AN INVESTIGATION OF MULTIPATH EFFECTS ON THE GPS SYSTEM DURING AUTO-RENDEZVOUS AND CAPTURE

ROVE Final Report

James [illegible]hie, Ph.D. and Francis W. Forest
Department of Electrical and Computer Engineering
Marquette University
1515 W. Wisconsin Ave.
Milwaukee, WI 53233

March 1995

Table of Contents

Executive Summary

The scattering from a space station cylinder has been modeled using physical optics with no assumptions except the form of the current induced on the surface of the cylinder. The computations were performed over a typical remotely piloted CTV auto-rendezvous and capture trajectory. During most of this trajectory, the GPS system will be used to remotely pilot the CTV. The optical guidance system will take over sometime before there is 100 meters between the CTV and SSF.

Simulation Methods

Data has been collected using this simulation model for a span of possible geometries. The geometry is defined by (1) the angle between the GPS satellite and the axis of the cylinder, and the angles (referenced by the plane containing the GPS satellite and the SSF) from SSF to the earth. Once the scattered field is found, the RCP component in the direction of the CTV is determined and normalized relative to the signal strength of the direct signal.

Results

The data indicates that the path difference between the direct signal and the scattered signal is less than 300 meters during the final stages of AR&C maneuvers, and during isolated sections of the trajectory when the GPS satellite, CTV, and SSF are approximately collinear. The scattered RCP signal rises slowly during the final stages of AR&C maneuvers to maximum values of approximately +2 dB or -4 dB for end-on approach and side approach, respectively. These values are only for particular locations of the GPS satellite, and the data for other locations are 10 to 20 dB below this maximum.

During the earlier stages of AR&C maneuvers when the three objects are roughly collinear, there are three scattering mechanisms: specular reflection; forward scattering; and forward scattering with an anomalous peak. Specular reflections have peaks within the range of -20 dB to -30 dB. Forward scattering is smaller with maximum values near -35 dB. The scattering near the anomalous peak has a maximal value of -44 dB.

Alternative Simulation Techniques

A method of moments solution for a two-dimensional cylinder has been computed. The method of moments makes no approximations. The results indicate that the physical optics current is a good approximation when compared to the exact numerical solution. However, only one linear polarization has been investigated, and it is well known that the other linear polarization (to create RCP) has a strong discontinuity. This is the source of the anomalous peak mentioned above.

The use of GTD and PTD has also been investigated. It has been found that GTD would not be appropriate due to the large distances involved. The use of PTD is a natural extension to the work accomplished in this report since PTD is a correction to the physical optics approximation and does not require smaller distances.

1. Introduction, Summary and Key Conclusions

1.1 Introduction

The proposed use of a Cargo Transport Vehicle (CTV) to carry hardware to the Space Station Freedom (SSF) during the construction phase of the SSF project requires remote maneuvering of the CTV. The CTV is not a manned vehicle. Obtaining the relative positions of the CTV and SSF for remote auto-rendezvous and capture (AR&C) scenarios will rely heavily on the Global Positioning System (GPS). The GPS system is expected to guide the CTV up to a distance of 100 to 300 meters from the SSF. At some point within this range, an optical docking system will take over the remote guidance for capture.

During any remote guidance by GPS it is possible that significant multipath signals may be caused by large objects in the vicinity of the module being remotely guided. This could alter the position obtained by the GPS system from the actual position [1]. Due to the nature of the GPS signals, it has been estimated that if the difference in distance between the Line of Sight (LOS) path and the multipath is greater than 300 meters, the GPS system is capable of discriminating between the direct signal and the reflected (or multipath) signal. However, if the path difference is less than 300 meters, one must be concerned.

This report details the work accomplished by the Electromagnetic Simulations Laboratory at Marquette University over the period December 1993 to May 1995. This work is an investigation of the strength and phase of a multipath signal arriving at the CTV relative to the direct or line of sight (LOS) signal. The signal originates at a GPS satellite in half geo-stationary orbit and takes two paths to the CTV: (1) the direct or LOS path from the GPS satellite to the CTV; and (2) a scattered path from the GPS satellite to the SSF module and then to the CTV.

1.1.1 *Background*

The path that the CTV takes on approach to SSF is shown in Figure 1-1. The initial phase brings the CTV within 2 km in altitude and 37 km behind the SSF orbit. The SSF orbit is represented by the line V_{BAR}. V_{BAR} is actually not a straight line, but can be represented as such for this discussion. Since the CTV is at a lower altitude, it will orbit the earth faster than SSF. Thus, the next phase of approach is parallel to V_{BAR} for approximately 37 km.

The most fuel-efficient maneuvering is done by burns that result in 2:1 ellipses relative to V_{BAR} with the major axis of the ellipse parallel to V_{BAR}. The first CTV burn results in an elliptical orbit up to V_{BAR}. However, due to the motion of SSF during this burn, the relative trajectory of CTV is assumed to be circular. After this burn, CTV is on V_{BAR}, and is 2 km ahead of SSF.

The remaining steps on approach are two elliptical burns, one that brings CTV within 300 meters of SSF, and another that places the CTV within 100 meters of SSF. The final approach is taken directly along V_{BAR} [2].

The global positioning system consists of a number of satellites at half geo-stationary orbit with highly accurate clocks. A minimum of four readings are usually necessary for a receiver to determine its location. The satellites used could be located anywhere in the LOS view of the receiver. Since the GPS satellites are at half geo-stationary orbit, they are also moving; however, the velocity of each is small enough to be ignored [3].

The GPS signals consist of a spread spectrum sequence or pseudo-random (pr) code that holds important information and is typically a very weak signal. Extensive filtering is used to determine the data transmitted along with the time of arrival. The carrier frequencies of interest are $L_1 = 1.57542$ GHz and $L_2 = 1.22760$ GHz [4]. In this report, L1, L2, shall be denoted as L_1, L_2, respectively.

The presence of a large object in the vicinity of a GPS receiver (such as on CTV) will scatter the GPS signal from the satellite in many directions. A portion of this will also reach the CTV. This will result in a delayed copy of the original signal at the GPS receiver on the CTV. The delayed copy is called a multipath (M/P) signal. The delay of this signal can be estimated as the time difference between the direct signal and the delayed signal. The time difference can be converted to a distance using the speed of light in vacuum. This distance shall be denoted as the path difference.

If the path difference is greater than 300 meters, the GPS receiver is able to overcome the effect of multipath via filtering. If the path difference is less than 300 meters, the multipath signal may cause an incorrect location reading on the receiver [5]. Given the relative distances between and velocities of the SSF and the CTV, it is essential that errors be avoided.

It is also true that the strength of the multipath signal relative to the direct signal is very important. If the multipath signal is too strong, then the GPS receiver will incorrectly determine its location. While it is presently unclear how strong a signal must be to cause an error, a study of how strong the signal shall be is important to the determination of how safe AR&C maneuvers will be.

Determination of the magnitude and phase of multipath during AR&C can be simulated using a GPS signal simulator [1,2]. By presenting the simulator with this information, an estimate from the receiver of its location can be verified. This investigation would provide valuable information regarding the difference between "dangerous" and insignificant multipath.

1.1.2 *Objectives*

The objectives of the work accomplished were to obtain the strength and phase of the multipath signal relative to the direct signal from the GPS satellite. This has been accomplished for a single cylinder the size of an SSF module.

1.1.3 *Methods*

In this section the approach used to estimate the strength of the multipath signal is reviewed. The solution is found along the CTV trajectory and includes the polarization of the reflected wave reaching the CTV, since the receiver is polarization sensitive. In addition, the signal is found relative to the LOS or direct path. No assumptions are made here regarding the locations of SSF, CTV or the GPS satellites.

The problem is fundamentally a bistatic radar cross section (RCS) problem. In radar, a signal is transmitted, scattered off a target and this scattered field is then used to locate the target [6]. In this work, the transmitted signal is the signal from the GPS satellite, the target is the SSF module, and the receiver is the CTV. However, for the purposes of this work, the important information is not the location of the target, but instead how strong the scattered signal is when compared to the direct signal.

Techniques for computing the scattered field from an object are generally broken into various groups according to the size of the scatterer in wavelengths. Very small objects can be approximated as a dipole moment with strength and direction [7]. As the object becomes larger, up to a few wavelengths, the method of moments (MoM) is applicable [8]. Once the size reaches the upper boundary of MoM, the finite difference-time domain (FD-TD) [9] method can be used. However, the size of the SSF is far too large for any of these techniques to be practical on presently available computers. Note that the SSF modules are over 10λ in radius and over 36λ in length (where λ is the wavelength).

For very high frequencies (as k approaches infinity, where k is the wavenumber $2\pi/\lambda$), other techniques are used based on the asymptotic expansion of Maxwell's equations. This family of methods, known as GTD, UTD, and UAT [10-12], are based on corrections to geometrical optics [13]. The advantages and disadvantages are discussed in detail in Section 4.2. This approach was not chosen and would not be applicable due to the large distances involved.

An alternative approach that is very widely used in RCS computations from large ships and aircraft is physical optics (PO). In PO [14], the current that is induced on the surface of a perfectly conducting scatterer is approximated as:

$$\boldsymbol{J}_s \approx 2\,\hat{n}\times\boldsymbol{H}^i \qquad (1)$$

for the illuminated portions of the object, and $\mathbf{J}_s$ is zero elsewhere. $\mathbf{H}^i$ is the incident magnetic field and $\hat{\mathbf{n}}$ is the outward normal unit vector from the surface. Once $\mathbf{J}_s$ is found, the scattered electric field $\mathbf{E}^s(\mathbf{r})$ due to $\mathbf{J}_s(\mathbf{r}')$ is:

$$\boldsymbol{E}^s(\boldsymbol{r}) = \frac{1}{4\pi j\omega\varepsilon_0}\nabla\times\nabla\times\int_{S'}\frac{\left[\boldsymbol{J}_s(\boldsymbol{r}')\right]e^{-jk|\boldsymbol{r}-\boldsymbol{r}'|}}{|\boldsymbol{r}-\boldsymbol{r}'|}\,ds' \qquad (2)$$

where $\omega = 2\pi f$ (f is the frequency), ϵ_0 is the permittivity of free space, $\mathbf{r}$ is a vector from the origin to the observation point, $\mathbf{r}'$ is a vector from the origin to a point of current, S' is the surface of the object, and the curl operations only act on the unprimed coordinates.

The direction of propagation of the scattered field will be used to compute the RCP component of the scattered field. This will allow comparison of the direct signal and the M/P signal.

The path of the CTV has been parameterized in a relative coordinate system (relative to the SSF cylinder). The code steps through the trajectory of the CTV and computes the M/P signal. If the path difference between direct and multipath signals is greater than 300 meters, the computation is not performed. The code returns zero multipath signal strength. The complete code is provided in Appendix 7.1.

Section 4 of this report discusses alternative approaches to the basic electromagnetic problem. A two dimensional simulation of scattering from a cylinder is shown to provide further insight into the scattering mechanisms present for this problem. The solution is found using the exact integral equation (electric field integral equation, or EFIE) in two dimensions [15]. A discussion of the basis for the solution and the results relevant to the cylinder problem are presented in Section 4.1. Included in this report (Appendix 7.2) is a Technical Report from the Electromagnetic Simulations Laboratory at Marquette University (MU-ESL) which describes the theory and code developed to solve for scattering from two-dimensional perfect conducting objects.

The theory of UTD [10,11] is reviewed in Section 4.2 for informational purposes. Included in this report (Appendix 7.3) is a Technical Report from MU-ESL which describes in great detail the theory and applications of UTD. The theory of the physical theory of diffraction (PTD) [16] is also reviewed in Section 4.3 along with a discussion of the relative applicability of PTD and GTD.

1.2 Summary of Results

1.2.1 *Issues Regarding the Problem Definition*

The scattering from a cylinder has been computed using the physical optics approximation for the current. No other approximations or assumptions have been made including no assumptions regarding the far field or Fresnel field approximations.

The integrations required to obtain the scattered field have been computed numerically using an N dimensional Romberg integration. The total scattered electric field is then projected onto the RCP component in the direction of propagation only. The direct or line of sight signal is then used to compute the relative strength and phase of the scattered field.

The trajectory of the CTV has been parameterized into 4,214 points that are calculated for each of the geometries investigated. The motion of the CTV between points is small enough for the magnitude data (dB down from direct signal) to appear very smooth; however, because of the distances and wavelengths involved, the phase of the scattered field relative to the direct signal varies very rapidly.

1.2.2 *Simulation Results*

The results of this numerical investigation include data for a CTV approaching the SSF with the SSF orbiting with the axis along the direction of travel (parallel axis), and with the axis perpendicular to the direction of travel (perpendicular axis). Figures 2-5(a,b) show the perpendicular and parallel axis cases, respectively. For the parallel axis approach, the data suggest that near the SSF, the scattered signal is 2 dB above the direct signal, whereas for the perpendicular axis approach, the scattered signal is 4 dB below the direct signal.

There are a number of geometrical situations that result in path differences less than 300 meters even though the SSF and CTV are separated by a rather large distance. For these cases, the scattering mechanisms can be broken into three groups: (1) the specular reflection components; (2) the forward scattering components; and (3) an anomalous peak component. These three mechanisms are listed in descending order with respect to strength and in descending degrees of accuracy. In short, the specular reflection component is computed the most precisely and is the strongest component to scattering from the cylinder.

1.2.3 *Alternative Techniques and Extensions of the Present Solution*

A number of alternatives have been investigated with the intent to either reinforce what has been accomplished or to plan for future investigations of possibly more complex problems. The first of these alternatives is the computation of scattering from a circular cylinder using the method of moments in two dimensions. The incident field in this case is linearly polarized. The solution illustrates the usefulness of physical optics and the nature of the corrections obtained when using the physical theory of diffraction.

The use of GTD has been exhaustively studied and Appendix 7.3 is a report from the Marquette University Electromagnetic Simulations Laboratory on GTD and addresses in particular the scattering from a cylinder. In addition, a brief introduction to PTD is presented as a more viable solution method. GTD in fact requires that the source or the observation be close to the scattering object.

1.3 Key Conclusions

1.3.1 *Conditions Pertaining to 300 Meter Path Difference (or less)*

Consider the distances that are involved in the computation of the path difference between the LOS and M/P signal paths. The GPS satellite is 20,240 km above the earth, the SSF is in LEO at 331 km above the earth, and the CTV is rising and approaching SSF by the path shown in Figure 1-1. Computations have been made that determine the relative orientations that result in a path difference of 300 meters or less. The conclusion is that only in cases where the three objects are almost collinear does the path difference become too small. Considering the large distance to the GPS satellite, this is very reasonable. See Figure 1-2. Since the CTV and SSF are very close to each other through much of the approach, it is conceivable that a GPS satellite could be aligned with the line defined between SSF and CTV.

1.3.2 *Multipath Signals Before the CTV Reaches V_{BAR}*

The 300 meter path difference requirement is broken in some places along the trajectory before the CTV reaches V_{BAR}. There are two possibilities: either forward scattering where the scattered field appears to have passed through the SSF, and specular reflection where the scattered field appears to bounce off the SSF. Obviously, all cases of forward scattering imply that the three objects are collinear. The specular reflections can only occur within the 300 meter requirement if the triangle as shown in Figure 1-2 is thin enough.

The specular reflections have nominal values near -35 dB with peaks reaching up to -20 dB. The forward scattering contributions have nominal values that are considerably smaller (-45 dB) with no strong peak. Because of the 300 meter path difference requirement, most of the notable multipath signals have a short duration. The exception to this is the case where the 37 km linear portion of the trajectory is almost in line with the line defined by the GPS satellite and the SSF. For this case, the nominal signal values are roughly -55 dB.

1.3.3 *Multipath Signals Once the CTV Reaches V_{BAR}*

Once the CTV reaches V_{BAR}, the distance between SSF and CTV is still 2 km, which is still in the far field. However, during the first elliptical burn, the CTV enters the Fresnel region, and the data appears less regular, making generalizations such as in the previous section more difficult and less meaningful. However, in this phase of the trajectory, there is an almost monotonic increase in signal level near the -20 dB mark. This value increases at a faster rate once the CTV reaches the final approach portion of the trajectory. At this point, the signal power increases to -4 dB for perpendicular approach or +2 dB for parallel approach. Within roughly 40 meters of the SSF, the CTV is truly in the near field, and no longer in the Fresnel zone. However, the optical guidance system is expected to assume control before the CTV reaches a 100 meter distance from the SSF.

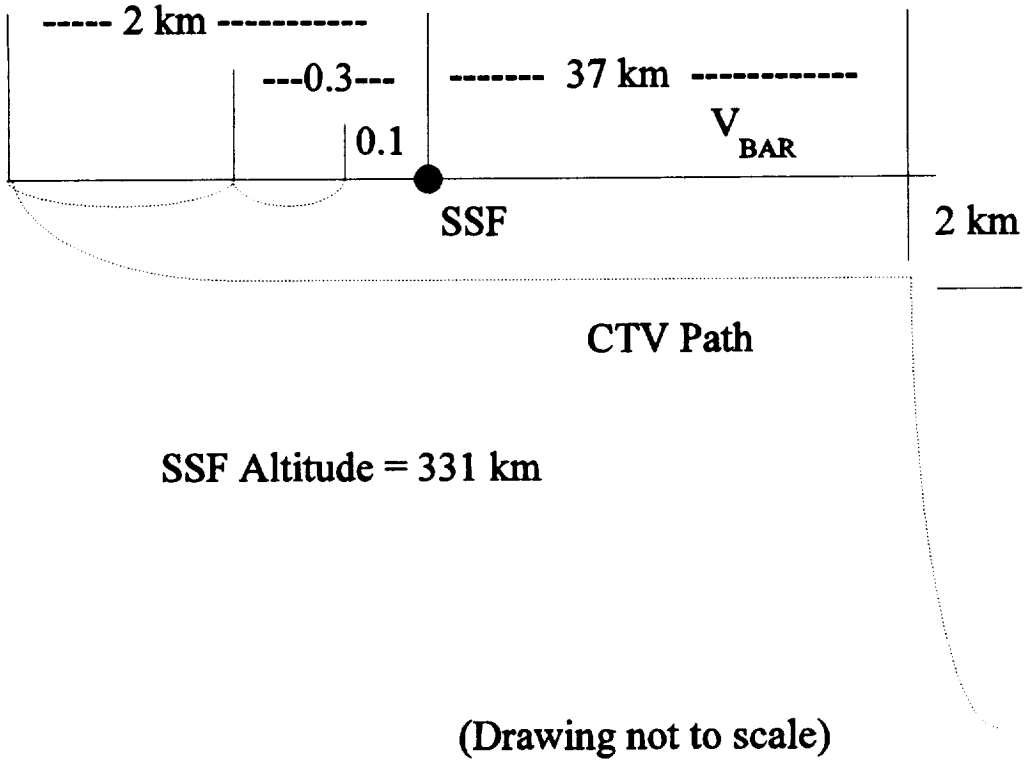


Figure 1-1. Graph illustrating path of CTV toward SSF.

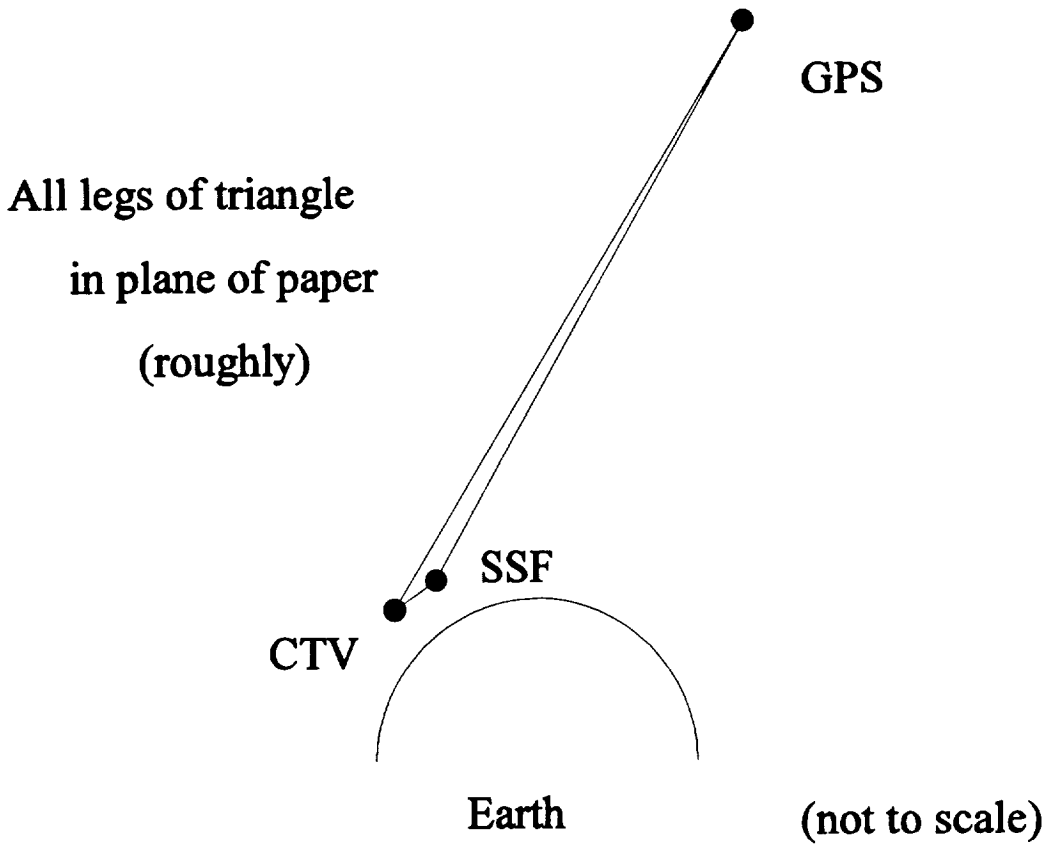


Figure 1-2. Typical scenario where all three objects are nearly collinear.

2. Problem Definition

In this section the methods used to estimate the strength of the multipath signal are developed. The solution includes the polarization of the reflected wave toward the CTV since the receiver is polarization sensitive. This signal strength is compared to the direct signal. The CTV path has also been parameterized with a distance between data points that is small enough to provide smooth results.

2.1 Physical Optics Solution

2.1.1 *On the Far-Field Approximation [17]*

A common approach in obtaining the radiation pattern of an arbitrary antenna is by calculating the electric and magnetic field components from the magnetic vector potential **A**, where:

$$\mathbf{A}(\mathbf{r}) = \frac{\mu_0}{4\pi}\int_{v'} \frac{\mathbf{J}_v(\mathbf{r}')\, e^{-jk|\mathbf{r}-\mathbf{r}'|}}{|\mathbf{r}-\mathbf{r}'|}\, dv' \qquad (1)$$

and $\mathbf{J}_v$ is the electric current density, $|\mathbf{r}-\mathbf{r}'|$ is the distance from a point on the source to the observation point, and μ_0 is the permeability of free space.

In order to simplify the discussion, the radiation pattern from a dipole antenna of finite length will be considered. At some observation point **r**, **A** is given by:

$$|\mathbf{r}-\mathbf{r}'| = R = \sqrt{(x-x')^2 + (y-y')^2 + (z-z')^2} \qquad (2)$$

If $r^2 = x^2 + y^2 + z^2$ and $z = r\cos\theta$, then:

$$R = \sqrt{r^2 - 2rz'\cos\theta + z'^2} \qquad (3)$$

Using a Taylor series approximation on R in (3), one has (assuming $r >> z'$):

$$R \approx r - z'\cos\theta + \frac{z'^2}{2r}\sin^2\theta + \frac{z'^3}{2r^2}\sin^2\theta\cos\theta + \ldots \qquad (4)$$

The first term only is used in the denominator of (1), and up to the second term is used in the exponential for the far field approximation. To determine the distance where the far field begins, the third term is used, and results in the familiar relation:

$$r > \frac{2D^2}{\lambda} \tag{5}$$

where D is, in general, the largest diagonal dimension of the radiator.

For the Fresnel region, the first three terms of (4) are retained, and by taking the derivative of the last term in (4), the boundary between the near field and the Fresnel region can be shown to be:

$$r = 0.62\sqrt{\frac{D^3}{\lambda}} \tag{6}$$

It is important to note that the above expressions as developed are for an aperture antenna with the observation considered to be a point source.

Frequency	Far Field Range (m)	Fresnel Range (m)
L_1	r > 1,113.3	46.9 < r < 1,113.3
L_2	r > 867.5	41.4 < r < 867.5

Table 2-1.
Field zones for frequencies of interest.

The reason for detailing the far field, Fresnel field, and near field regions is that the CTV trajectory passes through each of these regions when following the AR&C procedures. Table 2-1 shows the boundary between the regions for L_1 and L_2. As will be discussed, no approximations of this nature have been used.

2.1.2 *Definition of the Geometry*

Consider the cylinder of Figure 2-1. The cylinder has radius a and length 2L. The cylinder size was chosen according to the latest known dimensions of the SSF: 2.5 meters in radius and 9 meters in length. The axis of the cylinder is along the z axis. Without loss of generality, the GPS receiver can be chosen in the y-z plane at an angle θ in the usual spherical coordinate system where θ is restricted to be between 0 and 90°. The vector $\hat{a}_n$ represents the direction of propagation of the wave incident on the cylinder from the GPS transmitter (note that the location of the earth, and hence the positions of V_{BAR} and the CTV are not specified yet). As shown, the incident magnetic field has a component H_2 in the y-z plane, and an x component H_x.

These two components will be used to represent the RCP wave incident on the SSF.

2.1.3 *Theory of Physical Optics*

In PO [14], the current that is induced on the surface of a perfectly conducting scatterer is approximated as:

$$\boldsymbol{J}_s \approx 2\,\hat{n}\times\boldsymbol{H}^i \tag{7}$$

for the illuminated portions of the object, and is zero elsewhere. $\mathbf{H}^i$ is the incident magnetic field and $\hat{\mathbf{n}}$ is the outward normal unit vector from the surface. Once $\mathbf{J}_s$ is found, the scattered electric field $\mathbf{E}^s(\mathbf{r})$ due to $\mathbf{J}_s(\mathbf{r}')$ is:

$$\boldsymbol{E}^s(\boldsymbol{r}) = \frac{1}{4\pi j\omega\epsilon_0}\nabla\times\nabla\times\int_{S'}\frac{\left[\boldsymbol{J}_s(\boldsymbol{r}')\right]e^{-jk|\boldsymbol{r}-\boldsymbol{r}'|}}{|\boldsymbol{r}-\boldsymbol{r}'|}\,ds' \tag{8}$$

where ϵ_0 is the free space permittivity, k is the wavenumber, ω is the angular frequency, and the curl operations only act on the unprimed coordinates .

To formulate a structure for the numerical procedure, consider the vector:

$$\boldsymbol{F}(\boldsymbol{r}) = \frac{1}{4\pi j\omega\epsilon_0}\int_{S'}\frac{\left[\boldsymbol{J}_s(\boldsymbol{r}')\right]e^{-jk|\boldsymbol{r}-\boldsymbol{r}'|}}{|\boldsymbol{r}-\boldsymbol{r}'|}\,ds' \tag{9}$$

where

$$\boldsymbol{E}^s = \nabla\times\nabla\times\boldsymbol{F} \tag{10}$$

The vector **F** is defined by three scalar integrations, which can be performed numerically using any of a variety of standard procedures such as a Riemann sum or Romberg integration [18]. The number of terms in these sums will be rather large, due to the size (in wavelengths) of the object.

However, once **F** is found, the curl-curl operation must be performed. This is a differential operation that can be approximated by central finite differences. For each point that the scattered field is desired, 39 components of **F** over 19 locations would be required.

It is more efficient to not compute the curl-curl operation in a finite difference scheme, but rather to bring the curl-curl operation into the integrand:

$$\boldsymbol{E}^s(\boldsymbol{r}) = C\int_{S'}|\boldsymbol{J}_s(\boldsymbol{r}')|\left[\nabla\times\nabla\times\left(\hat{e}_J\phi\right)\right]ds' \tag{11}$$

where

$$C = \frac{1}{2\pi j \omega \varepsilon_0}$$
$$\Phi = \frac{e^{-jk|\mathbf{r}-\mathbf{r}'|}}{|\mathbf{r}-\mathbf{r}'|} \tag{12}$$

and the unit vector $\hat{e}_j$ is the direction of the current. This is the method used here.

2.1.4 *Specific Integrals Solved for Cylinder*

The analysis shall begin by combining the geometry of the cylinder with the PO formulation (11). Recall that the incident wave has a propagation vector in the y-z plane and the incident magnetic field has two components, one in the x direction, and one in the plane of the page. The angle of incidence θ is between 0 and 90°. This angle is the angle between the SSF cylinder axis and the GPS satellite where SSF and the GPS satellite are both in the y-z plane. The operating frequency may be either L_1 or L_2.

The unit vector describing the incident direction of propagation is:

$$\hat{a}_n^i = \gamma \hat{e}_y - \delta \hat{e}_z \tag{13}$$

where:

$$\sqrt{\gamma^2 + \delta^2} = 1 \tag{14}$$

The direction of H_2 can be found by noting that it is perpendicular to both the x direction and the direction of propagation:

$$dir H_2 = \hat{e}_x \times \hat{a}_n^i \tag{15}$$

The magnetic field can be written in RCP polarization as:

$$\mathbf{H}^i = H_1 \left[\hat{e}_x + j\gamma \hat{e}_z + j\delta \hat{e}_y \right] e^{-jk\hat{a}_n^i \cdot \mathbf{r}} \tag{16}$$

By performing the dot product in the exponential and noting that:

$$\mathbf{r} = x\hat{e}_x + y\hat{e}_y + z\hat{e}_z \tag{17}$$

the incident magnetic field can be written as:

$$\boldsymbol{H}^i = H_1\left[\hat{e}_x + j\gamma\hat{e}_z + j\delta\hat{e}_y\right]e^{-jk(\gamma y-\delta z)} \tag{18}$$

One can compute the surface current as postulated by PO and given in equation (7). The only portions of the cylinder illuminated by the incident wave are the top and half of the curved surface. For the top,

$$\hat{n} = \hat{e}_z \tag{19}$$

and the surface current on the top is given by:

$$\boldsymbol{J}_s = 2H_1 e^{-jk(\gamma y-\delta z)}\left[\hat{e}_y - j\delta\hat{e}_x\right] \tag{20}$$

with limits of:

$$z = \frac{L}{2}$$
$$\sqrt{x^2+y^2} \le a \tag{21}$$

For the curved surface,

$$\hat{n} = \hat{e}_\rho \tag{22}$$

and the surface current on the curved surface is given by:

$$\boldsymbol{J}_s = 2H_1 e^{-jk(\gamma y-\delta z)}\left[-j\gamma\hat{e}_\phi + (-\sin\phi + j\delta\cos\phi)\hat{e}_z\right] \tag{23}$$

Using the induced currents (20), (23), the following two integrals need to be solved:

$$\boldsymbol{E}^s(\boldsymbol{r}) = 2H_1 C e^{jk\frac{L}{2}\delta}\int_{\rho=0}^{a}\int_{\phi=0}^{2\pi} e^{-jk\gamma\rho\sin\phi}\,\eta(\boldsymbol{r},\boldsymbol{r}')\,\rho\,d\rho\,d\phi \tag{24}$$

on the top of the cylinder, and:

$$\boldsymbol{E}^s(\boldsymbol{r}) = 2H_1 C\int_{z=-\frac{L}{2}}^{\frac{L}{2}}\int_{\phi=-\pi}^{0} e^{jk(\gamma\rho\cos\phi-\delta z)}\eta(\boldsymbol{r},\boldsymbol{r}')\,\rho\,d\phi\,dz \tag{25}$$

on the side of the cylinder, where:

$$\eta(\boldsymbol{r},\boldsymbol{r}') = \nabla\times\nabla\times(\hat{e}_y\phi) \tag{26}$$

By use of various identities and Green's theorem formulations, a closed form expression for η can be obtained for an arbitrary direction of the current. This computation has been performed. The components of η are:

$$\begin{aligned}
&\hat{e_x}[\phi(e_{Jx}[(y-y')^2\xi_1+(z-z')^2\xi_1+2\xi_2]-e_{Jy}(x-x')(y-y')\xi_1-e_{Jz}(x-x')(z-z')\xi_1)\\
&\hat{e_y}[\phi(-e_{Jx}(x-x')(y-y')\xi_1+e_{Jy}[(x-x')^2\xi_1+(z-z')^2\xi_1+2\xi_2]-e_{Jz}(y-y')(z-z')\xi_1)\\
&\hat{e_z}[\phi(-e_{Jx}(x-x')(z-z')\xi_1-e_{Jy}(y-y')(z-z')\xi_1+e_{Jz}[(x-x')^2\xi_1+(y-y')^2\xi_1+2\xi_2])
\end{aligned} \tag{27}$$

where:

$$\xi_1=\left[\frac{k^2}{R^2}-\frac{j3k}{R^3}-\frac{3}{R^4}\right]$$
$$\xi_2=\left[\frac{jk}{R}-\frac{1}{R^2}\right] \tag{28}$$

and

$$\phi=\frac{e^{-jk|\boldsymbol{r}-\boldsymbol{r}'|}}{|\boldsymbol{r}-\boldsymbol{r}'|}$$
$$R=|\boldsymbol{r}-\boldsymbol{r}'| \tag{29}$$

and

$$\hat{e_J}=e_{Jx}\hat{e_x}+e_{Jy}\hat{e_y}+e_{Jz}\hat{e_z} \tag{30}$$

which is a constant vector with respect to the unprimed coordinates. The direction of $\hat{e}_j$ is represented by the terms enclosed by brackets [ ] in (20) and (23).

The cylinder size used is: 2.5 meters in radius and 9 meters in length. The N-dimensional Romberg integration routine as found in [18] has been used to perform the integrations in computing (24) and (25). The integration routine has a tolerance set to 0.1 percent.

2.2 Details of Problem Statement

2.2.1 *RCP Component Calculations*

The computations derived up to this point will provide the magnitude and phase of the x, y, and z components of the scattered field. Since the receiver on the CTV will be accepting only the RCP component of the scattered field, the electric field computed as the sum of (24) and (25) should be converted to only the RCP component.

The RCP component of the scattered field in the direction of the CTV can be found by first converting the Cartesian components to a spherical vector:

$$\begin{aligned} E_r &= E_x \sin\theta\cos\phi + E_y \sin\theta\sin\phi + E_z \cos\theta \\ E_\theta &= E_x \cos\theta\cos\phi + E_y \cos\theta\sin\phi - E_z \sin\theta \\ E_\phi &= -E_x \sin\phi + E_y \cos\phi \end{aligned} \tag{31}$$

Once the spherical components are computed, it is necessary to find the RCP component in the direction of CTV. The r component is dropped, and the other components must be a sum of RCP and LCP terms. This can be written as:

$$E_\theta \hat{e}_\theta + E_\phi \hat{e}_\phi = \alpha_1 e^{j\gamma_1} (\hat{e}_\theta + j\hat{e}_\phi) + \alpha_2 e^{j\gamma_2} (\hat{e}_\theta - j\hat{e}_\phi) \tag{32}$$

where the first term on the right side is the RCP term and the second term is the LCP term. By manipulating (32), it can be shown that the RCP magnitude (α_1) and phase (γ_1) are given by

$$\begin{aligned} \alpha_1 \sin\gamma_1 &= \frac{1}{2} (E_{\theta,i} - E_{\phi,r}) \\ &\text{or} \\ \alpha_1 \cos\gamma_1 &= \frac{1}{2} (E_{\theta,r} + E_{\phi,i}) \end{aligned} \tag{33}$$

and

$$\tan\gamma_1 = \frac{-E_{\theta,i} + E_{\phi,r}}{E_{\theta,r} + E_{\phi,i}} \tag{34}$$

where the subscripts i and r denote imaginary and real parts, respectively. The magnitude of α_1 is found using one of the relations in (33), according to numerical stability. In other words, if $\sin\gamma_1$ is too small, then the second relation is used. Equations (33) and (34) are used to determine the RCP component of the scattered field at the CTV.

2.2.2 *Reference Field Calculations*

The direct signal from the GPS satellite to the CTV is computed by providing additional information concerning the geometry of the problem. The information necessary is the angle of the earth with respect to the cylinder coordinates. Figure 2-2 illustrates the coordinate system used. All quantities in the figure labeled by an "r" are vectors.

The radius of the earth has a mean value of r_e = 6,370,949 meters. The length of r_0 is 22,240 km + r_e. The length of r_s is 331 km + r_e. It is necessary to find the location of the GPS satellite in the SSF coordinate system. This position is defined by the vector r_g and the angle θ.

r_c is the position of CTV in the SSF coordinate system. The vector $\mathbf{r}_s$ defines the position of the earth (in the SSF coordinate system) referenced by the angles θ_E and ϕ_E. From the figure,

$$|\boldsymbol{r}_0-\boldsymbol{r}_s|=|\boldsymbol{r}_g| \tag{35}$$

In addition, defining

$$\boldsymbol{r}_g=x_g\hat{\boldsymbol{e}}_x+y_g\hat{\boldsymbol{e}}_y+z_g\hat{\boldsymbol{e}}_z \tag{36}$$

and

$$\boldsymbol{r}_s=x_s\hat{\boldsymbol{e}}_x+y_s\hat{\boldsymbol{e}}_y+z_s\hat{\boldsymbol{e}}_z \tag{37}$$

it can be deduced from Figures 2-1 and 2-2 that $x_g=0$, and $y_g=-z_g\tan\theta$. Then, the relation to be solved is:

$$|\boldsymbol{r}_0|=\sqrt{x_s^2+(-z_g\tan\theta-y_s)^2+(z_g-z_s)^2} \tag{38}$$

The values of z_g and y_g determine the location of the GPS satellite in the SSF coordinate system.

The reference for the incident field is at the origin of the SSF cylinder. At this location, H^i is 1/377 A/m and E^i is 1 V/m. At the GPS satellite, the magnitude of **E** is r_g, and therefore, at the CTV:

$$|\boldsymbol{E}_{ref}|_{@CTV}=\frac{r_g}{|\boldsymbol{r}_g-\boldsymbol{r}_c|} \tag{39}$$

The phase is determined easily by finding the distance back to GPS and then to CTV:

$$arg\{E_{ref}\}=e^{jk(|\boldsymbol{r}_g|-|\boldsymbol{r}_g-\boldsymbol{r}_0|)} \tag{40}$$

Thus, the direct signal is:

$$\boldsymbol{E}_{ref}=\frac{|\boldsymbol{r}_g|}{|\boldsymbol{r}_g-\boldsymbol{r}_0|}e^{jk(|\boldsymbol{r}_g|-|\boldsymbol{r}_g-\boldsymbol{r}_c|)} \tag{41}$$

where the direct signal is RCP.

Preliminary tests of the code have been performed. The geometry has $\theta=0°$ which implies that only the top disk of the cylinder is illuminated. The earth is at $\theta_E=180°$, $\phi_E=90°$. This means that the GPS satellite, SSF, and the earth form a single line. The CTV is swept along this line (not the trajectory) from 300 meters above the SSF to 300 meters below SSF. This example illustrates the shadowing of the CTV signal by SSF. The magnitude of the ratio of (ideal) direct

signal to multipath signal is shown in Figure 2-3, and the phase angle (in degrees) is shown in Figure 2-4. Note that the phase varies much more quickly near SSF, where the CTV trajectory is in the near field.

2.2.3 *Parameterization of CTV Trajectory*

An assumption has been made for the geometry of the SSF with respect to the earth. Only situations where the module is one of two positions seem plausible. These two situations are depicted in Figure 2-5(a,b). In Figure 2-5(a), the angle ϕ_E may not appear to make a difference; however, recall that the GPS satellite is in the z-y plane (which is the plane of the paper). In Figure 2-5(b), the geometry again may appear to not require specification of all angles, but it is necessary because of the relative (and fixed) location of the GPS satellite.

The trajectory shown in Figure 1-1 has been parameterized beginning at the location 2 km below V_{BAR} and 37 km behind the SSF. Motion is along the trajectory in steps of 10 meters in x, where x is parallel to V_{BAR}. Thus, for each change of $\Delta x = 10$ m, the position of y is computed. This parameterization results in 4,214 points along the trajectory, where the last point is at SSF and is not computed. As will be seen in the results, the data points are sufficiently close together to allow for very smooth magnitude graphs. No granularity in the magnitude results has been observed.

The first 3,700 points are equally space along the line from the start to the first burn. This set of points is called "line 1". Points between 3,700 and 4,014 are on the approximately circular approach to V_{BAR}. These points are on "circle 1". The next 170 points are (up to 4,184) are on the first elliptical approach and are called "ellipse 1". The second elliptical approach is 20 points ("ellipse 2") up to 4,204. The remaining 10 points are equally spaced on the final approach, denoted as "final approach". Table 2-2 summarizes the points with respect to the portions of the trajectory.

Portion of Trajectory	Starting Point Number	Ending Point Number
Line 1	1	3,700
Circle 1	3,700	4,014
Ellipse 1	4,014	4,184
Ellipse 2	4,184	4,204
Final Approach	4,204	4,214

Table 2-2
Trajectory point numbers for portions of trajectory.

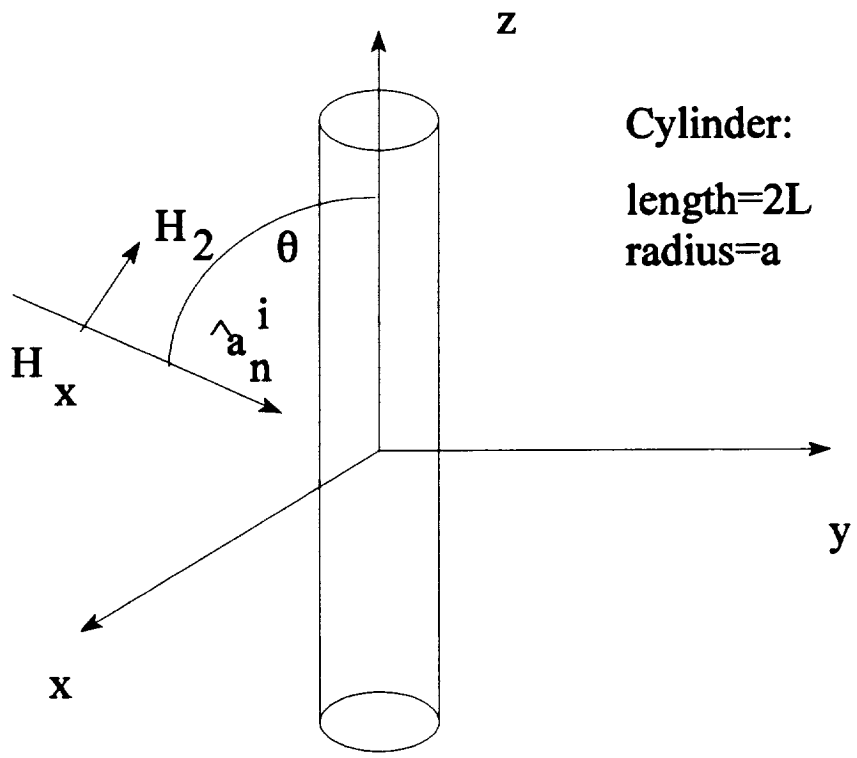


Figure 2-1. Geometry for scattering from cylinder.

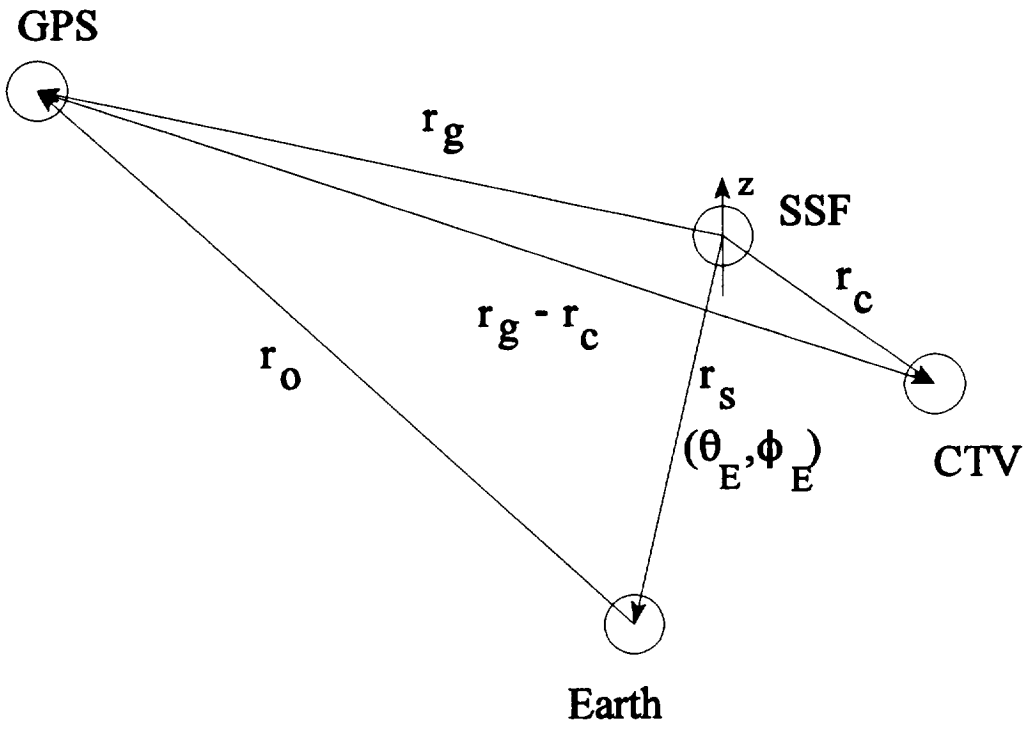


Figure 2-2. Geometry for computation of direct signal. Note definition of θ_E and ϕ_E relative to SSF coordinate system (z axis shown).

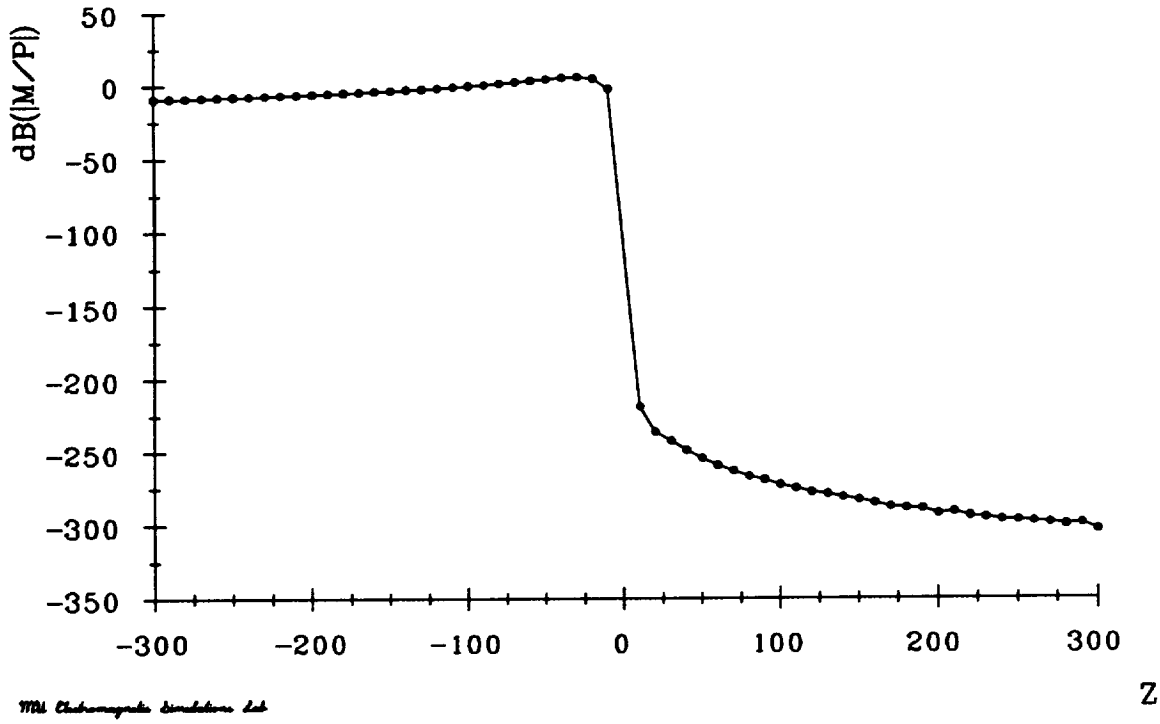


Figure 2-3. Magnitude (dB) of relative scattered signal strength for $\theta = 0$, $\theta_E = 180$, and for CTV along line defined by SSF and GPS ($Z > 0$ above SSF and $Z < 0$ below SSF).

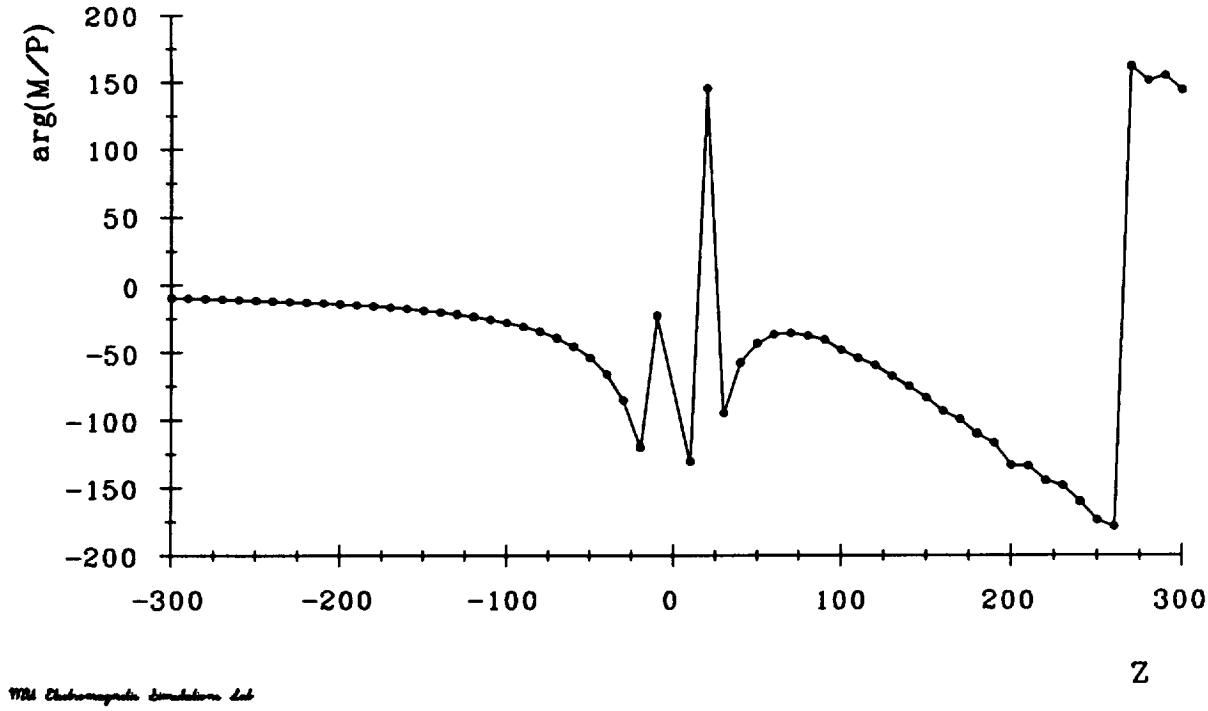


Figure 2-4. Phase of multipath signal for case of Figure 2-3.

