGER*4 matrix whose first column is the
0026      c                        vector of noise types (see subroutine
0027      c                        READ_NOISE), and whose second column is the
0028      c                        vector of counts-to-output for intermittent
0029      c                        noise generators.  This matrix applies to the
0030      c                        ESTIMATION noise model.
0031      c               npars:   noises x 3 REAL*4 array of noise parameters (see
0032      c                        subroutine READ_NOISE).  This matrix applies
0033      c                        to the ESTIMATION noise model.
0034      c              anoises:   INTEGER*4 number of noise generators in use for
0035      c                        ACTUAL SAMPLE GENERATION.
0036      c               andefs:   anoises x 2 INTEGER*4 matrix whose first column is the
0037      c                        vector of noise types (see subroutine
0038      c                        READ_NOISE), and whose second column is the
0039      c                        vector of counts-to-output for intermittent
0040      c                        noise generators.  This matrix applies to
0041      c                        ACTUAL SAMPLE GENERATION.
0042      c               anpars:   anoises x 3 REAL*4 array of noise parameters (see
0043      c                        subroutine READ_NOISE).  This matrix applies
0044      c                        to ACTUAL SAMPLE GENERATION.
0045      c               which:   INTEGER*4 value which determines whether the actual
0046      c                        noise samples used in generating the
0047      c                        observations are to be used in the estimation
0048      c                        routines (which = 0), or a separate estimation
0049      c                        noise model is to be used (which > 0).  Note
0050      c                        that if which is > 0 then anoises, andefs, and
0051      c                        anpars are used; otherwise only noises, ndefs,
0052      c                        and npars are used.
0053      c
0054      c-------------------------------------------------------------------------------
0055      c
0056            implicit none
0057            intrinsic dcmplx, jishft
```

MUSIC_INIT 18-Apr-1988 14:21:3
18-Dec-1987 17:42:0

```
0058            external rect, uni
0059      c
0060            real*4 uni
0061            complex*16 rect
0062      c
0063            integer*4 waves, noises, anoises, which
0064            integer*4 ndefs( noises, 1 ), andefs( anoises, 1 )
0065            real*4 npars( noises, 1 ), anpars( anoises, 1 )
0066            real*8 wpars( waves, 1 )
0067            complex*16 ccamp( waves )
0068      c
0069            integer*4 i, it1
0070      c
0071            do i = 1, waves
0072              ccamp(i) = rect( dcmplx( 10.0d0 ** ( wpars(i,1) / 20.0d0 ),
0073            1                                                    wpars(i,2) )
0074            end do
0075      c
0076            do i = 1, noises
0077              it1 = jishft( ndefs(i,1), -16 )
0078              if ( it1 .ne. 0 ) then
0079                ndefs(i,2) = npars(i,3) * ( 0.5 + uni( 10571 ) ) + 0.5
0080              else
0081                ndefs(i,2) = 0
0082              end if
0083            end do
0084      c
0085            if ( which .gt. 0 ) then
0086              do i = 1, anoises
0087                it1 = jishft( andefs(i,1), -16 )
0088                if ( it1 .ne. 0 ) then
0089                  andefs(i,2) = anpars(i,3) * ( 0.5 + uni( 10571 ) ) + 0.5
0090                else
0091                  andefs(i,2) = 0
0092                end if
0093              end do
0094            end if
0095      c
0096            return
0097            end
```

18-Apr-1988 14:21:58
17-Nov-1987 07:58:22

```
0001            subroutine music_l2p( a, ants, waves, rx, rw, fuzz, p, info )
0002    c
0003    c-------------------------------------------------------------------------------
0004    c
0005    c               This subroutine computes the L2 estimate of the P matrix, giv
0006    c       the (L2 estimate of the) A matrix (from the MUSIC algorithm).
0007    c
0008    c       Paramters:
0009    c       ----------
0010    c               a:  ants x waves COMPLEX*16 (L2) estimate of the A matrix.
0011    c               ants:  INTEGER*4 number of antenna array elements.
0012    c               waves:  INTEGER*4 number of impinging waves.
0013    c               rx:  ants x ants COMPLEX*16 Rx matrix estimate (signal
0014    c                                               covariance matrix).
0015    c               rw:  ants x ants COMPLEX*16 Rw matrix estimate (noise
0016    c                                               covariance matrix).  NOTE tha
0017    c                                               rw must NOT be normalized by
0018    c                                               lambda-min.
0019    c               fuzz:  REAL*8 value to be taken for zero ("tolerance").
0020    c               p:  waves x waves COMPLEX*16 output P matrix estimate (source
0021    c                                               covariance matrix).
0022    c               info:  INTEGER*4 status indicator (output); 0 for successful
0023    c                                               estimation, non-0 on failure
0024    c                                               singular value decomposition.
0025    c
0026    c-------------------------------------------------------------------------------
0027    c
0028            implicit none
0029            external open_va, free_va, exit_va
0030            external dc_mppsinv, sqdc_matdif, dc_matmpy, dc_matmpyherm
0031    c
0032            integer*4 ants, waves, info
0033            real*8 fuzz
0034            complex*16 a( ants, waves ), rx( ants, ants ), rw( ants, ants )
0035            complex*16 p( waves, waves )
0036    c
0037            integer*4 dc_type/7/, status
0038            integer*4 ants_sqr( 2 ), waves_by_ants( 2 )
0039            integer*4 prod_add, diff_add, wk1_add
0040            character*80 msg
0041    c
0042            ants_sqr(1)      = ants
0043            ants_sqr(2)      = ants
0044            waves_by_ants(1) = waves
0045            waves_by_ants(2) = ants
0046    c
0047            call open_va( prod_add, waves_by_ants, 2, dc_type, status, msg )
0048            call exit_va( status, msg )
0049            call open_va( diff_add, ants_sqr, 2, dc_type, status, msg )
0050            call exit_va( status, msg )
0051            call open_va( wk1_add, waves_by_ants, 2, dc_type, status, msg )
0052            call exit_va( status, msg )
0053    c
0054            call dc_mppsinv( a, ants, waves, fuzz, %val( prod_add ), info )
0055            call sqdc_matdif( rx, rw, ants, %val( diff_add ) )
0056            call dc_matmpy( %val( prod_add ), waves, ants, %val( diff_add ), ants,
0057            1                                                %val( wk1_add ) )
```

```
MUSIC_L2P                                                  18-Apr-1988 14:21:5
                                                           17-Nov-1987 07:58:2

0058            call dc_matmpyherm( %val( wk1_add ), waves, ants,
0059            1                                %val( prod_add ), waves, p )
0060     c
0061            call free_va( prod_add, status, msg )
0062            call exit_va( status, msg )
0063            call free_va( diff_add, status, msg )
0064            call exit_va( status, msg )
0065            call free_va( wk1_add, status, msg )
0066            call exit_va( status, msg )
0067     c
0068            return
0069            end


PROGRAM SECTIONS

    Name                                  Bytes   Attributes

  0 $CODE                                   414   PIC CON REL LCL    SHR    EXE    RD N
  1 $PDATA                                    4   PIC CON REL LCL    SHR NOEXE    RD N
  2 $LOCAL                                  524   PIC CON REL LCL NOSHR NOEXE    RD

    Total Space Allocated                   942


ENTRY POINTS

    Address  Type  Name

  0-00000000       MUSIC_L2P


VARIABLES

    Address  Type  Name              Address  Type  Name              Address  Type

 AP-00000008@ I*4  ANTS            2-00000060  I*4  DC_TYPE         2-0000006C  I*4
 AP-00000020@ I*4  INFO            2-00000010  CHAR MSG             2-00000068  I*4
 AP-0000000C@ I*4  WAVES           2-00000070  I*4  WK1_ADD


ARRAYS

    Address  Type  Name                Bytes  Dimensions

 AP-00000004@ C*16 A                    **  (*, *)
  2-00000000  I*4  ANTS_SQR              8  (2)
 AP-0000001C@ C*16 P                    **  (*, *)
 AP-00000014@ C*16 RW                   **  (*, *)
 AP-00000010@ C*16 RX                   **  (*, *)
  2-00000008  I*4  WAVES_BY_ANTS         8  (2)
```

```
                                                                  18-Apr-1988 14:3
                                                                  16-Mar-1988 09:2

0001            subroutine music_lpp_each( obsmat, samps, ants, a, waves, p, eps
0002           1                                  norm, tol, pvm, resm, mxitlim,
0003           2                                        mnitlim, atitlim, min_de
0004    c
0005    c-----------------------------------------------------------------------
0006    c
0007    c                  This subroutine forms Lp estimates of each of the signal
0008    c       vector samples which produce a collection of observation vector
0009    c       from an antenna array.  Estimates are formed using the IRLS algo
0010    c
0011    c       Parameters:
0012    c       -----------
0013    c               obsmat:  samps x ants COMPLEX*16 matrix whose ROWS are t
0014    c                        noisy observations from the antenna arra
0015    c               samps:   INTEGER*4 number of observation samples.
0016    c               ants:  INTEGER*4 number of elements in the antenna array
0017    c               a:  ants x waves COMPLEX*16 phase shift matrix for the a
0018    c                        array and the arriving waves (the columr
0019    c                        are the "signal vectors").
0020    c               waves:  INTEGER*4 number of arriving waves.
0021    c               p:  REAL*4 power of the residuals to be minimized (Lp
0022    c                        estimation to be used).
0023    c               eps:  REAL*8 minimum residual magnitude to be used.
0024    c               norm:  REAL*8 value to which to normalize the SQUARE ROO
0025    c                        the maximum IRLS weight.  If norm <= 0.0
0026    c                        normalization occurs.  Otherwise, all we
0027    c                        SQUARE ROOT values (w(i)) are divided by
0028    c                        value which results in a maximum weight
0029    c                        ROOT of norm.
0030    c               tol:  REAL*8 value to be taken for zero ("tolerance").
0031    c               pvm:  waves x samps COMPLEX*16 output matrix whose COLUM
0032    c                        the estimates of the signal-in-space vec
0033    c                        samples.
0034    c               resm:  samps x ants COMPLEX*16 output matrix whose ROWS
0035    c                        residuals from the above estimates.
0036    c               mxitlim:  INTEGER*4 number of iterations to permit for t
0037    c                        algorithm.
0038    c               mnitlim:  INTEGER*4 minimum number to which the iteratic
0039    c                        counter was decremented by the IRLS algo
0040    c                        Note:  mxitlim - mnitlim is usually the
0041    c                        number of iterations used in any IRLS es
0042    c                        however, since routine DCV_IRLS can plac
0043    c                        NEGATIVE value in mnitlim, this possibil
0044    c                        should be checked.
0045    c               atitlim:  INTEGER*4 number of the sample to which mnitli
0046    c                        applies.
0047    c               min_del:  REAL*8 square of the norm of the difference be
0048    c                        two subsequent estimates of the same vec
0049    c                        This variable is set to the difference p
0050    c                        by subroutine DCV_IRLS for sample number
0051    c                        atitlim.
0052    c
0053    c-----------------------------------------------------------------------
0054    c
0055            implicit none
0056            external open_va, free_va, exit_va
0057            external c16_vec_ext, dcv_irls, c16_col_ins, c16_vec_ins
```

MUSIC_LPP_EACH 18-Apr-1988 14:22:20
16-Mar-1988 09:26:50

```
0058    c
0059            integer*4 samps, ants, waves, mxitlim, mnitlim, atitlim
0060            real*4 p
0061            real*8 eps, norm, tol, min_del
0062            complex*16 obsmat( samps, ants ), a( ants, waves )
0063            complex*16 pvm( waves, samps ), resm( samps, ants )
0064    c
0065            integer*4 status, obs_add, sspc_add, resid_add
0066            integer*4 ants_lin( 1 ), waves_lin( 1 )
0067            integer*4 di_type/3/, sr_type/4/, dr_type/5/, dc_type/7/
0068            integer*4 i, itlim
0069            real*8 del
0070            character*80 msg
0071    c
0072            ants_lin(1)  = ants
0073            waves_lin(1) = waves
0074            call open_va( obs_add, ants_lin, 1, dc_type, status, msg )
0075            call exit_va( status, msg )
0076            call open_va( sspc_add, waves_lin, 1, dc_type, status, msg )
0077            call exit_va( status, msg )
0078            call open_va( resid_add, ants_lin, 1, dc_type, status, msg )
0079            call exit_va( status, msg )
0080    c
0081            min_del = 0.0d0
0082            mnitlim = mxitlim
0083            atitlim = 0
0084            do i = 1, samps
0085              itlim = mxitlim
0086              call c16_vec_ext( obsmat, samps, ants, i, %val( obs_add ) )
0087              call dcv_irls( a, ants, waves, %val( obs_add ),
0088     1                       p, eps, norm, tol, itlim,
0089     2                       %val( sspc_add ), %val( resid_add ), del )
0090              if ( itlim .lt. mnitlim ) then
0091                mnitlim = itlim
0092                atitlim = i
0093                min_del = del
0094              end if
0095              call c16_col_ins( %val( sspc_add ), waves, pvm, waves, samps, i )
0096              call c16_vec_ins( %val( resid_add ), ants, resm, samps, ants, i )
0097            end do
0098    c
0099            call free_va( obs_add, status, msg )
0100            call exit_va( status, msg )
0101            call free_va( sspc_add, status, msg )
0102            call exit_va( status, msg )
0103            call free_va( resid_add, status, msg )
0104            call exit_va( status, msg )
0105    c
0106            return
0107            end
```

```
                                                                18-Apr-1988 14:22:44    \
                                                                17-Nov-1987 08:36:13    f

0001            subroutine music_lpp_speis( a, n, m, w, robs, rnoise,
0002            1                                                fuzz, pmat, info )
0003    c
0004    c-------------------------------------------------------------------------------
0005    c
0006    c               This subroutine computes an Lp estimate of the P matrix, given
0007    c       the (L2 estimate of the) A matrix (from the MUSIC algorithm).  The
0008    c       approach used is due to Speiser, as follows:
0009    c               pmat = ( a* w a )- a* w ( robs - rnoise ) w a ( a* w a )-,
0010    c       where * signifies conjugate transpose and ( )- is matrix inverse.
0011    c
0012    c       Paramters:
0013    c       ----------
0014    c               a:  n x m COMPLEX*16 (L2) estimate of the A matrix.
0015    c               n:  INTEGER*4 number antenna array elements.
0016    c               m:  INTEGER*4 number of impinging waves.
0017    c               w:  n-element REAL*8 vector containing the (non-zero) elements
0018    c                           of the IRLS weight matrix (from MUSIC_LPW_YAR,
0019    c                           usually).
0020    c               robs:  n x n COMPLEX*16 Rx matrix estimate (signal covariance
0021    c                           matrix).
0022    c               rnoise:  n x n COMPLEX*16 Rw matrix estimate (noise covariance
0023    c                           matrix).  NOTE that rnoise must NOT be
0024    c                           normalized by lambda-min.
0025    c               fuzz:  REAL*8 value to be taken for zero ("tolerance").
0026    c               pmat:  m x m COMPLEX*16 output P matrix estimate (source
0027    c                           covariance matrix).
0028    c               info:  INTEGER*4 status indicator (output); 0 for successful
0029    c                           estimation, non-0 on failure of singular value
0030    c                           decomposition.
0031    c
0032    c-------------------------------------------------------------------------------
0033    c
0034            implicit none
0035            external open_va, free_va, exit_va
0036            external dc_mppsinv, sqdc_matdif
0037            external drdi_dc_mmpy, dc_drdi_mmpy, dc_matmpy, dc_matmpyherm
0038    c
0039            integer*4 n, m, info
0040            real*8 fuzz, w( n )
0041            complex*16 a( n, m ), robs( n, n ), rnoise( n, n ), pmat( m, m )
0042    c
0043            integer*4 dc_type/7/, status
0044            integer*4 n_sqr( 2 ), m_by_n( 2 ), n_by_m( 2 )
0045            integer*4 prod_add, diff_add, wk1_add, wk2_add
0046            character*80 msg
0047    c
0048            n_sqr(1)  = n
0049            n_sqr(2)  = n
0050            n_by_m(1) = n
0051            n_by_m(2) = m
0052            m_by_n(1) = m
0053            m_by_n(2) = n
0054            call open_va( prod_add, m_by_n, 2, dc_type, status, msg )
0055            call exit_va( status, msg )
0056            call open_va( diff_add, n_sqr, 2, dc_type, status, msg )
0057            call exit_va( status, msg )
```

```
MUSIC_LPP_SPEIS                                                 18-Apr-1988 14:22:44
                                                                17-Nov-1987 08:36:13

0058            call open_va( wk1_add, n_by_m, 2, dc_type, status, msg )
0059            call exit_va( status, msg )
0060            call open_va( wk2_add, m_by_n, 2, dc_type, status, msg )
0061            call exit_va( status, msg )
0062    c
0063            call drdi_dc_mmpy( w, n, a, m, %val( wk1_add ) )
0064            call dc_mppsinv( %val( wk1_add ), n, m, fuzz, %val( wk2_add ), info )
0065            call dc_drdi_mmpy( %val( wk2_add ), m, n, w, %val( prod_add ) )
0066    c
0067            call sqdc_matdif( robs, rnoise, n, %val( diff_add ) )
0068            call dc_matmpy( %val( prod_add ), m, n, %val( diff_add ), n,
0069            1                                                 %val( wk2_add ) )
0070            call dc_matmpyherm( %val( wk2_add ), m, n, %val( prod_add ), m, pmat
0071    c
0072            call free_va( prod_add, status, msg )
0073            call exit_va( status, msg )
0074            call free_va( diff_add, status, msg )
0075            call exit_va( status, msg )
0076            call free_va( wk1_add, status, msg )
0077            call exit_va( status, msg )
0078            call free_va( wk2_add, status, msg )
0079            call exit_va( status, msg )
0080    c
0081            return
0082            end


PROGRAM SECTIONS

    Name                                Bytes   Attributes

  0 $CODE                                 561   PIC CON REL LCL   SHR   EXE   RD NOW
  1 $PDATA                                  4   PIC CON REL LCL   SHR NOEXE   RD NOW
  2 $LOCAL                                644   PIC CON REL LCL NOSHR NOEXE   RD   W

    Total Space Allocated                1209


ENTRY POINTS

    Address  Type  Name

  0-00000000       MUSIC_LPP_SPEIS


VARIABLES

    Address  Type  Name              Address  Type  Name              Address  Type

  2-00000068  I*4  DC_TYPE         2-00000074  I*4  DIFF_ADD        AP-0000001C@ R*8
 AP-0000000C@ I*4  M              2-00000018  CHAR MSG             AP-00000008@ I*4
  2-0000006C  I*4  STATUS         2-00000078  I*4  WK1_ADD          2-0000007C  I*4
```

18-Apr-1988 14:23:03
16-Mar-1988 10:16:26

```
0001            subroutine music_lpw_yar( obs, samps, ants, a, waves, p, eps,
0002           1                                  norm, tol, weights, sspace,
0003           2                                  resids, itlim, minit, del, wtfun )
0004      c
0005      c-------------------------------------------------------------------------------
0006      c
0007      c                 This subroutine forms a "combined" Lp weight matrix and uses
0008      c          it to compute estimates of each of the signal vector samples which
0009      c          produce a collection of observation vector samples from an antenna
0010      c          array.  The weight matrix is formed via the IRLS algorithm using the
0011      c          weight function selected by the INTEGER*4 value wtfun.  If wtfun = 1,
0012      c          the WEIGHT function used is one over the p-th root of the sum (across
0013      c          the samples) of the residuals raised to the p-th power (I have labelled
0014      c          this approach the "Yarlagadda" algorithm).  If wtfun = 2, the LOSS
0015      c          function used (to compute the weight function) is the p-th root of the
0016      c          sum (across the samples) of the residuals raised to the p-th power (I
0017      c          have labelled this the "sum of powers loss" approach).  NOTE that the
0018      c          actual "weight" matrix returned is composed of the SQUARE ROOTS of the
0019      c          computed weights.
0020      c
0021      c          Parameters:
0022      c          -----------
0023      c                  obs:  samps x ants COMPLEX*16 matrix whose ROWS are the
0024      c                                noisy observations from the antenna array.
0025      c                  samps:  INTEGER*4 number of observation samples.
0026      c                  ants:  INTEGER*4 number of elements in the antenna array.
0027      c                  a:  ants x waves COMPLEX*16 phase shift matrix for the antenna
0028      c                                array and the arriving waves (the columns of a
0029      c                                are the "signal vectors").
0030      c                  waves:  INTEGER*4 number of arriving waves.
0031      c                  p:  REAL*4 power of the residuals to be minimized (Lp
0032      c                                estimation to be used).
0033      c                  eps:  REAL*8 minimum residual magnitude to be used.
0034      c                  norm:  REAL*8 value to which to normalize the SQUARE ROOT OF
0035      c                                the maximum weight.  If norm <= 0.0d0, no
0036      c                                normalization occurs.  Otherwise, all weight
0037      c                                SQUARE ROOT values (w(i)) are divided by a
0038      c                                value which results in a maximum weight SQUARE
0039      c                                ROOT of norm.
0040      c                  tol:  REAL*8 value to be taken for zero ("tolerance").
0041      c                  weights:  ants-element REAL*8 vector of SQUARE ROOTS of the
0042      c                                computed weights (diagonal elements of the
0043      c                                weight matrix).
0044      c                  sspace:  waves x samps COMPLEX*16 output matrix whose COLUMNS
0045      c                                are the estimates of the signal-in-space vector
0046      c                                samples.
0047      c                  resids:  samps x ants COMPLEX*16 output matrix whose ROWS are
0048      c                                the residuals from the above estimates.
0049      c                  itlim:  INTEGER*4 number of iterations to permit for the IRLS
0050      c                                algorithm.  NOTE that calling this routine with
0051      c                                itlim = 0 will result in a least squares (L2)
0052      c                                estimate, regardless of the value of p.
0053      c                  minit:  INTEGER*4 minimum number to which the iteration counter
0054      c                                was decremented by the IRLS algorithm.  Note:
0055      c                                itlim - minit is ordinarily the number of
0056      c                                iterations used in the IRLS estimate; however,
0057      c                                if an error occurs in routine DC_MPPSINV
```

