Naval Research Laboratory
Washington, DC 20375-5320

NRL/MR/5653--99-8419

An Introduction to ARRSTATS — A Computer Program for Simulating the Effects of Errors in Time- and Phase-Steered Planar Array Antennas

C. STAN WEST

Photonics Technology Branch
Optical Sciences Division

November 22, 1999

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave Blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	November 22, 1999	Final

4. TITLE AND SUBTITLE

An Introduction to ARRSTATS — A Computer Program for Simulating the Effects of Errors in Time- and Phase-Steered Planar Array Antennas

5. FUNDING NUMBERS

ONR — 63217N

6. AUTHOR(S)

C. Stan West

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
Washington, DC 20375-5320

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL/MR/5653--99-8419

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Bobby Junker
Office of Naval Research

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

A computer program is described that simulates time- and phase-steered array antennas subject to deterministic and random errors. The modeled array comprises a planar, rectangular grid of phase-steered elements grouped hierarchically into time-steered subarrays and time-steered subapertures. Random phase, time, and amplitude errors may be assigned, and the phases and times may be quantized. The simulation frequency may differ from the design frequency. The program assesses the antenna's performance by computing and analyzing the far-field radiation pattern or an ensemble of statistically identical patterns. It determines the location and power density of the main beam peak, the pointing error, the major and minor beam widths, the directivity, the ratio of main beam and sidelobe powers, and other measures. The program can tabulate measures as functions of a user-specified independent variable such as the operating frequency, a steering angle, an error parameter, a quantization specification, or a parameter of the array geometry. Results may be displayed as textual output of the performance measures, plots of radiation patterns, and plots of performance measures versus the independent variable.

14. SUBJECT TERMS

15. NUMBER OF PAGES

54

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std 239-18
298-102

CONTENTS

AN INTRODUCTION TO ARRSTATS — A COMPUTER PROGRAM FOR SIMULATING THE EFFECTS OF ERRORS IN TIME- AND PHASE-STEERED PLANAR ARRAY ANTENNAS

1. INTRODUCTION

Phased array antennas are remarkable for their suitability to many applications, which is partly because they steer quickly, allow adaptive processing, conform to special shapes, and produce a variety of radiation patterns. As low-sidelobe radiation patterns were being developed, it became apparent that errors in the phase or amplitude of the element excitations would limit the lowest achievable sidelobe level. Such errors might be due to manufacturing tolerances on the physical structure, inaccurate phase shifters, or non-uniform feeds, and can affect the average and peak sidelobe levels, beam width, pointing accuracy, and directivity, for example. Although one cannot predict the performance of a given antenna without measurements of the phase and amplitude accuracy of each element, one can draw conclusions about the behavior of an ensemble of antennas that are statistically identical but have different realizations of phase and amplitude errors. The present work applies this approach to a generalized array antenna that blends phase- and time-steering to achieve greater bandwidth at a reasonable cost.[1–3] The purpose of this report is to introduce the reader to a computer program for simulating time- and phase-steered planar array antennas subject to deterministic and random excitation errors. It is intended as an overview to the features and capabilities of the program and a guide to understanding the program's input, processing, and output.

The theoretical study of errors in array antennas has produced a large body of literature. Introductions to these works and key results and derivations may be found in several books.[4–6] The literature on simulations is more sparse, but two programs have been described recently against which this work may be contrasted. First, Chrisman[7] describes a program that simulates phase-steered planar arrays. It computes the directivity and cuts of the design and expected radiation patterns from the error statistics using theoretical formulas. One pattern cut intersects the main beam and boresight, and the other is normal to it through the main beam; from them the beam width is obtained in those directions. Second, Wright[8] briefly discusses the features of a simulation of phase-steered arrays. Its parameters include phase and amplitude errors, number of quantization bits, and bandwidth, and it can output two-dimensional beam patterns, sidelobe statistics, and measures of specialized interest. The program discussed here, called ARRSTATS, differs from those in Refs. 7 and 8 primarily in that it can model arrays with a hybrid phase- and time-steered architecture,[9] including strictly phased arrays and strictly time-steered arrays.[10] Also, it computes individual realizations of the hemispherical radiation pattern and obtains most measures of the array performance directly from the full pattern rather than from formulas or from pattern cuts chosen *a priori*. (Ref. 8 does not specify how its performance measures are obtained.) For example, the beam width cuts are always along the major and minor axes of the beam width ellipse, regardless of its orientation. Another special feature is a method for locating the beam peak that is highly accurate for nearly flat phase fronts; it is the only measure not obtained directly from the radiation pattern.

In more detail, the modeled array comprises a rectangular grid of phase-steered elements; these

Manuscript approved

are grouped into time-steered subarrays, which in turn are grouped into time-steered subapertures. The phases and times may be quantized. A random phase error may be associated with the elements, and random time errors may be assigned to the subarrays and subapertures. Also, random amplitude errors may be associated with the elements, subarrays, and subapertures. The user specifies the design frequency, at which the error-free times and phases would correctly steer the antenna, and the operating frequency, at which the array's behavior is simulated. The antenna may be steered to any direction. ARRSTATS assesses the array's performance by computing and analyzing the far-field radiation pattern or an ensemble of statistically identical patterns. It ignores polarization and mutual coupling between elements and assumes that the elements radiate uniformly into the forward hemisphere. For each computed pattern, it determines the following:

- the location and power density of the main beam peak
- the error in the main beam's location
- the main beam's angular limits, major and minor widths at half power, and orientation
- the directivity
- the ratios of powers in the main beam, sidelobes, and the radiation hemisphere
- the powers and distances of the sidelobes that are strongest and nearest to the main beam

Furthermore, it can track these measures as functions of a user-specified independent variable. ARRSTATS provides textual output of the performance measures, plots of radiation patterns, and plots of performance measures versus the independent variable.

ARRSTATS is a script written for version 5.3 of MATLAB, a commercial software package for technical computing.[11] Some elements of the program structure and some of the graphics facilities take advantage of features in version 5.3, but much of the code is compatible with earlier versions of MATLAB. ARRSTATS consists of one text file and is executed by typing its filename at the MATLAB command prompt. The program does not have an input user interface; instead, the user hardcodes input into the script before execution. These input points are tagged with the word "INPUT" in the code's comments.

The remainder of this report is structured as follows. Section 2 introduces the coordinate systems used for input and output, Section 3 explains the input parameters that describe the array, Section 4 specifies the model for the excitations, and Section 5 outlines the calculation of the radiation pattern. Section 6 describes the pattern measures, Section 7 discusses looping over multiple realizations and parameter values, and Section 8 exhibits the program's output. Two appendices are provided: Appendix A relates the variables used in this report and in the program, and Appendix B is a listing of the program code. Most of the report aims to describe aspects of the program's operation but does not give the details of the implementation or algorithms. The interested reader is directed to the code listing, specifically the comments that introduce each section of the program. Throughout this report, code variables are printed in a `monospaced` face, and braces ({}) enclose references to code line numbers except where the context suggests set notation.

2. COORDINATE SYSTEMS AND PROJECTIONS

ARRSTATS internally uses a three-dimensional Cartesian coordinate system to describe space. The array lies in the x-y plane and radiates into the half-space $z > 0$, as in Figure 1. (Although we describe a transmitter array, the case for a receiver array is identical.) The far-field spatial distribution of this radiation — that is, the radiation pattern — is a function of direction in the half-space. Two variables suffice to specify direction, and several pairs of variables are useful for this purpose. First, direction

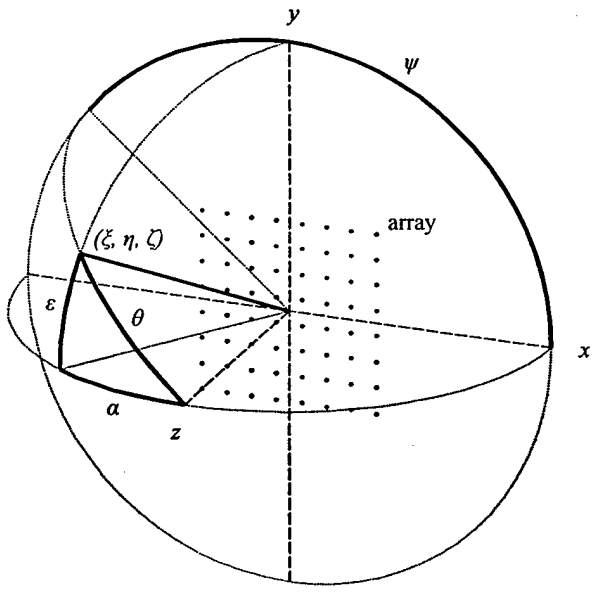

Figure 1 — Cartesian and spherical coordinates; θ, ψ, α, and ε are shown positive

cosines are the natural coordinates for calculating the array factor, as will be seen later. The direction cosines for a given direction are simply the Cartesian coordinates (ξ, η, ζ) of the corresponding unit vector. Specifying a direction in the half-space $z > 0$ requires only ξ and η; ζ may be obtained from the unit vector constraint if needed. Second, spherical coordinates are convenient for constructing flat projections of the radiation pattern, as it may be regarded as a function of location on a (curved) hemisphere. As shown in Figure 1, the polar angle θ for a given vector is the angle between the positive z axis and the vector, while the azimuth angle ψ is the angle between the positive x axis and the projection of the vector onto the x-y plane. Third, traditional antenna coordinates connect these simulations to an established context. Given the projection of a vector onto the x-z plane, the azimuth angle α is the angle between the projection and the z axis, while the elevation angle ε is the angle between the projection and the vector. These three sets of coordinates are related according to

$$\begin{aligned} \xi &= \sin\theta\cos\psi = -\cos\varepsilon\sin\alpha \\ \eta &= \sin\theta\sin\psi = \sin\varepsilon \\ \zeta &= \cos\theta = \cos\varepsilon\cos\alpha \end{aligned} \tag{1}$$

$$\begin{aligned} \cos\theta &= \cos\varepsilon\cos\alpha = \zeta \\ \tan\psi &= -\tan\varepsilon/\sin\alpha = \eta/\xi \end{aligned} \tag{2}$$

$$\begin{aligned} -\tan\alpha &= \xi/\zeta = \tan\theta\cos\psi \\ \sin\varepsilon &= \eta = \sin\theta\sin\psi. \end{aligned} \tag{3}$$

ARRSTATS also employs three projections of the hemisphere onto flat two-dimensional space: orthographic, Lambert azimuthal, and stereographic. The orthographic projection yields a view of the hemisphere from a particular vantage point without perspective distortion and is available for displaying the radiation pattern. The remaining projections both map the hemisphere to a disk such that boresight

corresponds to the center of the disk and grazing directions correspond to the perimeter of the disk. To describe these projections more specifically, we denote locations on the disk using polar coordinates (radius and angle). For both projections, the angular coordinate is set equal to the spherical azimuth angle ψ; the mapping from the spherical polar angle θ to the radius r distinguishes the two projections.

The Lambert projection preserves relative area: the ratio of two areas on the hemisphere equals that of the corresponding areas on the projected disk.[12] This property may be expressed by equating (to a proportionality constant) the spherical and planar surface areas:

$$\sin\theta \, d\theta \, d\psi = C r_L \, dr_L \, d\psi , \tag{4}$$

where the subscript "L" distinguishes the radius in the Lambert projection from that in the stereographic projection below. Canceling $d\psi$ and integrating both sides produces an integration constant, whose value and that of C are determined by the constraints that $r_L = 0$ when $\theta = 0$ and $r_L = 1$ when $\theta = \pi / 2$. One finds

$$r_L = \sqrt{2} \sin\frac{\theta}{2}, \tag{5}$$

which deviates only slightly from a linear relationship for $\theta \in [0, \pi / 2]$. When the radiation pattern is plotted with the Lambert projection, the areas occupied by structures such as the main beam and sidelobes are in true proportion to each other and to the total area.

While the stereographic projection does not preserve area, it is conformal.[12] The local scale is uniform in any direction; shape is preserved locally. Great and small circles on the hemisphere are projected into circles or straight lines, and the angle between two great circles on the hemisphere equals the angle at the intersection of their projected images. To derive the governing relationship, equate the aspect ratios of orthogonal derivatives, as

$$\frac{d\theta}{\sin\theta \, d\psi} = \frac{dr_S}{r_S \, d\psi}, \tag{6}$$

where the subscript "S" denotes the stereographic projection. Canceling and integrating as before yields

$$r_S = \tan\frac{\theta}{2}. \tag{7}$$

ARRSTATS internally uses the stereographic projection when identifying the major and minor axes of the beamwidth ellipse (see Section 6.3) and makes it available for plotting the radiation pattern.

3. ARRAY PARAMETERS

Several parameters describe the antenna's geometry and associated frequencies {29–94}. As Figure 2 illustrates, the antenna elements occupy a regular rectangular grid in the x-y plane; the element spacings in each dimension, d_x and d_y, may differ. The elements are grouped hierarchically at three levels. The array is subdivided into L_x by L_y subapertures, each of which has an associated time delay for steering. Descending to the next level, each subaperture contains M_x by M_y subarrays, each of which likewise has a steering time delay. Finally, each subarray contains N_x by N_y elements, each of which is phase-steered. Regular element spacing is maintained across subarray and subaperture boundaries. To

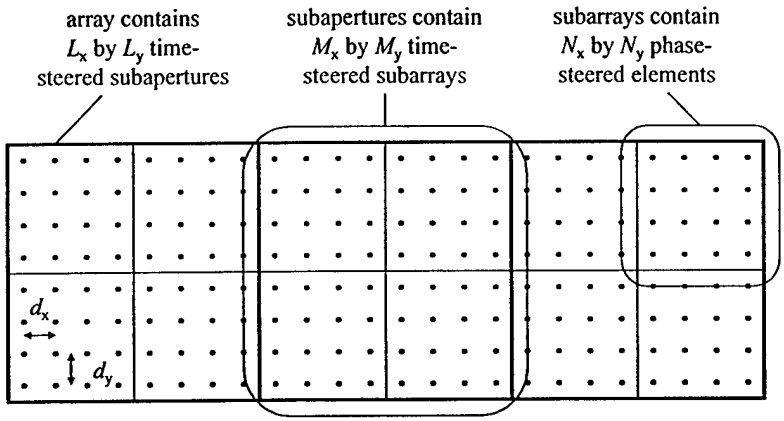

Figure 2 — Array structure and geometry

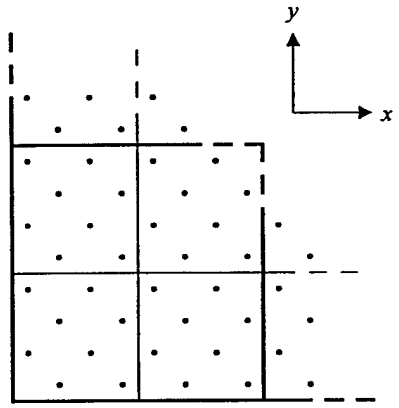

Figure 3 — Element layout when `diamond` flag is nonzero

simulate an array with one time-steered level and one phase-steered level, L_x and L_y should be set to 1; for a strictly phased array, also set M_x and M_y to 1. On the other hand, one may model a strictly time-steered array with one or two hierarchical levels by setting N_x and N_y to 1 and $\delta\varphi$ to 360° (this renders the phase shifters ineffective; see Section 4.1).

Two additional program parameters specify special antenna geometries. If the flag `diamond` is nonzero, elements are effectively removed from even diagonals (the zeroth diagonal originates at the lower-left element), leaving a diamond pattern of active elements as in Figure 3. The parameter `azmthOffst` rotates the antenna about the z axis (boresight); it is the angle of the positive x axis of the array above the azimuth reference. Within ARRSTATS, all calculations are performed in the antenna coordinate system; the steering vector provided by the user is transformed to the antenna coordinate system before processing, and output coordinates and plotted structures are transformed from the antenna coordinate system after processing.

Finally, two parameters supply frequency information. The reference or design frequency f_0 is that at which the time and phase delays would correctly steer the antenna in the absence of error and quantization. The operating frequency f is that at which the simulation is to be performed. Multiple frequencies may be considered sequentially as described in Section 7.

4. EXCITATIONS

4.1. Error-Free Excitations

We now construct the element excitations in detail to show the structure of the array model, beginning with the error-free case {338–410}. Because only one frequency is considered at a time, time delays may be expressed as equivalent phase delays. We therefore decompose the excitations into magnitude and phase as

$$\bar{a}_{n_x n_y} = |\bar{a}_{n_x n_y}| \exp(-i\bar{\Phi}_{n_x n_y}), \quad n_w \in \{0,1,\ldots,L_w M_w N_w - 1\}, \quad w \in \{\mathrm{x},\mathrm{y}\}, \tag{8}$$

where the overbars indicate the error-free case and the n_w label the elements across the face of the array, ignoring subaperture and subarray boundaries. The error-free magnitudes are made equal for all active elements and normalized to unit total power, so that

$$\sum_{n_x}\sum_{n_y} |\bar{a}_{n_x n_y}|^2 = 1. \tag{9}$$

The phases are derived from the condition that at the reference frequency the far-field radiation must interfere perfectly constructively in the direction of the steering vector. This implies that the phase must progress linearly across the face of the array as

$$\bar{\Phi}_{n_x n_y} = -k_0(d_x s_x n_x + d_y s_y n_y) + \text{const.}, \tag{10}$$

where $k_0 = 2\pi f_0 / c$ is the reference wave number and (s_x, s_y) are the direction cosines of the steering vector {96–166}. Considering the architecture of the array, the phases must be built up from the equivalent time delays at the subaperture and subarray levels and the phase delays at the element levels. Based on Eq. (10), the phase step in direction w between adjacent elements in a subarray must be

$$\Delta\varphi_w = -k_0 d_w s_w. \tag{11}$$

Likewise, since each subarray contains N_x by N_y elements, adjacent subarrays within a subaperture must have a phase difference of $N_w \Delta\varphi_w$, which is equivalent to a time step

$$\begin{aligned} \Delta t_w &= \frac{1}{\omega_0} N_w \Delta\varphi_w \\ &= -\frac{1}{c} N_w d_w s_w, \end{aligned} \tag{12}$$

where $\omega_0 = 2\pi f_0$ is the reference angular frequency, and the time step across subapertures follows similarly as

$$\Delta T_w = -\frac{1}{c} N_w M_w d_w s_w. \tag{13}$$

In practical arrays, the time and phase delays are often quantized, leading to violations of Eq. (10) for general steering angles. We suppose that the beamformer is capable of mitigating the effects of subaperture and subarray quantization by adjusting the subarray and element delays. That is, the delay lost or gained in each subaperture due to quantization can be balanced by additional or reduced delay in the subarrays, assuming that the quantization interval of the subarrays is less than that of the subapertures. Likewise, the error due to subarray quantization can be balanced by adjusting the element phases, subject to a similar assumption. To exhibit this scheme mathematically, we introduce the subaperture, subarray, and element quantization intervals δT, δt, and $\delta\varphi$, respectively. We also introduce subaperture labels l_x and l_y and subarray labels m_x and m_y; as the n_w ignore subaperture and subarray boundaries, so the m_w ignore subarray boundaries. More specifically, we obtain the l_w and m_w from the n_w according to

$$\begin{aligned} l_w &= \left\lfloor \frac{n_w}{N_w M_w} \right\rfloor, \quad l_w \in \{0,1,\ldots L_w - 1\}, \quad \text{and} \\ m_w &= \left\lfloor \frac{n_w}{N_w} \right\rfloor, \qquad m_w \in \{0,1,\ldots L_w M_w - 1\}, \end{aligned} \tag{14}$$

where $\lfloor x \rfloor$ is the greatest integer not exceeding x.

We define the subaperture time delays without quantization or error to be

$$\overline{T}_{n_x n_y} = l_x \Delta T_x + l_y \Delta T_y, \tag{15}$$

where the l_w are implicitly dependent upon the n_w, and the quantized subaperture time delays are then

$$\hat{T}_{n_x n_y} = \delta T \,\text{round}(\overline{T}_{n_x n_y} / \delta T), \tag{16}$$

where round (x) is the integer nearest x. The subarray time delays contain an additional term that compensates for the subaperture quantization:

$$\begin{aligned} \bar{t}_{n_x n_y} &= m_x \Delta t_x + m_y \Delta t_y - \hat{T}_{n_x n_y}, \\ \hat{t}_{n_x n_y} &= \delta t \,\text{round}(\bar{t}_{n_x n_y} / \delta t), \end{aligned} \tag{17}$$

where the m_w are implicitly dependent upon the n_w, and the element phase delays contain two similar terms:

$$\begin{aligned} \overline{\varphi}_{n_x n_y} &= n_x \Delta\varphi_x + n_y \Delta\varphi_y - \omega_0(\hat{T}_{n_x n_y} + \hat{t}_{n_x n_y}), \\ \hat{\varphi}_{n_x n_y} &= \delta\varphi \,\text{round}(\overline{\varphi}_{n_x n_y} / \delta\varphi) \end{aligned} \tag{18}$$

In so defining the delays, we have implicitly chosen the constant in Eq. (10) to be zero. This choice implies that the lowest and leftmost components (those with $l_w = 0$, $m_w = 0$, or $n_w = 0$) have no associated delay regardless of the steering vector, whereas the highest and rightmost components (having $l_w =$

$L_w - 1$, $m_w \bmod M_w = M_w - 1$, or $n_w \bmod N_w = N_w - 1$) have delays that depend strongly on the steering vector.

In the program, quantization may be avoided by setting the quantization steps to zero. The above equations are then equivalent to

$$\begin{aligned} \bar{T}_{n_x n_y} &= \hat{T}_{n_x n_y} = l_x \Delta T_x + l_y \Delta T_y, \\ \bar{t}_{n_x n_y} &= \hat{t}_{n_x n_y} = (m_x \bmod M_x)\Delta t_x + (m_y \bmod M_y)\Delta t_y, \quad \text{and} \\ \bar{\varphi}_{n_x n_y} &= \hat{\varphi}_{n_x n_y} = (n_x \bmod N_x)\Delta \varphi_x + (n_y \bmod N_y)\Delta \varphi_y. \end{aligned} \tag{19}$$

We note that $m_w \bmod M_w$ is the index of subarray m_w within its parent subaperture, and likewise $n_w \bmod N_w$ is the index of element n_w within its parent subarray.

For each element, the net (possibly quantized) phase at the operating frequency is the sum of equivalent phase contributions from the three hierarchical levels:

$$\hat{\Phi}_{n_x n_y} = \omega(\hat{T}_{n_x n_y} + \hat{t}_{n_x n_y}) + \hat{\varphi}_{n_x n_y}, \tag{20}$$

where $\omega = 2\pi f$ is the operating angular frequency. This quantized phase assumes the place of $\Phi_{n_x n_y}$ in Eq. (8). At the reference frequency ($\omega = \omega_0$) and with no phase quantization, this construction of the phase yields the linear progression of Eq. (10).

4.2. Erroneous Excitations

We model the errors in a real antenna by applying amplitude and phase errors to each level of the antenna hierarchy {525–549}. Amplitude errors multiply the error-free amplitudes by factors of the form $1 + \rho$ where ρ is a random number, distributed normally with zero mean. Each level contributes such errors, so that the erroneous amplitude for element (n_x, n_y) is

$$|a_{n_x n_y}| = |\bar{a}_{n_x n_y}|(1 + R_{l_x l_y})(1 + r_{m_x m_y})(1 + \rho_{n_x n_y}), \tag{21}$$

where R, r, and ρ correspond to subapertures, subarrays, and elements, respectively. This model allows corresponding elements in different subarrays to contribute distinct errors, and likewise for corresponding subarrays within different subapertures. The user specifies the standard deviations σ_R, σ_r, and σ_ρ of the respective amplitude errors.