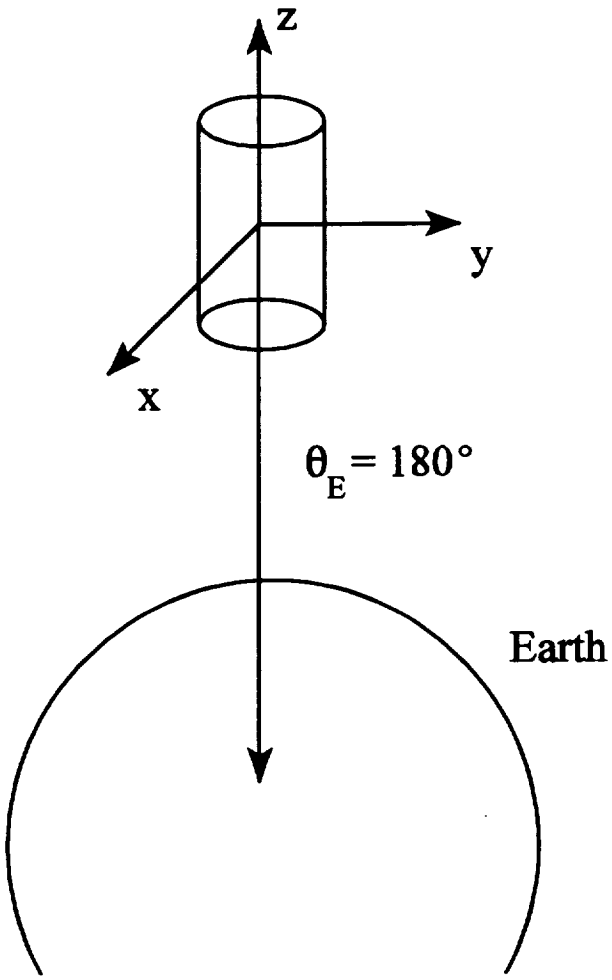


Figure 2-5(a). Geometry for $\theta_E = 180°$. Note that changing ϕ_E changes the angle of approach.

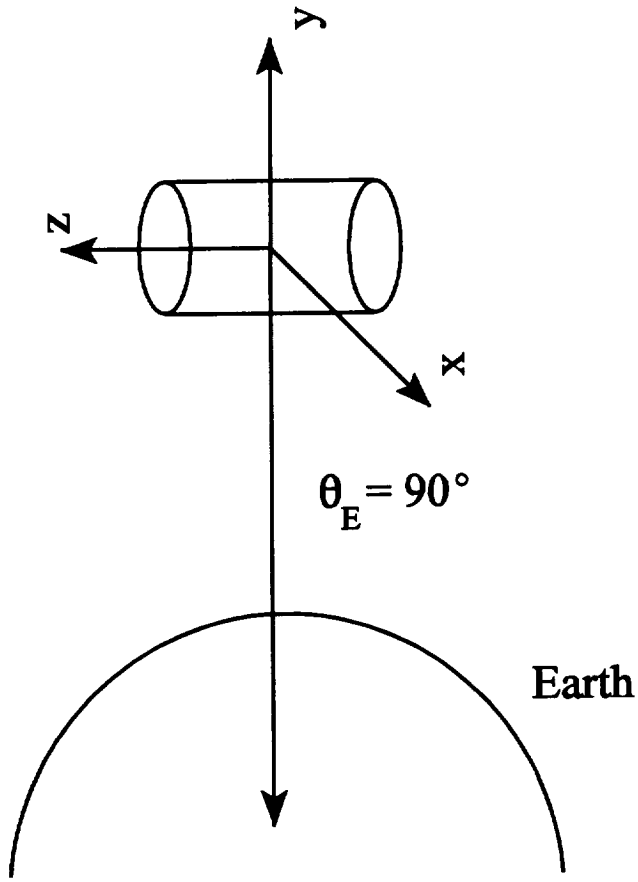


Figure 2-5(b). Geometry for $\theta_E = 90°$. Note that ϕ_E is -90° in this figure.

3. Simulation Results and Discussion

The physical optics solution for the geometry of Figure 2-1 is given in this section for angles θ from 0 to 90° in steps of 30°, and ϕ_e from 0 to 90° also in steps of 30°. The data is for frequencies L_1 and L_2 for θ_E at 90° (Figure 2-5b) or θ_E at 180° (Figure 2-5a). Recall that θ is the angle shown in Figure 2-1 that defines the location of the GPS satellite. the angles θ_E and ϕ_E determine the direction from the SSF module to the center of the earth. Note that the GPS satellite is always in the y-z plane and that the rotation of x and y by ϕ_E does not affect the SSF cylinder, only the location of the GPS satellite. As will be seen, the CTV always approaches the GPS satellite. Situations where the CTV is moving away from the GPS satellite have not but could easily be investigated.

3.1 Method or Nomenclature for Data Presentation

Each trajectory is broken into 4,214 points. These points consist of 3,700 along the initial line of approach at 2 km below V_{BAR} . The first burn is modeled as a circle, and approaches V_{bar} 2 km from the SSF. This is points 3,701 to 4,014. The first approaching ellipse is points 4,015 to 4,184, and the second ellipse is from 4,185 to 4,204. The final approach is a straight line of 10 points from 4,205 to 4,215. The point 4,215 is not computed since it is the dock location.

The following terminology has been used to identify common items. Points along the trajectory shall be denoted as tp-1 to tp-4014. The GPS satellite, the space station Freedom, and the CTV will be referred to as GPS, SSF, and CTV, respectively. The term path difference will be abbreviated to p/d, and the 300 meter requirement is "broken" if the p/d is less than 300 meters. The 300 meter requirement will be called the 300 requirement.

3.2 Description of Format for Data Presentation

Each set of data has been grouped as a variation in θ for fixed θ_E, ϕ_E, and frequency. The data is plotted as dB down from the direct signal as a function of the point on the trajectory. The different maneuvers along the trajectory are also indicated on each figure. Only data with a path difference less than 300 meters is shown. To speed computations, data with a p/d above 300 meters was not calculated.

The top graph of each figure represents the results on approach from roughly the middle of the first elliptical approach near V_{BAR}, and ends 10 meters before docking. The bottom graphs illustrate one of two effects. In most figures, the end of line 1 and the beginning of circle 1 is shown (case 1) to illustrate scattering that is present and has a p/d less than 300 meters. The second possibility was used if no data in case 1 is present. For case 2, the bottom graph shows data from the beginning of line 1 up to circle 1, tp-1 to tp-3810. However, for some cases (such as θ_E = 90°, ϕ_E=90°, f=L_2) data was also present from the beginning of line 1 and appears at the left end of the bottom graph. Note that no data was collected between tp-3810 and tp-4150 because of the 300 requirement.

Section 3.3 will summarize the results in a cursory fashion, and Section 3.4 will provide a detailed description of the scattering mechanisms present along with a more detailed description of the results.

3.3 Results

For the case of $\theta_E = 90°$, refer to Figure 3-1. Note that $\theta = 0$ is the same orientation for each value of ϕ_E. It is shown on each of Figures 3-2 to 3-9 for comparative purposes. In addition, values of ϕ_E between 90 and 180° provide identical results due to symmetry. As mentioned in Section 3.2, data for the z axis pointing to the right have not been (but could easily be) computed.

During the final maneuvers (top graph of figures), the 300 requirement is broken later and hence the data begins later as ϕ_E increases. This is independent of frequency. Except for $\theta = 0$, the data has nominal peak values near -35 dB, until CTV is very close to SSF. The sharp rise in data for $\theta = 0°$ at tp-4204 is due to the polarization on true backscatter. The point tp-4204 is the first point in the final approach. RCP backscatter is also RCP, and this situation breaks down quickly off axis. For $\theta = 0°$, the final approach is 1 to 2 dB above the direct signal except very near SSF. Note also that scattering at $\theta = 0°$ is due to the flat disk only.

Earlier in the approach (bottom graph of figures), the 300 requirement is broken in various locations depending on the orientation of GPS, SSF, and CTV. In addition, for $\theta = 0°$, the 300 requirement is broken early in the approach. Nominal dB values for the early approach data are -55 dB. The specular contributions vary in magnitude, varying from -35 dB to -20 dB at some peaks.

For the case of $\theta_E = 180°$, refer to Figure 3-10. Note that $\theta = 0$ is the same orientation for each value of ϕ_E. It is shown on each of Figures 3-11 to 3-18 for comparative purposes. In addition, values of ϕ_E between 0 and -90° provide identical results due to symmetry. As mentioned in Section 3.2, data for ϕ_E planes on the right have not been (but could easily be) computed.

During the final maneuvers (top graph of figures), the 300 requirement is broken earlier and hence the data begins earlier as ϕ_E increases. This is independent of frequency. The data has nominal peak values near -25 dB, until CTV is very close to SSF. The rise at the end of the data reaches values up to -4 dB.

Earlier in the approach (bottom graph of figures), the 300 requirement is broken in various locations depending on the orientation of GPS, SSF, and CTV. In addition, for $\theta = 0°$, $\phi_E = 0°$, the 300 requirement is broken early in the approach. Nominal dB values for the early approach data are -55 dB. The specular contributions again vary in magnitude from -35 dB to -20 dB at some peaks.

The phase of the scattered signal has also been computed for each case. The phase is plotted for the case of $\theta = 0°$, $\theta_E = 180°$, and $\phi_E = 0°$ in Figure 3-19. The top graph is the phase during the final approach, and the bottom graph is centered about the forward scattering null at tp-3700. The phase does not in this case or in any other data sets illustrate any typical trends. This is to be expected, since the distance traveled between any two points along the trajectory is at least 10 meters, which is more than 30 wavelengths. Thus, the phase of the solution varies very quickly.

3.4 Discussion

3.4.1 *Scattering Mechanisms*

This section shall discuss in more detail the results that have been obtained for data along line 1 and the beginning of circle 1 (which is the lower graphs in Figures 3-2 to 3-9 and 3-11 to 3-18). This information can be used to justify the solution found, and more importantly, to guide the interpretation of future, and possibly more complex situations.

A very useful pair of antenna array patterns for the radar tracking engineer is the sum and difference patterns. The sum pattern allows for rough location measurement by maximizing the signal received. This is accomplished by adjusting the phase shift on the received signals until their "sum" is maximized. The phase shift indicates the rough location of the target. At this time, the elements on one side of the array are given an extra 180° phase shift to create a "difference pattern". Then, the original phase shift is adjusted further to obtain the precise location of a target.

These two patterns appear similar except for a major difference in the main lobe. The sum pattern has a peak, but the difference pattern has a very sharp null. See Figure 3-20.

Two of the three basic effects discussed in this section are comparable to sum and difference patterns. When the scattering is "forward scattering", the reflected signal appears to travel through the target, and a deep null is present. See Figure 3-21. When the scattering is "specular reflection", it appears as if the signal has bounced off the target, and the sum pattern results (with a large signal at the exact bounce angle). See Figure 3-22. Much of the difference between forward scattering and specular reflection is due to the RCP nature of the incident signal. Note that forward scattering is not truly a difference pattern, since the sidelobes typically fall toward the null at the center. In a difference pattern, the nulls increase toward the center of the pattern.

The third basic mechanism is a partial flaw in the electromagnetic development that is common to all PO and PTD computations, which will be called an anomalous peak (compare to the Ufimtsev singularity, [19]). This singularity occurs because of the discontinuity in PO current on a smooth surface. See Figure 3-23. The scattering is depicted as forward scattering, but in fact a small peak occurs. This peak is due to the discontinuity in one component of the current.

Consider the case $\theta_E = 90°$, $\phi_E = 0°$, $f = L_1$ (Figure 3-2). Note that all of GPS, CTV and SSF are in the plane of Figure 3-1. In the bottom graph, one can see (centered around tp-3700) a forward scattering pattern except for the small peak at the center. This peak is an example of the anomalous peak since at tp-3700 GPS, SSF, and CTV are perfectly aligned. This is forward scattering, but because of the edges of the curved surface of SSF, one sees the anomalous peak at the center of the pattern (center refers to symmetrical center of the pattern).

For the same figure, when $\theta = 60°$, a truer forward scattering pattern results, and at $\theta = 30°$, an exceptionally clear difference pattern is present. The nulls in these are illustrated in the figure. As θ changes, the angle of forward scattering changes, and the locations of the forward scattering nulls follow exactly. This was checked by looking at the precise trajectory point numbers. At $\theta = 0°$, the triangle formed by GPS, SSF, and CTV is very thin at tp-1 and widens as the point number increases. Near the beginning of circle 1, the 300 requirement enters and no data is shown. Similar results are seen for $f = L_2$.

As ϕ_E increases from 0 to 30°, the forward scattering is dominated by specular reflection due to the angle of GPS relative SSF and CTV. Now, peaks in the specular reflection clearly show the 2:1 distances between $\theta = 90°$, 60°, and 30°. As ϕ_E increases further to 60° and 90°, the p/d is above 300 meters, and the specular reflections do not appear. For these cases, representative data for tp-1 to tp-3700 are shown.

Next consider the case $\theta_E = 180°$, $\phi_E = 0°$, $f = L_1$ (Figure 3-11). Note that all of GPS, CTV and SSF are in the plane of Figure 3-10. In the bottom graph, one can see centered about tp-3700 a forward scattering pattern for $\theta = 0°$. At tp-3700, GPS, CTV, and SSF are perfectly aligned. Since the current is only due to the top disk, no anomalous peak results. Again, as $\theta = 30°$ and 60°, the point of forward scattering shifts earlier in line 1, exactly matching the forward scattering angles. In addition, for $\theta = 90°$, the SSF, GPS, and CTV form a very thin triangle as before. However, once $\phi_E = 30°$, much of the specular reflections and the thin triangle have disappeared behind the 300 requirement. At $\phi_E = 90°$, some of the specular reflections still persist at $\theta = 90°$ and a small portion at $\theta = 60°$. One does not see the anomalous peak at $\theta = 90°$ because the alignment is not present.

3.4.2 *300 Meter Requirement*

The 300 meter requirement was used to speed up collection of the data. In short, if the path difference was greater than 300 meters, the code reported the p/d and moved on to the next point in the trajectory. It is quite obvious that when GPS, SSF, and CTV approximately form a line (in that order as shown in Figure 1-2), then the p/d will be small. This implies the importance of calculating accurate forward scattering from SSF. However, in many cases discussed in the previous section, specular reflection was the mechanism, implying a thin triangle of non-zero area.

The calculation of forward scattering may be suspect due to the anomalous peak, but also

because of the nature of the electromagnetic approximation. It just does not predict forward scattered fields as well as a more rigorous solution. There are no good methods of estimating the error, except to say it is not excessively large. In fact, the forward scattering is smaller than the specular reflection components.

The calculation of specular reflection components, however, is more precise. This means that the forward scattering may be a bit larger (or smaller) then the -45 dB nominally predicted, but the specular reflections will be quite close to the values predicted.

3.4.3 *Key Issues During Final Approach*

During the first phases of the trajectory, CTV is never closer than 2 km from SSF. At the end of circle 1, the distance is 2 km, and CTV is still in the far field. However, during the first or second ellipse, the 300 requirement is broken and data begins appearing. At about the same time, during the first ellipse the CTV leaves the far field and enters the Fresnel region. In some of the data sets, the periodic, side lobe structure is present. However, once within the Fresnel region, the regularity is less obvious. The Fresnel region extends to the point where the CTV is within 40 meters of the SSF. Assuming the optical system will assume control before the CTV is less than 100 meters of the SSF, the GPS system will not be required within the near field region.

Within the Fresnel zone, the data generally shows a slow increase with a few isolated nulls in the data. Once very close to the SSF, the data shows a +2 dB scattered field when the flat disk or top of the cylinder is approached, and a -4 dB scattered field when the curved side of the cylinder is approached. This is consistent with geometrical solutions to the problem in that the radius of curvature of the scattered field depends on the curvature of the scatterer (from GO). Thus, the curved surface tends to cause extra attenuation due to the finite radius of curvature in one direction, but the flat plate does not increase the attenuation, since both principle radii of curvature are infinite.

The solution used contains no approximations except for the current used. No other approximations were used. Thus, our solutions are as accurate as the current we have used. In the far field, the approximation of the current is very reasonable. However, as the observations move closer to the scatterer, the approximation becomes less reliable. Certainly, in the near field, the approximation for the current is suspect. However, in the Fresnel region, the current is still reasonable, though the results are not as accurate as the far field calculations.

In the near field, there is very little in the literature to base any observations upon. However, one sees in the data that the near field can cause some oscillations that are possibly due to a near field standing wave pattern created by the incident field when combined with the scattered field. This is one reasonable explanation for the oscillations that occur in many of the data figures, such as Figures 3-2 and 3-11.

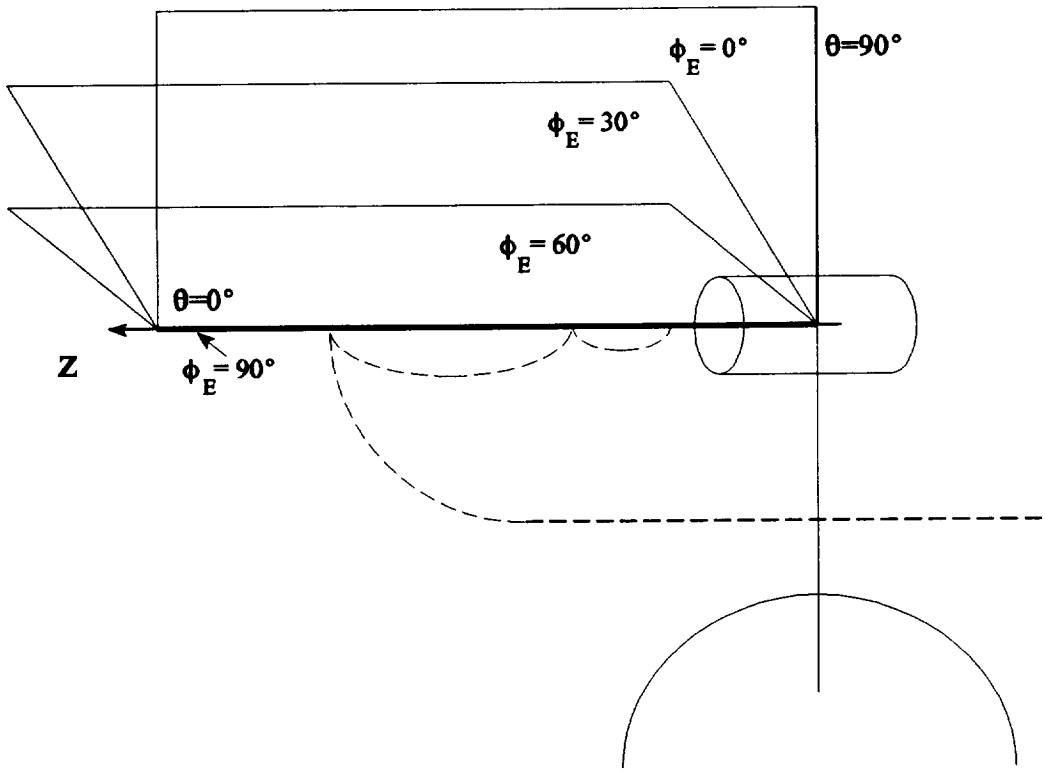


Figure 3-1. Location of ϕ_E planes relative to the earth and the trajectory for $\theta_E = 90°$. CTV trajectory is shown in dashed lines, the earth is the semicircle in the lower right, and the drawing is not to scale.

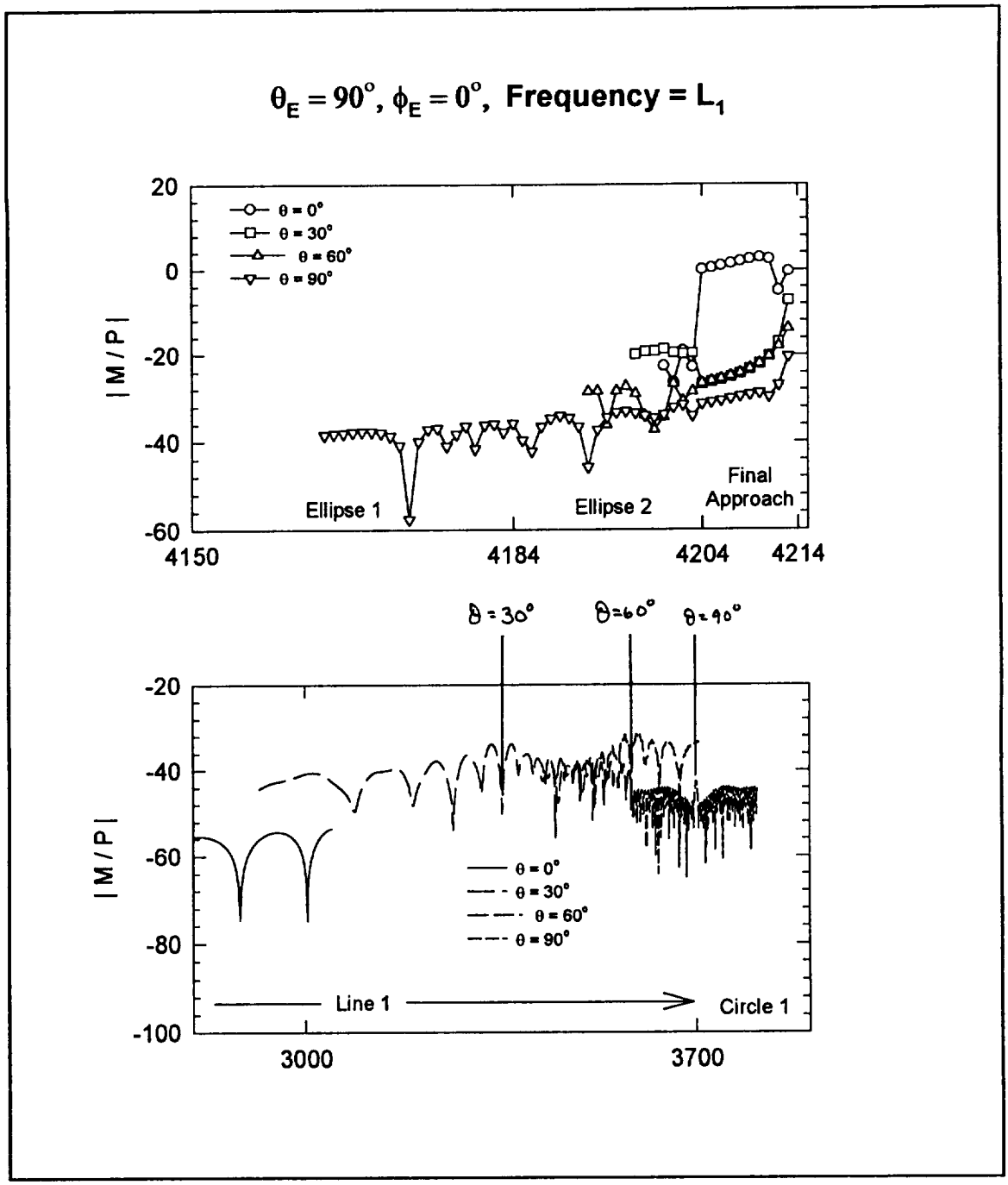


Figure 3-2. (190-x.dat)

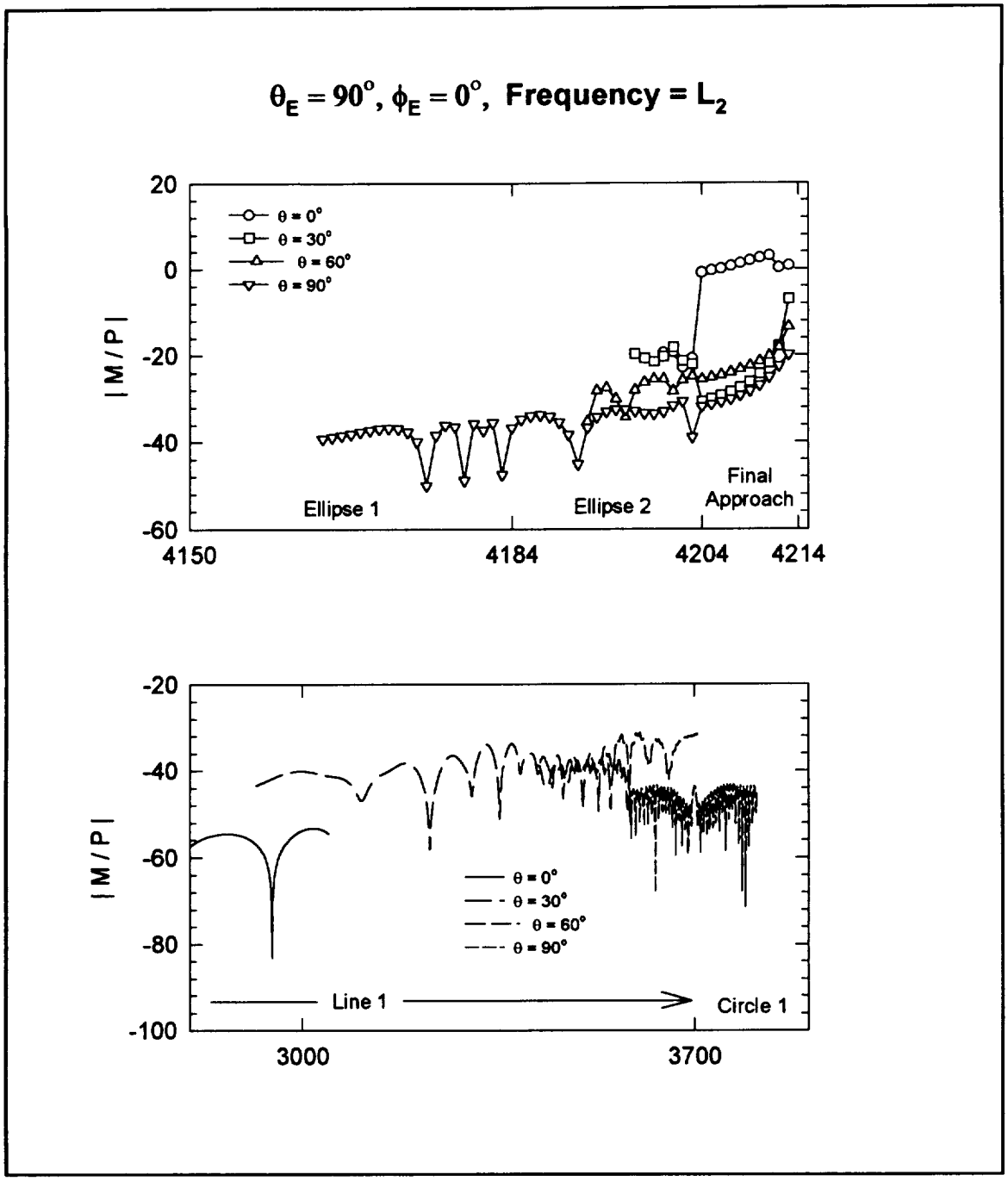


Figure 3-3. (290-x.dat)

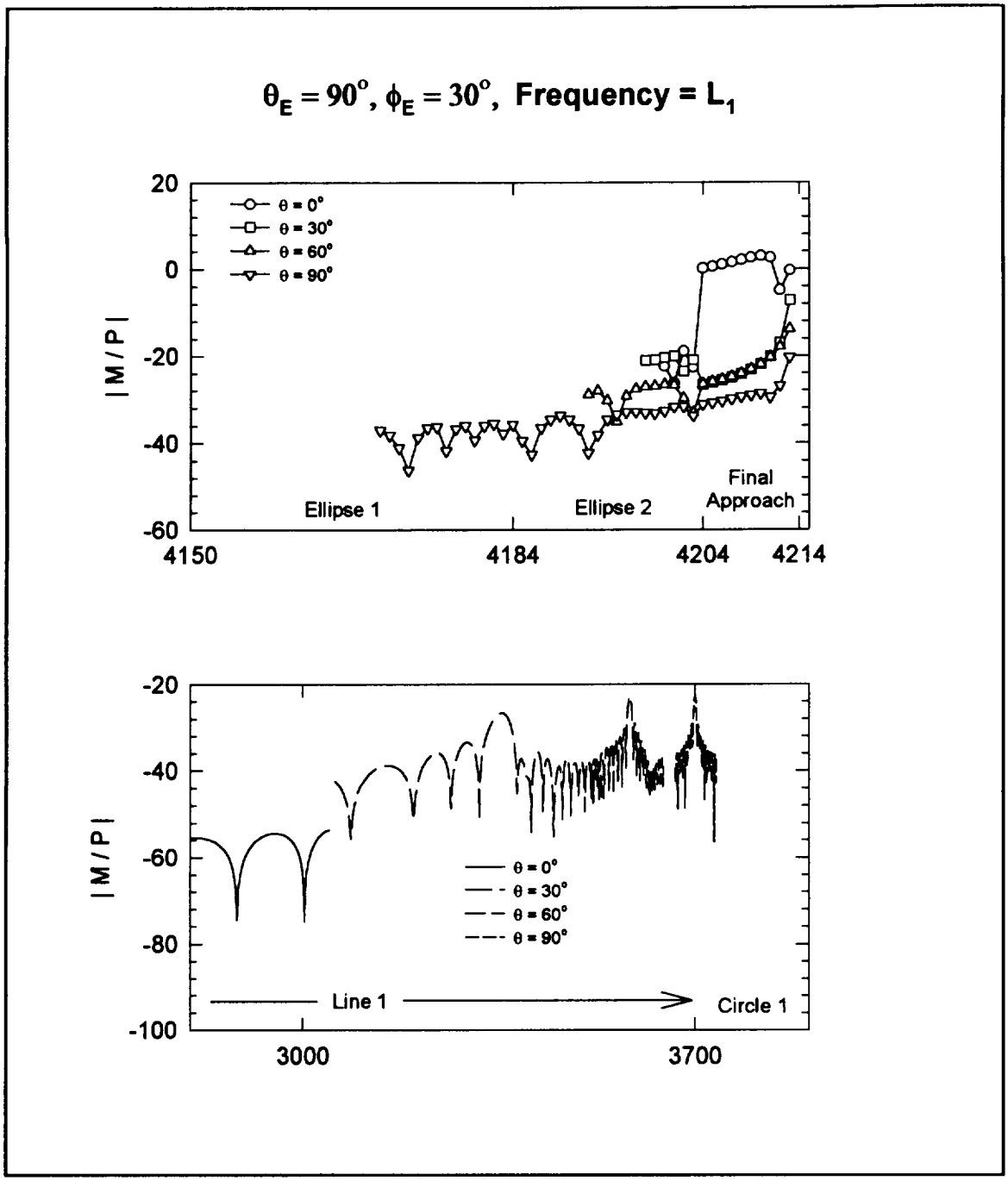


Figure 3-4. (193-x.dat)

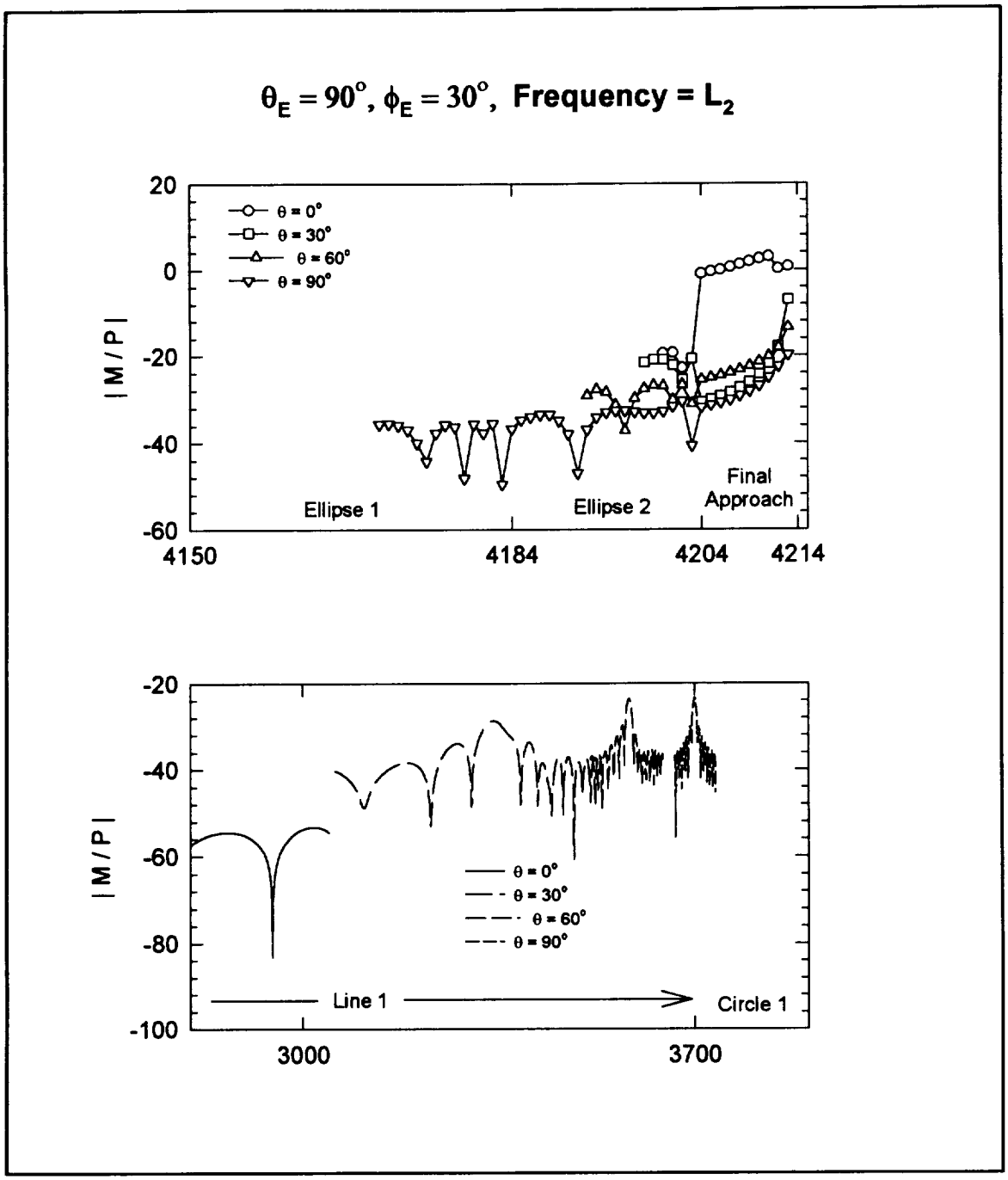


Figure 3-5. (293-x.dat)

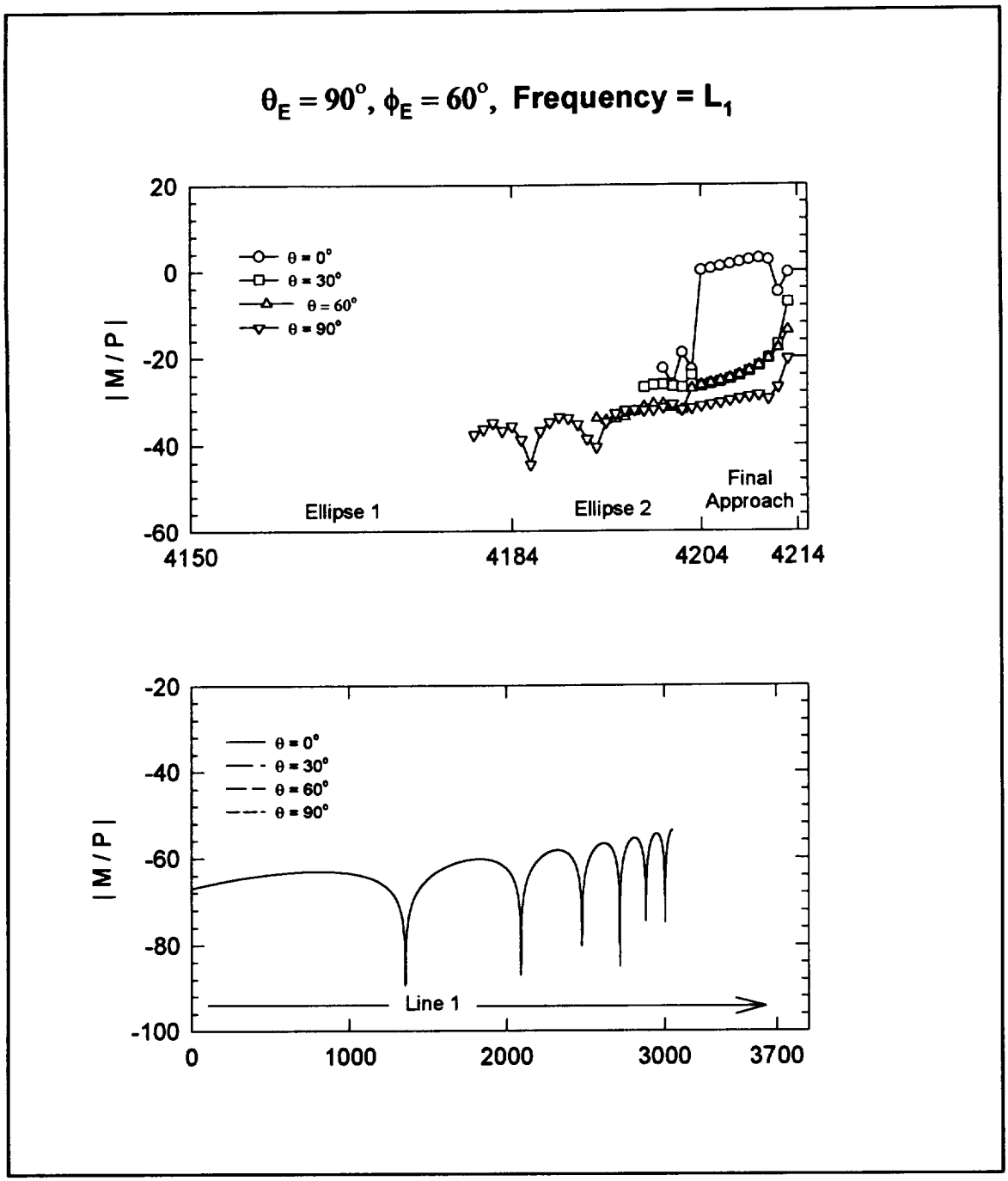


Figure 3-6. (196-x.dat)

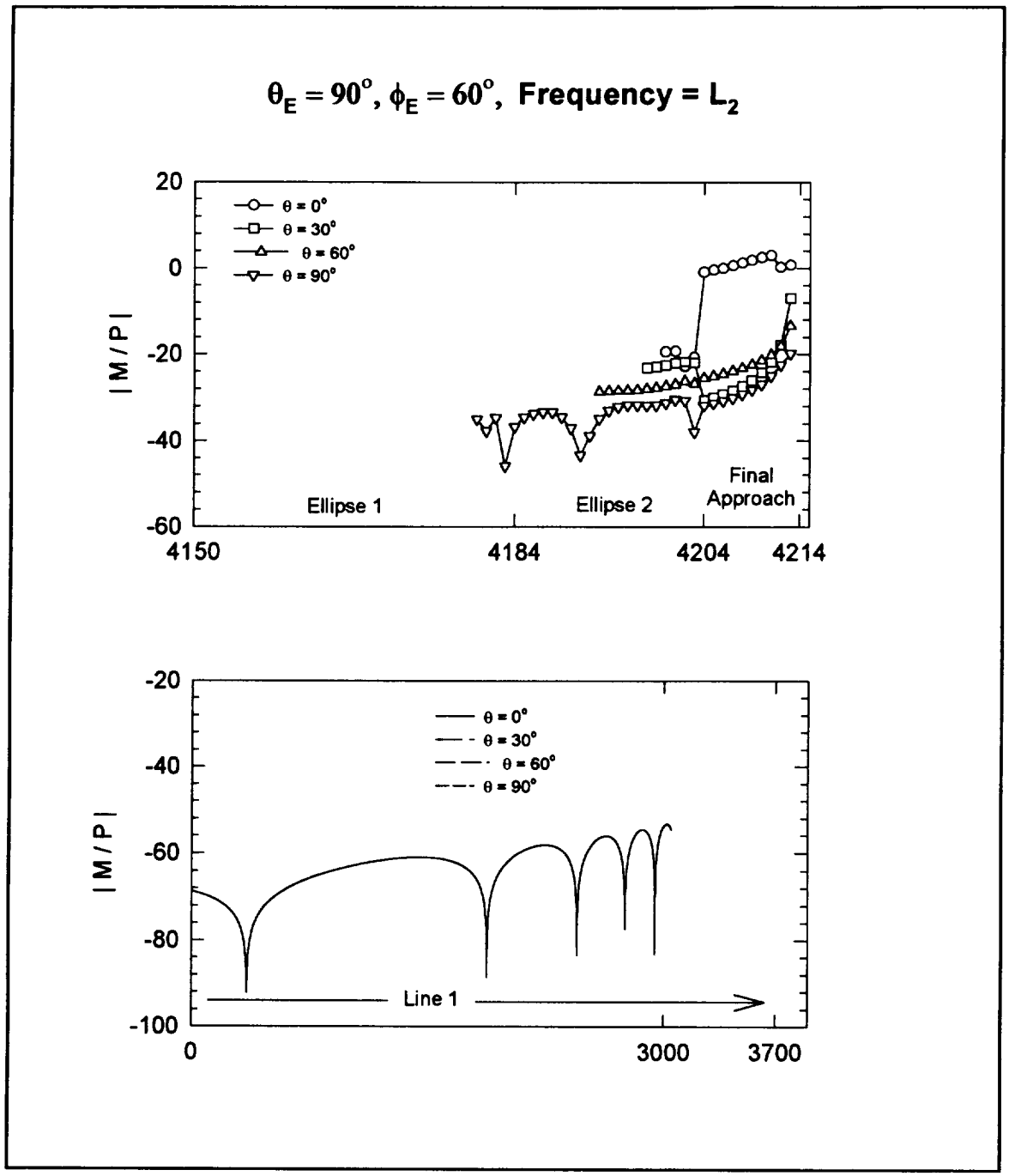


Figure 3-7. (296-x.dat)

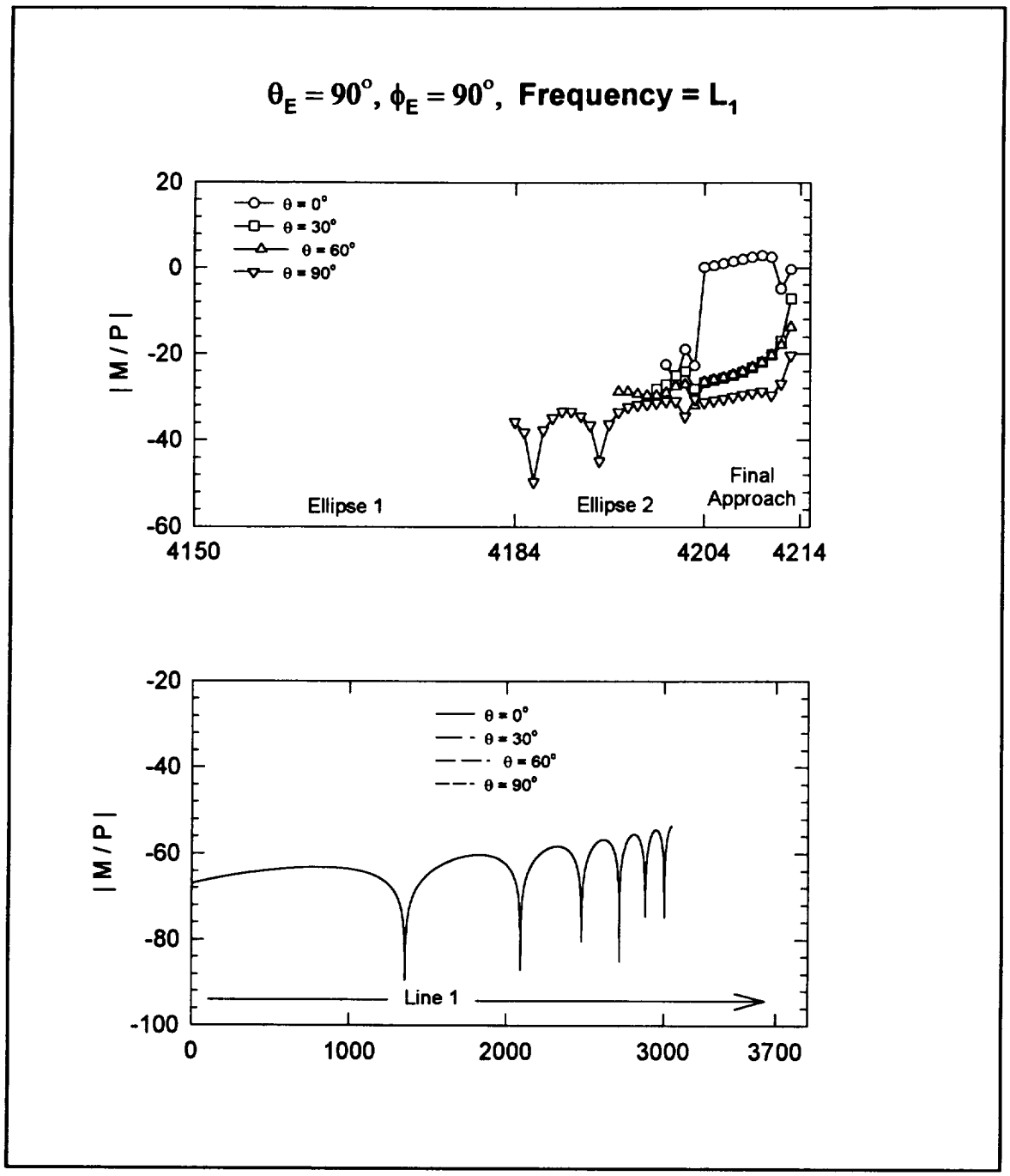


Figure 3-8. (199-x.dat)

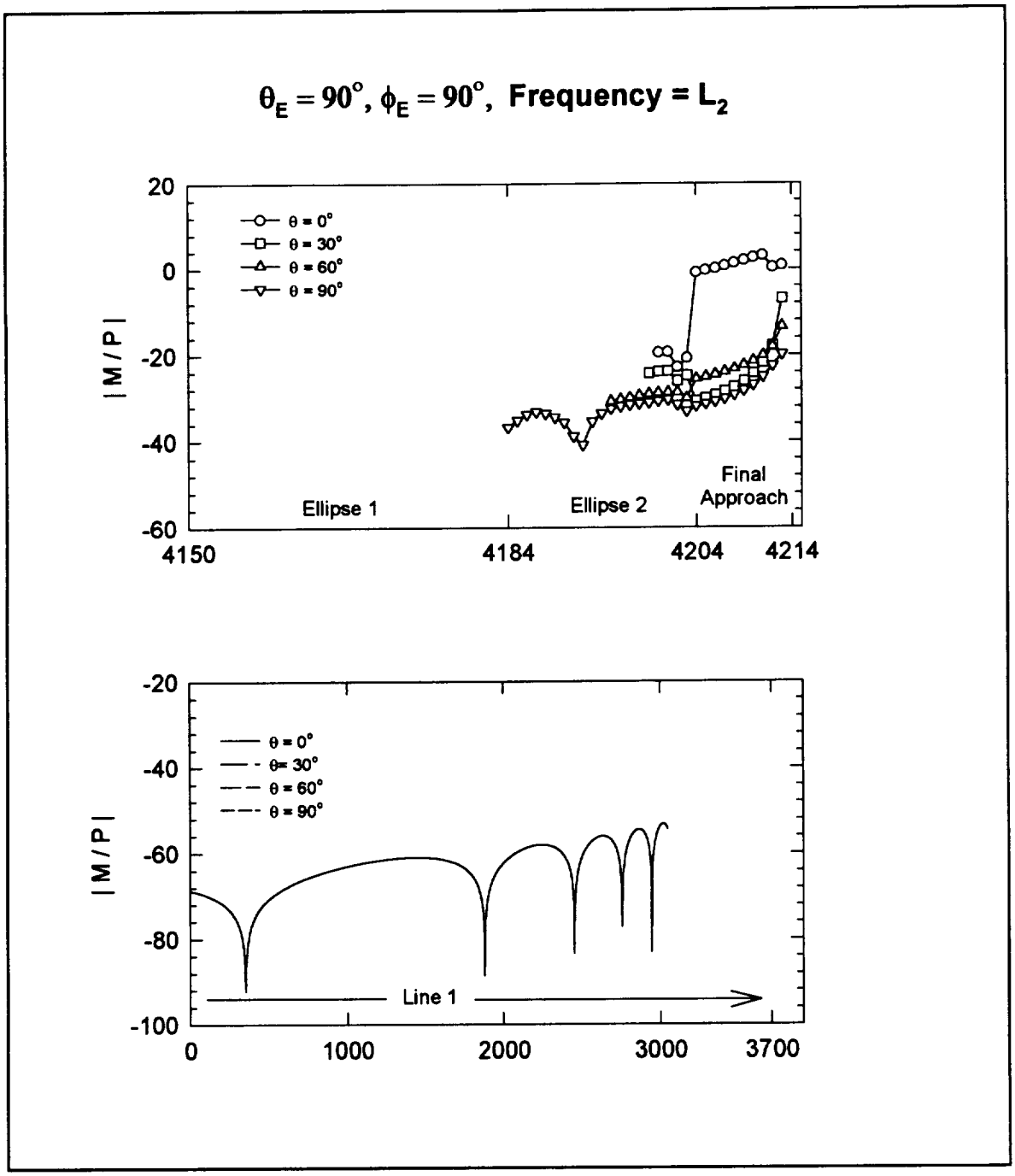


Figure 3-9. (299-x.dat)

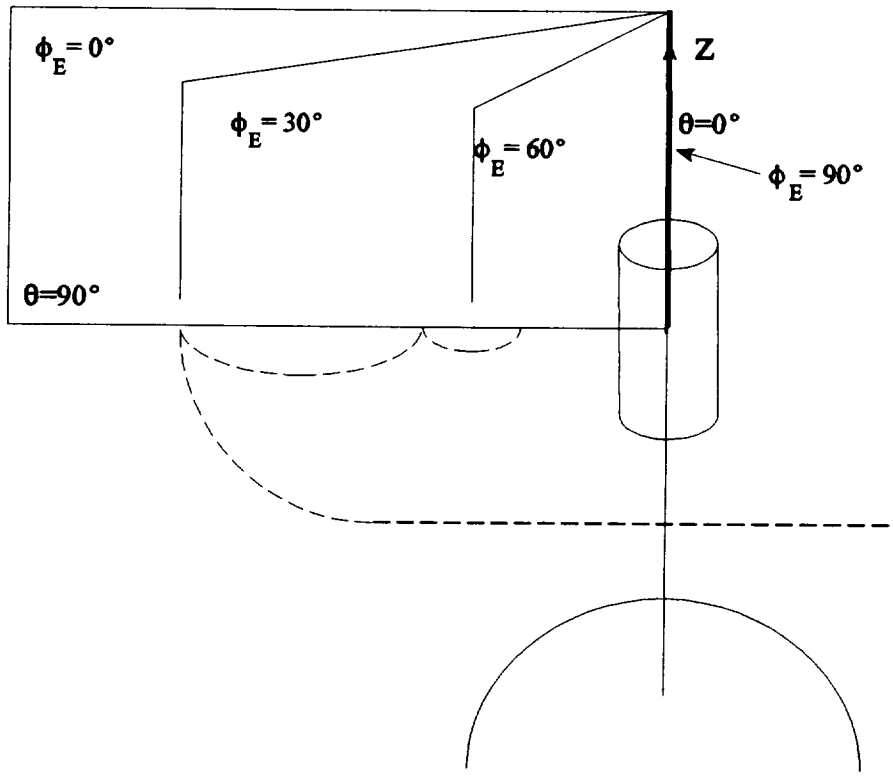


Figure 3-10. Location of ϕ_E planes relative to the earth and the trajectory for $\theta_E = 180°$. CTV trajectory is shown in dashed lines, the earth is the semicircle in the lower right, and the drawing is not to scale.