```
                                                                    18-Apr-1988 14:23:
                                                                    16-Mar-1988 10:16::

0058    c                               ("info", below, returned with a non-zero
0059    c                               value), this routine sets minit to the neg
0060    c                               of its current value minus one, and exits.
0061    c               del:  REAL*8 exit value of the square of the norm of the
0062    c                               difference between subsequent weight vecto
0063    c               wtfun:  INTEGER*4 value selecting whether the sum-of-power
0064    c                               (wtfun = 1) or the sum-of-powers-loss (wtf
0065    c                               2) method is to be used to compute the wei
0066    c                               (diagonal elements of the weight matrix).
0067    c
0068    c-----------------------------------------------------------------------
0069    c
0070            implicit none
0071            intrinsic cdabs
0072            external open_va, free_va, exit_va
0073            external dr_vecfill, dr_veccopy, drdi_dc_mmpy, dc_mppsinv
0074            external dc_drdi_mmpy, yar_ests, yar_resids, dr_vdel_nmsq
0075    c
0076            real*8 dr_vdel_nmsq
0077    c
0078            integer*4 samps, ants, waves, itlim, minit, wtfun
0079            real*4 p
0080            real*8 eps, norm, tol, weights( ants ), del
0081            complex*16 obs( samps, ants ), a( ants, waves )
0082            complex*16 sspace( waves, samps ), resids( samps, ants )
0083    c
0084            integer*4 status, antslin( 1 ), antswaves( 2 ), wavesants( 2 ), i,
0085            integer*4 oldw_add, wa_add, psinv_add, info
0086            integer*4 di_type/3/, sr_type/4/, dr_type/5/, dc_type/7/
0087            real*4 wexp, pinv
0088            real*8 maxweight, wnorm
0089            character*80 msg
0090    c
0091            antslin(1)   = ants
0092            antswaves(1) = ants
0093            antswaves(2) = waves
0094            wavesants(1) = waves
0095            wavesants(2) = ants
0096            call open_va( oldw_add, antslin, 1, dr_type, status, msg )
0097            call exit_va( status, msg )
0098            call open_va( wa_add, antswaves, 2, dc_type, status, msg )
0099            call exit_va( status, msg )
0100            call open_va( psinv_add, wavesants, 2, dc_type, status, msg )
0101            call exit_va( status, msg )
0102    c
0103            pinv = 1.0 / p
0104            if ( wtfun .eq. 1 ) then                !  Sum of powers.
0105              wexp = - 0.5 * pinv
0106            else                                    !  Sum of powers loss.
0107              wexp = ( p / 2.0 ) - 1.0
0108            end if
0109            maxweight = eps ** wexp
0110    c
0111    c               Compute initial (L2) estimate and residuals therefor.
0112    c
0113            del = tol + .5
0114            call dr_vecfill( weights, ants, 1.0d0 )
```

```
MUSIC_LPW_YAR                                                    18-Apr-1988 14:23:03
                                                                 16-Mar-1988 10:16:26

0115            call dc_mppsinv( a, ants, waves, tol, %val( psinv_add ), info )
0116            if ( info .ne. 0 ) then                  !  Error in psuedo-inverse.
0117              minit = - itlim - 1
0118              go to 10000
0119            end if
0120            call yar_ests( obs, samps, ants, %val( psinv_add ), waves, sspace )
0121            call yar_resids( obs, samps, ants, a, waves, sspace, resids )
0122    c
0123    c                 Iterate to the "Lp" solution, or exit if itlim = 0.
0124    c
0125            minit = itlim
0126            do while ( ( del .gt. tol ) .and. ( minit .gt. 0 ) )
0127              call dr_veccopy( weights, ants, %val( oldw_add ) )
0128    c
0129              do i = 1, ants
0130                weights(i) = 0.0d0
0131                do j = 1, samps
0132                  weights(i) = weights(i) + cdabs( resids(j,i) ) ** p
0133                end do
0134                if ( wtfun .eq. 2 ) weights(i) = weights(i) ** pinv
0135                if ( weights(i) .lt. eps ) then
0136                  weights(i) = maxweight             !  Note using SQRT( weights )..
0137                else
0138                  weights(i) = weights(i) ** wexp    !  Note using SQRT( weights ).
0139                end if
0140              end do
0141    c
0142              if ( norm .gt. 0.0d0 ) then
0143                wnorm = weights(1)
0144                do i = 2, ants
0145                  if ( weights(i) .gt. wnorm ) wnorm = weights(i)
0146                end do
0147                if ( wnorm .gt. 0.0d0 ) then
0148                  do i = 1, ants
0149                    weights(i) = weights(i) / wnorm
0150                  end do
0151                end if
0152              end if
0153    c
0154              call drdi_dc_mmpy( weights, ants, a, waves, %val( wa_add ) )
0155              call dc_mppsinv( %val( wa_add ), ants, waves, tol,
0156             1                                          %val( psinv_add ), info )
0157              if ( info .ne. 0 ) then
0158                minit = - minit - 1
0159                go to 10000
0160              end if
0161              call dc_drdi_mmpy( %val( psinv_add ), waves, ants, weights,
0162             1                                                   %val( psinv_add ) )
0163              call yar_ests( obs, samps, ants, %val( psinv_add ), waves, sspace )
0164              call yar_resids( obs, samps, ants, a, waves, sspace, resids )
0165    c
0166              minit = minit - 1
0167              del = dr_vdel_nmsq( weights, %val( oldw_add ), ants, tol )
0168            end do
0169    c
0170    10000   continue
0171            call free_va( oldw_add, status, msg )
```

```
MUSIC_LPW_YAR                                          18-Apr-1988 14:23:03
                                                       16-Mar-1988 10:16:26

0172            call exit_va( status, msg )
0173            call free_va( wa_add, status, msg )
0174            call exit_va( status, msg )
0175            call free_va( psinv_add, status, msg )
0176            call exit_va( status, msg )
0177    c
0178            return
0179            end


PROGRAM SECTIONS

    Name                                    Bytes   Attributes

  0 $CODE                                    1079   PIC CON REL LCL    SHR    EXE    RD NOWI
  1 $PDATA                                     16   PIC CON REL LCL    SHR NOEXE    RD NOWI
  2 $LOCAL                                    724   PIC CON REL LCL NOSHR NOEXE    RD    WI

    Total Space Allocated                    1819


ENTRY POINTS

    Address  Type  Name

  0-00000000       MUSIC_LPW_YAR


VARIABLES

    Address  Type  Name             Address  Type  Name             Address  Type  N

 AP-0000000C@ I*4  ANTS           2-00000088  I*4  DC_TYPE        AP-0000003C@ R*8  D
  2-00000084  I*4  DR_TYPE       AP-0000001C@ R*8  EPS                 **    I*4  I
 AP-00000034@ I*4  ITLIM              **       I*4  J              2-00000068  R*8  N
  2-00000014  CHAR MSG            AP-00000020@ R*8  NORM         2-00000074  I*4  C
     **       R*4  PINV           2-0000007C  I*4  PSINV_ADD     AP-00000008@ I*4  S
  2-00000070  I*4  STATUS         AP-00000024@ R*8  TOL          AP-00000014@ I*4  U
     **       R*4  WEXP              **       R*8  WNORM        AP-00000040@ I*4  U


ARRAYS

    Address  Type  Name                 Bytes  Dimensions

 AP-00000010@ C*16 A                     **   (*, *)
  2-00000000  I*4  ANTSLIN               4   (1)
  2-00000004  I*4  ANTSWAVES             8   (2)
 AP-00000004@ C*16 OBS                   **   (*, *)
 AP-00000030@ C*16 RESIDS                **   (*, *)
 AP-0000002C@ C*16 SSPACE                **   (*, *)
  2-0000000C  I*4  WAVESANTS             8   (2)
 AP-00000028@ R*8  WEIGHTS               **   (*)
```

18-Apr-1988 14:23:03
16-Mar-1988 10:16:26

```
c
        subroutine yar_ests( obs, samps, ants, psinv, waves, sspace )
c
c<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
c
c               This subroutine uses the "psuedo-inverse" psinv (computed by
c       MUSIC_LPW_YAR, above) to replace the COLUMNS of sspace with estimates
c       of signal-in-space vector samples.  The ROWS of obs are used to form
c       the estimates.  The matrix psinv is waves x ants COMPLEX*16.
c
c<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
c
        implicit none
c
        integer*4 samps, ants, waves
        complex*16 obs( samps, ants ), psinv( waves, ants )
        complex*16 sspace( waves, samps )
c
        integer*4 i, j, k
c
        do i = 1, samps
          do j = 1, waves
            sspace(j,i) = ( 0.0d0, 0.0d0 )
            do k = 1, ants
              sspace(j,i) = sspace(j,i) + psinv(j,k) * obs(i,k)
            end do
          end do
        end do
c
        return
        end
```

18-Apr-1988 14:23:03
16-Mar-1988 10:16:26

```
c
        subroutine yar_resids( obs, samps, ants, a, waves, sspace, resids )
c
c<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
c
c               This subroutine computes the residuals due to estimates of
c       the signal-in-space vector samples, for the "Yarlagadda" algorithm.
c
c<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
c
        implicit none
c
        integer*4 samps, ants, waves
        complex*16 obs( samps, ants ), a( ants, waves )
        complex*16 sspace( waves, samps ), resids( samps, ants )
c
        integer*4 i, j, k
c
        do i = 1, samps
          do j = 1, ants
            resids(i,j) = obs(i,j)
            do k = 1, waves
              resids(i,j) = resids(i,j) - a(j,k) * sspace(k,i)
            end do
          end do
        end do
c
        return
        end
```

```
                                                    18-Apr-1988 14:23:4
                                                    21-Mar-1988 16:37:4

0001            logical function
0002            1       music_plot_spect( apars, num_ants, wavelen, eigvec, select
0003            2                                 mi_begp, mi_delp, fskip, count, isk
0004    c
0005    c-------------------------------------------------------------------------
0006    c
0007    c               This function plots MUSIC "power" spectra according to the
0008    c       user's instructions.  The function returns .true. if the operator
0009    c       enters X (to stop DOA spectrum plots) in response to the "do you w
0010    c       a replot?" prompt, and .false. otherwise.
0011    c
0012    c       Parameters:
0013    c       -----------
0014    c               apars = matrix whose first column is the vector of ranges
0015    c                       the origin) of array antenna elements and
0016    c                       second column is the vector of angles (fro
0017    c                       "x" axis) of array antenna elements (in
0018    c                       RADIANS).  num_ants x 2.  REAL*8.
0019    c                  Note:  the position of the ith antenna element is
0020    c                  apars(i,1) * cos( apars(i,2) ) (x position),
0021    c                  apars(i,1) * sin( apars(i,2) ) (y position).
0022    c               num_ants = number of elements in antenna array.  INTEGER*4
0023    c               wavelen = wavelength of the incident waves.  REAL*8.  (Not
0024    c                       that the units of wavelength must be the s
0025    c                       as those in which the array_range(*) are
0026    c                       expressed.)
0027    c               eigvec:  num_ants x num_ants COMPLEX*16 matrix whose colum
0028    c                       are the applicable generalized eigenvector
0029    c                       The eigenvectors are ASSUMED to be arrange
0030    c                       ascending eigenvalue order, with the
0031    c                       eigenvector corresponding to the smallest
0032    c                       eigenvalue occupying column #1, and so on.
0033    c               select:  INTEGER*4 number of eigenvectors (columns of eigv
0034    c                       to be used in the computation (number of
0035    c                       "noise" eigenvectors).
0036    c               mi_begp:  REAL*4 minimum permitted value of the power (p)
0037    c                       which to raise the individual terms of the
0038    c                       which makes up the reciprocal of a MUSIC D
0039    c                       spectrum point (p = 2 in "ordinary" MUSIC)
0040    c               mi_delp:  REAL*4 minimum increment value for p.
0041    c               fskip:  INTEGER*4 number of p values to skip before the fi
0042    c                       value for which to plot.
0043    c               count:  INTEGER*4 total number of p values for which to pl
0044    c               iskip:  INTEGER*4 number of p values to skip between value
0045    c                       which to plot.
0046    c
0047    c-------------------------------------------------------------------------
0048    c
0049            implicit none
0050            external music_spectrumx, music_spectrumy, vecext, log_scale
0051            external cap, r4_col_ins, r4_col_ext, value_at
0052            external open_va, exit_va, free_va
0053            external tek_start, tek_grid, tek_title, tek_plot, tek_vec2, tek_fi
0054            external tek_point, tek_label
0055            include '6000$RT:[CHUCK.RESEARCH.FORTDIR]VATYPES.TXT'
0067            include '(real_constants)'
0118    c
```

```
MUSIC_PLOT_SPECT                                                    18-Apr-1988 14:23:41
                                                                    21-Mar-1988 16:37:45

0119            character*1 cap
0120            real*4 value_at
0121    c
0122            integer*4 num_ants, select, fskip, count, iskip
0123            real*4 mi_begp, mi_delp
0124            real*8 apars(num_ants,1), wavelen
0125            complex*16 eigvec(num_ants,num_ants)
0126    c
0127            logical replot
0128            integer*4 ch_spc/ 12 /
0129            integer*4 points/ 101 /, x_add, y_add, x_tics/ 5 /, max_tics/ 21 /
0130            integer*4 y_tics, linear( 1 ), status, pts_by_ct( 2 ), plots_add
0131            integer*4 i, ixat, iyat
0132            real*4 phi_start/ - pi4 /, phi_end/ pi4 /, ymin, ymax
0133            real*4 deg_start/ -180. /, deg_end/ 180. /, p, ypeak, yvaly, yat
0134            character*5  numlab
0135            character*10 answer
0136            character*80 msg
0137    c
0138            save points, phi_start, deg_start, phi_end, deg_end, x_tics
0139    c
0140            replot = .true.
0141            do while ( replot )
0142              type *
0143              type 90040, ' Plot MUSIC spectrum from what starting angle (',
0144           1                                        deg_start, ') (degrees)?  '
0145              accept 90000, answer
0146              read (unit=answer,fmt=*,err=10000) deg_start
0147              phi_start = deg_start * rad_p_deg4
0148    10000     continue
0149    c
0150              type 90040, '                            to what ending angle (',
0151           1                                        deg_end, ') (degrees)?  '
0152              accept 90000, answer
0153              read (unit=answer,fmt=*,err=10100) deg_end
0154              phi_end = deg_end * rad_p_deg4
0155    10100     continue
0156    c
0157              type 90030, ' Number of points (angles) at which to plot (',
0158           1                                                    points, ')?  '
0159              accept 90000, answer
0160              read (unit=answer,fmt=*,err=10200) points
0161    10200     continue
0162    c
0163              type 90030, ' Number of vertical grid lines to place on the plot (',
0164           1                                                    x_tics, ')?  '
0165              accept 90000, answer
0166              read (unit=answer,fmt=*,err=10300) x_tics
0167    10300     continue
0168    c
0169              if ( phi_start .ge. phi_end ) go to 2000
0170              if ( points .lt. 6 ) go to 2000
0171              if ( x_tics .lt. 0 ) go to 2000
0172              if ( x_tics .gt. max_tics ) go to 2000
0173    c
0174                linear(1)    = points
0175                pts_by_ct(1) = points
```

MUSIC_PLOT_SPECT 18-Apr-1988 14:23:41
21-Mar-1988 16:37:45

```
0176                pts_by_ct(2) = count
0177                call open_va( x_add, linear, 1, sr_type, status, msg )
0178                call exit_va( status, msg )
0179                call open_va( y_add, linear, 1, sr_type, status, msg )
0180                call exit_va( status, msg )
0181                call open_va( plots_add, pts_by_ct, 2, sr_type, status, msg )
0182                call exit_va( status, msg )
0183    c
0184                call music_spectrumx( phi_start, phi_end, points, %val( x_add ) )
0185                do i = 1, count
0186                  p = mi_begp + ( fskip + ( i - 1 ) * ( iskip + 1 ) ) * mi_delp
0187                  type 90060, 'p =', p, '.'
0188    c
0189                  call music_spectrumy( points, apars, num_ants, wavelen,
0190           1                                eigvec, select, p, %val( x_add ),
0191           2                                                    %val( y_add ) )
0192                  call r4_col_ins( %val( y_add ), points, %val( plots_add ), i )
0193    c
0194                  call vecext( %val( y_add ), points, ymax, ymin )
0195                  if ( i .eq. 1 ) then
0196                    ypeak = ymax
0197                    yvaly = ymin
0198                  else
0199                    if ( ymax .gt. ypeak ) ypeak = ymax
0200                    if ( ymin .lt. yvaly ) yvaly = ymin
0201                  end if
0202                end do
0203                call log_scale( ypeak, yvaly, ymax, ymin, y_tics )
0204    c
0205                type 90050, ' Ready to plot for phi from ',
0206           1                                deg_start, ' to ', deg_end, ' degrees.'
0207                accept 90000, answer
0208                call tek_start
0209                call tek_grid( 200, 150, 922, 691, deg_start, ymin,
0210           1                    deg_end, ymax, x_tics, y_tics, x_tics, y_tics, 0, 1 )
0211                call tek_title( 'Arrival Angle', 'DOA Measure',
0212           1                                      'DOA SPECTRUM FOR SELECTED p' )
0213    c
0214                call vecscale( %val( x_add ), points, deg_p_rad4 )
0215                do i = 1, count
0216                  call r4_col_ext( %val( plots_add ), points, i, %val( y_add ) )
0217                  call tek_plot( %val( x_add ), %val( y_add ), points, 1 )
0218    c
0219                  yat = value_at( %val( y_add ), points, .4 )
0220                  call tek_point( deg_end, yat, ixat, iyat )
0221                  p = mi_begp + ( fskip + ( i - 1 ) * ( iskip + 1 ) ) * mi_delp
0222                  write(unit=numlab,fmt=90010) p
0223                  call tek_label( numlab( 2 : 5 ), 4, ixat, iyat - 4, ch_spc, 0 )
0224                end do
0225                call tek_vec2( 1, 1, 1, 1, 0 )
0226    c
0227                call tek_finish( .false. )
0228                accept 90000, answer
0229    c
0230                call free_va( x_add, status, msg )
0231                call exit_va( status, msg )
0232                call free_va( y_add, status, msg )
```

```
MUSIC_PLOT_SPECT                                                    18-Apr-1988 14:23:41
                                                                    21-Mar-1988 16:37:45

0233                call exit_va( status, msg )
0234                call free_va( plots_add, status, msg )
0235                call exit_va( status, msg )
0236              go to 1000
0237    c
0238    2000      continue
0239                type *, char( 7 )
0240                type *, 'Couldn''t plot DOA spectrum using the selected ',
0241             1                                                    'parameters.'
0242                type *, 'Check for insane phi range or number of points.'
0243    c
0244    1000      continue
0245              type *
0246              type *, 'Enter X (or x) to abort DOA plotting:'
0247              type 90020, '        Do you wish a replot (new phi range) '
0248             1                                                  // '(y/[n]/x)?  '
0249              accept 90000, answer
0250              replot = ( cap( answer( 1 : 1 ) ) .eq. 'Y' )
0251            end do
0252    c
0253            music_plot_spect = ( cap( answer( 1 : 1 ) ) .eq. 'X' )
0254            return
0255    c
0256    90000   format( a120 )
0257    90010   format( f5.2 )
0258    90020   format( $a )
0259    90030   format( $a, i11, a )
0260    90040   format( $a, f7.2, a )
0261    90050   format( $ 2( a, f6.1 ), a )
0262    90060   format( 1x, a, f5.2, a )
0263    c
0264            end
```

18-Apr-1988 14:23:41
21-Mar-1988 16:37:45