Time and phase errors add to the error-free (but possibly quantized) time and phase delays. The subapertures and subarrays contribute random time errors $\tilde{T}$ and $\tilde{t}$, respectively, and the elements contribute random phase errors $\tilde{\varphi}$, all drawn from zero-mean normal distributions. The user specifies the corresponding standard deviations σ_T, σ_t, and σ_φ. Additionally, the user may specify a deterministic time error $\tilde{\mathcal{T}}$ for each subaperture. The net equivalent phase error for element (n_x, n_y) is

$$\tilde{\Phi}_{n_x n_y} = \omega(\tilde{\mathcal{T}}_{l_x l_y} + \tilde{T}_{l_x l_y} + \tilde{t}_{m_x m_y}) + \tilde{\varphi}_{n_x n_y}, \tag{22}$$

and the total erroneous phase is the sum of the quantized and error phases:

$$\Phi_{n_x n_y} = \hat{\Phi}_{n_x n_y} + \widetilde{\Phi}_{n_x n_y} . \tag{23}$$

Finally, the erroneous complex excitations are

$$a_{n_x n_y} = |a_{n_x n_y}| \exp(-i\Phi_{n_x n_y}) . \tag{24}$$

5. RADIATION PATTERN

In standard array theory with mutual coupling ignored, the field pattern is the product of the array factor and the element factor. In ARRSTATS, the element factor is unity, corresponding to uniform hemispherical radiation, so the field pattern equals the array factor. (See the code {2595–2715} for notes on expanding ARRSTATS.) Given the complex excitations, the array factor (and field pattern) in the direction (ξ, η) is given by the two-dimensional Fourier transform {551–561}

$$g(\xi,\eta) = \sum_{n_x} \sum_{n_y} \exp[-ik(\xi d_x n_x + \eta d_y n_y)] a_{n_x n_y} , \tag{25}$$

where $k = 2\pi f / c$ is the operating wave number. ARRSTATS uses a fast Fourier transform to obtain g in discrete directions given by

$$\begin{aligned} \xi_{q_x} &= \frac{2\pi q_x}{k d_x Q_x}, \quad q_x \in \{0,1,\ldots,Q_x - 1\}, \quad \text{and} \\ \eta_{q_y} &= \frac{2\pi q_y}{k d_y Q_y}, \quad q_y \in \{0,1,\ldots,Q_y - 1\}, \end{aligned} \tag{26}$$

where Q_x and Q_y are the number of points in the transform in each dimension {168–173}. The (ξ_{q_x}, η_{q_y}) grid is extrapolated to all of visible space using {412–497, 558}

$$\begin{aligned} g(\xi_{q_x + Q_x}, \eta_{q_y}) &= g(\xi_{q_x}, \eta_{q_y}) \quad \text{and} \\ g(\xi_{q_x}, \eta_{q_y + Q_y}) &= g(\xi_{q_x}, \eta_{q_y}), \end{aligned} \tag{27}$$

which are valid for all integers q_x and q_y. Because of the normalization condition of Eq. (9),

$$|g(\xi,\eta)| \le 1 \tag{28}$$

for all ξ and η, with the equality holding only where the excitations interfere perfectly constructively. Therefore, when the field pattern is expressed in decibels, 0 dB corresponds to perfectly constructive interference.

6. PATTERN MEASURES

The code at the heart of the program analyzes the radiation pattern to obtain several measures that quantify the characteristics and performance of the array. The following subsections describe these measures, generally focusing on the concepts behind them and on their interpretation rather than on the specific method of calculation. Details of the algorithms and further discussion may be found in the code.

6.1. Pointing Vector, Pointing Error, and Peak Power Density

One of the most significant pattern measures is the direction of maximum radiation, here called the pointing vector. The program determines it directly from the field pattern and also indirectly from the excitations; the final pointing vector is a weighted average of the two, as discussed below. In identifying the pointing vector from the field pattern {563–787}, the program first locates the pattern's maximum magnitude over the grid of discrete direction cosines. For a well-formed beam, the neighboring samples should fall off parabolically, so they are fitted to the elliptic paraboloid[13]

$$|g(\xi,\eta)| = \frac{1}{2}U\xi^2 + W\xi\eta + \frac{1}{2}V\eta^2 + X\xi + Y\eta + Z \tag{29}$$

in a least-squares sense. If $UV > W^2$, as should be the case for a normal beam, the conic is indeed elliptical (corresponding to its level curves), and its maximum occurs at $(\xi, \eta) = (p_{1x}, p_{1y})$, where p_{1x} and p_{1y} satisfy

$$\begin{pmatrix} U & W \\ W & V \end{pmatrix}\begin{pmatrix} p_{1x} \\ p_{1y} \end{pmatrix} + \begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}. \tag{30}$$

The coordinate pair (p_{1x}, p_{1y}) is taken to be the pointing vector for the first method. The deviation of the pattern samples from conic form is used to construct a covariance ellipse for p_{1x} and p_{1y} that expresses the uncertainty in their values.

The second method determines the pointing vector from the complex excitations {789–961}. Because a well-behaved array will have a nearly flat phase front, the excitation phases $\Phi_{n_x n_y}$ are fit to the plane $-kn_x d_x \xi - kn_y d_y \eta - \Phi_0$, weighted according to the excitation magnitudes. The pointing vector coordinates (p_{2x}, p_{2y}) are the direction cosines (ξ, η) that give the best fit. As with the first method, a covariance ellipse for p_{2x} and p_{2y} is obtained from a measure of the deviation from the plane.

Each method is useful but limited. The first, the fit of the transform, is robust even for poorly aimed and malformed beams, but it is limited by the resolution of the Fourier transform. The second, the fit of the excitations, is independent of transform resolution but accurate only for nearly planar phase fronts, approaching the exact solution as the phase and time errors and quantization intervals decrease. To obtain a single pointing vector (p_x, p_y), the program averages pointing vectors from the two methods, weighting each by the inverse of the area of its covariance ellipse {963–990}. An average covariance ellipse is also constructed in a consistent manner.

With the final pointing vector in hand, the pointing error γ is straightforwardly obtained {992–1046} from

$$\begin{aligned} \cos\gamma &= \mathbf{s}\cdot\mathbf{p} \\ &= s_x p_x + s_y p_y + s_z p_z, \end{aligned} \tag{31}$$

where s_z and p_z follow from unit magnitude constraints on $\mathbf{s}$ and $\mathbf{p}$. If one desires the uncertainty in γ due to uncertainty in $\mathbf{p}$, an alternate calculation based on

$$\sin\gamma = |\mathbf{s}\times\mathbf{p}| \tag{32}$$

may be selected in the program.

The peak power density

$$P_{\max} = |\, g(p_x, p_y)|^2 \tag{33}$$

is calculated directly from Eq. (25).

6.2. Main Beam Region

The main beam region is the set of field pattern samples that are judged to belong to the main beam. Although normally not of interest as a final measure, it is essential for obtaining subsequent measures. It may be constructed conceptually by imagining a contour at an adjustable level. Beginning at the pattern maximum, we decrease the level so that the contour expands in size, following the topology of the main beam. Eventually the contour will intersect a local minimum or a saddle point; the closed contour about the pattern maximum at that level delineates the main beam region inside from the sidelobe region outside. Equivalently, that contour is the lowest one containing the global maximum that encircles no other local maxima. The power level of the contour is called the beam depth. Generally, well-formed beams are deep (that is, the beam depth is much less than one), while malformed beams are shallow, but the user should keep in mind that the beam depth depends on the transform resolution. In the program {1048–1137}, the beam depth is stored in the variable `beamDepthDB` in decibels relative to P_{max} and is output to the user. Information obtained while determining the main beam region is used in finding the main beam width and roll, below, and the region itself is used directly in calculating the powers radiated into the main beam and sidelobes.

6.3. Main Beam Width and Roll

The level contours of the main beam generated by a two-dimensional array are nominally ellipses;[14,15] therefore they can be described by their center location, major and minor axes, and orientation. Having already obtained a measure of the beam's location in the form of the pointing vector, we use the major and minor axes and orientation of the elliptical contour at a given power level to describe the beam's shape {1139–1403}. The conventional power level is $P_{max}/2$, or about 3 dB down. The angle subtended by the ellipse's longest diameter — its major axis — is taken as the beam's major width (that is, full width at half maximum power); that subtended by its shortest diameter, the beam's minor width. The ellipse orientation gives the roll angle, but we must choose an origin for the orientation angle. Construct three great circles as in Figure 4: *A*, along the azimuth reference; *B*, connecting boresight and the pointing vector; and *C*, along the beam's major width. The roll angle is defined as the sum of the angle between *A* and *B* and that between *B* and *C*. It happens that the roll angle so defined is merely the orientation of the major width relative to the azimuth reference when viewed in the stereographic projection, which preserves angles between great circles. Moreover, because the great circles along the beam's major and minor widths intersect orthogonally on the hemisphere, their stereographic projections do also. These facts motivate the program's use of stereographic coordinates for fitting an ellipse to the level contour and determining its major and minor axes and orientation. However, the roll angle and beam widths thereby obtained are approximate for two reasons. First, for beams off boresight, the great circles along the major and minor widths project as circles, whereas the major and minor axes of the projected ellipse are straight line segments. Second, the local scale in the projection increases away from boresight, artificially enlarging the portion of the beam farthest from boresight.[12] The error may become significant for beams far from boresight. A more sophisticated method of determining the beam widths is suggested near the end of the program code.

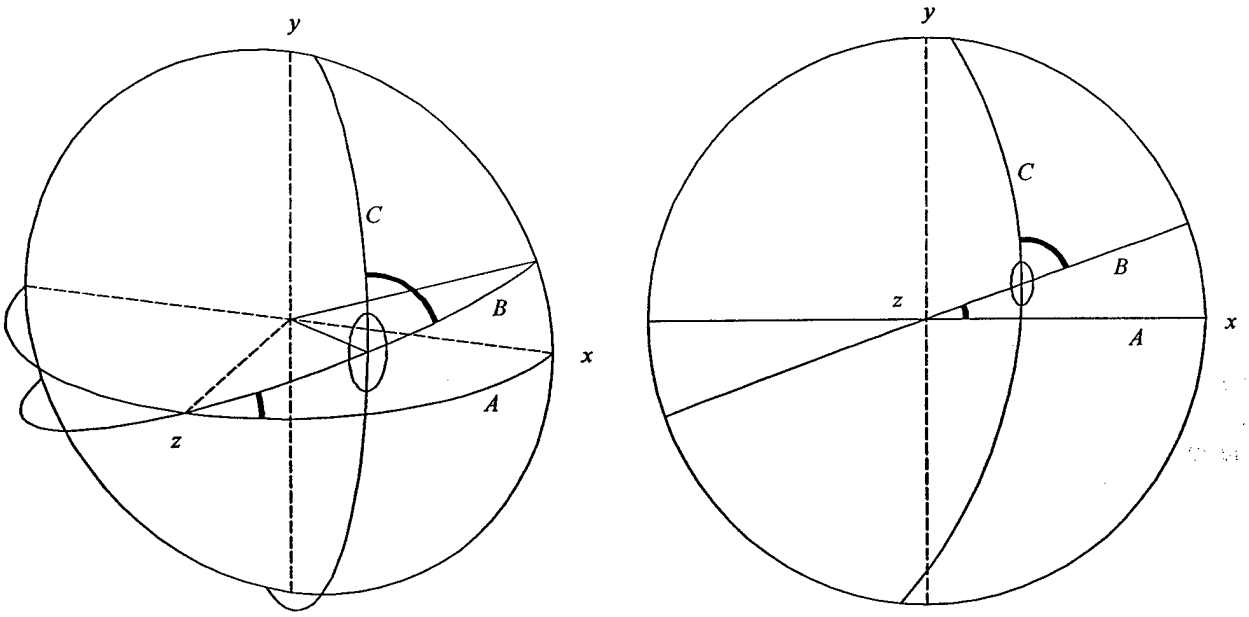

Figure 4 — Beam width ellipse and roll angle. The great circles *A*, *B*, and *C* are described in the text, and the angles between them, indicated with thick arcs, are added to obtain the roll angle. The left projection is orthographic; the right, stereographic. The spherical coordinates of the pointing vector are $\theta = 40°$ and $\psi = 20°$, the beam's major and minor widths are 16° and 8°, and the roll angle is 90°.

6.4. Directivity, Power Ratios, and Average Sidelobe Level

The next measures obtained all depend on integrals of the power pattern over solid angle regions {1405–1527}. The integrals are calculated from the discrete samples of the pattern using the midpoint approximation, as detailed in the code. The solid angle regions of interest are visible space (the half-space $z > 0$); the main beam, as described by the main beam region, above; and the sidelobes, defined as all regions of visible space not in the main beam. The program determines the total powers radiated into these regions; call them Π_v for visible space, Π_m for the main beam, and Π_s for the side lobes. We have

$$\Pi_v = \Pi_m + \Pi_s . \tag{34}$$

The directivity is the ratio of maximum to average power densities, assuming no back radiation into $z < 0$:

$$D = \frac{P_{max}}{\Pi_v / 4\pi} . \tag{35}$$

The program also calculates the power ratios Π_m / Π_v, Π_m / Π_s, and Π_v / Π_s, which generally decrease as the beam degrades. Finally, the average sidelobe level relative to the beam peak is

$$L_{avg} = \frac{\Pi_s}{\Omega_s P_{max}} , \tag{36}$$

where Ω_s is the solid angle occupied by the sidelobes.

6.5. Powers and Distances of Largest and Nearest Sidelobes

The last analysis identifies the sidelobe with the largest power density and the sidelobe closest to the main beam, and for each it reports the power density and angular distance from the main beam {1529–1677}. The power levels and locations of the sidelobes are obtained by fitting to elliptic paraboloids (Eq. (29)), and the power levels are reported relative to the beam peak P_{max}.

7. MULTIPLE REALIZATIONS AND PARAMETER VALUES

The analyses described above apply to the radiation pattern corresponding to a single set of array parameters and one realization of any random variables. ARRSTATS contains two outer loops with which it analyzes multiple radiation patterns; one is a loop over realizations of random variables, and the other loops over a user-selected independent variable. The loop over realizations {190–218, 521–523, 1679–1819, 1905} is motivated by the following: When simulating random errors, one is usually interested not in the performance obtained by one realization of the errors but rather in the performance statistics for an ensemble of statistically identical arrays. To that end, ARRSTATS can generate an arbitrary, user-specified number of realizations for which it will accumulate performance statistics. The program outputs the mean and standard deviation of each performance measure. The second loop {175–188, 326–336, 2578–2579}, over an independent variable, allows the user to examine the variation of performance measures as the variable changes. Possible independent variables include but are not limited to the operating and reference frequencies, steering angles, error parameters, quantization intervals, and even parameters of the array geometry. ARRSTATS produces a graph showing each measure as a function of the independent variable, as discussed below.

8. PROGRAM OUTPUT

ARRSTATS outputs its results in three ways: textual output of running statistics, a plot of the radiation pattern for the last realization in an ensemble, and a summary plot of the performance measures as functions of the independent variable. The textual output is a table like that in Fig. 5 printed to the MATLAB command window {1864–1903}. The values in the table are the means and standard deviations of the performance measures for all members of the statistical ensemble that have been realized thus far. The table may be interpreted according to the descriptions in Section 6, keeping in mind the following.

```
Means and [std devs] for 16 of 16 realizations
beam direction : (29.954, 60.007) deg, std dev 0.038 deg
pointing error :   0.0528 [0.0279] deg
peak power dens:  -0.426  [0.034 ] dB
beam depth     : -24.66   [1.49  ] dB re peak
beam width     : ( 2.095  [0.005 ], 1.813 [0.004]) deg
beam roll      :  59.56   [0.65  ] deg
directivity    :  38.894  [0.034 ] dB
power ratio m/v:  -1.360  [0.030 ] dB
power ratio m/s:   4.347  [0.113 ] dB
power ratio v/s:   5.707  [0.083 ] dB
avg sidelobe   : -41.61   [0.11  ] dB re peak
nrst sidelobe  : -13.16   [0.69] dB re peak, 3.00 [0.01] deg off beam
lgst sidelobe  : -12.30   [0.58] dB re peak, 3.11 [0.09] deg off beam
```

Figure 5 — Textual output of running statistics. The parameters of this example are in the code listing; for the values above, the standard deviation of the subarray time error is 10 ps (`stdTimeMPS` = 10).

The coordinates specifying the beam direction are the spherical coordinates (θ and ψ) of the average pointing vector {1821–1862}. The standard deviation of the beam direction is an estimate of the rms angular deviation of the ensemble of pointing vectors from their mean {1821–1862}. Both the beam direction coordinates and roll angle include compensation for the azimuth offset `azmthOffst`. The peak power density is relative to perfectly constructive interferenee, and the beam depth, average sidelobe level, and levels of the nearest and largest sidelobes are relative to the peak power density (suggested by the use of "dB re peak" in the table).

To aid in visualizing an array's performance, ARRSTATS can graphically present the radiation pattern and several of the performance measures as in Fig. 6 {220–324, 2138–2574}. A significant

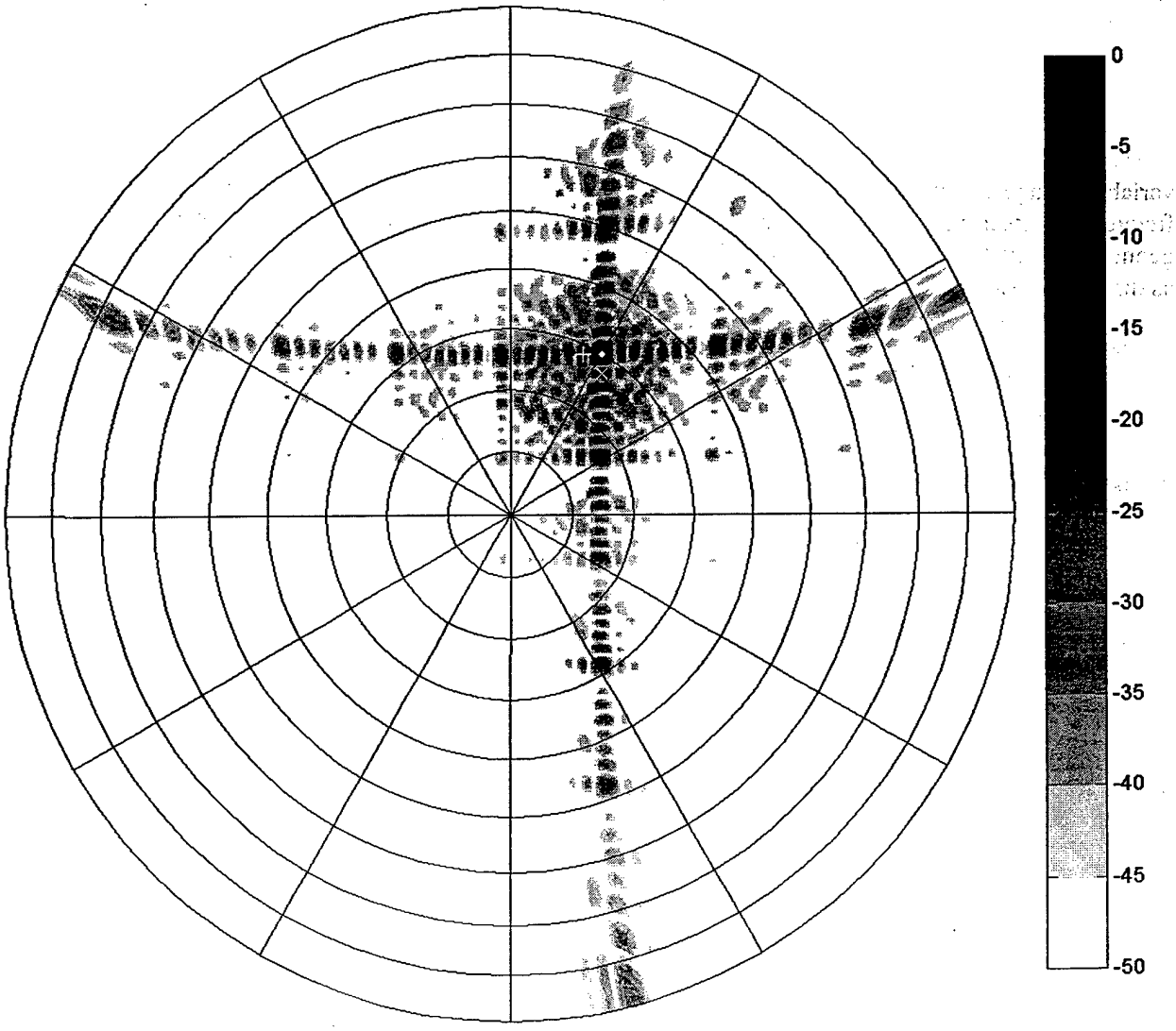

Figure 6 — Lambert projection of the radiation pattern with a spherical coordinate grid superimposed. Boresight is at the center, and the steering vector is ($\theta = 30°$, $\psi = 60°$). The white dot denotes the beam peak; ×, the largest sidelobe; and +, the nearest sidelobe. The legend gives the power in dB relative to perfectly constructive interference. The pattern is one realization of the case `stdTimeMPS` = 10 for the parameters given in the code listing.

plotting option is the choice of projection from among those described in Section 2. Briefly, the Lambert projection preserves the relative areas of regions (for example, the size of the main beam relative to the hemisphere or to prominent sidelobes), the stereographic projection preserves local shape (and the orthogonality of the major and minor beam width cuts), and the orthographic projection gives a picture of the hemisphere. The pattern may be plotted linearly or logarithmically in power, and the logarithmic depth may be specified. The user may also specify the color map and shading method to use. The performance measures that can be graphically indicated on the radiation pattern include the pointing vector, the axes of the uncertainty ellipse of the pointing vector, the actual and fitted beam width contours, the main beam region, the locations of the nearest and largest sidelobes, and other measures of less frequent interest. If multiple realizations are generated, the radiation pattern will be plotted only for the last realization as a representative of the ensemble.

If the outer loop over an independent variable is used, the summary figure plots the performance measures as functions of the independent variable, as in Fig. 7 {220–324, 1907–2136}. The figure groups fifteen measures (all except the beam direction) into eight subplots and utilizes distinct colors or line styles and both left and right axes for the ordinates. The title of each subplot gives the names of the measures; where multiple colors or linestyles appear, each measure name is followed by a color or style code in parenthesis, and where left and right axes are used, an axis code ("L" or "R") also appears. If more than one realization was generated for each value of the independent variable, error bars extend one standard deviation above and one standard deviation below each point.

9. SUMMARY

We have introduced a computer program for assessing the effects of errors in rectangular-grid planar array antennas. The most general array comprises phase-steered elements grouped into time-steered subapertures and subarrays; this includes strictly phase-steered arrays and time-steered arrays as particular cases. The time and phase delays may be quantized, and random errors may be assigned to the times, phases, and amplitudes. From simulated radiation patterns, the program obtains performance measures over statistical ensembles and as functions of a user-specified independent variable, producing textual and graphical output.

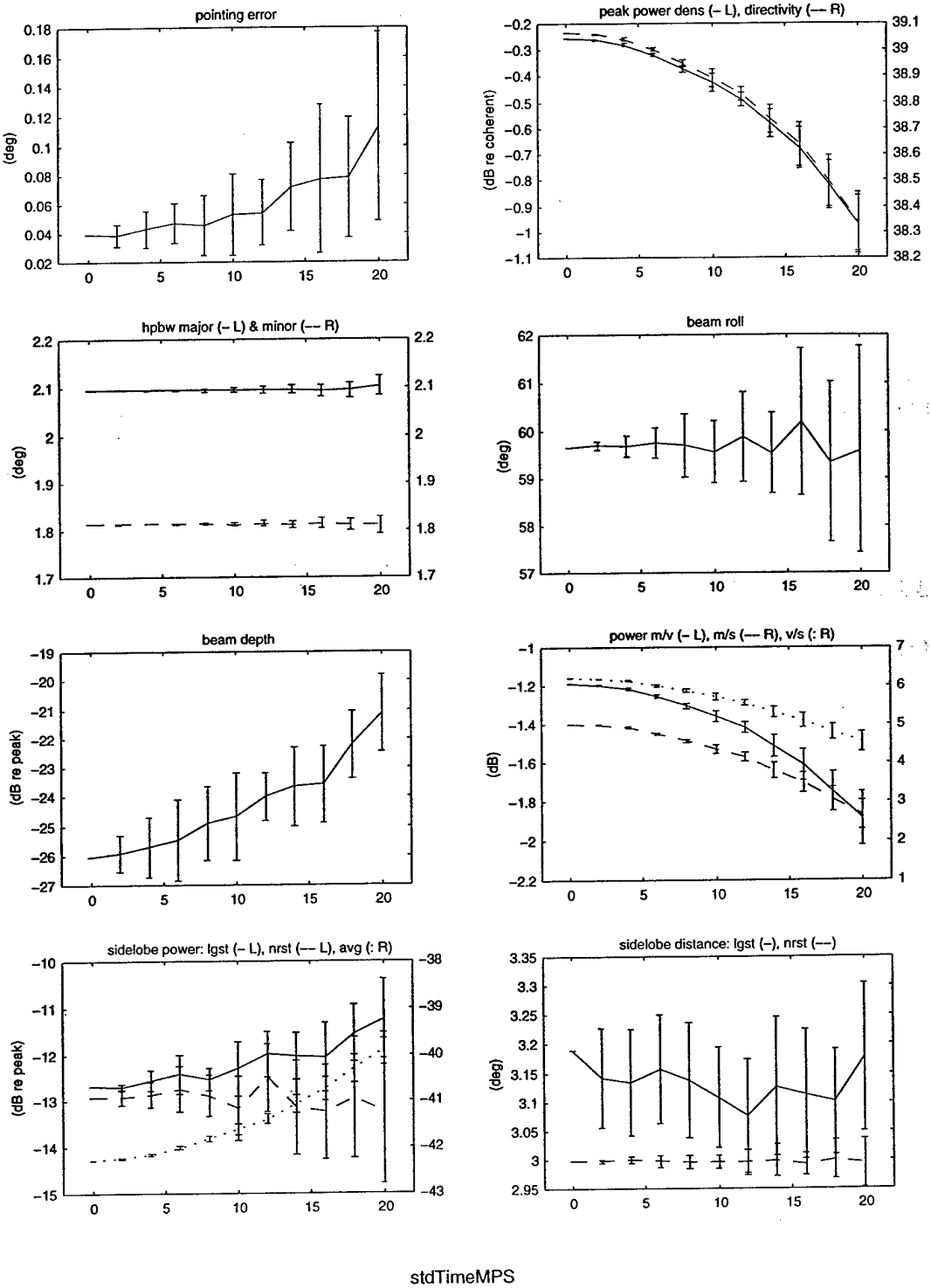

Figure 7 — Summary plots. This example shows the degradation of performance as the standard deviation of the subarray time error (`stdTimeMPS`, in picoseconds) increases. The parameters of the run may be found in the code listing.

