RL-780

SCATTERING BY DISTRIBUTIONS OF SMALL THIN PARTICLES

by

David Allen Ksienski

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical Engineering)
in The University of Michigan
1984

ABSTRACT

SCATTERING BY DISTRIBUTIONS OF SMALL THIN PARTICLES

by

David Allen Ksienski

Chairman: Thomas B.A. Senior

The scattering of electromagnetic radiation by distributions of particles occurs in a variety of circumstances. At radio wave frequencies the operation of radar units is affected by rain and ice crystals suspended within clouds, while at higher, optical frequencies the amount of solar radiation reaching the earth's surface can be affected by pollutants in the upper atmosphere. This study is restricted to particles which are small compared to the wavelength of the illuminating electromagnetic radiation. The particles are assumed to be composed of a homogeneous lossy dielectric, with conductors characterized by large permittivities and lossy materials described by permittivities with relatively large imaginary components.

In the investigation of scattering by small particles, it is often convenient and useful to solve for the scattered field in terms of a low frequency expansion (a Taylor series in powers of the maximum dimension of the particle over the wavelength of the incident electromagnetic field). Unfortunately, the usual techniques for computing the low frequency expansion fail if the particle is collapsed to a plate with vanishing thickness. The difficulty arises from an unanswered problem in classical physics, the construction of a vector potential. A solution to this problem is obtained and presented in the context of low frequency scattering.

In many cases of low frequency scattering, only the first term of the expansion is necessary to adequately characterize the scattered field. This is obtained by solving a static field scattering problem. Unfortunately, for particles which are very thin (but of finite thickness), existing numerical codes become highly unstable. An algorithm is developed expressly for the thin plate scattering problem which appears accurate over a wide class of thin plates, permitting arbitrarily shaped plates with complex permittivities. The solution is obtained using a finite element method with linear basis functions over triangular elements. The results of the program include the calculation of the dipole moments associated with the plates, and the manner in which these dipole moments affect the electrical properties of the entire distribution is discussed.

ACKNOWLEDGEMENTS

The author wishes to express his gratitude for the guidance and expertise provided by the chairman, Professor Thomas B.A. Senior. The author also wishes to acknowledge the technical and emotional support of his colleagues at The University of Michigan. Special thanks are due Ms. Wanita Rasey and Ms. Jean Ringe for their skillful and expeditious typing of the manuscript.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I. INTRODUCTION

The interaction of electromagnetic radiation with distributions of particles is significant in many areas. For example, the manner in which radio waves scatter from water droplets produced by clouds and the ice crystals which may be suspended within these same clouds is important for the operation of radar units as well as the reception capabilities of radios and television sets. The interaction of higher frequency electromagnetic radiation with particle distributions is also important, and the scattering of visible light from contaminants produced by factories is a phenomenon which is often all too visible. For the purpose of analysis, the individual particles may be considered as composed of a lossy dielectric. Conductors are then characterized by large permittivities while lossy materials are characterized by permittivities with relatively large imaginary components. One class of interaction or scattering of electromagnetic waves from particles is low frequency scattering. Despite the name, a particular band of frequencies is not implied, but rather the ratio of the size of the particle to the wavelength of the electromagnetic radiation is restricted to be small. Although an exact range of ratios is not dictated, the degree to which the ratio is small determines the extent to which the low frequency approximations are valid.