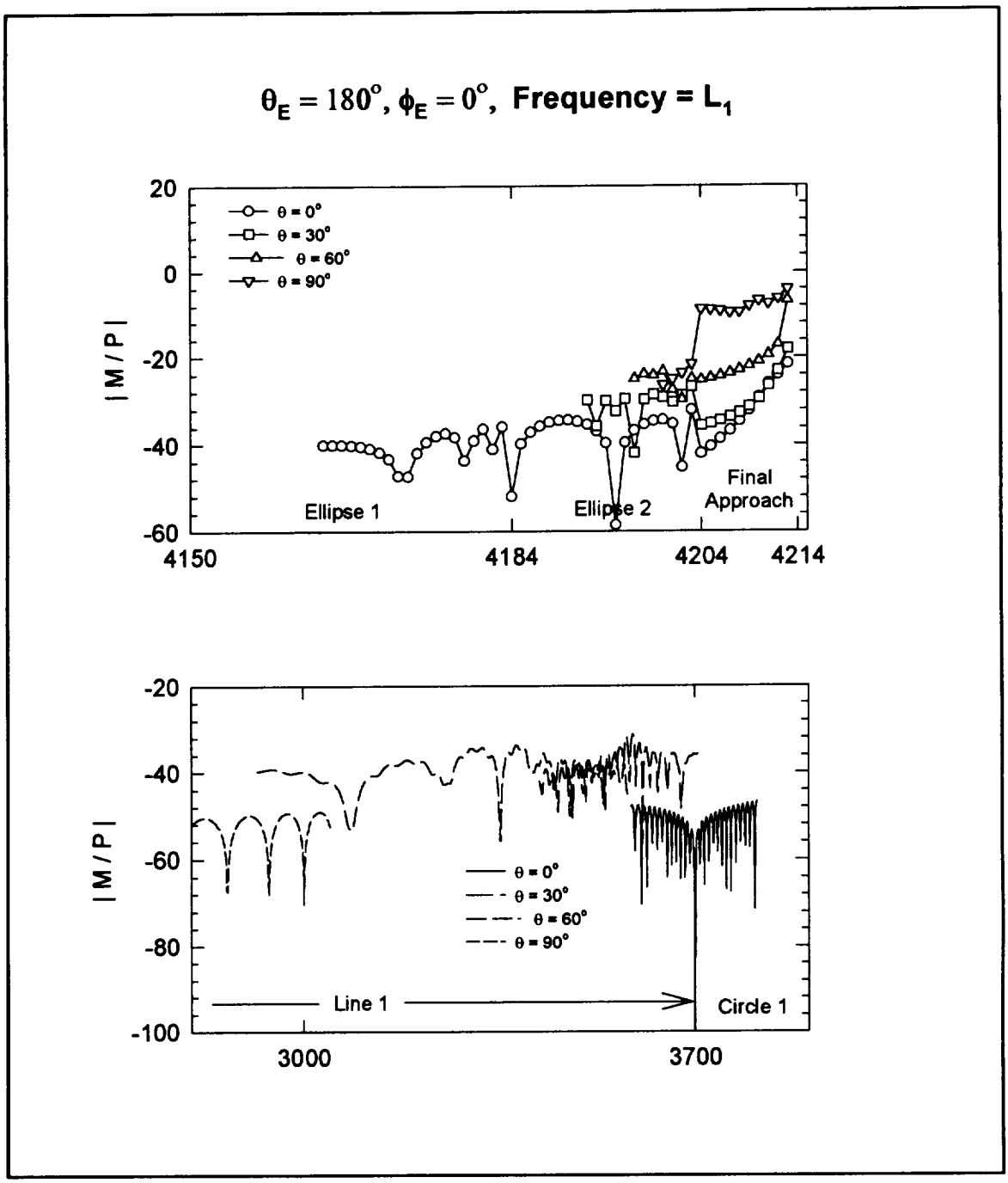


Figure 3-11. (100-x.dat)

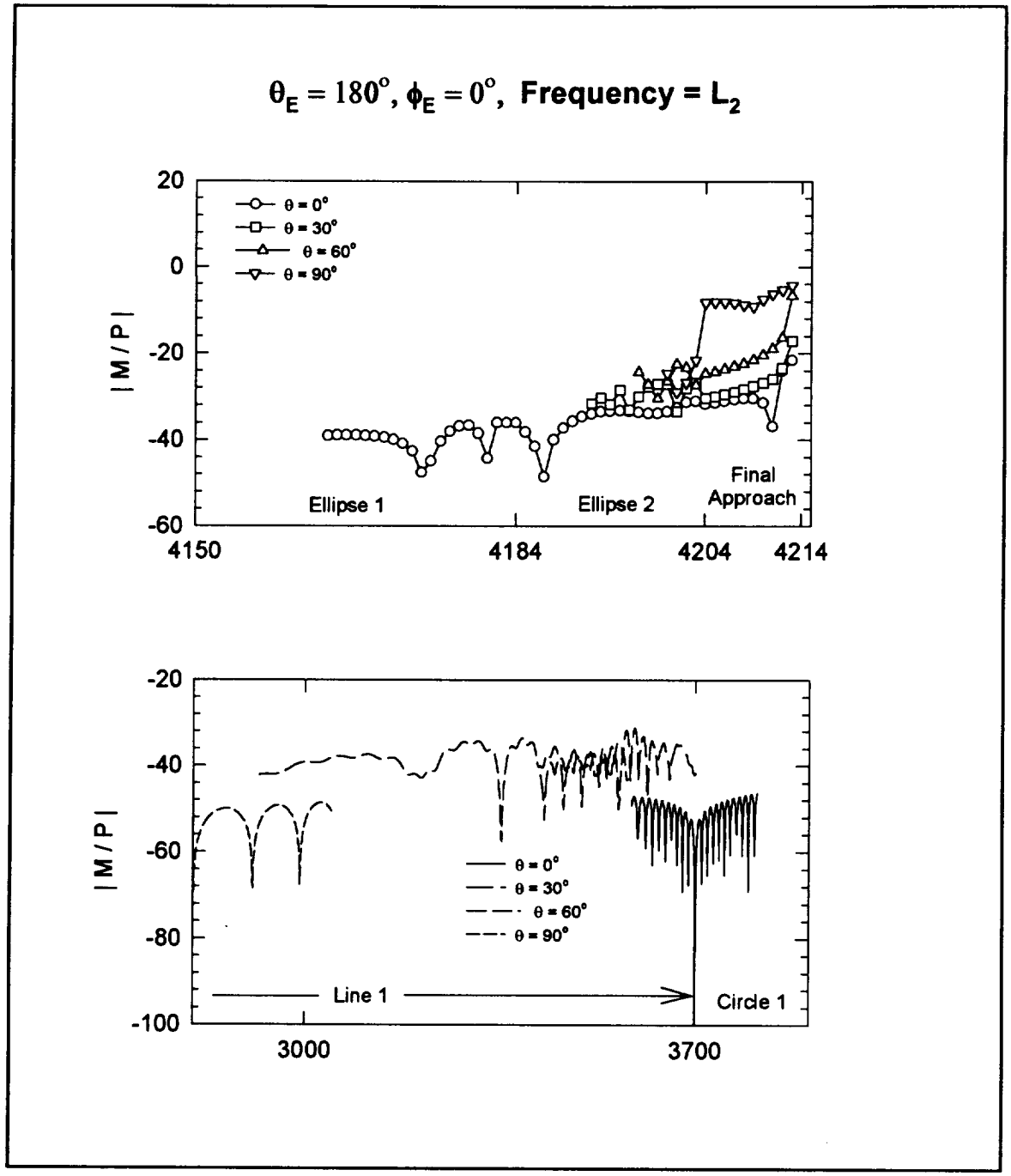


Figure 3-12. (200-x.dat)

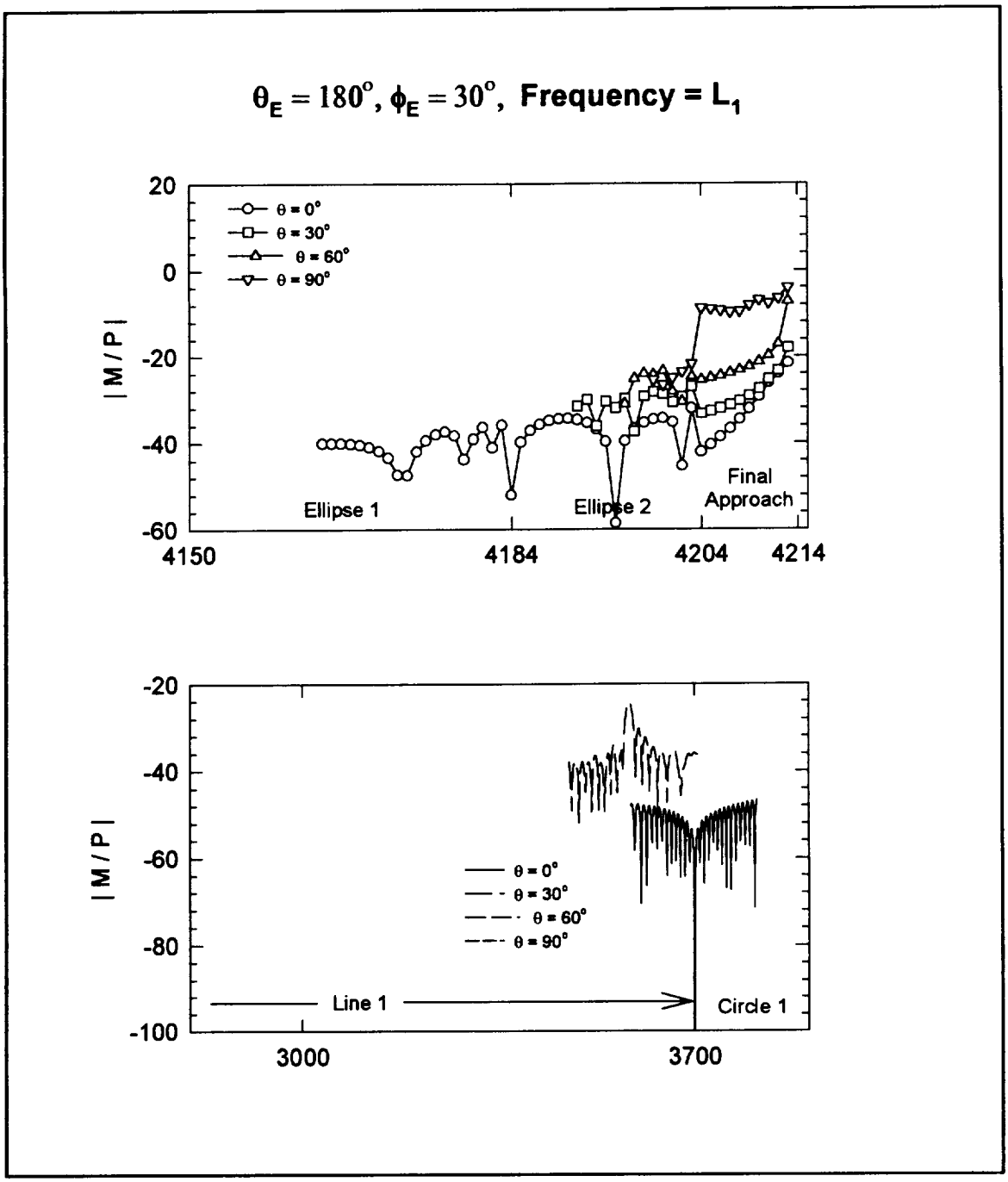


Figure 3-13. (103-x.dat)

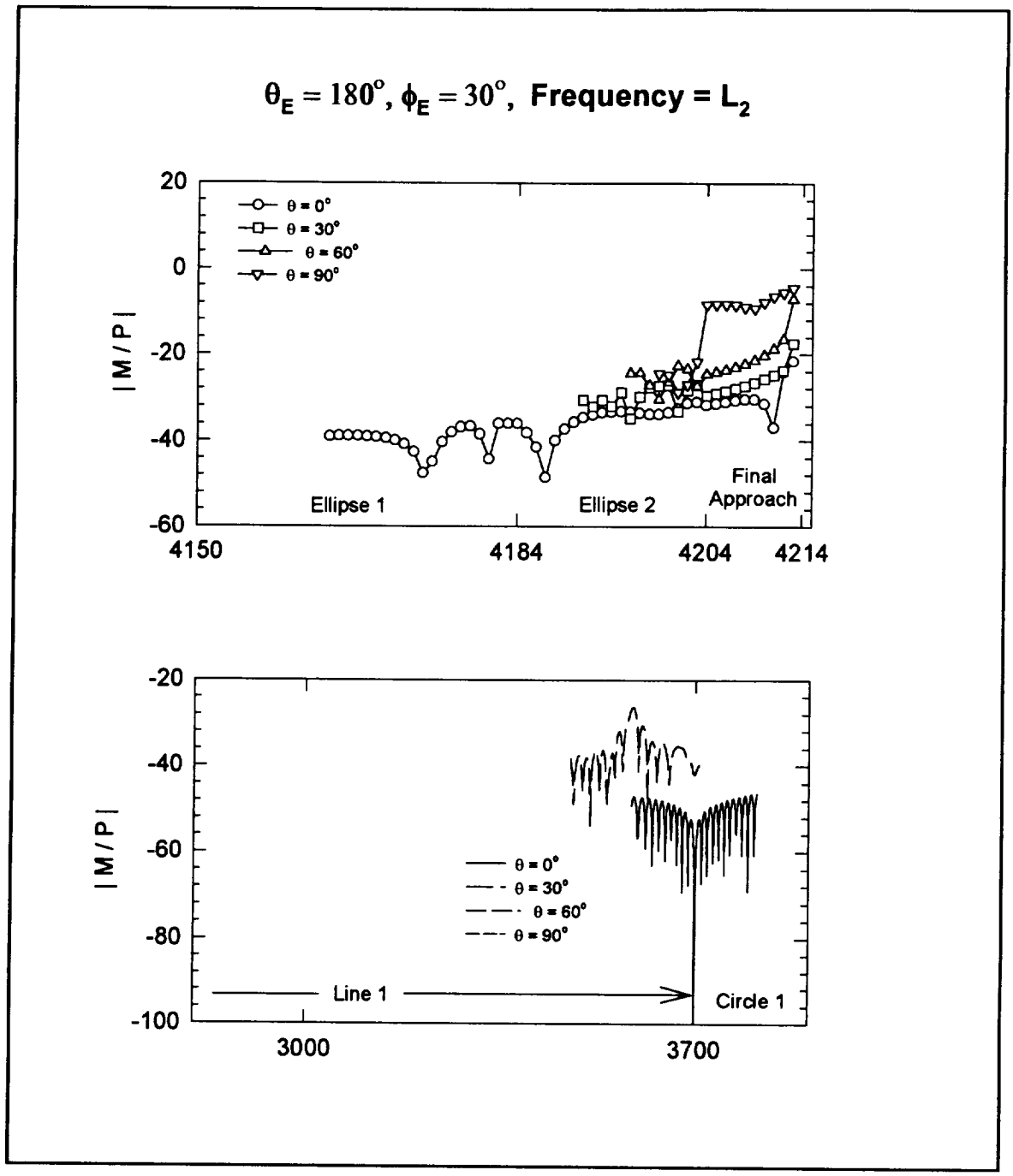


Figure 3-14.

(203-x.dat)

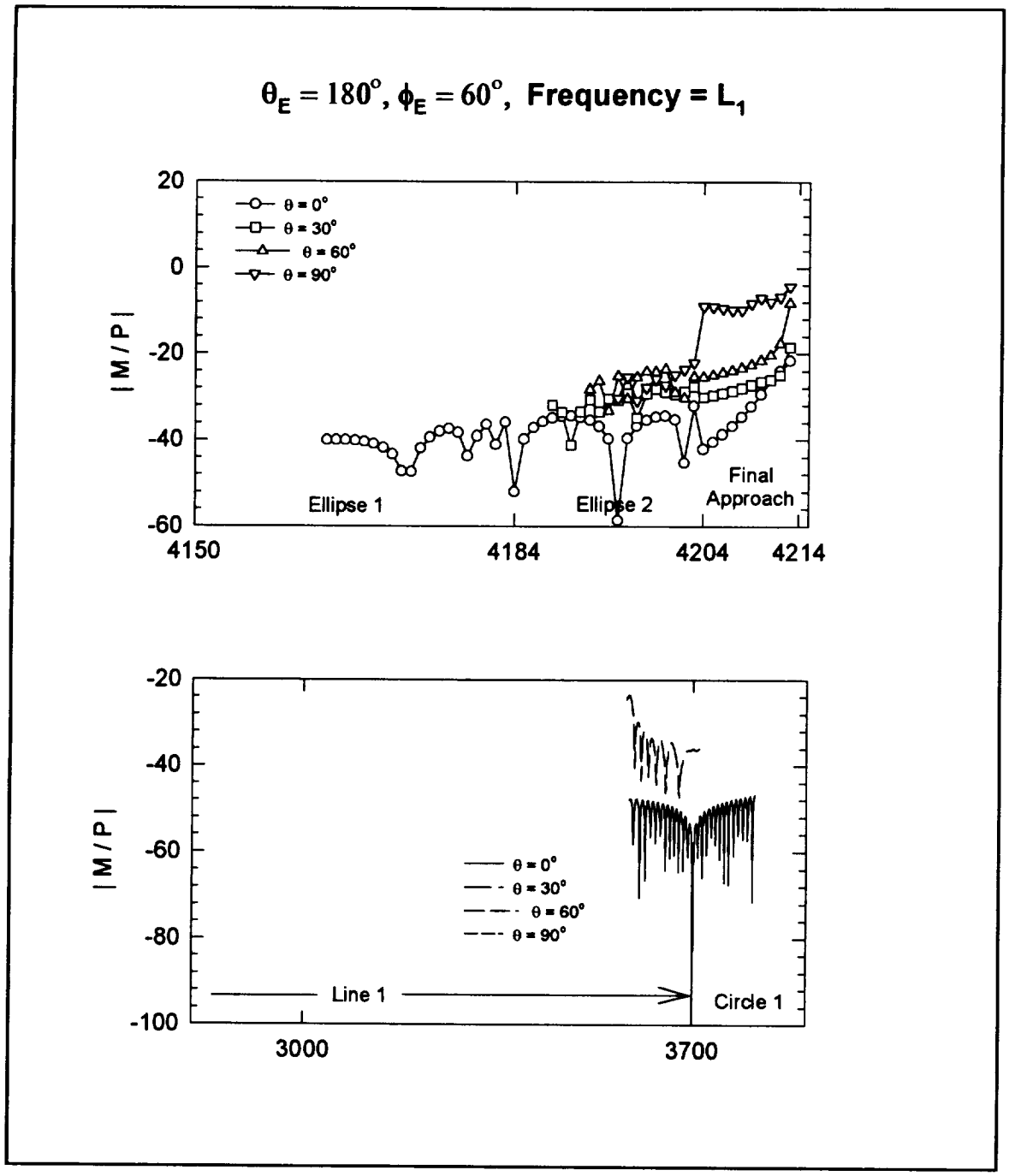


Figure 3-15.

(106-x.dat)

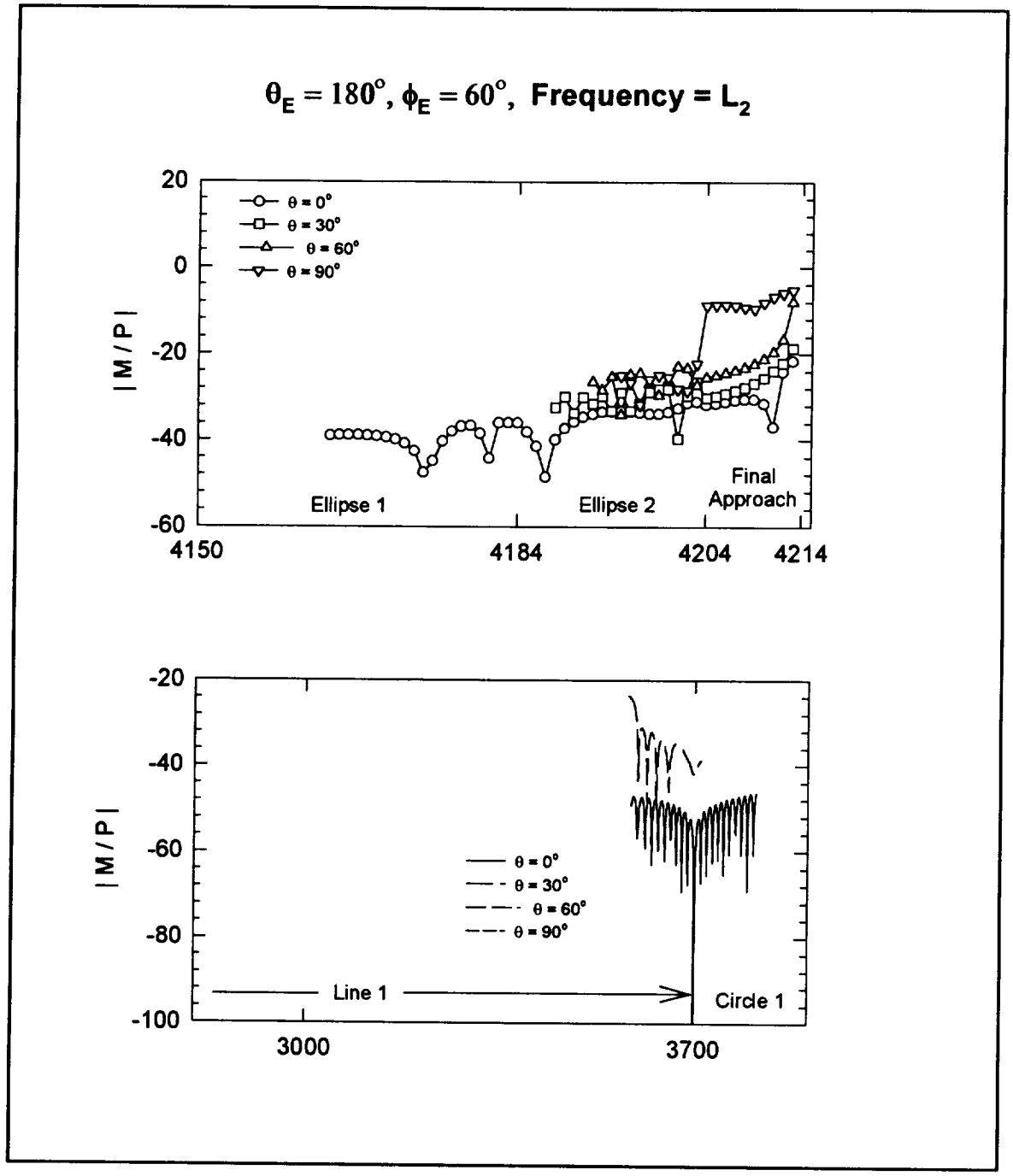


Figure 3-16.

(206-x.dat)

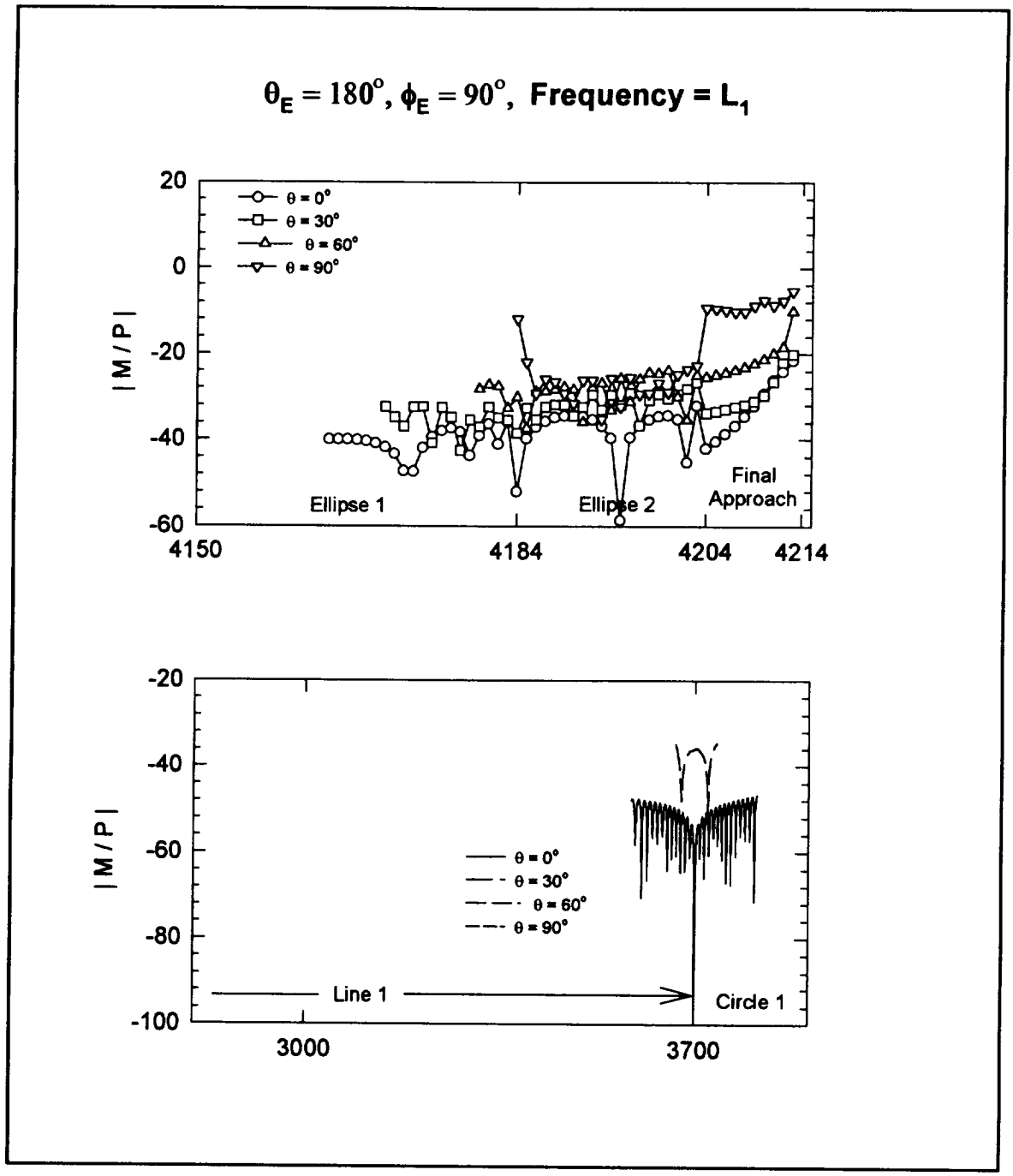


Figure 3-17.

(109-x.dat)

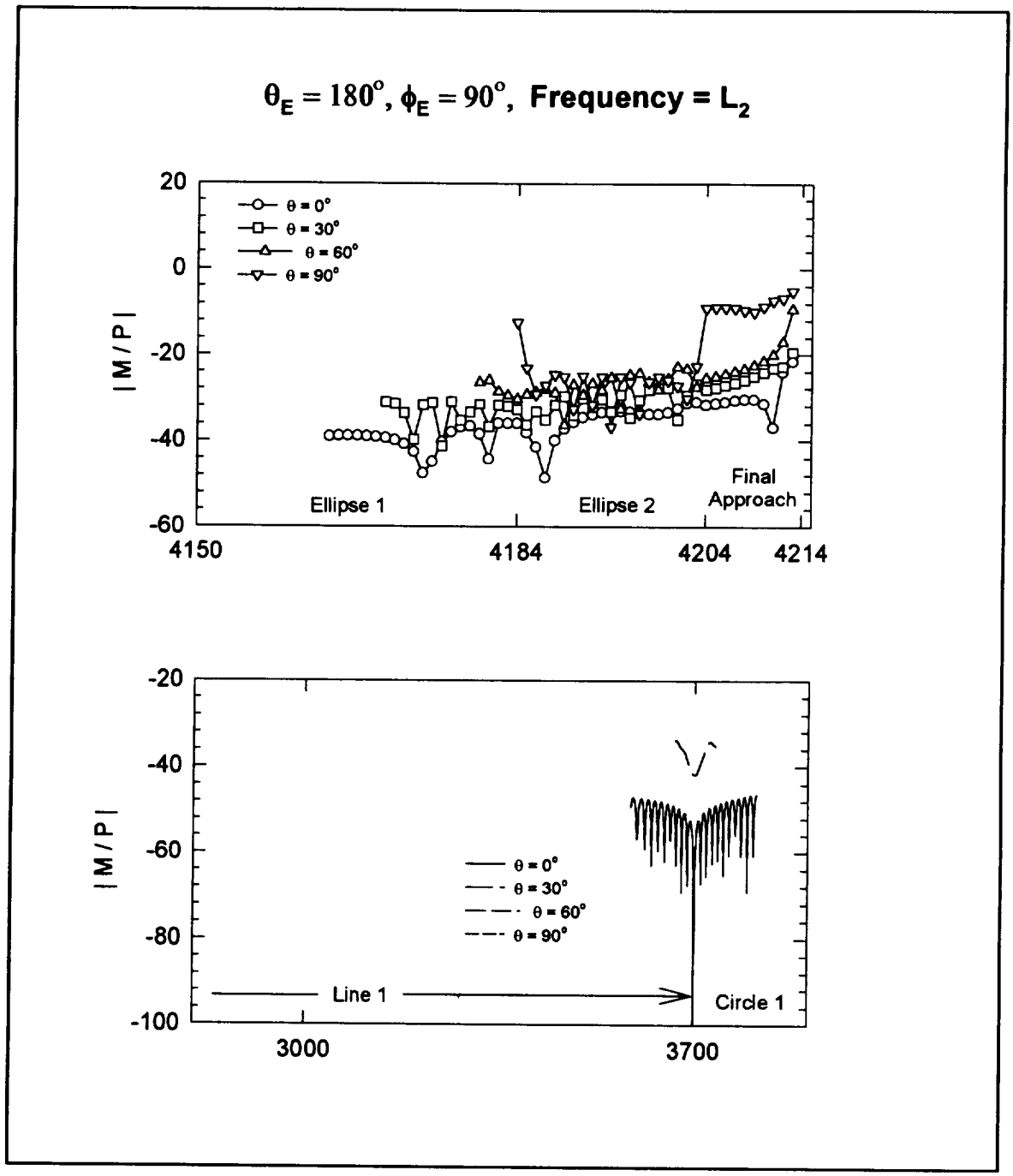


Figure 3-18.

(209-x.dat)

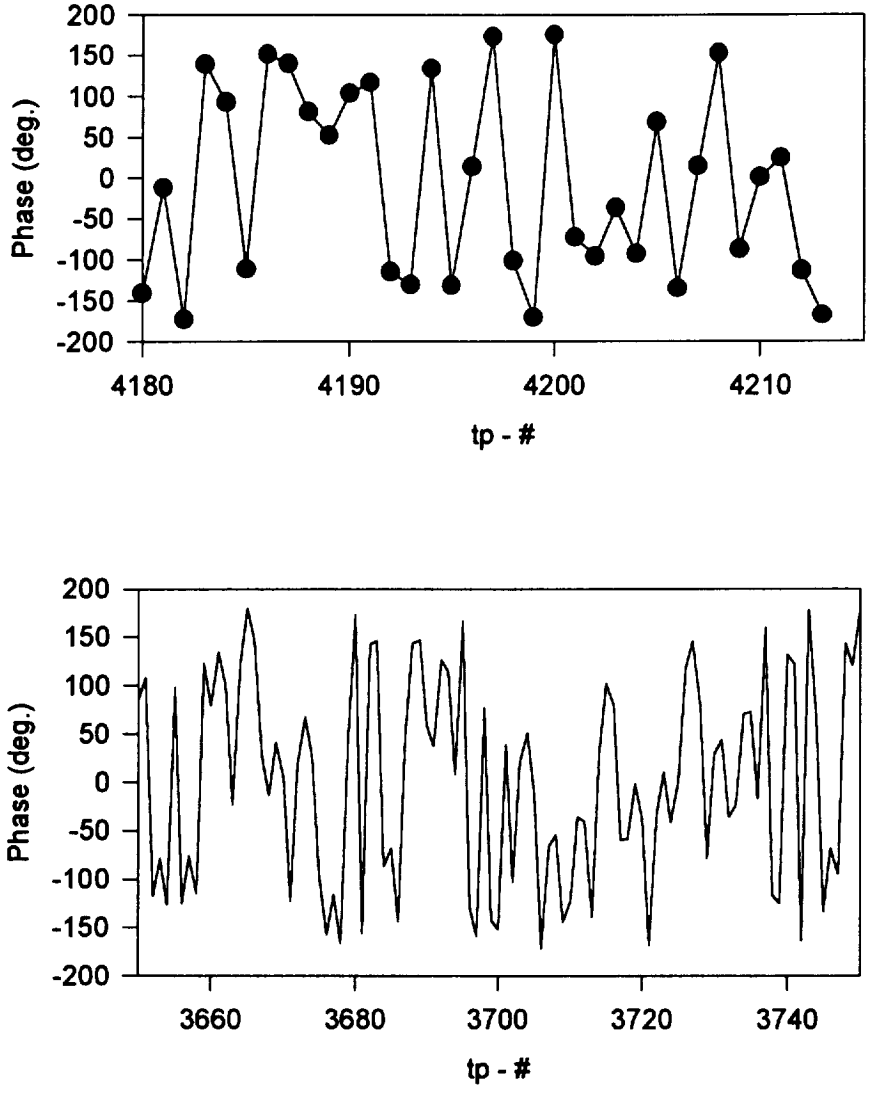


Figure 3-19. Phase of scattered field relative to direct field for case of Figure 3-11. Top graph is the phase during the approach along V_{BAR}, and the bottom graph is the phase centered around the forward scattering pattern.

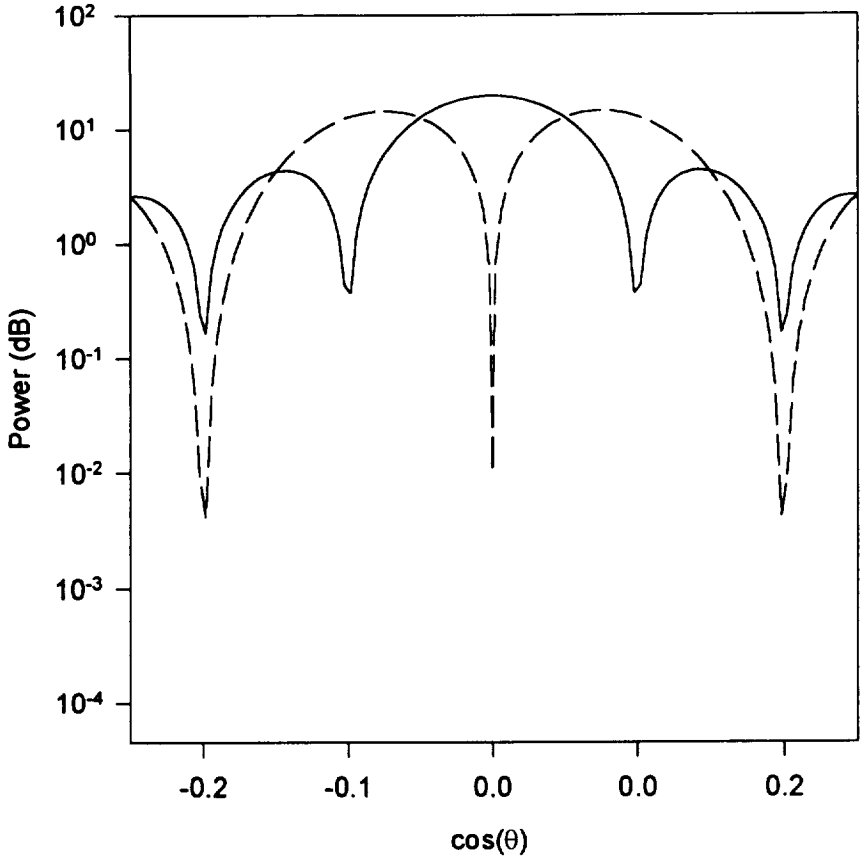


Figure 3-20. Sum (solid) and difference (dashed) patterns for a 20 element array. $\theta = 90°$ is broadside. Note the peak at 0 for a sum pattern, and a null for the difference pattern.

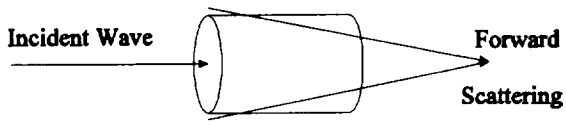


Figure 3-21. Example of forward scattering. Note that the wave appears to pass through the cylinder (and a null occurs).

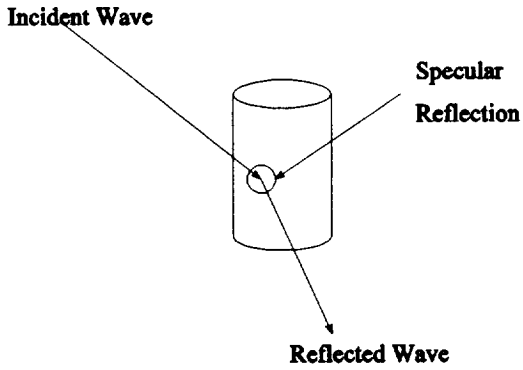


Figure 3-22. Example of specular reflection where wave appears to bounce off cylinder (a peak occurs).

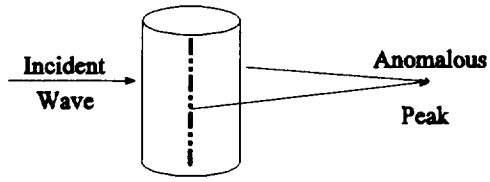


Figure 3-23. Example of the anomalous peak. Forward scattering is present, but a peak occurs. Suspected source is the discontinuity in current for one linear component..

4. Alternative Solution Techniques and Extensions of Present Solution

4.1 Method of Moments (Two-Dimensional Only)

A two-dimensional method of moments analysis has been performed for a cylinder with radius comparable to the SSF module. The polarization of the electric field is linear (rather than circular) in the code used. Figure 4-1 illustrates the geometry used. For frequency L_1, the radius is $13.13\lambda_1$ and for L_2 the radius is $10.23\lambda_2$. For each case, the segment length used along the cylinder has been chosen to be 0.097 wavelengths. The code for this solution has been written and the theory is provided in Appendix 7.2.

The reason for investigating this simplified version of the fundamental scattering problem is to study the current J_s that is induced on the cylinder. The MoM provides an exact solution, no assumptions or approximations have been made. This will verify the accuracy of the physical optics (PO) approximation for a simpler problem and illustrate the PTD corrections to the solution.

The integral equation used to solve this two dimensional problem is [15]:

$$\frac{k\eta}{4}\int_C J_z(\rho')H_0^{(2)}(k|\rho_m-\rho'|)dc' = E_z^i(\rho_m) \qquad (1)$$

where k is the wavenumber, η is the impedance of free space, C is the contour of the boundary between the perfect conductor and free space (in this case, C is a circle), J_z is the current density, $H_0^{(2)}$ is the Hankel function of the second kind and order zero (and is also the free space Green's function or propagator in two dimensions), ρ' is the location of the current, ρ_m is the location of the observation point on C, and E_z^i is the incident electric field which must be perpendicular to the paper. The unknown in (1) is the current density J_z. The current density is found by assuming pulse functions over small, equally sized intervals with unknown heights. The heights are found by solving the resulting matrix equation. For further information, see [15] and Appendix 7.2.

The case of L_1 requires over 600 pulses and L_2 requires over 800 pulses. The current for the case of frequency L_1 scattering off an infinite cylinder is shown in Figure 4-2, and the current for frequency L_2 is shown in Figure 4-3. The dashed line for the current density is the PO approximation for the current from 0 to 90°. The magnitude of the current follows the PO approximation value but deviates from the PO solution near and within the shadow region (defined as 90 to 180° in angle). As expected, the results for 180 to 360° are exactly symmetric. The phase is also very encouraging, since the phase varies linearly throughout the illuminated region. The slight increase in the slope of the phase as the angle is increased from 0 to 90° is due to the difference between the change in x and y along the circle.

The correction necessary to more precisely match the MoM solution is usually modeled

as diffraction from two points, each at 90 and 270°. These diffraction points could be accounted for using the physical theory of diffraction (PTD). The effect of this current is to radiate creeping waves that only influence the solution for observations near 180°. See Appendix 7.3.

These results for the simpler problem are encouraging. It can be concluded that the radius of the cylinder is large enough to permit use of PO as a good approximation to the true solution. However, it should be noted that the ends of the cylinder may cause significant changes to the PO solution. It is the ring at the top of the cylinder that would be the first (or dominant) term in the PTD analysis.

4.2 Geometrical Optics/Geometrical Theory of Diffraction (GO/GTD)

Geometrical Optics (GO), also known as ray techniques [13,20] have a wide variety of applications in engineering, including reflector design and laser cavity design. In GO, the rays define the phase of the field, and the amplitude is found by following intensity variations along each ray.

GO can be developed from Maxwell's equations using a high frequency approximation for cases where the wavenumber k approaches infinity (equivalent to frequency approaching infinity). Then, the Luneburg-Kline series [21] is used to expand the electric field; however, the terms in the series are $(\omega)^{-n}$ where $\omega = 2\pi f$ and n is an integer. An investigation of scattering from a wedge reveals that terms such as $n = -1/2$ are in fact significant. This revelation has led to the development of GTD. An excellent review of GTD for antenna analysis appears in [22].

The geometrical theory of diffraction, as first proposed by Keller [23] is a high frequency approximation that augments geometrical optics by computing the contribution from diffraction points. This theory breaks down (the electric field approaches infinity) as one computes the field near shadow boundaries. An excellent discussion of GTD for edges appears in [11].

A typical example is the scattering from a wedge, as shown in Figure 4-4. The wedge begins at the tip and is infinite in extent. The GO solution has an incident field in Regions I and II. The GO solution has a reflected solution in Region I. Region III has no incident or reflected field. The full GTD solution is the sum of the GO solution and contributions from diffraction points. GTD considers the tip of the wedge as a diffraction point with two contributions: one centered (in angle) along the incident shadow boundary (ISB), and one centered along the reflection shadow boundary (RSB). These contributions exist in all regions except for Region IV. As the angle of observation moves away from the shadow boundary, the diffracted fields become small. The rate that the diffracted fields decay depends on the distance from the diffraction point to the source and the observation.

The diffracted fields using Keller's diffraction coefficients also become singular at the shadow boundaries. The uniform theory of diffraction (UTD) corrects this singularity by recognizing the interaction between saddle points in the integration (this integration is the one

which is presently performed directly by using the code in Appendix 7.1). The interaction is included in Keller's original development through the use of a Fresnel transition function [24]. A second method of correcting for the singularity in Keller's original formulation is to return to Maxwell's equations and compute an asymptotic expansion of the solution directly. This is known as the uniform asymptotic theory of diffraction (UAT) [12] and results in a solution that also begins with Keller's formulation. UAT is not as prevalent in the literature when compared to UTD, but the solutions are very similar.

The UTD diffraction coefficients for a straight edge, a curved edge, and a smooth shadow boundary (such as the curved edge of the cylinder between the illuminated region and the shadow region) have been studied and codes developed for edges in order to evaluate the use of UTD in the present problem. As an example, the bistatic scattering from a circular disk has been computed and the results are shown in Figure 2-7 of Appendix 7.3.

The use of UTD for cases where both the source and observation are in the near field generally require higher order terms in the analysis. While this has not yet been formally investigated, the use of slope diffraction coefficients [25] is one possible method of formulating higher order terms. The use of the slope diffraction coefficients is prevalent especially when the first order diffraction coefficients are very small.

The second, and probably more rigorous choice is to find the higher order terms of the steepest descent integration directly. Note that this is often mentioned in the literature, but rarely accomplished due to the complexity of the problem.

The use of UTD can sometimes lead to caustic phenomena. A caustic occurs if there are a large number of GO rays and/or diffraction rays that contribute to a single observation point. For these cases, the singularity can be removed by converting the diffraction coefficients to an equivalent current [26]. This has been denoted the equivalent current method, or ECM, in the literature. More recently, the use of incremental length diffraction coefficients has become necessary for three-dimensional problems [27].

UTD is also capable of computing the effect of multiple diffraction [28]. In cases where a large number of closely spaced edges are present, and there is little specular reflection (as in low radar cross section targets), the effect of diffraction fields incident on a second edge and diffracting again can become significant. In the case of the SSF cylinder with solar panels, this may become a significant contribution to the total scattered field.

The use of UTD is severely hindered, however, if both the source and observation locations are far from the diffracting object. See, for example, Figure 1.5 of Appendix 7.3. That is not to say that one must be in the near field. In fact, we have found that the GTD solution breaks down. A GTD solution to the scattering from a two dimensional cylinder the size of the SSF is shown in Figure 2-11 of Appendix 7.3 (due to symmetry, only half of the cylinder is shown). Note that the solution should predominantly appear as a sin(x)/x function that is very

narrow. The peak in Figure 2-11 of Appendix 7.3 is anomalous, and occurs in the forward scattering direction. Backscatter is represented by the data at phi = 180°. The moment method solution to the scattering is shown in Figure 4-5. In the figure, backscatter is at $\theta = 180°$. Note also the slow rise to forward scattering at $\theta = 0°$. The absence of side lobe structure when compared to our data is due to 1) MoM solves only a two-dimensional problem for 2) linear polarization.

Therefore, it is concluded that UTD is not a very effective approach to the present problem. It would be more applicable to problems such as radiation from an antenna mounted on the solid rocket boosters or mounted on the space shuttle [22]. It should be pointed out that techniques to determine the optimal location of the antenna on a satellite can incorporate GTD in a very efficient fashion. The optimal choice may be parameterized as a location that results in a strong gain in a desired set of directions and nulls far removed from other specified directions.

4.3 Physical Optics/Physical Theory of Diffraction (PO/PTD)

The use of GTD and/or PTD has been compared since the inception of each technique. A set of reviews from the early 1970's discuss the advantages and disadvantages of each method [29-31]. Again, in the 1980's, RCS (radar cross section) prediction generated a number of review papers, where [32,33] are of particular interest.

The use of PTD solutions is nevertheless not as prevalent in the literature. First, the PO solution is computed using the physical optics approximation to the current as is done in Section 2.1.3. From the approximation to the current, the vector magnetic potential **A** is found by integrating over the current density, and the electric field **E** can be determined. The resulting **E** is only the scattered field. Note that the current is the geometrical optics solution for the current on the object. Thus, PO is indirectly derived from geometrical optics.

Once the PO solution has been obtained, the same (as in GTD) diffraction points are used to determine the PTD correction to the solution. However, the solution is now cast in a PO form, rather than a GO form. Hence, the diffractions are related, but are computed in a different fashion. The shadow boundaries do still exist, but by using PO/PTD the singularities at these boundaries are avoided. In other words, by computing the integral directly, the saddle points and their interaction are determined directly to avoid these singularities.

This use of diffraction coefficients is also restricted to the far field of the scattering object. The method of equivalent currents for edges in diffraction has been studied [19,34,35] where the only remaining singularity is in the exact forward scattering direction, also known as the Ufimtsev singularity.

Another characteristic of PO is the existence of a discontinuity in the current at the shadow boundary on the object. This was seen in the data collected as an anomalous peak at forward scattering. This can be corrected (in the far field) using the techniques discussed in [36]. There

are other second order effects such as creeping waves, etc., that are thoroughly discussed in [37] Usually, these second order effects are significant only if the first order (PO + PTD) is small.

The use of PO/PTD for near field investigations may also require higher order terms of the PTD equivalent currents. This problem is inherent to both UTD and PTD due to the simplification used, namely that the scattering is considered as a localized effect. The use of PTD avoids caustics, unlike GTD. However, the theory presented in [19,34,35] accounts for both the far field and the Fresnel field solution to the wedge problem shown in Figure 4-3.

The most obvious advantage to the authors of PTD over GTD is that neither the source or the observation need be close to the diffracting object. In fact, for many studies of radar cross section (RCS), PO/PTD is preferred for this very reason [37]. One major disadvantage of PO/PTD is that multiple reflections have not been and would be difficult to incorporate into the analysis. This is seen as important to the present problem, particularly when the solar panels are added to the scatterer.

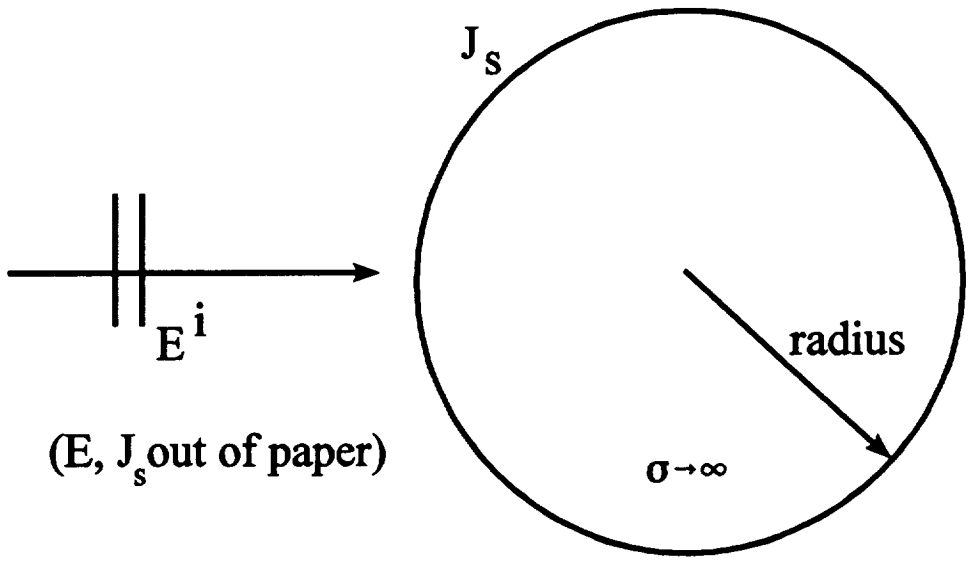


Figure 4-1. Geometry for MoM analysis.