Appendix A

LIST OF VARIABLES

This list of significant variables mentioned in this report gives their symbol in this report, coded name, and description. It does not include all program variables. Ellipses (...) stand for prefixes, and asterisks (*) indicate that the variable holds the specified quantity temporarily.

Text	Code	Description
x, y, z		coordinates in real space
ξ, η, ζ	`dirCosX, ...Y, ...Z`	direction cosines
s_x, s_y	`sx, sy`	steering vector direction cosines
θ	`...Polar`	spherical polar coordinate
ψ	`...Azmth`	spherical azimuth coordinate
α	`...Az`	traditional azimuth coordinate
ε	`...El`	traditional elevation coordinate
r		radial coordinate of projection
d_x, d_y	`dx, dy`	element spacings
L_x, L_y	`numLX, numLY`	numbers of subapertures
M_x, M_y	`numMX, numMY`	numbers of subarrays per subaperture
N_x, N_y	`numNX, numNY`	numbers of elements per subarray
l_x, l_y	`lx, ly`	subaperture labels
m_x, m_y	`mx, my`	subarray labels
n_x, n_y	`nx, ny`	element labels
	`diamond`	indicates diamond element pattern
	`azmthOffst`	antenna rotation angle about boresight
f_0	`fRef`	reference frequency
f	`fOpr`	operating frequency
k_0		reference wave number
k		operating wave number
ω_0		reference angular frequency
ω		operating angular frequency
σ_R	`stdAmplL`	standard deviation of subaperture amplitude error
σ_r	`stdAmplM`	standard deviation of subarray amplitude error
σ_ρ	`stdAmplN`	standard deviation of element amplitude error
R		random subaperture amplitude error
r		random subarray amplitude error

Text	Code	Description
ρ		random element amplitude error
$\bar{a}$	excMagIdl	error-free (ideal) excitation magnitude
a	excMag	erroneous (actual) excitation magnitude
δT	qntTimeL	subaperture time quantization interval
δt	qntTimeM	subarray time quantization interval
$\delta\varphi$	qntPhseN	element phase quantization interval
ΔT_x, ΔT_y		subaperture time steps
Δt_x, Δt_y		subarray time steps
$\Delta\varphi_x$, $\Delta\varphi_y$		element phase steps
$T_{n_x n_y}$	TimeL*	error-free subaperture time delays
$t_{n_x n_y}$	TimeM*	error-free subarray time delays
$\varphi_{n_x n_y}$	PhseN*	error-free element phase delays
$\hat{T}_{n_x n_y}$	TimeL	quantized subaperture time delays
$\hat{t}_{n_x n_y}$	TimeM	quantized subarray time delays
$\hat{\varphi}_{n_x n_y}$	PhseN	quantized element phase delays
σ_T	stdTimeL	standard deviation of subaperture time error
σ_t	stdTimeM	standard deviation of subarray time error
σ_φ	stdPhseN	standard deviation of element phase error
$\widetilde{\mathscr{T}}$	ofsTimeL	deterministic subaperture time error
$\widetilde{T}$		random subaperture time error
$\tilde{t}$		random subarray time error
$\tilde{\varphi}$		random element phase error
$\overline{\Phi}$		error-free excitation phase
$\hat{\Phi}$	excPhsIdl	quantized excitation phase
$\tilde{\Phi}$	excPhsErr	excitation phase error
Φ	excPhs	erroneous (actual) excitation phase
g	g	field pattern
Q_x, Q_y	tx, ty	number of points in Fourier transform
p_x, p_y	px, py	pointing vector direction cosines
γ	errPoint	pointing error
P_{max}	gSqrMax	maximum power density
L_{avg}	powerSideAvgDB	average relative sidelobe level (powerSideAvgDB = $10 \log_{10} L_{avg}$)
Π_v	powerVisb	power radiated into visible space
Π_m	powerMain	power radiated into the main beam
Π_s	powerSide	power radiated into the side lobes
D	directivityDB	directivity (directivityDB = $10 \log_{10} D$)

Appendix B

PROGRAM LISTING

```
% Calculate performance parameters of an array with excitation errors
%
% The excitation time and phase convention is exp (-i (omega t + phi));
% positive (negative) phases phi correspond to a leading (lagging)
% excitation.  Distances are stored in meters; times, nanoseconds;
% frequencies, gigahertz.  Angles (both geometric and phase) are always
% specified in radians.  Generally, the coordinate x increases with the
% column index; y, with the row index.
%
% Parameters that may be changed by the user are marked "INPUT" in
% comments.  Frequently-used inputs appear near the top of the program,
% but some inputs are defined elsewhere, particularly in the plotting
% section.
%
% Suggestions for improvements are provided at the end of the program.
%
% This program is coded for Matlab version 5.3 (Release 11), although
% most of the code will run under version 4.2.
%
% Written by Stan West, 1998, 1999
% U.S. Government work not subject to copyright

% Declare physical and conversion constants
%
c = 0.299792458;   % speed of light in m/ns
rpd = pi / 180;    % radians per degree
twopi = 2 * pi;

% Set operating frequency relative to reference frequency
%
% At the reference frequency, or fOpr = fRef, the time and phase delays
% will properly steer the antenna in the absence of error and
% quantization.
%
fRef =  3.0;                % INPUT reference frequency in GHz
fOpr = fRef + 1 * 0.5 / 2;  % INPUT operating frequency in GHz
% center frequency + {-1 ... 1} * bandwidth / 2

% Describe array geometry and error statistics
%
% Elements lie on a regular planar grid and are grouped heirarchically.
% The array is subdivided into numLX subapertures in the x dimension and
% numLY subapertures in the y dimension.  Each subaperture is
% time-steered and has associated with it a user-set deterministic
% absolute time error, a random absolute time error, and a random
% relative amplitude error. The random errors are normally-distributed
% with zero mean and user-set standard deviation. Each subaperture
% contains numMX by numMY subarrays, each of which, like the
% subapertures, is time-steered and has a random absolute time error and
% a random relative amplitude error.  Finally, each subarray contains
% numNX by numNY elements, each of which is phase-steered and has a
% random absolute phase error and a random relative amplitude error.
% All elements are spaced by dx in x and dy in y, even across subarrays
% and subapertures.
%
% Set parameters of subapertures
%
numLX =  1;               % INPUT number of subapertures in x
numLY =  1;               %   and y dimensions
qntTimeL = 0;             % INPUT quantization interval in ns; 0 for no quantization
ofsTimeL = zeros (numLY, numLX);  % INPUT deterministic absolute time error in each subaperture in ns
stdTimeL = 0.000;         % INPUT standard deviation of absolute time error in each subaperture in ns
stdAmplL = 0.0;           % INPUT standard deviation of relative amplitude error
%
% Set parameters of subarrays
%
numMX =  8;               % INPUT number of subarrays in x
numMY =  8;               %   and y dimensions per subaperture
qntTimeM = 0;             % INPUT quantization interval in ns; 0 for no quantization
stdTimeM = 0.00;          % INPUT standard deviation of absolute time error in each subarray in ns
stdAmplM = 0.0;           % INPUT standard deviation of relative amplitude error
%
% Set parameters of elements
%
dx = 1.6 * 0.0254;        % INPUT element spacing in x
```

```
dy = 1.6 * 0.0254;          %   and y dimensions in meters
numNX =  8;                 % INPUT number of elements in x
numNY =  8;                 %   and y dimensions per subarray
qntPhseN =  0 * rpd;        % INPUT quantization interval in radians; 0 for no quantization
stdPhseN =  0 * rpd;        % INPUT standard deviation of absolute phase error in each element in radians
stdAmplN = 0.0;             % INPUT standard deviation of relative amplitude error
%
% Set other parameters
%
diamond = 0;                % INPUT 0: full array; 1: eliminate excitations on even diagonals
azmthOffst =   0 * rpd;     % INPUT angle of the positive x axis of the array above the azimuth reference

% Alternatively, select an array
%
switch 0  % INPUT case number for arrays below or 0 to use values above
  case 1
    % Insert frequency, array, and error parameters here
  case 2
end

% Specify steering angle
%
% Two coordinate systems, described below, are available for specifying
% the steering angle: traditional azimuth/elevation coordinates and
% spherical coordinates.  Azimuth/elevation coordinates are converted to
% spherical coordinates for internal program use, and output is given in
% spherical coordinates.
%
% In the diagrams below, the antenna lies in the x'-y' plane with
% boresight along the positive z' axis.  The antenna's z axis coincides
% with z', and its x axis is at an angle azmthOffst above the x' axis,
% which is the azimuth reference.
%
% The equations relating the the Cartesian coordinates x', y', and z' of
% a unit vector, the spherical coordinates Polar and Azmth, and the
% traditional coordinates Az and El are
%
%      x' = sin Polar cos Azmth = -cos El sin Az
%      y' = sin Polar sin Azmth =  sin El
%      z' = cos Polar           =  cos El cos Az
%
%      cos Polar =  cos El cos Az    = z'
%      tan Azmth = -tan El / sin Az = y' / x'
%
%      -tan Az = tan Polar cos Azmth = x' / z'
%       sin El = sin Polar sin Azmth = y'
%
% Since (x', y', z') is a unit vector, its components are direction
% cosines.
%
switch 2
  case 1
    %                     y'          Use azimuth/elevation
    %                     |           coordinates.  Given the
    %                   _--|          projection of the steering vector
    %                  /   |          onto the x'-z' plane, the azimuth
    %        main __  /    |          is the angle between it and the
    %        beam   --__   |          z' axis, while the elevation is
    %              /     --o--------  the angle between it and the
    %           El |  _- /        x'  steering vector.  In the diagram,
    %              |_-  /            both angles are positive.
    %             _--__/
    % proj. in _-     /
    % xz plane    Az  z'
    %
    steerAz =  30 * rpd;            % INPUT azimuth angle
    steerEl =  30 * rpd;            % INPUT elevation angle
    steerPolar = acos  (cos (steerEl) * cos (steerAz));
    steerAzmth = atan2 (tan (steerEl), -sin (steerAz));
    %
  case 2
    %         proj. in    y'        Use spherical coordinates.  The
    %         xy plane    |         polar angle is the angle
    %                \_--|--_       between the z' axis and the
    %                 \  |   \ Azmth steering vector.  The azimuth
    %        main __   /\ |    \     angle is the angle between the
    %        beam   --/_ \|    |     x' axis and the projection of
    %               | | --o--------  the steering vector onto the
    %               | |  /        x' x'-y' plane.  In the diagram,
    %          Polar \ \/            both angles are positive.
    %                 \/
    %                 /
    %                z'
    %
    steerPolar =   30 * rpd;        % INPUT polar angle
    steerAzmth =   60 * rpd;        % INPUT azimuth angle
    %
  otherwise
    error ('Invalid switch parameter.');
end
steerAzmth = steerAzmth - azmthOffst;  % now relative to antenna's x axis