```
c
        real*4 function value_at( vec, ents, frac )
c
c<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><
c<
c<              This function returns element number 1 + ( ents - 1 ) * frac
c<      of the REAL*4 vector vec.  The index ents is INTEGER*4, and the
c<      parameter frac is REAL*4.  Obviously, the expression is truncated to
c<      an integer before being used to select the element to be returned.
c<      If truncation would result in a value <= 0, the first element is
c<      returned; and if it would result in a value > ents, the ents-th
c<      element is returned.
c<
c<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><
c
        implicit none
c
        integer*4 ents
        real*4 vec( ents ), frac
c
        integer*4 sel
c
        sel = 1 + ( ents - 1 ) * frac
        if      ( sel .le. 0 ) then
          sel = 1
        else if ( sel .gt. ents ) then
          sel = ents
        end if
c
        value_at = vec(sel)
        return
        end
```

```
                                                                    18-Apr-1988 14:24:15    VA
                                                                    20-Feb-1988 14:33:56    MU

0001            real*8 function music_power( eigvec, steer, n, m, p )
0002    c
0003    c-------------------------------------------------------------------------------
0004    c
0005    c               This function returns the MUSIC "power spectrum" sample
0006    c       selected by steering vector steer and "power" exponent p.  Division by
0007    c       zero and floating-point underflow and overflow are handled, and the
0008    c       value returned may be inverted without encountering division by zero or
0009    c       floating-point underflow or overflow.
0010    c
0011    c       Parameters:
0012    c       -----------
0013    c               eigvec:  n x n COMPLEX*16 matrix whose FIRST m columns are the
0014    c                               "noise" generalized eigenvectors defined in the
0015    c                               MUSIC algorithm.
0016    c               steer:  n-element COMPLEX*16 steering vector (for the chosen
0017    c                               look direction).
0018    c               n and m:  INTEGER*4 index values.
0019    c               p:  REAL*4 power to which to raise the terms used in
0020    c                               accumulating the DOA spectrum sample (p = 2 for
0021    c                               "ordinary" MUSIC).
0022    c
0023    c-------------------------------------------------------------------------------
0024    c
0025            implicit none
0026            external lib$establish, mpwr_fl5
0027            external mpwr_dpdtrm, mpwr_dpdsum, mpwr_pwrtrm, mpwr_pwrsum
0028            include '(fortran_limits)'
0057    c
0058            integer*4 lib$establish
0059            integer*4 mpwr_dpdtrm, mpwr_dpdsum
0060            real*8 mpwr_pwrtrm, mpwr_pwrsum
0061    c
0062            real*4 pb2lim
0063            parameter (pb2lim = 2.0 * minrecip4)
0064    c
0065            integer*4 n, m
0066            real*4 p
0067            complex*16 eigvec(n,n), steer(n)
0068    c
0069            integer*4 i, j, old_handler
0070            real*8 pb2, pwr_term
0071            complex*16 dpd_term, dotprod
0072    c
0073            if ( p .ge. pb2lim ) then
0074              pb2 = p / 2.0d0
0075            else
0076              pb2 = 0.0d0
0077            end if
0078            music_power = 0.0d0
0079    c
0080            do i = 1, m
0081              dotprod = ( 0.0d0, 0.0d0 )
0082    c
0083              do j = 1, n
0084                if ( mpwr_dpdtrm( eigvec(j,i), steer(j), dpd_term ) )
0085            1                                                   10000, 10010, 10020
```

```
MUSIC_POWER                                                    18-Apr-1988 14:24:15     \
                                                               20-Feb-1988 14:33:56     \

0086     10000       continue                                                ! Underflow.
0087                   dpd_term = ( 0.0d0, 0.0d0 )
0088                   go to 10010
0089     10020       continue                                                ! Overflow.
0090                   music_power = minrecip4
0091                   return                                                ! Error EXIT!
0092     10010       continue                                                ! No error.
0093     c
0094                   if ( mpwr_dpdsum( dotprod, dpd_term ) ) 11000, 11010, 10020
0095     11000       continue                                                ! Underflow.
0096                   dotprod = ( 0.0d0, 0.0d0 )
0097     11010       continue                                                ! No error.
0098                 end do
0099     c
0100                 pwr_term    = mpwr_pwrtrm( dotprod, pb2 )
0101                 music_power = mpwr_pwrsum( music_power, pwr_term )
0102               end do
0103     c
0104               old_handler = lib$establish( mpwr_fl5 )
0105               music_power = 1.0d0 / music_power
0106               call lib$establish( old_handler )
0107     c
0108               if      ( music_power .lt. minrecip4 ) then
0109                 music_power = minrecip4
0110               else if ( music_power .gt. maxrecip4 ) then
0111                 music_power = maxrecip4
0112               end if
0113     c
0114               return
0115               end
```

18-Apr-1988 14:24:15
20-Feb-1988 14:33:56

```
c
        integer*4 function mpwr_f15( sigargs, mechargs )
c
c!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
c                                                                      !
c               This is the condition handler in effect during the divide   !
c       in function MUSIC_POWER, above.  If floating point underflow occurs, !
c       this routine forces MUSIC_POWER to exit returning the minimum REAL*4  !
c       value which can be inverted.  If floating point overflow or division  !
c       by zero occurs, this routine forces MUSIC_POWER to exit returning the !
c       maximum invertible REAL*4 value.                                      !
c                                                                             !
        implicit none                                                         !
        external sys$unwind, lib$match_cond, s_to_dr                          !
        include 'SYS$LIBRARY:SIGDEF'                                          !
        include 'SYS$LIBRARY:MTHDEF'                                          !
        include '(fortran_limits)'                                            !
c                                                                             !
        integer*4 lib$match_cond                                              !
c                                                                             !
        integer*4 sigargs( * ), mechargs( * )                                 !
c                                                                             !
        integer*4 i                                                           !
c                                                                             !
        i = lib$match_cond( sigargs(2),                                       !
        1                   SS$_UNWIND,     ! unwind in progress.             !
        2                   SS$_FLTUND,     ! floating underflow trap.        !
        3                   SS$_FLTUND_F,   ! floating underflow fault.       !
        4                   SS$_FLTOVF,     ! floating overflow trap.         !
        5                   SS$_FLTDIV,     ! floating div. by zero trap.     !
        6                   SS$_FLTOVF_F,   ! floating overflow fault.        !
        7                   SS$_FLTDIV_F    ! floating div. by zero fault.    !
        8                   )                                                 !
c                                                                             !
        if      ( i .eq. 0 ) then               !  Unknown condition          !
          mpwr_f15 = SS$_RESIGNAL                                             !
        else if ( i .eq. 1 ) then               !  Unwinding.                 !
        else if ( i .lt. 4 ) then               !  Underflow.                 !
          call s_to_dr( mechargs(4), minrecip4 )                              !
          call sys$unwind( , )                                                !
        else                                    !  Overflow.                  !
          call s_to_dr( mechargs(4), maxrecip4 )                              !
          call sys$unwind( , )                                                !
        end if                                                                !
c                                                                             !
        return                                                                !
c                                                                             !
c!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
c
        end
```

18-Apr-1988 14:24:15
20-Feb-1988 14:33:56

```
c
        integer*4 function mpwr_dpdtrm( eigvec_elem, steer_elem, result )
c
c!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
c                                                                       !
c               This function places the product                        !
c       dconjg( eigvec_elem ) * steer_elem in the COMPLEX*16 variable
c       result.  Arithmetic exceptions are processed by the function
c       MPWR_FL1.  The parameters eigvec_elem and steer_elem are COMPLEX*16.
c               The function returns -1 if arithmetic underflow occurred, 0
c       if no arithmetic errors occurred, and +1 if arithmetic overflow or
c       divide by zero occurred.
c
        implicit none
        intrinsic dconjg
        external lib$establish, mpwr_fl1
c
        integer*4 lib$establish
c
        complex*16 eigvec_elem, steer_elem, result
c
        integer*4 old_handler
c
        old_handler = lib$establish( mpwr_fl1 )
        result = dconjg( eigvec_elem ) * steer_elem
        call lib$establish( old_handler )
c
        mpwr_dpdtrm = 0
        return
c
        end
```

```
                                                                    18-Apr-1988 14:24:15    \
                                                                    20-Feb-1988 14:33:56    [
0001    c                                                                                   !
0002            integer*4 function mpwr_fl1( sigargs, mechargs )                            !
0003    c                                                                                   !
0004    c.............  Condition handler for MPWR_DPDTRM and MPWR_DPDSUM.  ..........!
0005    c                                                                                   !
0006            implicit none                                                               !
0007            external sys$unwind, lib$match_cond                                         !
0008            include 'SYS$LIBRARY:SIGDEF'                                                !
0068            include 'SYS$LIBRARY:MTHDEF'                                                !
0110    c                                                                                   !
0111            integer*4 lib$match_cond                                                    !
0112    c                                                                                   !
0113            integer*4 sigargs( * ), mechargs( * )                                       !
0114    c                                                                                   !
0115            integer*4 i                                                                 !
0116    c                                                                                   !
0117            i = lib$match_cond( sigargs(2),                                             !
0118            1                         SS$_UNWIND,      ! unwind in progress.          !
0119            2                         SS$_FLTUND,      ! floating underflow trap.     !
0120            3                         SS$_FLTUND_F,    ! floating underflow fault.    !
0121            4                         SS$_FLTOVF,      ! floating overflow trap.      !
0122            5                         SS$_FLTDIV,      ! floating div. by zero trap.  !
0123            6                         SS$_FLTOVF_F,    ! floating overflow fault.     !
0124            7                         SS$_FLTDIV_F     ! floating div. by zero fault.!
0125            8                         )                                               !
0126    c                                                                                   !
0127            if      ( i .eq. 0 ) then                 !  Unknown condition            !
0128              mpwr_fl1 = SS$_RESIGNAL                                                   !
0129            else if ( i .eq. 1 ) then                 !  Unwinding.                   !
0130            else if ( i .lt. 4 ) then                 !  Underflow.                   !
0131              mechargs(4) = -1                                                          !
0132              call sys$unwind( , )                                                      !
0133            else                                      !  Overflow.                    !
0134              mechargs(4) = 1                                                           !
0135              call sys$unwind( , )                                                      !
0136            end if                                                                  !
0137    c                                                                             !
0138            return                                                              !
0139    c                                                                           !
0140    c!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
0141    c
0142            end
```

13-Apr-1988 14:24:15
20-Feb-1988 14:33:56

```
c
        integer*4 function mpwr_dpdsum( dotprod, dpd_term )
c
c!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
c                                                                        !
c               This function places the sum of the COMPLEX*16 values    !
c       dotprod and dpd_term in dotprod.  Arithmetic exceptions are      !
c       processed by the condition handler MPWR_FL1, above.              !
c               The function returns -1 if floating point underflow occurred,
c       zero if no errors occurred, and +1 if floating point overflow or
c       division by zero occurred.
c
        implicit none
        external lib$establish, mpwr_fl1
c
        integer*4 lib$establish
c
        complex*16 dotprod, dpd_term
c
        integer*4 old_handler
c
        old_handler = lib$establish( mpwr_fl1 )
        dotprod = dotprod + dpd_term
        call lib$establish( old_handler )
c                                                                        !
        mpwr_dpdsum = 0                                                  !
        return                                                           !
c                                                                        !
c!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
c
        end
```

18-Apr-1988 14:24:15
20-Feb-1988 14:33:56

```
c
        real*8 function mpwr_pwrtrm( dotprod, pb2 )
c
c!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
c                                                                      !
c               This function returns the value of the expression      !
c       dreal( dotprod * dconjg( dotprod ) ) ** pb2, where dotprod is   !
c       COMPLEX*16 and pb2 is REAL*8.  Arithmetic exceptions are processed
c       by the condition handler MPWR_FL3, below.  If floating point
c       underflow occurs, the function return value is corrected to zero.  If
c       floating point overflow or divide by zero occur, the maximum
c       invertible REAL*4 value is returned as the function result.
c
        implicit none
        intrinsic dreal, dconjg
        external lib$establish, mpwr_fl3
c
        integer*4 lib$establish
c
        real*8 pb2
        complex*16 dotprod
c
        integer*4 old_handler
c
        old_handler = lib$establish( mpwr_fl3 )
        mpwr_pwrtrm = dreal( dotprod * dconjg( dotprod ) ) ** pb2
        call lib$establish( old_handler )
c
        return
        end
```

```
                                                                    18-Apr-1988 14:24:15
                                                                    20-Feb-1988 14:33:56

0001    c                                                                               !
0002            integer*4 function mpwr_fl3( sigargs, mechargs )                        !
0003    c                                                                               !
0004    c.............  Condition handler for MPWR_PWRTRM and MPWR_PWRSUM.  ...........!
0005    c                                                                               !
0006            implicit none                                                           !
0007            external sys$unwind, lib$match_cond, s_to_dr                            !
0008            include 'SYS$LIBRARY:SIGDEF'                                            !
0068            include 'SYS$LIBRARY:MTHDEF'                                            !
0110            include '(fortran_limits)'                                              !
0139    c                                                                               !
0140            integer*4 lib$match_cond                                                !
0141    c                                                                               !
0142            integer*4 sigargs( * ), mechargs( * )                                   !
0143    c                                                                               !
0144            integer*4 i                                                             !
0145    c                                                                               !
0146            i = lib$match_cond( sigargs(2),                                         !
0147            1                       SS$_UNWIND,     ! unwind in progress.         !
0148            2                       SS$_FLTUND,     ! floating underflow trap.    !
0149            3                       SS$_FLTUND_F,   ! floating underflow fault.   !
0150            4                       SS$_FLTOVF,     ! floating overflow trap.     !
0151            5                       SS$_FLTDIV,     ! floating div. by zero trap. !
0152            6                       SS$_FLTOVF_F,   ! floating overflow fault.    !
0153            7                       SS$_FLTDIV_F    ! floating div. by zero fault.!
0154            8                       )                                               !
0155    c                                                                               !
0156            if      ( i .eq. 0 ) then               !  Unknown condition            !
0157              mpwr_fl3 = SS$_RESIGNAL                                               !
0158            else if ( i .eq. 1 ) then               !  Unwinding.                   !
0159            else if ( i .lt. 4 ) then               !  Underflow.                   !
0160              call s_to_dr( mechargs(4), 0.0 )                                      !
0161              call sys$unwind( , )                                                  !
0162            else                                    !  Overflow.                    !
0163              call s_to_dr( mechargs(4), maxrecip4 )                                !
0164              call sys$unwind( , )                                                  !
0165            end if                                                                  !
0166    c                                                                               !
0167            return                                                                  !
0168            end                                                                     !
```

```
                                                        18-Apr-1988 14:24:15
                                                        20-Feb-1988 14:33:56

0001     c                                                                     !
0002              subroutine s_to_dr( destination, source )                    !
0003     c                                                                     !
0004     c.......          Executes destination = source, REAL*8 destination, ..........!
0005     c....... REAL*4 source.                                               ..........!
0006     c                                                                     !
0007              implicit none                                                !
0008     c                                                                     !
0009              real*4 source                                                !
0010              real*8 destination                                           !
0011     c                                                                   !
0012              destination = source                                      !
0013              return                                                   !
0014     c                                                               !
0015     c!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
0016     c
0017              end


PROGRAM SECTIONS

     Name                                    Bytes   Attributes

   0 $CODE                                       8   PIC CON REL LCL   SHR   EXE   RD NOWRT

     Total Space Allocated                       8


ENTRY POINTS

     Address  Type  Name

   0-00000000       S_TO_DR


VARIABLES

     Address  Type  Name              Address  Type  Name

  AP-00000004@ R*8  DESTINATION    AP-00000008@ R*4  SOURCE
```

```
c
        real*8 function mpwr_pwrsum( pwr_sum, pwr_term )
c
c!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
c                                                                          !
c               This function returns the sum of the REAL*8 values pwr_sum !
c       and pwr_term.  Arithmetic exceptions are processed by condition    !
c       handler MPWR_FL3, above.  Arithmetic underflow causes the function !
c       to return 0.0, and arithmetic overflow and division by zero cause the !
c       function to return the largest invertible REAL*4 value.            !
c                                                                          !
        implicit none                                                      !
        external lib$establish, mpwr_fl3                                   !
c                                                                          !
        integer*4 lib$establish                                            !
c                                                                          !
        real*8 pwr_sum, pwr_term                                           !
c                                                                          !
        integer*4 old_handler                                              !
c                                                                          !
        old_handler = lib$establish( mpwr_fl3 )                            !
        mpwr_pwrsum = pwr_sum + pwr_term                                   !
        call lib$establish( old_handler )                                  !
c                                                                          !
        return                                                             !
c                                                                          !
c!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
c
        end
```

```
                                                    18-Apr-1988 14:24:39    VA
                                                     5-Jan-1988 08:14:47    MU
0001            subroutine music_spectrumx( phi_start, phi_end, phi_num, x_spectrum )
0002    c
0003    c-------------------------------------------------------------------------------
0004    c
0005    c               This subroutine produces abscissa values for the MUSIC "power
0006    c       spectrum" for arrival angles between phi_start and phi_end, inclusive.
0007    c
0008    c       Parameters:
0009    c       -----------
0010    c               phi_start:  REAL*4 minimum angle of arrival for which to
0011    c                           compute the "power" spectrum.
0012    c               phi_end:  REAL*4 maximum angle of arrival for which to compute
0013    c                           the "power" spectrum.
0014    c               phi_num:  INTEGER*4 number of spectrum samples to compute.
0015    c                           Samples will be computed at intervals of
0016    c                           ( phi_end - phi_start ) / ( phi_num - 1 ).
0017    c               x_spectrum:  phi_num-element REAL*4 output vector of abscissa
0018    c                           (phi) values.
0019    c
0020    c-------------------------------------------------------------------------------
0021    c
0022            implicit none
0023    c
0024            integer*4 phi_num
0025            real*4 phi_start, phi_end, x_spectrum(phi_num)
0026    c
0027            integer*4 i
0028            real*4 phi_del
0029    c
0030            phi_del = ( phi_end - phi_start ) / ( phi_num - 1 )
0031            do i = 1, phi_num
0032              x_spectrum(i) = phi_start + ( i - 1 ) * phi_del
0033            end do
0034    c
0035            return
0036            end
```

```
                                                    18-Apr-1988 14:24:51     V
                                                    20-Feb-1988 14:35:00     M
0001            subroutine music_spectrumy( phi_num, apars, num_ants, wavelen, eigvec,
0002            1                                   num_zeigs, p, x_spectrum, y_spectrum )
0003    c
0004    c-------------------------------------------------------------------------------
0005    c
0006    c               This subroutine produces ordinate values for the MUSIC "power
0007    c       spectrum" for arrival angles in vector x_spectrum, given the applicable
0008    c       generalized eigenvectors and the desired power to which to raise the
0009    c       spectrum point magnitudes.
0010    c
0011    c       Parameters:
0012    c       -----------
0013    c               phi_num:  INTEGER*4 number of spectrum samples to compute
0014    c                               (number of elements in x_spectrum and
0015    c                               y_spectrum).
0016    c               apars = matrix whose first column is the vector of ranges (from
0017    c                               the origin) of array antenna elements, and
0018    c                               whose second column is the vector of angles
0019    c                               (from the "x" axis) of array antenna elements
0020    c                               (in RADIANS).  num_ants x 2.  REAL*8.
0021    c                       Note:  the position of the ith antenna element is
0022    c                       apars(i,1) * cos( apars(i,2) ) (x position),
0023    c                       apars(i,1) * sin( apars(i,2) ) (y position).
0024    c               num_ants = number of elements in antenna array.  INTEGER*4.
0025    c               wavelen = wavelength of the incident waves.  REAL*8.  (Note
0026    c                               that the units of wavelength must be the same
0027    c                               as those in which the array_range(*) are
0028    c                               expressed.)
0029    c               eigvec:  num_ants x num_ants COMPLEX*16 matrix whose FIRST
0030    c                               - num_zeigs columns are the applicable "noise"
0031    c                               generalized eigenvectors.
0032    c               num_zeigs:  INTEGER*4 number of zero generalized eigenvalues
0033    c                               (number of eigenvectors to use in the
0034    c                               computation).
0035    c               p:  REAL*4 power to which to raise the magnitudes of the terms
0036    c                               which make up the reciprocal of a MUSIC DOA
0037    c                               spectrum point (p = 2 for "ordinary" MUSIC).
0038    c               x_spectrum:  phi_num-element REAL*4 vector of abscissa (phi)
0039    c                               values for which to compute the DOA spectrum
0040    c                               values.
0041    c               y_spectrum:  phi_num-element REAL*4 output vector of ordinate
0042    c                               ("power" spectrum sample) values.
0043    c
0044    c-------------------------------------------------------------------------------
0045    c
0046            implicit none
0047            intrinsic sngl, dble
0048            external music_power, sigval
0049            external open_va, exit_va, free_va
0050            include '6000$RT:[CHUCK.RESEARCH.FORTDIRIVATYPES.TXT'
0062    c
0063            real*8 music_power
0064    c
0065            integer*4 phi_num, num_ants, num_zeigs
0066            real*4 p, x_spectrum(phi_num), y_spectrum(phi_num)
0067            real*8 apars(num_ants,1), wavelen
0068            complex*16 eigvec(num_ants,num_ants)
```