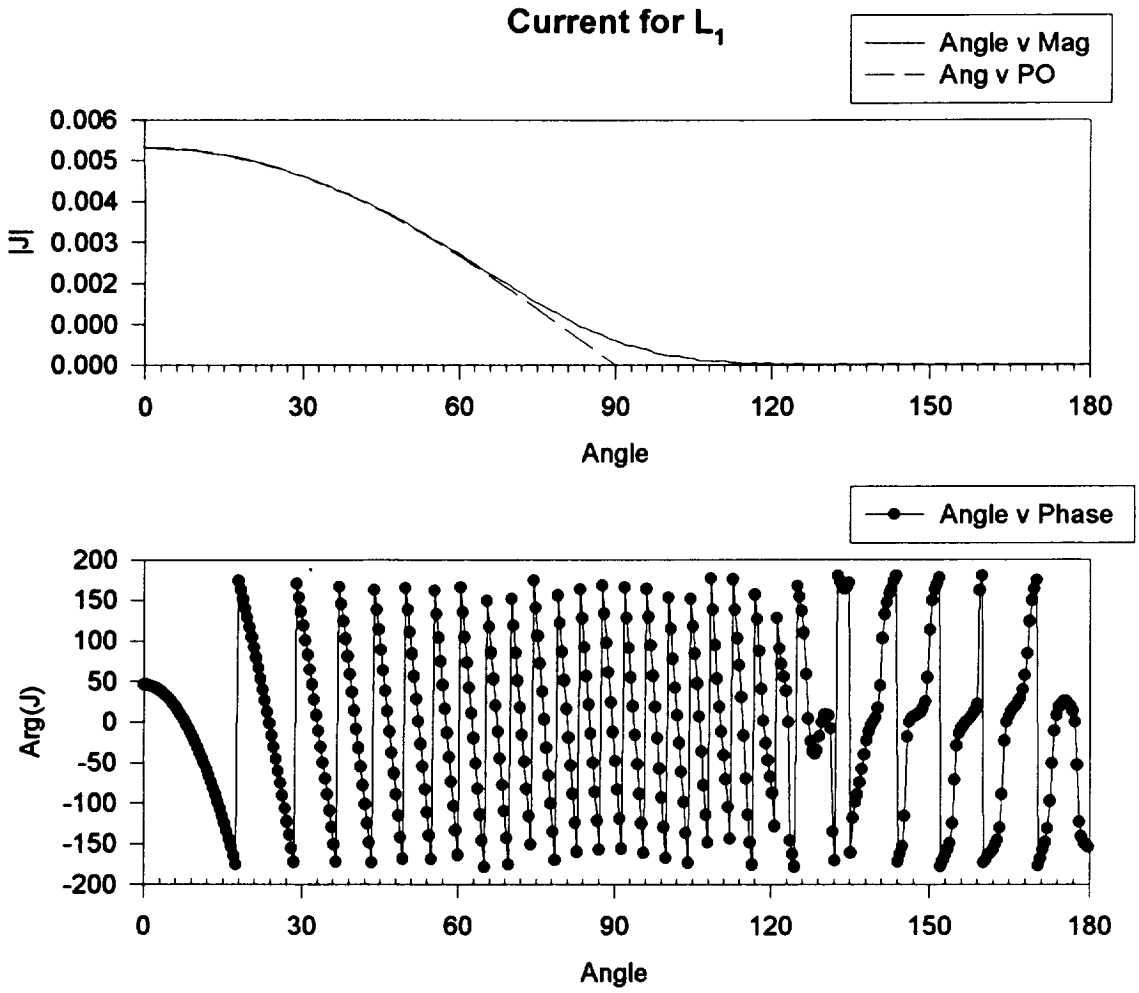


Figure 4-2. Magnitude (top) and Phase (bottom) of current on 2-D cylinder found using the method of moments for the frequency L_1. The PO approximation to the magnitude is shown as a dashed line.

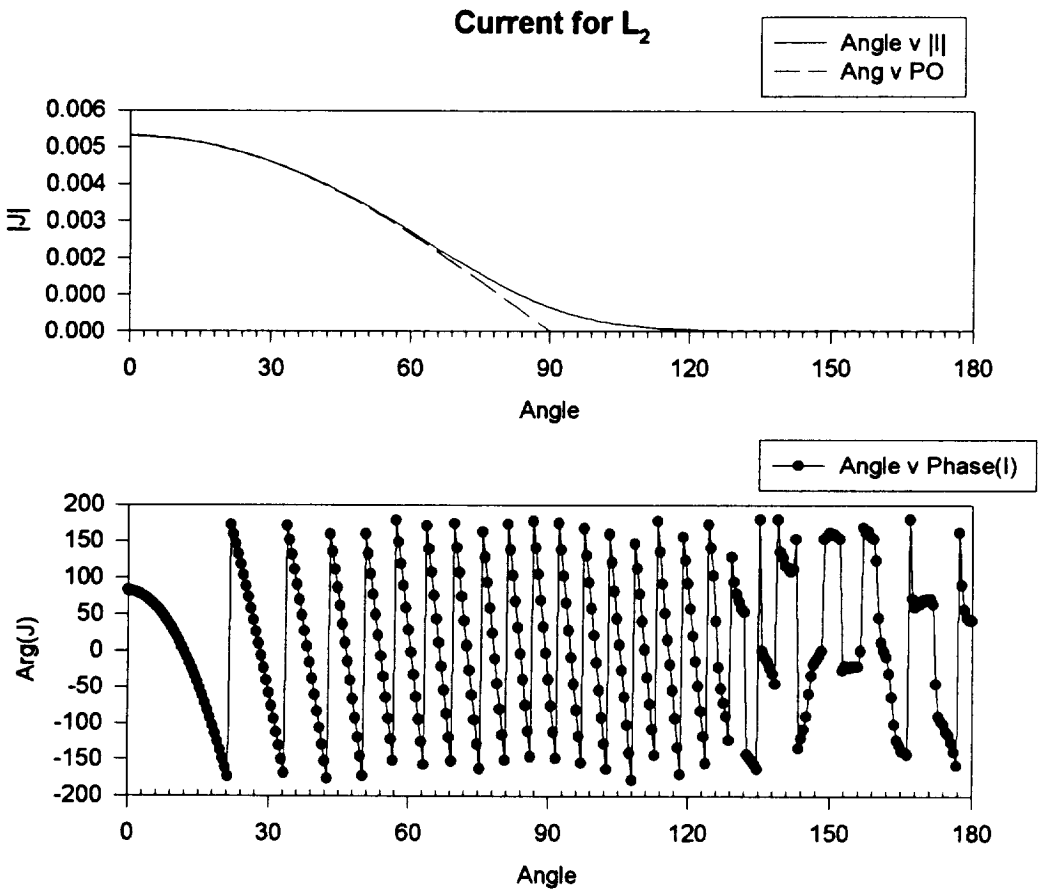


Figure 4-3. Magnitude (top) and Phase (bottom) of current on 2-D cylinder found using the method of moments for the frequency L_2. The PO approximation to the magnitude is shown as a dashed line.

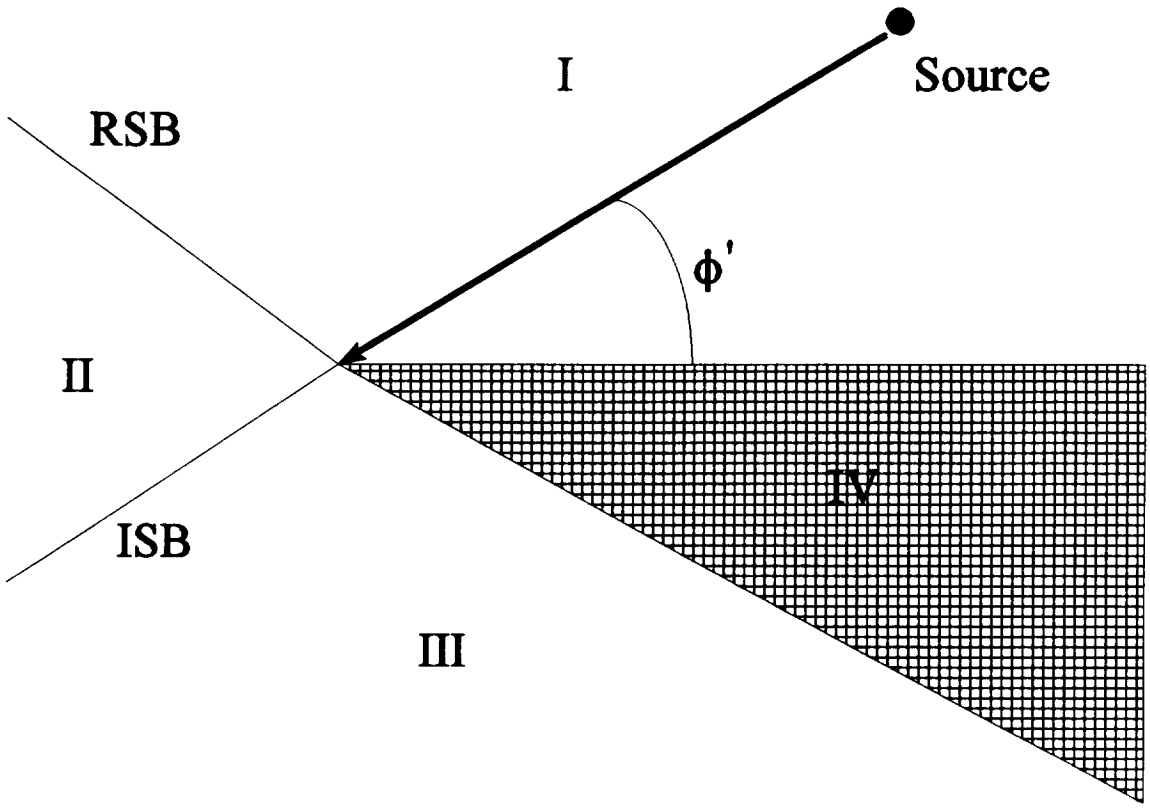


Figure 4-4. Typical wedge diffraction geometry.

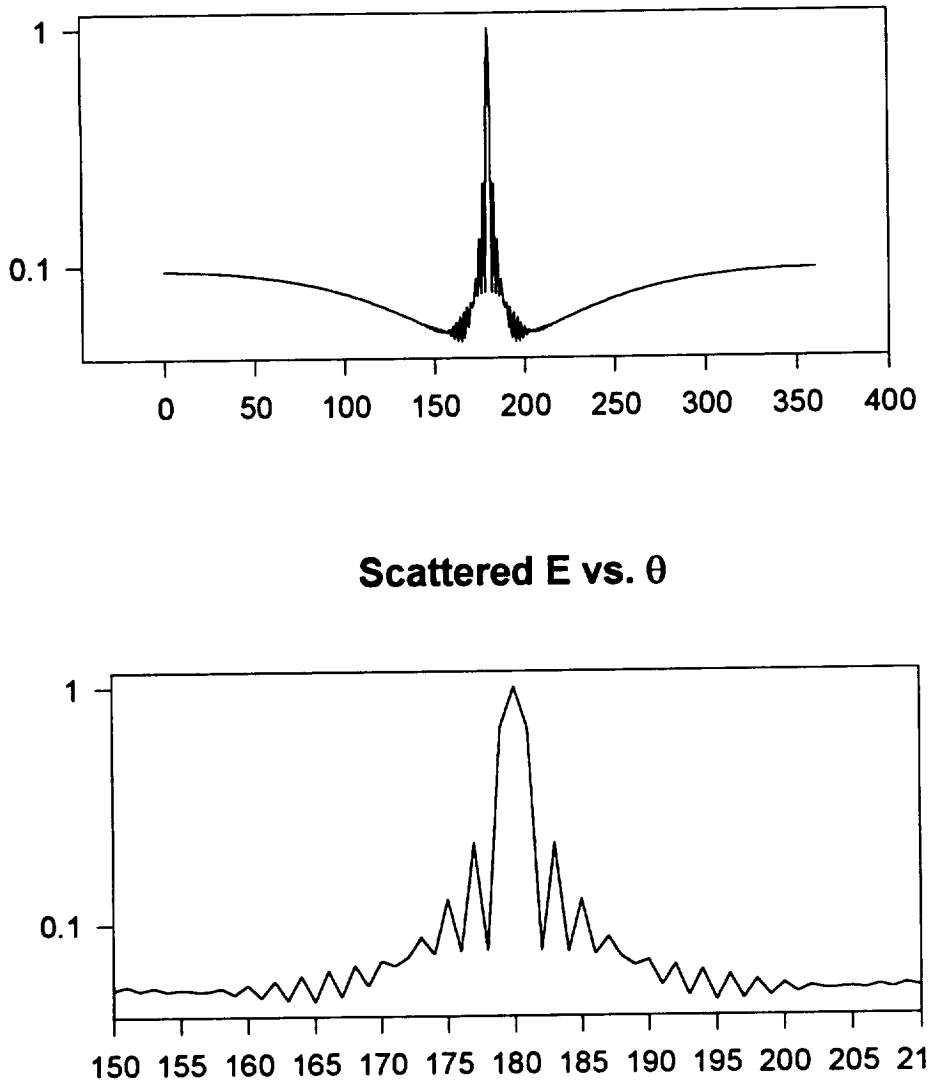


Figure 4-5. Far field pattern of 2-D cylinder found using MoM. Top: full range of angle. Bottom: Expanded view of sin(x)/x behavior of scattered field.

5. Conclusions

The scattering from a space station cylinder has been modeled using physical optics with no assumptions except the form of the current induced on the surface of the cylinder. The computations were performed over a typical remotely piloted CTV auto-rendezvous and capture trajectory.

During most of this trajectory, the GPS system will be used to remotely pilot the CTV. The trajectory traverses within the far field, enters the Fresnel region, then exits the Fresnel region and enters the near field region. The Fresnel region is when the distance from SSF to CTV is between 1.1 km to 40 meters. The near field begins roughly 40 meters from the space station Freedom. Since the optical guidance system will take over sometime before there is 100 meters between the CTV and SSF, the GPS system will be used only in the Fresnel and far field regions.

Data has been collected using this simulation model for a span of possible geometries. The geometry is defined by (1) the angle between the GPS satellite and the axis of the cylinder, and the angles (referenced by the plane containing the GPS satellite and the SSF) from SSF to the earth. The integrations are performed using an N-dimensional Romberg integration. Once the scattered field is found, the RCP component in the direction of the CTV is determined and normalized relative to the signal strength of the direct signal.

The data indicates that the path difference between the direct signal and the scattered signal is less than 300 meters during the final stages of AR&C maneuvers, and during isolated sections of the trajectory when the GPS satellite, CTV, and SSF are approximately collinear. The scattered RCP signal rises slowly during the final stages of AR&C maneuvers to maximum values of approximately +2 dB or -4 dB for end-on approach and side approach, respectively. These values are only for particular locations of the GPS satellite, and the data for other locations are 10 to 20 dB below this maximum.

During the earlier stages of AR&C maneuvers when the three objects are roughly collinear, there are three scattering mechanisms: specular reflection (where the signal appears to "bounce" off the SSF); forward scattering (where the signal appears to pass through or diffract around the SSF); and a special case of forward scattering where an anomalous peak occurs. This peak is denoted anomalous because it is due to the discontinuity of the approximation to the current. Specular reflections can have peaks in the range of -20 dB to -30 dB. Forward scattering is smaller with maximum values near -35 dB. The anomalous peak value is immaterial, but the scattering near this value has a maximal value of -44 dB. In some cases, the GPS satellite, the CTV, and SSF form a line for much of the early AR&C maneuvers. For this case, the maximum value is roughly -55 dB.

Alternative solutions have also been investigated. A method of moments solution for a two-dimensional cylinder has been computed. The method of moments makes no approximations. The results indicate that the physical optics current is a good approximation when compared to the

exact numerical solution. However, only one linear polarization has been investigated, and it is well known that the other linear polarization (to create RCP) has a strong discontinuity. This is the source of the anomalous peak mentioned above.

The use of GTD and PTD has also been investigated. It has been found that GTD would not be appropriate due to the large distances involved. For GTD to be accurate, one of the source or the observation must be in the near field. The use of PTD is a natural extension to the work accomplished in this report since PTD is a correction to the physical optics approximation. However, the discontinuity in the current has only been corrected for far field observations. In addition, once solar panels are added to the problem, the questions of shadowing and multiple reflections have not been addressed in the literature.

6. References

1. M. W. McDonald, "Multipath Effects in a Global Positioning Satellite System Receiver," 1992 NASA/ASEE Summer Faculty Fellowship Program Final Report, Marshall Space Flight Center, University of Alabama, August 1992

2. Todd Freestone, Private Communication.

3. J. Hurn, "GPS: *A Guide to the Next Utility*, Sunnyvale, CA: Trimble Navigation, 1989.

4. I. A Getting, "The Global Positioning System," *IEEE Spectrum*, pp. 36-47, December 1993.

5. W. Herbert Sims, III, Private Communication.

6. F. Nathanson, *Radar Design Principles*, 2nd Edition, New York: McGraw-Hill, 1990.

7. J. D. Jackson, *Classical Electrodynamics*, 2nd Edition, New York: John Wiley & Sons, 1975.

8. J. J. H. Wang, *Generalized Moment Methods in Electromagnetics*, New York: John Wiley & Sons, 1991.

9. K. S. Kunz, and R. J. Luebbers, *The Finite Difference Time Domain Method for Electromagnetics*, Ann Arbor, MI: CRC Press, 1993.

10. G. L. James, *Geometrical Theory of Diffraction for Electromagnetic Waves*, Third Edition Revised, London: Peregrinus, 1986.

11. C. A. Balanis, *Advanced Engineering Electromagnetics*, New York: John Wiley & Sons, 1989, Chapter 13.

12. S. W. Lee, G. A. Deschamps, "A Uniform Asymptotic Theory of EM Diffraction By a Curved Wedge," *IEEE Trans. Antennas and Propag.*, Vol. 24, No. 1, Jan. 1976, pp. 25-34.

13. M. Born, and E. Wolf, "Principles of Optics," 4th edition, New York: Pergamon, 1970.

14. R. F. Harrington, "Time Harmonic Electromagnetic Fields," New York: McGraw-Hill, 1961.

15. C. A. Balanis, *Advanced Engineering Electromagnetics*, New York: John Wiley & Sons, 1989, Chapter 12.

16. P. Y. Ufimtsev, "Method of Edge Waves in the Physical Theory of Diffraction," translated by U.S. Air Force Foreign Technology Division, Wright-Patterson AFB, OH, September 1971.

17. C. A. Balanis, *Antenna Theory: Analysis and Design*, New York: John Wiley & Sons, 1982, Chapter 4.

18. P. J. Davis and P. Rabinowitz, *Methods of Numerical Integration*, 2nd Edition, New York: Academic Press, 1984.

19. A. Michaeli, "Elimination of Infinities in Equivalent Edge Currents, Part I: Fringe Current Components," *IEEE Trans. Antennas and Prop.*, Vol. 34, July 1986, pp. 912-918.

20. G. A. Deschamps, "Ray Techniques in Electromagnetics," *Proc. of IEEE*, Vol. 60, No. 9, September 1972, pp. 1022 - 1035.

21. L. B. Felsen and N. Marcuvitz, *Radiation and Scattering of Waves*, Englewood Cliffs, NJ: Prentice-Hall, 1973.

22. P. H. Pathak, "High Frequency Techniques for Antenna Analysis," *Proc. of IEEE*, Vol. 80, No. 1, January 1982, pp. 44 - 65.

23. J. B. Keller, "Geometrical Theory of Diffraction," *J. Opt. Soc. Amer.*, Vol. 52, No. 2, Feb. 1962, pp. 116-130.

24. R. G. Kouyoumjian, P. H. Pathak, "A Uniform Geometrical Theory of Diffraction for an Edge in a Perfectly Conducting Surface," *Proc. IEEE.*, Vol. 62, No. 11, Nov. 1974, pp. 1448-1461.

25. R. G. Kouyoumjian, "The Geometrical Theory of Diffraction and its Application," in *Numerical and Asymptotic Techniques in Electromagnetics*, R. Mittra (Ed.), New York: Springer, 1975, Chapter 6.

26. C. E. Ryan Jr., L. Peters Jr., "Evaluation of Edge Diffracted Fields Including Equivalent Currents for Caustic Regions," *IEEE Trans. Ant. and Prop.*, Vol. 7, March 1970, pp. 292-299.

27. K. M. Mitzner, "Incremental Length Diffraction Coefficients," Aircraft Division Northrop Corp., Tech. Rep. No. AFAL-TR-73-296, April 1974.

28. V. Twersky, "Multiple Scattering of Waves and Optical Phenomena," *J. Opt. Soc. Amer.*, Vol. 52, No. 2, Feb. 1962, pp. 145-171.

29. V. A. Borovikov, and B. Y. Kinber, "Some Problems in the Asymptotic Theory of Diffraction," *Proc. of IEEE*, Vol. 62, No. 11, November 1974., pp. 1416 - 1437.

30. M. A. Plonus, R. Williams, and S. C. H. Wang, "Radar Cross Section of Curved Plates Using Geometrical and Physical Diffraction Techniques," *IEEE Trans. on Antennas and Propagation*, Vol. 26, No. 3, May 1978, pp. 488 - 493.

31. E. F. Knott, and T. B. A. Senior, "Comparison of Three High-Frequency Diffraction Techniques," *Proc. of IEEE*, Vol. 62, No. 11, November 1974, pp. 1468 - 1474.

32. N. N. Youssef, "Radar Cross Section of Complex Targets," *Proc. of IEEE*, Vol. 77, No. 5, May 1989, pp. 722 - 734.

33. E. F. Knott, "A Progression of High-Frequency RCS Prediction Techniques," *Proc. of IEEE*, Vol. 73, No. 2, February 1985, pp. 252 - 264.

34. A. Michaeli, "Equivalent Edge Currents for Arbitrary Aspects of Observation," *IEEE Trans. Antennas and Prop.*, Vol. 32, pp. 252-258, 1984.

35. A. Michaeli, "Elimination of Infinities in Equivalent Edge Currents, Part II: Physical Optics Components," *IEEE Trans. Antennas and Prop.*, Vol. 34, No. 8, Aug. 1986, pp. 1034-1037.

36. I. J. Gupta, and W. D. Burnside, "A Physical Optics Correction for Backscattering from Curved Surfaces," *IEEE Trans. on Antennas and Propagation*, Vol. 35, No. 5, May 1987, pp. 553 - 561.

37. D. P. Bouche, J-J. Bouquet, H. Manene, and R. Mittra, "Asymptotic Computation of the RCS of Low Observable Axisymmetric Objects at High Frequency," *IEEE Trans. on Antennas and Propagation*, Vol. 40, No. 10, October 1992, pp. 1165 - 1174.

7. Appendices

7.1 Physical Optics Code

The code consists of a main function and seven files, each of which contains one or more functions. A header file, "cplxmath.h" is also necessary. The files used are:

JVE4MAIN.C:	Contains the main function, and controls execution of the program.
HED-FT.C:	Prints a header banner to the output file.
NDIMRI.C:	Computes an N-dimensional integration of "fcnA".
FCN-A.C:	Computes the integrand that is integrated.
RCP.C:	Computes the RCP component of the scattered field.
EREF.C:	Computes the reference field (direct signal from the GPS satellite).
PATH.C:	Computes the location of the CTV in the trajectory.
CPLXMATH.C:	A collection of functions used to create complex math routines that are ANSI compatible.

The above code has been used on a Motorolla Delta Series 8000 Model 8640 Computer System (with four 88000 RISC processor chips and 128 MBytes of RAM and is rated at 152 MIPS) and on a 486 IBM compatible machine with a 50 MHz clock and 8 MBytes of RAM.

The output file from the code above is named GPS5.OUT. A post processor to filter the data has also been used on the machines mentioned above and is also given in the following code listings. The code has only one file: JOVE-PP.C.

The output file GPS5.OUT contains columns of data. The first column is the data point number in the trajectory. The second and third columns are the magnitude and phase (respectively) of the relative multipath signal. The fourth column is the second column in dB. The final column may contain 1 or 0 or -1, depending on the integration routine. A 1 denotes the integration converged successfully, a 0 denotes the integration was not needed (only if either the top or the curved side of the cylinder is not illuminated) and a -1 indicates that the integration has not converged. Every integration that has been attempted in this project has converged. In fact, one should be aware that the post processor is not presently capable of handling this situation and it is not known what the post processor will do in this situation.

JVE4MAIN.C

```
/*
-------------------------------------------------------
      Version 5.0:  Inclusion of Romberg integration...
                    After verification of code, will
                    be placed on apollo.
      Version 5.0a: Apollo version of 5.0.
*/

/*
-------------------------------------------------------
      Version 4.0:  Clean up of complete code,
                    This version also prints
                    0 for |M/P| and path difference
                    for arg(M/P) if path difference is
                    greater than 300.
-------------------------------------------------------

/*
-------------------------------------------------------
      Version 3:  Set up to user enter values of theta
                  and theta_e (also on motorolla
                  in older version where Z varies
                  from -300 to 300).  PC version
                  also includes path computations.
-------------------------------------------------------
*/

/*
-------------------------------------------------------
      Version 2:  includes the RCP
                  component calculation,
            includes comparison of reference signal
            and M/P signal
-------------------------------------------------------
*/

/*
-----------------------------------------------
      Test driver for Jove: GPS/M-P problem
-----------------------------------------------
*/


#include <stdio.h>
#include <time.h>
#include <math.h>
#include "/usr/users/richiej/bin/cplxmath/cplxmath.h"

#define TOL 1e-3
#define DX  10
#define L1  1.57542e9
#define L2  1.22760e9

/*frequency=1.57542e9;  L1=1.57542 GHz, and L2=1.2276 GHz   */

void print_banner(FILE *fp);
struct Complex matrix_element(int n,int j, double tol);
struct Complex RCP(struct Complex ex,struct Complex ey, struct Complex ez);
struct Complex direct(double theta, double theta_e, double phi_e,
            double lambda);
void path(int npath);
```

```
int dx,key;
double gama,delta,L,radius,lambda,theta_e,phi_e;
double X,Y,Z;


main()
{
char gch;
int termstat=0;
int l,n,m,j,first,last,npath,n_end,n1,n2,n3,n4;
int kk[6]={0,0,0,0,0,0};
double frequency,eps=8.854e-12,theta,temp,tol;
struct Complex zero,integral_value,constant,const1,E[5][3],totalE[3];
struct Complex rcp_comp,Eref;
FILE *out;
time_t t;

if ((out = fopen("gps5.out", "wt"))
    == NULL)
{
   fprintf(stderr, "Cannot open output file.\n");
   return 1;
}


zero.real=0;
zero.imag=0;

tol=TOL;
dx=DX;

/* ----------------------------------------
      Set up operating frequency
*/
printf("Enter '1' for L1, and '2' for L2:  ");
scanf ("%d",&l);

frequency=L1;
if(l==2)
      frequency=L2;

/* ----------------------------------------
      Set up incident angle
*/
theta=-90;
while( (theta<0)||(theta>90) )
      {
      printf("Enter value of theta (0->90):  ");
      scanf ("%lf",&theta);
      }

gama=sin(theta*PI/180.);
delta=cos(theta*PI/180.);

/* ----------------------------------------
      Set up angle of object w. r. t.
      earth, by defining theta_e, phi_e
*/
printf("Enter value of theta_e:  ");
scanf ("%lf",&theta_e);

printf("Enter value of phi_e:  ");
scanf ("%lf",&phi_e);
```

```
/* ---------------------------------------
      Set up object size (cylinder)
*/
L=9.;
radius=2.5;

/* ---------------------------------------
      Set operating wavelength
*/
lambda=3.e8/frequency;

/* ---------------------------------------
      Initialize constants for integral
*/
constant.imag=-4*PI*2*PI*frequency*eps;
constant.imag=1./constant.imag;
constant.imag*=(2./377.);

constant.real=0;

temp=PI*delta*L/lambda;
const1.real=cos(temp);
const1.imag=sin(temp);

/* ++++++++++++++++++++++++++++++++++++++++++++++++++++

      in E[5][3],
the first index is the geometry number:
            0:  top of cylinder
            1:  side of cylinder
            etc.
the second index is the component:
            0:  x
            1:  y
            2:  z
++++++++++++++++++++++++++++++++++++++++++++++++++++++
*/

/*  print header information                    */

print_banner(out);

time(&t);
fprintf(out,"Date/Time of run:      %s\n\n\n",ctime(&t));

fprintf(out,"Frequency is %lf GHz\nTheta = %le\ntheta_e = %le\nphi_e =
%le\nTolerance=%le\nPath step=%d\n\n",
            frequency/1e9,theta,theta_e,phi_e,tol,dx);

/*  Set up path information                    */
n_end=(37000+1700+200+100)/dx;
n_end=n_end+2000*PI/(2*dx);

n1=37000/dx; n2=n1+2000*PI/(2*dx); n3=n2+1700/dx; n4=n3+200/dx;

/*  Print path information                     */
fprintf(out,"Total points in path is %d\n",n_end);
fprintf(out,"line 1:  %d, circle 1:  %d, ellipse 1:  %d, ellipse 2:
%d\n\n",n1,n2,n3,n4);
/*fprintf(out,"Hit any key to begin calculations\n");
getch();

clrscr();                                          */
printf("Hit s to stop process\n");
```

```
fprintf(out," n       |M/P|          arg(M/P)       |M/P|dB      key\n");

for(npath=1;npath<n_end;npath++)
      {
/*  Set up Termination key as 's'
      if(kbhit())
      {
      gch=getch();
      if(gch=='s')
            {
            termstat=1;
            break;
            }
      }*/


      path(npath);      /*  Sets up X, Y, Z of CTV via path position    */

/*    For debugging information, X,Y,Z can be specified here
      X=0;Y=0;Z=-90;                                                    */

      fprintf(out," %d ",npath);


/*  Check M/P path difference.  If Eref.real = -9999, then M/P path
                        difference > 300, and Eref.imag is
                        M/P path difference                 */
      Eref=direct(theta,theta_e,phi_e,lambda);
/*printf("%lf, %lf\n",Eref.real,Eref.imag);     */

      if(Eref.real>-9990)
            {           /*  implies path diff < 300         */
            totalE[0]=zero;
            totalE[1]=zero;
            totalE[2]=zero;

            /*  n=0 is top of cylinder, n=1 is side of cyl.       */

            first=0;
            last =2;

            if(theta==90)/*  only integrate over side of cylinder */
                  first=1;

            if(theta== 0)/*  only integrate over top  of cylinder */
                  last=1;

            for(n=first;n<last;n++)
                  for(j=0;j<3;j++)
                  {/*-----------------------------------------   */
                    /*  Major computations to determine
                        scattered field at X, Y, Z of ctv   */
                        kk[n*3+j]=0;
                        integral_value=matrix_element(n,j,tol);
                        kk[n*3+j]=key;
                        E[n][j]=Multiply(integral_value,constant);

                        if(n==0)
                              E[n][j]=Multiply(E[n][j],const1);

                        totalE[j]=Add(totalE[j],E[n][j]);

                  }/*-----------------------------------------  */
```

```
/*  Now, compute the RCP component of totalE[j]                    */
            rcp_comp=RCP(totalE[0],totalE[1],totalE[2]);

/*  Compare Eref and M/P signals                                   */

            Eref=Divide(rcp_comp,Eref);

/*          print results                                           */

            fprintf(out,"   %le  %le  %le  ",Magnitude(Eref),
                  atan2(Eref.imag    ,Eref.real    )*180./PI,
                  10*log10(Magnitude(Eref)) );
            for(j=0;j<6;j++)
                  {
                  fprintf(out,"%d",kk[j]);
                  }
            fprintf(out,"\n");
            }

      if(Eref.real<-9990)
            {
            key=-99;
            fprintf(out,"     0.000       %le  %le   %d\n",
                  Eref.imag,Eref.real,key);
/*          getch();          */
            }

      }

time(&t);

if(termstat==0)
      fprintf(out,"\n\nNormal Termination\n\n");
if(termstat==1)
      fprintf(out,"\n\n   * * * User Terminated * * *\n\n\n");

fprintf(out,"Date/Time of end of run:      %s\n",ctime(&t));

fclose(out);

return 0;
}
```

```
/*
-----------------------------------------------------
      Code to parameterize the CTV path and to
      check if path difference is greater than
      300 meters...
-----------------------------------------------------
*/

#include <stdio.h>
#include <math.h>

#define PI 3.141592654

extern int dx;
extern double X,Y,Z,theta_e,phi_e;


void path(int n)
{
int nt,nl,nc,ne1,ne2,r;
double phi,dphi,t1,dtr,nux,nuy,nuz,eyax,eyay,eyaz,mag,xr,yr,zr;
double xa,ya;

dtr=PI/180.;

r=2000;
xa=37000;
ya=2000;

nl=37000/dx;
nc=nl+(r*PI)/(2*dx);
ne1=nc+1700/dx;
ne2=ne1+200/dx;

      if( n <= nl )
            {
            xa=xa-(float)n*dx;
            }
      if( ( n > nl) && (n <= nc ) )
            {
            phi=PI/2;
            dphi=(n-nl)*(float)dx/(float)r;

            phi=phi+dphi;

            if (n==nc)
                  phi=PI;

            xa=(float)r*cos(phi);
            ya=(float)r*sin(phi);
            }

      if( ( n > nc) && (n <= ne1) )
            {
            xa=-r+(n-nc)*dx;
            t1=xa+1150.;
            t1=t1*t1;
            t1=t1/(850.*850.);
            ya=sqrt(1-t1)*425.;
            }

      if( ( n > ne1) && (n <= ne2) )
```

```
                {
                xa=-300.+(n-ne1)*dx;
                t1=xa+200.;
                t1=t1*t1;
                t1=t1/(100.*100.);
                ya=sqrt(1-t1)*50.;
                }

        if( ( n > ne2) )
                {
                ya=0;
                xa=-100;
                xa=xa+(float)(n-ne2)*(float)dx;
                }


phi=phi_e*dtr;

if(theta_e>100)
        {
        X=xa*sin(phi);
        Y=xa*cos(phi);
        Z=-ya;
        }
if(theta_e<100)
        {
        X=ya*sin(phi);
        Y=ya*cos(phi);
        Z=-xa;
        }

return;
}
```

EREF.C

```
/*
-------------------------------------------------
      Computes the reference field
      real and imaginary parts...
-------------------------------------------------
*/
/*#include <stdio.h>
#include <conio.h>*/
#include <math.h>
#include "/usr/users/richiej/bin/cplxmath/cplxmath.h"

extern double X,Y,Z;

struct Complex direct(double theta, double theta_e, double phi_e,
            double lambda)
{
double xg=0,yg,zg,re=6.370949e6,rge,rg,rgc,rc,gps,omv,rd;
double dtr,a,b,c,mag,phase,xp,yp,zp,rp,dt;
struct Complex Edir;

gps=22240.*1000.;
omv=331.*1000.;
rge=re+gps;

dtr=PI/180.;

theta_e*=dtr;
phi_e*=dtr;
/*theta*=dtr;           */


  rp=re+omv;
/*xp=rp*sin(theta_e)*cos(phi_e);     */
  yp=rp*sin(theta_e)*sin(phi_e);
  zp=rp*cos(theta_e);
  if(theta_e==90) zp=0.;

c=rp*rp-rge*rge;

if((theta)!=90)
      {

      a=1./(cos(theta*dtr)*cos(theta*dtr));
      b=2.*yp*tan(theta*dtr)-2.*zp;

      dt=sqrt(b*b-4*a*c);
      zg=-b+dt;
      zg/=(2.*a);
      if(zg<0)
            {
            zg=-b-dt;
            zg/=(2.*a);
            }
      yg=-zg*tan(theta*dtr);
      }
else
      {
      zg=0;

      a=1.;
      b=-2.*yp;
/*  c is the same as above          */
```

```
        dt=sqrt(b*b-4*a*c);
        yg=-b+dt;
        yg/=(2.*a);
        if(yg>0)
              {
              yg=-b-dt;
              yg/=(2.*a);
              }
        }

rg=sqrt(xg*xg+yg*yg+zg*zg);
rgc=sqrt((xg-X)*(xg-X)+(yg-Y)*(yg-Y)+(zg-Z)*(zg-Z) );

phase=(2.*PI)/lambda;
phase*=(rg-rgc);
mag=rg/rgc;

Edir.real=mag*cos(phase);
Edir.imag=mag*sin(phase);

rc=sqrt(X*X+Y*Y+Z*Z);
rd=rg+rc-rgc;

/*printf("\n(rg=%lf)+(rc=%lf)-(rgc=%lf)=(rd=%lf)\n",rg,rc,rgc,rd);*/

if(rd>300)
      {
      Edir.real=-9999.;
      Edir.imag=rd;
      }

/*printf("zg=%le, yg=%le\n",zg,yg);
getch();                                          */

return Edir;

}
```

```
/*
--------------------------------
      integrator
--------------------------------
*/

/*#include <stdio.h>
#include <conio.h>            */
#include <math.h>
#include "/usr/users/richiej/bin/cplxmath/cplxmath.h"

/*  N is #dimensions          */
#define N 2

extern int key;
extern double delta,gama,L,radius,lambda;

struct Complex fcnA(int n, int j, double x1, double x2);

struct Complex matrix_element(int n, int jj, double tol)
{
int i,k[16],kt,m,mm,l,ll,nn[10];
int nn9,i9,nn8,i8,nn7,i7,nn6,i6,nn5,i5,nn4,i4,nn3,i3,nn2,i2,nn1,i1;
double a[N+1],b[N+1],h[N+1],al,e,p[10],d[10],cc[10],en;
double ee,g[10],x[10];

struct Complex zero,v[15],aa[15],u,aint,t1,t2;

zero.real=0;
zero.imag=0;

/*  Define upper and lower limits of integration      */
if(n==0)
      {
      a[1]=0.;
      b[1]=radius;
      a[2]=0;
      b[2]=2.*PI;
      }
if(n==1)
      {
      a[1]=-L/2.;
      b[1]= L/2.;
      a[2]=-PI;
      b[2]= 0;
      }

k[1]=1;
k[2]=2;

key=0;
m=11;
al=1.5;

if( (N<1)||(N>9)||(m<1)||(al<1.5)||(al>2) )
      {
/*    printf("bomb at 1\n");
      getch();                        */
      key=-9;
      return zero;
      }
```

```
h[1]=0.02;
h[2]=0.02;

for(i=1;i<=N;i++)
      {
      if( (h[i]<=0) || (h[i]>=1) )
            {
/*          printf("bomb at 2\n");
            getch();                              */
            key=-9;
            return zero;
            }
      d[i]=a[i];
      }
ee=tol;
mm=m;
for(i=N;i<=8;i++)
      {
      p[i+1]=0;
      nn[i+1]=1;
      d[i+1]=0;
      }
l=1;
for(i=1;i<=N;i++)
      cc[i]=b[i]-d[i];


repeat:
;
u=zero;
/*kt=0;                                       */

for(i=1;i<=N;i++)
      {
      g[i]=h[i]/k[l];
      nn[i]=1./g[i]+0.5;
      p[i]=cc[i]*g[i];
      }
nn9=nn[9];

for(i9=1;i9<=nn9;i9++)
      {
      x[9]=d[9]+p[9]*(i9-0.5);
      nn8=nn[8];
      for(i8=1;i8<=nn8;i8++)
            {
            x[8]=d[8]+p[8]*(i8-0.5);
            nn7=nn[7];
            for(i7=1;i7<=nn7;i7++)
                  {
                  x[7]=d[7]+p[7]*(i7-0.5);
                  nn6=nn[6];
                  for(i6=1;i6<=nn6;i6++)
                        {
                        x[6]=d[6]+p[6]*(i6-0.5);
                        nn5=nn[5];
                        for(i5=1;i5<=nn5;i5++)
                              {
                              x[5]=d[5]+p[5]*(i5-0.5);
                              nn4=nn[4];
                              for(i4=1;i4<=nn4;i4++)
                                    {
                                    x[4]=d[4]+p[4]*(i4-0.5);
                                    nn3=nn[3];
```

```
                    for(i3=1;i3<=nn3;i3++)
                          {
                          x[3]=d[3]+p[3]*(i3-0.5);
                          nn2=nn[2];

                /*        printf("nn2=%d, ",nn2);        */

                          for(i2=1;i2<=nn2;i2++)
                                {
                                x[2]=d[2]+p[2]*(i2-0.5);
                                nn1=nn[1];

                /*              if(i2==1) printf("nn1=%d\n",nn1);    */

                                for(i1=1;i1<=nn1;i1++)
                                      {
                                      x[1]=d[1]+p[1]*(i1-0.5);


            u=Add(u,fcnA(n,jj,x[1],x[2]));
}}}}}}}}}
/*printf(".");*/
for(i=1;i<=N;i++)
/*    u=u*p[i];   */
      {
      u.real*=p[i];
      u.imag*=p[i];
      }
v[l]=u;
/*printf("u=%le\n",u);              */
if( (l-1)<=0)
      {
      aa[1]=v[1];
      l++;
      goto repeat;
      }
if( (l-1)>0)
      {
      en=k[l];
      for(ll=2;ll<=l;ll++)
            {
            i=l+1-ll;
/*          v[i]=v[i+1]+(v[i+1]-v[i])/((en/k[i])*(en/k[i])-1.);    */
            t1.real=( (en*en)/(k[i]*k[i]) -1.);
            t1.imag=0.;
            t2=Subtract(v[i+1],v[i]);
            t1=Divide(t2,t1);
            v[i]=Add(v[i+1],t1);
            }
      }
aint=v[1];
key=1;
/*if(fabs(aint-aa[l-1])<fabs(aint*ee) )                              */
t1.real=aint.real*ee;
t1.imag=aint.imag*ee;

if(Magnitude(Subtract(aint,aa[l-1]))<(Magnitude(t1)))
      {
/*    printf("at 3, answer is %le\n",aint);
      getch();                                  */
      return aint;
      }
key=-1;
if(l==mm)
```

```
        {
/*      printf("at 4, answer is %le\n",aint);
        getch();                                */
        return aint;
        }
aa[l]=aint;
l=l+1;
k[l]=al*k[l-1];
goto repeat;

}
```

FCN_A.C

```
/*
*************************
This defines 2-D function
     to be integrated
*************************
*/

#include <math.h>
#include "/usr/users/richiej/bin/cplxmath/cplxmath.h"
/*#include <stdio.h>
#include <conio.h>      */

/*  Prototype functions       */

extern double delta,gama,L,radius,lambda;
extern double X,Y,Z;

struct Complex fcnA(int n, int j, double xr, double yp)
{
double k,t_m,r_m,r_p,x_m,y_m,x_p,y_p,f2,mag,beta,thickness,dtr;
double x,y,z,greend,R,R2,R3,R4;
struct Complex fcn_value,green,jcur,ejx,ejy,ejz,xi1,xi2;
struct Complex t1,t2,t3,zero;

zero.real=0; zero.imag=0;

k=2*PI/lambda;

/*  NOTE:  The metric for r(dphi) is in jcur!
                  (xr if n=0, and radius if n=1  */

if(n==0)/*  integral over top of cylinder
                        in this case, xr is radius, and yp
                        is the angle in radians                    */
      {
      x=xr*cos(yp);
      y=xr*sin(yp);
      z=L/2.;

      ejx.real=0;
      ejx.imag=-delta;
      ejy.real=1.;
      ejy.imag=0;
      ejz=zero;
      jcur.real= cos(k*gama*y)*xr;
      jcur.imag=-sin(k*gama*y)*xr;
      }

if(n==1)    /*  integral over side of cylinder,
                        in this case, xr is z, and yp is
                        phi angle in radians               */
      {
      x=radius*cos(yp);
      y=radius*sin(yp);
      z=xr;

      ejx.real=0;
      ejx.imag=gama*y/radius;
      ejy.real=0;
      ejy.imag=-gama*x/radius;
      ejz.real=-y/radius;
      ejz.imag=delta*x/radius;
```

```
        jcur.real= cos(k*(gama*y-delta*z))*radius;
        jcur.imag=-sin(k*(gama*y-delta*z))*radius;

        }


greend=sqrt((X-x)*(X-x)+(Y-y)*(Y-y)+(Z-z)*(Z-z));
green.real= cos(k*greend)/greend;
green.imag=-sin(k*greend)/greend;

R=greend; R2=R*R;R3=R2*R;R4=R3*R;

xi1.real=k*k/R2-3./R4;
xi1.imag=-3.*k/R3;

xi2.real=2./R2;
xi2.imag=2*k/R;

if(j==0)     /*  x component          */
        {

        t1.real=((Y-y)*(Y-y)+(Z-z)*(Z-z));
        t1.imag=0.;
        t1=Multiply(t1,xi1);

        t2=xi2;
/*        t1=Add(t1,t2)                */
        t1.real+=t2.real;
        t1.imag+=t2.imag;

        t1=Multiply(t1,ejx);

        t2.real=(X-x)*(Y-y);
        t2.imag=0;
        t2=Multiply(t2,ejy);
        t2=Multiply(t2,xi1);

        t3.real=(X-x)*(Z-z);
        t3.imag=0;
        t3=Multiply(t3,xi1);

        t3=Multiply(t3,ejz);

/*      t1=Subtract(t1,t2);
        t1=Subtract(t1,t3);          */

        t1.real-=(t2.real+t3.real);
        t1.imag-=(t2.imag+t3.imag);
        }

if(j==1)     /*  y component          */
        {
        t1.real=(X-x)*(Y-y);
        t1.imag=0;
        t1=Multiply(t1,xi1);
        t1=Multiply(t1,ejx);

        t2.real=(X-x)*(X-x)+(Z-z)*(Z-z);
        t2.imag=0;
        t2=Multiply(t2,xi1);
/*      t2=Add(t2,xi2);              */
        t2.real+=xi2.real;
        t2.imag+=xi2.imag;
```

```
        t2=Multiply(t2,ejy);

        t3.real=(Y-y)*(Z-z);
        t3.imag=0;
        t3=Multiply(t3,xi1);
        t3=Multiply(t3,ejz);

/*      t1=Subtract(t2,t1);
        t1=Subtract(t1,t3);                 */
        t1.real=t2.real-t1.real;
        t1.imag=t2.imag-t1.imag;
        t1.real=t1.real-t3.real;
        t1.imag=t1.imag-t3.imag;

        }
if(j==2)    /*  z component               */
        {
        t1.real=(X-x)*(Z-z);
        t1.imag=0;
        t1=Multiply(t1,xi1);
        t1=Multiply(t1,ejx);

        t2.real=(Y-y)*(Z-z);
        t2.imag=0;
        t2=Multiply(t2,xi1);
        t2=Multiply(t2,ejy);

        t3.real=(X-x)*(X-x)+(Y-y)*(Y-y);
        t3.imag=0;
        t3=Multiply(t3,xi1);
/*      t3=Add(t3,xi2);                     */
        t3.real+=xi2.real;
        t3.imag+=xi2.imag;

        t3=Multiply(t3,ejz);

/*      t1=Add(t1,t2);                      */
        t1.real+=t2.real;
        t1.imag+=t2.imag;

/*      t1=Subtract(t3,t1);                 */
        t1.real=t3.real-t1.real;
        t1.imag=t3.imag-t1.imag;

        }

t1=Multiply(t1,green);

fcn_value=Multiply(t1,jcur);

return fcn_value;
}
```

RCP.C

```
/*
-------------------------------------------------------------
      Function that computes RCP component of E_x, E_y, E_z if
      given x,y,z of observation point (global variables)
-------------------------------------------------------------
*/

/*#include <stdio.h>
#include <conio.h>            */
#include <math.h>
#include "/usr/users/richiej/bin/cplxmath/cplxmath.h"


/*extern double gamma,delta,L,radius,lambda;          */

extern double X,Y,Z;

struct Complex RCP(struct Complex ex,struct Complex ey, struct Complex ez)
{
double theta,phi,r,gamma2,alpha2,xx,yy;
struct Complex er,et,ep,sphi,cphi,stht,ctht,e1,e2,e3,zero;
struct Complex rcpterm;
zero.real=0;
zero.imag=0;

sphi.imag=0;
cphi.imag=0;
stht.imag=0;
ctht.imag=0;

r=sqrt(X*X+Y*Y+Z*Z);
theta=acos(Z/r);
if((Y==0)&&(X==0))
      phi=0;
else
      phi=atan2(Y,X);

sphi.real=sin(phi);
cphi.real=cos(phi);
stht.real=sin(theta);
ctht.real=cos(theta);

/*  Computation of E_r              */

e1=Multiply(ex,stht);
e1=Multiply(e1,cphi);

e2=Multiply(ey,stht);
e2=Multiply(e2,sphi);

e3=Multiply(ez,ctht);

er=Add(e1,e2);
er=Add(er,e3);

/*  Computation of E_theta          */

e1=Multiply(ex,ctht);
e1=Multiply(e1,cphi);

e2=Multiply(ey,ctht);
e2=Multiply(e2,sphi);
```

```
e3=Multiply(ez,stht);

et=Add(e1,e2);
et=Subtract(et,e3);

/*  Computation of E_phi            */

e1=Multiply(ex,sphi);

e2=Multiply(ey,cphi);

e3=zero;

ep=Subtract(e2,e1);

/*printf("\nR  %le  %le  %le\n",er.real, et.real, ep.real);
printf(  "I  %le  %le  %le\n",er.imag, et.imag, ep.imag);          */

/*  Now, compute RCP term of given field  */


yy=-ep.real+et.imag;
xx= ep.imag+et.real;

gamma2=atan2(yy,xx);

if( (fabs(xx) )<0.1)
      alpha2=(+ep.imag+et.real)/(2.*cos(gamma2));
if( (fabs(xx) )>0.1)
      alpha2=(-ep.real+et.imag)/(2.*sin(gamma2));

rcpterm.real=alpha2*cos(gamma2);
rcpterm.imag=alpha2*sin(gamma2);

return rcpterm;
}
```

```
/*
-------------------------------------
  Prints program banner
  to output file
-------------------------------------
*/

#include <stdio.h>

void print_banner(FILE *fp)
{
fprintf(fp,"\n\n     ************************************");
fprintf(fp,"\n\n       PO Solution to Auto-Rendezvous and  ");
fprintf(fp,"\n\n             capture of OMV by SSF         ");
fprintf(fp,
       "\n\n\n                        by");
fprintf(fp,"\n\n          J. Richie and F. Forest");
fprintf(fp,"\n\n     ************************************");
fprintf(fp,"\n\n\n");
}
```

CPLXMATH.H

```
#ifndef _CPLXMATH_H_
#define _CPLXMATH_H_

/*************************************************************************/
/*                                                                       */
/*      Jeff Swart            October, 1992              Turbo C++ 3.0     */
/*                                                                       */
/*  #defines, structures, and function prototypes for Complex math.      */
/*                                                                       */
/*************************************************************************/

/*************************************************************************/
/*  #Defines                                                             */
/*************************************************************************/

/* Pi                                                                    */
#define PI 3.1415927



/*************************************************************************/
/*  Structures                                                           */
/*************************************************************************/

/* Complex number in Cartesian form                                      */
struct Complex
 {
  double real,                  /* Real part of Complex number           */
       imag;                   /* Imaginary part of Complex number      */
  };


/*************************************************************************/
/*  Function prototypes                                                  */
/*************************************************************************/

struct Complex Divide(struct Complex num_cart,
                      struct Complex denom_cart);

      /*  Added by J. Richie, January, 1993  */

struct Complex Multiply(struct Complex num1,
                        struct Complex num2);

struct Complex Add(struct Complex add1,
                   struct Complex add2);

struct Complex Subtract(struct Complex subp,
                        struct Complex subm);

double Magnitude(struct Complex z);

struct Complex Conjugate(struct Complex z);


#endif
```

CPLXMATH.C

```
/*#include <stdio.h>
#include <conio.h>*/
#include <math.h>
#include "cplxmath.h"

/*************************************************************************/
/*                                                                       */
/*      Jeff Swart            October, 1992             Turbo C++ 3.0    */
/*                                                                       */
/*  Basic Complex math functions.  Prototypes are located in CPLXMATH.H. */
/*                                                                       */
/*************************************************************************/

/*************************************************************************/
/*  'RadToDeg' will convert a value given in radians as a double to the  */
/*  equivalent value in degrees, and return the result as a double.      */
/*************************************************************************/


/*************************************************************************/
/*  'DivComplexCart' will divide two Complex numbers given in Cartesian  */
/*  RENAMED TO Divide:  J. richie, 2/93                                  */
/*  form as structures of type 'cart_Complex', and return the result in  */
/*  Cartesian form as a structure of type 'cart_Complex'.                */
/*************************************************************************/

struct Complex Divide(struct Complex num,
                      struct Complex denom)
 {
  /***********************************************************************/
  /*  Variable declarations                                              */
  /***********************************************************************/

  struct Complex                      /* Polar form of numerator        */
                     /* Polar form of denominator       */
                result      ;    /* Polar form of result           */


  double mag_1,mag_2,angle_1,angle_2;

  /***********************************************************************/
  /*  Code                                                               */
  /***********************************************************************/

  /* Convert numerator and denominator to polar form                     */

  mag_1=Magnitude(num);
  mag_2=Magnitude(denom);
  angle_1=atan2(num.imag,num.real);
  angle_2=atan2(denom.imag,denom.real);

  /* Calculate polar form of result                                      */
  mag_1=mag_1/mag_2;
  angle_1=angle_1-angle_2;
  result.real=mag_1*cos(angle_1);
  result.imag=mag_1*sin(angle_1);


  /* Return Cartesian form of result                                     */
```

```
  return result;

  }  /* End function 'Divide'                                          */

/*  ------------------------------------------------------
      ADDED BY J. RICHIE, JANUARY 1993
    ------------------------------------------------------

  Function to obtain product of two Complex numbers                    */

struct Complex Multiply(struct Complex num1,
                        struct Complex num2)
  {
  /*  Variable declarations                                 */

  struct Complex result;

  /*  Code                                                  */
  result.real=num1.real*num2.real-num1.imag*num2.imag;
  result.imag=num1.real*num2.imag+num1.imag*num2.real;

  return result;

  }

/*  Function to obtain sum of two Complex numbers                    */

struct Complex Add(struct Complex add1,
                   struct Complex add2)
  {
  /*  Variable declarations                                 */

  struct Complex result;

  /*  Code                                                  */
  result.real=add1.real+add2.real;
  result.imag=add1.imag+add2.imag;

  return result;

  }

/*  Function to obtain difference of two Complex numbers             */

struct Complex Subtract(struct Complex subp,
                        struct Complex subm)
  {
  /*  Variable declarations                                 */

  struct Complex result;

  /*  Code                                                  */
  result.real=subp.real-subm.real;
  result.imag=subp.imag-subm.imag;

  return result;

  }

/*  Function to return the magnitude of a Complex number             */

double Magnitude(struct Complex z)
```

```
{
double mag_value;

mag_value=sqrt( (z.real*z.real) + (z.imag*z.imag) );
return mag_value;
}

/*  Function to return the Complex conjugate of a Complex number  */

struct Complex Conjugate(struct Complex z)
{
struct Complex conjug;

conjug.real=z.real;
conjug.imag=(-1)*z.imag;
return conjug;
}
```

```
/*
-------------------------------------------
      Program to convert tag files
      to n column format
-------------------------------------------
*/

#include <stdio.h>
#include <conio.h>
#include <string.h>

int main()
{
FILE *in,*out;

char file_in[13],file_out[13],buffer[80],test[5],end_str[5];
int count1,n,count1max,count2max,i,j;
long int key;
double x1,x2,x3;

clrscr();

/* ---------------------------------------
      Get file names
*/

printf("enter  input file name:  ");
scanf("%s",file_in);

printf("enter output file name:  ");
scanf("%s",file_out);

/* ---------------------------------------
      Open input file
*/

if ((in = fopen(file_in, "rt"))
    == NULL)
{
   fprintf(stderr, "Cannot open input file.\n");
   return 1;
}

/* ---------------------------------------
      Open output file
*/

if ((out = fopen(file_out, "wt"))
    == NULL)
{
   fprintf(stderr, "Cannot open output file.\n");
   return 1;
}

/*printf("files successfully opened\n");  */

/* ---------------------------------------
      Set up test strings
*/

strcpy(test," n ");
strcpy(end_str,"Dat");
```

```
/* ---------------------------------------
      Read input file until
      first sighting of " n "
*/

do
      {
      fgets(buffer,85,in);
/*    printf("%s",buffer);
      getch();                  */
      }while((strncmp(buffer,test,3)));
fprintf(out,"%s",buffer);
/* ---------------------------------------
      Initialize counters to count
      amount of data read
*/

      count1=0;

/* ---------------------------------------
      Read in data and write to output
      file
*/

while(!feof(in))
      {
      fgets(buffer,85,in);

      if(!strncmp(buffer,end_str,3))      /*  if "Dat" is seen, end reading*/
            break;

      if(strncmp(buffer,test,3) )   /*  test for data or next set   */
            {
            sscanf(buffer," %d %lg %lg %lg %ld",
                  &n,&x1,&x2,&x3,&key);

/*          getch();          */
            if(key!=-99)
                  {
                  fprintf(out,"%d  %le  %le  %le  %ld\n",
                        n,x1,x2,x3,key);
                  count1++;
/*                if(!(count1%10))

                  getch();          */
                  }
            }
/*    getch();                        */
      }
      count1max=count1;

printf("finished reading data resulting in %d points\n",
            count1max);

/* ---------------------------------------
      Close all files
*/

fclose(in);
fclose(out);

/* ---------------------------------------
      Final Clean up
*/
```

```
printf("Data management finished, hit any key to continue");
getch();

return 0;

}
```

7.2 MoM Code and Report

The following is a technical report from the Marquette University Electromagnetic Simulations Laboratory that discusses the theory and code used in the two-dimensional moment method analysis.