```

```
% Set transform size
%
% (See comments elsewhere related to the discrete Fourier transform.)
%
tx = 2^9;  % INPUT number of transform points in x
ty = 2^9;  %   and y

% Declare independent variable and its values
%
% The main loop iterates over values of the independent variable named
% below.  The independent variable may be any parameter, including, for
% example, those describing geometry, frequency, error, quantization,
% and steering angle.  It may also be an otherwise unknown variable that
% is transformed to a known program parameter by custom code in the main
% loop.  To effectively disable the loop, set a dummy variable to a
% scalar value.
%
indVarName = 'stdTimeMPS';          % INPUT name of independent variable
indVar = 0 : 2 : 20;                % INPUT vector of values it will assume
indVar = indVar (:);                % make it a column vector
indVarLen = length (indVar);

% Initialize statistics variables
%
% If the excitations are random, one is often interested in the mean and
% standard deviation of the performance measures.  For each value of the
% independent variable, the program will generate numRlz realizations of
% the random excitations and accumulate the statistics of the
% performance measures.
%
numRlz =  16;                                   % INPUT number of realizations to generate
numAcc           = nan * ones (indVarLen, 1);   % number of realizations accumulated
pxS              = nan * ones (indVarLen, 2);
pyS              = nan * ones (indVarLen, 2);
pzS              = nan * ones (indVarLen, 2);
errPointS        = nan * ones (indVarLen, 2);
% errPointUncS     = nan * ones (indVarLen, 2);  % see later calculation of pointing error
beamPowerDBS     = nan * ones (indVarLen, 2);
beamDepthDBS     = nan * ones (indVarLen, 2);
hpbwMjrS         = nan * ones (indVarLen, 2);
hpbwMnrS         = nan * ones (indVarLen, 2);
rollS            = nan * ones (indVarLen, 2);
directivityDBS   = nan * ones (indVarLen, 2);
powerMainVisbDBS = nan * ones (indVarLen, 2);
powerMainSideDBS = nan * ones (indVarLen, 2);
powerVisbSideDBS = nan * ones (indVarLen, 2);
powerSideAvgDBS  = nan * ones (indVarLen, 2);
slNrstDistS      = nan * ones (indVarLen, 2);
slNrstPowrDBS    = nan * ones (indVarLen, 2);
slLgstDistS      = nan * ones (indVarLen, 2);
slLgstPowrDBS    = nan * ones (indVarLen, 2);

% Initialize graphics
%
cmap = jet;       % INPUT color map
invertBkgd = 0;   % INPUT 0: figure background is lowest value of the colormap;
                  %       1: "                        highest
sumBW = 0;        % INPUT 0: summary plot in color; 1: in black and white
figPat = 1;       % INPUT figure number for power pattern
figSum = 2;       %   and summary
cbarVert = 0;     % INPUT 0 for horizontal bar below pattern, 1 for vertical to right
%
% Set window positions
%
cbarSize = 0.15;  % colorbar size relative to pattern
marginWid = 8;    % width margin in pixels
marginHgt = 44;   % height margin in pixels
screenSize = get (0, 'screenSize');
screenWid = screenSize (3);
screenHgt = screenSize (4);
figPatWid = 0.45 * (screenSize (3) - 4 * marginWid);     % width of pattern figure
figSumWid = screenSize (3) - 4 * marginWid - figPatWid;  % width of summary figure
if cbarVert
                                                         % height of pattern figure
  figPatHgt = figPatWid;
  figPatWid = figPatWid * (1 + cbarSize);
else
  figPatHgt = figPatWid * (1 + cbarSize);
end
figure (figPat);                    % create and position pattern figure
figPat = gcf;                       % update in case figPat couldn't be created
set (figPat, ...
  'position', [marginWid                          ...
               screenHgt-marginHgt-figPatHgt ...
               figPatWid                          ...
               figPatHgt], ...
  'paperUnits', 'inches')
if cbarVert
  set (figPat, ...
    'PaperOrientation', 'landscape', ...
    'PaperPosition', [0.5 0.5 10 7.5]);
else
  set (figPat, ...
```

```
260         'PaperOrientation', 'portrait', ...
261         'PaperPosition', [0.5 0.5 7.5 10]);
262     end
263     clf;
264     figure (figSum);                    % create and position summary figure
265     figSum = gcf;                       % update in case figSum couldn't be created
266     set (figSum, ...
267       'position', [screenWid-marginWid-figSumWid ...
268                    marginHgt                     ...
269                    figSumWid                     ...
270                    screenHgt-2*marginHgt]);
271     orient tall;
272     clf;
273     %
274     % Set pattern figure colors
275     %
276     if invertBkgd
277       bkgd = cmap (size (cmap, 1), :);  % take background from high...
278     else
279       bkgd = cmap (1, :);               %   or low end of colormap
280     end
281     whitebg (figPat, bkgd);             % set pattern colors
282     set (figPat, 'color', bkgd);        % override default background of whitebg
283     %
284     % Set summary figure colors and linestyles
285     %
286     whitebg (figSum, 'w');              % set summary colors
287     set (figSum, 'color', 'w');         % override default background of whitebg
288     if sumBW
289       discrim = 'lineStyle';
290       discrimValue = {'-'; '--'; ':'};  % line styles
291       discrimName = discrimValue;       % names for legends will be same as style codes
292       set (figSum, ...                  % black color forces cycling through line styles
293         'DefaultAxesLineStyleOrder', discrimValue, ...
294         'DefaultAxesColorOrder', [0 0 0]);
295     else
296       discrim = 'color';
297       set (figSum, ...                                          % reset (if previously b&w, for
298         'defaultAxesColorOrder', 'default', ...                 %   example)
299         'defaultAxesLineStyleOrder', 'default');
300       discrimValue = get (figSum, 'defaultAxesColorOrder');     % put colors in array of RGB coordinates
301       colorOrderHSV = rgb2hsv (discrimValue);                   % equivalent HSV coordinates
302       discrimValue = num2cell (discrimValue, 2);                % put each row in a cell
303       colorNames = ['RYGCBMKW'];                                % first six by hue, then black & white
304       discrimName = colorNames (1 + ...                         % convert H to an integer 0..5
305         mod (round (6 * colorOrderHSV (:, 1)), 6));             %   then index into colorNames
306       unsat = find (colorOrderHSV (:, 2) <= 0.25);              % unsaturated (gray) colors
307       if ~isempty (unsat)
308         discrimName (unsat) = colorNames (7 + ...               % threshold V to 0 (K) or 1 (W)
309           (colorOrderHSV (unsat, 3) > 0.5));                    %   then index into colorNames
310       end
311       discrimName = cellstr (discrimName);                      % put each letter in a cell
312       clear colorOrderHSV colorNames unsat;
313     end
314     %
315     % Set miscellaneous common properties
316     %
317     set ([figPat figSum], ...
318       'invertHardCopy', 'off', ...
319       'defaultTextFontSize', 8, ...
320       'defaultAxesFontSize', 8, ...
321       'toolbar', 'none');
322     drawnow;
323     clear marginWid marginHgt screenSize screenWid screenHgt;
324     clear figPatWid figSumWid figPatHgt bkgd;
325
326     % Loop over independent variable
327     %
328     for indx = 1 : indVarLen
329
330       % Set value of independent variable
331       %
332       eval ([indVarName ' = indVar (indx);']);  % set variable to value
333       %
334       % Code may be inserted below to transform the independent variable to
335       % known program variables.
336       stdTimeM = stdTimeMPS * 1e-3;
337
338       % Define labels for substructures
339       %
340       % Here we construct row and column vectors (corresponding to x and y,
341       % respectively) that tell to which subaperture, subarray, or element a
342       % given position corresponds.  The sample vectors are for numLX =
343       % numLY = 2, numMX = numMY = 3, and numNX = numNY = 2.
344       %
345       numLMX = numLX * numMX;             % total number of subarrays
346       numLMY = numLY * numMY;
347       numLMNX = numLMX * numNX;           % total number of elements
348       numLMNY = numLMY * numNY;
349       nx =  0 : numLMNX - 1;              % e.g., [0 1 2 3 4 5 6 7 8 9 10 11]
350       ny = (0 : numLMNY - 1)';            %       [0 1 2 3 4 5 6 7 8 9 10 11]'
351       mx = floor (nx / numNX);            %       [0 0 1 1 2 2 3 3 4 4  5  5]
```

```
352     my = floor (ny / numNY);            %       [0 0 1 1 2 2 3 3 4 4  5  5]'
353     lx = floor (nx / (numMX * numNX));  %       [0 0 0 0 0 0 1 1 1 1  1  1]
354     ly = floor (ny / (numMY * numNY));  %       [0 0 0 0 0 0 1 1 1 1  1  1]'
355
356     % Calculate ideal (error-free) excitation magnitudes
357     %
358     excMagIdl = ones (numLMNY, numLMNX);            % uniform weighting
359     if diamond                                      % zero every other element
360       excMagIdl = excMagIdl ...
361         .* rem (ones (numLMNY, 1) * nx + ny * ones (1, numLMNX), 2);
362     end
363     excMagIdl = excMagIdl / sum (excMagIdl(:).^2);  % normalize to unit power
364
365     % Calculate ideal (error-free) excitation phases
366     %
367     % The ideal excitation phases are those that produce perfectly
368     % constructive interference in the direction of the steering vector at
369     % the reference frequency.  This implies a linear phase progression
370     %
371     %       excPhsIdl = -k0 (dx sx nx + dy sy ny) + const.,
372     %
373     % where k0 (= 2 pi fRef) is the reference wave vector.  Quantization
374     % prevents the array from achieving this flat phase front for all
375     % steering angles.  However, the beamformer simulated here mitigates
376     % effects due to subaperture quantization by adjusting the subarray
377     % delays, which is effective if the subarrays are quantized at a finer
378     % interval than the subapertures.  Likewise, it compensates for
379     % subarray quantization with the element phasers.
380     %
381     % We choose the constant in the phase progression to be zero, which
382     % means that the lowest and leftmost components (those for which lx,
383     % ly, mx, my, nx, or ny is zero) are never delayed, while the highest
384     % and rightmost components have delays that depend strongly on the
385     % steering vector.
386     %
387     sx = sin (steerPolar) * cos (steerAzmth);       % direction cosines
388     sy = sin (steerPolar) * sin (steerAzmth);       %   for steering
389     sz = cos (steerPolar);                          % used much later
390     TimeLX = -dx * sx * numNX * numMX * lx / c;
391     TimeLY = -dy * sy * numNY * numMY * ly / c;
392     TimeL = ones (numLMNY, 1) * TimeLX + TimeLY * ones (1, numLMNX);
393     if qntTimeL ~= 0                                % quantize?
394       TimeL = round (TimeL / qntTimeL) * qntTimeL;  % yes
395     end;
396     TimeMX = -dx * sx * numNX * mx / c;
397     TimeMY = -dy * sy * numNY * my / c;
398     TimeM = ones (numLMNY, 1) * TimeMX + TimeMY * ones (1, numLMNX) - TimeL;
399     if qntTimeM ~= 0
400       TimeM = round (TimeM / qntTimeM) * qntTimeM;
401     end;
402     TimeNX = -dx * sx * nx / c;
403     TimeNY = -dy * sy * ny / c;
404     PhseN = twopi * fRef * ...
405       (ones (numLMNY, 1) * TimeNX + TimeNY * ones (1, numLMNX) - TimeL - TimeM);
406     if qntPhseN ~= 0
407       PhseN = round (PhseN / qntPhseN) * qntPhseN;
408     end;
409     excPhsIdl = twopi * fOpr * (TimeL + TimeM) + PhseN;
410     clear TimeLX TimeLY TimeL TimeMX TimeMY TimeM TimeNX TimeNY TimeN PhseN;
411
412     % Prepare transform mapping
413     %
414     % The far-field array factor is the Fourier transform of the complex
415     % excitations.  Considering the x dimension only (the operation in the
416     % y dimension is analogous), the discrete Fourier transform (DFT) used
417     % later calculates the far-field array factor at the direction cosine
418     % cx as
419     %
420     %                 tx-1
421     %       g (cx ) = sum  exp (-i k cx dx q) e ,
422     %                 q=0                     q
423     %
424     % where the e[q] are the complex excitations (zero-padded if tx
425     % exceeds the array size), k (= 2 pi / lambda) is the operating wave
426     % vector, and lambda (= c / fOpr) is the operating wavelength.  The
427     % argument of the exponential in the transform may be written -i 2 pi
428     % (cx / lambda) * (dx q), where cx / lambda is the spatial frequency
429     % and dx q is the spatial coordinate.  The direction cosines for which
430     % g is calculated in the DFT are
431     %
432     %             lambda p
433     %       cx  = -------- , p = 0, 1, ..., tx - 1
434     %         p     dx tx
435     %
436     % so that the argument of the exponential is -i 2 pi p q / tx.  We
437     % have
438     %
439     %       g (cx    ) = g (cx )  for any integer p.
440     %            p+tx         p
441     %
442     % Given g (cx[p]), the array factor may be obtained at any angle using
443     %
```

```
%               tx-1   / p    dx cx       \
%      g (cx) = sum  S | -- - ------ , tx | g (cx )
%               p=0    \ tx   lambda     /      p
%
% where the geometric progression
%
%                  1  tx-1
%      S (x, tx) = -- sum  exp (i 2 pi q x)
%                  tx q=0
%
%                  1  exp (i 2 pi x tx) - 1
%                = -- ----------------------
%                  tx   exp (i 2 pi x) - 1
%
% is an interpolating function, but this formula is not used below.
% Incidentally, note that
%
%                    1  | sin (tx pi x) |
%      |S (x, tx)| = -- | ------------- | ,
%                    tx |  sin (pi x)   |
%
% an expression that often appears in array theory.
%
% After the DFT is calculated, the program maps the results into the
% region of cx-cy space where the direction cosines have magnitude 1
% or less, tiling as necessary to fill the region.  The portion of
% that region for which cx^2 + cy^2 <= 1 corresponds to visible real
% space (radiating waves).  Later processing requires a border of at
% least one element outside the visible region. This section prepares
% the mapping.
%
txLim = tx * dx * fOpr / c;                 % values of p (not necessarily
tyLim = ty * dy * fOpr / c;                 %   integer) for which cx and cy are 1
txIndxLimMin = -floor (txLim      ) - 1;    % largest integers p for which
tyIndxLimMin = -floor (tyLim      ) - 1;    %   cx, cy > -1
txIndxLimMax =  floor (txLim + 0.5) + 1;    % smallest integers p for which cx,
tyIndxLimMax =  floor (tyLim + 0.5) + 1;    %   cy > (1 + half element spacing)
  % Extra half element spacing is needed only for surface plots with
  % flat shading; see graphics code below
ax = txIndxLimMax - txIndxLimMin + 1;       % number of angle samples in x
ay = tyIndxLimMax - tyIndxLimMin + 1;       %   and y
txIndx =  txIndxLimMin : txIndxLimMax  ;    % row vector of indices
tyIndx = (tyIndxLimMin : tyIndxLimMax)';    % column vector
dirCosX = txIndx / txLim;                   % corresponding direction cosines
dirCosY = tyIndx / tyLim;                   %   cx and cy
dirCosShiftX = (txIndx - 0.5) / txLim;      % shifted direction cosines for use
dirCosShiftY = (tyIndx - 0.5) / tyLim;      %   with flat shading
txIndx = txIndx - tx * floor (txIndx / tx); % zero-based indices into columns
tyIndx = tyIndx - ty * floor (tyIndx / ty); %   (x) and rows (y) of DFT results
  % The above lines accomplish the tiling function by folding the
  % indices into the interval [0, tx - 1]
tIndx = tyIndx * ones (1, ax) ...           % indices to elements (one-based,
  + ones (ay, 1) * txIndx * ty + 1;         %   column-ordered)
clear txIndxLimMin tyIndxLimMin txIndxLimMax tyIndxLimMax txIndx tyIndx;

% Prepare far-field angle mapping
%
% Physically, the array factor is a function of position on a
% hemisphere.  The direction cosines used in the Fourier transform are
% the x and y coordinates of points on the unit hemisphere.  Below we
% determine the region of the transform results that corresponds to
% visible space, namely cx^2 + cy^2 <= 1, and calculate the z
% coordinates of points in visible space according to
%
%                  2     2 1/2
%      cz = (1 - cx  - cy )   , Re cz >= 0.
%
% For points outside visible space, cz is set to zero.
%
dirCosXMtx = ones (ay, 1) * dirCosX;                % now a matrix
dirCosYMtx = dirCosY * ones (1, ax);
radSqr = dirCosXMtx.^2 + dirCosYMtx.^2;             % squared radius
visBool = logical (radSqr < 1);                     % 1 in visible space, 0 elsewhere
dirCosZMtx = zeros (ay, ax);                        % 0 outside visible space
dirCosZMtx (visBool) = sqrt (1 - radSqr (visBool)); % positive in visible space
clear radSqr;

% Loop over realizations
%
for rlzNum = 1 : numRlz

  % Calculate excitation magnitudes with error
  %
  excMagErr = 1 + stdAmplN * randn (numLMNY, numLMNX);  % element-level error
  err       = 1 + stdAmplM * randn (numLMY , numLMX );  % temporary matrix
  excMagErr = excMagErr .* err (my + 1, mx + 1);        % subarray-level error
  err       = 1 + stdAmplL * randn (numLY  , numLX  );  % temporary
  excMagErr = excMagErr .* err (ly + 1, lx + 1);        % subaperture-level error
  excMag    = excMagErr .* excMagIdl;                   % actual (with error) magnitude
  clear err excMagErr;

  % Calculate excitation phases with error
```

```
536         %
537         excPhsErr = stdPhseN * randn (numLMNY, numLMNX);      % element-level error
538         err       = stdTimeM * randn (numLMY , numLMX ) ...   % temporary matrix
539           * twopi * fOpr;                                     %   (equivalent phase)
540         excPhsErr = excPhsErr + err (my + 1, mx + 1);         % subarray-level error
541         err       = (ofsTimeL + stdTimeL * randn (numLY, numLX)) * twopi * fOpr;
542         excPhsErr = excPhsErr + err (ly + 1, lx + 1);         % subaperture-level error
543         excPhs    = excPhsErr + excPhsIdl;                    % actual (with error) phases
544         clear err excPhsErr;
545
546         % Assemble complex excitations
547         %
548         exc = excMag .* exp (-i * excPhs);
549         % clear excMag excPhs;
550
551         % Calculate the field pattern (array factor)
552         %
553         % Here the DFT is calculated and the result rearranged into the
554         % desired region of direction cosine space.  The element factor is
555         % unity.  Mutual coupling is ignored.
556         %
557         g = fft2 (exc, ty, tx);       % first element is zero frequency
558         g = g (tIndx);                % rearrange
559         gSqr = real (conj (g) .* g);  % squared magnitude
560         gMag = sqrt (gSqr);           % magnitude
561         clear g;
562
563         % Determine actual beam direction by fitting the transform
564         %
565         % This first of two methods for calculating the beam pointing vector
566         % uses the information in the Fourier transform of the excitations.
567         % First we locate the element of g with the largest value.  (If the
568         % maximum value of elements is obtained by more than one element,
569         % this code will use the one with the smallest column index and the
570         % smallest row index within that column.  Later processing will
571         % determine whether the multiple maxima are all within the main
572         % beam.)  To estimate the location of the maximum of the underlying
573         % continuous function, that element and its eight neighbors are
574         % fitted to the elliptic paraboloid [1]
575         %
576         %      1    2             1    2
577         %      - U x  + W x y + - V y  + X x + Y y + Z = |g (x, y)|
578         %      2                2
579         %
580         % (in the direction cosine coordinate system) in a least-squares
581         % sense.  We employ the technique of QR decomposition to find the
582         % least-squares solution.  Define the solution vector
583         %
584         %                        T
585         %      a = [U W V X Y Z]
586         %
587         % of length M = 6, the ordinate (column) vector b with elements
588         %
589         %      b  = |g (x , y )|
590         %       i         i   i
591         %
592         % of length N = 9 (number of fitted points), and the design matrix A
593         % having rows
594         %
595         %                1  2          1  2
596         %      A   = [ - x   x y   - y   x   y   1 ] ,
597         %       i,:     2  i   i i  2  i   i   i
598         %
599         % where the (x[i], y[i]) are the coordinates of the points
600         % neighboring and including the maximum element.  QR decomposition
601         % of A factorizes it as
602         %
603         %      A = Q R ,
604         %
605         % where Q is unitary (Q' Q = eye) and R is upper triangular.  (We
606         % use Matlab's economy size decomposition, for which Q is M-by-N and
607         % R is N-by-N.)  The least-squares solution of
608         %
609         %      A a = b
610         %
611         % is given by
612         %
613         %           -1
614         %      a = R   Q' b .
615         %
616         % The error in the solution depends on the degree to which the
617         % values of |g| depart from parabolic form, which is indicated by
618         % the reduced chi-square
619         %
620         %         2    1     2
621         %      chi   = -- chi ,
622         %        nu    nu
623         %
624         % where nu = N - M is the number of degrees of freedom and
625         %
```

```
626         %         2
627         %      chi = (A a - b)' (A a - b) .
628         %
629         % The covariance matrix for the solution vector is normally given by
630         % the matrix inverse of the curvature matrix
631         %
632         %      alpha = A' A ,
633         %
634         % whose elements are the second partial derivatives of chi^2 with
635         % respect to the elements of the solution vector [2]:
636         %
637         %                1   d      d       2
638         %      alpha   = - ------ ------ chi  .
639         %           mn   2 d a[m] d a[n]
640         %
641         % To incorporate the degree of deviation from parabolic form, we
642         % scale the covariance matrix by the reduced chi-square, as
643         %
644         %                  2        -1      2        -1
645         %      C     = chi   (A' A)   = chi   (R' R)  .
646         %       poly      nu               nu
647         %
648         % This completes the least-squares procedure.
649         %
650         % Having obtained the coefficients of the best-fit polynomial, we
651         % locate its maximum.  The coordinates (plx, ply) of the maximum
652         % solve
653         %
654         %      / U W \ / plx \   / X \   / 0 \
655         %      |     | |     | + |   | = |   | ;
656         %      \ W V / \ ply /   \ Y /   \ 0 /
657         %
658         % that is,
659         %
660         %      / plx \   1 /  V -W \ / X \
661         %      |     | = - |       | | |   | ,
662         %      \ ply /   D \ -W  U / \ Y /
663         %
664         % where
665         %
666         %           2
667         %      D = W  - U V
668         %
669         % is the discriminant of the polynomial (and the negative
670         % determinant of the matrix).  If the maximum so found lies outside
671         % the interpolation region, the region is expanded by one sample in
672         % each direction and the least-squares fit is repeated.  This loop
673         % continues until the interpolation region exceeds a certain size or
674         % a satisfactory maximum is found.  Assuming a maximum has been
675         % found, the covariance matrix for plx and ply is calculated next.
676         % We first form the derivative matrix or Jacobian
677         %
678         %           d (plx, ply)
679         %      J  = ------------
680         %       p       d (a)
681         %
682         %          / d plx d plx d plx d plx d plx d plx \
683         %          | ----- ----- ----- ----- ----- ----- |
684         %          |  d U   d W   d V   d X   d Y   d Z  |
685         %        = |                                     | .
686         %          | d ply d ply d ply d ply d ply d ply |
687         %          | ----- ----- ----- ----- ----- ----- |
688         %          \  d U   d W   d V   d X   d Y   d Z  /
689         %
690         % The plx derivatives are
691         %
692         %      d plx   V X - W Y
693         %      ----- = --------- V
694         %       d U        D^2
695         %
696         %      d plx   W^2 Y + U V Y - 2 W V Y
697         %      ----- = ------------------------
698         %       d W                D^2
699         %
700         %      d plx   W X - U Y
701         %      ----- = --------- W
702         %       d V        D^2
703         %
704         %      d plx   V
705         %      ----- = -
706         %       d X    D
707         %
708         %      d plx     W
709         %      ----- = - - ,
710         %       d Y      D
711         %
712         % and the ply derivatives follow by exchanging U for V and X for Y
713         % everywhere.  The covariance matrix for plx and ply is simply
714         %
715         %                    T
716         %      C  = J  C     J  .
717         %       p    p  poly  p
```

```
718       %
719       % If no satisfactory maximum was found by the above procedure, the
720       % location of the maximum of |g| is used.  A covariance matrix is
721       % fabricated for which the area of the 2 sigma ellipse equals the
722       % area of four grid squares to indicate the uncertainty in the
723       % actual location of the maximum.
724       %
725       % [1] D. H. von Seggern, _CRC Standard Curves and Surfaces_.  Boca
726       % Raton, FL: CRC, 1993.
727       %
728       % [2] P. R. Bevington and D. K. Robinson, _Data Reduction and Error
729       % Analysis for the Physical Sciences_, 2nd ed.  New York, NY:
730       % McGraw-Hill, 1992, pp. 121-125.
731       %
732       [gSqrMaxAct, gSqrMaxRow] = max (gSqr .* visBool);     % maximum values and their rows
733       [gSqrMaxAct, gSqrMaxCol] = max (gSqrMaxAct);          % overall maximum and column
734       gSqrMaxRow = gSqrMaxRow (gSqrMaxCol);                 % corresponding row
735       intRad = 1;                                           % interpolation radius
736       p1OK = 0;
737       while ~p1OK & intRad < 4
738         p1Nbrs = [-intRad : intRad];
739         p1x = dirCosXMtx (gSqrMaxRow + p1Nbrs, gSqrMaxCol + p1Nbrs);  % neighbors
740         p1y = dirCosYMtx (gSqrMaxRow + p1Nbrs, gSqrMaxCol + p1Nbrs);
741         p1z = gMag       (gSqrMaxRow + p1Nbrs, gSqrMaxCol + p1Nbrs);
742         p1x = p1x (:);
743         p1y = p1y (:);
744         p1z = p1z (:);
745         pDesMtx = [p1x.*p1x/2 p1x.*p1y p1y.*p1y/2 ...       % design matrix
746           p1x p1y ones(size(p1x))];
747         [Q, R] = qr (pDesMtx, 0);                           % now pDesMtx = Q * R; Q' * Q = eye
748         pPoly = R \ (Q' * p1z);                             % solves pDesMtx * pPoly = p1z
749         dof = length (p1z) - length (pPoly);                % degrees of freedom in the fit
750         chiSqr = p1z - pDesMtx * pPoly;                     % deviations only
751         chiSqr = chiSqr' * chiSqr;                          % now sum of squared deviations
752         pPolyVar = inv (R' * R) * chiSqr / dof;             % pDesMtx' * pDesMtx = R' * R
753         pVec = -[pPoly(1) pPoly(2); pPoly(2) pPoly(3)] ...  % find critical point
754           \ [pPoly(4); pPoly(5)];
755         p1OK = ...                                          % is interpolated point
756             (pVec (1) > dirCosX (gSqrMaxCol - intRad)) ...  %   inside neighborhood?
757           & (pVec (1) < dirCosX (gSqrMaxCol + intRad)) ...  %   (pathological cases can
758           & (pVec (2) > dirCosY (gSqrMaxRow - intRad)) ...  %   place it outside)
759           & (pVec (2) < dirCosY (gSqrMaxRow + intRad));
760         if ~p1OK
761           intRad = intRad + 1;
762         end
763       end
764       if p1OK
765         p1x = pVec (1);                                     % keep interpolated point
766         p1y = pVec (2);
767         pDet = pPoly (2)^2 - pPoly (1) * pPoly (3);         % determinant
768         p1xDer = [[1 -1]*(pPoly([3 2]).*pPoly([4 5]))*pPoly(3)/pDet
769                   [1 1 -2]*(pPoly([2 1 2]).*pPoly([2 3 3]).*pPoly([5 5 4]))/pDet
770                   [1 -1]*(pPoly([2 1]).*pPoly([4 5]))*pPoly(2)/pDet
771                   pPoly(3)
772                   -pPoly(2)
773                   0 ]' / pDet;                              % derivatives
774         p1yDer = [[1 -1]*(pPoly([2 3]).*pPoly([5 4]))*pPoly(2)/pDet
775                   [1 1 -2]*(pPoly([2 3 2]).*pPoly([2 1 1]).*pPoly([4 4 5]))/pDet
776                   [1 -1]*(pPoly([1 2]).*pPoly([5 4]))*pPoly(1)/pDet
777                   -pPoly(2)
778                   pPoly(1)
779                   0 ]' / pDet;
780         p1Var = [p1xDer; p1yDer] * pPolyVar * [p1xDer; p1yDer]';  % covariance matrix
781       else                                                  % interpolated point is outside
782         p1x = dirCosXMtx (gSqrMaxRow, gSqrMaxCol);          % use location of
783         p1y = dirCosYMtx (gSqrMaxRow, gSqrMaxCol);          %   actual maximum
784         p1Var = diag (1 ./ (pi * [txLim tyLim].^2));        % 2 sigma area = 4 grid squares
785       end
786       clear intRad p1OK p1Nbrs p1z pDesMtx Q R pPoly;
787       clear dof chiSqr pPolyVar pVec pDet p1xDer p1yDer;
788
789       % Determine actual beam direction by fitting the excitation phases
790       %
791       % The second method for calculating the pointing vector uses the
792       % excitation magnitudes and phases, not the transform.  For brevity,
793       % define
794       %
795       %       Delta  (x, y) = k n dx x + k m dy y + theta    ,
796       %            mn                                  mn
797       %
798       % where the theta[m,n] are the excitation phases, and let e[m,n]
799       % denote the excitation magnitudes.  When we express the power
800       % pattern as
801       %
802       %                2
803       %       |g (x, y)|  = sum sum sum sum e       e
804       %                       m1  n1  m2  n2  m1,n1  m2,n2
805       %
806       %                     * exp [i (Delta       - Delta      )]
807       %                                 m1,n1         m2,n2
808       %
```

```
809         %                                2
810         %                 = sum  e    + 2  sum   sum' e      e
811         %                   m,n   m,n      m1,n1 m2,n2  m1,n1  m2,n2
812         %
813         %                   * cos (Delta      - Delta     ) ,
814         %                              m1,n1        m2,n2
815         %
816         % where the primed sum is over distinct pairs (m1, n1) and (m2, n2),
817         % we see that the maximum occurs where the cosine contributions are
818         % largest.  If the phase front is nearly flat, the arguments of the
819         % cosines will be small for (x, y) near the direction of phase front
820         % propagation.  To fourth order in the arguments,
821         %
822         %                2           2
823         %     |g (x, y)|  = (sum e  )  + 2  sum   sum  e      e
824         %                     m,n  mn       m1,n1 m2,n2  m1,n1  m2,n2
825         %
826         %                    /   1                              2
827         %                   | | - - (Delta      - Delta      )
828         %                    \   2        m1,n1        m2,n2
829         %
830         %                        1                             4 \
831         %                     + -- (Delta      - Delta      )  | .
832         %                       24        m1,n1        m2,n2   /
833         %
834         % Keeping terms to only second order, we are motivated to find the x
835         % and y (implicit in Delta[mn]) that minimize
836         %
837         %        2                                                    2
838         %     chi  =  sum   sum  e      e      (Delta      - Delta     )  ,
839         %        2   m1,n1 m2,n2  m1,n1  m2,n2       m1,n1        m2,n2
840         %
841         % where the subscript 2 denotes the second-order truncation.  Taking
842         % derivatives with respect to x and y and rearranging yields the
843         % normal equations
844         %
845         %     / 0 \                          / n1-n2 \
846         %     |   | =  sum   sum  e      e    |       |
847         %     \ 0 /   m1,n1 m2,n2  m1,n1  m2,n2 \ m1-m2 /
848         %
849         %               /                                          \
850         %             * | [n1-n2 m1-m2] Pi + theta      - theta     | ,
851         %               \                        m1,n1        m2,n2 /
852         %
853         % where Pi = k [dx*x dy*y]'.  Here the sums contain a total of
854         % (numLMNX^2 numLMNY^2) terms, which may be of the order of one
855         % million for a typical array.  To reduce this number, we transform
856         % the least-squares problem to an equivalent but simpler problem.
857         % First, the above normal equations may be rewritten with the column
858         % vector [n1-n2 m1-m2]' replaced by [n1 m1]', which may be seen by
859         % separating the column vector into two sums and exchanging (m1,n1)
860         % with (m2,n2) in one of the sums.  Second, the row vector may be
861         % separated into two sums to obtain