```
MUSIC_SPECTRUMY                                          18-Apr-1988 14:24:51     VA
                                                         20-Feb-1988 14:35:00     MU

0069     c
0070             integer*4 i, steerv_add, linear(1), status
0071             character*80 msg
0072     c
0073             linear(1) = num_ants
0074             call open_va( steerv_add, linear, 1, dc_type, status, msg )
0075             call exit_va( status, msg )
0076     c
0077             do i = 1, phi_num
0078               call sigval( apars, num_ants, dble( x_spectrum(i) ), wavelen,
0079             1                                                %val( steerv_add ) )
0080               y_spectrum(i) = sngl( music_power( eigvec, %val( steerv_add ),
0081             1                                    num_ants, num_zeigs, p ) )
0082             end do
0083     c
0084             call free_va( steerv_add, status, msg )
0085             call exit_va( status, msg )
0086     c
0087             return
0088             end


PROGRAM SECTIONS

    Name                                  Bytes   Attributes

  0 $CODE                                   270   PIC CON REL LCL   SHR   EXE   RD NOWRT LOI
  1 $PDATA                                    8   PIC CON REL LCL   SHR NOEXE   RD NOWRT LOI
  2 $LOCAL                                  368   PIC CON REL LCL NOSHR NOEXE   RD   WRT LOI

    Total Space Allocated                   646


ENTRY POINTS

    Address  Type  Name

  0-00000000       MUSIC_SPECTRUMY


VARIABLES

    Address  Type  Name              Address  Type  Name              Address  Type  Name

      **      I*4  I               2-00000004  CHAR MSG            AP-0000000C@ I*4  NUM_AI
 AP-0000001C@ R*4  P               AP-00000004@ I*4  PHI_NUM         2-00000058  I*4  STATUS
 AP-00000010@ R*8  WAVELEN
```

18-Apr-1988 14:25:09 VAX I
24-Sep-1987 14:59:07 NOISI

```
0001            subroutine noise_combine( mat, n, m, vec, defs, out )
0002      c
0003      c-------------------------------------------------------------------------------
0004      c
0005      c                  This subroutine multiplies the n x m REAL*8 matrix mat by the
0006      c          m-element COMPLEX*8 vector vec to produce the n-element COMPLEX*16
0007      c          vector out.  The multiplication INCLUDES elements of vec whose
0008      c          corresponding types (high words of first column of n x 2 INTEGER*4
0009      c          matrix defs) are 2 (indicating "burst" noise).  These elements are
0010      c          ALSO added to randomly selected elements of out, so the sources are
0011      c          "burst" sources ONLY IF THE APPROPRIATE ENTRIES IN mat ARE ZERO.
0012      c
0013      c-------------------------------------------------------------------------------
0014      c
0015            implicit none
0016            intrinsic jishft
0017            external uni
0018      c
0019            real*4 uni
0020      c
0021            integer*4 n, m, defs( m, 1 )
0022            real*8 mat( n, m )
0023            complex*8 vec( m )
0024            complex*16 out( n )
0025      c
0026            integer*4 i, j
0027      c
0028            do i = 1, n
0029              out(i) = ( 0.0d0, 0.0d0 )
0030              do j = 1, m
0031                out(i) = out(i) + mat(i,j) * vec(j)
0032              end do
0033            end do
0034      c
0035            do i = 1, m
0036              if ( ( jishft( defs(i,1), -16 ) .eq. 2 )
0037           1       .and. ( vec(i) .ne. ( 0.0, 0.0 ) ) ) then
0038                j = 1 + n * uni( 10571 )
0039                if ( j .gt. n ) j = n
0040                out(j) = out(j) + vec(i)
0041              end if
0042            end do
0043      c
0044            return
0045            end
```

```
                                                    18-Apr-1988 14:25:31     VAX
                                                    19-Dec-1987 09:16:11     OBS

0001            subroutine obsvec( sig_mat, num_ants, num_waves, ndefs, npars,
0002            1                        correl, noises, andefs, anpars, acorrel,
0003            2                        anoises, sig_vec, clean, obs, noise, select )
0004      c
0005      c-------------------------------------------------------------------------------
0006      c
0007      c               This subroutine computes a noisy observation vector from an
0008      c       array of antenna elements in the presence of multiple incident radar
0009      c       waves.
0010      c
0011      c       Parameters:
0012      c       -----------
0013      c               sig_mat = num_ants x num_waves COMPLEX*16 matrix of phase
0014      c                         shifts due to antenna position ("signal
0015      c                         matrix").
0016      c               num_ants = number of antenna elements in the array.  INTEGER*4.
0017      c               num_waves = INTEGER*4 number of waves present.
0018      c               ndefs = noises x 2 INTEGER*4 matrix whose first column is the
0019      c                         vector of noise types (see subroutine
0020      c                         READ_NOISE), and whose second column is the
0021      c                         vector of counters for intermittent noise.
0022      c                         This matrix applies to the ESTIMATION NOISE
0023      c                         MODEL.
0024      c               npars = noises x 3 REAL*4 array of noise parameters (see
0025      c                         subroutine READ_NOISE).  This matrix applies to
0026      c                         the ESTIMATION NOISE MODEL.
0027      c               correl = num_ants x noises REAL*8 matrix of noise source
0028      c                         multipliers for each antenna element.  This
0029      c                         matrix applies to the ESTIMATION NOISE MODEL.
0030      c               noises = INTEGER*4 number of noise sources used in the
0031      c                         ESTIMATION NOISE MODEL.
0032      c               andefs = anoises x 2 INTEGER*4 matrix whose first column is the
0033      c                         vector of noise types (see subroutine
0034      c                         READ_NOISE), and whose second column is the
0035      c                         vector of counters for intermittent noise.
0036      c                         This matrix applies to ACTUAL SAMPLE
0037      c                         GENERATION.
0038      c               anpars = anoises x 3 REAL*4 array of noise parameters (see
0039      c                         subroutine READ_NOISE).  This matrix applies to
0040      c                         ACTUAL SAMPLE GENERATION.
0041      c               acorrel = num_ants x anoises REAL*8 matrix of noise source
0042      c                         multipliers for each antenna element.  This
0043      c                         matrix applies to ACTUAL SAMPLE GENERATION.
0044      c               anoises = INTEGER*4 number of noise sources used in ACTUAL
0045      c                         SAMPLE GENERATION.
0046      c               sig_vec = num_waves-element COMPLEX*16 vector of "signal-in-
0047      c                         space" values (complex amplitudes and phases
0048      c                         of signal sources, noise-free, expressed in
0049      c                         "rectangular" coordinates).
0050      c               clean = num_ants-element COMPLEX*16 observation vector, before
0051      c                         corruption by noise.
0052      c               obs = computed observation vector.  num_ants elements.
0053      c                         COMPLEX*16.  Includes noise as determined by
0054      c                         the value of select, below.
0055      c               noise = computed noise vector.  num_ants elements.  COMPLEX*16.
0056      c                         The entries in this vector may be those
0057      c                         actually added to the noise-free observation
```

```
                                                    18-Apr-1988 14:25:31     VA>
                                                    19-Dec-1987 09:16:11     OBS

0058    c                               to produce "obs", or they may be produced by a
0059    c                               separate noise model, as determined by
0060    c                               "select", below.
0061    c                   select = indicator to choose what values of noise are placed in
0062    c                               the noise vector.  INTEGER*4.  Selects
0063    c                               "actual" noise values if = 0, selects values
0064    c                               from a different ("estimation") noise model if
0065    c                               > 0.  Note that the parameters "andefs",
0066    c                               "anpars", "acorrel", and "anoises" are not
0067    c                               referenced unless select > 0.
0068    c
0069    c-------------------------------------------------------------------------------
0070    c
0071            implicit none
0072            external dc_matvmpy, dc_vecsum, make_noise, dc_veccopy
0073            external open_va, free_va, exit_va
0074            include '6000$RT:[CHUCK.RESEARCH.FORTDIR]VATYPES.TXT'
0086    c
0087            integer*4 num_ants, num_waves, noises, anoises, select
0088            integer*4 ndefs( noises, 1 ), andefs( anoises, 1 )
0089            real*4 npars( noises, 1 ), anpars( anoises, 1 )
0090            real*8 correl( num_ants, noises ), acorrel( num_ants, anoises )
0091            complex*16 sig_mat( num_ants, num_waves ), sig_vec( num_waves )
0092            complex*16 clean( num_ants ), obs( num_ants ), noise( num_ants )
0093    c
0094            integer*4 noisevec_add, va_status, linear( 1 )
0095            character*80 va_msg
0096    c
0097            linear(1) = num_ants
0098            call open_va( noisevec_add, linear, 1, dc_type, va_status, va_msg )
0099            call exit_va( va_status, va_msg )
0100    c
0101            call dc_matvmpy( sig_mat, num_ants, num_waves, sig_vec, clean )
0102            call make_noise( noise, num_ants, ndefs, npars, correl, noises )
0103    c
0104            if (select .gt. 0) then
0105              call make_noise( %val( noisevec_add ), num_ants,
0106            1                                 andefs, anpars, acorrel, anoises )
0107            else
0108              call dc_veccopy( noise, num_ants, %val( noisevec_add ) )
0109            end if
0110    c
0111            call dc_vecsum( clean, num_ants, %val( noisevec_add ), obs )
0112    c
0113            call free_va( noisevec_add, va_status, va_msg )
0114            call exit_va( va_status, va_msg )
0115    c
0116            return
0117            end
```

```
                                                    18-Apr-1988 14:26:03    VA
                                                    14-Apr-1988 09:32:39    PI

0001            character*1 function
0002            1       pick_music_peak( apars, num_ants, wavelen, eigvec,
0003            2                                 select, lunit, mi_begp, mi_delp,
0004            3                                         fskip, num_this, iskip )
0005    c
0006    c-------------------------------------------------------------------------------
0007    c
0008    c               This function records peaks in MUSIC "power" spectra according
0009    c       to the user response to the add / delete / replot prompt.  During a
0010    c       particular function call, peaks are recorded for num_this values of p,
0011    c       and num_this may be less than or equal to the total number of p values
0012    c       used in the problem.
0013    c
0014    c       Parameters:
0015    c       -----------
0016    c               apars = matrix whose first column is the vector of ranges (from
0017    c                       the origin) of array antenna elements and whose
0018    c                       second column is the vector of angles (from the
0019    c                       "x" axis) of array antenna elements (in
0020    c                       RADIANS).  num_ants x 2.  REAL*8.
0021    c                     Note:  the position of the ith antenna element is
0022    c                     apars(i,1) * cos( apars(i,2) ) (x position),
0023    c                     apars(i,1) * sin( apars(i,2) ) (y position).
0024    c               num_ants = number of elements in antenna array.  INTEGER*4.
0025    c               wavelen = wavelength of the incident waves.  REAL*8.  (Note
0026    c                       that the units of wavelength must be the same
0027    c                       as those in which the array_range(*) are
0028    c                      expressed.)
0029    c               eigvec:  num_ants x num_ants COMPLEX*16 matrix whose FIRST
0030    c                       select columns are the applicable "noise"
0031    c                       generalized eigenvectors.
0032    c               select:  INTEGER*4 number of "noise" eigenvectors (columns of
0033    c                       eigvec) to be used in the computation.
0034    c               lunit:  INTEGER*4 Logical Unit Number of the file in which to
0035    c                       record the estimated angles of arrival (the
0036    c                       locations of the detected spectrum peaks).
0037    c                       This must be an UNFORMATTED, FIXED record
0038    c                       length, 2 quadwords/record, RELATIVE
0039    c                       organization, DIRECT access file.  The first
0040    c                       record holds the number of spectrum peaks
0041    c                       detected (updated by this routine).
0042    c               mi_begp:  REAL*4 minimum value of the power (p) to which to
0043    c                       raise the terms used in accumulating a DOA
0044    c                       spectrum sample (p = 2 for "ordinary" MUSIC).
0045    c               mi_delp:  REAL*4 minimum increment for p.
0046    c               fskip:  INTEGER*4 number of p values to skip before the first
0047    c                       p value for which to record the peaks.
0048    c               num_this:  INTEGER*4 number of p values for which to record
0049    c                       peaks during the current function call.
0050    c               iskip:  INTEGER*4 number of p values to skip between values for
0051    c                       which to record peaks.
0052    c
0053    c-------------------------------------------------------------------------------
0054    c
0055            implicit none
0056            external cap, spect_peak, readdoa, bulge_doa, crunch_doa, bracket_doa
0057            include '(real_constants)'
```

```
PICK_MUSIC_PEAK                                          18-Apr-1988 14:26:03    VA
                                                         14-Apr-1988 09:32:39    PI

0108    c
0109            real*8 spect_peak, readdoa
0110            character*1 cap
0111    c
0112            integer*4 num_ants, select, lunit, fskip, num_this, iskip
0113            real*4 mi_begp, mi_delp
0114            real*8 apars( num_ants, 1 ), wavelen
0115            complex*16 eigvec( num_ants, num_ants )
0116    c
0117            logical first
0118            real*4 p
0119            integer*4 num_picked, p_num, rec_num, i, lleft
0120            real*8 lower/ -pi8 /, upper/ pi8 /, tol/ .01 /, doa
0121            real*8 down/ 3.0d0 /, pwr, langl, uangl, perf
0122            character*8 answer
0123    c
0124            save lower, upper, tol, down
0125    c
0126            pick_music_peak = ' '
0127            do while ( ( pick_music_peak .ne. 'X' )
0128           1                                .and. ( pick_music_peak .ne. 'R' )
0129           2                                .and. ( pick_music_peak .ne. 'C' ) )
0130              read (unit=lunit,rec=1) num_picked
0131              type *
0132              if ( num_picked .gt. 0 ) then
0133                type 90070, ' Use what # of db down to compute sharpness (',
0134           1                                                        down, 'db)?  '
0135                accept 90010, answer
0136                read (unit=answer,fmt=*,err=11000) down
0137    11000       continue
0138    c
0139                type *, 'DOA spectrum peaks currently specified at the following ',
0140           1                                                    'angles (in degrees):'
0141                type 90060, down
0142                lleft = 19
0143                first = .true.
0144                rec_num = 2
0145    c
0146                do p_num = 1, num_this
0147                  p = mi_begp + ( fskip + ( p_num - 1 ) * ( iskip + 1 ) ) * mi_delp
0148    c
0149                  do i = 1, num_picked
0150                    if ( lleft .le. 0 ) then
0151                      accept 90010, answer
0152                      type 90060, down
0153                      lleft = 20
0154                      first = .true.
0155                    else
0156                      lleft = lleft - 1
0157                      first = .false.
0158                    end if
0159    c
0160                    doa = readdoa( lunit, rec_num )
0161                    call bracket_doa( doa, apars, num_ants, wavelen, eigvec,
0162           1                          select, p, down, pwr, langl, uangl, perf )
0163                    if ( first .or. ( i .eq. 1 ) ) then
0164                      type 90020, p, doa * deg_p_rad8, pwr, langl * deg_p_rad8,
```

```
0165            1                                       uangl * deg_p_rad8, perf
0166                      else
0167                        type 90050, doa * deg_p_rad8, pwr, langl * deg_p_rad8,
0168            1                                       uangl * deg_p_rad8, perf
0169                      end if
0170                    end do
0171                  end do
0172    c
0173                else
0174                  type *, 'No DOA spectrum peaks currently specified.'
0175                end if
0176    c
0177                type *
0178                if ( num_picked .eq. 0 ) then
0179                  type 90000, ' Add a peak set, replot, continue '
0180            1                          // 'with procedure, or give up (a/r/c/x)?  '
0181                  accept 90010, pick_music_peak
0182                  pick_music_peak = cap( pick_music_peak )
0183                  if ( pick_music_peak .eq. 'D' ) pick_music_peak = ' '
0184                else
0185                  type *, 'Add or delete a peak set, replot, continue ',
0186            1                                         'with procedure,'
0187                  type 90000, '                                        '
0188            1                                       // 'or give up (a/d/r/c/x)?  '
0189                  accept 90010, pick_music_peak
0190                  pick_music_peak = cap( pick_music_peak )
0191                end if
0192    c
0193                type *
0194                if      ( pick_music_peak .eq. 'A' ) then
0195                  type 90040, ' Lower end of peak search interval (degrees) (',
0196            1                                       lower * deg_p_rad8, ')?  '
0197                  accept 90010, answer
0198                  read (unit=answer,fmt=*,err=10000) lower
0199                    lower = lower * rad_p_deg8
0200    10000         continue
0201    c
0202                  type 90040, ' Upper end of peak search interval (degrees) (',
0203            1                                       upper * deg_p_rad8, ')?  '
0204                  accept 90010, answer
0205                  read (unit=answer,fmt=*,err=10010) upper
0206                    upper = upper * rad_p_deg8
0207    10010         continue
0208    c
0209                  type 90040, ' Locate peak to within how many degrees '
0210            1                          // '("tolerance") (', tol * deg_p_rad8, ')?  '
0211                  accept 90010, answer
0212                  read (unit=answer,fmt=*,err=10020) tol
0213                    tol = tol * rad_p_deg8
0214    10020         continue
0215    c
0216                  num_picked = num_picked + 1
0217                  doa = spect_peak( upper, lower, tol, mi_begp + fskip * mi_delp,
0218            1                           apars, num_ants, wavelen, eigvec, select )
0219                  rec_num = 2
0220                  do while ( rec_num .le. num_picked )
0221                    if ( doa .lt. readdoa( lunit, rec_num ) ) then
```

```
0222                    rec_num = rec_num - 1
0223                    go to 10040
0224                  end if
0225                end do
0226    10040       continue
0227                call bulge_doa( lunit, rec_num - 1, num_this )
0228                write (unit=lunit,rec=rec_num) doa
0229                rec_num = rec_num + num_picked
0230    c
0231                do i = 2, num_this
0232                  write (unit=lunit,rec=rec_num)
0233           1        spect_peak( upper, lower, tol, mi_begp +
0234           2                ( fskip + ( i - 1 ) * ( iskip + 1 ) ) * mi_delp,
0235           3                        apars, num_ants, wavelen, eigvec, select )
0236                  rec_num = rec_num + num_picked
0237                end do
0238    c
0239              else if ( pick_music_peak .eq. 'D' ) then
0240                type 90000, ' Delete which peak (integer) (0 to cancel ' //
0241           1                                                    'deletion)?  '
0242                accept 90010, answer
0243                read (unit=answer,fmt=*,err=10030) rec_num
0244                if ( rec_num .gt. 0 ) call crunch_doa( lunit, rec_num, num_this )
0245    10030       continue
0246              end if
0247            end do
0248    c
0249            return
0250    c
0251    90000   format( $a )
0252    90020   format( 1x, 0pf5.2, 4x, 0pf9.4, 1x, 1pg11.4,
0253           1                                2( 1x, 0pf9.4 ), 1x, 1pg11.4 )
0254    90010   format( a8 )
0255    90030   format( a, f5.2 )
0256    90040   format( $a, 1pg11.4, a )
0257    90050   format( 10x, 0pf9.4, 1x, 1pg11.4, 2( 1x, 0pf9.4 ), 1x, 1pg11.4 )
0258    90060   format( '            Arrival      Peak     ---', 0pf5.2,
0259           1                                              'db Down --'
0260           2       / '   p         Angle      Height     Lower     Upper     ',
0261           3                                              'Sharpness'
0262           4       / ' -----    --------- ----------- --------- --------- ',
0263           5                                              '-----------' )
0264    90070   format( $a, 0pf7.2, a )
0265    c
0266            end
```

```
                                                    18-Apr-1988 14:27:18    VA
                                                    26-Feb-1988 12:02:38    RE
0001            subroutine read_antennae( ant_pars, num_ants, old_ants, def_exp )
0002    c
0003    c-------------------------------------------------------------------------------
0004    c
0005    c               This subroutine sets a group of antenna array element position
0006    c       values, allowing the operator to change the positions if desired.
0007    c       Default values are determined as follows:  If old_ants <> num_ants or
0008    c       if def_exp < 0, "standard" defaults are assigned; otherwise default
0009    c       values are assumed to be present in array ant_pars on entry.
0010    c               If necessary, the routine converts operator entries to polar
0011    c       coordinates.
0012    c
0013    c       Parameters:
0014    c       -----------
0015    c               ant_pars:  num_ants x 2 REAL*8 output matrix whose first column
0016    c                       is the vector of antenna array element ranges from the
0017    c                       origin, and whose second column is the vector of
0018    c                       antenna array element angles from the x axis.
0019    c               num_ants:  Number of entries in the ant_range and ant_angle
0020    c                       vectors.  Also the number of element positions read in.
0021    c                       INTEGER*4, input.
0022    c               old_ants:  Number of antenna elements in the experiment from
0023    c                       which default values (if any) were obtained.
0024    c                       INTEGER*4, input.
0025    c               def_exp:  Number of the experiment from which default values
0026    c                       (if any) were obtained.  INTEGER*4, input.
0027    c
0028    c-------------------------------------------------------------------------------
0029    c
0030            implicit none
0031            external ck_polar, cap, nonblank
0032            intrinsic dreal, dimag, dcmplx
0033            include '(real_constants)'
0034    c
0035            logical nonblank
0036            integer*4 ck_polar
0037            character*1 cap
0038    c
0039            integer*4 num_ants, old_ants, def_exp
0040            real*8 ant_pars( num_ants, 1 )
0041    c
0042            integer*4 i, j, ant_num, status
0043            real*8 p1, p2
0044            complex*16 position
0045            character*15 instr
0046    c
0047            type *
0048            if ( ( old_ants .eq. num_ants ) .and. ( def_exp .ge. 0 ) ) then
0049              type 90000, ' Use old antennae positions ([Y]/N)?  '
0050            else
0051              p2 = pi8t2 / num_ants
0052              do i = 1, num_ants
0053                ant_pars(i,1) = 1.0d0
0054                ant_pars(i,2) = ( i - 1 ) * p2
0055              end do
0056              type 90000, ' Use default antennae positions ([Y]/N)?  '
0057            end if
```

READ_ANTENNAE 18-Apr-1988 14:27:18 VA
26-Feb-1988 12:02:38 RE