Two-Dimensional
Method of Moments
TM^z Case

Technical Report 7

Dr. Richie
March 24, 1993

1. Introduction

This report is a discussion of the method of moments for TM^z two dimensional problems. The code given is written for the solution of a metallic cylinder, but can be modified for any cross section. The second section provides a brief summary of the solution techniques, and the third section lists the output files. The appendix is a listing of the code.

2. Solution Technique

The integral equation to be solved is a two-dimensional EFIE, as given in [1]:

$$\frac{\beta\eta}{4}\int_C J_z(\varrho')H_0^{(2)}(\beta\,|\,\varrho_m-\varrho'\,|)\,dc'=E_z^i(\varrho_m) \tag{1}$$

where ϱ_m is the observation point (on the surface of the object), and ϱ' is the integration variable over the contour C of the object. To avoid the singularity in $H_0^{(2)}$, a thickness of 10^{-6} has been incorporated. See figure 1. The solution is found by writing J as:

$$J_z(\varrho')=\sum_{n=0}^{N-1}\alpha_n g_n(\varrho') \tag{2}$$

where

$$g_n(\varrho')=1 \quad \theta_n\le\theta'\le\theta_{n+1} \tag{3}$$

and is zero otherwise, where

$$\theta_n{}^\circ=\frac{360\times n}{N} \tag{4}$$

To solve, we substitute J_z into eqn.(1):

$$\frac{\beta\eta}{4}\sum_{n}\alpha_n\int_{\theta'}g_n[\varrho'(\theta')]H_0^{(2)}(\beta\,|\,\varrho_m(\theta_m)-\varrho'(\theta')\,|\,)\varrho'(\theta')d\theta'=E_z^i[\varrho_m(\theta_m)] \tag{5}$$

where the explicit dependence on angle has been indicated , and

$$\theta_m{}^\circ=\frac{(n+0.5)\times 360^\circ}{N} \tag{6}$$

the factor of 0.5 places the observation points at the center of the basis function segments.

Note that we are solving a matrix equation where each value of m gives us an equation of the form in eqn.(5) We have N unknowns in the sum for each equation. In other words, we have:

$$Ax=b \tag{7}$$

where:

$$A_{mn}=\int_{\theta_n}^{\theta_{n+1}}H_0^{(2)}(\beta\,|\,\varrho_m-\varrho'\,|\,)\varrho'd\theta' \tag{8}$$

$$x_n=\alpha_n \tag{9}$$

and

$$b_m=E_z^i(\varrho_m)=e^{j\beta x}\qquad x=\varrho'\cos(\theta_m) \tag{10}$$

for a plane wave incident on the x axis, as shown in figure 1. The value of x in eqn.(9) is dependent on the value of ϱ_m, and the shape of the object. The factor, $\beta\eta/4$ is ignored in the computations.

The code first calculates the entries of **A**, **x**, and **b**, and then "inverts" the matrix by replacing **A** with the LU decomposition of **A**. Then, the solution is found using forward and backward substitution. The solution is then checked by evaluating both sides of eqn.(6) and computing the mean squared error.

Once the soluion has been checked, the far field scattered radiation is calculated using [1]:

$$H_0^{(2)}(\beta|\varrho_m-\varrho'|) \approx \sqrt{\frac{2j}{\pi\beta\varrho_m}}\, e^{-j\beta\varrho_m} e^{j\beta\varrho'\cos(\phi_m-\phi')} \tag{11}$$

Hence, the scattered field is given by the integration:

$$E_z^s(\phi_m) = K\int_C J_z(\varrho') e^{j\beta\varrho'\cos(\phi_m-\phi')}\, dc' \tag{12}$$

Substituting the approximate solution for the induced surface current into eqn.(11), we have:

$$E_z^s(\phi_m) = \tilde{K}\sum_n \alpha_n \int_{\theta_n}^{\theta_{n+1}} e^{j\beta\varrho'(\theta')\cos(\theta_m-\theta')\varrho'(\theta')}\, d\theta' \tag{13}$$

The value of K is ignored since the far field pattern is normalized by the program.

3. Output Files

DATA.OUT: Summary of all computations, including the date/time each task is completed.
MAT.2D: Listing of all matrix elements
EX.2D: Listing of the b vector
L_U.2D: Listing of the matrix in L-U decomp form
CUR.2D Listing of the x vector (solution)

A program, rp.c, does exist that will read in the data.out file and write a file that is only the far field data. This can then be used in Tell-A-Graf for visual graphics display.

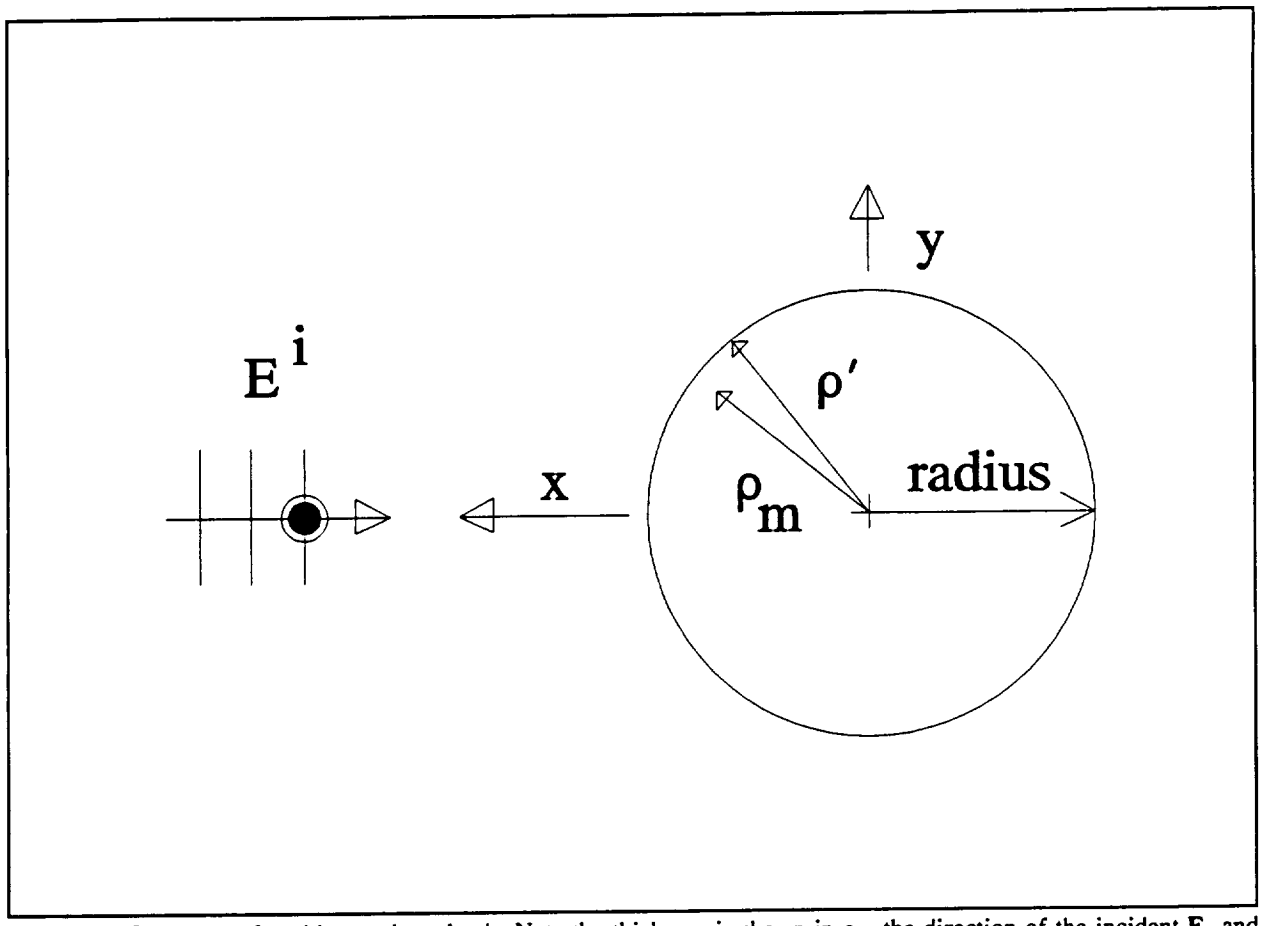


Figure 1. Geometry of problem to be solved. Note the thickness is shown in ρ_m, the direction of the incident **E**, and the radius is indicated (entered into the program in wavelengths).

The project is named 2D_TMZ.PRJ. This is the file that should be loaded into the Borland IDE (the editor). The files in the project are listed below:

TMZ_MN.C	Main program file.
INTEGRAT.C	Does Simpson's Rule integration
FCN.C	Function that is integrated
R.C	Radius function (here is circle)
B_T_FCN.C	Holds the basis and testing (not used) functions
H_0.C	Computes the values of J_0 and Y_0 using a polynomial approximation (see Abromowitz and Stegan)
EXCIT.C	Computes the b vector
L_U.C	Finds LU decomp and has solver and solution check
PATTERN.C	Sets up Far Field Integration and Summation
CPLXMATH.H	Header file for the complex mathematics stuff...
CPLXMATH.C	Holds complex math functions
HED_FT.C	Prints banner to output file

APPENDIX I

CODE LISTING: 2d_tmz.prj

TMZ_NM.C

```
/*
************************************************
        This is main for the 2-D TM^z program
************************************************
*/

#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#include <conio.h>
#include <math.h>
#include <time.h>
#include "cplxmath.h"


/*  prototype functions                                          */
struct Complex matrix_element(int n, int m,int flag);
void excitation(void);
void L_U_Decomposition(void);
void Solve(void);
void Solution_Check(void);
void print_banner(FILE *fp);
double pattern(int phi,int flag);

/* Declare external variables                                    */
int N;
double radius;
double huge *s;
struct Complex huge *a, *b, *x; /*  These are for the matrix and vectors
                                    where Ax=b                   */

FILE *data_out;
time_t t;

main()/*  ---BEGINNING OF MAIN----------------------------------*/
{
int n,m,n1,m1,integral_flag,phi;
```

```
double beta,eta,epsilon,mu,factor;
double ff_max;
struct Complex det;
FILE *fp_out;

clrscr();

printf("Enter the radius parameter:  ");
scanf("%lf",&radius);

/*  First, set up a do loop to get the value of N and insure
        that enough memory is available, and allocate the
        memory                                                  */

        printf("Enter the size of the matrix:  ");
        scanf("%d",&N);

        if((a=farcalloc( (long) (N*N),sizeof (struct Complex)))==NULL)
                {
                printf("Out of memory in pointer initialization\n");
                exit(1);
                }
        if((b=malloc((long)N*sizeof(struct Complex)))==NULL)
                {
                printf("Out of memory in pointer b\n");
                exit(1);
                }
        if((x=malloc((long)N*2*sizeof(struct Complex)))==NULL)
                {
                printf("Out of memory in pointer x\n");
                exit(1);
                }
        printf("\n");

/*  Set up output file for miscellaneous data stuff             */
if((data_out=fopen("data.out","wt"))==NULL) {
        puts("cannot initialize data_out file\n");
        exit(1);
        }
print_banner(data_out);

fprintf(data_out,"\nN is %d \nradius is %f\n",N,radius);
time(&t);
fprintf(data_out,"\nDate/Time run:          %s\n",ctime(&t));
```

```
fclose(data_out);

/*  Fill the matrix A using integrate function                    */
beta=3.141592654*2;
epsilon=8.854e-12;
mu=4*3.141592654*1e-7;
eta=sqrt(mu/epsilon);
factor=beta*eta/4.;
/*printf("factor is %f\n",factor);     factor is beta*eta/4    */

integral_flag=0;  /*  for filling the matrix                    */

clrscr();
gotoxy(1,1);
printf("Filling %dX%d matrix for MoM",N,N);

for(n=0;n<N;n++)
        {
        gotoxy(5,5);
        printf("integrating at row %d  ",n+1);
        for(m=0;m<N;m++)
                {
                *(a+(N*n+m))=matrix_element(n,m,integral_flag);
/*              (a+(N*n+m))->real*=factor;
                (a+(N*n+m))->imag*=factor;
        getche();*/
                }
        }
gotoxy(10,10);
printf("L-U decomposition on matrix...\n\n\n");

/*  write matrix to file mat.2d                                   */

if((fp_out=fopen("mat.2d","wt"))==NULL) {
        puts("cannot initialize file\n");
        exit(1);
        }

if((data_out=fopen("data.out","at"))==NULL) {
        puts("cannot re-open data_out file\n");
        exit(1);
        }
time(&t);
fprintf(data_out,"\nDate/Time matrix finished:  %s\n",ctime(&t));
```

```
for(n=0;n<N;n++)
        for(m=0;m<N;m++)
                {
                fprintf(fp_out,"%d %d %f %f\n",n,m,
                        ((a+N*n+m)->real),((a+N*n+m)->imag));
                }
fclose(fp_out);
fclose(data_out);

/* Create the source vector (or excitation) b                    */
excitation();

/* write source vector to file ex.2d                             */
if((fp_out=fopen("ex.2d","wt"))==NULL) {
        puts("cannot initialize file\n");
        exit(1);
        }
for(n=0;n<N;n++)
        {
        fprintf(fp_out,"%d %f %f\n",n,((b+n)->real),((b+n)->imag));
        }
fclose(fp_out);

/* Now, solve the matrix, First, LU decomposition                */
L_U_Decomposition();

/* Write L-U decompostion to disk                                */
if((fp_out=fopen("l_u.2d","wt"))==NULL) {
        puts("cannot initialize file\n");
        exit(1);
        }

det.real=1;
det.imag=0;

for(n=0;n<N;n++)
        {
        for(m=0;m<N;m++)
                {
                fprintf(fp_out,"%d %d %f %f\n",n,m,
                        ((a+N*n+m)->real),((a+N*n+m)->imag));
                }
        det=Multiply(det,*(a+N*n+n));
```

```
        }
if((data_out=fopen("data.out","at"))==NULL)
        {
        puts("cannot open for D data_out file\n");
        exit(1);
        }
time(&t);
fprintf(data_out,"\n\nDate/Time L-U Decomp done:  %s\n\n",ctime(&t));
fprintf(data_out,"Magnitude of Determinant of system is %e\n\n",
                                                        Magnitude(det));


fclose(fp_out);

/*  Do forward and backward substitutions to get x in Ax=b      */
Solve();

/*  Write solution to cur.2d                                    */
if((fp_out=fopen("cur.2d","wt"))==NULL) {
        puts("cannot initialize file\n");
        exit(1);
        }
fprintf(data_out,"The final current vector is:\n");
for(n=0;n<N;n++)
        {
        fprintf(fp_out,"%d %f %f\n",n,((x+n)->real),((x+n)->imag));
        fprintf(data_out,"%d:  %f+j(%f)\n",n+1,((x+n)->real),((x+n)->imag));
        }
fprintf(data_out,"\n\nChecking the matrix solution:");
fclose(fp_out);
fclose(data_out);

/*  Before checking solution, read in old A matrix              */
if((fp_out=fopen("mat.2d","rt"))==NULL) {
        puts("cannot open file\n");
        exit(1);
        }

for(n=0;n<N;n++)
        for(m=0;m<N;m++)
                {
                fscanf(fp_out,"%d %d %lf %lf",&n1,&m1,
                        &((a+N*n+m)->real),&((a+N*n+m)->imag));
                }
fclose(fp_out);
```

```
Solution_Check();

farfree(a);
if((s=farcalloc( (long) (370),sizeof (double)))==NULL)
        {
        printf("Out of memory in pointer initialization\n");
        exit(1);
        }

integral_flag=1; /*  1 is for far field pattern                    */
ff_max=0;

gotoxy(1,17);
printf("Computing the Far field scattering...");
gotoxy(5,20);

for(n=0;n<361;n++)
        {
        if(!(n%36))
                {
                printf(".");
                }
        phi=n*10;
        *(s+n)=pattern(phi,integral_flag);
        if( (*(s+n))>ff_max )
                ff_max=*(s+n);
        }

if((data_out=fopen("data.out","at"))==NULL) {
        puts("cannot re-open data_out file\n");
        exit(1);
        }
time(&t);
fprintf(data_out,"\nDate/Time FF data finished:  %s\n",ctime(&t));
fprintf(data_out,"Angle\t\t\tNormalized Pattern\n\n");

printf("\nnormalization is %lf\n",ff_max);
getch();

for(n=0;n<361;n++)
        {
        phi=n;
        *(s+n)=*(s+n)/ff_max;
```

```
        fprintf(data_out,"\t%d \t\t%e\n",phi, *(s+n) );
        }

time(&t);
fprintf(data_out,"\n\n\nRun Completion Time: %s\n", ctime(&t));


fclose(data_out);

farfree(s);
free(x);
free(b);

return(0);
}
```

INTEGRAT.C

```
/*
*****************************
  Does integration for
  matrix elements...
*****************************
*/

#include <stdio.h>
#include <conio.h>
#include <math.h>
#include "cplxmath.h"

struct Complex fcn(int n, int m, double x,int flg);

extern int N;
extern double radius;


struct Complex matrix_element(int n, int m, int flag)
{
int ni,k;
struct Complex integral_value,error,sum,y1,y2,t1,t2,t3,t4;
double x,delta_x,er,delta_theta,a,b,h;

delta_theta=360./(double)N;
```

```
a=n*delta_theta;
b=(n+1)*delta_theta;
integral_value.real=-9e-9;
integral_value.imag=0;

ni=40;

do
        {
        ni+=20;
        sum.real=0;
        sum.imag=0;
        delta_x=(b-a)/ni;

        h=delta_x*3.141592654/180.;

        t1=fcn(n,m,a,flag);
        t2=fcn(n,m,b,flag);
        sum=Add(t1,t2);
        for(k=1;k<ni-1;k=k+2)
                {
                x=a+delta_x*k;
                y1=fcn(n,m,x,flag);
                y1.real=4*y1.real;
                y1.imag=4*y1.imag;

                y2=fcn(n,m,x+delta_x,flag);
                y2.real=2*y2.real;
                y2.imag=2*y2.imag;

                sum=Add(sum,y1);
                sum=Add(sum,y2);
                }
        y1=fcn(n,m,x+2*delta_x,flag);
        y1.real=4*y1.real;
        y1.imag=4*y1.imag;

        sum=Add(sum,y1);
        sum.real=sum.real*h/3.;
        sum.imag=sum.imag*h/3.;

        error.real=fabs(sum.real-integral_value.real);
        error.imag=fabs(sum.imag-integral_value.imag);
```

```
        er = Magnitude(error);

        integral_value.real = sum.real;
        integral_value.imag = sum.imag;
/*printf("%d is n, and value of integral is %f\n",n,integral_value);
getch();
if(!(ni%20))
        {
printf("%d is n, and value of integral is %f = (%f)\n",ni,
                                integral_value.real,integral_value.imag);
getch();                                                                */
        }

/*if(!(ni%1000))
        printf(".");*/

}while (er > 1e-6);

integral_value.real = sum.real;
integral_value.imag = sum.imag;

/*printf("final n at %d  \n",ni);*/
return integral_value;
}
```

FCN.C

```
/*
**************************
This defines function
        to be integrated
**************************

integral_flag (in main), is flag here.

flag =  0 -> matrix fill
flag =  1 -> far field pattern
flag = -1 -> error computation

*/

#include <math.h>
#include "cplxmath.h"
/*#include <stdio.h>                */
```

```
/*  Prototype functions                 */
double j_0(double x    );
double y_0(double x    );

double basis(double t);
double testf(double r      );

double  r  (double theta,double ka);

extern int N;
extern double radius;

struct Complex fcn(int n, int m, double t,int flg)
{
double k,t_m,r_m,r_p,x_m,y_m,x_p,y_p,f1,f2,mag,beta,thickness,pi,dtr;


struct Complex fcn_value,h0;

pi=3.141592654;
k=2*pi;
dtr=pi/180.;
beta=k;
thickness=1e-6;

switch(flg)
        {
        case 0:             /*  matrix fill                                 */
                {

                t_m=(double)(m+0.5)*360/(double)N;
                t_m=t_m*dtr;

                t=t*dtr;

                r_m=r(t_m,k)-thickness;
                x_m=r_m*cos(t_m);
                y_m=r_m*sin(t_m);
                r_p=r(t,k);
                x_p=r_p*cos(t);
                y_p=r_p*sin(t);

                f1=basis(t)*r_p;
```

```
        mag = beta*sqrt( ((x_m-x_p)*(x_m-x_p)) + ((y_m-y_p)*(y_m-y_p)) );
                h0.real = j_0(mag);
                h0.imag = y_0(mag);

                break;

                }

        case 1:             /*  far field pattern  m is phi*10      */

                {

                t* = dtr;
                t_m = (double)m/10.;
                t_m* = dtr;
                r_p = r(t,k);
                f1 = basis(t)*r_p;

                f2 = cos(t_m-t);
                f2 = f2*beta*r_p;

                h0.real = cos(f2);
                h0.imag = sin(f2);
                }

        }/*  end of switch          */

/*              h0.real = 1.;
                h0.imag = 0.;*/

                fcn_value.real = h0.real*f1;
                fcn_value.imag = h0.imag*f1;

return fcn_value;
}
```

R.C

```
/*
*******************************************************
        This function will describe the 2-d EFIE
        surface to be solved by using an r(theta)
        function
```

```
*****************************************************
*/

/*  include files                                   */

/*  External variables                                  */
extern double radius;

/*  prototype functions                                 */


/*  function for a cylinder                         */
double r(double theta,double ka)
{
/*  NOTE:  theta is in radians!!!                   */
double r_value;

r_value=radius;

return r_value;
}
```

B_T_FCN.C

```
/*
*****************************************************
        This module holds the basis
                and
        testing functions
*****************************************************
*/

double basis(double t)
{
double basis_value;
/*  for pulse basis, can use: constant!                 */


basis_value=1;
/* for Galerkin, can use the following                          */
/*
ans=testf(r_prime, theta_prime);
*/
```

```
return basis_value;
}

double testf(double t_t)
{
double testf_value;
/*  for pulse basis, can use:  constant!                    */
testf_value=1;
return testf_value;
}
```

H_0.C

```
/*
*****************************************
        This function returns the value
        of J_0(x), by polynomial approximation
        (ONLY to be used if no recursion
        is done).  See section 9.4 of
        Abramowitz and Stegan
*****************************************
*/

/*  Include functions                                       */
#include <math.h>

/*  prototype functions, if any                             */

double j_0(double x)
{
double a[7],b[8],x_3,x_3_2,
                    x_3_3,
                    x_3_4,
                    x_3_5,
                    x_3_6,
                    x_3_8,
                    x_3_10,
                    x_3_12;

double f_0,theta_0,j_0_value;

if(x < 3)
        {
```

```
        a[0]=1.0;
        a[1]=2.2499997;
        a[2]=1.2656208;
        a[3]=.3163866;
        a[4]=.0444479;
        a[5]=.0039444;
        a[6]=.0002100;

        x_3=   x/3;
        x_3_2= x_3*x_3;
        x_3_4= x_3_2*x_3_2;
        x_3_6= x_3_4*x_3_2;
        x_3_8= x_3_4*x_3_4;
        x_3_10=x_3_6*x_3_4;
        x_3_12=x_3_6*x_3_6;

        j_0_value=     a[0]
                  -a[1]*x_3_2
                  +a[2]*x_3_4
                  -a[3]*x_3_6
                  +a[4]*x_3_8
                  -a[5]*x_3_10
                  +a[6]*x_3_12;
        }
else
        {
        /*      coefficients for f_0                    */
        a[0]=.79788456;
        a[1]=.00000077;
        a[2]=.00552740;
        a[3]=.00009512;
        a[4]=.00137237;
        a[5]=.00072805;
        a[6]=.00014476;

        /*      coefficients for theta_0        */
        b[0]=1;
        b[1]=.78539816;
        b[2]=.04166397;
        b[3]=.00003954;
        b[4]=.00262573;
        b[5]=.00054125;
        b[6]=.00029333;
        b[7]=.00013558;
```

```
        x_3=3/x;

        x_3_2=x_3 *x_3;
        x_3_3=x_3_2*x_3;
        x_3_4=x_3_3*x_3;
        x_3_5=x_3_4*x_3;
        x_3_6=x_3_5*x_3;

        f_0=    a[0]
                -a[1]*x_3
                -a[2]*x_3_2
                -a[3]*x_3_3
                +a[4]*x_3_4
                -a[5]*x_3_5
                +a[6]*x_3_6;

        theta_0= b[0]*x
                -b[1]
                -b[2]*x_3
                -b[3]*x_3_2
                +b[4]*x_3_3
                -b[5]*x_3_4
                -b[6]*x_3_5
                +b[7]*x_3_6;

        j_0_value=f_0*cos(theta_0)/sqrt(x);
        }

return(j_0_value);
} /*  end of function, j_0(x)                                    */


double y_0(double x)
{
double a[7],b[8],x_3,x_3_2,
                x_3_3,
                x_3_4,
                x_3_5,
                x_3_6,
                x_3_8,
                x_3_10,
                x_3_12;

double f_0,theta_0,y_0_value;
```

```
if(x < 3)
        {

        double pi=3.141592654;

        a[0]=.36746691;
        a[1]=.60559366;
        a[2]=.74350384;
        a[3]=.25300117;
        a[4]=.04261214;
        a[5]=.00427916;
        a[6]=.00024846;

        x_3=   x/3;
        x_3_2= x_3*x_3;
        x_3_4= x_3_2*x_3_2;
        x_3_6= x_3_4*x_3_2;
        x_3_8= x_3_4*x_3_4;
        x_3_10=x_3_6*x_3_4;
        x_3_12=x_3_6*x_3_6;

        y_0_value=    (2/pi)*log(x/2.)*j_0(x)
                +a[0]
                +a[1]*x_3_2
                -a[2]*x_3_4
                +a[3]*x_3_6
                -a[4]*x_3_8
                +a[5]*x_3_10
                -a[6]*x_3_12;
        }
else
        {
        /*      coefficients for f_0                    */
        a[0]=.79788456;
        a[1]=.00000077;
        a[2]=.00552740;
        a[3]=.00009512;
        a[4]=.00137237;
        a[5]=.00072805;
        a[6]=.00014476;

        /*      coefficients for theta_0        */
        b[0]=1;
        b[1]=.78539816;
```

```
        b[2] = .04166397;
        b[3] = .00003954;
        b[4] = .00262573;
        b[5] = .00054125;
        b[6] = .00029333;
        b[7] = .00013558;

        x_3 = 3/x;

        x_3_2 = x_3  *x_3;
        x_3_3 = x_3_2*x_3;
        x_3_4 = x_3_3*x_3;
        x_3_5 = x_3_4*x_3;
        x_3_6 = x_3_5*x_3;

        f_0 =     a[0]
                 -a[1]*x_3
                 -a[2]*x_3_2
                 -a[3]*x_3_3
                  + a[4]*x_3_4
                 -a[5]*x_3_5
                  + a[6]*x_3_6;

        theta_0 = b[0]*x
                 -b[1]
                 -b[2]*x_3
                 -b[3]*x_3_2
                  + b[4]*x_3_3
                 -b[5]*x_3_4
                 -b[6]*x_3_5
                  + b[7]*x_3_6;

        y_0_value = f_0*sin(theta_0)/sqrt(x);
        }

return(y_0_value);
} /*  end of function, y_0(x)                                    */
```

EXCIT.C

```
/*
*******************************
This file creates excitation
```

```
vector, b
*******************************
*/
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include "cplxmath.h"

/* Declare external variables                                    */
extern int N;
extern double radius;
extern struct Complex huge *a, *b, *x;
extern FILE *data_out;


/* Prototypes                                                    */
double r(double theta,double ka);


void excitation(void)
{
int n;
double dtr,x,pi=3.141592654,theta,ka;

dtr=pi/180.;
ka=pi;

for(n=0;n<N;n++)
        {
        theta=(double)(n+0.5)*360/(double)N;
        x=r(theta,ka);
        x=x*cos(theta*dtr);

        (b+n)->real= cos(2*pi*x);
        (b+n)->imag=-sin(2*pi*x);
        }
if((data_out=fopen("data.out","at"))==NULL) {
        puts("cannot open data_out file for B!\n");
        exit(1);
        }

fprintf(data_out,"\n\nThe B vector is:\n");

for(n=0;n<N;n++)
```

```
        {
        fprintf(data_out,"%d %f + (%f)\n",n + 1,((b + n)- > real),((b + n)- > imag));
        }
fclose(data_out);

}
```

L_U.C

```
/*
*************************************
        This function takes a
        matrix (complex) and finds
        the solution x, to Ax = b
*************************************
*/
#include < stdio.h >
#include < conio.h >
#include < math.h >
#include < alloc.h >
#include < stdlib.h >
#include "cplxmath.h"


extern int N;
extern double radius;
extern struct Complex huge *a, *b, *x; /*  These are for the matrix and
                                           vector where Ax = b                  */
extern FILE *data_out;

void L_U_Decomposition(void)
{
int i,j,k;
struct Complex sum_1;

/*  Now, begin the L - U decomposition to find the inverse and to solve
        the matrix, also find the determinant...The U portion is in the
        upper part of the original matrix memory, the L portion is in the
        lower part of the original matrix memory, where it is assumed that
        the L part has a unity diagonal.  Thus, the determinant of the
        matrix is the product of the diagonal elements                      */
```

```
for(k=0;k<N;k++)
        {
        for(i=k+1;i<N;i++)
                {
                *(a+N*i+k)=Divide( *(a+N*i+k),*(a+N*k+k) );
                for(j=k+1;j<N;j++)
                        {
                        *(a+N*i+j)=Subtract(*(a+N*i+j),Multiply
                                (*(a+N*i+k),*(a+N*k+j) ));
                        }
                }
        }

/*  Compute the Determinant...by finding product of diagonal elements      */
sum_1.real=1;
sum_1.imag=0;

for(i=0;i<N;i++)
        sum_1=Multiply(sum_1,*(a+N*i+i) );

printf("The determinant of the system is %e\n",sum_1);

/*-------END OF LU DECOMPOSITION----------------------------------*/
return;
}

/*-------BEGINNING OF "SOLVE" FUNCTION...-----------------------*/
void Solve(void)
{
int i,j;
struct Complex sum_1;

/*  Forward substitution, to find the intermediate vector in LU solution*/

*(x)=*(b);
for(i=1;i<N;i++)
        {
        sum_1.real=0;
        sum_1.imag=0;
        for(j=0;j<i;j++)
                {
                sum_1=Add(sum_1,Multiply(*(a+N*i+j),*(x+j) ) );
                }
        *(x+i)=Subtract(*(b+i),sum_1) ;
```

```
        }

/*  Backward substitution, to complete process and find solution    */
*(x+N-1)=Divide(*(x+N-1),*(a+(N)*(N-1)+N-1));
for(i=N-2;i>=0;i--)
        {
        sum_1.real=0;
        sum_1.imag=0;
        for(j=i+1;j<N;j++)
                {
                sum_1=Add(sum_1,Multiply(*(a+N*i+j),*(x+j)));
                }
        *(x+i)=Divide( Subtract(*(x+i),sum_1),*(a+N*i+i));
        }

/*for(i=0;i<N;i++)
        {
        printf("%d:  %f+j(%f)\n",i+1,((x+i)->real),((x+i)->imag));
        }       */
}/*----------END OF "SOLVE" FUNCTION----------*/

/*----------REWRITE AS "SOLUTION_CHECK" FUNCTION----------*/
#include <time.h>

void Solution_Check(void)
{
int i,j;
double error;
struct Complex sum_1,r_1;
FILE *out;
time_t t;


/*  Now, check the result by multiplying
        the result by the original matrix                */

error=0;

if((out=fopen("data.out","at"))==NULL)
        {
        puts("cannot open for error analysis, data_out file\n");
        exit(1);
        }
fprintf(out,"\n\n");
```

```
for(i=0;i<N;i++)

        {
        sum_1.real=0;
        sum_1.imag=0;
        for(j=0;j<N;j++)
                {
                sum_1=Add(sum_1,Multiply(*(a+N*i+j),*(x+j)));
                }
        fprintf(out," Ax=%f+j(%f); \t\tb=%f+j(%f)\n",sum_1.real,sum_1.imag,((b+i)->real),((b+i)->imag));
        r_1=Subtract(sum_1,*(b+i));
        error=error+Magnitude(r_1);
        }
fprintf(out,"\nThe total MSE in the computation is:  %f\n\n",error);

fclose(out);
/*----------END OF "SOLUTION_CHECK" FUNCTION----------*/

}
```

PATTERN.C

```
/*
**********************************
  Module to compute far field
  pattern due to calculated
  currents
**********************************
*/

#include <stdio.h>
#include <math.h>
#include "cplxmath.h"


/* Declare external variables                                          */
extern int N;
extern double radius;
extern struct Complex huge *a, *b, *x; /*  These are for the matrix and vectors
                                        where Ax=b                      */

/*  Prototype functions                         */
struct Complex matrix_element(int n, int m, int flag);
```

```
double pattern(int phi,int flag)  /*------BEGINNING OF FUNCTION----*/
{
int n;
double solution;
struct Complex sum_ff,term_ff;

sum_ff.real=0;
sum_ff.imag=0;

for(n=0;n<N;n++)
        {
        term_ff=matrix_element(n,phi,flag);
        term_ff=Multiply ( term_ff,*(x+n) );
        sum_ff=Add(sum_ff,term_ff);
        }
solution=Magnitude(sum_ff);

return solution;
}
```

CPLXMATH.H

```
#ifndef _CPLXMATH.H_
#define _CPLXMATH.H_

/*****************************************************************************/
/*                                                                           */
/*     Jeff Swart          October, 1992          Turbo C++ 3.0              */
/*                                                                           */
/*  #defines, structures, and function prototypes for Complex math.          */
/*                                                                           */
/*****************************************************************************/

/*****************************************************************************/
/*  #Defines                                                                 */
/*****************************************************************************/

/* Pi                                                                        */
#define PI 3.1415927
```

```
/*****************************************************************************/
/*  Structures                                                               */
/*****************************************************************************/

/* Complex number in Cartesian form                                          */
struct Complex
 {
  double real,              /* Real part of Complex number                   */
         imag;              /* Imaginary part of Complex number              */
 };



/*****************************************************************************/
/*  Function prototypes                                                      */
/*****************************************************************************/

struct Complex Divide(struct Complex num_cart,
                      struct Complex denom_cart);

        /*  Added by J. Richie, January, 1993  */

struct Complex Multiply(struct Complex num1,
                        struct Complex num2);

struct Complex Add(struct Complex add1,
                   struct Complex add2);

struct Complex Subtract(struct Complex subp,
                        struct Complex subm);

double Magnitude(struct Complex z);

struct Complex Conjugate(struct Complex z);


#endif
```

CPLXMATH.C

```
#include <math.h>
#include "cplxmath.h"
```

```
/*************************************************************************/
/*                                                                       */
/*     Jeff Swart          October, 1992          Turbo C++ 3.0          */
/*                                                                       */
/*  Basic Complex math functions.  Prototypes are located in CPLXMATH.H.  */
/*                                                                       */
/*************************************************************************/

/*************************************************************************/
/*  'RadToDeg' will convert a value given in radians as a double to the  */
/*  equivalent value in degrees, and return the result as a double.      */
/*************************************************************************/


/*************************************************************************/
/*  'DivComplexCart' will divide two Complex numbers given in Cartesian  */
/*  RENAMED TO Divide:  J. richie, 2/93                                  */
/*  form as structures of type 'cart_Complex', and return the result in  */
/*  Cartesian form as a structure of type 'cart_Complex'.                */
/*************************************************************************/

struct Complex Divide(struct Complex num,
                      struct Complex denom)
{
 /************************************************************************/
 /*  Variable declarations                                               */
 /************************************************************************/

 struct Complex                 /* Polar form of numerator         */
                                /* Polar form of denominator       */
                result    ;     /* Polar form of result            */


 double mag_1,mag_2,angle_1,angle_2;

 /************************************************************************/
 /*  Code                                                                */
 /************************************************************************/

 /* Convert numerator and denominator to polar form                     */

 mag_1 = Magnitude(num);
 mag_2 = Magnitude(denom);
 angle_1 = atan2(num.imag,num.real);
```

```
 angle_2 = atan2(denom.imag,denom.real);

 /* Calculate polar form of result                          */
 mag_1 = mag_1/mag_2;
 angle_1 = angle_1-angle_2;
 result.real = mag_1*cos(angle_1);
 result.imag = mag_1*sin(angle_1);


 /* Return Cartesian form of result                         */
 return result;

 } /* End function 'Divide'                                 */


/* ------------------------------------
        ADDED BY J. RICHIE, JANUARY 1993
   ------------------------------------

 Function to obtain product of two Complex numbers         */


struct Complex Multiply(struct Complex num1,
                                    struct Complex num2)
 {
 /* Variable declarations                                   */

 struct Complex result;

 /* Code                                                    */
 result.real = num1.real*num2.real-num1.imag*num2.imag;
 result.imag = num1.real*num2.imag + num1.imag*num2.real;

 return result;

 }

/* Function to obtain sum of two Complex numbers            */

struct Complex Add(struct Complex add1,
                                    struct Complex add2)
 {
 /* Variable declarations                                   */
```

```
 struct Complex result;

 /*  Code                                                    */
 result.real = add1.real + add2.real;
 result.imag = add1.imag + add2.imag;

 return result;

 }

/*  Function to obtain difference of two Complex numbers     */

struct Complex Subtract(struct Complex subp,
                                    struct Complex subm)
 {
 /*  Variable declarations                                   */

 struct Complex result;

 /*  Code                                                    */
 result.real = subp.real-subm.real;
 result.imag = subp.imag-subm.imag;

 return result;

 }

/*  Function to return the magnitude of a Complex number     */

double Magnitude(struct Complex z)
{
double mag_value;

mag_value = sqrt( (z.real*z.real) + (z.imag*z.imag) );
return mag_value;
}

/*  Function to return the Complex conjugate of a Complex number  */

struct Complex Conjugate(struct Complex z)
{
struct Complex conjug;

conjug.real = z.real;
```

```
conjug.imag = (-1)*z.imag;
return conjug;
}
```

HED_FT.C

```
/*
*********************************************
        Prints banner to FILE pointer fp
*********************************************
*/

#include <stdio.h>

void print_banner(FILE *fp)
{
fprintf(fp,"\n\n     ************************************************************");
fprintf(fp,"\n\n       2-D TM^z code written by James E. Richie, Ph.D.");
fprintf(fp,"\n\n     ************************************************************");
fprintf(fp,"\n\n\n\n");
}
```

APPENDIX II

RP.C

```
/*
*************************************************************************
        Main module for post processing of data from bay model
*************************************************************************
*/

/*      This program should be used to get the data into a format
            which is more useable than the old NEC format, by taking
            the columns and massaging them to get new columns that are
            meaningful.

*/

/*  Include appropriate header files                */

#include  < stdio.h >
#include  < string.h >
#include  < math.h >


/*  Prototypes for the functions used by this portion               */
void read_data(void);

/*  Define global variables                                 */
float x_start,y_start,z,x_incr,y_incr;
int   nx,ny;
char  power[13],antenna_length[3],freq_code[3],

        file_name[13],file_out[13];
/*

        */

/*      Beginning of main module ------------------------  */

main(void)
{
int i;
```

```
/*  Main Menu, that switches between 2d (surface) plots, and 1d plots
                for NEC data, and can be used to simplify much analysis          */

for(i=0;i<40;i++)
printf("\n");

printf("*****************************************\n");
printf("  NEC Post-Processor including graphics  \n");
printf("     written by James E. Richie, Ph.D.    \n");
printf("*****************************************\n\n\n\n");

for(i=0;i<10;i++)
printf("\n");


read_data();

return(0);
} /* ---------------------  End of Main ----------------------------  */



/*
*****************************************************************
                    This module reads in all the data
*****************************************************************
*/


/*  Include header files                                            */
#define MAX_LINE_LEN 140

/*  Prototype functions (if any)                                    */


void read_data(void) /*  ----------------------------  */
{
int i,j;
FILE *fp_in,*fp_out;
float x,y,e1,e2,e3,e4,e5,e6;
char line[140],*ch,chr;

/* Get and verify input file name                   */
```

```
printf("enter file name (output file from NEC):  ");
scanf("%s",file_name);

fp_in = fopen(file_name, "r");

if (fp_in = = NULL)
        printf("Error opening input file %s\n", file_name);

/*  Now, need to filter through the first x lines of output file from
        NEC to get to data we want                                      */

/*  Now, read in the field data                                         */
nx=0;
  do{
    nx+=1;
    fgets(line, MAX_LINE_LEN, fp_in);
    }while( (strncmp(line, "Angle", 5) != 0)&&(nx < 1e4));

if(nx > 9e3)
    {
    printf("wrong file type\n");
    exit(1);
    }


printf("Enter number of data points (all theta or phi):  ");
scanf("%d",&nx);
ny=1;

printf("Enter the output file name:  ");
scanf("%s",file_out);

        fp_out = fopen(file_out, "w");

        if (fp_out = = NULL)
             {
              printf("Error opening output file %s\n", file_out);
             exit(1);
             }

       fprintf(fp_out,"%d  %d\n",nx,ny);

             for(j=0;j<nx;j++)
                    {
```

```
                fscanf(fp_in,"%f %f", &e1,&e2 );

                fprintf(fp_out,"%f %f\n",e1,e2);
            }       /* this closes the j loop          */

    fclose(fp_out);

printf("\n");

/* close the file                                      */
fclose(fp_in);
printf("file successfully parsed and closed\n\n");

}
```

7.3 GO/GTD Code and Report

The following is a technical report from the Marquette University Electromagnetic Simulations Laboratory that discusses the theory and codes used in the GTD analysis. This report was also the required report for an independent study performed by Mr. Francis W. Forest under the supervision of Dr. Richie. The independent study is applied to Mr. Forest's progress toward his Ph.D. requirements and has laid the groundwork for a dissertation topic.

Analysis of Electromagnetic Scattering from a 3-D Cylinder Using GTD

Technical Report #12

Francis W. Forest

Electromagnetic Simulations Laboratory

December 16, 1994

INTRODUCTION

In numerically solving the majority of electromagnetic scattering problems the method used is usually dictated by the size of the scatterer as compared to the wavelength of the incident wave. An attempt to solve a problem which entails a scatterer that is relatively large compared to the incident wavelength can lead to a solution that is poorly convergent, inefficient, and impractical if the appropriate numerical method is not used. Canonical solutions exist for simple geometry problems. They are of the form of infinite summation eigenfunction solutions. Due to the infinite number of terms and the fact that a high frequency analysis is desired, the canonical solution as it stands is intractable. For scattering from a large conductive object at high frequencies the Method of Moments (MOM), Finite Element Analysis (FEA), Finite Difference Time Domain (FDTD), and Conjugate Gradient Method (CGM) fall into the category just stated. Numerical methods involving asymptotic solutions such as Geometrical Optics (GO) are better suited for high frequency scattering from large objects. For this reason, an investigation will be conducted in order to determine if GO and its counterpart the Uniform Theory of Diffraction (UTD) are appropriate for solving the three dimensional circular cylinder problem.

THEORETICAL DISCUSSION

This section contains a brief theory of geometrical optics, the Geometrical Theory of Diffraction and Uniform Theory of Diffraction. The solutions techniques in this section will be applied to the

finite cylinder model.