862         %
863         %     / 0 \                        / n \
864         %     |   | = ( sum e  ) sum e     |   | ( [n m] Pi + theta  )
865         %     \ 0 /     m,n  mn  m,n  m,n \ m /                    mn
866         %
867         %               /          / n \ \
868         %             - | sum e    |   | | sum e   ( [n m] Pi + theta  ) .
869         %               \ m,n  mn  \ m / / m,n  mn                    mn
870         %
871         % Based on the second term, we define
872         %
873         %                sum e   ( [n m] Pi + theta  )
874         %                m,n  mn                    mn
875         %     Delta = - ---------------------------- .
876         %                          sum e
877         %                          m,n  mn
878         %
879         % Finally, we may rewrite the normal equations as
880         %
881         %     / 0 \            / n \
882         %     | 0 | = sum e    | m | ( [n m 1] Gamma + theta  ) ,
883         %     \ 0 /   m,n  m,n \ 1 /                         mn
884         %
885         % where Gamma = [k*dx*x k*dy*y Delta]' and the third row follows
886         % from the definition of Delta.  These normal equations contain only
887         % (numLMNX numLMNY) terms, nominally on the order of 1000.  They
888         % find the plane  k n dx x + k m dy y + Delta  that best fits
889         % -theta[m,n] in a weighted least-squares sense.  The solution is
890         % found using QR decomposition of the design matrix, which has rows
891         % [n m 1].
892         %
893         % The error in the solution is determined not by the deviations of
894         % the -theta[mn] from the best-fit plane nor by the deviations
895         % Delta[m1,n1] - Delta[m2,n2] appearing in chi^2 earlier, for the
896         % solution is exactly the power pattern maximum to second order in
897         % the cosine arguments.  However, the error does depend on the
898         % fourth and higher powers of the cosine arguments.  So motivated,
899         % we consider the fourth-order merit function
900         %
```

```
901         %         2                                                  2
902         %      chi  =  sum   sum  e      e      (Delta      - Delta     )
903         %         4   m1,n1 m2,n2  m1,n1  m2,n2       m1,n1        m2,n2
904         %
905         %               /      1                           2 \
906         %              | | 1 - -- (Delta      - Delta     )  |
907         %               \      12       m1,n1        m2,n2   /
908         %
909         % and observe that chi4^2 =< chi2^2 everywhere.  We interpret the
910         % difference chi2^2 - chi4^2 as indicative of the error in the
911         % solution, and we scale the covariance matrix by that amount.  The
912         % covariance matrix used is the inverse of the curvature matrix for
913         % chi2^2; that curvature matrix is
914         %
915         %               2
916         %      alpha = k  sum   sum  e      e
917         %                 m1,n1 m2,n2  m1,n1  m2,n2
918         %
919         %                / (n1-n2)dx \
920         %               | |           | [ (n1-n2)dx (m1-m2)dy ] .
921         %                \ (m1-m2)dy /
922         %
923         % Because we construct the design matrix for the simpler
924         % least-squares problem, we must construct alpha explicitly.
925         % However, this can be accomplished by analytically expanding the
926         % differences and factoring the sums.
927         %
928         phsX = ones (numLMNY, 1) * nx * dx * (twopi * fOpr / c);
929         phsY = ny * ones (1, numLMNX) * dy * (twopi * fOpr / c);
930         phsX = phsX (:);
931         phsY = phsY (:);
932         excMagV = excMag (:);
933         desMtx = [phsX phsY ones(numLMNX*numLMNY,1)] ...
934           .* (sqrt (excMagV) * ones (1, 3));
935         [Q, R] = qr (desMtx, 0);  % now desMtx = Q * R and Q' * Q = eye [conj. transpose]
936         excPhsWgt = -excPhs (:) .* sqrt (excMagV);
937         p2Vec = R \ (Q' * excPhsWgt);  % desMtx * p2Vec = excPhsWgt
938         p2x = p2Vec (1);
939         p2y = p2Vec (2);
940         DeltaPhs = [phsX phsY ones(numLMNX*numLMNY,1)] * p2Vec + excPhs (:);
941         sumExcMagDelta1 = excMagV .* DeltaPhs;
942         sumExcMagDelta2 = sumExcMagDelta1 .* DeltaPhs;
943         sumExcMagDelta3 = sumExcMagDelta2 .* DeltaPhs;
944         sumExcMagDelta4 = sumExcMagDelta3 .* DeltaPhs;
945         sumExcMagDelta0 = sum (excMagV);
946         sumExcMagDelta1 = sum (sumExcMagDelta1);
947         sumExcMagDelta2 = sum (sumExcMagDelta2);
948         sumExcMagDelta3 = sum (sumExcMagDelta3);
949         sumExcMagDelta4 = sum (sumExcMagDelta4);
950         chiSqrRed = (  2 * sumExcMagDelta4 * sumExcMagDelta0 ...
951                      - 8 * sumExcMagDelta3 * sumExcMagDelta1 ...
952                      + 6 * sumExcMagDelta2 * sumExcMagDelta2 ) / 12;
953         chiSqrRed = max (0, chiSqrRed);  % in case of roundoff error
954         excMagPhs = excMagV' * [phsX phsY];
955         crvMtx = 2 * sumExcMagDelta0 * [phsX phsY]' ...
956             * ([phsX phsY] .* [excMagV excMagV]) ...
957           - 2 * excMagPhs' * excMagPhs;
958         p2Var = chiSqrRed * inv (crvMtx);
959         clear phsX phsY excMagV desMtx Q R excPhsWgt p2Vec DeltaPhs;
960         clear sumExcMagDelta0 sumExcMagDelta1 sumExcMagDelta2 sumExcMagDelta3 sumExcMagDelta4;
961         clear chiSqrRed excMagPhs crvMtx;
962
963         % Construct pointing vector
964         %
965         % Above we constructed two pointing vectors by different methods.
966         % The method of fitting the transform is robust even for large
967         % errors but limited by the transform resolution.  On the other
968         % hand, the method of fitting the excitation phases is independent
969         % of transform resolution but accurate only for small errors,
970         % approaching the exact solution as the phase errors decrease.  We
971         % wish to obtain a single pointing vector for subsequent use, and
972         % for this purpose we form a weighted average.  Specifically, each
973         % vector is weighted by the inverse of the area of its covariance
974         % ellipse, which is pi times the determinant of the covariance
975         % matrix.  Similarly, a single covariance matrix is obtained by
976         % weighing each covariance matrix by the square of the pointing
977         % vector weights, normalized to avoid effectively halving the
978         % covariance matrix when the two incoming matrices are nearly equal.
979         %
980         area1 = det (p1Var);
981         area2 = det (p2Var);
982         areaTot = area1 + area2;
983         wght1 = area2 / areaTot;
984         wght2 = area1 / areaTot;
985         px = wght1 * p1x + wght2 * p2x;
986         py = wght1 * p1y + wght2 * p2y;
987         pVar = (wght1^2 * p1Var + wght2^2 * p2Var) / (wght2^2 + wght1^2);
988         clear area1 area2 areaTot wght1 wght2;
989         clear p1x p1y p1Var;
990         clear p2x p2y p2Var;
991
992         % Calculate peak power density and pointing error
```

```
993         %
994         % To avoid inaccuracies due to interpolation, the peak power
995         % density is obtained by explicitly evaluating the Fourier
996         % transform at the pointing vector.  The pointing error is
997         % straightforwardly calculated from the dot product of the steering
998         % and pointing vectors.  However, an optional second method is
999         % coded that makes use of the pointing vector covariance matrix to
1000        % calculate the uncertainty (standard deviation) of the pointing
1001        % error due to uncertainty in the pointing vector.  If this
1002        % uncertainty is desired, also uncomment lines elsewhere that refer
1003        % to errPointUnc and errPointUncS.
1004        %
1005        pxy2 = px^2 + py^2;
1006        peakVisb = (pxy2 <= 1);
1007        if peakVisb
1008          gSqrMax = exp (-i * twopi * (fOpr / c) ...          % complex field
1009            * (  px * dx * ones (numLMNY, 1) * nx ...
1010               + py * dy * ny * ones (1, numLMNX))) .* exc;
1011          gSqrMax = sum (gSqrMax (:));                        % phaser sum
1012          gSqrMax = real (conj (gSqrMax) * gSqrMax);          % power
1013          pz = sqrt (1 - pxy2);
1014          if 0                                                % INPUT 0 to skip std dev, 1 to calc
1015            pCrs = [ 0  -sz  sy                               % cross product
1016                     sz  0  -sx
1017                    -sy  sx  0 ] * [px py pz]';
1018            pCrsDer = [  -sy*px/pz -sz-sy*py/pz               % rows correspond to
1019                       sz+sx*px/pz     sx*py/pz               %   components of pCrs;
1020                        -sy            sx         ];          %   columns, to pVec
1021            pCrsVar = pCrsDer * pVar * pCrsDer';              % covariance matrix
1022            pCrsMag = sqrt (pCrs' * pCrs);                    % magnitude
1023            if pCrsMag > 0
1024              pCrsMagDer = pCrs' / pCrsMag;                   % derivative exists
1025              pCrsMagVar = pCrsMagDer * pCrsVar * pCrsMagDer';
1026            else                                              % derivative doesn't exist
1027              pCrsMagVar = trace (pCrsVar) / 3;               % average of principal variances
1028            end
1029            errPoint = asin (pCrsMag);
1030            errPointUnc = abs (1 / sqrt (1 - pCrsMag^2)) * sqrt (pCrsMagVar);
1031            clear pCrs pCrsDer pCrsVar pCrsMag pCrsMagDer pCrsMagVar;
1032          else
1033            errPoint = acos (min (1, ...                      % dot product for error; min
1034              sx * px + sy * py + sz * pz));                  %   prevents roundoff problems
1035            errPointUnc = nan;
1036          end
1037        else                                                  % maximum is invisible
1038          gSqrMax = gSqrMaxAct;
1039          px = nan;
1040          py = nan;
1041          pz = nan;
1042          errPoint = nan;
1043          errPointUnc = nan;
1044        end
1045        beamPowerDB = 10 * log10 (gSqrMax);
1046        clear pxy2;
1047
1048        % Determine main beam region
1049        %
1050        % The angular domain of the main beam is constructed starting with
1051        % the maximum element.  The largest neighboring element is added on,
1052        % followed by the largest neighbor of either point, and so on.  This
1053        % accretion continues until any neighbor of the largest element on
1054        % the main beam border exceeds the element added previously.
1055        % Effectively, elements are added with values descending from the
1056        % peak until an opportunity to ascend is reached. All visible
1057        % elements outside of the main beam are declared to be in the
1058        % sidelobes.
1059        %
1060        beamWidLvl = gSqrMax / 2;                             % power level where beam width is measured
1061        adjc = [-ay-1 -ay -ay+1 -1 1 ay-1 ay ay+1];           % relative indices of neighbors
1062          % Note: We must include the diagonal neighbors in order to
1063          % correctly descend a structure such as [1 0.4; 0.5 0.9].
1064        beamBool = logical (zeros (ay, ax));                  % build main beam in Boolean variable
1065        brdrLen = 1;
1066        brdrIndx = (gSqrMaxCol - 1) * ay + gSqrMaxRow;        % start with maximum
1067        brdrVal = 0;                                          % any value will do here
1068        beamBool (brdrIndx) = 1;                              % main beam begins with maximum
1069        adjcIndx = brdrIndx + adjc';                          %   and neighbors
1070        adjcIndx = adjcIndx (visBool (adjcIndx));             %   that are visible
1071        adjcVal = gSqr (adjcIndx);                            % get values
1072        beamDepth = inf;                                      % get the loop started
1073        beamVisb = 1;
1074        capVisb = (gSqrMaxAct > beamWidLvl);                  % usually true unless resolution is too low
1075        while max (adjcVal) <= beamDepth                      % are the new neighbors all downhill?
1076          brdrLen = brdrLen - 1;                              % yes; remove element from border
1077          brdrIndx = brdrIndx (1 : brdrLen);
1078          brdrVal  = brdrVal  (1 : brdrLen);
1079          beamBool (adjcIndx) = ones (size (adjcIndx));       % add neighbors to main beam
1080          for adjcPtr = 1 : length (adjcIndx)                 %   and to border
1081            pos = sum (brdrVal <= adjcVal (adjcPtr));         % ordered least to greatest
1082            brdrIndx = [brdrIndx(1:pos) adjcIndx(adjcPtr) brdrIndx(pos+1:brdrLen)];
1083            brdrVal  = [brdrVal(1:pos)  adjcVal(adjcPtr)  brdrVal(pos+1:brdrLen) ];
1084            brdrLen = brdrLen + 1;
```

```
1085          end
1086          beamDepth = brdrVal (brdrLen);                  % pick largest element from border
1087          adjcIndx = brdrIndx (brdrLen) + adjc;           % neighbors of chosen element
1088          adjcIndx = adjcIndx (visBool (adjcIndx));       % eliminate invisible points
1089          if length (adjcIndx) < length (adjc)            % were some invisible?
1090            beamVisb = 0;                                 % yes; clear flag
1091            if beamDepth >= beamWidLvl                    % are we below the threshold?
1092              capVisb = 0;                                % no; the cap is partially invisible
1093            end
1094          end
1095          adjcIndx = adjcIndx (~beamBool (adjcIndx));     % use only new elements
1096          adjcVal = gSqr (adjcIndx);                      % and get their values
1097        end
1098        capClosed = capVisb & (beamDepth < beamWidLvl);   % closed contour at beamWidLvl?
1099        sideBool = visBool & ~beamBool;                   % sidelobe region
1100        beamIndx = find (beamBool);
1101        if max (max (gSqr (sideBool))) < gSqrMaxAct       % duplicate maximum outside beam?
1102          beamExist = 1;                                  % no; the beam is identified
1103          if ~peakVisb
1104            disp ('Warning: The main beam peak is invisible; some calculations');
1105            disp ('  may return NaN.');
1106          elseif ~capVisb
1107            disp ('Warning: The beam width contour of the main beam is partially');
1108            disp ('  invisible; some calculations may return NaN.');
1109          elseif ~capClosed
1110            disp ('Warning: The main beam is insufficiently deep for obtaining');
1111            disp ('  its width; some calculations may return NaN.');
1112          elseif ~beamVisb
1113            disp ('Warning: The main beam is partially invisible; some');
1114            disp ('  calculations may return NaN.');
1115          end
1116        else
1117          beamExist = 0;                                  % yes; the beam is ambiguous
1118          disp ('Warning: The main beam is not identifiable; some ');
1119          disp ('  calculations will return NaN.');
1120          px = nan;                                       % strike earlier results
1121          py = nan;
1122          pz = nan;
1123          errPoint = nan;
1124          beamVisb = 0;
1125          beamIndx = [];
1126          peakVisb = 0;
1127          capVisb = 0;
1128          capClosed = 0;
1129          beamDepth = nan;
1130          sideBool = visBool;                             % treat visible space as sidelobes
1131        end
1132        if beamDepth == 0
1133          beamDepthDB = -inf;                             % avoid warning message
1134        else
1135          beamDepthDB = 10 * log10 (beamDepth / gSqrMax);
1136        end
1137        clear adjc beamBool brdrLen brdrIndx brdrVal adjcIndx adjcVal beamDepth adjcPtr pos;
1138
1139        % Determine main beam width and roll
1140        %
1141        % The analysis of the beam's width and roll is conducted using a
1142        % stereographic projection, for which projections of great circles
1143        % intersect at the same angles as the great circles on a sphere.
1144        % (See the comments in the plotting section below for details.)
1145        % This property allows us to obtain, in the limit of a narrow beam,
1146        % the correct roll angle and the beam widths along two orthogonal
1147        % great circles.
1148        %
1149        % The actual calculations are based on fitting the half-power contour
1150        % of the main beam to an ellipse.  First the contour is obtained in
1151        % direction cosine space, then the coordinates are transformed to
1152        % stereographic coordinates.  The contour is fitted to the conic
1153        % section
1154        %
1155        %      1    2           1    2
1156        %     - U x  + W x y + - V y  + X x + Y y + Z = 0
1157        %     2               2
1158        %
1159        % using a simple algorithm that minimizes the algebraic distance as
1160        % follows.  Define the design matrix D to have rows
1161        %
1162        %                1  2            1  2
1163        %     D   = [ ( - x  ) (x y ) ( - y  )  x    y   1] ,
1164        %      i,:       2  i     i  i    2  i     i   i
1165        %
1166        % where the (x[i], y[i]) are the points along the contour, and let the
1167        % coefficient vector be
1168        %
1169        %                       T
1170        %     a = [U W V X Y Z] .
1171        %
1172        % The algebraic distance between a point and a conic section is the
1173        % left-hand side of the conic section equation, so that the distance
1174        % between a point i along the contour and the ellipse described by the
1175        % vector a is simply D[i,:]a.  We seek the minimum of the sum of
1176        % squared algebraic distances, which is just ||D a||^2, subject to the
```

```
% constraint ||a||^2 = 1.  We therefore introduce the constrained
% objective function
%
%                  2
%      E = ||D a||  - lambda (||a|| - 1)
%
%           T T              T
%        = a D D a - lambda (a a - 1)
%
% where lambda is a Lagrange multiplier.  The minimum is found
% analytically to occur when
%
%       T
%      D D a = lambda a ,
%
% which is an eigenvalue equation.  The desired coefficient vector, a,
% corresponds to the minimum eigenvalue.
%
% Using the coefficients of the best-fit ellipse, we now calculate
% the beam characteristics.  First, a sign change is applied to the
% coefficients if necessary to force U (and therefore V) to be
% negative.  For convenience, we rewrite the conic section as
%
%      1  T       T
%      - p  A p + B  p + Z = 0 ,
%      2
%
% where p = [x; y],
%
%          / U W \
%      A = |     | ,
%          \ W V /
%
% and B = [X; Y].  We first find the center of the ellipse in order
% to draw it later.  Replacing p with p + p1 in the conic section
% yields
%
%      1   T          T      T
%      - p1  A p1 + (p  A + B ) p1 + Z1 = 0 ,
%      2
%
% where
%
%           1  T       T
%      Z1 = - p  A p + B  p + Z
%           2
%
% is defined for later use.  When p coincides with the center, the
% linear term vanishes; therefore, p solves
%
%      A p + b = 0 .
%
% We next find the roll angle, which is conceptually defined as
% follows, using spherical, not stereographic, coordinates.  If the
% ellipse center is not at boresight, rotate it (and the antenna
% pattern) to boresight along the great circle connecting the two.
% The angle from the great circle with azimuth 0 to the great circle
% along the beam's major axis (direction of maximum width) is the
% roll angle. Alternatively, construct the great circle connecting
% the ellipse center and boresight.  The roll angle is the sum of
% two angles, the angle from the great circle with azimuth 0 to the
% constructed great circle and the angle from the constructed great
% circle to the great circle along the beam's major axis.  Now the
% roll angle so defined is merely the apparent orientation of the
% major axis when viewed in the stereographic projection.  In a
% (stereographic) coordinate system rotated by that angle, the
% off-diagonal element of A (the coefficient W) vanishes; therefore,
% we seek the coordinate system that diagonalizes A.  Replacing p
% with R p2 in the original conic section gives
%
%      1   T   T             T
%      - p2  (R  A R) p2 + (B  R) p2 + Z = 0 .
%      2
%
% The new quadratic coefficient R^T A R will be diagonal if the
% columns of R are the eigenvectors of A.  The new diagonal elements
% U2 and V2 become the eigenvalues, which are explicitly
%
%           U + V   /  2  / U - V \2 \1/2
%      U2 = ----- + | W + | ----- |  |    and
%             2     \     \   2   /  /
%
%           U + V   /  2  / U - V \2 \1/2
%      V2 = ----- - | W + | ----- |  |
%             2     \     \   2   /  /
%
% The eigenvalue with the smaller magnitude (U2 above, since U and V
% are negative) corresponds to the major axis.  Therefore, the
% corresponding eigenvector points along the direction of the major
% axis; the other eigenvector, along the minor axis.  The roll angle
% is obtained from the two components of the major axis; explicitly,
% it satisfies
```

```
1269        %
1270        %                      2 W
1271        %      tan (2 roll) = ----- .
1272        %                     U - V
1273        %
1274        % We also use the eigenvectors to draw the ellipse later.  Last, the
1275        % eigenvalues yield the major and minor full widths at half maximum
1276        % of the main beam as
1277        %
1278        %     /    Z1 \1/2       /    Z1 \1/2
1279        %     | -8 -- |     and  | -8 -- |   ,
1280        %     \    U2 /          \    V2 /
1281        %
1282        % respectively.  As these were derived in the stereographic
1283        % projection, a factor of (1 + cos polar) is applied to obtain the
1284        % approximate widths in real angles.
1285        %
1286        if capClosed
1287          %
1288          % Construct the contour
1289          %
1290          capIndx = beamIndx (gSqr (beamIndx) >= beamWidLvl);  % elements at or above level
1291          capBool = logical (zeros (ay, ax));
1292          capBool (capIndx) = ones (size (capIndx));
1293          adjc = [ay ay+1 1 -ay+1 -ay -ay-1 -1 ay-1];  % clockwise in matrix row-column coordinates
1294          dirIndx = 1;                                 % initial index into adjc
1295          intIndx = capIndx (1);                       % initial index of interpolation center
1296          dirIndxSt = 0;                               % get loop started
1297          intIndxSt = 0;
1298          capContX = [];                               % empty contour coordinates
1299          capContY = [];
1300          while capBool (intIndx + adjc (dirIndx))     % next element is inside cap?
1301            intIndx = intIndx + adjc (dirIndx);        % keep moving until edge is reached
1302          end
1303          while (intIndx ~= intIndxSt) | (dirIndx ~= dirIndxSt)  % back at starting point?
1304            adjcInc = abs (adjc (dirIndx));                      % no; get magnitude
1305            if (adjcInc == 1) | (adjcInc == ay)                  % looking across row or column?
1306              if adjcInc == ay                                   % across column?
1307                capContY1 = dirCosYMtx (intIndx);                % yes; interpolate in x
1308                capContX1 = dirCosXMtx (intIndx) + (beamWidLvl - gSqr (intIndx)) ...
1309                  * (dirCosXMtx (intIndx + adjc (dirIndx)) - dirCosXMtx (intIndx)) ...
1310                  / (gSqr       (intIndx + adjc (dirIndx)) - gSqr       (intIndx));
1311              else                                               % across row
1312                capContX1 = dirCosXMtx (intIndx);                % interpolate in y
1313                capContY1 = dirCosYMtx (intIndx) + (beamWidLvl - gSqr (intIndx)) ...
1314                  * (dirCosYMtx (intIndx + adjc (dirIndx)) - dirCosYMtx (intIndx)) ...
1315                  / (gSqr       (intIndx + adjc (dirIndx)) - gSqr       (intIndx));
1316              end
1317              if isempty (capContX)                              % first point?
1318                capContX = capContX1;                            % yes; store it
1319                capContY = capContY1;
1320                intIndxSt = intIndx;                             % remember starting point
1321                dirIndxSt = dirIndx;
1322              elseif (capContX (length (capContX)) ~= capContX1) ...  % duplicate? (e.g.,
1323                   & (capContY (length (capContY)) ~= capContY1)      %   element equals level)
1324                capContX = [capContX; capContX1];                % no; append it
1325                capContY = [capContY; capContY1];
1326              end
1327            end                                                  % (no diagonal interpolation)
1328            dirIndx = dirIndx + 1;                               % next direction
1329            if dirIndx > length (adjc)                           % cycle
1330              dirIndx = 1;
1331            end
1332            adjcInc = abs (adjc (dirIndx));                      % no; get magnitude of step
1333            if capBool (intIndx + adjc (dirIndx))                % next element is inside cap?
1334              intIndx = intIndx + adjc (dirIndx);                % yes; becomes new interpolation center
1335              dirIndx = dirIndx - length (adjc) / 2 + 1;         % reverse, then ahead one increment
1336              if (adjcInc == 1) | (adjcInc == ay)                % stepped in row or column?
1337                dirIndx = dirIndx + 1;                           % yes; ahead an extra increment
1338              end                                                %   (useless to look back diagonally)
1339              if dirIndx < 1                                     % cycle
1340                dirIndx = dirIndx + length (adjc);
1341              end
1342            end
1343          end  % while                                           % contour complete
1344          %
1345          % Fit an ellipse in stereographic coordinates
1346          %
1347          capContZ = sqrt (1 - capContX.^2 - capContY.^2);
1348          capContXS = capContX ./ (1 + capContZ);                    % to stereographic coords
1349          capContYS = capContY ./ (1 + capContZ);
1350          cDesMtx = [capContXS.*capContXS/2 capContXS.*capContYS ...  % to form design matrix
1351            capContYS.*capContYS/2 capContXS ...                      %   for least-squares fit
1352            capContYS ones(size(capContXS))];
1353          cDesMtx = cDesMtx' * cDesMtx;                              % done
1354          capContX = capContX ([1:length(capContX) 1]);              % close contour for plotting
1355          capContY = capContY ([1:length(capContY) 1]);
1356          capContZ = capContZ ([1:length(capContZ) 1]);
1357          [cEigVec, cEigVal] = eig (cDesMtx);                        % eigenvectors and -values
1358          [cEigValMin, cEigValMinIdx] = min (diag (cEigVal));        % minimum eigenvalue
1359          cPoly = cEigVec (:, cEigValMinIdx);                        %   and matching vector
1360          cPoly = -cPoly * sign (cPoly (1));                         % to have negative eigenvalues below
```

```
1361          %
1362          % Obtain beam characteristics from ellipse coefficients
1363          %
1364          cHessMtx = [cPoly(1) cPoly(2)                                  % Hessian matrix;
1365                      cPoly(2) cPoly(3)];                                %   second derivatives
1366          cDervMtx = [cPoly(4)                                           % first derivative matrix
1367                      cPoly(5)];
1368          capCenter = -cHessMtx \ cDervMtx;                              % ellipse center
1369          capConst = cPoly (6) + cDervMtx' * capCenter / 2;              % new constant coefficient
1370          [cEigVec, cEigVal] = eig (cHessMtx);
1371          [cEigVal, cEigValOrd] = sort (diag (cEigVal));                 % ascending order
1372          cEigVec = cEigVec (:, cEigValOrd);                             % corresponding order
1373          if cEigVec (1, 2) == 0                                         % special case?
1374            roll = pi / 2;
1375          else
1376            roll = atan (cEigVec (2, 2) / cEigVec (1, 2));               % angle to major axis
1377          end
1378          roll = roll - pi/2 + azmthOffst;                               % make -pi/2 <
1379          roll = roll - ceil (roll / pi) * pi + pi/2 - azmthOffst;       %   roll + azmthOffst <= pi/2
1380          hpbw = sqrt (-8 * capConst ./ cEigVal);                        % two-vector
1381          %
1382          % Construct the fitted ellipse for plotting (in stereographic coordinates)
1383          %
1384          theta = (0 : 10 / 360 : 1) * twopi;
1385          cFit = capCenter * ones (size (theta)) ...  % calculate some
1386            + 0.5 * (cEigVec * diag (hpbw) ...        %   points along the
1387                  * [sin(theta); cos(theta)]);        %   fitted ellipse
1388          cFitR2 = sum (cFit.^2);                     % squared radius in stereographic coords
1389          cFitZ = (1 - cFitR2) ./ (1 + cFitR2);       % z direction cosine
1390          cFitX = cFit (1, :) .* (1 + cFitZ);         % undo stereographic projection
1391          cFitY = cFit (2, :) .* (1 + cFitZ);
1392          hpbw = hpbw * (1 + pz);                     % undo widths, too
1393          hpbwMjr = hpbw (2);
1394          hpbwMnr = hpbw (1);
1395        else
1396          roll = nan;
1397          hpbwMjr = nan;
1398          hpbwMnr = nan;
1399        end
1400        clear capBool adjc dirIndx intIndx dirIndxSt intIndxSt adjcInc capContX1 capContY1;
1401        clear capContXS capContYS cDesMtx cEigVec cEigVal cEigValMin cEigValMinIdx cPoly;
1402        clear cHessMtx cDervMtx capCenter capConst cEigValOrd hpbw;
1403        clear theta cFit cFitR2;
1404
1405        % Calculate power in visible space, main beam, and sidelobes; main
1406        % beam and sidelobe solid angles; and average sidelobe level
1407        %
1408        % These calculations involve integrals over the hemisphere or portions
1409        % of it.  The integrals are carried out in direction cosine
1410        % coordinates by multiplying the integrand by the appropriate
1411        % Jacobian.
1412        %
1413        % The integrals are evaluated using the midpoint approximation, for
1414        % which the starting point is the Taylor series expansion of g (x, y)
1415        % to second order:
1416        %
1417        %      g (a + u, b + v)
1418        %
1419        %            /                  1  2                1  2    \
1420        %          = | g + u g  + v g  + - u g   + u v g   + - v g   |   .
1421        %            \       x      y   2    xx        xy   2    yy /a,b
1422        %
1423        % Then
1424        %
1425        %      / c/2 / d/2
1426        %      | du  | dv  g (a + u, b + v) =
1427        %      /-c/2 /-d/2
1428        %
1429        %            /         1        2      2     \
1430        %          = | c d g + -- c d (c g  + d g  ) |   .
1431        %            \         24         xx     yy  /a,b
1432        %
1433        % The midpoint approximation keeps the first term and neglects the
1434        % quadratic terms.  The maximum amount neglected is
1435        %
1436        %      c d /  2 |         |    2 |         | \
1437        %      --- | c  |g  (a, b)| + d  |g  (a, b)| | ,
1438        %      24  \    | xx      |      | yy      | /
1439        %
1440        % which we use as the error estimate for each interior point of the
1441        % integration, approximating the second derivatives with scaled second
1442        % differences.
1443        %
1444        % To the interior error is added an estimate of the error due to
1445        % finite sampling at the integral limits; the estimate is half the
1446        % value of the integrand at the outermost samples.
1447        %
1448        % (The error estimate calculations have been commented out for
1449        % speed.)
1450        %
1451      % visbEdgeIndx = visBool;
1452      % visbEdgeIndx (2 : ay - 1, 2 : ax - 1) = ...
```

```
1453      %   ( visbEdgeIndx (2 : ay - 1, 3 : ax    ) & visbEdgeIndx (1 : ay - 2, 2 : ax - 1) ...
1454      %   & visbEdgeIndx (2 : ay - 1, 1 : ax - 2) & visbEdgeIndx (3 : ay    , 2 : ax - 1) );
1455      % visbEdgeIndx = find (visBool - visbEdgeIndx);
1456      % beamEdgeIndx = zeros (ay, ax);
1457      % beamEdgeIndx (beamIndx) = ones (size (beamIndx));
1458      % sideEdgeIndx = beamEdgeIndx;
1459      % beamEdgeIndx (2 : ay - 1, 2 : ax - 1) = ...
1460      %   ( beamEdgeIndx (2 : ay - 1, 3 : ax    ) & beamEdgeIndx (1 : ay - 2, 2 : ax - 1) ...
1461      %   & beamEdgeIndx (2 : ay - 1, 1 : ax - 2) & beamEdgeIndx (3 : ay    , 2 : ax - 1) );
1462      % beamEdgeIndx (beamIndx) = 1 - beamEdgeIndx (beamIndx);
1463      % beamEdgeIndx = find (beamEdgeIndx);
1464      % sideEdgeIndx (2 : ay - 1, 2 : ax - 1) = ...
1465      %   ( sideEdgeIndx (2 : ay - 1, 3 : ax    ) | sideEdgeIndx (1 : ay - 2, 2 : ax - 1) ...
1466      %   | sideEdgeIndx (2 : ay - 1, 1 : ax - 2) | sideEdgeIndx (3 : ay    , 2 : ax - 1) );
1467      % sideEdgeIndx (beamIndx) = zeros (size (beamIndx));
1468      % sideEdgeIndx = find (sideEdgeIndx);
1469        areaFact = zeros (ay, ax);                              % areal factor for integrating:
1470        areaFact (visBool) = ...                                %   Jacobian [1 / cos polar]
1471          1 ./ (txLim * tyLim * dirCosZMtx (visBool));          %   and grid spacing
1472        areaFactLim = 2 / sqrt (txLim * tyLim);                 % edge points may exceed this
1473        tooBig = find (areaFact > areaFactLim);                 %   arbitrary limit; force those
1474        areaFact (tooBig) = ones (size (tooBig)) * areaFactLim; %   that do to comply
1475      % sldAng = sum (areaFact (:));
1476      % sldAngErrRel = abs (sldAng / (2 * pi) - 1);
1477      % areaFactUnc = zeros (ay, ax);
1478      % areaFactUnc (2 : ay - 1, 2 : ax - 1) = ...              % error estimate based on second-
1479      %   (  abs (diff (areaFact (2 : ay - 1, :)', 2)') ...     %   order Taylor expansion
1480      %    + abs (diff (areaFact (:, 2 : ax - 1) , 2) ) ) / 24;
1481      % areaFactUnc ([1 ay], :) = areaFactUnc ([2 ay-1], :);    % assume errors at outer edge equal
1482      % areaFactUnc (:, [1 ax]) = areaFactUnc (:, [2 ax-1]);    %   those of nearest neighbors
1483        intgrnd = areaFact .* gSqr;                             % to integrate gSqr
1484      % intgrndUnc = zeros (ay, ax);                           % error estimate for integrand
1485      % intgrndUnc (2 : ay - 1, 2 : ax - 1) = ...
1486      %   (  abs (diff (intgrnd (2 : ay - 1, :)', 2)') ...
1487      %    + abs (diff (intgrnd (:, 2 : ax - 1) , 2) ) ) / 24;
1488      % intgrndUnc ([1 ay], :) = intgrndUnc ([2 ay-1], :);     % assume errors at outer edge equal
1489      % intgrndUnc (:, [1 ax]) = intgrndUnc (:, [2 ax-1]);     %   those of nearest neighbors
1490        powerVisb = sum (intgrnd (:));
1491      % powerVisbUnc = sum (intgrndUnc (:));
1492        directivityDB = 10 * log10 (4 * pi * gSqrMax / powerVisb);
1493        if beamExist
1494      %   sldAngMain = sum (areaFact (beamIndx));               % main beam solid angle
1495      %   sldAngMainUnc = sum (areaFactUnc (beamIndx)) + sum (areaFact (beamEdgeIndx)) / 2;
1496          powerMain = sum (intgrnd (beamIndx));               % power in the main beam
1497      %   powerMainUnc = sum (intgrndUnc (beamIndx)) + sum (intgrnd (beamEdgeIndx)) / 2;
1498          powerSide = sum (intgrnd (sideBool));               % power in the sidelobes
1499      %   powerSideUnc = sum (intgrndUnc (:)) - sum (intgrndUnc (beamIndx))...
1500      %     + sum (intgrnd (sideEdgeIndx)) / 2;
1501          sldAngSide = sum (areaFact (sideBool));             % sidelobe equivalent solid angle
1502      %   sldAngSideUnc = sum (areaFactUnc (sideBool)) - sum (areaFactUnc (beamIndx)) ...
1503      %     + sum (areaFact (sideEdgeIndx)) / 2;
1504        else
1505      %   sldAngMain = nan;
1506      %   sldAngMainUnc = nan;
1507          powerMain = nan;
1508      %   powerMainUnc = nan;
1509          powerSide = nan;
1510      %   powerSideUnc = nan;
1511          sldAngSide = nan;
1512      %   sldAngSideUnc = nan;
1513        end
1514        powerMainVisbDB = 10 * log10 (powerMain / powerVisb);   % ratio of main beam to visible power
1515      % powerMainVisbUnc = powerMainUnc / powerVisb ...
1516      %   + powerMain * powerVisbUnc / powerVisb^2;
1517        powerVisbSideDB = 10 * log10 (powerVisb / powerSide);   % ratio of visible to sidelobe power
1518      % powerVisbSideUnc = powerVisbUnc / powerSide ...
1519      %   + powerVisb * powerSideUnc / powerSide^2;
1520        powerMainSideDB = 10 * log10 (powerMain / powerSide);   % ratio of main beam to sidelobe power
1521      % powerMainSideUnc = powerMainUnc / powerSide ...
1522      %   + powerMain * powerSideUnc / powerSide^2;
1523        powerSideAvgDB = 10 * log10 (powerSide / (sldAngSide * gSqrMax));  % average sidelobe power
1524      % powerSideAvgUnc = powerSideUnc / sldAngSide ...
1525      %   + powerSide * sldAngSideUnc / sldAngSide^2;
1526        clear intgrnd areaFact areaFactLim tooBig;
1527      % clear intgrndUnc areaFactUnc;
1528
1529        % Locate nearest and largest sidelobes
1530        %
1531        % We wish to identify the sidelobe closest in angle to the main beam
1532        % and the sidelobe with the largest peak power.  We first find the
1533        % local maxima in the sidelobe region, then for each we determine
1534        % the possible ranges for actual distance from the main beam and
1535        % peak power.  (The uncertainties arise from the discrete sampling
1536        % of the array factor.)  Using the ranges we select those peaks that
1537        % could possibly be the closest or largest.  For each of these
1538        % candidates a more precise location and peak power is computed by
1539        % interpolating over neighboring data.  Finally, based on these
1540        % results, the closest and nearest sidelobes are identified.
1541        %
1542        if peakVisb
1543          %
1544          % Find local maxima (sidelobe peaks), using discrete differences
```

```
% to approximate derivatives.  The differences are formed from the
% magnitude of the array factor, not the squared magnitude; since
% the behavior should already be parabolic near peaks, squaring
% would produce fourth-order behavior and make second-order
% interpolation less accurate.
%
% Key to the variables below:
%   X   first differences in x
%   Y   first differences in y
%   XX  second differences in x
%   YY  second differences in y
%   X2  first differences in x with double step
%   XY  cross differences in x and y
%   XC  nonzero where first difference in x changes sign
%   YC  nonzero where first difference in y changes sign
%
gMagX  = gMag (:, 2:ax) - gMag (:, 1:ax-1);
gMagY  = gMag (2:ay, :) - gMag (1:ay-1, :);
gMagXX = [zeros(ay,1)  (gMagX (:, 2:ax-1)  - gMagX (:, 1:ax-2))      zeros(ay,1)];
gMagYY = [zeros(1,ax); (gMagY (2:ay-1, :)  - gMagY (1:ay-2, :));     zeros(1,ax)];
gMagX2 = [zeros(ay,1)  (gMag  (:, 3:ax  )  - gMag  (:, 1:ax-2))/2    zeros(ay,1)];
gMagXY = [zeros(1,ax); (gMagX2(3:ay  , :)  - gMagX2(1:ay-2, :))/2;   zeros(1,ax)];
gMagXC = [zeros(ay,1)  (gMagX (:, 2:ax-1) .* gMagX (:, 1:ax-2) < 0)  zeros(ay,1)];
gMagYC = [zeros(1,ax); (gMagY (2:ay-1, :) .* gMagY (1:ay-2, :) < 0); zeros(1,ax)];
slIndx = find ( sideBool ...                    % identify sidelobe points where
  &  gMagXC      &  gMagYC      ...             %   first derivatives change sign,
  & (gMagXX < 0) & (gMagYY < 0) ...             %   second derivatives are negative, and
  & (gMagXY.^2 - gMagXX .* gMagYY < 0) );       %   discriminant is negative
if isempty (slIndx)                             % none found?
  slIndx = [];
  slNrstDist = nan;
  slNrstPowrDB = nan;
  slNrstVec = nan * ones (1, 3);
  slLgstDist = nan;
  slLgstPowrDB = nan;
  slLgstVec = nan * ones (1, 3);
else                                            % found some peaks
  %
  % Compute possible ranges of distances and powers; identify
  % candidates for closest and largest peaks
  %
  toward = sign (px - dirCosXMtx (slIndx)) * ay ...   % index increment to neighbor
         + sign (py - dirCosYMtx (slIndx));           %   closer to pointing vector
  slCosDistMax = ...                                  % cosine of maximum possible
      px * dirCosXMtx (slIndx + toward) ...           %   angle between pointing
    + py * dirCosYMtx (slIndx + toward) ...           %   vector and each peak;
    + pz * dirCosZMtx (slIndx + toward);              %   dot product
  slCosDistMin = ...                                  % likewise for minimum
      px * dirCosXMtx (slIndx - toward) ...           %   possible angle
    + py * dirCosYMtx (slIndx - toward) ...
    + pz * dirCosZMtx (slIndx - toward);
  slNrstBool = (slCosDistMax >= max (slCosDistMin));  % true if peak might be the closest
  slPowr = (gMag (slIndx) ...                         % estimate largest possible
    + ( -gMagXX (slIndx) ...                          %   interpolated power by adding
        + 2 * abs (gMagXY (slIndx)) ...               %   an error estimate based on the
        - gMagYY (slIndx) ) / 8).^2;                  %   differences computed above
  slLgstBool = (slPowr >= max (gSqr (slIndx)));       % true if peak might be the largest
  slCandIndx = slIndx (slNrstBool | slLgstBool);      % candidates for closest and largest
  numCand = length (slCandIndx);                      % number of candidates
  %
  % Interpolate powers and locations for candidates
  %
  slx = zeros (numCand, 1);                           % allocate space for x and
  sly = zeros (numCand, 1);                           %   y direction cosines
  slPowr = zeros (numCand, 1);                        %   and powers
  for slPtr = 1 : numCand                             % loop through candidates
    slCol = ceil (slCandIndx (slPtr) / ay);           % separate index into column
    slRow = slCandIndx (slPtr) - (slCol - 1) * ay;    %   and row indices
    intRad = 1;                                       % interpolation radius
    slOK = 0;                                         % initialize
    while ~slOK & intRad < 3                          % no answer yet but too early to bail?
      slNbrs = [-intRad : intRad];                    % offsets to neighbors
      slxFit = dirCosXMtx (slRow + slNbrs, slCol + slNbrs);  % x, y, and z coordinates of
      slyFit = dirCosYMtx (slRow + slNbrs, slCol + slNbrs);  %   neighbors (using magnitude,
      slzFit = gMag       (slRow + slNbrs, slCol + slNbrs);  %   not power, for z)
      slxFit = slxFit (:);
      slyFit = slyFit (:);
      slzFit = slzFit (:);
      slDesMtx = [slxFit.*slxFit/2 slxFit.*slyFit slyFit.*slyFit/2 ...  % design matrix
        slxFit slyFit ones(size(slxFit))];
      [Q, R] = qr (slDesMtx, 0);                      % now slDesMtx = Q * R; Q' * Q = eye
      slPoly = R \ (Q' * slzFit);                     % solves slDesMtx * slPoly = slzFit
      slVec = -[slPoly(1) slPoly(2); slPoly(2) slPoly(3)] ...  % find critical point
        \ [slPoly(4); slPoly(5)];
      slOK = ...                                      % is interpolated point
          (slVec (1) > dirCosX (slCol - intRad)) ...  %   inside neighborhood?
        & (slVec (1) < dirCosX (slCol + intRad)) ...  %   (pathological cases can
        & (slVec (2) > dirCosY (slRow - intRad)) ...  %   place it outside)
        & (slVec (2) < dirCosY (slRow + intRad));
      if ~slOK                                        % outside neighborhood?
        intRad = intRad + 1;                          % yes; cast a wider net
      end
```

```
1637            end                                                 % end interpolation attempts
1638            if slOK                                             % interpolation successful?
1639              slx (slPtr) = slVec (1);                          % yes; keep interpolated point
1640              sly (slPtr) = slVec (2);
1641              slPowr (slPtr) = ...                              % interpolate power
1642                ([slx(slPtr).*slx(slPtr)/2 slx(slPtr).*sly(slPtr) ...
1643                  sly(slPtr).*sly(slPtr)/2 slx(slPtr) sly(slPtr) 1] * slPoly).^2;
1644            else                                                % interpolation failed
1645              slx (slPtr) = dirCosXMtx (slCandIndx (slPtr));    % use grid location of
1646              sly (slPtr) = dirCosYMtx (slCandIndx (slPtr));    %   sampled maximum
1647              slPowr (slPtr) = gSqr (slCandIndx (slPtr));       % use sampled power
1648            end
1649          end                                                   % end of loop through candidates
1650          %
1651          % Select closest and largest peaks
1652          %
1653          slz = sqrt (1 - slx.^2 - sly.^2);                     % z direction cosines
1654          slDist = acos (px * slx + py * sly + pz * slz);       % angular distances
1655          [slNrstDist, slNrstIndx] = min (slDist);                          % smallest distance
1656          slNrstPowrDB = 10 * log10 (slPowr (slNrstIndx) / gSqrMax);        %   and corresponding power
1657          slNrstVec = [slx(slNrstIndx) sly(slNrstIndx) slz(slNrstIndx)];    % keep vector for plotting
1658          [slLgstPowrDB, slLgstIndx] = max (slPowr);                        % largest power
1659          slLgstPowrDB = 10 * log10 (slLgstPowrDB / gSqrMax);               %   converted to dB
1660          slLgstDist = slDist (slLgstIndx);                                 % and corresponding distance
1661          slLgstVec = [slx(slLgstIndx) sly(slLgstIndx) slz(slLgstIndx)];    % keep vector
1662        end
1663      else                                                      % peak is invisible
1664        slIndx = [];
1665        slNrstDist = nan;
1666        slNrstPowrDB = nan;
1667        slNrstVec = nan * ones (1, 3);
1668        slLgstDist = nan;
1669        slLgstPowrDB = nan;
1670        slLgstVec = nan * ones (1, 3);
1671      end
1672      clear gMagX gMagY gMagXX gMagYY gMagX2 gMagXY gMagXC gMagYC;
1673      clear toward slCosDistMax slCosDistMin;
1674      clear slNrstBool slLgstBool slCandIndx numCand;
1675      clear slx sly slz slPowr slPtr slCol slRow intRad slOK;
1676      clear slNbrs slxFit slyFit slzFit slDesMtx Q R slPoly slVec;
1677      clear slDist slnrstindx slLgstIndx;
1678
1679      % Record characteristics
1680      %
1681      % In order to calculate running means and standard deviations of n
1682      % realizations, we accumulate the mean and variance
1683      %
1684      %           1  n
1685      %      M  = - sum x   and
1686      %       n   n m=1  m
1687      %
1688      %             1    n           2
1689      %      V  = ----- sum (x  - M )
1690      %       n   n - 1 m=1   m    n
1691      %
1692      % using the updating formulas
1693      %
1694      %                  x  - M
1695      %                   n    n-1
1696      %      M  = M    + ---------  and
1697      %       n    n-1       n
1698      %
1699      %           n - 2       1           2
1700      %      V  = ----- V    + - (x  - M   )  .
1701      %       n   n - 1  n-1   n   n    n-1
1702      %
1703      % The running mean is simply M[n], and the standard deviation is
1704      % sqrt (V[n]).  M is accumulated in the first column of a matrix; V
1705      % in the second.  This method is more immune to roundoff error than
1706      % accumulating the sums of values and squares [1].
1707      %
1708      % If the beam is invisible, the excitations and array factor are
1709      % saved to an automatically-named file.
1710      %
1711      % [1] N. J. Higham, _Accuracy and Stability of Numerical Algorithms_.
1712      % Philadelphia, PA: SIAM, 1996, pp. 12-13.
1713      %
1714      if beamVisb | (numRlz == 1)
1715        if isnan (numAcc (indx))
1716          numAcc (indx) = 1;
1717          pxS                 (indx, 1) = px;
1718          pxS                 (indx, 2) = 0;
1719          pyS                 (indx, 1) = py;
1720          pyS                 (indx, 2) = 0;
1721          pzS                 (indx, 1) = pz;
1722          pzS                 (indx, 2) = 0;
1723          errPointS           (indx, 1) = errPoint;
1724          errPointS           (indx, 2) = 0;
1725        % errPointUncS         (indx, 1) = errPointUnc;
1726        % errPointUncS         (indx, 2) = 0;
1727          beamPowerDBS        (indx, 1) = beamPowerDB;
1728          beamPowerDBS        (indx, 2) = 0;
```

```
1729          beamDepthDBS        (indx, 1) = beamDepthDB;
1730          beamDepthDBS        (indx, 2) = 0;
1731          hpbwMjrS            (indx, 1) = hpbwMjr;
1732          hpbwMjrS            (indx, 2) = 0;
1733          hpbwMnrS            (indx, 1) = hpbwMnr;
1734          hpbwMnrS            (indx, 2) = 0;
1735          rollS               (indx, 1) = roll;
1736          rollS               (indx, 2) = 0;
1737          directivityDBS      (indx, 1) = directivityDB;
1738          directivityDBS      (indx, 2) = 0;
1739          powerMainVisbDBS    (indx, 1) = powerMainVisbDB;
1740          powerMainVisbDBS    (indx, 2) = 0;
1741          powerMainSideDBS    (indx, 1) = powerMainSideDB;
1742          powerMainSideDBS    (indx, 2) = 0;
1743          powerVisbSideDBS    (indx, 1) = powerVisbSideDB;
1744          powerVisbSideDBS    (indx, 2) = 0;
1745          powerSideAvgDBS     (indx, 1) = powerSideAvgDB;
1746          powerSideAvgDBS     (indx, 2) = 0;
1747          slNrstDistS         (indx, 1) = slNrstDist;
1748          slNrstDistS         (indx, 2) = 0;
1749          slNrstPowrDBS       (indx, 1) = slNrstPowrDB;
1750          slNrstPowrDBS       (indx, 2) = 0;
1751          slLgstDistS         (indx, 1) = slLgstDist;
1752          slLgstDistS         (indx, 2) = 0;
1753          slLgstPowrDBS       (indx, 1) = slLgstPowrDB;
1754          slLgstPowrDBS       (indx, 2) = 0;
1755        else
1756          numAcc (indx) = numAcc (indx) + 1;
1757          factV = (numAcc (indx) - 2) / (numAcc (indx) - 1);
1758          dev = px - pxS (indx, 1);
1759          pxS                 (indx, 2) = pxS             (indx, 2) * factV + dev^2 / numAcc (indx);
1760          pxS                 (indx, 1) = pxS             (indx, 1)         + dev   / numAcc (indx);
1761          dev = py - pyS (indx, 1);
1762          pyS                 (indx, 2) = pyS             (indx, 2) * factV + dev^2 / numAcc (indx);
1763          pyS                 (indx, 1) = pyS             (indx, 1)         + dev   / numAcc (indx);
1764          dev = pz - pzS (indx, 1);
1765          pzS                 (indx, 2) = pzS             (indx, 2) * factV + dev^2 / numAcc (indx);
1766          pzS                 (indx, 1) = pzS             (indx, 1)         + dev   / numAcc (indx);
1767          dev = errPoint - errPointS (indx, 1);
1768          errPointS           (indx, 2) = errPointS       (indx, 2) * factV + dev^2 / numAcc (indx);
1769          errPointS           (indx, 1) = errPointS       (indx, 1)         + dev   / numAcc (indx);
1770        % dev = errPointUnc - errPointUncS (indx, 1);
1771        % errPointUncS       (indx, 2) = errPointUncS    (indx, 2) * factV + dev^2 / numAcc (indx);
1772        % errPointUncS       (indx, 1) = errPointUncS    (indx, 1)         + dev   / numAcc (indx);
1773          dev = beamPowerDB - beamPowerDBS (indx, 1);
1774          beamPowerDBS        (indx, 2) = beamPowerDBS    (indx, 2) * factV + dev^2 / numAcc (indx);
1775          beamPowerDBS        (indx, 1) = beamPowerDBS    (indx, 1)         + dev   / numAcc (indx);
1776          dev = beamDepthDB - beamDepthDBS (indx, 1);
1777          beamDepthDBS        (indx, 2) = beamDepthDBS    (indx, 2) * factV + dev^2 / numAcc (indx);
1778          beamDepthDBS        (indx, 1) = beamDepthDBS    (indx, 1)         + dev   / numAcc (indx);
1779          dev = hpbwMjr - hpbwMjrS (indx, 1);
1780          hpbwMjrS            (indx, 2) = hpbwMjrS        (indx, 2) * factV + dev^2 / numAcc (indx);
1781          hpbwMjrS            (indx, 1) = hpbwMjrS        (indx, 1)         + dev   / numAcc (indx);
1782          dev = hpbwMnr - hpbwMnrS (indx, 1);
1783          hpbwMnrS            (indx, 2) = hpbwMnrS        (indx, 2) * factV + dev^2 / numAcc (indx);
1784          hpbwMnrS            (indx, 1) = hpbwMnrS        (indx, 1)         + dev   / numAcc (indx);
1785          dev = roll - rollS (indx, 1);
1786          rollS               (indx, 2) = rollS           (indx, 2) * factV + dev^2 / numAcc (indx);
1787          rollS               (indx, 1) = rollS           (indx, 1)         + dev   / numAcc (indx);
1788          dev = directivityDB - directivityDBS (indx, 1);
1789          directivityDBS      (indx, 2) = directivityDBS  (indx, 2) * factV + dev^2 / numAcc (indx);
1790          directivityDBS      (indx, 1) = directivityDBS  (indx, 1)         + dev   / numAcc (indx);
1791          dev = powerMainVisbDB - powerMainVisbDBS (indx, 1);
1792          powerMainVisbDBS (indx, 2) = powerMainVisbDBS (indx, 2) * factV + dev^2 / numAcc (indx);
1793          powerMainVisbDBS (indx, 1) = powerMainVisbDBS (indx, 1)         + dev   / numAcc (indx);
1794          dev = powerMainSideDB - powerMainSideDBS (indx, 1);
1795          powerMainSideDBS (indx, 2) = powerMainSideDBS (indx, 2) * factV + dev^2 / numAcc (indx);
1796          powerMainSideDBS (indx, 1) = powerMainSideDBS (indx, 1)         + dev   / numAcc (indx);
1797          dev = powerVisbSideDB - powerVisbSideDBS (indx, 1);
1798          powerVisbSideDBS (indx, 2) = powerVisbSideDBS (indx, 2) * factV + dev^2 / numAcc (indx);
1799          powerVisbSideDBS (indx, 1) = powerVisbSideDBS (indx, 1)         + dev   / numAcc (indx);
1800          dev = powerSideAvgDB - powerSideAvgDBS (indx, 1);
1801          powerSideAvgDBS  (indx, 2) = powerSideAvgDBS  (indx, 2) * factV + dev^2 / numAcc (indx);
1802          powerSideAvgDBS  (indx, 1) = powerSideAvgDBS  (indx, 1)         + dev   / numAcc (indx);
1803          dev = slNrstDist - slNrstDistS (indx, 1);
1804          slNrstDistS         (indx, 2) = slNrstDistS     (indx, 2) * factV + dev^2 / numAcc (indx);
1805          slNrstDistS         (indx, 1) = slNrstDistS     (indx, 1)         + dev   / numAcc (indx);
1806          dev = slNrstPowrDB - slNrstPowrDBS (indx, 1);
1807          slNrstPowrDBS       (indx, 2) = slNrstPowrDBS   (indx, 2) * factV + dev^2 / numAcc (indx);
1808          slNrstPowrDBS       (indx, 1) = slNrstPowrDBS   (indx, 1)         + dev   / numAcc (indx);
1809          dev = slLgstDist - slLgstDistS (indx, 1);
1810          slLgstDistS         (indx, 2) = slLgstDistS     (indx, 2) * factV + dev^2 / numAcc (indx);
1811          slLgstDistS         (indx, 1) = slLgstDistS     (indx, 1)         + dev   / numAcc (indx);
1812          dev = slLgstPowrDB - slLgstPowrDBS (indx, 1);
1813          slLgstPowrDBS       (indx, 2) = slLgstPowrDBS   (indx, 2) * factV + dev^2 / numAcc (indx);
1814          slLgstPowrDBS       (indx, 1) = slLgstPowrDBS   (indx, 1)         + dev   / numAcc (indx);
1815        end
1816      else
1817        eval (['save case' sprintf('%0.0f',indx) '-' sprintf('%0.0f',rlzNum) ' exc gSqr']);
1818      end
1819      clear factV dev
1820
```

```
1821        % Calculate spherical coordinates of average pointing vector
1822        %
1823        % Let <px>, <py>, and <pz> denote the average direction cosines of
1824        % the pointing vectors, and let vx, vy, and vz denote the
1825        % corresponding variances.  We wish to express the direction of the
1826        % average pointing vector [<px> <py> <pz>] in spherical coordinates.
1827        % First, note that the average pointing vector has the norm
1828        %
1829        %              2       2       2
1830        %     p = (<px>  + <py>  + <pz> ) ,
1831        %
1832        % which is less than one if not all realizations are colinear.  The
1833        % spherical angles are then given by
1834        %
1835        %                       1      2       2 1/2
1836        %     sin pointPolar = - (<px>  + <py> )     and
1837        %                       p
1838        %
1839        %                       <py>
1840        %     tan pointAzmth = ---- ,
1841        %                       <px>
1842        %
1843        % where we use sin pointPolar instead of the cosine for accuracy
1844        % near boresight.
1845        %
1846        % Also, we wish to estimate the rms angular deviation of a
1847        % realization of the pointing vector from the mean.  Adopting an
1848        % unsophisticated method, we add the variances vx and vy to obtain
1849        % an equivalent area in the x-y direction cosine plane, then divide
1850        % the area by cos PointPolar to yield a solid angle on the
1851        % hemisphere.  The square root of that area yields the rms angular
1852        % deviation.
1853        %
1854        radSqr = pxS (indx, 1)^2 + pyS (indx, 1)^2;
1855        pointPolar = asin (sqrt (radSqr / (radSqr + pzS (indx, 1)^2)));
1856        pointAzmth = atan2 (pyS (indx, 1), pxS (indx, 1));
1857        if isnan (pointPolar)
1858          pointStdDev = nan;
1859        else
1860          pointStdDev = sqrt ((pxS (indx, 2) + pyS (indx, 2)) / cos (pointPolar));
1861        end
1862        clear radSqr
1863
1864        % Print performance characteristics
1865        %
1866        % Generally, the following statements print the results of the
1867        % analysis followed by the standard deviations of each result in
1868        % curly brackets.
1869        %
1870        if 1  % INPUT 0 to suppress output, 1 to print
1871          fprintf (1, '\nMeans and [std devs] for %0.0f of %0.0f realizations\n', rlzNum, numRlz);
1872          fprintf (1, 'beam direction : (%0.3f, %0.3f) deg, std dev %0.3f deg\n', ...
1873            pointPolar / rpd, (pointAzmth + azmthOffst) / rpd, pointStdDev / rpd);
1874          fprintf (1, 'pointing error : %8.4f [%0.4f] deg\n', ...
1875            errPointS (indx, 1) / rpd, sqrt (errPointS (indx, 2)) / rpd);
1876        % fprintf (1, 'pntng error unc: %8.4f [%0.4f] deg (2 sigma)\n', ...
1877        %   2 * errPointUncS (indx, 1) / rpd, 2 * sqrt (errPointUncS (indx, 2)) / rpd);
1878          fprintf (1, 'peak power dens: %7.3f  [%0.3f ] dB\n', ...
1879            beamPowerDBS (indx, 1), sqrt (beamPowerDBS (indx, 2)));
1880          fprintf (1, 'beam depth     : %6.2f   [%0.2f  ] dB re peak\n', ...
1881            beamDepthDBS (indx, 1), sqrt (beamDepthDBS (indx, 2)));
1882          fprintf (1, 'beam width     : (%6.3f  [%0.3f ], %0.3f [%0.3f]) deg\n', ...
1883            hpbwMjrS (indx, 1) / rpd, sqrt (hpbwMjrS (indx, 2)) / rpd, ...
1884            hpbwMnrS (indx, 1) / rpd, sqrt (hpbwMnrS (indx, 2)) / rpd);
1885          fprintf (1, 'beam roll      : %6.2f   [%0.2f  ] deg\n', ...
1886            (rollS (indx, 1) + azmthOffst) / rpd, sqrt (rollS (indx, 2)) / rpd);
1887          fprintf (1, 'directivity    : %7.3f  [%0.3f ] dB\n', ...
1888            directivityDBS (indx, 1), sqrt (directivityDBS (indx, 2)));
1889          fprintf (1, 'power ratio m/v: %7.3f  [%0.3f ] dB\n', ...
1890            powerMainVisbDBS (indx, 1), sqrt (powerMainVisbDBS (indx, 2)));
1891          fprintf (1, 'power ratio m/s: %7.3f  [%0.3f ] dB\n', ...
1892            powerMainSideDBS (indx, 1), sqrt (powerMainSideDBS (indx, 2)));
1893          fprintf (1, 'power ratio v/s: %7.3f  [%0.3f ] dB\n', ...
1894            powerVisbSideDBS (indx, 1), sqrt (powerVisbSideDBS (indx, 2)));
1895          fprintf (1, 'avg sidelobe   : %6.2f   [%0.2f  ] dB re peak\n', ...
1896            powerSideAvgDBS (indx, 1), sqrt (powerSideAvgDBS (indx, 2)));
1897          fprintf (1, 'nrst sidelobe  : %6.2f   [%0.2f] dB re peak, %0.2f [%0.2f] deg off beam\n', ...
1898            slNrstPowrDBS (indx, 1), sqrt (slNrstPowrDBS (indx, 2)), ...
1899            slNrstDistS (indx, 1) / rpd, sqrt (slNrstDistS (indx, 2)) / rpd);
1900          fprintf (1, 'lgst sidelobe  : %6.2f   [%0.2f] dB re peak, %0.2f [%0.2f] deg off beam\n', ...
1901            slLgstPowrDBS (indx, 1), sqrt (slLgstPowrDBS (indx, 2)), ...
1902            slLgstDistS (indx, 1) / rpd, sqrt (slLgstDistS (indx, 2)) / rpd);
1903        end
1904
1905      end  % loop over realizations
1906
1907      % Plot performance characteristics as function of independent variable
1908      % (summary plot).
1909      %
1910      % If more than one realization has been accumulated for any value of
1911      % the independent variable, the mean is plotted with uncertainty bars.
1912      % The extension of the uncertainty bar above the mean equals one
```

```
% standard deviation, and likewise below the mean.  Otherwise, only
% the values for the one realization are plotted.
%
% Plots that show more than one measure distinguish them by color or
% line style and may use both a left and right axis.  The color or
% style and axis for each measure is given in codes in parentheses in
% the title of the plot.  The first code abbreviates the color or line
% style:
%
%      R  red          -   solid
%      G  green        --  dashed
%      B  blue         :   dotted
%      C  cyan
%      M  magenta
%      Y  yellow
%      K  black
%      W  white
%
% and the second letter indicates the axis, L for left and R for
% right.
%
if indVarLen > 1
  figure (figSum);
  clf;
  axesSum = zeros (12, 1);  % space for axes handles
  %
  % Axes 1: pointing error
  %
  subplot (4, 2, 1);
  axesSum (1) = gca;
  if any (numAcc > 1)
    hline = errorbar (indVar, errPointS (:, 1)  / rpd, sqrt (errPointS (:, 2)) / rpd);
    set (hline, discrim, discrimValue {1});  % possibly override errorbar's default solid line style
    set (hline (1), 'linestyle', '-');       %   but leave the error bars themselves solid
  else
    plot (indVar, errPointS (:, 1) / rpd);
  end
  ylabel ('(deg)');
  title ('pointing error');
  %
  % Axes 2 and 3: beam power and directivity
  %
  subplot (4, 2, 2);
  axesSum (2) = gca;
  if any (numAcc > 1)
    hline = errorbar (indVar, beamPowerDBS (:, 1), sqrt (beamPowerDBS (:, 2)));
    set (hline, discrim, discrimValue {1});
    set (hline (1), 'linestyle', '-');
  else
    plot (indVar, beamPowerDBS (:, 1));
  end
  ylabel ('(dB re coherent)');
  title (strcat ('peak power dens (', discrimName {1}, ...
                 ' L), directivity (', discrimName {2}, ' R)'));
  axesSum (3) = axes ('position', get (gca, 'position'));
  if any (numAcc > 1)
    hline = errorbar (indVar, directivityDBS (:, 1), sqrt (directivityDBS (:, 2)));
    set (hline, discrim, discrimValue {2});
    set (hline (1), 'linestyle', '-');
  else
    plot (indVar, directivityDBS (:, 1), discrim, discrimValue {2});
  end
  set (gca, 'yAxisLocation', 'right', 'color', 'none');
  %
  % Axes 4 and 5: beam widths
  %
  subplot (4, 2, 3);
  axesSum (4) = gca;
  if any (numAcc > 1)
    hline = errorbar (indVar, hpbwMjrS (:, 1)  / rpd, sqrt (hpbwMjrS (:, 2)) / rpd);
    set (hline, discrim, discrimValue {1});
    set (hline (1), 'linestyle', '-');
  else
    plot (indVar, hpbwMjrS (:, 1) / rpd);
  end
  ylabel ('(deg)');
  title (strcat ('hpbw major (', discrimName {1}, ...
                 ' L) & minor (', discrimName {2}, ' R)'));
  axesSum (5) = axes ('position', get (gca, 'position'));
  if any (numAcc > 1)
    hline = errorbar (indVar, hpbwMnrS (:, 1)  / rpd, sqrt (hpbwMnrS (:, 2)) / rpd);
    set (hline, discrim, discrimValue {2});
    set (hline (1), 'linestyle', '-');
  else
    plot (indVar, hpbwMnrS (:, 1) / rpd, discrim, discrimValue {2});
  end
  set (gca, 'yAxisLocation', 'right', 'color', 'none');
  yLimL = get (axesSum (4), 'ylim');
  yLimR = get (axesSum (5), 'ylim');
  if yLimL (1) < yLimR (2)
    set (axesSum (4), 'xlimmode', 'manual', 'ylim', [yLimR(1) yLimL(2)]);
    set (axesSum (5), 'xlimmode', 'manual', 'ylim', [yLimR(1) yLimL(2)]);
```

```
2005          % Setting xlimmode to manual prevents rescaling of the x axis when
2006          % the y axis is changed.
2007        end
2008        %
2009        % Axes 6: beam roll
2010        %
2011        subplot (4, 2, 4);
2012        axesSum (6) = gca;
2013        if any (numAcc > 1)
2014          hline = errorbar (indVar, rollS (:, 1)  / rpd, sqrt (rollS (:, 2)) / rpd);
2015          set (hline, discrim, discrimValue {1});
2016          set (hline (1), 'linestyle', '-');
2017        else
2018          plot (indVar, rollS (:, 1) / rpd);
2019        end
2020        ylabel ('(deg)');
2021        title ('beam roll');
2022        %
2023        % Axes 7: beam depth
2024        %
2025        subplot (4, 2, 5);
2026        axesSum (7) = gca;
2027        if any (numAcc > 1)
2028          hline = errorbar (indVar, beamDepthDBS (:, 1), sqrt (beamDepthDBS (:, 2)));
2029          set (hline, discrim, discrimValue {1});
2030          set (hline (1), 'linestyle', '-');
2031        else
2032          plot (indVar, beamDepthDBS (:, 1));
2033        end
2034        ylabel ('(dB re peak)');
2035        title ('beam depth');
2036        %
2037        % Axes 8 and 9: power ratios
2038        %
2039        subplot (4, 2, 6);
2040        axesSum (8) = gca;
2041        if any (numAcc > 1)
2042          hline = errorbar (indVar, powerMainVisbDBS(:,1), sqrt (powerMainVisbDBS(:,2)));
2043          set (hline, discrim, discrimValue {1});
2044          set (hline (1), 'linestyle', '-');
2045        else
2046          plot (indVar, powerMainVisbDBS(:,1));
2047        end
2048        ylabel ('(dB)');
2049        title (strcat ('power m/v (', discrimName {1}, ...
2050                       ' L), m/s (', discrimName {2}, ...
2051                       ' R), v/s (', discrimName {3}, ' R)'));
2052        axesSum (9) = axes ('position', get (gca, 'position'));
2053        if any (numAcc > 1)
2054          hline = errorbar (indVar, powerMainSideDBS(:,1), ...
2055                            sqrt (powerMainSideDBS(:,2)));
2056          set (hline, discrim, discrimValue {2});
2057          set (hline (1), 'linestyle', '-');
2058          set (gca, 'nextplot', 'add');
2059          hline = errorbar (indVar, powerVisbSideDBS(:,1), ...
2060                            sqrt (powerVisbSideDBS(:,2)));
2061          set (hline, discrim, discrimValue {3});
2062          set (hline (1), 'linestyle', '-');
2063          set (gca, 'nextplot', 'replace');
2064        else
2065          plot (indVar, powerMainSideDBS(:,1), discrim, discrimValue {2});
2066          set (gca, 'nextplot', 'add');
2067          plot (indVar, powerVisbSideDBS(:,1), discrim, discrimValue {3});
2068          set (gca, 'nextplot', 'replace');
2069        end
2070        set (gca, 'yAxisLocation', 'right', 'color', 'none');
2071        %
2072        % Axes 10 and 11: sidelobe powers
2073        %
2074        subplot (4, 2, 7);
2075        axesSum (10) = gca;
2076        if any (numAcc > 1)
2077          hline = errorbar (indVar, slLgstPowrDBS (:, 1), sqrt (slLgstPowrDBS (:, 2)));
2078          set (hline, discrim, discrimValue {1});