```
0108    c
0109    20000   continue
0110            accept 90010, instr
0111            if ( cap( instr( 1 : 1 ) ) .ne. 'N' ) go to 40000
0112    c
0113            i = 1
0114    30000   continue
0115              type 90020, ' Antenna number (', i, ')?  '
0116              accept 90010, instr
0117              if ( nonblank( instr ) ) then
0118                read (unit=instr,fmt=*,err=30000) ant_num
0119                if ( ( ant_num .lt. 1 ) .or. ( ant_num .gt. num_ants ) ) then
0120                  type *, char( 7 ), 'Antenna number out of range.'
0121                  go to 30000
0122                end if
0123              else
0124                ant_num = i
0125              end if
0126    c
0127    30010   continue
0128              type 90030, ' Range or x (', ant_pars(i,1), ')?  '
0129              accept 90010, instr
0130              if ( nonblank( instr ) ) then
0131                read (unit=instr,fmt=*,err=30010) p1
0132              else
0133                p1 = ant_pars(i,1)
0134              end if
0135    c
0136    30020   continue
0137              p2 = ant_pars(i,2) * deg_p_rad8
0138              type 90030, ' Angle (DEGREES) or y (', p2, ')?  '
0139              accept 90010, instr
0140              if ( nonblank( instr ) ) then
0141                read (unit=instr,fmt=*,err=30020) p2
0142              end if
0143    c
0144              type 90000, ' Above in polar or rectangular coordinates ([P]/R)?  '
0145              accept 90020, instr
0146              if ( cap( instr( 1 : 1 ) ) .eq. 'R' ) then
0147                status = ck_polar( dcmplx( p1, p2 ), position )
0148                if (status .ne. 0) then
0149                  type *, char( 7 ), 'Overflow converting to polar coordinates.'
0150                  go to 30000
0151                end if
0152                ant_pars(ant_num,1) = dreal( position )
0153                ant_pars(ant_num,2) = dimag( position )
0154              else
0155                ant_pars(ant_num,1) = p1
0156                ant_pars(ant_num,2) = p2 * rad_p_deg8
0157              end if
0158    c
0159              if ( ant_num .eq. i ) i = i + 1
0160            if ( i .le. num_ants ) go to 30000
0161    c
0162            type *
0163            type 90000, ' Entered antenna positions OK ([Y]/N)?  '
0164            go to 20000
```

```
READ_ANTENNAE                                                  18-Apr-1988 14:27:18    VA
                                                               26-Feb-1988 12:02:38    RE

0165     c
0166     40000    continue
0167              return
0168     c
0169     90000    format( $a )
0170     90010    format( a15 )
0171     90020    format( $ a, i5, a )
0172     90030    format( $ a, 1pg15.7, a )
0173     c
0174              end


PROGRAM SECTIONS

     Name                                 Bytes   Attributes

  0 $CODE                                 1061   PIC CON REL LCL   SHR   EXE   RD NOWRT LC
  1 $PDATA                                 321   PIC CON REL LCL   SHR NOEXE   RD NOWRT LO
  2 $LOCAL                                 264   PIC CON REL LCL NOSHR NOEXE   RD   WRT QU

     Total Space Allocated                1646


ENTRY POINTS

     Address  Type  Name

   0-00000000       READ_ANTENNAE


VARIABLES

     Address  Type  Name              Address  Type  Name              Address  Type  Name

   2-00000030  I*4  ANT_NUM        AP-00000010@ I*4  DEF_EXP            **      I*4  I
      **       I*4  J              AP-00000008@ I*4  NUM_ANTS       AP-0000000C@ I*4  OLD_A
   2-00000018  R*8  P2              2-00000020  C*16 POSITION           **      I*4  STATU


ARRAYS

     Address  Type  Name                Bytes  Dimensions

 AP-00000004@ R*8  ANT_PARS              **  (*, 1)


LABELS

     Address   Label       Address   Label       Address   Label       Address   Label

   0-000000CC  20000     0-00000118  30000     0-000001E4  30010     0-0000026C  30020
   1-0000012F  90010'    1-00000132  90020'    1-00000138  90030'
```

```
                                                    18-Apr-1988 14:27:36     V
                                                    27-Feb-1988 10:21:08     R
0001            subroutine read_lp( def_exp, begp, endp, delp, eps, norm, itlim )
0002    c
0003    c-------------------------------------------------------------------------------
0004    c
0005    c               This subroutine sets parameters for Lp estimation control, and
0006    c       allows the operator to change the values if desired.  if def_exp >= 0,
0007    c       the values of begp, endp, delp, eps, norm, and itlim on entry are
0008    c       assumed to have been obtained from a previous experiment, otherwise
0009    c       are assumed to be "standard" default values.
0010    c
0011    c       Parameters:
0012    c       -----------
0013    c               def_exp:  Number of the experiment (if any) from which defaults
0014    c                         were obtained. INTEGER*4, input.
0015    c               begp:  REAL*4 starting power of residual magnitudes to be
0016    c                         minimized (Lp estimation).
0017    c               endp:  REAL*4 ending power of residual magnitudes to be
0018    c                         minimized (Lp estimation).
0019    c               delp:  REAL*4 increment for power of residual magnitudes to be
0020    c                         minimized.
0021    c               eps:  REAL*8 minimum residual magnitude to use.
0022    c               norm:  REAL*8 value to which to normalize IRLS weights (0.0d0
0023    c                         for non-normalized IRLS).
0024    c               itlim:  INTEGER*4 maximum number of IRLS iterations to perform.
0025    c
0026    c-------------------------------------------------------------------------------
0027    c
0028            implicit none
0029            external cap, nonblank
0030    c
0031            logical nonblank
0032            character*1 cap
0033    c
0034            integer*4 def_exp, itlim
0035            real*4 begp, endp, delp
0036            real*8 eps, norm
0037    c
0038            integer*4 count
0039            real*8 temp
0040            character*15 instr
0041    c
0042            type *
0043            count = ( endp - begp ) / delp + 1
0044            if ( def_exp .ge. 0 ) then
0045              type 90040, 'Use old Lp estimation parameters (',
0046            1                                   count, ' iterations) ([Y]/N)?  '
0047            else
0048              type 90040, 'Use default Lp estimation parameters (',
0049            1                                   count, ' iterations) ([Y]/N)?  '
0050            end if
0051    c
0052    20000   continue
0053            accept 90010, instr
0054            if ( cap( instr( 1 : 1 ) ) .ne. 'N' ) go to 40000
0055    c
0056    30000   continue
0057              type 90050, ' Lp estimates starting at what p (>= 1) (', begp, ')?  '
```

```
READ_LP                                                    18-Apr-1988 14:27:36    VA>
                                                           27-Feb-1988 10:21:09    RE<

0058              accept 90010, instr
0059              if ( nonblank( instr ) ) then
0060                read (unit=instr,fmt=*,err=30000) temp
0061                if ( ( temp .lt. 1.0 ) .or. ( temp .gt. 3.0 ) ) then
0062                  type *, char( 7 ), 'Limits on p are 1 <= p <= 3.'
0063                  go to 30000
0064                end if
0065                begp = temp
0066              end if
0067      c
0068              type 90050, '                  ending at what p ((= 3) (', endp, ')?  '
0069              accept 90010, instr
0070              if ( nonblank( instr ) ) then
0071                read (unit=instr,fmt=*,err=30000) temp
0072                if ( ( temp .lt. begp ) .or. ( temp .gt. 3.0 ) ) then
0073                  type *, char( 7 ), 'Ending p must be >= beginning and <= 3.'
0074                  go to 30000
0075                end if
0076                endp = temp
0077              end if
0078      c
0079              if ( endp .gt. begp ) then
0080                type 90050, '          incrementing by what p ( > 0) (',
0081             1                                                    delp, ')?  '
0082                accept 90010, instr
0083                if ( nonblank( instr ) ) then
0084                  read (unit=instr,fmt=*,err=30000) temp
0085                  if ( temp .le. 0.0 ) then
0086                    type *, char( 7 ), 'Increment for p must be > 0.'
0087                    go to 30000
0088                  end if
0089                  delp = temp
0090                end if
0091              else
0092                delp = 3.0
0093              end if
0094      c
0095      30010   continue
0096              type 90020, ' Minimum residual magnitude to use (', eps, ')?  '
0097              accept 90010, instr
0098              if ( nonblank( instr ) ) then
0099                read (unit=instr,fmt=*,err=30010) temp
0100                if ( temp .le. 0.0d0 ) then
0101                  type *, char( 7 ), 'Value for epsilon must be greater than 0.'
0102                  go to 30010
0103                end if
0104                eps = temp
0105              end if
0106      c
0107      30020   continue
0108              type 90030, ' Normalize weight SQUARE ROOTS to what value (', norm,
0109             1                                                          ')'
0110              type 90000, '                                          (0 for '
0111             1                                                 // 'unnormalized)?  '
0112              accept 90010, instr
0113              if ( nonblank( instr ) ) then
0114                read (unit=instr,fmt=*,err=30020) temp
```

```
0115                if ( temp .lt. 0.0d0 ) then
0116                  type *, char( 7 ), 'Normalization value must be >= 0.'
0117                  go to 30020
0118                end if
0119                norm = temp
0120              end if
0121        c
0122        30030 continue
0123              type 90040, 'Maximum number of IRLS iterations to permit (',
0124             1                                                    itlim, ')?  '
0125              accept 90010, instr
0126              if ( nonblank( instr ) ) then
0127                read (unit=instr,fmt=*,err=30030) count      !  count scratch.
0128                if ( count .lt. 1 ) then
0129                  type *, char( 7 ), 'Iteration limit must be >= 1.'
0130                  go to 30030
0131                end if
0132                itlim = count
0133              end if
0134        c
0135              type *
0136              count = ( endp - begp ) / delp + 1
0137              type 90040, 'Entered parameters OK (', count,
0138             1                                    ' iterations) ([Y]/N)?  '
0139              go to 20000
0140        c
0141        40000 continue
0142              return
0143        c
0144        90000 format( $a )
0145        90010 format( a80 )
0146        90020 format( $ a, 1pg15.7, a )
0147        90030 format( a, 1pg15.7, a )
0148        90040 format( $, 1x, a, i4, a )
0149        90050 format( $ a, f5.2, a )
0150        c
0151              end
```

```
                                                    18-Apr-1988 14:28:11     V
                                                    27-Feb-1988 10:18:28     R

0001            subroutine read_mmisc( def_exp, carrlen, smp_int, tol )
0002    c
0003    c-------------------------------------------------------------------------------
0004    c
0005    c               This subroutine sets miscellaneous parameters required by the
0006    c       MUSIC / Lp algorithms, allowing the operator to change the values if
0007    c       desired.  If def_exp is >= 0, defaults from a previous experiment are
0008    c       assumed to be present in the values carrlen, smp_int, and tol on entry.
0009    c       If def_exp is < 0, "standard" defaults are assumed to be present in the
0010    c       values.  Note that the value of def_exp serves only to select what
0011    c       prompt is used, and that defaults are ASSUMED to be supplied regardless
0012    c       of the value of def_exp.
0013    c
0014    c       Parameters:
0015    c       -----------
0016    c               def_exp:  Number of the experiment (if any) from which defaults
0017    c                         were obtained.  INTEGER*4, input.
0018    c               carrlen:  REAL*8 carrier wavelength of incident waves.  Units
0019    c                         MUST be the same as those in which antenna
0020    c                         ranges are measured.
0021    c               smp_int:  REAL*8 sample interval, in carrier wavelengths.
0022    c               tol:  REAL*8 "tolerance"; value to be taken for zero.
0023    c
0024    c-------------------------------------------------------------------------------
0025    c
0026            implicit none
0027            external cap, nonblank
0028    c
0029            logical nonblank
0030            character*1 cap
0031    c
0032            integer*4 def_exp
0033            real*8 carrlen, smp_int, tol
0034    c
0035            character*1  nsel
0036            character*90 instr
0037    c
0038            type *
0039            if ( def_exp .ge. 0 ) then
0040              type 90000, ' Use old wavelength, sample interval, and tolerance '
0041            1                                                    // '([Y]/N)?  '
0042            else
0043              type 90000, ' Use default wavelength, sample interval, and '
0044            1                                          // 'tolerance ([Y]/N)?  '
0045            end if
0046    c
0047    20000   continue
0048            accept 90010, instr
0049            if ( cap( instr( 1 : 1 ) ) .ne. 'N' ) go to 40000
0050    c
0051    30000   continue
0052              type 90020, ' Carrier wavelength (', carrlen, ')?  '
0053              accept 90010, instr
0054              if ( nonblank( instr ) ) read (unit=instr,fmt=*,err=30000) carrlen
0055    c
0056    30010   continue
0057              type 90020, ' Sample interval (', smp_int,
```

```
READ_MMISC                                                    18-Apr-1988 14:28:11    VAX F
                                                              27-Feb-1988 10:13:28    READ_

0058            1                                          ') (Carrier Wavelengths)?  '
0059              accept 90010, instr
0060              if ( nonblank( instr ) ) read (unit=instr,fmt=*,err=30010) smp_int
0061    c
0062    30020   continue
0063              type 90020, ' Tolerance (value to take for zero) (', tol, ')?  '
0064              accept 90010, instr
0065              if ( nonblank( instr ) ) read (unit=instr,fmt=*,err=30020) tol
0066    c
0067            type *
0068            type 90000, ' Entered parameters OK ([Y]/N)?  '
0069            go to 20000
0070    c
0071    40000   continue
0072            return
0073    c
0074    90000   format( $a )
0075    90010   format( a80 )
0076    90020   format( $ a, 1pg15.7, a )
0077    c
0078            end


PROGRAM SECTIONS

    Name                                Bytes   Attributes

  0 $CODE                                609   PIC CON REL LCL    SHR    EXE    RD NOWRT LONG
  1 $PDATA                               282   PIC CON REL LCL    SHR NOEXE    RD NOWRT LONG
  2 $LOCAL                               188   PIC CON REL LCL NOSHR NOEXE    RD    WRT LONG

    Total Space Allocated               1079


ENTRY POINTS

    Address  Type  Name

  0-00000000       READ_MMISC


VARIABLES

    Address  Type  Name             Address  Type  Name             Address  Type  Name

 AP-00000008@ R*8  CARRLEN       AP-00000004@ I*4  DEF_EXP       2-00000000  CHAR INSTR
 AP-0000000C@ R*8  SMP_INT       AP-00000010@ R*8  TOL
```

```
                                                    18-Apr-1988 14:28:25    VA
                                                    27-Feb-1988 10:00:08    RE

0001            subroutine read_noise( def_exp, ants, old_ants, noises, old_noises,
0002            1                      defs, pars, correl, prompt_sel, old_which )
0003    c
0004    c-------------------------------------------------------------------------------
0005    c
0006    c               This subroutine sets a group of noise characteristic values,
0007    c       permitting the operator to change the characteristics, if desired.
0008    c       The value prompt_sel determines whether "actual" (prompt_sel < 0) or
0009    c       "model" (prompt_sel >= 0) noise generators are being defined.
0010    c               Defaults are determined for "actual" noise generators as
0011    c       follows:  If def_exp > 0, noises = old_noises, and old_which > 0, then
0012    c       default values are assumed to be present in the arrays defs and pars.
0013    c       If, in addition, ants = old_ants, then default values are assumed to be
0014    c       present in array correl on entry as well.  If default values are not
0015    c       assumed to be present in an array, then "standard" defaults are
0016    c       assigned.  For "model" noise generators, defaults are established in
0017    c       the same way except that old_which is ignored (since all experiments
0018    c       use "model" noise generators).
0019    c
0020    c       Parameters:
0021    c       -----------
0022    c               def_exp:  INTEGER*4 number of the experiment from which default
0023    c                                   values (if any) were obtained.
0024    c               ants:  INTEGER*4 number of elements in the antenna array.
0025    c               old_ants:  INTEGER*4 number of antenna array elements in the
0026    c                                   experiment from which default values
0027    c                                   (if any) were obtained.
0028    c               noises:  INTEGER*4 number of noise generators to be used.
0029    c               old_noises:  INTEGER*4 number of noise generators used in the
0030    c                                   experiment from which default values
0031    c                                   (if any) were obtained.
0032    c               defs:  noises x 2 INTEGER*4 matrix whose first column is the
0033    c                                   vector of generator noise types (see
0034    c                                   below).
0035    c               pars:  noises x 3 REAL*4 array of noise parameters for each
0036    c                                   antenna (mean, std. dev., ave. rep.
0037    c                                   rate).
0038    c               correl:  ants x noises REAL*8 matrix of noise source
0039    c                                   multipliers for each antenna element
0040    c                                   (noise correlation matrix).
0041    c               prompt_sel:  INTEGER*4 value indicating whether the noise
0042    c                                   sources being defined are those for
0043    c                                   actual sample generation (prompt_sel
0044    c                                   < 0), those for the estimator noise
0045    c                                   model (prompt_sel > 0), or both (i.e.
0046    c                                   estimators use actual noise)
0047    c                                   (prompt_sel = 0).
0048    c               old_which:  INTEGER*4 value indicating whether actual
0049    c                                   (old_which = 0) or model (old_which >
0050    c                                   0) noise generators were used in the
0051    c                                   estimation routines of the experiment
0052    c                                   from which default values (if any) were
0053    c                                   obtained.
0054    c
0055    c       Noise types:
0056    c       ------------
0057    c               type = hhll (hex):  hh = high word (16 bits), ll = low word.
```

```
                                                                    18-Apr-1988 14:28:25     VA
                                                                    27-Feb-1988 10:00:08     RE
0058    c               ll = 1 => fixed noise (constant amplitude and phase);
0059    c                                 pars( i, 1 ) = real part of noise.
0060    c                                 pars( i, 2 ) = imaginary part of noise.
0061    c               ll = 2 => uniform noise; pars( i, 1 ) = mean (real & imag.).
0062    c                                  pars( i, 2 ) = width (real & imag.).
0063    c               ll = 3 => gaussian noise; pars( i, 1 ) as for uniform.
0064    c                                   pars( i, 2 ) = std. dev. (real & i.).
0065    c               ll = others => not used.
0066    c               hh = 0 => noise type determined by ll.
0067    c               hh = 1 => impulsive noise, type determined by ll;
0068    c                                 pars( i, 3 ) = average repetit. rate.
0069    c               hh = 2 => burst noise, type determined by ll;
0070    c                                 (applied to randomly selected anten.),
0071    c                                 (all multipliers = 0.0d0),
0072    c                                 pars( i, 3 ) = average repetit. rate.
0073    c
0074    c-------------------------------------------------------------------------------
0075    c
0076            implicit none
0077            intrinsic jishft, jiand, jior, dcmplx, dble
0078            intrinsic sngl, dreal, dimag, alog10, jmod
0079            external cap, nonblank, pr_ntype, rect, ck_polar
0080            include '(fortran_limits)'
0109            include '(real_constants)'
0160    c
0161            logical nonblank
0162            integer*4 ck_polar
0163            complex*16 rect
0164            character*1 cap
0165    c
0166            integer*4 def_exp, ants, old_ants, noises, old_noises
0167            integer*4 defs( noises, 1 ), prompt_sel, old_which
0168            real*4 pars( noises, 1 )
0169            real*8 correl( ants, noises )
0170    c
0171            logical have_one
0172            integer*4 i, j, noise_num, it1, it2, ant_fnoise
0173            real*4 t1, t2
0174            complex*16 z1
0175            character*15 instr
0176    c
0177            type *
0178            if      ( prompt_sel .lt. 0 ) then
0179              type *,'              Noise sources to be used in ',
0180            1                                       'ACTUAL SAMPLE GENERATION'
0181            else if ( prompt_sel .eq. 0 ) then
0182              type *,'   Noise sources for BOTH actual sample ',
0183            1                                       'generation AND estimation noise model'
0184            else
0185              type *,'            Noise sources to be used in the ',
0186            1                                       'ESTIMATION NOISE MODEL'
0187            end if
0188            type *
0189    c
0190            if ( ( def_exp .lt. 0 )
0191            1       .or. ( noises .ne. old_noises )
0192            2       .or. ( ( prompt_sel .lt. 0 ) .and. ( old_which .le. 0 ) )
```