The analysis of high frequency electromagnetic scattering from a large conductive structure is quite fascinating. At sufficiently high frequencies, the phase of the electromagnetic wave oscillates very rapidly, which results in destructive interference with incoming waves. As a result, only localized (stationary phase) points remain and these are the main contributors to the scattered radiation. Reflection as well as diffraction is a localized phenomenon [1, pp. 754] [2]. Consider a simple conductive body as in Figure 1-1. Using GO one observes that the scattered electric field is only present on the illumination side of the scatterer. On the shadow side no GO field exists. Does this suggest that no electromagnetic radiation exists in the shadow region? Certainly not. It does suggest that other factors and contributors are involved in calculating the total scattered field from a conductive object. For the problem in Figure 1-1, the electric fields in the shadow region originate by diffraction of the incident field into the shadow region. The diffraction can come in the form of straight edge, curved edge, surface, or higher-order diffraction. The numerical technique used with GO to modify the fields in the illumination region as well as provide the appropriate fields in the shadow region, due to diffraction, is the Geometrical Theory of Diffraction (GTD). The method which ensures that the fields are finite at a shadow interface is known as the Uniform Theory of Diffraction (UTD).

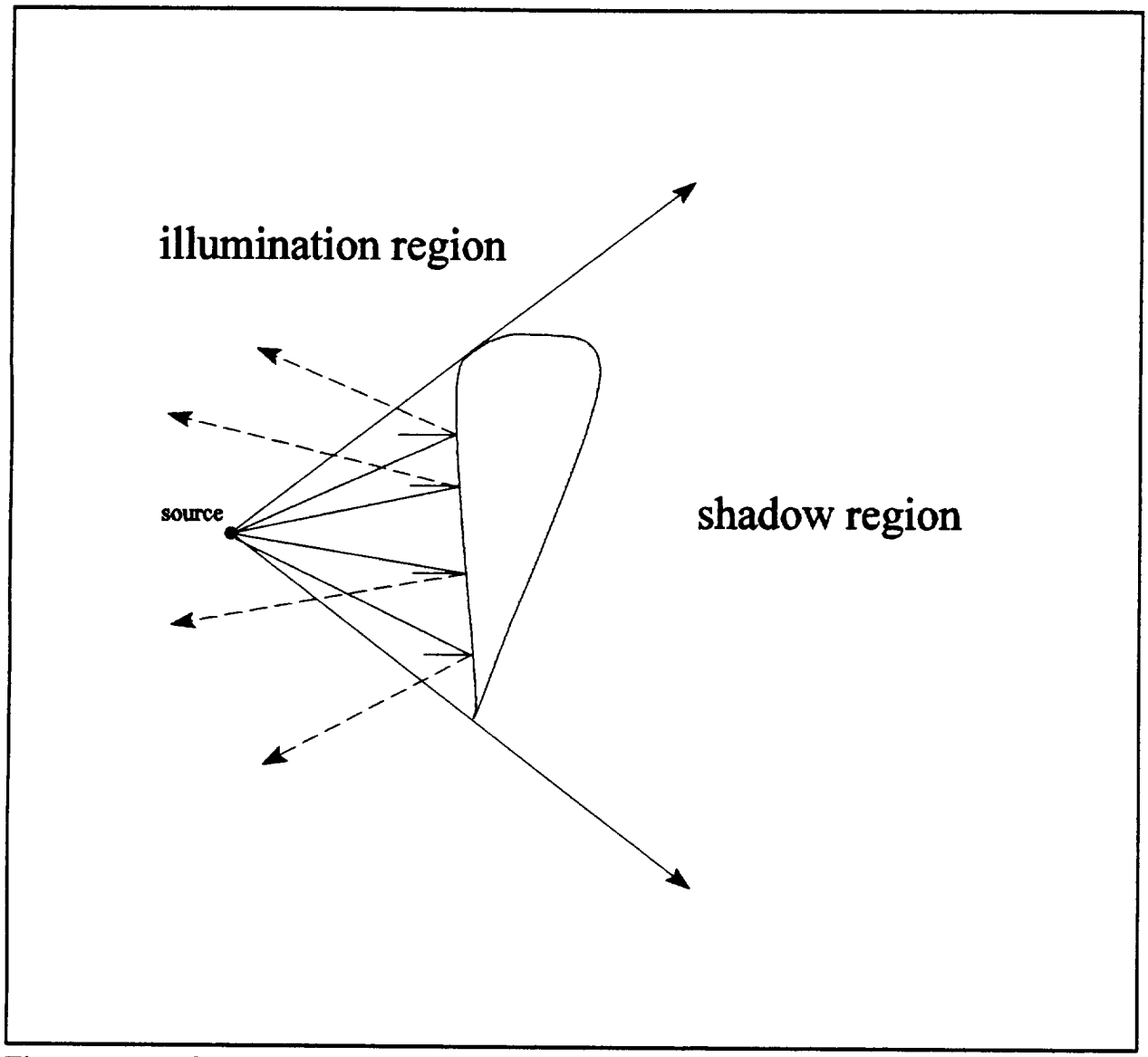


Figure 1-1: Perfectly conducting object and GO rays.

GEOMETRICAL OPTICS (GO)

Geometrical optics is a ray tracing technique for approximating the wave propagation of the incident, reflected and refracted fields. Geometrical optics gets its name from its original use as a technique for analyzing the propagation of light at high frequencies. Treating an electromagnetic wave like that of a light wave will provide appropriate magnitudes, but phase and polarization properties are not described. In order to obtain these properties a Luneberg-Kline high-frequency expansion can be used [3, pp. 166]. Basically, an asymptotic high frequency solution to Maxwell's equations in a source-free simple medium is desired. Begin by writing the electric field for large ω in a series of

$$\boldsymbol{E}(\boldsymbol{R},\omega) = e^{-j\beta_0\Psi(\boldsymbol{R})}\sum_{m=0}^{\infty}\frac{\boldsymbol{E}_m(\boldsymbol{R})}{(j\omega)^m} \tag{1-1}$$

where **R** is the position vector and β_0 is the phase constant for free-space. Substituting this series into the Helmholtz wave equation for the electric field **E** and noting the condition that the divergence of **E** is zero, an eikonal equation and transport equations are obtained by equating like powers of ω [3, pp. 166]. Since the first-order solutions of the electric field of (1-1) are of main interest for now, enforcing the conditional equations in [1, pp. 751] will allow the first-order term of (1-1) to be

$$\boldsymbol{E}(s) \approx e^{-j\beta_0\Psi(s)}\boldsymbol{E}_0(s) \tag{1-2}$$

where s is the distance along the ray path. From [3, pp. 166] it can be shown that the electric field a distance s away from some reference can be written as

$$E(s) = E_0(0)\sqrt{\frac{\rho_1\rho_2}{(\rho_1+s)(\rho_2+s)}}\, e^{-j\beta s} \qquad \text{(1-3)}$$

and is referred to as the GO field. In (1-3), $E_0(0)$ is the complex electric field at the reference point $s = 0$, ρ_1 and ρ_2 are the radii of curvature of the wave front at $s = 0$. The distance along the ray path is s. An astigmatic (not meeting at a single point) tube of rays is shown in Figure 1-2 which identifies the parameters used in (1-3). The quantity under the square root in (1-3) results from the requirement of conservation of power within the tube of rays. The power passing through the differential surface $d\sigma_0$ is the same as that passing through $d\sigma$. This approach is used to find the appropriate expressions for the GO scattering equations.

In general, the electric field at a distance s from the reflection point on a scatterer is

$$E^{sc}(s) = E^{i}(Q)\bar{R}A(s)e^{-j\beta s} \; . \qquad \text{(1-4)}$$

For simple medium and a perfectly conductive scatterer

$$\bar{R} = \begin{bmatrix} erefl & 0 \\ 0 & mrefl \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \text{(1-5)}$$

where *erefl* is the reflection coefficient of the electric field and *mrefl* is the reflection coefficient of the magnetic field.

In (1-4), *A(s)* is the spatial attenuation term. It contains the spreading and divergence information of the reflected wave. The spatial attenuation term relates the magnitude of one wave front surface with another wave front surface, and depends on the radii of curvature of the object, as well as

distance s. The spatial attenuation factor $A(s)$ is [1, pp. 755]

$$A(s) = \sqrt{\frac{\rho_1^r \rho_2^r}{(\rho_1^r + s)(\rho_2^r + s)}} \tag{1-6}$$

where ρ^r_1 and ρ^r_2 are the principal radii of curvature of the reflected wave front at the point of reflection Q. The phase factor $e^{-j\beta s}$ is simply the spatial phase shift that occurs over the distance s taken from the reflection point on the surface to the observation point.

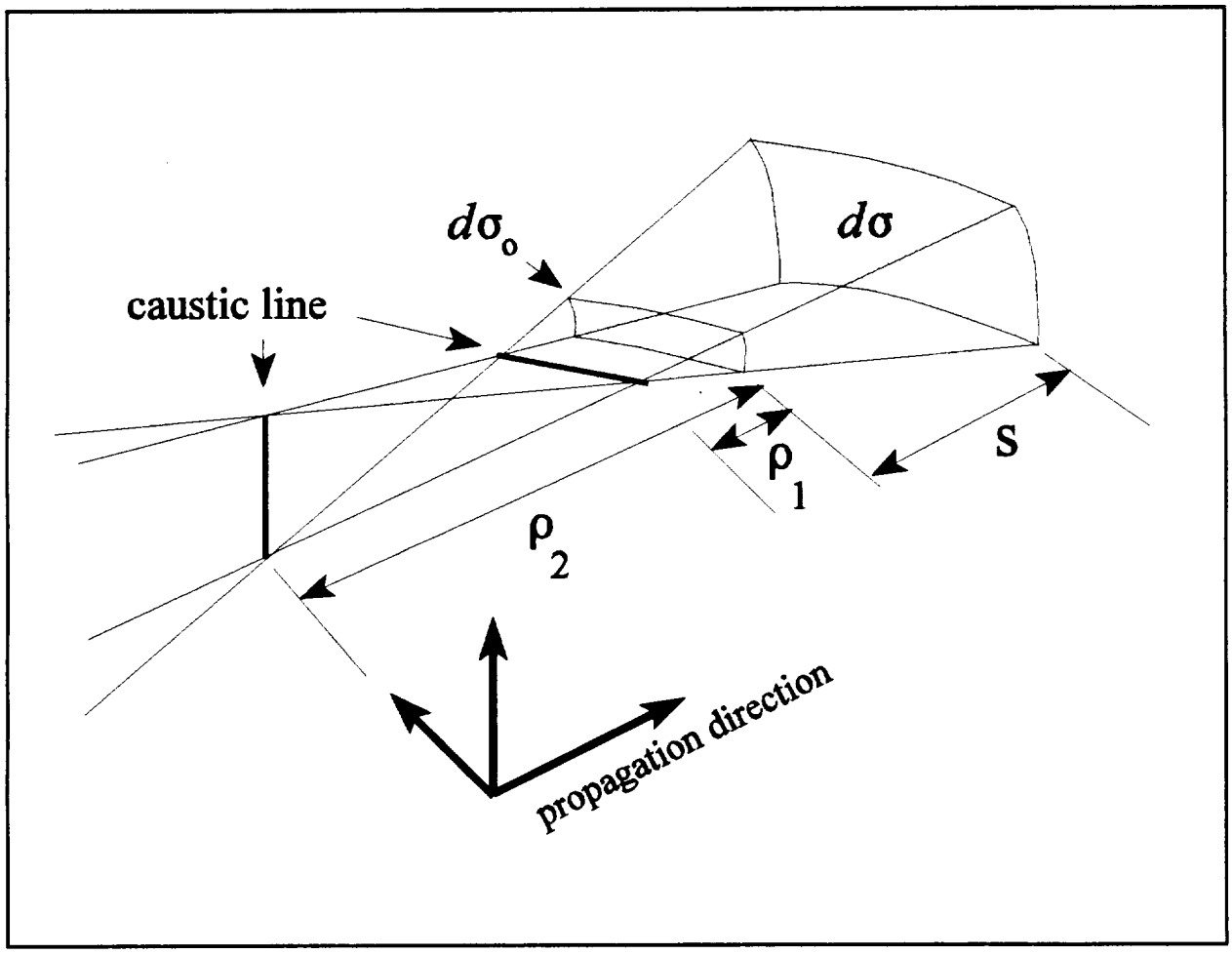


Figure 1-2: Astigmatic tube of rays and caustic lines.

UNIFORM THEORY OF DIFFRACTION (UTD)

Geometrical Theory of Diffraction (GTD) is only confined to the small number of canonical solutions available. Three types of diffraction will be discussed: 1) diffraction from a straight edge, 2) diffraction from a curved edge, and 3) diffraction from a smooth curved surface. The information presented in this section will be applied to the finite cylinder model.

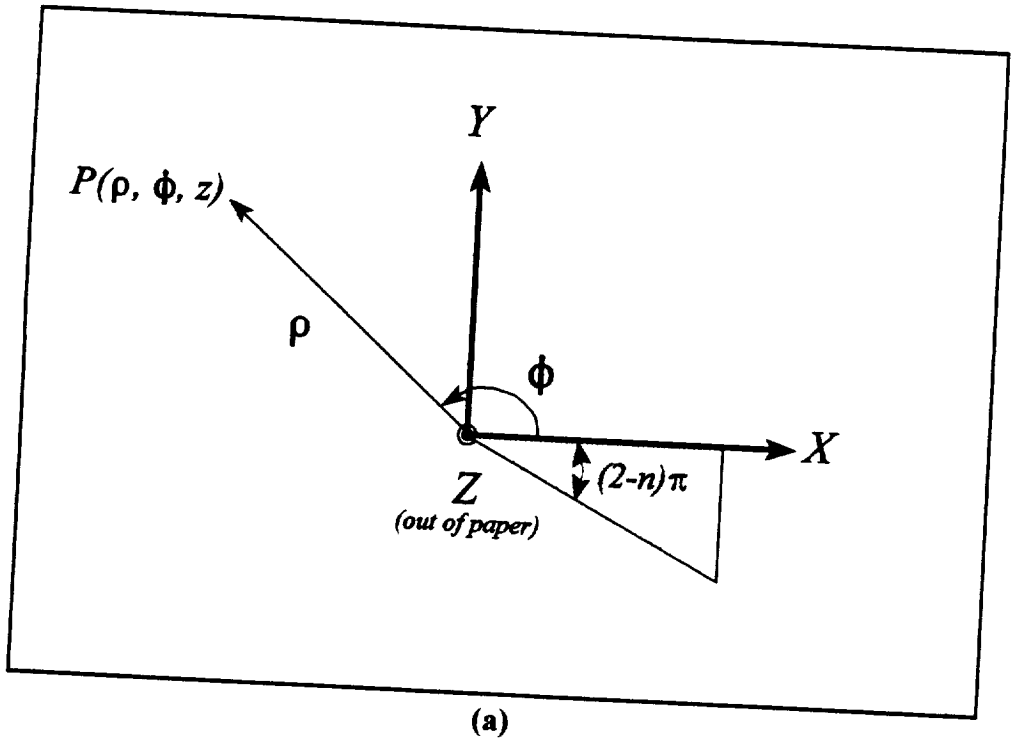
A scattered electric field created by diffraction is described by [1, pp. 767]

$$\boldsymbol{E}^{sd} = \boldsymbol{E}^{i}(Q)\boldsymbol{D}\,A(s',s)e^{-j\beta s} \tag{1-7}$$

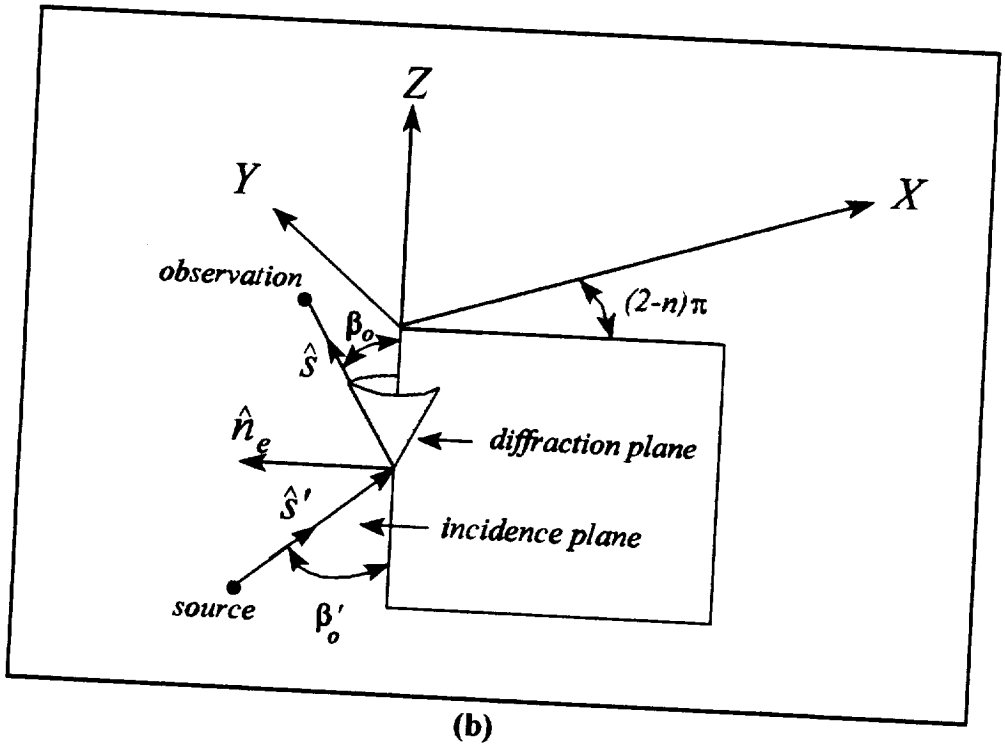
where **D** is a diadic and is termed the diffraction coefficient. The diffraction coefficient has functions such as calculating fields in the shadow region, modifying the field in the illumination region and matching the field conditions at the transition regions.

Fermat's principal of diffraction is used throughout the theory of ray tracing. Simply stated, Fermat's principal says that the "network" of rays between two points P_1 and P_2 (with a third point P_0 on the surface of an object and is stationary) follow a ray path $(P_1 P_0 P_2)$ that makes the optical distance between P_1 and P_2 an extremum (minimum). Fermat's principal is basically a variational principal on the classical GO theory [4, pp. 133].

If the components of the incident and diffracted fields are referenced by an edge-fixed coordinate system (Figure 1-3a) the dyadic diffraction coefficient has been found to be the sum of seven dyads [5]. This corresponds to a 3x3 matrix with seven non-vanishing elements. By introducing an alternative coordinate system, which changes the reference point of the rays, the diffraction coefficient could be reduced. The ray-fixed coordinate system is shown in Figure 1-3b. By defining a plane of diffraction and making appropriate reference to the incident and reflected field components, the diffraction coefficient is reduced to a 2x2 matrix with two non-vanishing elements [1, pp. 808].



(a)



(b)

Figure 1-3: (a) Edge-fixed and (b) Ray-fixed coordinate system.

It is not the intent of this paper to present a detailed theoretical discussion of the development of the diffraction coefficient and the solution of certain canonical models. In this report, only a brief theoretical discussion will pursue with the main emphasis on attempting to solve the electromagnetic scattering of a three dimensional cylinder using UTD.

STRAIGHT EDGE DIFFRACTION

Straight edge diffraction is a certain type of diffraction that can be solved using UTD because it resembles a canonical structure which has an exact solution. From a localized standpoint the diffraction from a straight edge is the same as diffraction from a wedge. The wedge is a canonical problem which has an exact solution [4, pp. 63].

Consider the situation of a line source in close proximity to a two dimensional perfectly conducting wedge as depicted in Figure 1-4. As the observation point moves about the z-axis at an angle ϕ and a distance ρ from the wedge tip, the total electromagnetic (EM) field is comprised of the summation of different fields in different regions. If the observation point is anywhere in region 1, the total EM field is the superposition of the incident, reflected, and diffracted fields. Region 2 contains the incident and diffracted fields, and region 3 only contains the diffracted field. Note the reflection and incident shadow boundaries at $\phi = \pi - \phi'$ and $\phi = \pi + \phi'$, respectively. Care must be taken when specifying the angles ϕ and ϕ'. They are to be measured from the fixed face of the wedge.

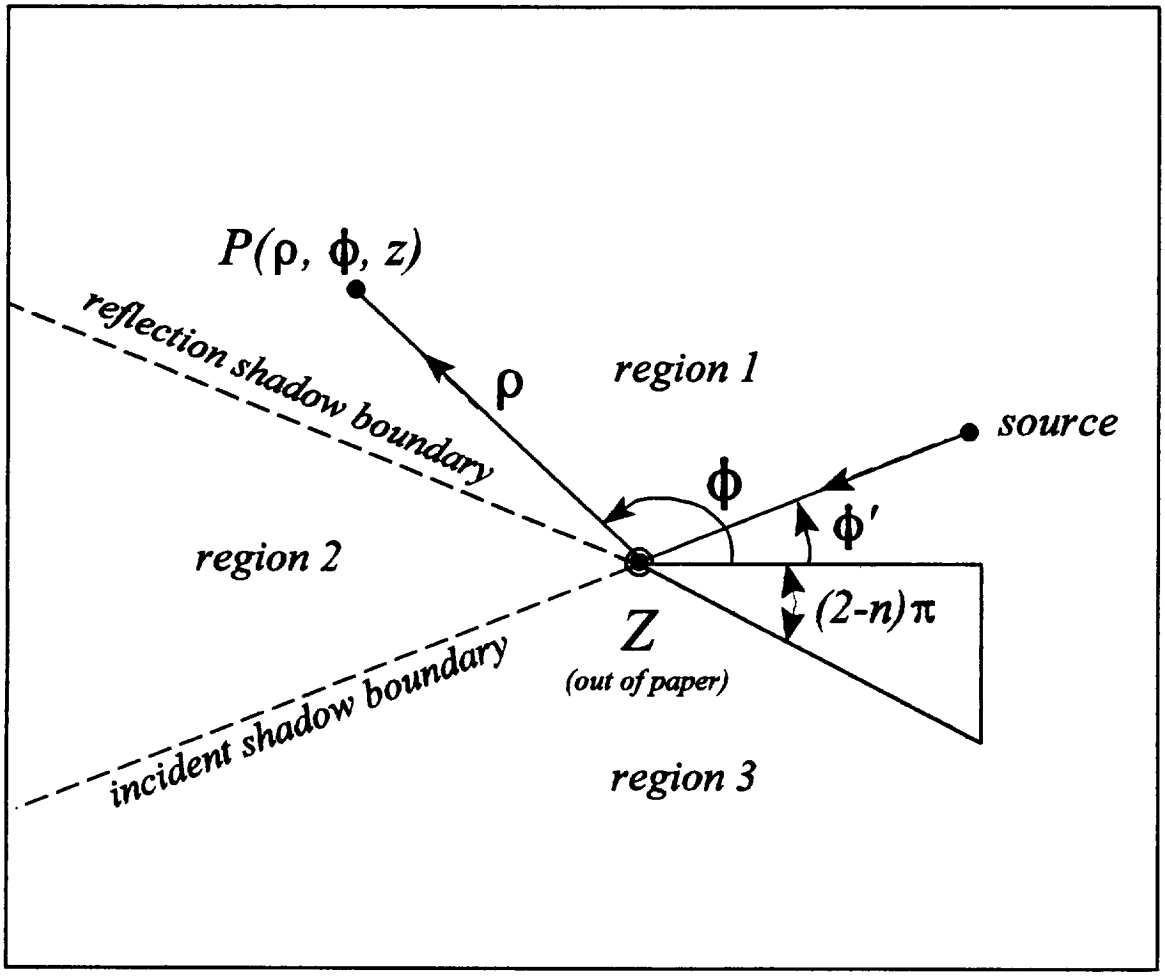


Figure 1-4: Line source in close proximity to a two dimensional perfect conducting wedge.

The diffraction coefficient for the wedge-type problem is obtained in the following fashion [1, pp. 771-773]:

1 Modal techniques are used to find the total radiated field for an electric line source, which upon satisfying the boundary conditions, a Green's function solution is obtained. The Green's function solution is in the form of an infinite series. The series is an exact solution to the time-harmonic inhomogeneous wave equation of a line source and wedge in simple medium.

2 The Hankel function on the Green's function solution is replaced with the first term of its asymptotic expansion when the line source is assumed to be far away from the vertex of the wedge. This is then substituted into the Green's function and an expression for the electric and magnetic fields is obtained [1, Equation 13-40a].

3 Due to the rapid variation in phase of the field and the boundary conditions, in order to use the classical Method of Steepest Descent for isolated poles and saddle points, the resulting equation in step 2 must be transformed into an integral and then evaluated for large $\beta\rho$ using the Method of Steepest Descent.

An excellent presentation of the execution of the above steps is given in [1, pp. 771-796]. The resulting diffraction coefficient is known as Keller's diffraction coefficient and is valid provided the observation point is sufficiently removed from the "transition regions". At the shadow boundary the coefficient approaches infinity. The region in the neighborhood of the incident and reflected shadow boundary is referred to as the transition region. In the transition region, the field undergoes its most rapid changes [1, pp. 783]. A modification of Keller's diffraction coefficient keeps the fields finite at the incident and reflection boundaries and is known as the Uniform Diffraction Coefficient. The Uniform Diffraction Coefficients are given as [5]

$$D^{i,r}(\rho,\xi^{\mp},n) = -\frac{e^{-j\frac{\pi}{4}}}{2n\sqrt{2\pi\beta}}\left\{C^{+}(\xi^{\mp},n)F\left[\beta\rho g^{+}(\xi^{\mp})\right] + C^{-}(\xi^{\mp},n)F\left[\beta\rho g^{-}(\xi^{\mp})\right]\right\} \quad \textbf{(1-8)}$$

where

$$C^{+}(\xi^{\mp},n) = \cot\left(\frac{\pi+\xi^{\mp}}{2n}\right)$$

$$C^{-}(\xi^{\mp},n) = \cot\left(\frac{\pi-\xi^{\mp}}{2n}\right)$$

$$\xi^{\mp} = \phi \mp \phi'$$

$$F\left[\beta\rho g^{-}(\xi^{\mp})\right] = 2j\sqrt{\beta\rho g^{-}(\xi^{\mp})}\,e^{j\beta\rho g^{-}(\xi^{\mp})}\int_{\sqrt{\beta\rho g^{-}(\xi^{\mp})}}^{\infty} e^{-j\tau^2}\,d\tau$$

$$F\left[\beta\rho g^{+}(\xi^{\mp})\right] = 2j\sqrt{\beta\rho g^{+}(\xi^{\mp})}\,e^{j\beta\rho g^{+}(\xi^{\mp})}\int_{\sqrt{\beta\rho g^{+}(\xi^{\mp})}}^{\infty} e^{-j\tau^2}\,d\tau$$

In (1-8), $D^{i}(\rho,\xi^{-},n)$ is referred to as the incident diffraction coefficient for a unit amplitude incident plane wave, and $D^{r}(\rho,\xi^{+},n)$ is the reflected diffraction coefficient for a unit amplitude incident plane wave [1, pp. 787]. It may sound strange that one identifies an incident and reflected diffraction field, but this is merely the terminology used to say that the diffraction coefficient is computed and referenced from the incident and reflected shadow boundaries, respectively. In the above equations

g^+ and g^- are representatives of the angular separation between the observation point and the incident and reflected boundaries, respectively. When observations are made along the shadow boundaries, $g^{\pm} = 0$.

Upon performing the asymptotic expansion and the Method of Steepest Descent, part of the field solution resembles the Fresnel integral. In order to adequately express the field quantities in the transition region, a Fresnel transition function is used which is proportional to the Fresnel integral. Asymptotic expressions for the transition function are found in [5]. The Fresnel routine $F[\beta \rho g^{\pm}(\xi^{\pm})]$ is a measure of separation between saddle points and poles.

If both the source distance ρ' and observation distance ρ from the edge of the wedge are finite, a more accurate estimate of the distance ρ would be the so-called distance parameter L [1, pp. 805]. For the case presently being discussed,

$$L = \frac{\rho\,\rho'}{\rho+\rho'}\begin{cases}\rho' \to \infty \\ \approx \rho \\ \rho \to \infty \\ \approx \rho'\end{cases} . \qquad \textbf{(1-9)}$$

In general, L can be found by satisfying the condition that the total field must be continuous along the incident and shadow boundaries. As a result [5],

$$L = \frac{s(\rho_e^i + s)\,\rho_1^i \rho_2^i \sin^2\beta'_0}{\rho_e^i(\rho_1^i + s)(\rho_2^i + s)} , \qquad \textbf{(1-10)}$$

where ρ_1^i, ρ_2^i = radii of curvature of the incident wave front at the diffraction point, ρ_e^i = radius of curvature of the incident wave front in the edge fixed plane of incidence, and s = the distance from the diffraction point to the observation point. The angle β'_0 is the angle at which the incoming wave

is obliquely incident onto the diffraction point. Using the ray-fixed coordinate system:

$$L = \begin{cases} s\sin^2\beta'_0 & \textit{plane wave incident} \\ \dfrac{ss'\sin\beta_0\sin\beta'_0}{s\sin\beta_0 + s'\sin\beta'_0} & \textit{cylindrical wave incident} \\ \dfrac{ss'\sin^2\beta'_0}{s+s'} & \textit{conical and spherical wave incidences} \end{cases} \tag{1-11}$$

where s' is the distance from the source to the diffraction point.

The final scalar diffraction coefficients D_S and D_H (soft and hard diffraction coefficients, respectively) are named after their polarization resemblance with acoustic boundary conditions of the way the pressure field vanishes on the surface. Soft refers to a Dirichlet boundary and hard implies a Neumann boundary. Obviously, in electromagnetics one is referring to the electric and magnetic field polarization with respect to the plane of incidence. The diffraction coefficients are given as [3, pp. 177]

$$\begin{aligned} D_S(L;\phi,\phi';n,\beta'_0) &= D^i(L,\phi-\phi',n,\beta'_0) - D^r(L,\phi+\phi',n,\beta'_0) \\ D_H(L;\phi,\phi';n,\beta'_0) &= D^i(L,\phi-\phi',n,\beta'_0) + D^r(L,\phi+\phi',n,\beta'_0) \end{aligned} \tag{1-12}$$

where the scalar diffraction coefficients in (1-8) have been normalized to $\sqrt{\lambda}$. Therefore, (1-8) can now be generalized as

$$D^i(L,\phi-\phi',n,\beta'_0) = -\frac{e^{-j\frac{\pi}{4}}}{2n\sqrt{2\pi\beta}\sin\beta'_0}\left\{\cot\left[\frac{\pi+(\phi-\phi')}{2n}\right]F[\beta L g^+(\phi-\phi')] + \cot\left[\frac{\pi-(\phi-\phi')}{2n}\right]F[\beta L g^-(\phi-\phi')]\right\} \tag{1-13}$$

and

$$D^{r}(L,\phi+\phi',n,\beta'_0) = -\frac{e^{-j\frac{\pi}{4}}}{2n\sqrt{2\pi\beta}\sin\beta'_0}\left\{\cot\left[\frac{\pi+(\phi+\phi')}{2n}\right]F[\beta Lg^{+}(\phi+\phi')] + \cot\left[\frac{\pi-(\phi+\phi')}{2n}\right]F[\beta Lg^{-}(\phi+\phi')]\right\} \quad \textbf{(1-14)}$$

normalized by $\sqrt{\lambda}$.

Returning to (1-7), the way in which the field intensity varies is described by the spatial attenuation factor *A(s',s)*. For plane and conical, cylindrical, and spherical wave incidences, *A(s',s)* is, respectively:

$$A(s',s) = \frac{1}{\sqrt{s}}$$

$$A(s',s) = \frac{1}{s\sin\beta_o} \quad \textbf{(1-15)}$$

$$A(s',s) = \sqrt{\frac{s'}{s(s'+s)}}$$

As an example of the characteristics of the diffraction coefficients, consider a plane wave with unit amplitude incident on a half-plane at $\phi' = 30°$. A half-plane is created when the angle of the wedge in Figure 1-4 is such that n = 2. Using (1-7) and (1-12)-(1-14), plots of the magnitude of the incident and reflected diffraction fields for soft polarization at the indicated observation points $P(\rho,\phi)$ are shown in Figures 1-5a and b. Note that if the observation point is close to the half-plane the diffraction function is more broad. Recall that if Keller's diffraction coefficients were used, the field magnitude would approach infinity as the observation point approaches the shadow boundary.

Figure 1-5a: Incident Diffracted Field
Soft Polarization - Plane wave Diffraction By A Half-Plane

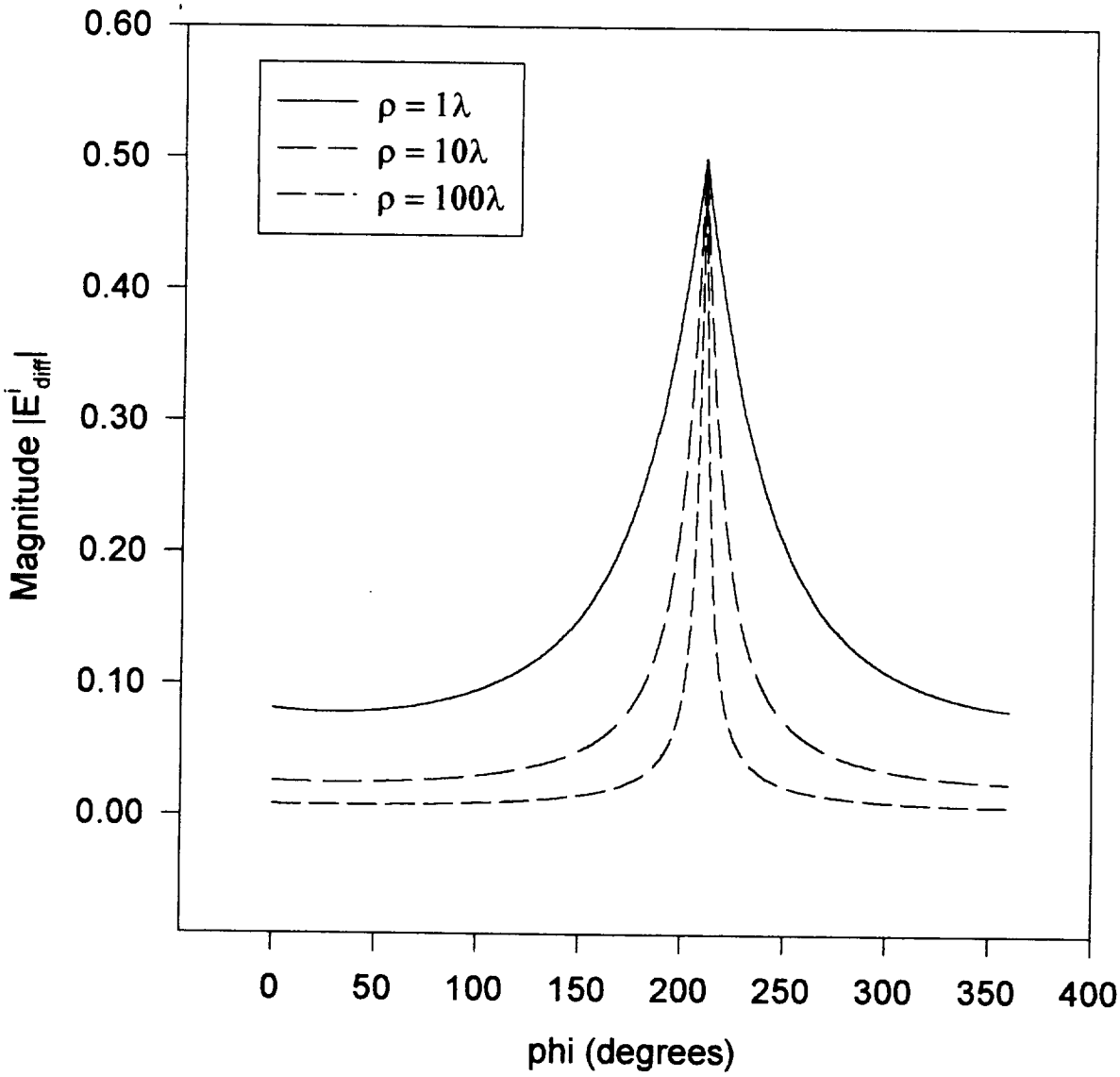
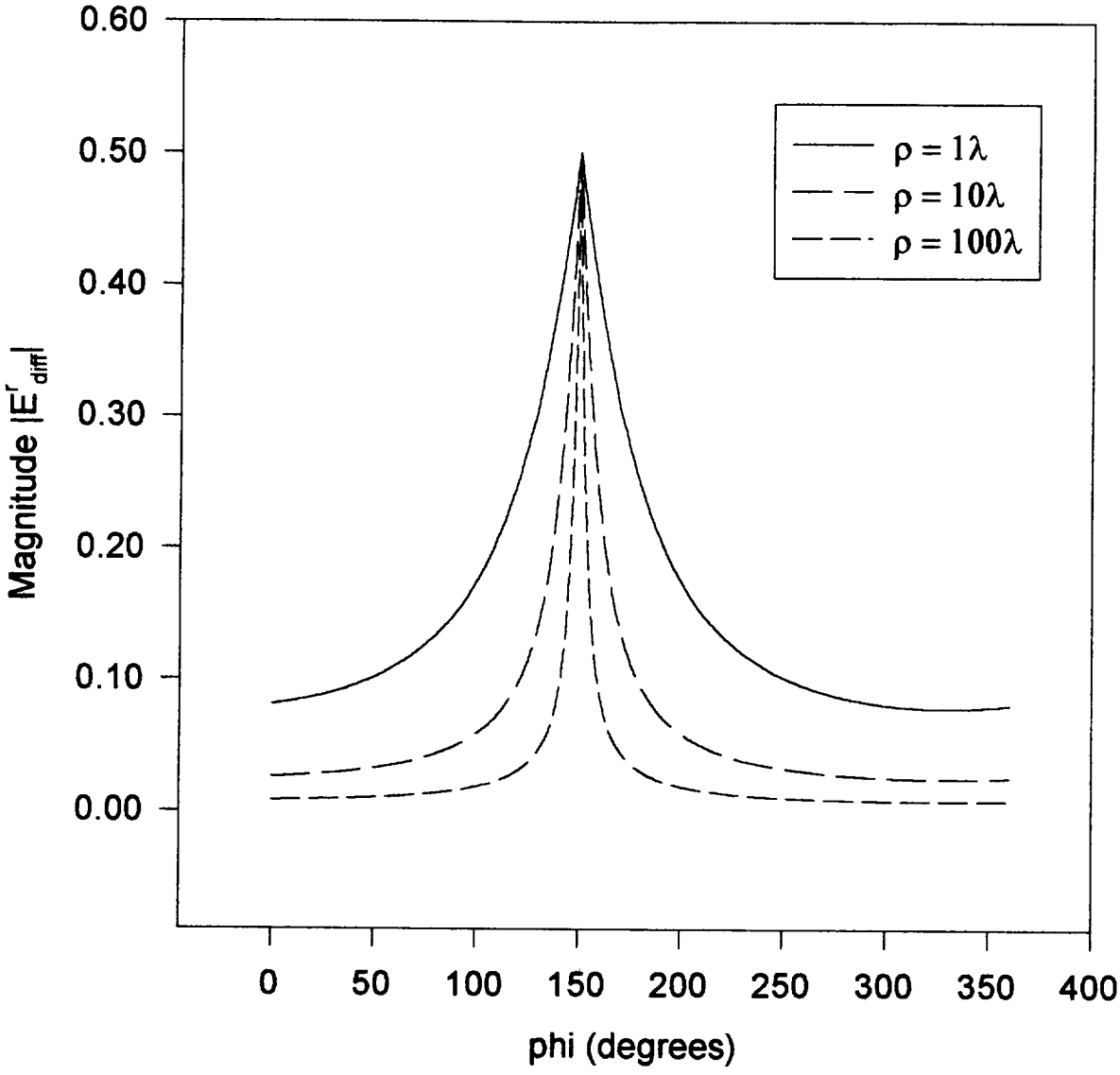


Figure 1-5b: Reflected Diffracted Field
Soft Polarization - Plane Wave Diffraction By A Half-Plane



CURVED EDGE DIFFRACTION

Recall that diffraction is a localized phenomenon. Therefore, a diffraction point can be approximated by a wedge such that its straight edge is tangent to the curved edge at the diffraction point. In addition, the wedge is oriented so that the plane surfaces are tangent to the curved surfaces of the structure. This is illustrated in Figure 1-6. It seems feasible then that the theory developed for the straight edge also applies for the curved edge problem. Since the curved shape of the structure will alter the diffraction field, the curvature must be accounted for. The corresponding modifications are made to the distance parameter L and the spatial attenuation factor $A(s',s)$ since the arguments of each involve the shape of the reflected wave front and caustic locations due to a curved edge. It is described in [1]-[5] that

$$L^{i} = \frac{s(\rho_e^i + s)\rho_1^i \rho_2^i \sin^2\beta'_0}{\rho_e^i(\rho_1^i + s)(\rho_2^i + s)} \tag{1-16}$$

and

$$L^{ro,rm} = \frac{s(\rho^r + s)\rho_1^r \rho_2^r \sin^2\beta'_0}{\rho^r(\rho_1^r + s)(\rho_2^r + s)} \tag{1-17}$$

where

ρ_1^i and ρ_2^i = radii of curvature of the incident wave front at the diffraction point

ρ_1^r and ρ_2^r = principal radii of curvature of the reflected wave front at the diffraction point

ρ_e^i = radius of curvature of the incident wave front in the edge-fixed plane of incidence

ρ^r = distance between the caustics of the diffracted ray in the direction of reflection.

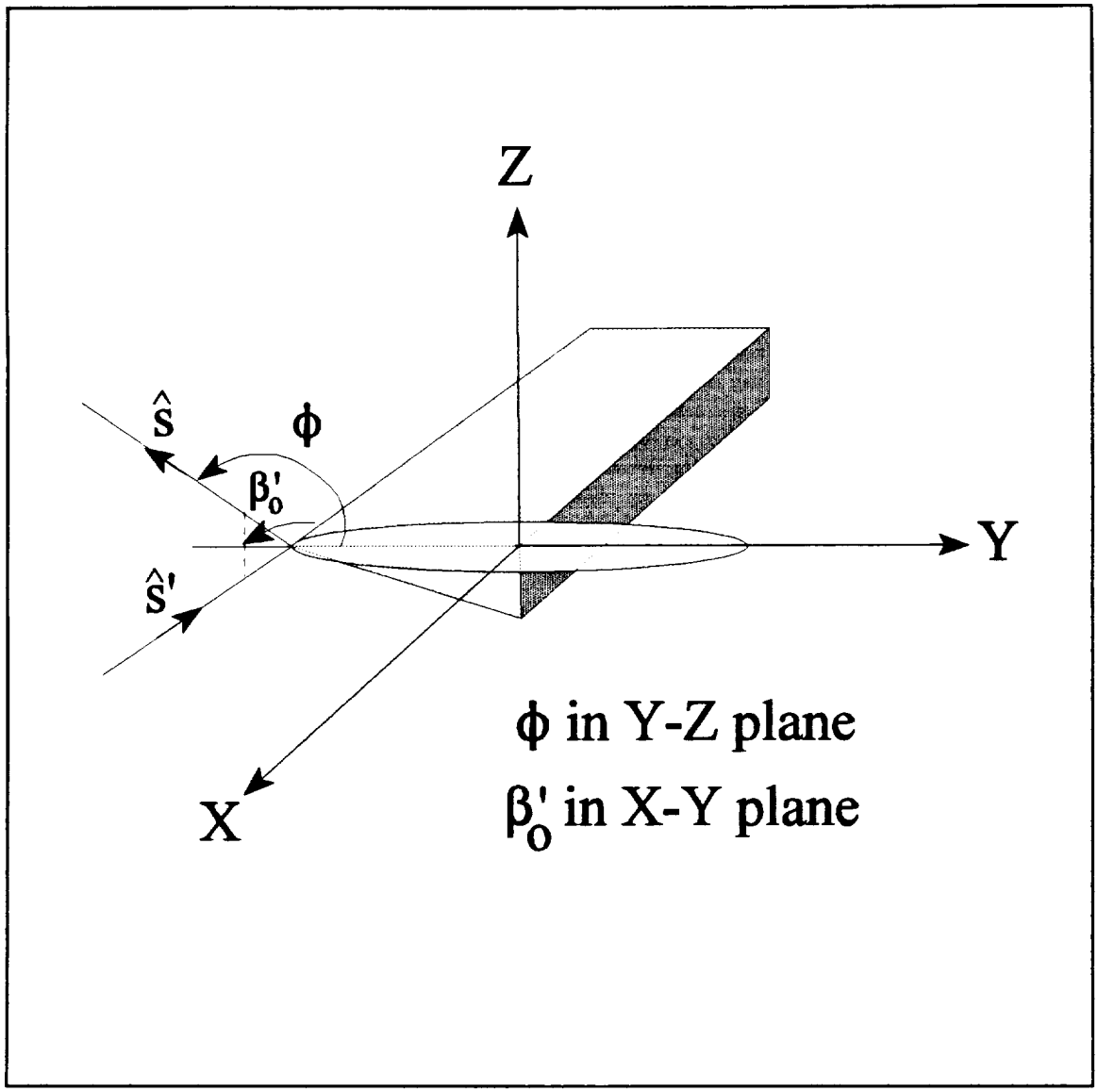


Figure 1-6: Position of wedge with respect to a diffraction point for curved edge diffraction.

The superscripts *ro* and *rn* indicate that all of the radii of curvatures pertaining to the reflected wave must be calculated from the reflection boundary π-ϕ' and the reflection boundary $(2n\text{-}1)\pi$, respectively. Thus, the uniform diffraction coefficients for curved edge diffraction are

$$D^{i}(L^{i},\phi-\phi',n,\beta'_0) = -\frac{e^{-j\frac{\pi}{4}}}{2n\sqrt{2\pi\beta}\sin\beta'_0}\left\{\cot\left[\frac{\pi+(\phi-\phi')}{2n}\right]F[\beta L^{i}g^{+}(\phi-\phi')] + \cot\left[\frac{\pi-(\phi-\phi')}{2n}\right]F[\beta L^{i}g^{-}(\phi-\phi')]\right\} \quad \textbf{(1-18)}$$

and

$$D^{r}(L^{i},\phi+\phi',n,\beta'_0) = -\frac{e^{-j\frac{\pi}{4}}}{2n\sqrt{2\pi\beta}\sin\beta'_0}\left\{\cot\left[\frac{\pi+(\phi+\phi')}{2n}\right]F[\beta L^{rn}g^{+}(\phi+\phi')] + \cot\left[\frac{\pi-(\phi+\phi')}{2n}\right]F[\beta L^{rn}g^{-}(\phi+\phi')]\right\} . \quad \textbf{(1-19)}$$

For $A(s',s)$,

$$A(s',s) = \sqrt{\frac{\rho_c}{s(\rho_c+s)}}$$

$$\frac{1}{\rho_c} = \frac{1}{\rho_e} - \frac{\hat{n}_e\cdot(\hat{s}'-\hat{s})}{\rho_g\sin^2\beta'_0} \quad \textbf{(1-20)}$$

where

ρ_c = distance between caustic at edge and second caustic of diffracted ray

ρ_e = radius of curvature of the incident wave front in the edge-fixed plane of incidence which contains unit vectors $\hat{s}$ and $\hat{e}$.

ρ_g = radius of curvature of the edge at the diffraction point

$\hat{n}$ = unit vector normal to the edge at the diffraction point and directed away from the center of curvature

$\hat{s}'$ = unit vector in the direction of incidence

$\hat{s}$ = unit vector in the direction of diffraction

β'_0 = angle between $\hat{s}'$ and tangent to the edge at the point of diffraction

$\hat{e}$ = unit vector tangent to the edge at the point of diffraction.

CURVED SURFACE DIFFRACTION

For curved surface diffraction, the diffraction process takes place when an incident ray grazes the surface of a smooth curved structure. The ray is essentially divided into two parts at the grazing point Q: one ray continues in a straight path and is considered a GO ray, the other part follows the surface of the object into the shadow region as a surface wave (creeping wave). This surface wave sheds diffracted rays tangentially as it propagates. This effect is shown in Figure 1-7. As a consequence of Fermat's principal of surface diffraction, the surface ray is the shortest ray distance between Q and a point Q_s on the shadow side, as depicted in Figure 1-8. The surface ray follows a geodesic curve! The incident and diffracted fields are phase matched to the surface ray at points Q and Q_s.