When low frequency scattering is analyzed, the scattered electromagnetic radiation from a particle may be modeled by expressing the scattered field as a Taylor series expansion in powers of the maximum dimension of the particle over the wavelength of the incident electromagnetic field (Stevenson, 1954). In many cases, the ratio is sufficiently small to permit characterization of the scattered field by the first term in the series which is a zeroth order term and corresponds to solving a static field problem (infinite wavelength). The solution of the static field problem is invariably much easier than the solution of the general dynamic problem, and when the size of the particle is sufficiently small compared to the wavelength, it permits a compact description of the scattered field. Particles which are thin (one dimension smaller than the others) are important both as approximations which permit various simplifications in the analysis and as a region where existing numerical codes exhibit instabilities. Thin particles are also a common constituent of aerosols, with the relatively large surface area helping the particle remain suspended in the atmosphere. The numerical solution of the static field problem for bodies with axial symmetry has been presented in Senior and Willis (1982), and involves the evaluation of an integral over the surface of the body. Unfortunately, as the thickness of the body is decreased, numerical inaccuracies increase (Willis, 1982). The reformulation of this integral into a form which is extremely stable and accurate for thin plates is one of the contributions of this investigation.

Thin plates have been studied by Harrington and Mautz (1975) using the electric field integral equation and by Inspektorov (1982) using the magnetic field integral equation. Harrington and Mautz (1975) approximate the plates using a resistive sheet and thus ignore any effects resulting from a normal polarization of the sheet. Although Inspektorov (1982) considers finite thickness plates, his formulation becomes increasingly unstable as the plate thickness decreases. Both formulations consider the dynamic scattering problem and require the solution of a pair of coupled integral equations. For the problem of scattering by electrically small particles (i.e., the Rayleigh region), a simpler approach is possible. As shown by Keller et al (1972), the scattered field may be characterized by a polarization tensor. Further, if the object contains at least one axis of symmetry, at most three of the tensor elements are independent, corresponding to polarization along three perpendicular axes. The calculation of the dipole moments necessitates the previous calculation of the potential which may be obtained from a single scalar integral equation. This generally requires the solution of a potential problem (Senior, 1982), however this may be circumvented by using zero-degree harmonics (Senior and Ksienski, 1984). Unfortunately, when the scattering object is collapsed to a plate of infinitesimal thickness both of the techniques fail. Further, for thin plates (small but finite thickness), the solution of the potential problem develops numerical difficulties (Willis, 1982).

Distributions of particles are important for examining composite materials (e.g., Polder and Van Santen, 1946; and Bergman, 1978), as well as clouds of particles, and regular arrays of elements designed to provide an artificial dielectric for the purpose of microwave lenses (e.g., Kock, 1948). For particles and wavelengths such that the particle interaction is far field and the particle size and inter-particle distance are small compared to a wavelength, the distribution may be analyzed using the Clausius-Mosotti-Lorentz-Lorenz formulation (e.g., von Hippel, 1954). If the particles are spherical, the Clausius-Mosotti-Lorentz-Lorenz formulation may be reduced to a formulation due to Maxwell Garnett (1904), which has been experimentally verified for sparse distributions of particles (for which the formulation is rigorously correct) as well as for dense distributions of particles (Bohren and Battan, 1980).

This investigation is concerned with low frequency scattering from distributions of particles. The particles, as well as the inter-particle distance, are assumed small relative to the wavelength of the exciting field. Additionally, one dimension of the particle is assumed smaller than the other two dimensions, and in Chapter II the particles considered have zero thickness and are nothing more than a boundary condition on an open surface.

Chapter II discusses the problem of low frequency scattering from a single thin particle. Although low frequency scattering has been previously discussed for a solid body, the problem of scattering from an open surface defeats the standard techniques (e.g., Stevenson, 1954; Kleinman, 1965; and Senior, 1982). The solution of the problem involves the construction of a vector potential F, given $\nabla \times \bar{F} = \bar{f}$ which is a problem from classical physics. Solving problems which involve scattering from particles with zero thickness is not merely an academic exercise. Particles which are very thin may be modeled by open surfaces, and this generally permits some simplification in the analysis. For example, scattering from resistive sheets was considered by Harrington and Mautz (1975) and shown to be an effective model for thin (finite thickness) dielectric shells.