```
0193            3       ) then
0194              do i = 1, noises
0195                defs(i,1) = 3
0196                pars(i,1) = 0.0
0197                pars(i,2) = 0.03162278
0198                pars(i,3) = 0.0
0199              end do
0200              go to 10000
0201            else
0202              if ( ants .eq. old_ants ) then
0203                type 90000, ' Use old noise generators ([Y]/N)?  '
0204              else
0205    10000       continue
0206                do i = 1, ants
0207                  do j = 1, noises
0208                    if ( i .eq. j ) then
0209                      correl(i,j) = 1.0d0
0210                    else
0211                      correl(i,j) = 0.0d0
0212                    end if
0213                  end do
0214                end do
0215                type 90000, ' Use default noise generators ([Y]/N)?  '
0216              end if
0217            end if
0218    c
0219    20000   continue
0220            accept 90010, instr
0221            if ( cap( instr( 1 :_1 ) ) .ne. 'N' ) go to 40000
0222    c
0223            i = 1
0224    30000   continue
0225              type 90020, ' Noise source number (', i, ')?  '
0226              accept 90010, instr
0227              if ( nonblank( instr ) ) then
0228                read (unit=instr,fmt=*,err=30000) noise_num
0229                if ( ( noise_num .lt. 1 ) .or. ( noise_num .gt. noises ) ) then
0230                  type *, char( 7 ), 'Source number out of range.'
0231                  go to 30000
0232                end if
0233              else
0234                noise_num = i
0235              end if
0236    c
0237    30010   continue
0238              it1 = jishft( defs(i,1), -16 )
0239              if      ( it1 .eq. 0 ) then
0240                type 90000, ' Continuous, Intermittent, or Burst noise '
0241    1                                   // '(Continuous) (C, I, or B)?  '
0242              else if ( it1 .eq. 1 ) then
0243                type 90000, ' Continuous, Intermittent, or Burst noise '
0244    1                                   // '(Intermittent) (C, I, or B)?  '
0245              else
0246                type 90000, ' Continuous, Intermittent, or Burst noise '
0247    1                                   // '(Burst) (C, I, or B)?  '
0248              end if
0249              accept 90010, instr
```

```
READ_NOISE                                                    18-Apr-1988 14:28:25     VA)
                                                              27-Feb-1988 10:00:08     RE/

0250              instr( 1 : 1 ) = cap( instr( 1 : 1 ) )
0251              if ( nonblank( instr ) ) then
0252                if      ( instr( 1 : 1 ) .eq. 'C' ) then
0253                  it1 = 0
0254                else if ( instr( 1 : 1 ) .eq. 'I' ) then
0255                  it1 = 1
0256                else if ( instr( 1 : 1 ) .eq. 'B' ) then
0257                  it1 = 2
0258                else
0259                  type *, char( 7 ), 'Please enter C<CR>, c<CR>, I<CR>, ',
0260             1                               'i<CR>, B<CR>, b<CR>, or <CR>.'
0261                  go to 30010
0262                end if
0263              end if
0264     c
0265     30030    continue
0266              it2 = jiand( defs(i,1), 65535 )
0267              call pr_ntype( it2 )
0268              accept 90010, instr
0269              if ( nonblank( instr ) ) then
0270                it2 = index( 'FUG', cap( instr( 1 : 1 ) ) )
0271              end if
0272              if ( it2 .eq. 0 ) go to 30030
0273            defs(noise_num,1) = jior( jishft( it1, 16 ), it2 )
0274     c
0275            if ( it2 .eq. 1 ) then
0276              if ( ck_polar( dcmplx( dble( pars(i,1) ), dble( pars(i,2) ) ), z1 )
0277             1                .ne. 0 ) z1 = ( 1.0d0, 0.0d0 )
0278              t1 = sngl( dreal( z1 ) )
0279              if ( t1 .lt. minrecip4 ) t1 = 1.0
0280              t2 = sngl( dimag( z1 ) ) * deg_p_rad4
0281            else
0282              t1 = pars(i,1)
0283              if ( pars(i,2) .ge. minrecip4 ) then
0284                t2 = pars(i,2)
0285              else
0286                t2 = 1.0
0287              end if
0288            end if
0289     30040  continue
0290              if ( it2 .eq. 1 ) then
0291                type 90030, ' Noise magnitude (', 20.0 * alog10( t1 ), ') (db)?  '
0292              else
0293                type 90030, ' Distribution mean (', t1, ')?  '
0294              end if
0295              accept 90010, instr
0296              if ( nonblank( instr ) ) then
0297                read (unit=instr,fmt=*,err=30040) pars(noise_num,1)
0298                if ( it2 .eq. 1 ) pars(noise_num,1) =
0299             1                                  10.0 ** ( pars(noise_num,1) / 20.0 )
0300              else
0301                pars(noise_num,1) = t1
0302              end if
0303     c
0304     30050    continue
0305              if ( it2 .eq. 1 ) then
0306                type 90030, ' Noise phase (', t2, ') (DEGREES)?  '
```

```
0307              else
0308                type 90030, ' Noise power (', 20.0 * alog10( t2 ), ') (db)?  '
0309              end if
0310              accept 90010, instr
0311              if ( nonblank( instr ) ) then
0312                read (unit=instr,fmt=*,err=30050) pars(noise_num,2)
0313                if ( it2 .ne. 1 ) pars(noise_num,2) =
0314             1                                  10.0 ** ( pars(noise_num,2) / 20.0 )
0315              else
0316                pars(noise_num,2) = t2
0317              end if
0318            if ( it2 .eq. 1 ) then
0319              z1 = rect( dcmplx( dble( pars(noise_num,1) ),
0320             1                   rad_p_deg8 * pars(noise_num,2) ) )
0321              pars(noise_num,1) = sngl( dreal( z1 ) )
0322              pars(noise_num,2) = sngl( dimag( z1 ) )
0323            end if
0324        c
0325        30060   continue
0326            if ( it1 .ne. 0 ) then
0327              type 90030,
0328             1        ' Average repetition interval(', pars(i,3), ') (samples)?  '
0329              accept 90010, instr
0330              if ( nonblank( instr ) ) then
0331                read (unit=instr,fmt=*,err=30060) pars(noise_num,3)
0332              else
0333                pars(noise_num,3) = pars(i,3)
0334              end if
0335            end if
0336        c
0337            if ( it1 .ne. 2 ) then
0338              have_one = .false.
0339        30070     continue
0340                if ( have_one ) then
0341                  type 90000, ' Apply this source at which antenna (<CR> for '
0342             1                                                    // 'next source)?  '
0343                else
0344                  ant_fnoise = jmod( noise_num, ants )
0345                  if ( ant_fnoise .eq. 0 ) ant_fnoise = ants
0346                  type 90020, ' Apply this source at which antenna (', ant_fnoise,
0347             1                                                          ')?  '
0348                end if
0349                accept 90010, instr
0350                if ( nonblank( instr ) ) then
0351                  read (unit=instr,fmt=*,err=30075) j
0352                  if ( ( j .lt. 1 ) .or. ( j .gt. ants ) ) go to 30075
0353                  if ( have_one ) then
0354                    type 90030, '                                    Multiplier (',
0355             1                                                   correl(j,i), ')?  '
0356                  else
0357                    type 90030, '                Multiplier(', correl(j,i), ')?  '
0358                  end if
0359                  accept 90010, instr
0360                  if ( nonblank( instr ) ) then
0361                    read (unit=instr,fmt=*,err=30075) correl(j,noise_num)
0362                  else
0363                    correl(j,noise_num) = correl(j,i)
```

```
0364                    end if
0365                    have_one = .true.
0366                    go to 30070
0367    30075           continue
0368                      type *, char( 7 ), 'Error on input.  Please re-enter ',
0369           1                                                    'from start.'
0370                      go to 30070
0371                  else if ( .not. have_one ) then
0372                    j = ant_fnoise
0373                    type 90030, '                Multiplier(', correl(j,i), ')?  '
0374                    accept 90010, instr
0375                    if ( nonblank( instr ) ) then
0376                      read (unit=instr,fmt=*,err=30075) correl(j,noise_num)
0377                    else
0378                      correl(j,noise_num) = correl(j,i)
0379                    end if
0380                    have_one = .true.
0381                    go to 30070
0382                  end if
0383            else
0384              do j = 1, ants
0385                correl(j,noise_num) = 0.0d0
0386              end do
0387            end if
0388    c
0389              if ( noise_num .eq. i ) i = i + 1
0390            if ( i .le. noises ) go to 30000
0391    c
0392            type *
0393            type 90000, ' Entered noise parameters OK ([Y]/N)?  '
0394            go to 20000
0395    c
0396    40000   continue
0397            return
0398    c
0399    90000   format( $a )
0400    90010   format( a15 )
0401    90020   format( $ a, i5, a )
0402    90030   format( $ a, 1pg15.7, a )
0403    c
0404            end
```

```
                                                        18-Apr-1988 14:28:25
                                                        27-Feb-1988 10:00:08

0001    c
0002            subroutine pr_ntype( ntype )
0003    c
0004    c*******************************************************************************
0005    c
0006    c                This subroutine prompts for noise distribution type.  The type
0007    c       of noise distribution included in the prompt (the default type) is
0008    c       selected by the INTEGER*4 value ntype (1 = fixed (constant magnitude
0009    c       and phase), 2 = uniform, 3 = gaussian)
0010    c
0011    c*******************************************************************************
0012    c
0013            integer*4 ntype
0014    c
0015            character*21 head/' Noise distribution ('/
0016            character*17 tail/') (F, U, or G)?  '/
0017    c
0018            if ( ntype .eq. 1 ) type 90000, head // 'Fixed' // tail
0019            if ( ntype .eq. 2 ) type 90000, head // 'Uniform' // tail
0020            if ( ntype .eq. 3 ) type 90000, head // 'Gaussian' // tail
0021    c
0022            return
0023    c
0024    90000   format ( $a )
0025    c
0026            end


PROGRAM SECTIONS

    Name                                Bytes   Attributes

  0 $CODE                                 226   PIC CON REL LCL   SHR   EXE   RD NOWRT
  1 $PDATA                                 15   PIC CON REL LCL   SHR NOEXE   RD NOWRT
  2 $LOCAL                                 48   PIC CON REL LCL NOSHR NOEXE   RD   WRT

    Total Space Allocated                 289


ENTRY POINTS

    Address  Type  Name

  0-00000000       PR_NTYPE


VARIABLES

    Address  Type  Name             Address  Type  Name             Address  Type  Nam

  2-00000000  CHAR HEAD          AP-00000004@ I*4  NTYPE          2-00000015  CHAR TAI
```

```
                                                        18-Apr-1988 14:28:55     VA
                                                        26-Feb-1988 12:03:36     RE

0001            subroutine read_waves( num_waves, old_waves, def_exp,
0002            1                                                   wave_pars, mtypes )
0003      c
0004      c-------------------------------------------------------------------------------
0005      c
0006      c               This subroutine sets a group of wave characteristic values,
0007      c       allowing the operator to change the values if desired.  Default values
0008      c       are determined as follows:  If old_waves <> num_waves or def_exp < 0,
0009      c       "standard" defaults are assigned, otherwise default values are assumed
0010      c       to be present in arrays wave_pars and mtypes on entry.
0011      c
0012      c       Parameters:
0013      c       -----------
0014      c               num_waves:  Number of rows in the output matrices.  Also the
0015      c                           number of wave parameter sets read in.
0016      c                           INTEGER*4, input.
0017      c               old_waves:  Number of incident waves in the experiment from
0018      c                           which default values (if any) were obtained.
0019      c                           INTEGER*4, input.
0020      c               def_exp:  Number of the experiment from which default values
0021      c                           (if any) were obtained.  INTEGER*4, input.
0022      c               wave_pars:  num_waves x 9 REAL*8 matrix whose first 8 columns
0023      c                           are as follows:
0024      c                           1) vector of strengths of the waves, expressed
0025      c                                   in db (referred to arbitrary unit
0026      c                                   amplitude).
0027      c                           2) vector of starting carrier phases of the
0028      c                                   arriving waves, RADIANS.
0029      c                           3) vector of wave angles of arrival, from the x
0030      c                                   axis, RADIANS.
0031      c                           4) vector of modulation cycle lengths, in
0032      c                                   number of carrier cycles per modulation
0033      c                                   cycle.
0034      c                           5) vector of modulation duty cycles (expressed
0035      c                                   as fractions of modulation cycle
0036      c                                   length, 0.0 < mod_duty(i) <= 1.0)
0037      c                                   (applicable only to ramp and
0038      c                                   rectangular modulation).
0039      c                           6) vector of "starting" modulation phases, in
0040      c                                   number of carrier cycles.
0041      c                           7) vector of amplitude modulation ratios
0042      c                                   (maximum / minimum).
0043      c                           8) vector of phase modulation total phase
0044      c                                   shifts (in RADIANS).
0045      c               mtypes:  num_waves x 2 INTEGER*4 matrix whose first column is
0046      c                           the vector of amplitude modulation types (see
0047      c                           below), and whose second column is the vector
0048      c                           of phase modulation types (see below).
0049      c
0050      c               Modulation types:  1 = sine, 2 = triangle, 3 = increasing ramp,
0051      c       4 = decreasing ramp, 5 = rectangular.
0052      c
0053      c-------------------------------------------------------------------------------
0054      c
0055            implicit none
0056            intrinsic index
0057            external cap, nonblank, pr_mtype
```

READ_WAVES 18-Apr-1988 14:28:55
26-Feb-1988 12:03:36

```
0058            include '(real_constants)'
0109    c
0110            logical nonblank
0111            character*1 cap
0112    c
0113            integer*4 num_waves, old_waves, def_exp
0114            integer*4 mtypes( num_waves, 1 )
0115            real*8 wave_pars( num_waves, 1 )
0116    c
0117            integer*4 i, j, wave_num, it1
0118            real*8 t1, t2, t3
0119            character*15 instr
0120    c
0121            type *
0122            if ( ( old_waves .eq. num_waves ) .and. ( def_exp .ge. 0 ) ) then
0123              type 90000, ' Use old wave characteristics ([Y]/N)?  '
0124            else
0125              t1 = 1.0d0  / num_waves
0126              t2 = 30.0d0 * t1
0127              t3 = pi8t2  * t1
0128              do i = 1, num_waves
0129                wave_pars(i,1) = i * t2
0130                wave_pars(i,2) = 0.0d0
0131                wave_pars(i,3) = ( i - 1 ) * t3
0132                wave_pars(i,4) = 30.0d0
0133                wave_pars(i,5) = t1
0134                wave_pars(i,6) = ( i - 1 ) * t2
0135                wave_pars(i,7) = 100.0d0
0136                wave_pars(i,8) = 0.0d0
0137                mtypes(i,1)    = 5
0138                mtypes(i,2)    = 5
0139              end do
0140              type 90000, ' Use default wave characteristics ([Y]/N)?  '
0141            end if
0142    c
0143    20000   continue
0144            accept 90010, instr
0145            if ( cap( instr( 1 : 1 ) ) .ne. 'N' ) go to 40000
0146    c
0147            i = 1
0148    30000   continue
0149              type 90020, ' Wave number (', i, ')?  '
0150              accept 90010, instr
0151              if ( nonblank( instr ) ) then
0152                read (unit=instr,fmt=*,err=30000) wave_num
0153                if ( ( wave_num .lt. 1 ) .or. ( wave_num .gt. num_waves ) ) then
0154                  type *, char( 7 ), 'Wave number out of range.'
0155                  go to 30000
0156                end if
0157              else
0158                wave_num = i
0159              end if
0160    c
0161    30010   continue
0162              type 90030, ' Wave strength (', wave_pars(i,1), ') (db)?  '
0163              accept 90010, instr
0164              if ( nonblank( instr ) ) then
```

```
READ_WAVES                                                    18-Apr-1988 14:28:55
                                                              26-Feb-1988 12:03:36

0165                read (unit=instr,fmt=*,err=30010) wave_pars(wave_num,1)
0166              else
0167                wave_pars(wave_num,1) = wave_pars(i,1)
0168              end if
0169        c
0170        30015 continue
0171              type 90030, ' Carrier start phase (', wave_pars(i,2) * deg_p_rad8,
0172             1                                                  ') (DEGREES)?  '
0173              accept 90010, instr
0174              if ( nonblank( instr ) ) then
0175                read (unit=instr,fmt=*,err=30015) wave_pars(wave_num,2)
0176                wave_pars(wave_num,2) = wave_pars(wave_num,2) * rad_p_deg8
0177              else
0178                wave_pars(wave_num,2) = wave_pars(i,2)
0179              end if
0180        c
0181        30020 continue
0182              type 90030, ' Angle of arrival (', wave_pars(i,3) * deg_p_rad8,
0183             1                                                  ') (DEGREES)?  '
0184              accept 90010, instr
0185              if ( nonblank( instr ) ) then
0186                read (unit=instr,fmt=*,err=30020) wave_pars(wave_num,3)
0187                wave_pars(wave_num,3) = wave_pars(wave_num,3) * rad_p_deg8
0188              else
0189                wave_pars(wave_num,3) = wave_pars(i,3)
0190              end if
0191        c
0192              t1 = wave_pars(i,6) / wave_pars(i,4)
0193        30030 continue
0194              type 90030, ' Modulation length (', wave_pars(i,4),
0195             1                                    ') (Carrier Cycles) (>=1)?  '
0196              accept 90010, instr
0197              if ( nonblank( instr ) ) then
0198                read (unit=instr,fmt=*,err=30030) t2
0199              else
0200                t2 = wave_pars(i,4)
0201              end if
0202              if ( t2 .lt. 1.0d0 ) go to 30030
0203              wave_pars(wave_num,4) = t2
0204        c
0205        30040 continue
0206              type 90030, ' Modulation starting phase (', t1 * 360.0d0,
0207             1                                                  ') (DEGREES)?  '
0208              accept 90010, instr
0209              if ( nonblank( instr ) ) then
0210                read (unit=instr,fmt=*,err=30040) wave_pars(wave_num,6)
0211                wave_pars(wave_num,6) =
0212             1        ( wave_pars(wave_num,6) / 360.0d0 ) * wave_pars(wave_num,4)
0213              else
0214                wave_pars(wave_num,6) = t1 * wave_pars(wave_num,4)
0215              end if
0216        c
0217        30050 continue
0218              type 90030, ' Modulation duty cycle (', wave_pars(i,5),
0219             1                                                  ') (>0, <=1)?  '
0220              accept 90010, instr
0221              if ( nonblank( instr ) ) then
```

```
READ_WAVES                                                    18-Apr-1988 14:28:
                                                              26-Feb-1988 12:03:

0222                read (unit=instr,fmt=*,err=30050) t1
0223              else
0224                t1 = wave_pars(i,5)
0225              end if
0226              if ( ( t1 .le. 0.0d0 ) .or. ( t1 .gt. 1.0d0 ) ) go to 30050
0227              wave_pars(wave_num,5) = t1
0228      c
0229      30060   continue
0230              call pr_mtype( 'Amplitude', mtypes(i,1) )
0231              accept 90010, instr
0232              if ( nonblank( instr ) ) then
0233                it1 = index( 'STIDR', cap( instr( 1 : 1 ) ) )
0234              else
0235                it1 = mtypes(i,1)
0236              end if
0237              if ( it1 .eq. 0 ) go to 30060
0238              mtypes(wave_num,1) = it1
0239      c
0240      30070   continue
0241              type 90030, ' Amplitude modulation ratio (', wave_pars(i,7),
0242             1                          ') (max / min)?  '
0243              accept 90010, instr
0244              if ( nonblank( instr ) ) then
0245                read (unit=instr,fmt=*,err=30070) wave_pars(wave_num,7)
0246              else
0247                wave_pars(wave_num,7) = wave_pars(i,7)
0248              end if
0249      c
0250      30080   continue
0251              call pr_mtype( 'Phase', mtypes(i,2) )
0252              accept 90010,- instr
0253              if ( nonblank( instr ) ) then
0254                it1 = index( 'STIDR', cap( instr( 1 : 1 ) ) )
0255              else
0256                it1 = mtypes(i,2)
0257              end if
0258              if ( it1 .eq. 0 ) go to 30080
0259              mtypes(wave_num,2) = it1
0260      c
0261      30090   continue
0262              type 90030, ' Phase modulation total shift (',
0263             1                          wave_pars(i,8) * deg_p_rad8, ') (DEGREES)?
0264              accept 90010, instr
0265              if ( nonblank( instr ) ) then
0266                read (unit=instr,fmt=*,err=30090) wave_pars(wave_num,8)
0267                wave_pars(wave_num,8) = wave_pars(wave_num,8) * rad_p_deg8
0268              else
0269                wave_pars(wave_num,8) = wave_pars(i,8)
0270              end if
0271      c
0272              if ( wave_num .eq. i ) i = i + 1
0273            if ( i .le. num_waves ) go to 30000
0274      c
0275            type *
0276            type 90000, ' Entered wave characteristics OK ([Y]/N)?  '
0277            go to 20000
0278      c
```

READ_WAVES 18-Apr-1988 14:28:5
26-Feb-1988 12:03:3