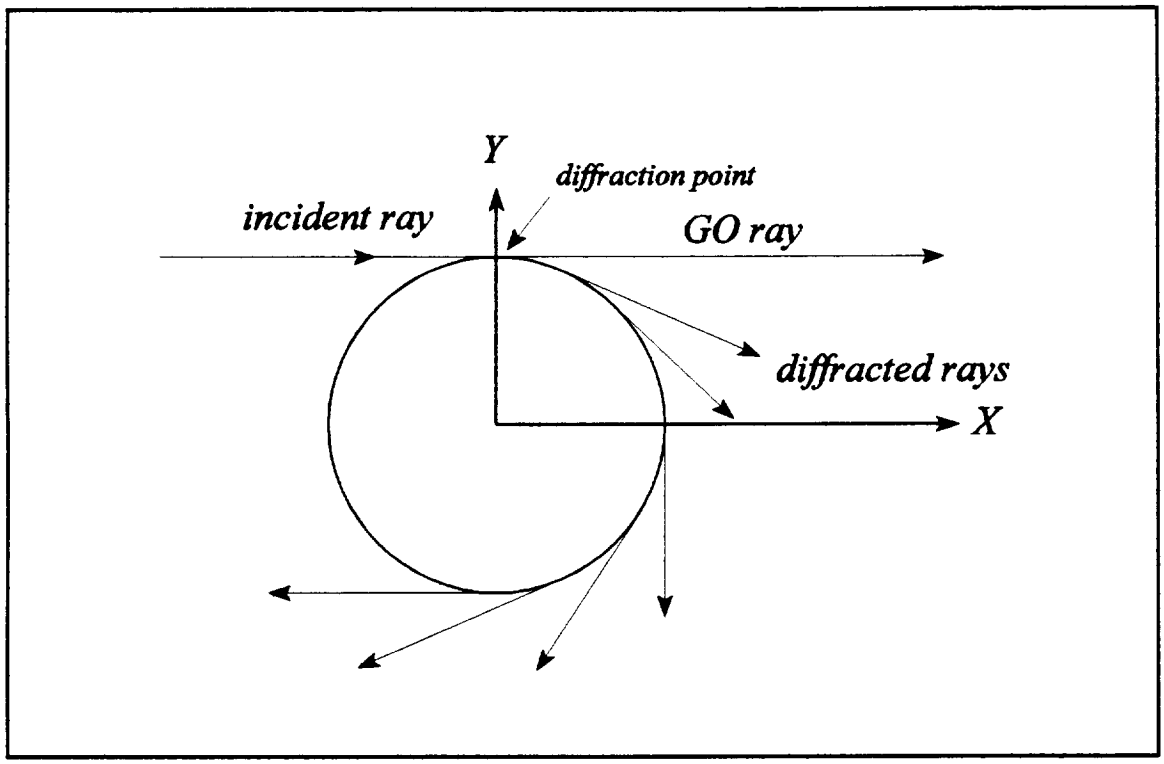


Figure 1-7: Curved surface diffraction

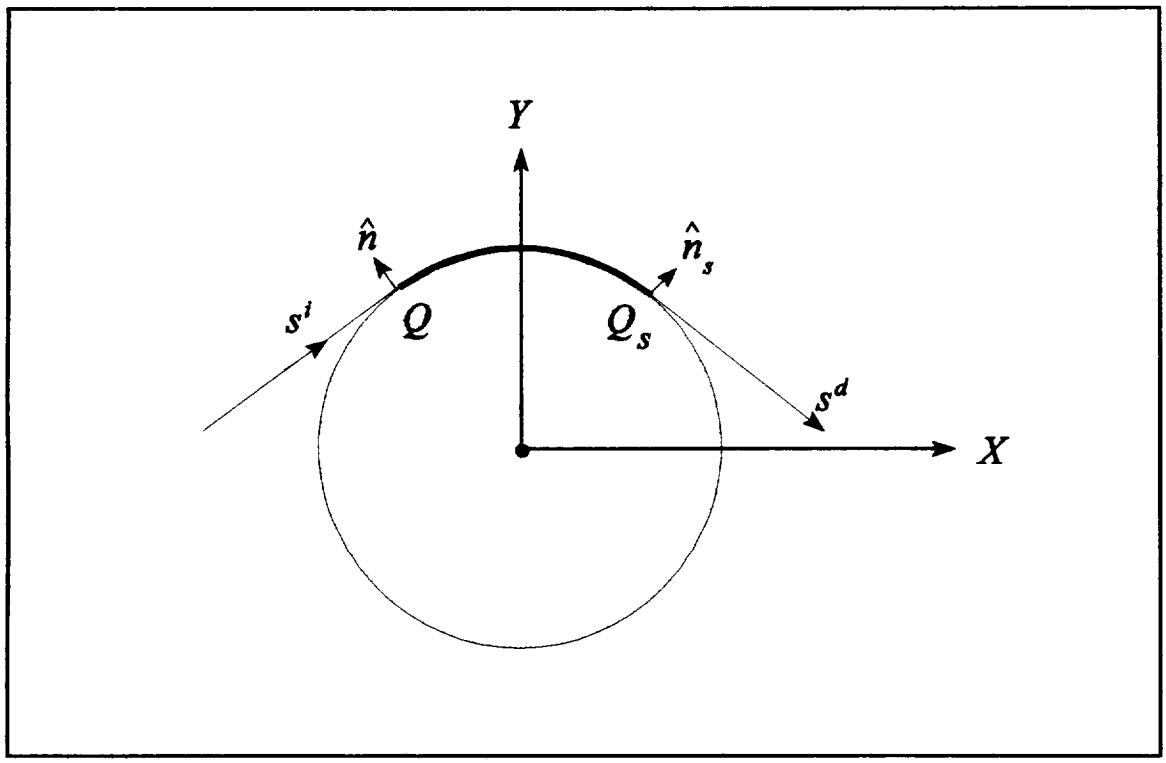


Figure 1-8: Fermat's principal of surface diffraction.

If the object was a closed surface such as a circular cylinder, it could be imagined that the surface wave continues around the object on a geodesic path an infinite number of times and sheds diffracted rays tangentially all along the path. This would seem to indicate that diffracted waves exist in the illumination side. It should be noted however that the surface ray decays exponentially as it propagates along the surface. Therefore, diffracted fields in the illumination side are considered negligible for large cylinders. Surface ray fields should not be considered real physical quantities. Their purpose is to serve as a transfer function between the incident field at Q and the diffracted field at Q_s [3, pp. 189].

It was explained that the modified Fresnel function was used in edge diffraction to more accurately predict the diffracted electromagnetic field in the transition regions. In curved surface diffraction the field behavior is more like that of the Airy function (described in [4, pp. 22-23]). An in depth explanation into the theory of surface diffraction would be very involved at this point, therefore the reader is directed to the literature for more details.

It is not necessary to discuss the foundation of curved surface diffraction from a generalized sense. The appropriate equations used in calculating the smooth curved surface diffraction will be detained until the application part of this paper. At that time only equations pertaining to a circular cylinder will be discussed.

APPLICATION

An evaluation into the validity of using GO and UTD as a numerical technique for solving the finite circular cylinder scattering problem will be given in this section.

The cylinder model can effectively be partitioned into three distinct parts (objects) as shown in Figure 2-1. As an initial step, a two dimensional analysis will be done on each part separately. If GO and UTD are successful then the model will be expanded to three dimensions.

The following parameters are used throughout the investigation:

Incident wave

- uniform electric field plane wave
- soft polarization (electric field is parallel to the plane of the disk)
- oblique incidence at 120 degrees (for the disk and ring)
- normal incidence (for cylinder)
- operating frequency: 1.57542 GHz (λ = 0.193856 m)

Observation

- observation distance: 300 meters (from center of object)
- in simple medium

Scatterer

- finite cylinder with perfect electric conductor surface
- radius a = 2.5 meters
- length = 9.0 meters
- in simple medium

Part A - Top of Circular Cylinder

The top of the cylinder resembles a circular disk of radius 2.5 meters. Due to the physical structure of the disk, the scattered field is comprised of a reflected field (from the flat surface) and a diffracted field (from each of the main diffraction points). This is shown in Figure 2-2.

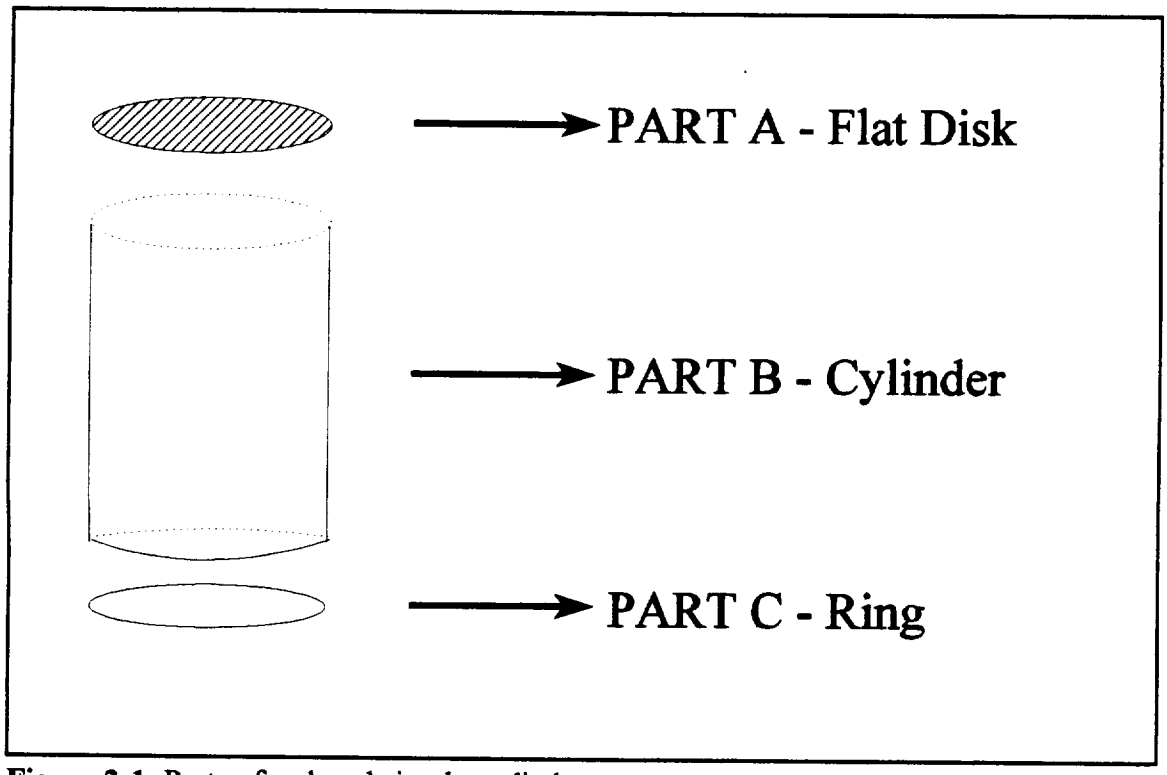


Figure 2-1: Parts of a closed circular cylinder.

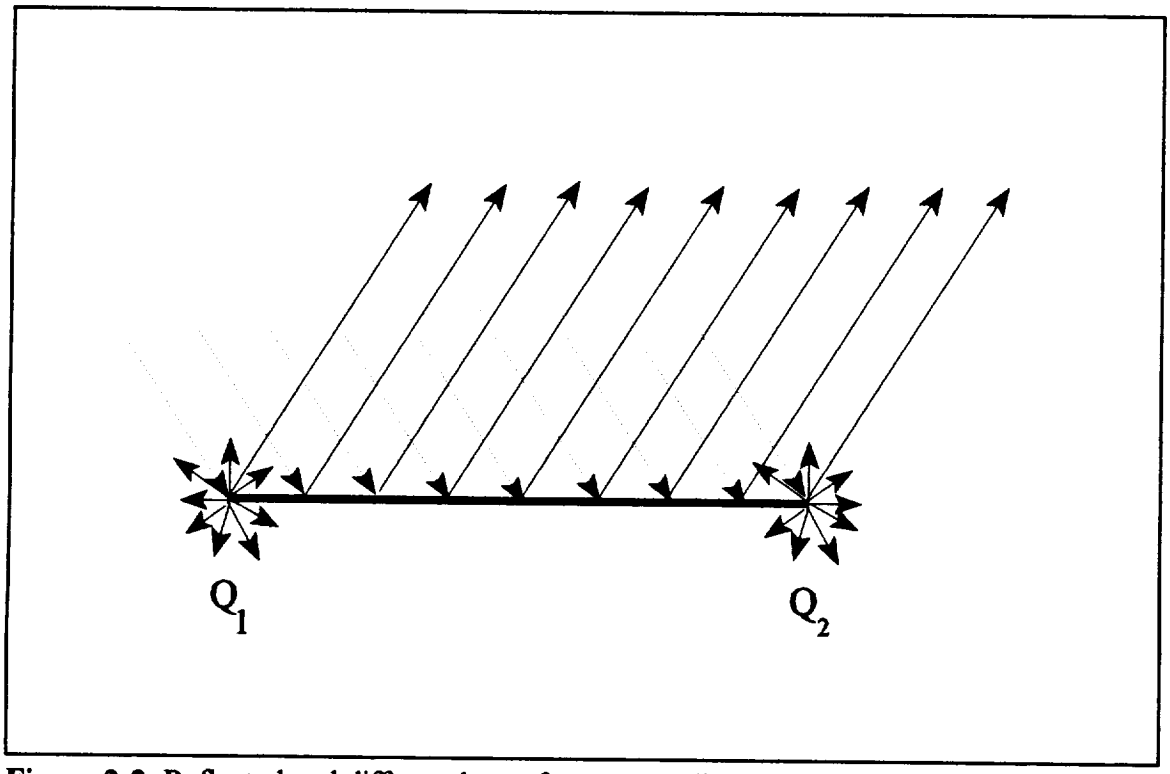


Figure 2-2: Reflected and diffracted rays from a two dimensional flat disk.

When a uniform electric field plane wave is incident onto the disk with soft polarization, the first-order diffraction points are located on the disk as shown in Figure 2-3. These points, Q_1 and Q_2 ,are the dominating diffraction points (stationary points). Other first-order diffraction points tend to be negligible or cancel one another and will not be considered in the problem. Therefore, the diffraction from the disk can be reduced to a two dimensional problem.

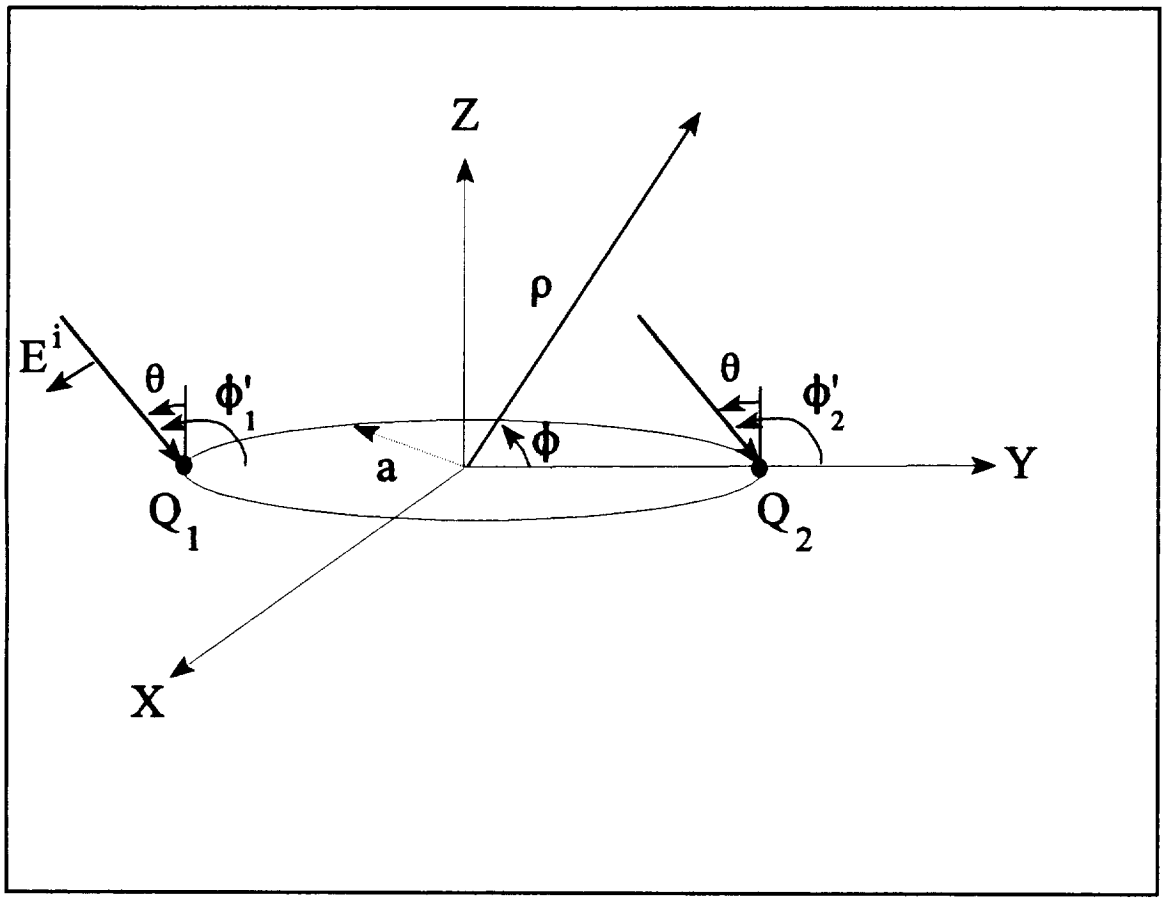


Figure 2-3: Diffraction points of a circular disk due to an incident plane wave with soft polarization orientation.

A complete analysis would require higher-order diffraction terms which would account for polarization dependence [6]. However, since the electric field is parallel to the disk surface, second-

order regular diffractions do not occur at Q_1 and Q_2 for soft polarization. Hard polarization does produce first-, second-, and third-order diffractions at Q_1 and Q_2 [6]. Other types of diffraction due to creeping waves and slope diffraction are existent, but will not be included. A brief discussion of these diffraction types will pursue later. The diffraction from the curved edge of a disk can be obtained by assuming that the diffraction points are the same from the tip of a half plane. This is illustrated in Figure 1-6 when n = 2.

The incident uniform plane wave can be expressed as

$$\bar{E}^{inc} = \hat{a}_x E_o e^{j\beta(y\sin\theta + z\cos\theta)} = \hat{a}_x E_o e^{j\beta(y\cos\phi' + z\sin\phi')} \quad . \tag{2-1}$$

In cylindrical coordinates

$$y = \rho\cos\phi \quad and \quad z = \rho\sin\phi \quad , \tag{2-2}$$

and using the trigonometric identity

$$\cos(A - B) = \cos A\cos B + \sin A\sin B \quad , \tag{2-3}$$

the incident electric field becomes

$$\bar{E}^{inc} = \hat{a}_x E_o e^{j\beta\rho\cos(\phi - \phi')} \quad . \tag{2-4}$$

Similarly, the reflected field is

$$\bar{E}^{refl} = -\hat{a}_x E_o e^{j\beta(y\cos\phi' - z\sin\phi')} = -\hat{a}_x E_o e^{j\beta\rho[\cos(\phi + \phi')]} \quad . \tag{2-5}$$

The diffracted electric fields from Q_1 and Q_2 for bistatic field scattering will now be calculated.

Recall from (1-7),

$$E^{sd} = E^{i}(Q)\, D\, A(s', s)\, e^{-j\beta s} \quad . \tag{2-6}$$

Since only soft polarization is considered, **D** in (2-6) is simply D_s of (1-12).

In Figure 2-4a, s_1, s_2, ϕ_1 and ϕ_2 are ray coordinate values, while ρ and ϕ_p are the main coordinate values. Normally, reference is made to the main coordinate system, therefore the ray coordinates must be made with respect to ρ and ϕ_p. From Figure 2-4b

$$s_1 = \sqrt{(a + y')^2 + (z')^2} \quad . \tag{2-7}$$

Note that

$$\rho^2 = (y')^2 + (z')^2 \quad \textit{and} \quad y' = \rho \cos\phi_\rho \quad . \tag{2-8}$$

Substituting (2-8) into (2-7) reveals

$$s_1 = \sqrt{\rho^2 + a^2 + 2a\rho\cos\phi_\rho} \quad . \tag{2-9}$$

Using another Law of Cosines, ϕ_ρ is found from (2-7) and (2-8) with

$$y' = s_1 \cos\phi_1 - a \quad . \tag{2-10}$$

Therefore,

$$\phi_1 = \cos^{-1}\left[\frac{s_1^2 + a^2 - \rho^2}{2as_1}\right] . \tag{2-11}$$

Similarly, for diffraction point Q_2

$$s_2 = \sqrt{\rho^2 + a^2 - 2a\rho\cos\phi_\rho} \tag{2-12}$$

and

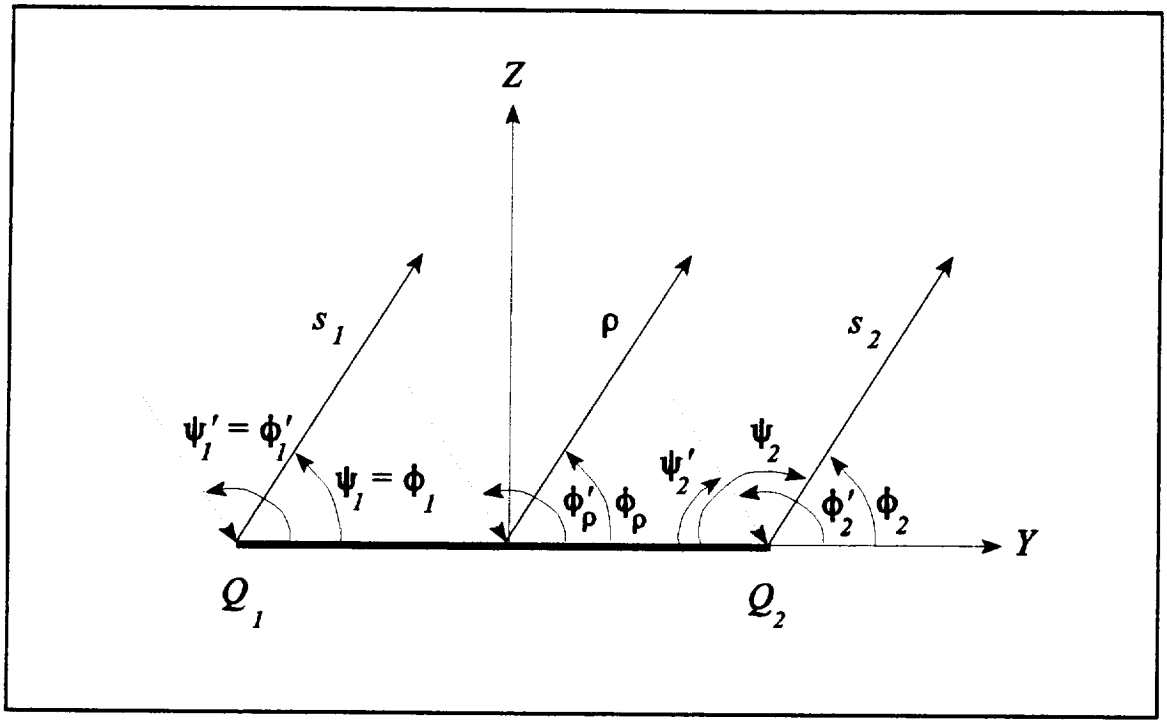
$$\phi_2 = \cos^{-1}\left[\frac{\rho^2 - s_2^2 - a^2}{2as_2}\right]. \tag{2-13}$$

The electric field at the diffraction points Q_1 and Q_2 are

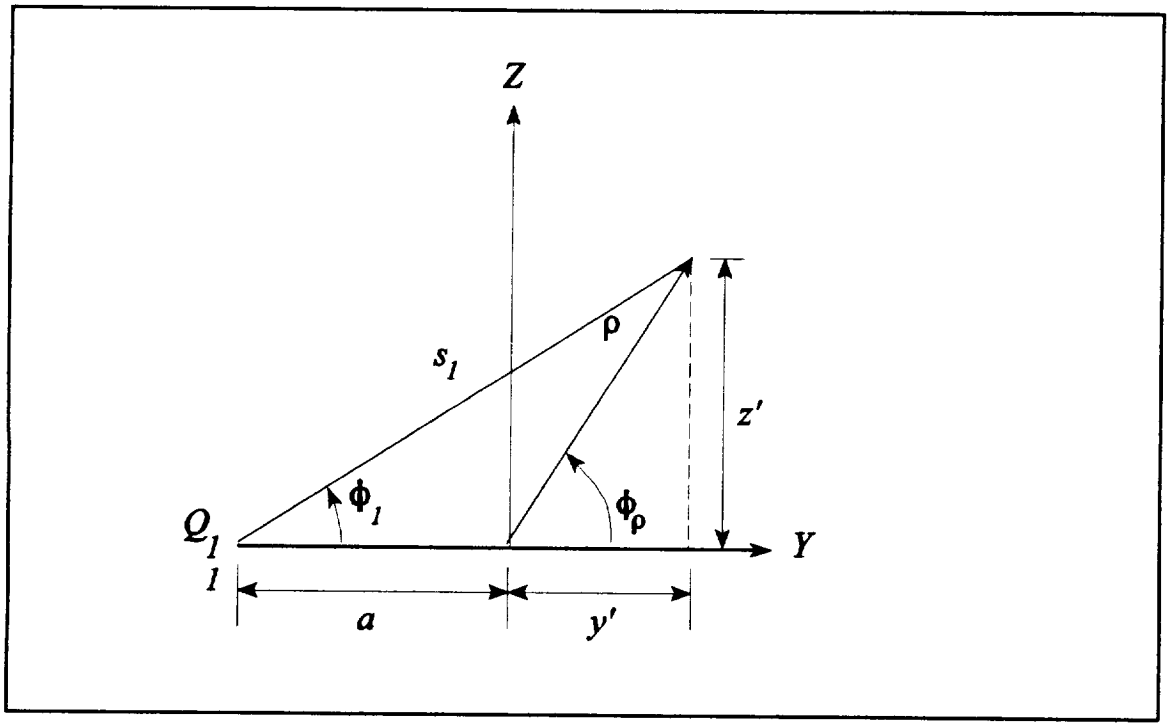
$$\bar{E}(Q_1) = \hat{a}_x E_o e^{-j\beta a\cos\phi'} \quad and \quad \bar{E}(Q_2) = \hat{a}_x E_o e^{j\beta a\cos\phi'} . \tag{2-14}$$

The attenuation factor $A(s',s)$ from (1-20) will now be calculated. From Figure 2-5, $\rho_g = a$, $\rho_e = \infty$, and the vectors $\hat{s}'$,$\hat{s}$, and $\hat{n}_e$ are

$$\begin{aligned} \hat{s}' &= [0, -\cos\phi', -\sin\phi'] \\ \hat{s} &= [0, \cos\phi, \sin\phi] \\ \hat{n}_{e1} &= [0, -1, 0] \quad @Q_1 \\ \hat{n}_{e2} &= [0, 1, 0] \quad @Q_2 \end{aligned} \tag{2-15}$$



(a)



(b)

Figure 2-4: Geometry of disk for (a) the general case, and (b) diffraction point Q_1.

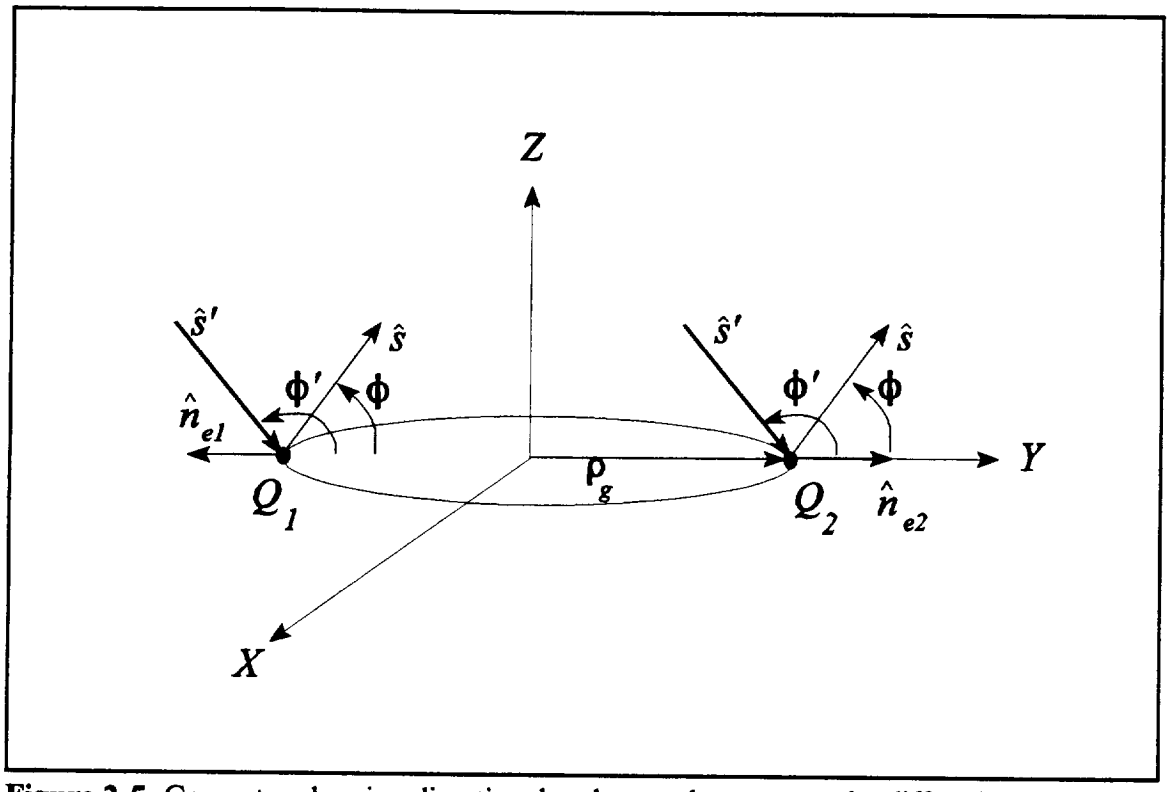


Figure 2-5: Geometry showing directional and normal vectors at the diffraction points.

Upon substituting (2-15) into (1-20),

$$A(s',s_1) = \sqrt{\frac{\rho_{c_1}}{s_1(\rho_{c_1}+s_1)}} \qquad \textbf{(2-16)}$$

where

$$\rho_{c_1} = -\frac{a}{\cos\phi' + \cos\phi} \quad . \tag{2-17}$$

For the diffraction point Q_2,

$$A(s', s_2) = \sqrt{\frac{\rho_{c_2}}{s_2(\rho_{c_2} + s_2)}} \tag{2-18}$$

where

$$\rho_{c_2} = \frac{a}{\cos\phi' + \cos\phi} \quad . \tag{2-19}$$

The phase function from the diffraction points Q_1 and Q_2 are simply $e^{-j\beta s1}$ and $e^{-j\beta s2}$, respectively, where s_1 and s_2 are given in (2-9) and (2-12).

Next the distance parameters L^i, L^{ro}, and L^{rn} for Q_1 and Q_2, which are used in (1-18) and (1-19), will be calculated. Recall from (1-16)

$$L^i = \frac{s(\rho_e^i + s)\rho_1^i \rho_2^i \sin^2\beta'_0}{\rho_e^i(\rho_1^i + s)(\rho_2^i + s)} \quad .$$

Since the incident wave is a uniform plane wave, $\rho^i_1 = \rho^i_2 = \rho^i_e = \infty$. Therefore (1-16) reduces to

$$L^i = \frac{s\rho_2^i \sin^2\beta'_0}{(\rho_2^i + s)} \quad . \tag{2-20}$$

Assuming $\rho^i_2 >> s$,

$$L^{\prime} = \frac{s\rho_2^{\prime}\sin^2\beta'_0}{\rho_2^{\prime}} \Rightarrow L^{\prime} = s\sin^2\beta'_o \,. \qquad \textbf{(2-21)}$$

For the disk-shaped problem at hand:

$$s = s_1 \quad for \quad Q_1 \,,$$

$$s = s_2 \quad for \quad Q_2 \,,$$

$$and \quad \beta'_o = \frac{\pi}{2} \,.$$

Therefore,

$$L_1^{\prime} \approx s_1 \quad and \quad L_2^{\prime} \approx s_2 \,. \qquad \textbf{(2-22)}$$

The radii of curvatures in the distance parameters L^{ro} and L^{rn} of (1-17) are calculated from the respective boundaries π-ϕ' and $[(2n-1)\pi-\phi']$ at the diffraction points Q_1 and Q_2. Summarizing the results

Q_1	Q_2
$\rho_r^1 = -\left[\frac{a}{2\cos\phi'}\right]$	$\rho_r^1 = \left[\frac{a}{2\cos\phi''}\right]$
$\rho_r^2 = \infty$	$\rho_r^2 = \infty$
$\rho_{ro}^r = \rho_r^1$	$\rho_{ro}^r = \rho_r^1$
$\rho_{rn}^r = \rho_{ro}^1$	$\rho_{rn}^r = \rho_{ro}^1$

Inputting the above information into (1-17) for the condition $\beta'_o = \pi/2$ shows

$$L^{ro} = L^{rm} \approx s \tag{2-23}$$

for both Q_1 and Q_2.

As mentioned in the theoretical section, care must be taken with respect to the how the appropriate angles are to be taken. In referencing Figure 2-4a, note that

$$\psi_1 = \phi_1 \qquad 0 \le \phi_1 < 2\pi$$

$$\psi_2 = \begin{cases} \pi - \phi_2 & 0 \le \phi_2 \le \pi \\ 3\pi - \phi_2 & \pi < \phi_2 < 2\pi \end{cases}$$

Therefore, in (1-18) and (1-19), g^- and g^+ must be replaced with

$$g^- = \psi - \psi' \; ; \quad g^+ = \psi + \psi' \tag{2-24}$$

Using the calculated quantities from (2-5)-(2-24), the total scattered electric field from the disk is

$$E^s_{total} = E^{GO} + E^{sd}(Q_1) + E^{sd}(Q_2) \tag{2-25}$$

where

$$E^{GO} = \bar{E}^{refl} = -\hat{a}_x E_o e^{j\beta\rho[\cos(\phi+\phi')]} \qquad [\phi_1 \text{ and } \phi_2] \le (\pi - \phi') \,,$$

$$E^{sd}(Q_1) = E^i(Q_1)\, D_S(L;\psi_1,\psi'_1;n,\beta'_o) A(s',s_1)\, e^{-j\beta s_1} \tag{2-26}$$

$$\Rightarrow E^{sd}(Q_1) = -E_o e^{-j\beta a\cos\phi'}\left[\frac{e^{-j\frac{\pi}{4}}}{2n\sqrt{2\pi\beta}\sin\beta'_0}\left\{\cot\left[\frac{\pi+(\psi_1-\psi'_1)}{2n}\right]F[\beta L\, {}^{i}g^{+}(\psi_1-\psi'_1)] + \right.\right.$$

$$\cot\left[\frac{\pi-(\psi_1-\psi'_1)}{2n}\right]F[\beta L\, {}^{i}g^{-}(\psi_1-\psi'_1)] - \left(\cot\left[\frac{\pi+(\psi_1+\psi'_1)}{2n}\right]F[\beta L^{ro}g^{+}(\psi_1+\psi'_1)] + \right.$$

$$\left.\left.\left.\cot\left[\frac{\pi-(\psi_1+\psi'_1)}{2n}\right]F[\beta L^{rn}g^{-}(\psi_1+\psi'_1)]\right)\right\}\right]\sqrt{\frac{\rho_{c_1}}{s_1(\rho_{c_1}+s_1)}}\ \sqrt{\lambda}\, e^{-j\beta s_1}\ .$$

and

$$E^{sd}(Q_2) = E^{i}(Q_2)\, D_S(L;\psi_2,\psi'_2;n,\beta'_o)A(s',s_2)\, e^{-j\beta s_2} \qquad \text{(2-27)}$$

$$\Rightarrow E^{sd}(Q_2) = -E_o e^{j\beta a\cos\phi'}\left[\frac{e^{-j\frac{\pi}{4}}}{2n\sqrt{2\pi\beta}\sin\beta'_0}\left\{\cot\left[\frac{\pi+(\psi_2-\psi'_2)}{2n}\right]F[\beta L\, {}^{i}g^{+}(\psi_2-\psi'_2)] + \right.\right.$$

$$\cot\left[\frac{\pi-(\psi_2-\psi'_2)}{2n}\right]F[\beta L\, {}^{i}g^{-}(\psi_2-\psi'_2)] - \left(\cot\left[\frac{\pi+(\psi_2+\psi'_2)}{2n}\right]F[\beta L^{ro}g^{+}(\psi_2+\psi'_2)] + \right.$$

$$\left.\left.\left.\cot\left[\frac{\pi-(\psi_2+\psi'_2)}{2n}\right]F[\beta L^{rn}g^{-}(\psi_2+\psi'_2)]\right)\right\}\right]\sqrt{\frac{\rho_{c_2}}{s_2(\rho_{c_2}+s_2)}}\ \sqrt{\lambda}\, e^{-j\beta s_2}\ .$$

A "C"-Program (listed in Appendix I) was created in order to calculate (2-25). The individual contributions of the geometrical optics and diffraction fields are shown in Figure 2-6. The total electric scattering fields of a bistatic scattering analysis are shown in Figure 2-7. It is interesting to note that the total geometrical optics field only occurs at the specular point. Based on this work,

geometrical optics fails to predict the correct bistatic scattered field when a plane wave is incident on a flat plate. The infinite radii of curvature of the incident field causes the equations to breakdown. The observations made here confirm statements made in [1, pp. 756 and 805] about restrictions to the incident wave and the geometry of the scatterer. Note the behavior of the diffraction fields in Figure 2-6. They take on the characteristics in Figure 1-5, of the theoretical section, for an observation distance far away from the scatterer.

Other diffraction types exist but were not accounted for in this investigation. One source of diffracted fields is from creeping waves which travel around the outer rim of the disk and shed diffracted rays tangentially as the surface wave propagates around the perimeter of the disk. The procedure for obtaining this diffracted field contribution is discussed in [6].

If the electric field is obliquely incident at angles less than 15 degrees from the center axis, equivalent currents need to be computed in order to correct for an axial caustic that occurs due to the convergence of an infinite number of diffracted rays from the disk rim. At the incident angle of 120 degrees, this effect does not appear to effect the results in Figure 2-7.

To summarize the flat plate study, using GO and UTD to solve for the electric field in a bistatic analysis results in two very localized (concentrated) groups of rays about the specular angle and the incident boundary of the model. Geometrical optics fails to predict the correct bistatic scattered field when a plane wave is incident on a flat plate. Due to this outcome, a three dimensional analysis is not necessary.

Figure 2-6: Reflection and Diffraction Fields From Circular Disk - Soft Polarization $\phi' = 120$ degrees

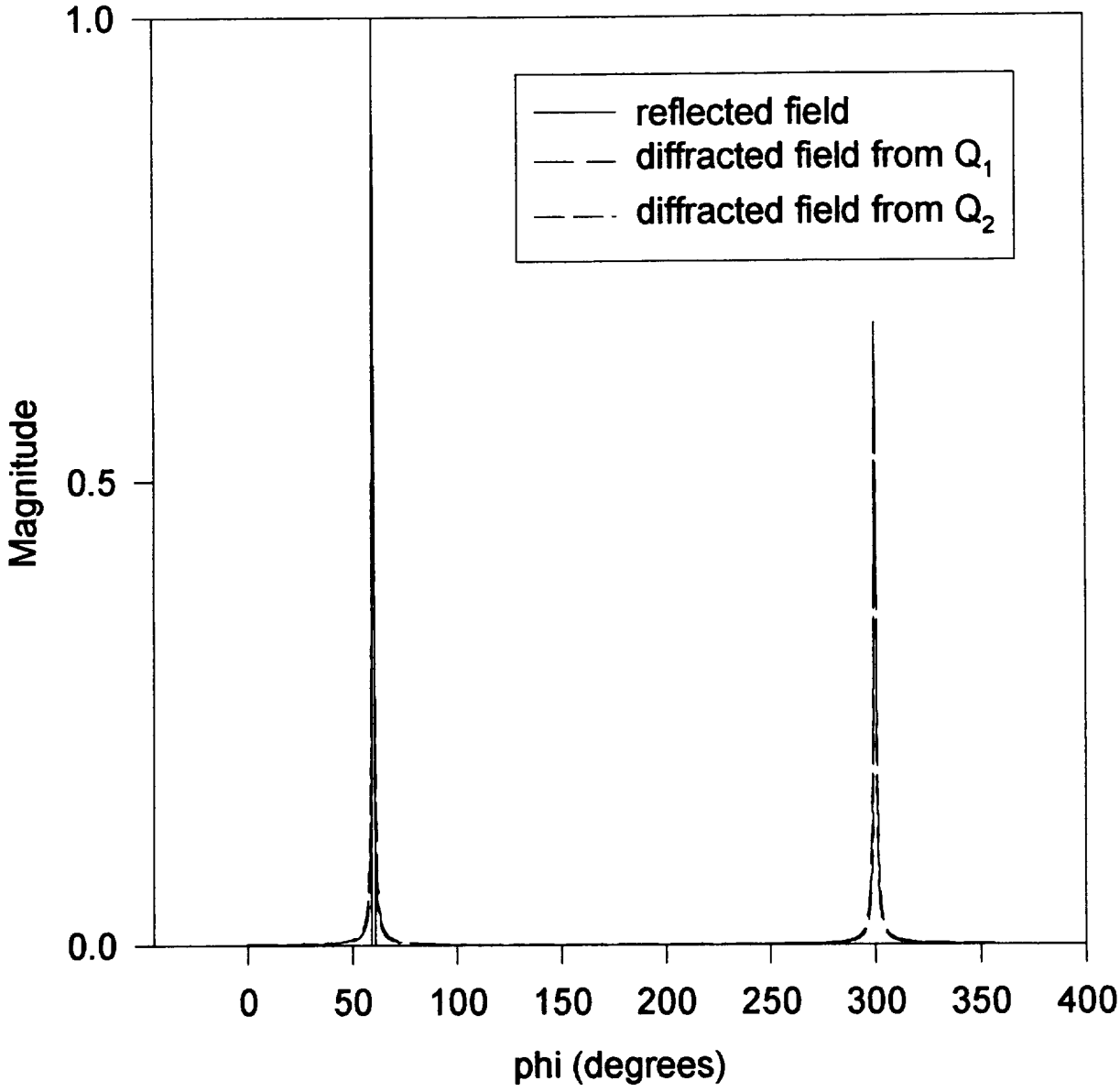
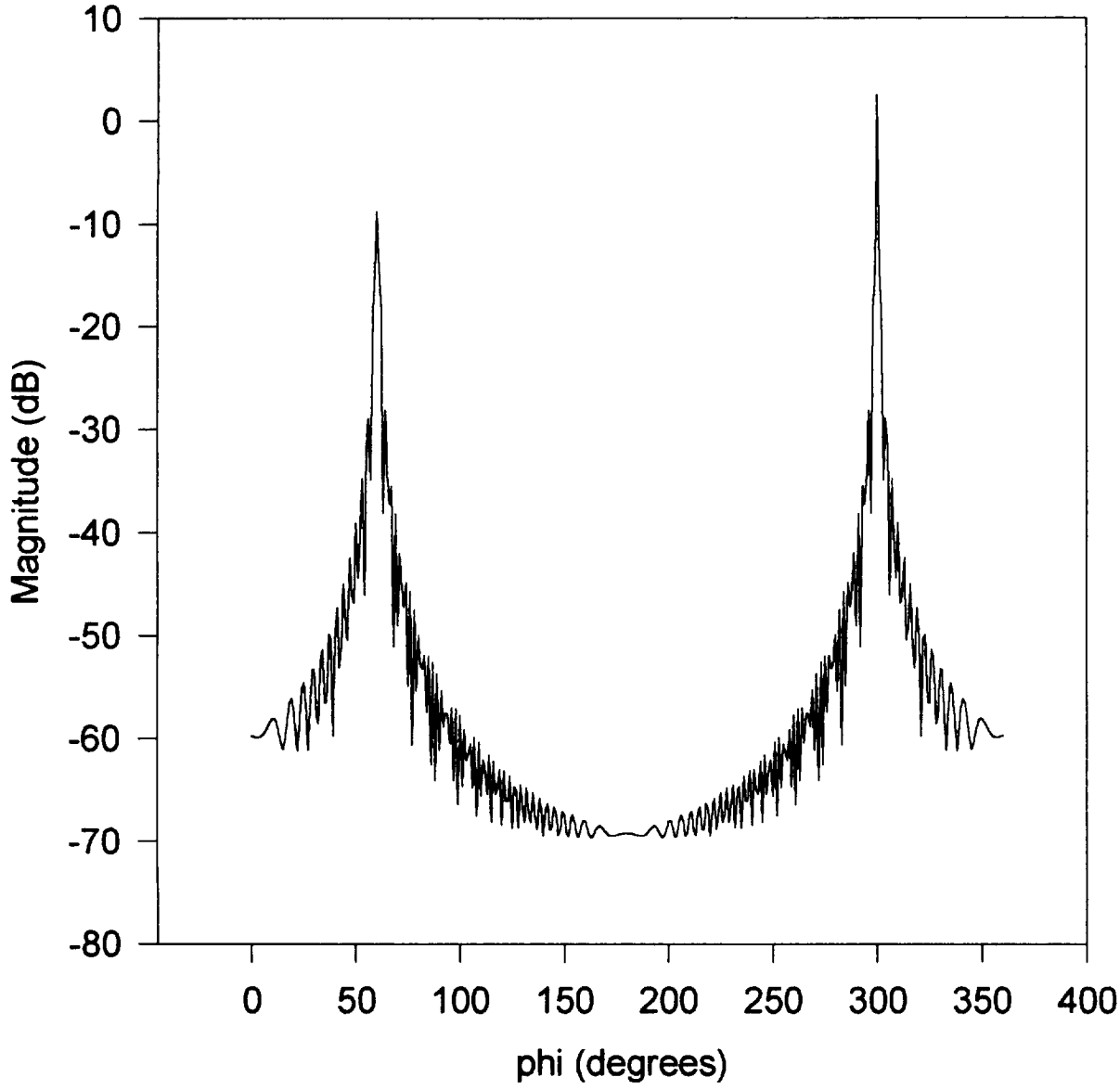


Figure 2-7: Total Bistatic Scattered Field
Plane Wave Incident on Disk - Soft Polarization
$\phi' = 120$ degrees



Part B - Normal Incidence on a Circular Cylinder

An incident angle of normal incidence has initially been chosen due to simplicity. The consequences of oblique incidence will be discussed later. A two dimensional model will first be analyzed and if appropriate a three dimensional model will be constructed.

A two dimensional circular cylinder of radius 2.5 meters is shown in Figure 2-8. It is struck with a uniform electric field plane wave for soft polarization such that

$$E_z^i = E_o e^{-j\beta\rho\cos\phi} \quad . \tag{2-28}$$

Due to Fermat's principal of reflection, the GO scattered field is comprised of only the reflected field in the illumination region. Recall, in the theoretical section it was explained that the surface waves attenuate rapidly and therefore have negligible contribution in the illumination region for large cylinders. In the shadow region, electromagnetic fields are solely due to surface diffraction from reflection points having a incident field at grazing. From [4, pp. 83] the reflected field is

$$E_z^r = \sqrt{\frac{a}{2\rho}\sin|\phi|}\; e^{-j\beta\left(\rho - 2a\sin\frac{\phi}{2}\right)} \tag{2-29}$$

$$0 < \phi < \pi \qquad \beta\rho \to \infty \quad (\textit{far-field}) \quad .$$

Upon writing the reflection field in the form of the radii of curvature as in [4], it can be shown that

$$E_z^r(\rho,\phi) = -E_z^i(a,\phi_o)\sqrt{\frac{a\cos\phi'}{a\cos\phi' + 2s}}\; e^{-j\beta s} \quad . \tag{2-30}$$

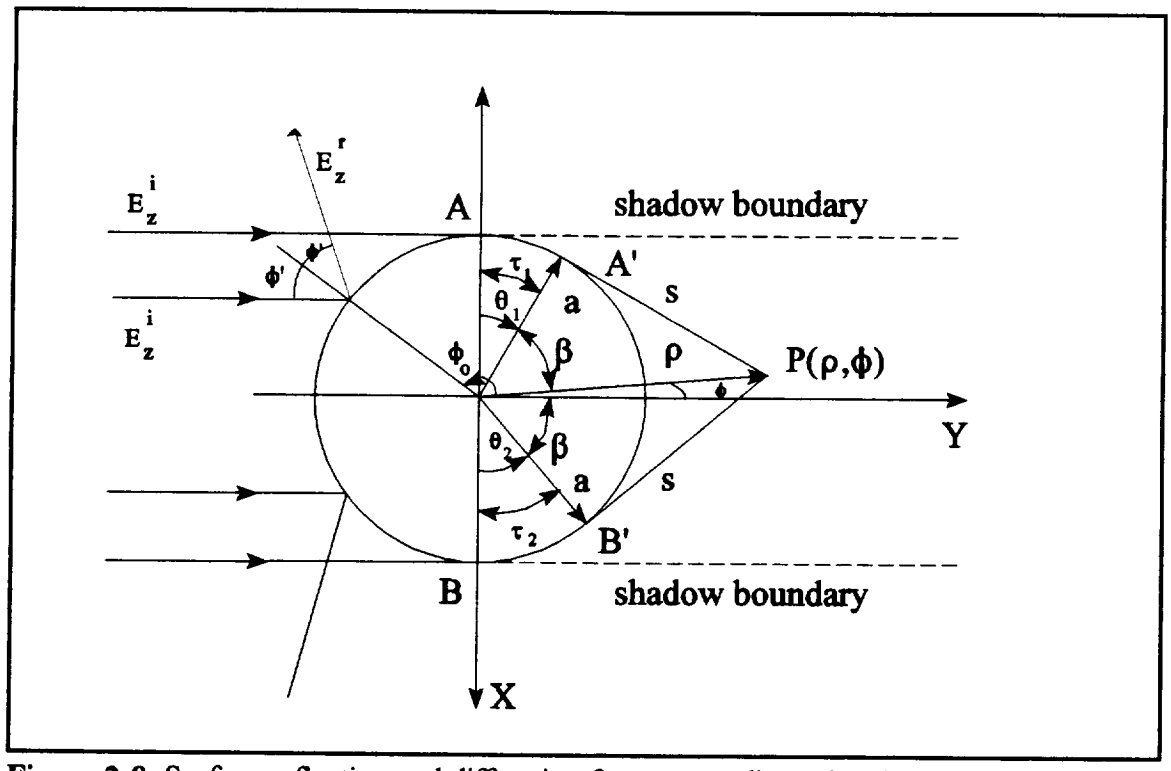


Figure 2-8: Surface reflection and diffraction from a two dimensional cylinder.

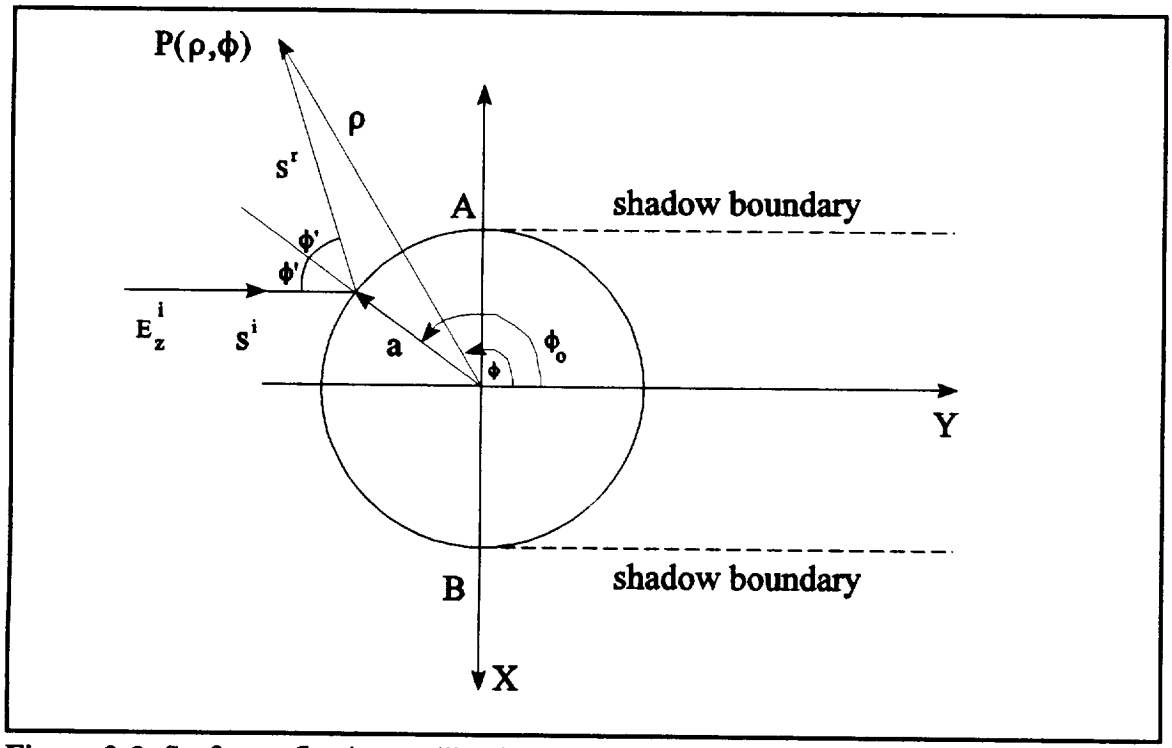


Figure 2-9: Surface reflection on illumination side of a circular cylinder.