Chapters III and IV consider the problem of static scattering from a thin dielectric plate. The solution of this problem is of course necessary in the low frequency expansion. In contrast to Chapter II, the problem considered in Chapters III and IV is primarily numerical. Previous studies in static scattering from dielectric particles have encountered numerical difficulties when the objects became thin (e.g., Senior, 1975; Herrick, 1976; and Willis, 1982). This is believed to result from the singularity associated with the surface integral formulation. In Chapter III the surface integral formulation of the problem is recast using a volume integral which has a less singular kernel. The integro-differential equation is solved for the potential which is constrained to vary linearly along its smallest dimension. The finite element method (e.g., Zienkowicz, 1982) is employed using triangular elements with linear basis functions, which guarantee C_0 continuity. The result of the approach is a highly efficient, very stable matrix problem with all of the matrix elements evaluated accurately using an analytic evaluation of the volume integral. The results are consistent with expectations and the program is able to operate over a wide range of thicknesses. The dipole moments for several shapes are presented and these are compared to the results of a simple linear predictor model which is derived in Chapter IV.

Chapter V is concerned with sparse distributions of particles. These distributions of particles are analyzed in terms of the density and dipole moments of the constituent particles to arrive at an artificial dielectric description of the distribution. The primary focus of this chapter is the rigorous derivation of the Clausius-Mosotti-Lorentz-Lorenz equation (e.g., von Hippel, 1954), which describes the effective permittivity of the artificial dielectric. The chapter concludes with a comment on distributions of plates, and the accuracy of the method is verified for a dense cubical distribution of thin plates.

Finally, it must be noted that this dissertation reflects the author's preference of electric quantities over magnetic quantities. Thus, the use of an electric exciting field, electric potentials, dielectric constants, and electric dipole moments. In fact, exactly analogous magnetic problems can be solved by merely reading magnetic for electric, permeability for permittivity, etc., which in Chapter IV would then result in the calculation of magnetic dipole moments.

CHAPTER II. LOW FREQUENCY SCATTERING

2.1 The Low Frequency Expansion

The low frequency expansion involves an expansion of the scattered field in powers of the frequency. If the dimensions of the particle are small compared to the wavelength of the illuminating electromagnetic radiation then only a few terms of the expansion may be necessary to accurately characterize the scattered field. The numerical problems encountered with the low frequency expansion are generally simpler than those associated with the general dynamic problem. An additional benefit is that by expanding the solution explicitly in powers of frequency the scattered field becomes known for a range of frequencies, in contrast to the dynamic case where the problem must be resolved for each frequency desired.

The method of the low frequency expansion has been developed in Stevenson (1953), Kleinman (1965), and Senior (1982). However, if the body is collapsed to a surface with zero thickness (perhaps as a model of a thin metal sheet), mathematical difficulties are encountered. There are several reasons for using the zero thickness model, not the least of which is the fact that the mathematical problems encountered are interesting. For solid bodies which are thin, numerical difficulties often arise in attempting to evaluate the associated integrals over the opposing surfaces which are close together. Finally, by reducing the problem to a zero thickness plate it is hoped that simplifications would result in the analysis.

In performing a low frequency expansion, the first step is to expand all quantities in powers of frequency. Specifically, the electric and magnetic fields, $\bar{E}$ and $\bar{H}$, are expanded in powers of (ik), where k is related to the circular frequency ω by

$$k = \omega(\mu\varepsilon)^{1/2} \quad ;$$

μ and ε are the permeability and permittivity of free space and the time convention used is $e^{-i\omega t}$. The field quantities $\bar{E}$ and $\bar{H}$ may be expressed in terms of sources. Charge ρ and current $\bar{J}$ are sources which, since they arise from the scattering problem, are assumed to be constrained to the surface of the plate. These quantities are also expanded in powers of frequency,

$$\bar{E} = \sum_{m=o}^{\infty} \bar{E}_m(ik)^m \quad , \quad \bar{H} = \sum_{m=o}^{\infty} \bar{H}_m(ik)^m$$

$$\rho = \sum_{m=o}^{\infty} \rho_m(ik)^m \quad , \quad \bar{J} = \sum_{m=o}^{\infty} \bar{J}_m(ik)^m \quad .$$