```
0279    c               Sort entered wave parameters into ascending DOA order.
0280    c
0281    40000   continue
0282            do i = 1, num_waves - 1
0283              wave_num = i
0284              do j = i + 1, num_waves
0285                if ( wave_pars(j,3) .lt. wave_pars(wave_num,3) ) wave_num = j
0286              end do
0287              if ( wave_num .ne. i ) then
0288                do j = 1, 8
0289                  t1 = wave_pars(i,j)
0290                  wave_pars(i,j) = wave_pars(wave_num,j)
0291                  wave_pars(wave_num,j) = t1
0292                end do
0293                do j = 1, 2
0294                  it1 = mtypes(i,j)
0295                  mtypes(i,j) = mtypes(wave_num,j)
0296                  mtypes(wave_num,j) = it1
0297                end do
0298              end if
0299            end do
0300    c
0301            return
0302    c
0303    90000   format( $a )
0304    90010   format( a15 )
0305    90020   format( $ a, i5, a )
0306    90030   format( $ a, 1pg15.7, a )
0307    c
0308            end
```

18-Apr-1988 14:28:55
26-Feb-1988 12:03:36

```
0001    c
0002            subroutine pr_mtype( kind, ptype )
0003    c
0004    c*******************************************************************************
0005    c
0006    c                This subroutine prompts for modulation type.  The CHARACTER*
0007    c       string kind is prepended to the prompt, and the type of modulation
0008    c       included in the prompt (the default type) is selected by the INTEGER
0009    c       value ptype (1 = sine, 2 = triangular, 3 = increasing ramp,
0010    c       4 = decreasing ramp, 5 = rectangular).
0011    c
0012    c*******************************************************************************
0013    c
0014            integer*4 ptype
0015            character*(*) kind
0016    c
0017            character*18 head/' modulation type ('/
0018            character*23 tail/') (S, T, I, D, or R)?  '/
0019    c
0020            if ( ptype .eq. 1 ) type 90000,
0021            1                       ' ' // kind // head // 'Sinusoidal' // tail
0022            if ( ptype .eq. 2 ) type 90000,
0023            1                       ' ' // kind // head // 'Triangular' // tail
0024            if ( ptype .eq. 3 ) type 90000,
0025            1                       ' ' // kind // head // 'Increasing Ramp' // tail
0026            if ( ptype .eq. 4 ) type 90000,
0027            1                       ' ' // kind // head // 'Decreasing Ramp' // tail
0028            if ( ptype .eq. 5 ) type 90000,
0029            1                       ' ' // kind // head // 'Rectangular' // tail
0030    c
0031            return
0032    c
0033    90000   format ( $a )
0034    c
0035            end
```

18-Apr-1988 14:30:50
11-Mar-1988 08:31:27

```
0001            subroutine sigmat_estm( apars, num_ants, doa, num_doas, wavelen,
0002            1                                                       sigmat )
0003      c
0004      c-------------------------------------------------------------------------------
0005      c
0006      c               This subroutine forms an estimate of the "signal matrix", whos
0007      c       columns are the "signal vectors" determined by subroutine SIGVAL, fror
0008      c       estimated angles of arrival.
0009      c
0010      c       Parameters:
0011      c       -----------
0012      c               apars = matrix whose first column is the vector of ranges (fr
0013      c                       the origin) of the antenna array elements, and
0014      c                       whose second column is the vector of angles
0015      c                       (from the "x" axis) of the antenna array
0016      c                       elements (RADIANS).  num_ants x 2.  REAL*8.
0017      c                   Note that the position of the ith antenna element is
0018      c                       apars(i,1) * cos( apars(i,2) ) (x position),
0019      c                       apars(i,1) * sin( apars(i,2) ) (y position).
0020      c               num_ants = number of antenna elements in the array.  INTEGER*4
0021      c               doa = vector of estimated angles of arrival of the incoming
0022      c                       waves.  num_doas elements.  REAL*8.
0023      c               num_doas = number of estimated angles of arrival.  INTEGER*4.
0024      c               wavelen = wavelength of the incident waves.  REAL*8.  (Note
0025      c                       that the units of wavelength are assumed to be
0026      c                       those in which the array ranges are expressed)
0027      c               sigmat = computed signal matrix.  num_ants rows, num_doas
0028      c                       columns.  COMPLEX*16.
0029      c
0030      c-------------------------------------------------------------------------------
0031      c
0032            implicit none
0033      c
0034            external sigval, dc_colfill
0035            external open_va, free_va, exit_va
0036      c
0037            integer*4 num_ants, num_doas
0038            real*8 apars( num_ants, 1 ), doa( num_doas ), wavelen
0039            complex*16 sigmat( num_ants, num_doas )
0040      c
0041            integer*4 dc_type/7/, linear(1), va_status
0042            character*80 va_msg
0043      c
0044            integer*4 sigvec_add, i
0045      c
0046      c             Make a place to receive computed signal vectors.
0047      c
0048            linear( 1 ) = num_ants
0049            call open_va( sigvec_add, linear, 1, dc_type, va_status, va_msg )
0050            call exit_va( va_status, va_msg )
0051      c
0052      c             Now compute signal vectors in turn, placing the results into
0053      c       the columns of sigmat.
0054      c
0055            do i = 1, num_doas
0056              call sigval( apars, num_ants, doa(i), wavelen, %val( sigvec_add ) )
0057              call dc_colfill( sigmat, num_ants, num_doas, i, %val( sigvec_add ) )
```

```
SIGMAT_ESTM                                                    18-Apr-1988 14:30:50
                                                               11-Mar-1988 08:31:27

0058              end do
0059     c
0060     c                Discard work array and exit.
0061     c
0062              call free_va( sigvec_add, va_status, va_msg )
0063              call exit_va( va_status, va_msg )
0064     c
0065              return
0066              end


PROGRAM SECTIONS

     Name                                    Bytes   Attributes

  0 $CODE                                      225   PIC CON REL LCL    SHR    EXE    RD NOW
  1 $PDATA                                       4   PIC CON REL LCL    SHR NOEXE    RD NOW
  2 $LOCAL                                     360   PIC CON REL LCL NOSHR NOEXE    RD    W

     Total Space Allocated                     589


ENTRY POINTS

     Address  Type  Name

  0-00000000        SIGMAT_ESTM     -


VARIABLES

     Address  Type  Name              Address  Type  Name              Address  Type  I

  2-00000054  I*4   DC_TYPE         2-00000060  I*4   I              AP-00000008@ I*4   I
  2-0000005C  I*4   SIGVEC_ADD     2-00000004  CHAR  VA_MSG           2-00000058  I*4   '


ARRAYS

     Address  Type  Name                  Bytes  Dimensions

 AP-00000004@ R*8   APARS                   **  (*, 1)
 AP-0000000C@ R*8   DOA                     **  (*)
  2-00000000  I*4   LINEAR                   4  (1)
 AP-00000018@ C*16  SIGMAT                  **  (*, *)


FUNCTIONS AND SUBROUTINES REFERENCED

  Type  Name              Type  Name              Type  Name              Type  Name

        DC_COLFILL              EXIT_VA                 FREE_VA                 OPEN_VA
```

18-Apr-1988 14:31:
10-Sep-1987 08:58::

```
0001            subroutine sigmat_form( apars, num_ants, wpars, num_waves, sigmat
0002      c
0003      c-----------------------------------------------------------------------
0004      c
0005      c               This subroutine forms the "signal matrix", whose columns a
0006      c       the "signal vectors" determined by subroutine SIGVAL.
0007      c
0008      c       apars = matrix whose first column is the vector of ranges (from th
0009      c                       origin) of the antenna array elements, and whose s
0010      c                       column is the vector of angles (from the "x" axis)
0011      c                       the antenna array elements (RADIANS).  num_ants x
0012      c                       REAL*8.
0013      c               Note that the position of the ith antenna element is
0014      c                       apars(i,1) * cos( apars(i,2) ) (x position),
0015      c                       apars(i,1) * sin( apars(i,2) ) (y position).
0016      c       num_ants = number of antenna elements in the array.  INTEGER*4.
0017      c       wpars = matrix whose third column is the vector of angles (from th
0018      c                       axis) of arrival of the incident waves, and whose
0019      c                       column is the vector of wavelengths of the inciden
0020      c                       waves.  num_waves x 9.  REAL*8.  (Note that the un
0021      c                       of wavelength are assumed to be those in which the
0022      c                       array ranges are expressed.)
0023      c       num_waves = number of incident waves for which the signal matrix i
0024      c                       to be formed.  INTEGER*4.
0025      c       sigmat = computed signal matrix.  num_ants rows, num_waves columns
0026      c                       COMPLEX*16.
0027      c
0028      c-----------------------------------------------------------------------
0029      c
0030            implicit none
0031      c
0032            external sigval, dc_colfill
0033            external open_va, free_va, exit_va
0034      c
0035            integer*4 num_ants, num_waves
0036            real*8 apars( num_ants, 1 ), wpars( num_waves, 1 )
0037            complex*16 sigmat( num_ants, num_waves )
0038      c
0039            integer*4 dc_type/7/, linear(1), va_status
0040            character*80 va_msg
0041      c
0042            integer*4 sigvec_add, i
0043      c
0044      c               Make a place to receive computed signal vectors.
0045      c
0046            linear( 1 ) = num_ants
0047            call open_va( sigvec_add, linear, 1, dc_type, va_status, va_msg )
0048            call exit_va( va_status, va_msg )
0049      c
0050      c               Now compute signal vectors in turn, placing the results in
0051      c       the columns of sigmat.
0052      c
0053            do i = 1, num_waves
0054              call sigval( apars, num_ants,
0055           1                          wpars(i,3), wpars(i,9), %val( sigvec_add )
0056              call dc_colfill( sigmat, num_ants, num_waves, i, %val( sigvec_ad
0057            end do
```

```
SIGMAT_FORM                                                    18-Apr-1988 14:31
                                                               10-Sep-1987 08:58

0058     c
0059     c               Discard work array and exit.
0060     c
0061               call free_va( sigvec_add, va_status, va_msg )
0062               call exit_va( va_status, va_msg )
0063     c
0064               return
0065               end


PROGRAM SECTIONS

     Name                                    Bytes   Attributes

  0 $CODE                                      248   PIC CON REL LCL   SHR   EXE   RD
  1 $PDATA                                       4   PIC CON REL LCL   SHR NOEXE   RD
  2 $LOCAL                                     372   PIC CON REL LCL NOSHR NOEXE   RD

     Total Space Allocated                     624

ENTRY POINTS

     Address  Type  Name

   0-00000000       SIGMAT_FORM

VARIABLES

     Address  Type  Name              Address  Type  Name              Address  Ty

   2-00000054  I*4  DC_TYPE         2-00000060  I*4  I              AP-00000008@ I
   2-0000005C  I*4  SIGVEC_ADD      2-00000004  CHAR VA_MSG          2-00000058  I

ARRAYS

     Address  Type  Name                 Bytes  Dimensions

  AP-00000004@ R*8  APARS                  **  (*, 1)
   2-00000000  I*4  LINEAR                  4  (1)
  AP-00000014@ C*16 SIGMAT                 **  (*, *)
  AP-0000000C@ R*8  WPARS                  **  (*, 1)

FUNCTIONS AND SUBROUTINES REFERENCED

   Type  Name             Type  Name             Type  Name             Type  Name

         DC_COLFILL             EXIT_VA                FREE_VA                OPEN_VA
```

18-Apr-1988 14:31:25
9-Sep-1987 16:22:11

```
0001            subroutine sigval( apars, num_ants, arriv_angl, wavelen, sig_vec_val
0002      c
0003      c-------------------------------------------------------------------------
0004      c
0005      c               This subroutine computes the value of the signal vector from
0006      c       array of receivers due to an incident wave, as a function of the angl
0007      c       of arrival of the incident wave.
0008      c
0009      c       Parameters:
0010      c       -----------
0011      c               apars = matrix whose first column is the vector of ranges (fr
0012      c                       the origin) of array antenna elements, and
0013      c                       whose second column is the vector of angles
0014      c                       (from the "x" axis) of array antenna elements
0015      c                       (RADIANS).  num_ants x 2.  REAL*8.
0016      c                   Note:  the position of the ith antenna element is
0017      c                   apars(i,1) * cos( apars(i,2) ) (x position),
0018      c                   apars(i,1) * sin( apars(i,2) ) (y position).
0019      c               num_ants = number of elements in antenna array.  INTEGER*4.
0020      c               arriv_angl = angle (from the "x" axis) of arrival for which t
0021      c                       signal vector value is to be computed.
0022      c                       Expressed in RADIANS.  REAL*8.
0023      c               wavelen = wavelength of the incident wave.  REAL*8.  (Note th
0024      c                       the units of wavelength must be the same as
0025      c                       those in which the array_range(*) are
0026      c                       expressed.)
0027      c               sig_vec_val = computed signal vector value.  num_ants element
0028      c                       COMPLEX*16.  Each element is a complex value
0029      c                       expressing the phase shift due to the positic
0030      c                       of the corresponding array antenna element,
0031      c                       based on the incident wavelength and the angl
0032      c                       of arrival.
0033      c
0034      c-------------------------------------------------------------------------
0035      c
0036            implicit none
0037      c
0038            include '(real_constants)'
0039      c
0040            intrinsic dcos, dsin, dcmplx
0041      c
0042            integer*4 num_ants
0043            real*8 apars( num_ants, 1 ), arriv_angl, wavelen
0044            complex*16 sig_vec_val( num_ants )
0045      c
0046            integer*4 cond_code, i
0047            real*8 phase, phase_rate
0048      c
0049            phase_rate = pi8t2 / wavelen
0050      c
0051            do i = 1, num_ants
0052              phase = phase_rate * apars(i,1) * dcos ( apars(i,2) - arriv_angl )
0053              sig_vec_val(i) = dcmplx( dcos( phase ), -dsin( phase ) )
0054            end do
0055      c
0056            return
0057            end
```

18-Apr-1988 14:31:3
9-Sep-1987 16:59:1

```
0001            subroutine sigvec( num_waves, carr_camp, wpars, mtypes,
0002            1                                          samp_num, samp_int, sig_vec
0003      c
0004      c-----------------------------------------------------------------------
0005      c
0006      c               This subroutine computes the "signal-in-space" vector,
0007      c       consisting of the "complex amplitudes" (amplitudes and phases,
0008      c       expressed in "rectangular" coordinates) of the arriving waves for
0009      c       sample number samp_num.
0010      c
0011      c       Parameters:
0012      c       -----------
0013      c               num_waves = INTEGER*4 number of waves impinging on the ante
0014      c                           array.
0015      c               carr_camp = num_waves-element COMPLEX*16 vector of carrier
0016      c                           complex amplitudes (from SNRs and "starting'
0017      c                           phases).
0018      c               wpars = num_waves x 9 REAL*8 matrix whose columns include:
0019      c                           4) vector of modulation cycle lengths, in
0020      c                                   number of carrier cycles per modula
0021      c                                   cycle.
0022      c                           5) vector of modulation duty cycles (expres
0023      c                                   as fractions of modulation cycle
0024      c                                   lengths, 0.0 < mod_duty(i) <= 1.0)
0025      c                                   (apply only to ramp and rectangular
0026      c                                   modulation).
0027      c                           6) vector of "starting" modulation phases,
0028      c                                   number of carrier cycles.
0029      c                           7) vector of amplitude modulation ratios
0030      c                                   (maximum / minimum).
0031      c                           8) vector of phase modulation total phase
0032      c                                   shifts (in RADIANS).
0033      c               mtypes = num_waves x 2 INTEGER*4 matrix whose first column
0034      c                           the vector of amplitude modulation types an
0035      c                           whose second column is the vector of phase
0036      c                           modulation types (see below).
0037      c               samp_num = INTEGER*4 number of the sample for which the com
0038      c                           amplitudes are to be computed.
0039      c               samp_int = REAL*8 number of carrier cycles between samples.
0040      c               sig_vec = num_waves-element COMPLEX*16 output vector of com
0041      c                           amplitudes.
0042      c
0043      c               Modulation types:  1 = sine, 2 = triangle, 3 = increasing r
0044      c       4 = decreasing ramp, 5 = rectangular.
0045      c
0046      c-----------------------------------------------------------------------
0047      c
0048            implicit none
0049            external modulated
0050      c
0051            complex*16 modulated
0052      c
0053            integer*4 num_waves, mtypes( num_waves, 1 )
0054            integer*4 samp_num
0055            real*8 wpars( num_waves, 1 ), samp_int
0056            complex*16 carr_camp( num_waves ), sig_vec( num_waves )
0057      c
```

```
SIGVEC                                                          18-Apr-1988 14:31:
                                                                 9-Sep-1987 16:53:

0058              integer*4 i
0059     c
0060              do i = 1, num_waves
0061                sig_vec(i) = modulated( i, num_waves, carr_camp, wpars, mtypes,
0062              1                                                      samp_num, samp_in
0063              end do
0064     c
0065              return
0066              end


PROGRAM SECTIONS

     Name                                    Bytes   Attributes

  0 $CODE                                      184   PIC CON REL LCL   SHR   EXE   RD
  2 $LOCAL                                     228   PIC CON REL LCL NOSHR NOEXE   RD

     Total Space Allocated                     412


ENTRY POINTS

     Address  Type  Name

   0-00000000       SIGVEC


VARIABLES

     Address  Type  Name              Address  Type  Name              Address  Ty

   2-00000000  I*4  I              AP-00000004@ I*4  NUM_WAVES      AP-00000018@ R


ARRAYS

     Address  Type  Name                  Bytes  Dimensions

  AP-00000008@ C*16 CARR_CAMP               **  (*)
  AP-00000010@ I*4  MTYPES                  **  (*, 1)
  AP-0000001C@ C*16 SIG_VEC                 **  (*)
  AP-0000000C@ R*8  WPARS                   **  (*, 1)


FUNCTIONS AND SUBROUTINES REFERENCED

   Type  Name

    C*16 MODULATED
```

```
                                                                    18-Apr-1988 14:38:26
                                                                    2-Apr-1988 17:00:57

0001            integer*4 function spawn_if( log_unit, prog_num )
0002    c
0003    c-------------------------------------------------------------------------------
0004    c
0005    c               This function reads (and locks) the record with primary key
0006    c       value prog_num in the indexed-organization file open on logical unit
0007    c       number log_unit.  If the specified record is locked when the read
0008    c       attempt is made, the function loops until the read (and lock) is
0009    c       successful.
0010    c               Once the desired record has been read and locked, the functi
0011    c       checks the "spawned" indication in the record and, if it is 0, spawn
0012    c       a subprocess which executes the command file named in the record, se
0013    c       the "spawned" indication to -1, and rewrites (and unlocks) the recor
0014    c       If the "spawned" indication is already -1, the function returns with
0015    c       taking any action.
0016    c               The value returned by the function is 0 if all operations
0017    c       succeeded (whether the subprocess was spawned or not), 1 if the read
0018    c       failed, 2 if the read succeeded but the spawn failed, and 3 if both
0019    c       read and spawn succeeded but the rewrite failed.
0020    c               The FORTRAN program MAKE_SPAWN_IDX can be used to create a f
0021    c       of the required format.  The text file
0022    c       6000$RT:[CHUCK.RESEARCH.FORTDIR]SPAWNED_DEF.TXT contains the
0023    c       definitions required to access such a file.  The parameters log_unit
0024    c       and prog_num are INTEGER*4.
0025    c
0026    c-------------------------------------------------------------------------------
0027    c
0028            implicit none
0029            external lib$spawn, lib$wait
0030            include '(system_symbols)'
0148            include 'SYS$LIBRARY:FORIOSDEF'
0234            include '6000$RT:[CHUCK.RESEARCH.FORTDIR]SPAWNED_DEF.TXT'
0277    c
0278            integer*4 lib$spawn
0279    c
0280            integer*4 log_unit, prog_num
0281    c
0282            integer*4 out_int
0283            character*1  uparrow( 3 )/ 27, '[', 'A' /, spaces( 70 )/ 70*' ' /
0284    c
0285    10000   continue
0286            read (unit=log_unit,keyeq=prog_num,keyid=0,iostat=spawn_if)
0287            1                                   sp_num, sp_stat, sp_file, sp_proc
0288            if      ( spawn_if .eq. FOR$IOS_SPERECLOC ) then        !  Record lo
0289              unlock (unit=log_unit)
0290              type *, 'Spawn:  Record #', prog_num, ' locked.  Waiting 1 sec.',
0291            1                                                                upar
0292              call lib$wait( 1.0 )
0293              type *, spaces, uparrow
0294              go to 10000
0295            else if ( spawn_if .ne. 0 ) then                        !  Error rea
0296              unlock (unit=log_unit)
0297              spawn_if = 1
0298              return
0299            end if
0300    c
0301            if ( sp_stat .eq. 0 ) then              !  Batch program not started
```