In order to calculate the reflected field at observation point $P(\rho,\phi)$, it is necessary to find the angle ϕ' on the surface of the cylinder (on the illumination side) with respect to ϕ that gives the specular reflection at $P(\rho,\phi)$. This is illustrated in Figure 2-9. This search routine was incorporated in the "C" program of the cylinder problem (Appendix II).

As mentioned earlier, the diffraction fields originate from reflection points with grazing incident field. Surface waves account for the diffracted fields in the shadow region. For greater continuity between the illumination and shadow regions, an effective reflection boundary can be setup in order to create a transition region about the shadow boundary. The various regions are shown in Figure 2-10. Special Airy functions are needed for this transition region. These are based on asymptotic solutions of the exact canonical problem. From [4], the diffracted field for soft polarization can be written as

$$E^{d}(s) = D\bar{E}(A)\frac{e^{-j\beta(\tau+s)}}{\sqrt{s}} \qquad \textbf{(2-31)}$$

where

$$D = -\sqrt{\frac{2}{j\beta}}M\left[\tilde{\rho}_{-}(x,y)\right],$$

$$M = \left(\frac{\beta a}{2} \right)^{\frac{1}{3}}$$

$$x = M \frac{\tau}{a}$$

$$y = \frac{1}{M} \sqrt{\frac{\beta s}{2}} ,$$

and

$$\tilde{\rho}_{-}(x,y) = \rho(x) - y\,sgn(x) K_{-}(y|x|)\, e^{j\frac{\pi}{4}} .$$

The modified Pekeris function is

$$\rho(x) = \hat{\rho}(x) + \frac{1}{2\sqrt{(\pi)x}} \tag{2-32}$$

where $\rho(x)$ is called the Pekeris function and is defined as

$$\hat{\rho}(x) = \frac{e^{j\frac{\pi}{6}}}{2\sqrt{\pi}} \sum_{n=1}^{\infty} \frac{e^{\alpha_n x e^{-j\frac{5\pi}{6}}}}{\left[A_i'(-\alpha_n)\right]^2} \qquad x>0 . \tag{2-33}$$

The zeros α_n of the Airy function and the associated values are tabulated in [4]. For large negative x the Pekeris function is given as

$$\hat{\rho}(x) \approx \frac{\sqrt{(-x)}}{2} e^{j\left(\frac{x^3}{12}+\frac{\pi}{4}\right)} \quad x \to -\infty \ . \tag{2-34}$$

When the argument x is zero,

$$p(0) = 0.354\, e^{\frac{j\pi}{6}}$$

In (2-31), K_(x) is the modified Fresnel integral and is given in [4, pp. 20] as

$$K_{-}(x) \approx \frac{1}{2} \frac{e^{\left\{-j\left(\tan^{-1}\left[x^2+1.5x+1\right]-\frac{\pi}{4}\right)\right\}}}{\sqrt{\pi x^2+x+1}} \quad x \geq 0 \tag{2-35}$$

Expressions for the modified Pekeris and modified Fresnel functions were obtained from a steepest descent of the exact eigenfunction solution of a cylinder. Then, an asymptotic expansion of the residue series part of the solution is of the form of the Airy function. A special Airy function relationship is used to ensure convergence of the integral expression for the total electric field in [4, pp. 83].

In order to have a diffracted field on the reflection boundary in transition region one, the diffraction point must be on the illumination side of the cylinder. This does not obey the modified Fermat's principal, but is to be viewed simply as a geometric path to assist in calculation of the modified Pekeris function when x is negative.

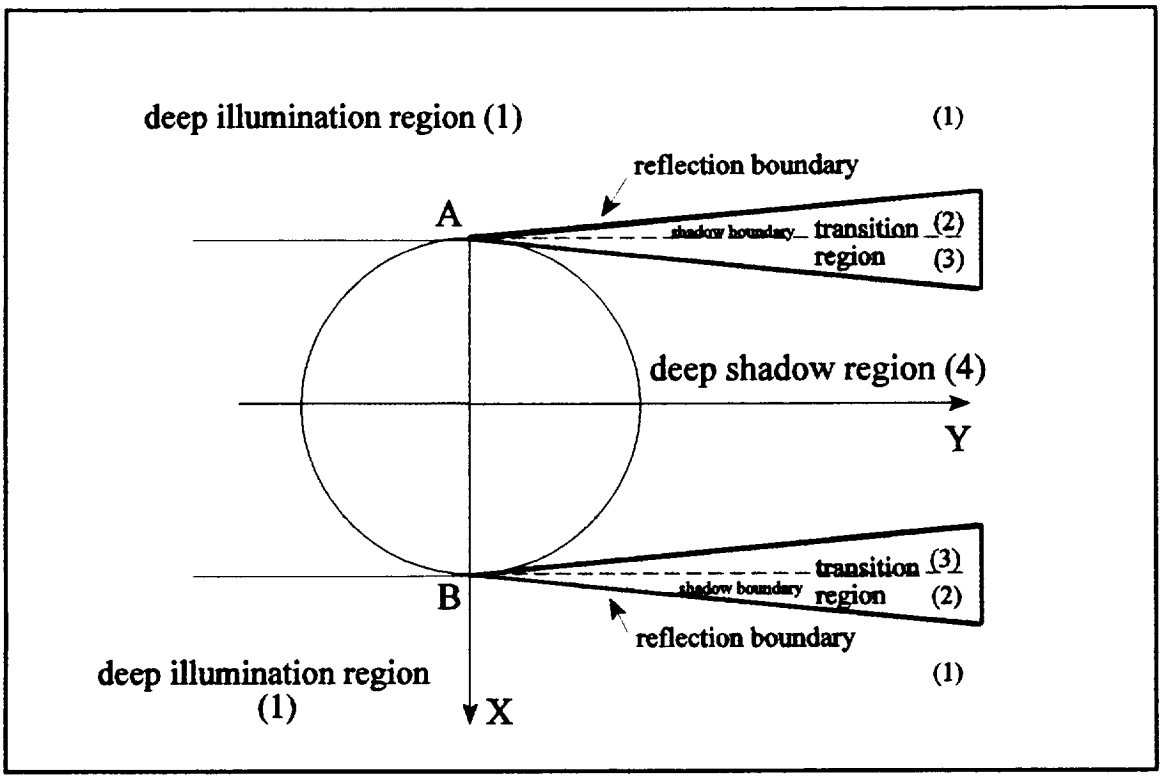
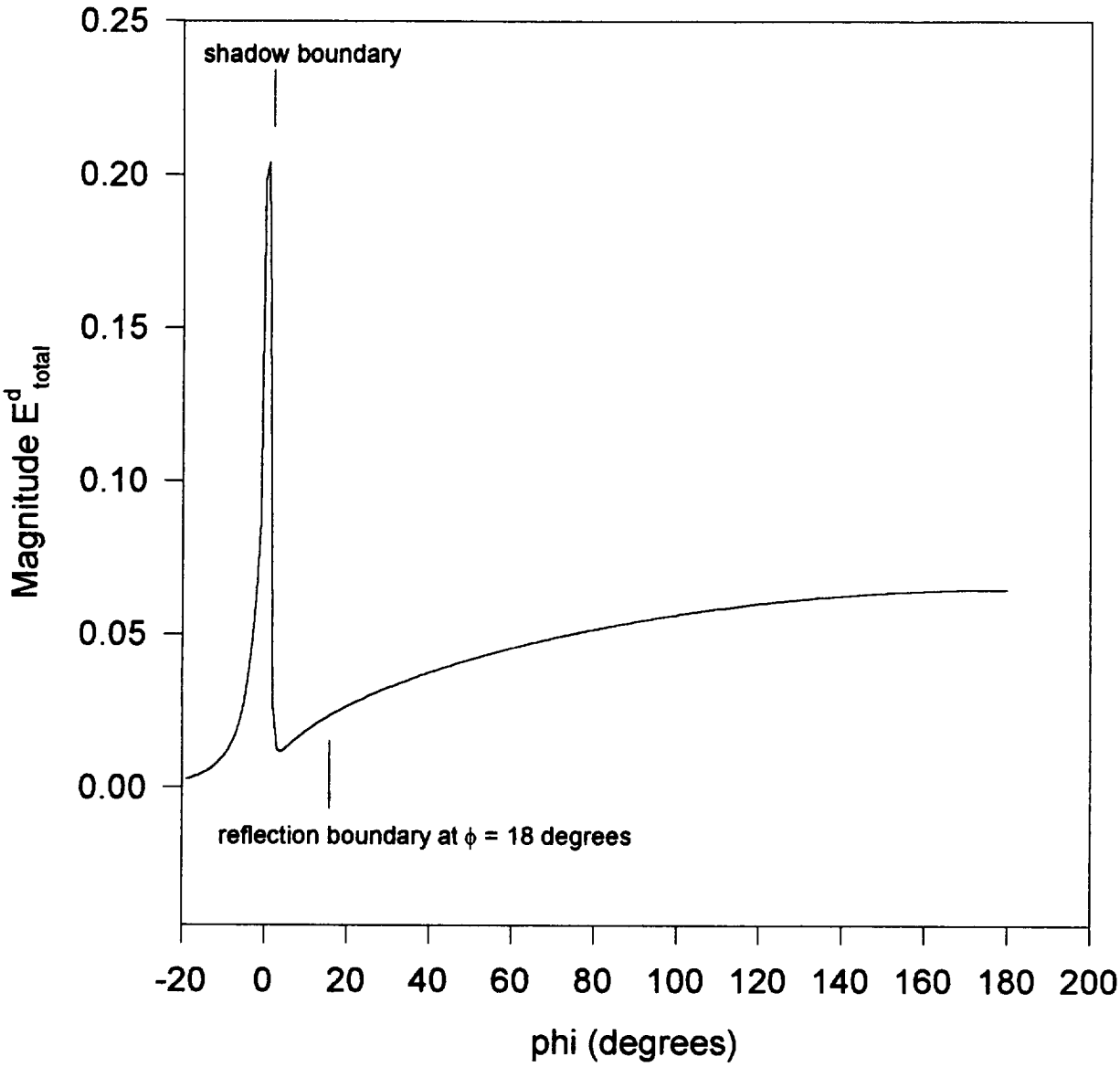


Figure 2-10: Transition region about the shadow boundary of a two dimensional perfectly conducting circular cylinder.

The combined electric scattered field from the two dimensional circular cylinder is shown in Figure 2-11. The field in the illumination region is only due to the reflected field and the scattered field in the shadow region is solely due to diffraction. These results were compared with the method of moments for a two dimensional circular cylinder under the same conditions and parameters. The GO and UTD solution failed to adequately model the scattering behavior.

Figure 2-11: Total Scattered Electric Field From Upper Half of Cylinder - Soft Polarization $\phi' = 180$ degrees



One explanation for the discrepancy between the method of moments solution and the one obtained in this work is: the Pekeris function is constructed for the total field. Since the scattered field does not include the incident field, this produces an error in the predicted pattern. The spike at the shadow boundary is due to the fact that the incident field is not present in the calculations which would compensate or (cancel) the discontinuity. The equations for negative argument of the Pekeris function did produce a smooth transition across the reflection boundary, however. This is also shown in Figure 2-11.

If the incident field was obliquely incident on the cylinder, the surface waves would follow a geodesic in the form of a helical path, as shown in Figure 2-12. The only modification that would be required is in (2-31), where a is replaced with the radii of curvature of the helical path. In this respect, Figure 2-11 can be viewed simply as a cross section of a part of the helical path and not a cylinder [4].

Due to the inadequateness of the GO and UTD solution using (2-30) and (2-31) for a two dimensional circular cylinder, a three dimensional analysis will not be performed.

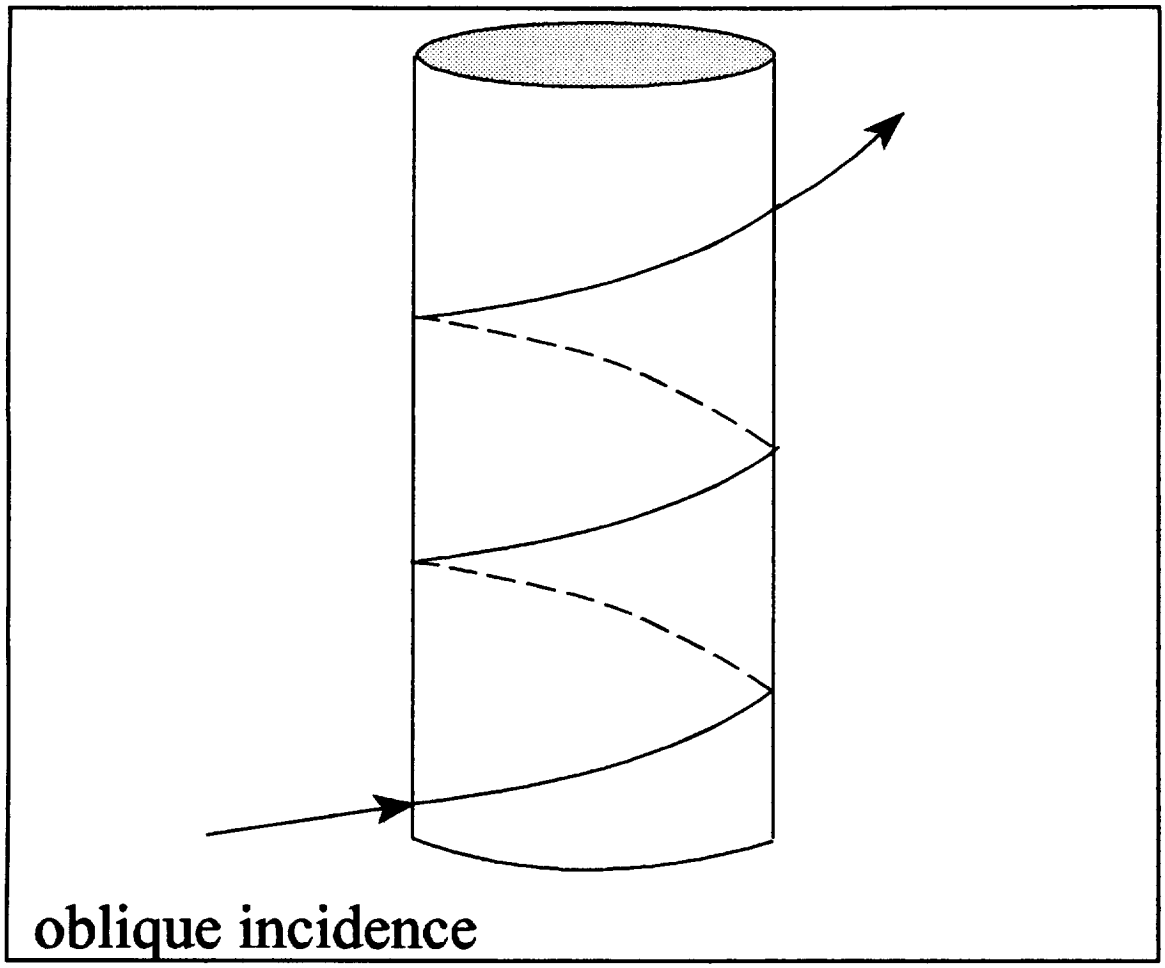


Figure 2-12: Path of surface wave on cylinder for oblique incidence.

Part C - Oblique Incidence on a Ring

The analysis of this object will be very brief due to the outcomes of the inadequate solutions from the circular disk and cylinder. A plane wave obliquely incident on the ring will have one primary diffraction point. Like the circular disk, higher-order diffraction terms exist, but will not be investigated here.

It is assumed that the off-axis diffraction point contributions are negligible or will cancel each other out. Modeling the ring with one diffraction point on the principal plane is the same as that of Figure 1-6. It has been the author's experience that the curved edge diffraction fields are very similar to the straight edge diffraction fields when ρ is sufficiently far removed from the diffraction point. Therefore, one would expect a diffraction pattern similar to the combination of Figures 5a and 5b.

Since a three dimensional problem has been ruled unobtainable using the present GO and GTD formulas for the circular disk and cylinder, it will not be attempted here.

CONCLUSION

In this report, plane wave scattering from a finite circular cylinder was investigated using the geometrical optics and the geometrical theory of diffraction. In the literature, GO and GTD are used extensively in monostatic radar cross-section analyses., however no information on bistatic scattering using these methods could be found. Based on the results presented, GO fails to predict the correct bistatic scattering field when a plane wave is incident on either a flat plate or cylinder. As stated previously, these results are in agreement with statements made in [1] about restrictions to the incident wave and the geometry of the scatterer.

For the cylinder, it was learned that the equations used for obtaining the scattered fields were

based on equations of the total electric field about a cylinder. As a result, GTD could not appropriately satisfy the discontinuity at the shadow boundary. In addition, the equations used were based on asymptotic expansions of the exact canonical solution for a cylinder, and tend not to converge properly at the shadow boundary. The Pekeris equations are for large argument rather than small arguments. This illustrates how extensive GTD is used in solving radar cross-section problems.

As stated earlier, with some regret, a three dimensional analysis was not possible. However, a better insight into the strengths and limitations of GO and GTD has been attained..

REFERENCES

[1] C.A. Balanis, *Advanced Engineering Electromagnetics*. New York: John Wiley & Sons, 1989.

[2] P.H. Pathak, "High-Frequency Techniques for Antenna Analysis," *Proc. IEEE*, vol. 80, pp. 44-65, Jan. 1992.

[3] R. Mittra, *Topics in Applied Physics: Numerical and Asymptotic Techniques in Electromagnetics*. New York: Springer-Verlag. 1975.

[4] G.L. James, *Geometrical Theory of Diffraction for Electromagnetic Waves, 3rd ed.* England: Peter Peregrinus Ltd. 1986.

[5] R.G. Kouyoumjian and P.H. Pathak, "A Uniform Geometrical Theory of Diffraction for an Edge in a Perfectly Conducting Surface," *Proc. IEEE*, vol. 62, no. 11, pp. 1448-1460, Nov. 1974.

[6] D.P. Marsland, C.A. Balanis, and S.A. Brumley, "High Order Diffractions from a Circular Disk," *IEEE Trans. Antennas and Propagat.* vol. AP-35, no. 12. pp. 1436-1444, 1987.

APPENDIX I

The code in this appendix has been written in C, using the Borland turbo C compiler. This program will compute and plot the electric field quantity of a uniform plane wave onto a disk-shapped scatterer. The code is "Project Grouped" together with the following files: GTDCURVE.C, EREFLECT.C, AC_ATTEN.C, DIFFCOEF.C, FRESNEL.C, PHASE.C, and CPLXMATH.C

```
/*                    GTDCURVE.C

************************************************************************
  This is the main trunk of the GTD program for computing and
  plotting the electric field quantity of a uniform plane wave
  onto a disk-shapped surface.  The observation distance
  is r meters.  THIS IS FOR TWO EDGES AND SOFT POLARIZATION


  This program includes reflection and diffraciton fields.
                        Bistatic Scattering Case
************************************************************************/


/* Header Files */

#include <stdio.h>
#include <math.h>
#include "cplxmath.h"

/* Prototype Functions */
struct Complex Ereflect(double a,double th,double th_i, double beta);
struct Complex A_c(double s, double pc);
struct Complex dif_coef(double r, double th_ray, double thp_ray, double beta0_inc,
                        double n, double Lro, double Lrn, int edge);
struct Complex phase_fn(double beta, double s, int);

/* External Variable Declaration */
/*  These variables originate in DIFFCOEF.C */
struct Complex dc_soft, dc_hard, dc_i, dc_r;


/*----------------Beginning of Main----------------------*/

main()
{
struct Complex E_diff[2][361], E_obs[361], E_refl[361],as,tempc,
                                    E0_surf, phase, dif_term;
double th_i, r, a, freq, c, pi,lambda, beta, beta0_inc, n, s_pri,
            s, thp_ray, Lro, Lrn, l, pc, th_ray, norm_obs, temp, conv_dtr,
            conv_rtd, th_neso,th_nesn, tol, temp2, thr, phir, phi_ray;

int th, phi, edge;
FILE *fp_out;

/* Initializing vectors */

for(th=0; th<361; th++)
{
E_refl[th].real=0.;
E_refl[th].imag=0.;
E_diff[0][th].real=0.;
E_diff[0][th].imag=0.;
```

```
E_diff[1][th].real=0.;
E_diff[1][th].imag=0.;
E_obs[th].real=0.;
E_obs[th].imag=0.;
}
tol=1.e-6;
clrscr();    /* clear screen */

                                        /* r is the observation distance (in meters) from center
                                                                of scatter. */
r= 300.;

/*----------------Scatterer Specifications----------------*/

a=2.5;      /*radius of disk in meters. */

/*----------------End of Scatterer Specifications----------------*/


/*----------Specifications of Incident Electric Field----------*/

freq=1.57542e+9;  /*operating frequency*/

                                        /* Direction of Propagation of Incident Wave        */
th_i= 120.;

/*--------End of Specifications of Incident Electric Field--------*/

pi=4.*atan(1.);
c = 2.99795638e+8;
lambda = c/freq;
beta = 2.*pi/lambda;
conv_dtr=pi/180.;
conv_rtd=180./pi;

/*--------Beginning of Calculations to determine the reflected
                        and diffracted electric fields at the observation point.---*/


/* Calculation of the diffracted electric field from scatterer.      */
/* The reflected field is also calculated in this routine.  */

/* ray coordinate values */
n=2.;
beta0_inc=90.;


edge=1;      /* flag for curved edge diffraction */


/* Edge 1 calculations */

thp_ray=th_i;

for(phi=0; phi<361; phi++)
{
        phir=((double)phi)*conv_dtr;
        /* s is the distance from diffraction point to obs. pt.*/
        s=sqrt((r*r)+(a*a)+(2.*a*r*cos(phir)));
        phi_ray=(s*s+a*a-r*r)/(2.*a*s);
        if(fabs(fabs(phi_ray)-1.) < tol)/* Argument in acos(x) must be  -1<= x <=1. */
        {
                if(phi_ray > 0.) phi_ray= 1.;
                if(phi_ray < 0.) phi_ray= -1.;
        }

        phi_ray=acos(phi_ray); /* in radians*/
        if(phi>180) phi_ray=2.*pi-phi_ray;
```

```
                    th_ray=phi_ray*conv_rtd;
                    if(th_ray <= 180.-th_i) E_refl[phi]=Ereflect(r,phir,th_i,beta);
                    E_refl[phi].real*=-1.;
                    E_refl[phi].imag*=-1.;
                    E0_surf.real=cos(beta*a*cos(th_i*conv_dtr));
                    E0_surf.imag=-sin(beta*a*cos(th_i*conv_dtr));
                    Lro=s/lambda;
                    Lrn=s/lambda;
                    l=s/lambda;
                    pc=-a/(cos(th_i*conv_dtr)+cos(phi_ray));
                    as=A_c(s, pc); /*Attenuation Parameter*/
                    dif_term=dif_coef(l,th_ray,thp_ray,beta0_inc,n,Lro,Lrn,edge);
                    dif_term.real*=sqrt(lambda); /* unnormalized diffraction coef.*/
                    dif_term.imag*=sqrt(lambda);
                    dc_i.real*=sqrt(lambda);
                    dc_i.imag*=sqrt(lambda);
                    dc_r.real*=sqrt(lambda);
                    dc_r.imag*=sqrt(lambda);
                    phase=phase_fn(beta, s, 0);
                    E_diff[0][phi]=Multiply(E0_surf,dif_term);
                    E_diff[0][phi]=Multiply(E_diff[0][phi],as);
                    E_diff[0][phi]=Multiply(E_diff[0][phi],phase);
                    dc_i=Multiply(dc_i,as);
                    dc_r=Multiply(dc_r,as);
                    dc_i=Multiply(dc_i,phase);
                    dc_r=Multiply(dc_r,phase);
}

/*Edge 2 calculations*/

for(phi=0; phi<361; phi++)
{
        phir=((double)phi)*conv_dtr;
        /* s is the distance from diffraction point to obs. pt.*/
        s=sqrt((r*r)+(a*a)-(2.*a*r*cos(phir)));
        phi_ray=(r*r-s*s-a*a)/(2.*a*s);
        if(fabs(fabs(phi_ray)-1.) < tol)/* Argument in acos(x) must be  -1<= x <=1. */
        {
                if(phi_ray > 0.) phi_ray= 1.;
                if(phi_ray < 0.) phi_ray= -1.;
        }

        phi_ray=acos(phi_ray); /* in radians*/
        if(phi>180) phi_ray=2.*pi-phi_ray;
        if(phi_ray <= pi) th_ray=180.-phi_ray*conv_rtd;
        if(phi_ray > pi) th_ray=540.-(phi_ray*conv_rtd);
        thp_ray=180.-th_i;
        if((phi_ray*conv_rtd) < (180.-th_i))
        {
                E_refl[phi].real=0.0;
                E_refl[phi].imag=0.0;
        }
        E0_surf.real= cos(beta*a*cos(th_i*conv_dtr));
        E0_surf.imag= sin(beta*a*cos(th_i*conv_dtr));
        Lro=s/lambda;
        Lrn=s/lambda;
        l=s/lambda;
        pc=a/(cos(th_i*conv_dtr)+cos(phi_ray));
        as=A_c(s, pc); /*Attenuation Parameter*/
        dif_term=dif_coef(l,th_ray,thp_ray,beta0_inc,n,Lro,Lrn,edge);
        dif_term.real*=sqrt(lambda);
        dif_term.imag*=sqrt(lambda);
        dc_i.real*=sqrt(lambda);
        dc_i.imag*=sqrt(lambda);
        dc_r.real*=sqrt(lambda);
        dc_r.imag*=sqrt(lambda);
        phase=phase_fn(beta, s, 0);
```

```
            E_diff[1][phi]=Multiply(E0_surf,dif_term);
            E_diff[1][phi]=Multiply(E_diff[1][phi],as);
            E_diff[1][phi]=Multiply(E_diff[1][phi],phase);
            dc_i=Multiply(dc_i,as);
            dc_r=Multiply(dc_r,as);
            dc_i=Multiply(dc_i,phase);
            dc_r=Multiply(dc_r,phase);
}

for(th=0;th<361;th++) /*Observation Efield Diffraction fields.*/
{
            E_obs[th]=Add(E_refl[th],E_diff[0][th]);
            E_obs[th]=Add(E_obs[th],E_diff[1][th]);
}
if((fp_out=fopen("efields.dat","wt"))==NULL) {
                        puts("cannot open efields.out file\n");
                        exit(1);
                        }

for(th=0;th<361;th++)
                        {
                        tempc=Add(E_diff[0][th],E_diff[1][th]);
                        fprintf(fp_out,"%d  %e %e %e %e\n", th, Magnitude(E_refl[th]),
                                                Magnitude(E_diff[0][th]), Magnitude(E_diff[1][th]),
                                                Magnitude(tempc));
                        }
fclose(fp_out);

if((fp_out=fopen("eobsdb.dat","wt"))==NULL) {
                        puts("cannot open eobsdb.dat file\n");
                        exit(1);
                        }

for(th=0;th<361;th++)
                        {
                        temp= Magnitude(E_obs[th]);
                        temp2=20.*log10(temp);
                        fprintf(fp_out,"%d  %f %f\n", th, temp, temp2);
                        }

fclose(fp_out);

return(0);
}


/*                      EREFLECT.C

*************************************************************************
  This routine will compute the direct and reflected components of the
  electric field plane wave.  (See Balanis page 811.)
*************************************************************************/

#include <math.h>
#include "cplxmath.h"

struct Complex Ereflect(double r,double theta,double theta_p,double beta)
{
struct Complex green_fn, E_go;
double pi, conv_dtr;

pi=4.*atan(1.);
conv_dtr=pi/180.;
theta*=conv_dtr;
theta_p*=conv_dtr;

green_fn.real=cos(beta*r*cos(theta+theta_p));
```

```
green_fn.imag=sin(beta*r*cos(theta+theta_p));

return(green_fn);
}
/*                      AC_ATTEN.C

************************************************************
 This program will calculate the attenuation factor A
 for curved edge diffraction.
 (See Balanis page 819, Equation 13-100.)
************************************************************/


#include <math.h>
#include "cplxmath.h"

struct Complex A_c(double s, double pc)
{
struct Complex aresult;
double a;

        a=pc/(s*(pc+s));
        if(a>0.)
        {
                aresult.real=sqrt(a);
                aresult.imag=0.;
        }
        else
        {
                aresult.real=0.;
                aresult.imag=sqrt(-a);
        }
return(aresult);
}


/*                  DIFFCOEF.C

*********************************************************************
 This routine computes the uniform diffraction coefficients


 This program was converted from FORTRAN (found at end of Chapter 13
 in Balanis) By Francis W. Forest -  1994


 The necessary modifications were made so that straight or edge
 diffraction coefficients can be calculated
*********************************************************************/

#include <stdio.h>
#include <math.h>
#include "cplxmath.h"

struct Complex ftf(double X);

/* Declaration of external variables*/
extern struct Complex dc_soft, dc_hard, dc_i, dc_r;


struct Complex dif_coef(double R,double phi,double phip,double beta_inc,
                                                        double n,double Lro,double Lrn,int edge)
{
struct Complex ct, ft[4], d[4], term, ftf_ret;
double pi, utpi, conv_dtr, sin_beta, dkl, ufn, betaR, N, dN_fract,
        integer, dN, sml_tol, tol, ang, A, X, Y;
```

```
int sign, i;
FILE *fp_out;

/* setting constants */

pi=4.*atan(1.);
ct.real=-.05626977;
ct.imag= .05626977;
utpi = 159.15494309e-3;
sml_tol=.001;
tol=1.e-8;
conv_dtr=pi/180.;


phi *=conv_dtr;
phip *=conv_dtr;
sin_beta=sin(beta_inc*conv_dtr);

dc_soft.real=0.0;
dc_soft.imag=0.0;
dc_hard.real=0.0;
dc_hard.imag=0.0;

if(fabs(n-0.5)<sml_tol) return(dc_soft,dc_hard);
dkl=2.*pi*R;
ufn=1./n;
betaR=phi-phip;
term.real=ct.real/(n*sin_beta);
term.imag=ct.imag/(n*sin_beta);
sign=1;
i=0;
while(i <= 3)
{
        if(i==2 && edge==1) dkl=2.*pi*Lro;
        if(i==3 && edge==1) dkl=2.*pi*Lm;
        dN=ufn*(0.5*sign+utpi*betaR);

/*************************************************************************
        This part was added in order to round "dN" up or down.  The numerical
        accuracy of dN was not allowing N to be the proper integer value.*/
        dN_fract=modf(dN,&integer);
        if(dN < 0.)
        {
                if(fabs(0.5-dN_fract) <= tol) dN= integer-0.5;
                if(fabs(1.0-dN_fract) <= tol) dN= integer-1.0;
        }
        if(dN >=0.)
        {
                if(fabs(0.5-dN_fract) <= tol) dN= integer+0.5;
                if(fabs(1.0-dN_fract) <= tol) dN= integer+1.0;
        }
/*************************************************************************/

        if(dN < tol) N=dN-0.5;
        if(dN >= tol) N=dN+0.5;
        modf(N, &integer);
        N=integer;
        ang=2.*pi*n*N-betaR;
        A=2.*cos(0.5*ang)*cos(0.5*ang);
        X=dkl*A;
        Y=pi+sign*betaR;

        if(fabs(X) >= 1.e-10)
        {
                ftf_ret=ftf(X);

                ft[i].real=ftf_ret.real/tan(0.5*Y*ufn);
                ft[i].imag=ftf_ret.imag/tan(0.5*Y*ufn);
```

```
		}
		else
		{
			ft[i].real=0.0;
			ft[i].imag=0.0;
			if(fabs(cos(0.5*Y*ufn)) >= 1.e-3)
			{
				if(Y < 0.)
				{
					ft[i].real= -1.7725*sqrt(dkl);
					ft[i].imag= -1.7725*sqrt(dkl);
				}
				if(Y > 0.)
				{
					ft[i].real= 1.7725*sqrt(dkl);
					ft[i].imag= 1.7725*sqrt(dkl);
				}
				ft[i].real=ft[i].real*n;
				ft[i].imag=(ft[i].imag-(2.*dkl*(pi-sign*ang)))*n;
			}
		}
		sign=-sign;
		d[i]=Multiply(term,ft[i]);
		if(i>= 1) betaR=phi+phip;
		++i;
}
dc_i=Add(d[0],d[1]);
dc_r=Add(d[2],d[3]);
dc_soft=Subtract(dc_i,dc_r);
dc_hard=Add(dc_i,dc_r);
return(dc_soft);
}


/*			FRESNEL.C

*************************************************************************
 This program computes the Fresnel transition function
 (Converted from FORTRAN code found at the end of Chapter 13
 in Balanis, Advanced Engineering Electromagnetics.
*************************************************************************/

#include <math.h>
#include "cplxmath.h"

struct Complex ftf(double XF)
{
struct Complex fx[8], fxx[8], Ftf, Ftf_temp, exp_term;
double pi, XX[8], X;
int n;

/* Constants used for linear interpolation */

pi=4.*atan(1.);

XX[0]=0.3;
XX[1]=0.5;
XX[2]=0.7;
XX[3]=1.0;
XX[4]=1.5;
XX[5]=2.3;
XX[6]=4.0;
XX[7]=5.5;

fx[0].real=0.5729;
fx[0].imag=0.2677;
fx[1].real=0.6768;
fx[1].imag=0.2682;
```

```
fx[2].real=0.7439;
fx[2].imag=0.2549;
fx[3].real=0.8095;
fx[3].imag=0.2322;
fx[4].real=0.8730;
fx[4].imag=0.1982;
fx[5].real=0.9240;
fx[5].imag=0.1577;
fx[6].real=0.9658;
fx[6].imag=0.1073;
fx[7].real=0.9797;
fx[7].imag=0.0828;

fxx[0].real=0.0;
fxx[0].imag=0.0;
fxx[1].real=0.5195;
fxx[1].imag=0.0025;
fxx[2].real=0.3355;
fxx[2].imag=-.0665;
fxx[3].real=0.2187;
fxx[3].imag=-.0757;
fxx[4].real=0.1270;
fxx[4].imag=-.0680;
fxx[5].real=0.0638;
fxx[5].imag=-.0506;
fxx[6].real=0.0246;
fxx[6].imag=-.0296;
fxx[7].real=0.0093;
fxx[7].imag=-.0163;

exp_term.real=0.70710678;
exp_term.imag=0.70710678;
X=fabs(XF);
if(X > 5.5)
{
        Ptf.real=1.+(75./(16.*X*X)-0.75)*1./(X*X);
        Ptf.imag=(0.5-15./(8.*X*X))*1./X;
}
else
{
        if(X >= 0.3)
        {
                n=1;
                while( (X >= XX[n]) && (n<=6)) n++;

                Ptf.real=fxx[n].real*(X-XX[n])+fx[n].real;
                Ptf.imag=fxx[n].imag*(X-XX[n])+fx[n].imag;

        }
        else
        {
                Ptf.real=sqrt(pi*X)-2.*X*exp_term.real;
                Ptf.imag=-2.*X*exp_term.imag;
                Ptf_temp.real=2./3.*X*X*exp_term.real;
                Ptf_temp.imag=-2./3.*X*X*exp_term.imag;
                Ptf=Subtract(Ptf,Ptf_temp);
                Ptf=Multiply(Ptf,exp_term);
                Ptf_temp.real=cos(X);
                Ptf_temp.imag=sin(X);
                Ptf=Multiply(Ptf,Ptf_temp);
        }
}
if(XF < 0) Ptf.imag*=(-1.0);
return(Ptf);
}

/*              PHASE_FN.C
```

```
*************************************************************************
 This routine will calculate the phase factor of the respective wave.
*************************************************************************/

/* flag is a marker that specifies e^(+j*beta*s) or e^(-j*beta*s) */

#include <math.h>
#include "cplxmath.h"

struct Complex phase_fn(double beta0, double s,int flag)
{

struct Complex phase;

phase.real=cos(beta0*s);
phase.imag=-sin(beta0*s);
if(flag==1) phase.imag=sin(beta0*s);

return(phase);
}


/*                      CPLXMATH.C                                                */
#include <math.h>
#include "cplxmath.h"

/*************************************************************************/
/*                                                                       */
/*    Jeff Swart        October, 1992          Turbo C++ 3.0    */
/*                                                                       */
/* Basic Complex math functions.  Prototypes are located in CPLXMATH.H.  */
/*                                                                       */
/*************************************************************************/

/*************************************************************************/
/* 'RadToDeg' will convert a value given in radians as a double to the   */
/* equivalent value in degrees, and return the result as a double.       */
/*************************************************************************/


/*************************************************************************/
/* 'DivComplexCart' will divide two Complex numbers given in Cartesian   */
/* RENAMED TO Divide:  J. richie, 2/93                                   */
/* form as structures of type 'cart_Complex', and return the result in   */
/* Cartesian form as a structure of type 'cart_Complex'.                 */
/*************************************************************************/

struct Complex Divide(struct Complex num,
                                        struct Complex denom)
{
 /************************************************************************/
 /* Variable declarations                                                */
 /************************************************************************/

 struct Complex                 /* Polar form of numerator       */
                                        /* Polar form of denominator     */
                        result    ;   /* Polar form of result          */


 double mag_1,mag_2,angle_1,angle_2;

 /************************************************************************/
 /* Code                                                                 */
 /************************************************************************/

 /* Convert numerator and denominator to polar form                      */

 mag_1=Magnitude(num);
```

```
 mag_2=Magnitude(denom);
 angle_1=atan2(num.imag,num.real);
 angle_2=atan2(denom.imag,denom.real);

 /* Calculate polar form of result                    */
 mag_1=mag_1/mag_2;
 angle_1=angle_1-angle_2;
 result.real=mag_1*cos(angle_1);
 result.imag=mag_1*sin(angle_1);


 /* Return Cartesian form of result                   */
 return result;

 } /* End function 'Divide'                           */


/* ------------------------------------
          ADDED BY J. RICHIE, JANUARY 1993
   ------------------------------------

 Function to obtain product of two Complex numbers    */


struct Complex Multiply(struct Complex num1,
                                        struct Complex num2)
 {
 /* Variable declarations                              */

 struct Complex result;

 /* Code                                               */
 result.real=num1.real*num2.real-num1.imag*num2.imag;
 result.imag=num1.real*num2.imag+num1.imag*num2.real;

 return result;

 }

/* Function to obtain sum of two Complex numbers       */

struct Complex Add(struct Complex add1,
                                        struct Complex add2)
 {
 /* Variable declarations                              */

 struct Complex result;

 /* Code                                               */
 result.real=add1.real+add2.real;
 result.imag=add1.imag+add2.imag;

 return result;

 }

/* Function to obtain difference of two Complex numbers */

struct Complex Subtract(struct Complex subp,
                                        struct Complex subm)
 {
 /* Variable declarations                              */

 struct Complex result;

 /* Code                                               */
 result.real=subp.real-subm.real;
 result.imag=subp.imag-subm.imag;
```

```
 return result;

 }

/* Function to return the magnitude of a Complex number                    */

double Magnitude(struct Complex z)
{
double mag_value;

mag_value=sqrt( (z.real*z.real) + (z.imag*z.imag) );
return mag_value;
}

/* Function to return the Complex conjugate of a Complex number   */

struct Complex Conjugate(struct Complex z)
{
struct Complex conjug;

conjug.real=z.real;
conjug.imag=(-1)*z.imag;
return conjug;
}
```

APPENDIX II

The code in this Appendix was used to
The code in this appendix has been written in C, using the Borland turbo C compiler. This program will computeand plot the electric field quantity of a uniform plane wave onto a circular cylinder. This program is "Project Grouped" together with the following files: GTDCYLDR.C, PEKERIS.C, K_FN.C, PHASE.C, and CPLXMATH.C.

```
/*                    GTDCYLDR.C

**************************************************************************
 This is the main trunk of the GTD program for computing and
 plotting the electric field quantity of a uniform plane wave
 onto a circular cylinder. The observation distance
 is r meters.

 The diffracted fields are currently set for soft polarization.
**************************************************************************/


/* Header Files */

#include <stdio.h>
#include <math.h>
#include "cplxmath.h"

/* Prototype Functions */
double sfind(double a, double r, double phir);
struct Complex phase_fn(double beta, double s, int p_flag);
struct Complex pekeris(double x);
struct Complex K_fn(double x);

/* External Variable Declaration*/
double phi_p;

/*----------Beginning of Main----------*/

main()
{
struct Complex E_go[361], E_diff[2][361], E_obs[361], E0_surf, phase,
                                    Efactor, pvalue, kvalue, comp_tmp, pxy;

double r, a, freq, c, lambda, pi, beta, conv_dtr, conv_rtd, phir,
            s, atten, m, y, psi, th, tau, x, sgn_mth, phi_sgn, temp,
            norm_term;
int phi, d_inc;
FILE *fp_out;

pi=4.*atan(1.);
conv_dtr=pi/180.;
conv_rtd=180./pi;

r=300.; /* Observation distance from center of cylinder */
a=2.5; /* radius of cylinder */
freq=1.57542e+9;  /*operating frequency*/
c = 2.99795638e+8;
lambda = c/freq;
beta = 2.*pi/lambda;

clrscr();  /* clear output screen */

/* Initializing vectors */
```

```
for(phi=0; phi<361; phi++)
{
        E_go[phi].real=0.;
        E_go[phi].imag=0.;
        E_diff[0][phi].real=0.;
        E_diff[0][phi].imag=0.;
        E_diff[1][phi].real=0.;
        E_diff[1][phi].imag=0.;
        E_obs[phi].real=0.;
        E_obs[phi].imag=0.;
}


if((fp_out=fopen("cylinder.dat","wt"))==NULL)
        {
        puts("cannot open cylinder.dat file\n");
        exit(1);
        }

/*-------Beginning of Calculations to determine the reflected
                and diffracted electric fields at the observation point.---*/


/*Calculation of the reflected electric field from scatterer.      */
for(phi=4;phi<181;phi++)
{
        phir=(double)phi;
        if(phir > 90.) phir=-1.*(180.-phir);
        phir*=conv_dtr;
        if(phi < 180) s=sfind(a, r, phir);
        if(phi == 180)
        {
                s=r-a;
                phi_p=0.;
        }
        if(phir<0.) phir+=pi;
        phir*=conv_rtd;
        E0_surf.real=-cos(beta*a*cos(phi_p));
        E0_surf.imag=-sin(beta*a*cos(phi_p));
        phase=phase_fn(beta, s, 0);
        atten=(a*cos(phi_p))/(a*cos(phi_p)+2.*s);
        atten=sqrt(atten);
        E_go[phi]=Multiply(E0_surf,phase);
        E_go[phi].real*=atten;
        E_go[phi].imag*=atten;
        fprintf(fp_out,"%d %f %f %f %e\n",phi, phir, phi_p*conv_rtd,
                                                s, Magnitude(E_go[phi]));
}
fclose(fp_out);

/*************************************************************************/

/*Calculation of the diffracted electric field from scatterer.      */

if((fp_out=fopen("cyldiff.dat","wt"))==NULL)
        {
        puts("cannot open cyldiff.dat file\n");
        exit(1);
        }

s= r*r-a*a;
s=sqrt(s);

m=beta*a/2.;
m=pow(m,.33333333);
y=sqrt(beta*s/2.)*1./m;
psi=acos(a/r);  /* in radians */
```

```
/* Calculation of diffracted field on the shadow boundary */
phi_p=pi/2.; /* in radians */
E0_surf.real=-1.*cos(beta*a*cos(phi_p));
E0_surf.imag=-1.*sin(beta*a*cos(phi_p));
phase=phase_fn(beta, s, 0);
Efactor.real= 0.354/y*cos(pi/12.);
Efactor.imag=-0.354/y*sin(pi/12.);
E_diff[0][0]=Multiply(E0_surf,phase);
E_diff[0][0]=Multiply(E_diff[0][0],Efactor);

/* Calculation of the diffracted field in the shadow region */
phi_sgn=-1.;
for(d_inc=0;d_inc<2;d_inc++)
{
        for(phi=89;phi>-4;phi--)
        {
                phir=((double)phi)*conv_dtr*phi_sgn;
                if(phi==90)phir=89.522529*conv_dtr*phi_sgn;
                th=pi/2.-phir-psi;
                tau=th*a;
                x=m*th;
                pvalue=pekeris(x);
                kvalue=K_fn(y*fabs(x));
                sgn_mth=1.;
                if(m*th < 0.) sgn_mth=-1.;
                comp_tmp=Multiply(kvalue,phase_fn(pi/4.,1., 1));
                comp_tmp.real*=y*sgn_mth;
                comp_tmp.imag*=y*sgn_mth;
                pxy=Subtract(pvalue,comp_tmp);

                phase=phase_fn(beta, tau+s, 0);
                comp_tmp.real=(-1./sqrt(beta))*m;
                comp_tmp.imag=( 1./sqrt(beta))*m;
                E_diff[d_inc][90-phi]=Multiply(E0_surf,comp_tmp);
                E_diff[d_inc][90-phi]=Multiply(E_diff[d_inc][90-phi],pxy);
                E_diff[d_inc][90-phi]=Multiply(E_diff[d_inc][90-phi],phase);
                E_diff[d_inc][90-phi].real/=sqrt(s);
                E_diff[d_inc][90-phi].imag/=sqrt(s);

                fprintf(fp_out,"%d %f %e\n", d_inc, phir*conv_rtd,
                                        Magnitude(E_diff[d_inc][90-phi]));
        }
        phi_sgn=1.;
}

for(phi=4;phi<181;phi++)
{
        fprintf(fp_out,"%d %e\n",phi, Magnitude(E_go[phi]));
}
fclose(fp_out);

                if((fp_out=fopen("ecyldb.dat","w"))==NULL) {
                puts("cannot open ecyldb.dat file\n");
                exit(1);
                }


for(phi=1;phi<181;phi++)
                {
                temp= Magnitude(E_go[phi]);
                temp=20.*log10(temp);
                fprintf(fp_out,"%d %e\n", phi, temp);
                }
fclose(fp_out);
return(0);
}
```

```
/*********************************************************************
  Routine to find distance value 's' and angle phi_p from known
  observation distance rho and angle phi.
*********************************************************************/

double sfind(double a, double r, double phi)
{
double snum, pi, conv_dtr, sden, rtemp, s, rdiff, tol, smax;
int phi_i, i, flag;

tol=1.e-12;
pi=4.*atan(1.);
conv_dtr=pi/180.;

smax=sqrt(r*r-a*a);
s=-1.;
for(phi_i=1;phi_i<91;phi_i++)
{

        phi_p=(double)phi_i;
        phi_p*=conv_dtr;
        snum=-1.*(a*cos(phi_p)*sin(phi)+a*sin(phi_p)*cos(phi));
        sden=cos(2.*phi_p)*sin(phi)+sin(2.*phi_p)*cos(phi);
        if(fabs(snum) < tol && fabs(sden) < tol)
                /*perform L'Hopital's Rule*/
                {
                snum=a*(sin(phi_p)*sin(phi) - cos(phi_p)*cos(phi));
                sden=2.*(cos(2.*phi_p)*cos(phi) - sin(2.*phi_p)*sin(phi));
                s=snum/sden;
                }
        if(fabs(sden) > tol ) s=snum/sden;  /* Singularity - skip over */
        if( s>= 0. && (s<=smax))  /* 0.<= s <= (s=sqrt(r*r-a*a))*/
        {
         rtemp=a*a+s*s+2.*a*s*cos(-phi_p);
         rtemp=sqrt(rtemp);
         rdiff=r-rtemp;
         if( fabs(rdiff) <1.e-6) return(s);
         if(rdiff > 0.)
                {
                flag=0;
                phi_p-=1.*conv_dtr;
                i=0;
                while(flag==0)
                        {
                        i++;
                        phi_p+=1./pow(2.,i)*conv_dtr;
                        snum=-1.*(a*cos(phi_p)*tan(phi)+a*sin(phi_p));
                        sden=cos(2.*phi_p)*tan(phi)+sin(2.*phi_p);
                        if(fabs(sden) > tol) s=snum/sden;
                        /*if s is >0 but less than the desired value, the denominator
                                might go to zero.  To account for this, the following if
                                statement is used to increment the angle phi_p such that
                                s becomes greater than the desired value.  When this happens
                                the search routine resumes as normal.*/
                        if(fabs(sden) <= tol) phi_p+=1./pow(2.,i)*conv_dtr;
                        rtemp=a*a+s*s+2.*a*s*cos(-phi_p);
                        rtemp=sqrt(rtemp);
                        rdiff=r-rtemp;
                        if(rdiff > 0.) phi_p-=1./pow(2.,i)*conv_dtr;
                        if(fabs(rdiff) < 1.e-6) flag=1;
                        }

                return(s);
                }
        }

}
        printf("error in finding s  %d %f\n", phi_i, s);
```

```
        exit(1);
        return(0);

}

/*                      PEKERIS.C

*************************************************************************
 This program will calculate the Pekeris function.
*************************************************************************/


#include <math.h>
#include "cplxmath.h"

struct Complex pekeris(double x)
{

struct Complex exp_tmp, exp_term, psum, pout;
double pi, alpha[3], Ai_p[3], arg_cos, arg_sin;
int n;

pi=4.*atan(1.);
exp_tmp.real=-1.*cos(pi/6.)/(2.*sqrt(pi));
exp_tmp.imag=-1.*sin(pi/6.)/(2.*sqrt(pi));

alpha[1]=2.338;
alpha[2]=4.088;
Ai_p[1]=.701;
Ai_p[2]=-.803;
psum.real=0.;
psum.imag=0.;

for(n=1;n<3;n++)
{
        arg_cos=alpha[n]*x*cos(5.*pi/6.);
        arg_sin=alpha[n]*x*sin(5.*pi/6.);
        exp_term.real= cos(arg_sin)*(cosh(arg_cos)+sinh(arg_cos));
        exp_term.imag=-sin(arg_sin)*(cosh(arg_cos)+sinh(arg_cos));
        exp_term.real*=1./(Ai_p[n]*Ai_p[n]);
        exp_term.imag*=1./(Ai_p[n]*Ai_p[n]);
        psum=Add(psum,exp_term);
}

pout=Multiply(exp_tmp,psum);
return pout;
}


                                        K_FN.C
/*************************************************************************
 This program will calculate the Modified Fresnel function.
 (Based on EQN 2.40 in G.L. James - Geometrical Theory of Diffraction
 for Electromagnetic Waves, 3rd ed.)
*************************************************************************/


#include <math.h>
#include "cplxmath.h"

struct Complex K_fn(double x)
{

struct Complex exp_term, kout;
double pi, c, arg;

pi=4.*atan(1.);
```

```
c=2.*sqrt(pi*x*x+x+1.);
arg=atan(x*x+1.5*x+1.-pi/4.);

exp_term.real=cos(arg);
exp_term.imag=-sin(arg);

kout.real=exp_term.real/c;
kout.imag=exp_term.imag/c;

return kout;
}
```