When the expansions are inserted into Maxwell's equations

$$\nabla \times \bar{H} = \bar{J} + \frac{\partial \bar{D}}{\partial t}$$

$$\nabla \times \bar{E} = -\frac{\partial \bar{B}}{\partial t}$$

$$\nabla \cdot \bar{J} = -\frac{\partial \rho}{\partial t}$$

where $\bar{B} = \mu\bar{H}$ and $\bar{D} = \varepsilon\bar{E}$, the following relation is obtained inter alia:

$$Z_0 \nabla \times \bar{H}_1 = -\bar{E}_0 \quad . \tag{2.1}$$

The constant Z_0 is the free space impedance, $\bar{H}_1$ is the first order magnetic field, and $\bar{E}_0$ is the zeroth order, or static, electric field. Equation (2.1) also constitutes a problem which must be solved in the low frequency expansion. The problem is to determine $\bar{H}_1$ given $\bar{E}_0$. Since $\bar{E}_0$ is assumed known, it may be defined in terms of a scalar potential, ϕ, which is also assumed known. The precise context in which (2.1) arises varies with the different treatments, and the problem is not generally stated in terms of a first order magnetic field and a zeroth order electric field. However, the formulation in (2.1) does give the problem some physical significance. The body is assumed to have zero net charge, which is mathematically stated by requiring

$$\int_B \hat{n} \cdot \nabla\phi \, ds = 0 \quad .$$

The standard solution (Stevenson, 1954) for a solid body is then given by

$$Z_0\bar{H}_1 = -\nabla \times \int_B \hat{n}(\phi - \Phi)G \, ds'$$

where G is the free space static Green's function and Φ is the solution to the interior Neumann problem

$$\frac{\partial \Phi}{\partial n} = \frac{\partial \phi}{\partial n} .$$

The boundary value of $\partial\phi/\partial n$ is known since $\bar{E}_0$ and hence $\nabla\phi$ is known everywhere. The solution of the interior Neumann problem does involve the solution of an integral equation, which from a numerical perspective is time consuming. This problem may be circumvented through the use of zero-degree harmonics, as shown in Senior and Ksienski (1984). Unfortunately, both the standard solution via the Neumann problem and the solution via zero-degree harmonics break down if the body is collapsed to a zero thickness plate. The solution via zero-degree harmonics breaks down if the volume of the body vanishes because the evaluation of the kernel associated with the integral becomes ambiguous. The method of solving the problem by first solving the Neumann problem becomes impossible since the Neumann problem is a three dimensional problem which must be solved in the interior of the body which is only a two dimensional surface of discontinuity. Further, the symmetry of the problem would appear to indicate that both ϕ and Φ are continuous across the surface of the plate which forces $\bar{H}_1$ to be identically zero. As the solution of (2.1) for $\bar{H}_1$ is necessary for the continued development of a low frequency expansion, it is apparent that the expansion cannot be obtained for a zero thickness plate. A solution which is valid for the case of a zero thickness plate is the subject of the remainder of this chapter.

2.2 Scattering from a Plate

In obtaining an $\bar{H}_1$ which satisfies Eq. (2.1), an additional constraint which is applied is that $\bar{H}_1$ must be physically reasonable.