2079          set (hline (1), 'linestyle', '-');
2080          set (gca, 'nextplot', 'add');
2081          hline = errorbar (indVar, slNrstPowrDBS (:, 1), sqrt (slNrstPowrDBS (:, 2)));
2082          set (hline, discrim, discrimValue {2});
2083          set (hline (1), 'linestyle', '-');
2084          set (gca, 'nextplot', 'replace');
2085        else
2086          plot (indVar * ones (1, 2), ...
2087            [slLgstPowrDBS(:,1) slNrstPowrDBS(:,1)]);
2088        end
2089        ylabel ('(dB re peak)');
2090        title (strcat ('sidelobe power: lgst (', discrimName {1}, ...
2091                                  ' L), nrst (', discrimName {2}, ...
2092                                  ' L), avg (',  discrimName {3}, ' R)'));
2093        axesSum (11) = axes ('position', get (gca, 'position'));
2094        if any (numAcc > 1)
2095          hline = errorbar (indVar, powerSideAvgDBS (:, 1), sqrt (powerSideAvgDBS (:, 2)));
2096          set (hline, discrim, discrimValue {3});
```

```
2097          set (hline (1), 'linestyle', '-');
2098        else
2099          plot (indVar, powerSideAvgDBS (:, 1), discrim, discrimValue {3});
2100        end
2101        set (gca, 'yAxisLocation', 'right', 'color', 'none');
2102        %
2103        % Axes 12: sidelobe distances
2104        %
2105        subplot (4, 2, 8);
2106        axesSum (12) = gca;
2107        if any (numAcc > 1)
2108          hline = errorbar (indVar, slLgstDistS (:, 1) / rpd, sqrt (slLgstDistS (:, 2)) / rpd);
2109          set (hline, discrim, discrimValue {1});
2110          set (hline (1), 'linestyle', '-');
2111          set (gca, 'nextplot', 'add');
2112          hline = errorbar (indVar, slNrstDistS (:, 1) / rpd, sqrt (slNrstDistS (:, 2)) / rpd);
2113          set (hline, discrim, discrimValue {2});
2114          set (hline (1), 'linestyle', '-');
2115          set (gca, 'nextplot', 'replace');
2116        else
2117          plot (indVar * ones (1, 2), ...
2118            [slLgstDistS(:,1) slNrstDistS(:,1)] / rpd);
2119        end
2120        ylabel ('(deg)');
2121        title (strcat ('sidelobe distance: lgst (', discrimName {1}, ...
2122                                    '), nrst (', discrimName {2}, ')'));
2123        %
2124        % Touch up
2125        %
2126        set (findobj (gcf, 'type', 'axes'), 'xlim', ...     % make x-axis limits uniform
2127          [min(indVar) max(indVar)] ...
2128          + 0.1 * (max (indVar) - min (indVar)) * [-1 1]);
2129        axesSumTitle = axes ('position', [0 0 1 1], ...     % create invisible axes
2130          'color', 'none', 'visible', 'off', ...            %   for titling
2131          'defaultTextFontSize', 10, ...
2132          'defaultTextHorizontalAlignment', 'center');
2133        text (0.5, 0.05, indVarName, ...                    % display name of independent
2134          'horizontalAlignment', 'center');                 %   variable
2135      end
2136      clear hline
2137
2138      % Plot last realization
2139      %
2140      % Three projections of the hemisphere are available: Lambert,
2141      % stereographic, and orthographic.
2142      %
2143      % Lambert projection:
2144      %
2145      % The Lambert projection preserves the relative areas of portions of
2146      % the hemisphere.  That is, the ratio of areas of two regions on the
2147      % projection is the same as on the hemisphere.  The azimuth coordinate
2148      % of a point in the projection is the same as its azimuth coordinate
2149      % on the hemisphere, while its radius  r  from the center of the
2150      % projection is related to the polar angle.  This relationship may be
2151      % derived by setting the spherical surface area  sin polar d(polar)
2152      % d(azmth)  equal to a constant times the planar surface area  r dr
2153      % d(azmth)  and integrating.  The radius is then given by
2154      %
2155      %           1/2     polar
2156      %      r = 2    sin ----- .
2157      %                     2
2158      %
2159      % For computer graphics the Cartesian coordinates are more convenient;
2160      % they are
2161      %
2162      %      u = r cos azmth = R cx
2163      %
2164      %      v = r sin azmth = R cy
2165      %
2166      % where
2167      %
2168      %           -1/2     polar           -1/2
2169      %      R = 2     sec ----- = (1 + cz)
2170      %                     2
2171      %
2172      % For points outside visible space (that is, for cx^2 + cy^2 > 1), cz
2173      % is zero so that R = 1 and no transformation is applied.
2174      %
2175      % Stereographic projection:
2176      %
2177      % The stereographic projection preserves angles on the hemisphere.  To
2178      % derive the governing relationship for the projection, use the same
2179      % azimuthal angle for the projection as for the hemisphere, and let
2180      % the radius be a function of the polar angle: r = f (polar).  Equate
2181      % the aspect ratios of orthogonal derivatives, as
2182      %
2183      %          d(polar)            dr
2184      %      ------------------ = ----------
2185      %      sin polar d(azmth)    r d(azmth)
2186      %
```

```
2187    %                          f'(polar) d(polar)
2188    %                        = ------------------
2189    %                          f (polar) d(azmth)
2190    %
2191    %   (where f' is the first derivative), rearrange, and integrate to
2192    %  obtain
2193    %
2194    %                      polar   1 - cos polar
2195    %       f (polar) = tan ----- = ------------- ,
2196    %                        2       sin polar
2197    %
2198    % up to an arbitrary multiplicative constant.  The projected Cartesian
2199    % coordinates of a point (cx, cy, cz) on the sphere are
2200    %
2201    %       u = r cos azmth = R cx
2202    %
2203    %       v = r sin azmth = R cy
2204    %
2205    % where
2206    %
2207    %           f (polar)         1           1
2208    %       R = --------- = ------------- = ------ .
2209    %           sin polar   1 + cos polar   1 + cz
2210    %
2211    % For points outside visible space (that is, for cx^2 + cy^2 > 1), cz
2212    % is zero so that R = 1 and no transformation is applied.
2213    %
2214    % The center of projection is opposite boresight (cx = cy = 0, cz =
2215    % -1), and with the above choice of multiplicative constant, the plane
2216    % of projection is the cz = 0 plane.
2217    %
2218    % Orthographic projection:
2219    %
2220    % The orthographic projection gives a 3D view of the hemisphere.
2221    %
2222    if 1                      % INPUT 1 to plot, 0 to skip
2223      proj = 1;               % INPUT 1 for 2D equal-area Lambert
2224                              %       2 for 2D stereographic
2225                              %       3 for orthographic (3D hemisphere)
2226      coordSys = 1;           % INPUT 1 for grid of spherical coordinates
2227                              %       2 for grid of traditional coordinates
2228      faceColor = 'flat';     % INPUT 'flat' or 'interp' shading
2229      pointZoom =  0;         % INPUT magnification factor or 0 for centered full view
2230      % The "show" inputs below are coded only for 2D views.
2231      showBeamRegion   = 0;  % INPUT 1 to show main beam region
2232      showWidthRegion  = 0;  %                region above width contour
2233      showWidthContAct = 0;  %                actual width contour
2234      showWidthContFit = 0;  %                fitted width contour (ellipse)
2235      showPointGrid    = 0;  %                main beam peak on sampled grid
2236      showPoint        = 1;  %                interpolated main beam peak (pointing vector)
2237      showPointUnc     = 0;  %                axes of uncertainty ellipse of pointing vector
2238      showSidelobeGrid = 0;  %                sidelobe peaks on sampled grid
2239      showSidelobeNrst = 1;  %                nearest sidelobe
2240      showSidelobeLgst = 1;  %                largest sidelobe
2241      %
2242      % Prepare for plotting
2243      %
2244      if strcmp (faceColor, 'interp')
2245        dirCosXMtxSurf = dirCosXMtx;                  % use true grid
2246        dirCosYMtxSurf = dirCosYMtx;
2247        dirCosZMtxSurf = dirCosZMtx;
2248        visBoolSurf    = visBool   ;
2249      elseif strcmp (faceColor, 'flat')
2250        dirCosXMtxSurf = ones (ay, 1) * dirCosShiftX;  % use shifted grid to center
2251        dirCosYMtxSurf = dirCosShiftY * ones (1, ax);  %   patches on data points
2252        radSqr = dirCosXMtxSurf.^2 + dirCosYMtxSurf.^2;
2253        visBoolSurf = (radSqr < 1);
2254        dirCosZMtxSurf = zeros (ay, ax);
2255        dirCosZMtxSurf (visBoolSurf) = sqrt (1 - radSqr (visBoolSurf));
2256        clear radSqr;
2257      else
2258        error ('Illegal value of faceColor.');
2259      end
2260      if strcmp (faceColor, 'interp')             % identify visible points plus those immediately
2261        gBlnkIndx = zeros (ay, ax);               %   or diagonally adjacent (Boolean for now)
2262        gBlnkIndx (1:ay-1, 1:ax-1) = visBool (2:ay, 2:ax);
2263        gBlnkIndx (1:ay-1, 2:ax  ) = gBlnkIndx (1:ay-1, 2:ax  ) | visBool (2:ay  , 1:ax-1);
2264        gBlnkIndx (2:ay  , 1:ax-1) = gBlnkIndx (2:ay  , 1:ax-1) | visBool (1:ay-1, 2:ax  );
2265        gBlnkIndx (2:ay  , 2:ax  ) = gBlnkIndx (2:ay  , 2:ax  ) | visBool (1:ay-1, 1:ax-1);
2266      else                                        % identify visible points plus those immediately
2267        gBlnkIndx = visBoolSurf;                  %   but not diagonally adjacent (Boolean for now)
2268        gBlnkIndx (1:ay-1, 1:ax  ) = gBlnkIndx (1:ay-1, 1:ax  ) | visBoolSurf (2:ay  , 1:ax);
2269        gBlnkIndx (1:ay-1, 1:ax-1) = gBlnkIndx (1:ay-1, 1:ax-1) | visBoolSurf (2:ay  , 2:ax);
2270        gBlnkIndx (1:ay  , 1:ax-1) = gBlnkIndx (1:ay  , 1:ax-1) | visBoolSurf (1:ay  , 2:ax);
2271      end
2272      gBlnkIndx = ~gBlnkIndx;                     % 0 in visible region plus appropriate border
2273      gZeroIndx = (gSqr == 0);                    % 1 where zero, 0 elsewhere (Boolean for now)
2274      gOKIndx = find (~(gBlnkIndx | gZeroIndx));  % 1 for nontrivial values
2275      gZeroIndx = find (gZeroIndx);               % convert to indices
2276      gBlnkIndx = find (gBlnkIndx);
2277      if 1                                        % INPUT 0 for linear, 1 for decibel plot
2278        % Decibel plot
```

```
2279          gPlot = zeros (ay, ax);
2280          clim = [-50 0];                                % INPUT dB range
2281          gPlot (gOKIndx) = max (clim (1), 10 * log10 (gSqr (gOKIndx)));
2282          gPlot (gZeroIndx) = clim (1) * ones (size (gZeroIndx));  % condition log of 0
2283        else
2284          % Linear plot
2285          gPlot = gSqr;
2286          clim = [0 1];
2287        end
2288        if invertBkgd                                    % set appropriate value for invisible points
2289          gPlot (gBlnkIndx) = clim (2) * ones (size (gBlnkIndx));
2290        else
2291          gPlot (gBlnkIndx) = clim (1) * ones (size (gBlnkIndx));
2292        end
2293        clear visBoolSurf gBlnkIndx gZeroIndx gOKIndx;
2294        figure (figPat);
2295        fore = get (figPat, 'defaultLineColor');     % expect black or white
2296        if invertBkgd  % choose fore- and background colors compatible with color map
2297          fore = 1 - fore;                               % swap
2298        end
2299        back = 1 - fore;
2300        switch coordSys
2301          case 1
2302            grid1Polar = (0 : 10 :  90) * rpd;       % lines of constant polar angle
2303            grid1Azmth = (0 :  5 : 360) * rpd - azmthOffst;
2304            grid2Polar = (0 :  5 :  90) * rpd;       % lines of constant azimuth
2305            grid2Azmth = (0 : 30 : 360) * rpd - azmthOffst;
2306            [grid1Polar, grid1Azmth] = meshgrid (grid1Polar, grid1Azmth);
2307            [grid2Azmth, grid2Polar] = meshgrid (grid2Azmth, grid2Polar);
2308          case 2
2309            grid1Az = (-90 : 10 : 90) * rpd;         % lines of constant azimuth
2310            grid1El = (-90 :  5 : 90) * rpd;
2311            grid2Az = (-90 : 10 : 90) * rpd;         % lines of constant elevation
2312            grid2El = (-90 : 10 : 90) * rpd;
2313            [grid1Az, grid1El] = meshgrid (grid1Az, grid1El);
2314            [grid2El, grid2Az] = meshgrid (grid2El, grid2Az);
2315            grid1Polar = acos  (cos (grid1El) .* cos (grid1Az));
2316            grid1Azmth = atan2 (tan (grid1El),  -sin (grid1Az)) - azmthOffst;
2317            grid2Polar = acos  (cos (grid2El) .* cos (grid2Az));
2318            grid2Azmth = atan2 (tan (grid2El),  -sin (grid2Az)) - azmthOffst;
2319          otherwise
2320            error ('Invalid value of coordSys.');
2321        end
2322        %
2323        % Plot array factor
2324        %
2325        if proj == 1
2326          %
2327          % Lambert projection
2328          %
2329          radFactSurf = 1 ./ sqrt (1 + dirCosZMtxSurf);  % radius factor for surface plot
2330          hSurf = surf (radFactSurf .* dirCosXMtxSurf, ...
2331                        radFactSurf .* dirCosYMtxSurf, gPlot);
2332          grid1RadFact = sqrt (2) * sin (grid1Polar / 2);
2333          line (grid1RadFact .* cos (grid1Azmth), ...
2334                grid1RadFact .* sin (grid1Azmth), ...
2335                clim (2) * ones (size (grid1RadFact)), 'color', fore);
2336          grid2RadFact = sqrt (2) * sin (grid2Polar / 2);
2337          line (grid2RadFact .* cos (grid2Azmth), ...
2338                grid2RadFact .* sin (grid2Azmth), ...
2339                clim (2) * ones (size (grid2RadFact)), 'color', fore);
2340          set (gca, 'drawmode', 'fast');                  % no hidden objects to worry about
2341          xylim = [-1 1];
2342          zlim = clim;
2343          viewAz = 0;                                     % Matlab azimuth and
2344          viewEl = 90 * rpd;                              %   elevation (but in radians)
2345        elseif proj == 2
2346          %
2347          % Stereographic projection
2348          %
2349          radFactSurf = 1 ./ (1 + dirCosZMtxSurf);        % radius factor for surface plot
2350          hSurf = surf (radFactSurf .* dirCosXMtxSurf, ...
2351                        radFactSurf .* dirCosYMtxSurf, gPlot);
2352          grid1RadFact = tan (grid1Polar / 2);
2353          line (grid1RadFact .* cos (grid1Azmth), ...
2354                grid1RadFact .* sin (grid1Azmth), ...
2355                clim (2) * ones (size (grid1RadFact)), 'color', fore);
2356          grid2RadFact = tan (grid2Polar / 2);
2357          line (grid2RadFact .* cos (grid2Azmth), ...
2358                grid2RadFact .* sin (grid2Azmth), ...
2359                clim (2) * ones (size (grid2RadFact)), 'color', fore);
2360          set (gca, 'drawmode', 'fast');                  % no hidden objects to worry about
2361          xylim = [-1 1];
2362          zlim = clim;
2363          viewAz = 0;                                     % Matlab azimuth and
2364          viewEl = 90 * rpd;                              %   elevation (but in radians)
2365        elseif proj == 3
2366          %
2367          % Orthographic projection (3D hemisphere)
2368          %
2369          float = 1.02;  % radius of annotations relative to unit hemisphere
2370          hSurf = surf (dirCosXMtxSurf, dirCosYMtxSurf, dirCosZMtxSurf, gPlot);
```

```
2371        grid1RadFact = sin (grid1Polar);
2372        line (grid1RadFact .* cos (grid1Azmth) * float, ...
2373              grid1RadFact .* sin (grid1Azmth) * float, ...
2374              cos (grid1Polar) * float, 'color', fore);
2375        grid2RadFact = sin (grid2Polar);
2376        line (grid2RadFact .* cos (grid2Azmth) * float, ...
2377              grid2RadFact .* sin (grid2Azmth) * float, ...
2378              cos (grid2Polar) * float, 'color', fore);
2379        set (gca, 'drawmode', 'normal');              % remove hidden objects
2380        xylim = [-1 1] * float;
2381        zlim  = [-1 1] * float;
2382        switch 1  % INPUT 1 to look down main beam; 2, down boresight; 3, custom
2383          case 1
2384            % Look down main beam
2385            viewAz = pi/2 + steerAzmth + azmthOffst;  % Matlab azimuth and
2386            viewEl = pi/2 - steerPolar;               %   elevation (but in radians)
2387          case 2
2388            % Look down boresight
2389            viewAz = 0;                               % Matlab azimuth and
2390            viewEl = 90 * rpd;                        %   elevation (but in radians)
2391            set (gca, 'drawmode', 'fast');            % no hidden surfaces
2392          otherwise
2393            % Look somewhere
2394            viewAz = 60 * rpd;  % INPUT custom view azimuth
2395            viewEl = 30 * rpd;  %   and elevation (Matlab coordinates)
2396        end
2397      end  % plotting
2398      %
2399      % Annotate plot (2D only)
2400      %
2401      if (proj == 1) | (proj == 2)
2402        switch proj  % set appropriate radius factor
2403          case 1
2404            radFact = 1 ./ sqrt (1 + dirCosZMtx);
2405          case 2
2406            radFact = 1 ./ (1 + dirCosZMtx);
2407        end
2408        if showBeamRegion & beamExist
2409          line (radFact (beamIndx) .* dirCosXMtx (beamIndx), ...
2410                radFact (beamIndx) .* dirCosYMtx (beamIndx), ...
2411                clim (2) * ones (size (beamIndx)), ...
2412                'linestyle', 'none', 'marker', '+', 'color', fore);
2413        end
2414        if showWidthRegion & capClosed
2415          line (radFact (capIndx) .* dirCosXMtx (capIndx), ...
2416                radFact (capIndx) .* dirCosYMtx (capIndx), ...
2417                clim (2) * ones (size (capIndx)), ...
2418                'linestyle', 'none', 'marker', 'x', 'color', fore);
2419        end
2420        if showWidthContAct & capClosed
2421          switch proj
2422            case 1
2423              radFactCapAct = 1 ./ sqrt (1 + capContZ);
2424            case 2
2425              radFactCapAct = 1 ./ (1 + capContZ);
2426          end
2427          line (radFactCapAct .* capContX, radFactCapAct .* capContY, ...
2428            clim (2) * ones (size (capContX)), 'linestyle', ':', 'color', back);
2429          clear radFactCapAct
2430        end
2431        if showWidthContFit & capClosed
2432          switch proj
2433            case 1
2434              radFactCapFit = 1 ./ sqrt (1 + cFitZ);
2435            case 2
2436              radFactCapFit = 1 ./ (1 + cFitZ);
2437          end
2438          line (radFactCapFit .* cFitX, radFactCapFit .* cFitY, ...
2439            clim (2) * ones (size (cFitX)), 'linestyle', '-', 'color', back);
2440          clear radFactCapFit
2441        end
2442        if showPointGrid
2443          line (radFact (gSqrMaxRow, gSqrMaxCol) .* dirCosXMtx (gSqrMaxRow, gSqrMaxCol), ...
2444                radFact (gSqrMaxRow, gSqrMaxCol) .* dirCosYMtx (gSqrMaxRow, gSqrMaxCol), ...
2445                clim (2), 'linestyle', 'none', 'marker', '*', 'color', back);
2446        end
2447        if showPointUnc
2448          switch proj
2449            case 1
2450              pointRadFact = 1 / sqrt (1 + pz);
2451              Jacob = [1+pz+px^2/pz      px*py/pz                          % Jacobian of
2452                       py*px/pz  1+pz+py^2/pz] / (2*(1+pz)^(3/2));  %   projection
2453            case 2
2454              pointRadFact = 1 / (1 + pz);
2455              Jacob = [1+pz+px^2/pz      px*py/pz                          % Jacobian of
2456                       py*px/pz  1+pz+py^2/pz] / (1+pz)^2;          %   projection
2457          end
2458          pVarProj = Jacob * pVar * Jacob';  % covariance matrix in this projection
2459          [pEigVec, pEigVal] ...             % diagonalize: pVar
2460            = eig (pVarProj);                %   = pEigVec * pEigVal * pEigVec'
2461          pPrAx = 2 * pEigVec ...            % scale principal axes (columns of
2462            * sqrt (pEigVal);                %   pEigVec) to 2 standard deviations
```

```
2463            line (pointRadFact * px + [-1 1]' * pPrAx (1, :), ...
2464                  pointRadFact * py + [-1 1]' * pPrAx (2, :), ...
2465                  clim (2) * [1 1], ...
2466                  'linestyle', '-', 'color', back);
2467            clear pointRadFact Jacob pVarProj pEigVec pEigVal pPrAx
2468          elseif showPoint
2469            switch proj
2470              case 1
2471                pointRadFact = 1 / sqrt (1 + pz);
2472              case 2
2473                pointRadFact = 1 / (1 + pz);
2474            end
2475            line (pointRadFact * px, pointRadFact * py, clim (2), ...
2476              'linestyle', 'none', 'marker', '.', 'color', back)
2477            clear pointRadFact
2478          end
2479          if showSidelobeGrid
2480            line (radFact (slIndx) .* dirCosXMtx (slIndx), ...
2481                  radFact (slIndx) .* dirCosYMtx (slIndx), ...
2482                  clim (2) * ones (size (slIndx)), ...
2483                  'linestyle', 'none', 'marker', 's', 'color', back);
2484          end
2485          if showSidelobeNrst
2486            switch proj
2487              case 1
2488                radFactSlNrst = 1 / sqrt (1 + slNrstVec (3));
2489              case 2
2490                radFactSlNrst = 1 / (1 + slNrstVec (3));
2491            end
2492            line (radFactSlNrst .* slNrstVec (1), ...
2493                  radFactSlNrst .* slNrstVec (2), ...
2494                  clim (2), 'linestyle', 'none', 'marker', '+', 'color', back);
2495            clear radFactSlNrst
2496          end
2497          if showSidelobeLgst
2498            switch proj
2499              case 1
2500                radFactSlLgst = 1 / sqrt (1 + slLgstVec (3));
2501              case 2
2502                radFactSlLgst = 1 / (1 + slLgstVec (3));
2503            end
2504            line (radFactSlLgst .* slLgstVec (1), ...
2505                  radFactSlLgst .* slLgstVec (2), ...
2506                  clim (2), 'linestyle', 'none', 'marker', 'x', 'color', back);
2507            clear radFactSlLgst
2508          end
2509        end  % annotations
2510        %
2511        % Arrange graphics properties
2512        %
2513        axesPat = gca;
2514        rotate (get (axesPat, 'children'), ...  % rotate plotted objects to
2515          [0 0 1], azmthOffst / rpd);           %   compensate for azmthOffst
2516        set (hSurf, 'edgecolor', 'none');
2517        set (hSurf, 'facecolor', faceColor);
2518        colormap (cmap);
2519        set (axesPat, 'clim', clim);
2520        if cbarVert                  % position the colorbar
2521          axesCbar = colorbar ('vert');
2522          set (axesCbar, ...
2523            'units', 'normalized', ...
2524            'position', [(1+0.2*cbarSize)/(1+cbarSize) 0.05 0.4*cbarSize/(1+cbarSize) 0.9]);
2525          set (axesPat, ...
2526            'units', 'normalized', ...
2527            'position', [0 0 1/(1+cbarSize) 1]);
2528        else
2529          axesCbar = colorbar ('horiz');
2530          set (axesCbar, ...
2531            'units', 'normalized', ...
2532            'position', [0.05 0.3*cbarSize/(1+cbarSize) 0.9 0.5*cbarSize/(1+cbarSize)]);
2533          set (axesPat, ...
2534            'units', 'normalized', ...
2535            'position', [0 cbarSize/(1+cbarSize) 1 1/(1+cbarSize)]);
2536        end
2537        set (figPat, 'children', ...  % put color bar in front of pattern but let
2538          [axesCbar axesPat]');       %   axesPat remain the current axis
2539        set (axesPat, ...
2540          'xlim', xylim, ...
2541          'ylim', xylim, ...
2542          'zlim', zlim , ...
2543          'dataAspectRatio', diff ([xylim' xylim' zlim']), ...
2544          'visible', 'off', ...
2545          'view', [viewAz/rpd  viewEl/rpd]);
2546        if proj == 3
2547          cameraDist = norm ( ...                          % distance from camera
2548              get (axesPat, 'cameraPosition') ...          %   to the surface
2549            - get (axesPat, 'cameraTarget'), 2);
2550          set (axesPat, 'cameraViewAngle', ...             % set view angle to
2551            2 * atan (float * diff (zlim) ...              %   exactly span the
2552                    / (cameraDist * diff (xylim))) / rpd); %   unit sphere
2553        end
2554        if (pointZoom ~= 0) & peakVisb
```

```
2555          pxRot = cos (azmthOffst) * px - sin (azmthOffst) * py;  % compensate for
2556          pyRot = sin (azmthOffst) * px + cos (azmthOffst) * py;  %   azmthOffst
2557          if proj == 1
2558            pointRadFact = 1 / sqrt (1 + pz);
2559            set (axesPat, 'xlim', pointRadFact * pxRot + [-1 1] / pointZoom, ...
2560                          'ylim', pointRadFact * pyRot + [-1 1] / pointZoom);
2561          elseif proj == 2
2562            pointRadFact = 1 / (1 + pz);
2563            set (axesPat, 'xlim', pointRadFact * pxRot + [-1 1] / pointZoom, ...
2564                          'ylim', pointRadFact * pyRot + [-1 1] / pointZoom);
2565          elseif proj == 3
2566            set (axesPat, 'cameraViewAngle', ...
2567              2 * atan (float / (cameraDist * pointZoom)) / rpd);
2568          end
2569          clear pxRot pyRot pointRadFact
2570        end  % zoom
2571        clear dirCosXMtxSurf dirCosYMtxSurf dirCosZMtxSurf;
2572        clear fore back gridPolar gridAzmth gridAzmthSmth;
2573        clear radFactSurf gridFact viewAz viewEl float radFact;
2574      end
2575
2576      drawnow;
2577
2578    end  % loop over independent variable
2579    clear indx
2580
2581    % Print warnings, if any
2582    %
2583    if any (numAcc < numRlz)
2584      disp ('Warning: at least one realization could not be fully');
2585      disp ('  analyzed; inspect the matfiles in the current directory.');
2586    end
2587
2588    clear cbarSize
2589    clear lx ly mx my nx ny;
2590    clear numLMX numLMY numLMNX numLMNY;
2591    clear tIndx rlzNum;
2592
2593    % end of program
2594
2595    % Suggestions for improvements
2596    %
2597    % Implement non-uniform excitation magnitudes:
2598    %
2599    %   As needed, non-uniform illumination weights may be coded
2600    %   straightforwardly.
2601    %
2602    % Implement element factor and improve integration of radiated powers:
2603    %
2604    %   Currently the element factor is implicitly coded as one everywhere,
2605    %   so that the elements radiate uniformly into the hemisphere.  A more
2606    %   realistic element factor is cos polar [1], being one at broadside
2607    %   and zero at grazing.  Note, however, that in integrating the data
2608    %   over solid angles to calculate radiated powers, we divide the data
2609    %   by cos polar.  Performing this multiplication and division in
2610    %   succession could yield invalid data near grazing, where cos polar is
2611    %   small. Obviously, this difficulty could be avoided by maintaining
2612    %   the unscaled data for use in the integration while using the scaled
2613    %   data for all other processing.  The calculations of the pointing
2614    %   vector from the excitation phases and of the peak power should also
2615    %   account for the element factor.
2616    %
2617    %   More generally, one might wish to apply an arbitrary element factor.
2618    %   If it is small near grazing, the difficulty described above
2619    %   persists.  One solution is to specify the element factor relative to
2620    %   cos polar; the user ensures that all values of that ratio are
2621    %   reasonable.  The data used in the integration are scaled by the
2622    %   ratio, while the data used elsewhere are scaled by both the ratio
2623    %   and cos polar.  One could also allow element factors defined over
2624    %   the entire sphere, including radiation into z < 0, perhaps by
2625    %   storing the back radiation pattern in a second g matrix and
2626    %   modifying the analysis routines.
2627    %
2628    %   Broadening our perspective, we note that these problems are
2629    %   ultimately due to the integration algorithm, in that it blindly
2630    %   applies a Jacobian that diverges at grazing.  One consequence is
2631    %   that data cells whose centers are just inside the border of visible
2632    %   space contribute their entire value scaled by a large Jacobian,
2633    %   whereas those whose centers are just outside contribute nothing.
2634    %   More correctly, both should contribute about half of their value
2635    %   scaled according to some average location of the contributing
2636    %   region.  Cells almost entirely outside of visible space should
2637    %   contribute very little.  A better algorithm would reproduce this
2638    %   behavior.
2639    %
2640    % Allow pattern cuts:
2641    %
2642    %   Often one is interested in a cut of the pattern along a path on the
2643    %   unit hemisphere, such as cuts through the main beam along azimuth
2644    %   and elevation curves or along the great circles of maximum and
2645    %   minimum beam width.  If only a graphical presentation is desired,
2646    %   the cut could be interpolated from the pattern.  If an analysis is
```

```
2647    %   also desired, specialized routines would probably be required, as
2648    %   the current routines expect data over two directional coordinates.
2649    %
2650    % Consider alternate calculation of beam widths and roll:
2651    %
2652    %   The beam widths and roll are derived from an ellipse fitted to a
2653    %   level contour of the main beam.  Currently the fit is performed in
2654    %   the boresight-centered stereographic projection regardless of the
2655    %   pointing vector.  This projection preserves the orientation and
2656    %   orthogonality of the major and minor axes of the ellipse.  However,
2657    %   because scale in the stereographic projection increases away from
2658    %   boresight, off-boresight contours are expanded toward the edge of
2659    %   the projection.  Although scale is uniform in all directions for an
2660    %   infinitesimal region, radial scale is exaggerated relative to
2661    %   azimuthal scale for a finite region.  For a broad beam off
2662    %   boresight, the distortion of beam width may be significant.
2663    %
2664    %   An improvement might be achieved with two changes.  First, center
2665    %   the projection on the pointing vector, assuming that the contour
2666    %   will be found to lie centered on the pointing vector as well.
2667    %   Second, instead of the stereographic projection, use the azimuthal
2668    %   equidistant projection, for which distances measured on a line
2669    %   passing through the center of the projection are true.  Regardless
2670    %   of beam width, the lengths of the major and minor axes of the
2671    %   projected ellipse will equal those of the ellipse on the hemisphere,
2672    %   provided that the center of the ellipse is also the center of the
2673    %   projection.
2674    %
2675    %   In centering the projection on the pointing vector, one is free to
2676    %   choose the orientation of the projection relative to the spherical
2677    %   coordinate system, and a judicious choice of this angle facilitates
2678    %   calculation of the beam's roll angle.  Construct at boresight on the
2679    %   hemisphere two tangent axes u and v; let positive u be directed
2680    %   toward spherical azimuth zero and positive v toward pi/2.  Next,
2681    %   construct the arc connecting boresight with the pointing vector.
2682    %   Translate the u-v origin and system along this arc without rotation
2683    %   in the plane locally tangent to the hemisphere, so that u, v, and
2684    %   the arc maintain the same local orientation.  If the ellipse is
2685    %   centered on the pointing vector, the roll angle will be the angle
2686    %   from the positive u axis to the major axis.
2687    %
2688    % Allow linear arrays:
2689    %
2690    %   The current analysis routines cannot handle linear arrays because of
2691    %   several incompatibilities.  For example, the main beam of
2692    %   one-dimensional arrays with isotropic element patterns is a cone
2693    %   about the axis of the array, so the direction of radiation is
2694    %   specified by a single number -- the angle between the axis and the
2695    %   cone.  However, the current analysis routines seek a two-parameter
2696    %   specification of the main beam direction and will generally fail.
2697    %   Likewise, the beam width is described by one number, but the current
2698    %   routines seek two parameters.  Furthermore, analyses that depend on
2699    %   these values (such as determining the proximity of sidelobes) will
2700    %   also fail.  If linear arrays are if interest, the analysis routines
2701    %   must be expanded.  One might also add graphics routines tailored to
2702    %   one-dimensional arrays.
2703    %
2704    %   Note that a non-isotropic element pattern will generally produce a
2705    %   variation in the direction orthogonal to the main beam contour,
2706    %   allowing the analysis routines to proceed. Results thereby obtained
2707    %   should be interpreted accordingly.  (Using a non-isotropic element
2708    %   pattern will not benefit the routine that locates the pointing
2709    %   vector from the excitation phases.  One might ignore its results
2710    %   when the final pointing vector is calculated, using only the
2711    %   pointing vector obtained from the Fourier transform.)
2712    %
2713    %   [1] R. Tang and R. W. Burns, "Phased Arrays," in _Antenna
2714    %   Engineering Handbook_, 3rd ed., R. C. Johnson, Ed.  New York, NY:
2715    %   McGraw-Hill, 1993, Ch. 20, Sec. 3.
```