```
SPAWN_IF                                                    18-Apr-1988 14:38:2
                                                            2-Apr-1988 17:00:5

0302              spawn_if = lib$spawn( , sp_file, '6000$RT:[CHUCK.PARAMS]SPAWN', ,
0303             1                                                        out_int )
0304              if ( spawn_if .ne. SS$_NORMAL ) then
0305                unlock (unit=log_unit)
0306                spawn_if = 2
0307                return
0308              end if
0309     c
0310              sp_stat = -1
0311              rewrite (unit=log_unit,err=20000) sp_num, sp_stat, sp_file, sp_pr
0312             end if
0313     c
0314             unlock (unit=log_unit)
0315             spawn_if = 0
0316             return
0317     c
0318     20000   continue                                          !  Rewrite error.
0319             unlock (unit=log_unit)
0320             spawn_if = 3
0321             return
0322     c
0323             end


PROGRAM SECTIONS

    Name                                  Bytes   Attributes

  0 $CODE                                   365   PIC CON REL LCL   SHR   EXE   RD N
  1 $PDATA                                   72   PIC CON REL LCL   SHR NOEXE   RD N
  2 $LOCAL                                  300   PIC CON REL LCL NOSHR NOEXE   RD

    Total Space Allocated                   737


ENTRY POINTS

    Address  Type  Name

  0-00000000  I*4  SPAWN_IF


VARIABLES

    Address  Type  Name             Address  Type  Name             Address  Type

 AP-00000004@ I*4  LOG_UNIT      2-000000B4  I*4  OUT_INT      AP-00000008@ I*4
  2-000000AC  I*4  SP_NUM        2-00000049  CHAR SP_PROC       2-000000B0  I*4
```

18-Apr-1988 14:38:46
20-Feb-1988 18:22:34

```
0001            real*8 function spect_peak( iu, il, tol, p, apars, num_ants,
0002            1                                          wavelen, eigvec, num_zeigs )
0003    c
0004    c-------------------------------------------------------------------------------
0005    c
0006    c                   This function returns the angle, in RADIANS, at which a peak
0007    c       occurs in the MUSIC DOA spectrum, given an interval to search for th
0008    c       peak.  The method used is called the "golden section search" (see
0009    c       Haykin, Simon, "Radar Array Processing for Angle of Arrival
0010    c       Estimation", ARRAY SIGNAL PROCESSING, S. Haykin, Editor, Prentice-Ha
0011    c       Inc., Englewood Cliffs, New, Jersey, 1985, pp. 213-215).
0012    c
0013    c       Parameters:
0014    c       -----------
0015    c                   il:  REAL*8 lower limit of interval to search, in RADIANS.
0016    c                   iu:  REAL*8 upper limit of interval to search, in RADIANS.
0017    c                   tol:  REAL*8 value of desired resolution, in RADIANS.  When
0018    c                                 interval being searched is reduced to <= tol
0019    c                                 radians, or when the search interval stops
0020    c                                 changing between iterations, the routine exi
0021    c                                 with a result of the midpoint of the current
0022    c                                 search interval.
0023    c                   p:  REAL*4 power to which to raise the terms used in
0024    c                                 accumulating a DOA spectrum sample (p = 2 fo
0025    c                                 "ordinary" MUSIC).
0026    c                   apars:  matrix whose first column is the vector of ranges (f
0027    c                                 the origin) of array antenna elements and wh
0028    c                                 second column is the vector of angles (from
0029    c                                 "x" axis) of array antenna elements (in
0030    c                                 RADIANS).  num_ants x 2.  REAL*8.
0031    c                           Note:  the position of the ith antenna element is
0032    c                           apars(i,1) * cos( apars(i,2) ) (x position),
0033    c                           apars(i,1) * sin( apars(i,2) ) (y position).
0034    c                   num_ants:  number of elements in antenna array.  INTEGER*4.
0035    c                   wavelen:  wavelength of the incident wave.  REAL*8.  (Note t
0036    c                                 the units of wavelength must be the same as
0037    c                                 those in which the array_range(*) are
0038    c                                 expressed.)
0039    c                   eigvec:  num_ants x num_ants COMPLEX*16 matrix whose FIRST
0040    c                                 num_zeigs columns are the "noise" generalize
0041    c                                 eigenvectors defined in the MUSIC algorithm.
0042    c                   num_zeigs:  INTEGER*4 number of "noise" eigenvectors to be u
0043    c                                 in evaluating the MUSIC DOA measure.
0044    c
0045    c-------------------------------------------------------------------------------
0046    c
0047            implicit none
0048            external open_va, exit_va, free_va
0049            external music_power, sigval
0050    c
0051            real*8 music_power
0052    c
0053            integer*4 num_ants, num_zeigs
0054            real*4 p
0055            real*8 il, iu, tol, apars( num_ants, 1 ), wavelen
0056            complex*16 eigvec( num_ants, num_ants )
0057    c
```

```
SPECT_PEAK                                                    18-Apr-1988 14:38:46
                                                              20-Feb-1988 18:22:34

0058            integer*4 steerv_add, linear(1), status, dc_type/7/
0059            real*8 u, l, a, b, fa, fb, ivl, oldivl, tau/1.6180339887498895/
0060            character*80 msg
0061      c
0062            linear(1) = num_ants
0063            call open_va( steerv_add, linear, 1, dc_type, status, msg )
0064            call exit_va( status, msg )
0065      c
0066            if ( iu .gt. il ) then
0067              u = iu
0068              l = il
0069            else
0070              u = il
0071              l = iu
0072            end if
0073            ivl = u - l
0074            a = u - ivl / tau
0075            call sigval( apars, num_ants, a, wavelen, %val( steerv_add ) )
0076            fa = music_power( eigvec, %val( steerv_add ), num_ants, num_zeigs, p
0077            b = l + ivl / tau
0078            call sigval( apars, num_ants, b, wavelen, %val( steerv_add ) )
0079            fb = music_power( eigvec, %val( steerv_add ), num_ants, num_zeigs, p
0080      c
0081            oldivl = tau * ivl
0082            do while ( ( ivl .gt. tol ) .and. ( ivl .lt. oldivl ) )
0083              oldivl = ivl
0084      c
0085              if ( fa .lt. fb ) then
0086                l   = a
0087                ivl = u - l
0088                a   = b
0089                fa  = fb
0090                b   = l + ivl / tau
0091                call sigval( apars, num_ants, b, wavelen, %val( steerv_add ) )
0092                fb = music_power( eigvec, %val( steerv_add ),
0093           1                                          num_ants, num_zeigs, p )
0094              else
0095                u   = b
0096                ivl = u - l
0097                b   = a
0098                fb  = fa
0099                a   = u - ivl / tau
0100                call sigval( apars, num_ants, a, wavelen, %val( steerv_add ) )
0101                fa = music_power( eigvec, %val( steerv_add ),
0102           1                                          num_ants, num_zeigs, p )
0103              end if
0104            end do
0105      c
0106            call free_va( steerv_add, status, msg )
0107            call exit_va( status, msg )
0108      c
0109            spect_peak = ( u + l ) / 2.0d0
0110            return
0111            end
```

```
                                                        18-Apr-1988 14:39:08    VA
                                                        27-Mar-1988 15:10:03    SQ

0001            subroutine sq_vlgsvd( a, b, n, fuzz, eig, vec, condit, info )
0002    c
0003    c-------------------------------------------------------------------------------
0004    c
0005    c               This subroutine uses Van Loan's simplified algorithm for the
0006    c       generalized singular value decomposition (SVD) to compute the
0007    c       generalized eigenvalues and eigenvectors of the matrix pair (a, b).
0008    c
0009    c       Parameters
0010    c       ----------
0011    c               a, b    n x n COMPLEX*16 UPPER TRIANGULAR, non-singular
0012    c                                       matrices.
0013    c               n       INTEGER*4 number of rows / columns in a and b.
0014    c               fuzz    REAL*8 tolerance value ("closeness to zero").
0015    c               eig     n-element REAL*8 vector of generalized eigenvalues.
0016    c               vec     n x n COMPLEX*16 matrix of generalized eigenvectors
0017    c                                       (columns of vec are eigenvectors).
0018    c               condit  REAL*8 estimate of the inverse of the condition number
0019    c                                       of matrix a.
0020    c               info    INTEGER*4 success indicator.  0 if routine succeeded,
0021    c                                       > 0 if a or b is singular, -1 if any
0022    c                                       singular value has a significant
0023    c                                       imaginary part, < -1 if any computed
0024    c                                       singular values / vectors are in error.
0025    c
0026    c-------------------------------------------------------------------------------
0027    c
0028            implicit none
0029            external dc_matmpy, va_dc_trinv, va_dc_sqsvd, dc_matcopy, dc_vecmaxim
0030            external open_va, exit_va, eig_from_sv
0031    c
0032            real*8 dc_vecmaxim
0033    c
0034            integer*4 n, info
0035            real*8 fuzz, eig(n), condit
0036            complex*16 a(n,n), b(n,n), vec(n,n)
0037    c
0038            integer*4 status, linear(1), square(2), sv_err
0039            integer*4 bai_add, sv_add, imt_add, ai_add, rmt_add
0040            integer*4 dr_type/5/, dc_type/7/
0041            character*80 msg
0042    c
0043            linear( 1 ) = n
0044            square( 1 ) = n
0045            square( 2 ) = n
0046    c
0047    c               Create working space.
0048    c
0049            call open_va( ai_add, square, 2, dc_type, status, msg )
0050            call exit_va( status, msg )
0051            call open_va( bai_add, square, 2, dc_type, status, msg )
0052            call exit_va( status, msg )
0053            call open_va( sv_add, linear, 1, dc_type, status, msg )
0054            call exit_va( status, msg )
0055            call open_va( imt_add, square, 2, dc_type, status, msg )
0056            call exit_va( status, msg )
0057            call open_va( rmt_add, square, 2, dc_type, status, msg )
```

```
SQ_VLGSVD                                                    18-Apr-1988 14:39:08     V
                                                             27-Mar-1988 15:10:03     S

0058              call exit_va( status, msg )
0059        c
0060        c                 Compute a inverse.
0061        c
0062              call dc_matcopy( a, n, n, %val( ai_add ) )
0063              call va_dc_trinv( %val( ai_add ), n, condit, info )
0064              if ( info .ne. 0 ) go to 10000
0065        c
0066        c                 Do SVD of b * a-inverse.
0067        c
0068              call dc_matmpy( b, n, n, %val( ai_add ), n, %val( bai_add ) )
0069              call va_dc_sqsvd( %val( bai_add ), n, %val( sv_add ), %val( lmt_add ),
0070             1                  %val( rmt_add ), sv_err )
0071              if ( dc_vecmaxim( %val( sv_add ), n ) .gt. fuzz ) info = -1
0072              if ( sv_err .gt. 0 ) info = - 1 - sv_err
0073        c
0074        c                 Compute eigenvalues and eigenvectors from a inverse and SVD of
0075        c         b * a-inverse.
0076        c
0077              call dc_matmpy( %val( ai_add ), n, n,  %val( rmt_add ), n, vec )
0078              call eig_from_sv( %val( sv_add ), n, fuzz, eig )
0079        c
0080        c                 Release working space and exit.
0081        c
0082        10000     continue
0083              call free_va( ai_add, status, msg )
0084              call exit_va( status, msg )
0085              call free_va( bai_add, status, msg )
0086              call exit_va( status, msg )
0087              call free_va( sv_add, status, msg )
0088              call exit_va( status, msg )
0089              call free_va( lmt_add, status, msg )
0090              call exit_va( status, msg )
0091              call free_va( rmt_add, status, msg )
0092              call exit_va( status, msg )
0093        c
0094              return
0095              end
```

```
SQ_VLGSVD                                                  18-Apr-1988 14:39:08
                                                           27-Mar-1988 15:10:03

PROGRAM SECTIONS

     Name                                  Bytes   Attributes

  0 $CODE                                    590   PIC CON REL LCL   SHR   EXE   RD NOWRT
  1 $PDATA                                     8   PIC CON REL LCL   SHR NOEXE   RD NOWRT
  2 $LOCAL                                   648   PIC CON REL LCL NOSHR NOEXE   RD   WRT

     Total Space Allocated                  1246

ENTRY POINTS

     Address  Type  Name

  0-00000000       SQ_VLGSVD

VARIABLES

     Address  Type  Name             Address  Type  Name             Address  Type  Na

  2-00000070  I*4  AI_ADD         2-00000064  I*4  BAI_ADD       AP-0000001C@ R*8  CO
     **       I*4  DR_TYPE      AP-00000010@ R*8  FUZZ          AP-00000020@ I*4  IN
  2-0000000C  CHAR MSG           AP-0000000C@ I*4  N              2-00000074  I*4  RM
  2-00000068  I*4  SV_ADD         2-00000060  I*4  SV_ERR

ARRAYS

     Address  Type  Name                Bytes  Dimensions

 AP-00000004@ C*16 A                     **  (*, *)
 AP-00000008@ C*16 B                     **  (*, *)
 AP-00000014@ R*8  EIG                   **  (*)
  2-00000000  I*4  LINEAR                 4  (1)
  2-00000004  I*4  SQUARE                 8  (2)
 AP-00000018@ C*16 VEC                   **  (*, *)

LABELS

     Address   Label

  0-000001F3  10000

FUNCTIONS AND SUBROUTINES REFERENCED

  Type  Name             Type  Name             Type  Name             Type  Name

        DC_MATCOPY             DC_MATMPY        R*8  DC_VECMAXIM           EIG_FROM_SV
        OPEN_VA                VA_DC_SQSVD           VA_DC_TRINV
```

18-Apr-1988 14:39:57
11-Nov-1987 09:13:34

```
0001            subroutine va_dc_qr( matrix, rows, cols, out_mat )
0002    c
0003    c-------------------------------------------------------------------------------
0004    c                This subroutine uses subroutines from LINPAK and the virtual
0005    c       array system (copyright 1987 by Dwight Day) to perform a QR
0006    c       decomposition of the rows x cols COMPLEX*16 matrix matrix.  The indeces
0007    c       rows and cols are INTEGER*4.
0008    c                On exit, the cols x cols COMPLEX*16 matrix out_mat contains
0009    c       the upper triangular result of the decomposition (the R part of the QR
0010    c       decomposition).
0011    c                Note that rows MUST BE GREATER THAN OR EQUAL TO cols.  If this
0012    c       is not the case, the routine EXITS WITHOUT DOING ANY COMPUTATION!.
0013    c-------------------------------------------------------------------------------
0014    c
0015            implicit none
0016            external zqrdc, open_va, free_va, exit_va
0017    c
0018            integer*4 rows, cols
0019            complex*16 matrix(rows,cols), out_mat(cols,cols)
0020    c
0021            integer*4 i, j, status
0022            integer*4 dint_type/9/, dcmp_type/7/, linear(1)
0023            integer*4 work1, work2, work3
0024            character*80 msg
0025    c
0026            if ( rows .lt. cols ) return
0027    c
0028            linear(1) = cols
0029            call open_va( work1, linear, 1, dcmp_type, status, msg )
0030            call exit_va( status, msg )
0031            call open_va( work2, linear, 1, dint_type, status, msg )
0032            call exit_va( status, msg )
0033            call open_va( work3, linear, 1, dcmp_type, status, msg )
0034            call exit_va( status, msg )
0035    c
0036            call zqrdc( matrix, rows, rows, cols,
0037            1               %val( work1 ), %val( work2 ), %val( work3 ), 0 )
0038    c
0039            call free_va( work1, status, msg )
0040            call exit_va( status, msg )
0041            call free_va( work2, status, msg )
0042            call exit_va( status, msg )
0043            call free_va( work3, status, msg )
0044            call exit_va( status, msg )
0045    c
0046            do i = 1, cols
0047              do j = 1, cols
0048                if ( j .lt. i ) then
0049                  out_mat(i,j) = ( 0.0d0, 0.0d0 )
0050                else
0051                  out_mat(i,j) = matrix(i,j)
0052                end if
0053              end do
0054            end do
0055    c
0056            return
0057            end
```

```
                                                        18-Apr-1988 14:40:22     \
                                                        23-Jun-1987 08:56:18     \

0001            subroutine va_dc_sqsvd( mat, n, sv, left, right, info )
0002    c
0003    c-------------------------------------------------------------------------------
0004    c
0005    c               This subroutine uses subroutine ZSVDC from LINPAK and
0006    c       subroutines from the virtual array system (copyright 1987 by Dwight
0007    c       Day) to perform the singular value decomposition of the n x n
0008    c       COMPLEX*16 matrix mat.  The index n is INTEGER*4.
0009    c               NOTE THAT THE INPUT MATRIX mat IS OVERWRITTEN.
0010    c               Results of the decomposition are returned in the n-element
0011    c       COMPLEX*16 vector sv (singular values) and the n x n COMPLEX*16
0012    c       matrices left (left-hand result matrix) and right (right-hand result
0013    c       matrix).
0014    c               The INTEGER*4 value info is 0 if the SVD succeeded, > 0 if some
0015    c       of the singular values and singular vectors were uncomputable.
0016    c
0017    c-------------------------------------------------------------------------------
0018    c
0019            implicit none
0020            external zsvdc, open_va, exit_va, free_va
0021    c
0022            integer*4 n, info
0023            complex*16 mat(n,n), sv(n), left(n,n), right(n,n)
0024    c
0025            integer*4 dcmp_type/7/, linear(1), status, errv_add, wk_add
0026            character*80 msg
0027    c
0028            linear(1) = n
0029            call open_va( errv_add, linear, 1, dcmp_type, status, msg )
0030            call exit_va( status, msg )
0031            call open_va( wk_add, linear, 1, dcmp_type, status, msg )
0032            call exit_va( status, msg )
0033    c
0034            call zsvdc( mat, n, n, n, sv, %val( errv_add),
0035            1               left, n, right, n, %val( wk_add ), 11, info )
0036    c
0037            call free_va( errv_add, status, msg )
0038            call exit_va( status, msg )
0039            call free_va( wk_add, status, msg )
0040            call exit_va( status, msg )
0041    c
0042            return
0043            end
```

```
                                                    18-Apr-1988 14:39:31    VA
                                                    27-Mar-1988 14:52:52    VA

0001            subroutine va_dc_trinv( matrix, size, condit, panic )
0002    c
0003    c-------------------------------------------------------------------------------
0004    c
0005    c               This subroutine uses the routines ZTRCO and ZTRDI from LINPAK,
0006    c       and subroutines from the Virtual Array system (copyright 1987 by Dwight
0007    c       Day) to estimate the condition number and compute the inverse of the
0008    c       size x size COMPLEX*16 UPPER TRIANGULAR matrix matrix.  The index size
0009    c       is INTEGER*4.  The REAL*8 value condit is set to an estimate of the
0010    c       inverse of the conition number of matrix.  The INTEGER*4 value panic is
0011    c       set to 0 if the inversion is successful, panic > 0 if matrix is
0012    c       singular.
0013    c               NOTE that the inverse REPLACES the input matrix (except that
0014    c       the strictly lower triangle of the matrix is never referenced).
0015    c
0016    c-------------------------------------------------------------------------------
0017    c
0018            implicit none
0019            external ztrco, ztrdi, open_va, exit_va, free_va
0020            include '6000$RT:[CHUCK.RESEARCH.FORTDIR]VATYPES.TXT'
0032    c
0033            integer*4 size, panic
0034            real*8 condit
0035            complex*16 matrix( size, size )
0036    c
0037            integer*4 size_lin( 1 ), work_add, vstatus
0038            character*80 msg
0039    c
0040            size_lin(1) = size
0041            call open_va( work_add, size_lin, 1, dc_type, vstatus, msg )
0042            call exit_va( vstatus, msg )
0043    c
0044            call ztrco( matrix, size, size, condit, %val( work_add ), 1 )
0045    c
0046            call free_va( work_add, vstatus, msg )
0047            call exit_va( vstatus, msg )
0048    c
0049            call ztrdi( matrix, size, size, 0, 011, panic )
0050    c
0051            return
0052            end
```

VITA

Charles William Kriel

Candidate for the Degree of

Doctor of Philosophy

Thesis: L_p-NORM ESTIMATION TECHNIQUES APPLIED TO MULTIPLE EMITTER LOCATION

Major Field: Electrical and Computer Engineering

Biographical:

Personal Data: Born in Stillwater, Oklahoma, August 29, 1952, the son of Karl W. and Olive R. Kriel. Married to Pamela Anne Judkins on July 16, 1977.

Education: Graduated from Perkins High School, Perkins, Oklahoma, in May, 1970; received Bachelor of Science degree in Electrical Engineering and Masters of Electrical Engineering degree from Oklahoma State University in August, 1975; completed requirements for the Doctor of Philosophy degree at Oklahoma State University in May, 1988.

Professional Organizations: Eta Kappa Nu, Tau Beta Pi, Phi Kappa Phi.

Professional Experience: Instructor, Department of Computer Science, Chapman College, 1978; Teaching Assistant, Department of Electrical and Computer Engineering, Oklahoma State University, January,1987, to present.