REFERENCES

1. R. J. Mailloux, "Array grating lobes due to periodic phase, amplitude, and time delay quantization," *IEEE Trans. Antennas and Propagation*, vol. 32, pp. 1364–1368, Dec. 1984.
2. A. P. Goutzoulis and D. K. Davies, "Switched fiber optic delay architectures," in *Photonic Aspects of Modern Radar*, H. Zmuda and E. N. Toughlian, Eds. Boston, MA: Artech House, 1994, pp. 351–380.
3. A. K. Agrawal and E. L. Holzman, "Beamformer architectures for active phased-array radar antennas," *IEEE Trans. Antennas and Propagation*, vol. 47, pp. 432–442, Mar. 1999.
4. R. J. Mailloux, *Phased Array Antenna Handbook*, ch. 7. Norwood, MA: Artech House, 1994.
5. R. C. Hansen, *Phased Array Antennas*, Sec. 12.4. New York: John Wiley & Sons, 1998.
6. B. D. Steinberg, *Principles of Aperture and Array System Design*, ch. 13. New York: John Wiley & Sons, 1976.
7. B. P. Chrisman, "Planar array antenna design analysis," in *Proc. Tactical Communications Conf.*, 1990, vol. 1, pp. 705–731.
8. P. J. Wright, "Simulation of phased array antennas," in *Tenth International Conf. on Antennas and Propagation*, 1997, vol. 1, pp. 498–501.
9. A. K. Agrawal, E. L. Williamson, and J. G. Ferrante, "Design criteria for wideband active phased array antennas," in *IEEE Antennas and Propagation Society International Symposium 1997 Digest*, vol. 2, pp. 714–717.
10. J. J. Lee et al., "Photonic wideband array antennas," *IEEE Trans. Antennas and Propagation*, vol. 43, pp. 966-982, Sept. 1995.
11. The MathWorks, Inc., 3 Apple Hill Dr., Natick, MA 01760-2098.
12. J. P. Snyder, *Map Projections: A Working Manual*, US Geological Survey Professional Paper 1395. Washington, DC: United States Government Printing Office, 1987.
13. D. H. von Seggern, *CRC Standard Curves and Surfaces*. Boca Raton, FL: CRC, 1993.
14. R. S. Elliott, "Beamwidth and directivity of large scanning arrays," *Microwave J.*, vol. 7, pp. 74–82, Jan. 1964.
15. R. S. Elliott, "The Theory of Antenna Arrays," in *Microwave Scanning Antennas*, vol. 2, R. C. Hansen, Ed. New York: Academic, 1966, pp. 1–69.