If $\bar{H}_1$ is expressed in terms of a current distribution, this requirement may be stated precisely by constraining the current to the scattering object and, in the case of the plate, by requiring that the current does not flow off the edge of the plate (i.e., the normal component of $\bar{J}_1$ on the edge of the plate must be zero). Obtaining a solution for $\bar{H}_1$ in terms of the current may be facilitated by solving

$$\nabla_s \cdot \bar{J}_1 = c\rho_0 \tag{2.2}$$

for $\bar{J}_1$, where

$$\rho_0 = -\frac{1}{Z_0} \left.\frac{\partial\phi}{\partial z}\right|_-^+$$

and c is the speed of light. Since the current $\bar{J}_1$ and charge ρ_0 are both surface distributions, the divergence of $\bar{J}_1$ is specified with the surface differential operator ∇_s. The plate is assumed to lie in the x-y plane, and the notation $\partial\phi/\partial z|_-^+$ denotes the discontinuity of $\partial\phi/\partial z$ across the surface of the plate. The solution of (2.2) for $\bar{J}_1$ produces a solution of (2.1) if $\bar{H}_1$ is defined in terms of the current distribution as

$$\bar{H}_1 = \nabla \times \int_B \bar{J}_1 G \, ds' \quad .$$

This may be shown by substituting this definition of $\bar{H}_1$ into (2.1) and taking the field point away from the plate,

$$\bar{E}_0 = -Z_0\nabla \times \bar{H}_1 = -Z_0\nabla \times \nabla \times \left(\int_B \bar{J}_1 G\, ds'\right)$$

$$= -Z_0\nabla\nabla \cdot \int_B \bar{J}_1 G\, ds' \ .$$

Bringing the differential operator inside the integral and converting the operator to the primed coordinate system yields

$$\bar{E}_0 = Z_0\nabla \int_B \bar{J}_1 \cdot \nabla' G\, ds'$$

$$= -Z_0\nabla \int_B G\nabla_s' \cdot \bar{J}_1\, ds' + Z_0\nabla \int_{C_B} (\bar{J}_1 G) \cdot \hat{\tau}'\, d\ell' \ ,$$

where $\hat{\tau}$ is the outward normal to the plate on the edge of the plate and lies in the plane of the plate and C_B denotes the contour surrounding the plate. The contribution of the second integral is zero if $\hat{\tau} \cdot \bar{J}_1$ is constrained to be zero as discussed above. Finally, using equation (2.2) reduces the above equation to

$$\bar{E}_0 = -\frac{1}{\varepsilon}\nabla \int_B G\rho_0\, ds' \ ,$$

which is the definition of $\bar{E}_0$ in terms of the charge distribution.

It should be noted that both the solution (2.1) for $\bar{H}_1$ and (2.2) for $\bar{J}_1$ allow considerable freedom. Since only the curl of $\bar{H}_1$ is specified, $\bar{H}_1$ is only unique to within the gradient of a scalar. Similarly, in (2.2), only the divergence of $\bar{J}_1$ is specified and thus $\bar{J}_1$ is known only to within the curl of a vector. As only a particular solution is required for (2.1) or (2.2), this ambiguity can be exploited to yield a solution which is easy to implement. Three solutions are proposed for (2.2), the first being the rather natural restriction of representing $\bar{J}_1$ as the gradient of a scalar.

If $\bar{J}_1$ is represented as

$$\bar{J}_1 = \nabla_s \psi$$

then from (2.2)

$$\nabla_s^2 \psi = c\rho_0$$

which is a two dimensional Poisson problem for ψ. Since the forcing function is $c\rho_0$ and ρ_0 tends to infinity near the edges of the plate, the Poisson problem will in general be difficult to solve. However, if the geometry of the plate is simple, for example a circular plate, then the charge distribution resulting from a uniform incident electrostatic field is known analytically. The Poisson problem may then be solved analytically, and the solution for a circular plate is given in Senior and Ksienski (1984). Unfortunately, for a general plate geometry the solution of a Poisson problem with an unbounded forcing function is not a problem which is well suited for numerical methods, and an alternate solution is needed.

If $\bar{J}_1$ is now restricted to $\bar{J}_1 = \hat{x}J_x$, assuming $\bar{E}_0^{inc} = -\nabla x$, then

$$\frac{\partial J_x}{\partial x} = c\rho_0$$

and

$$J_x = c \int \rho_0 \, dx \quad .$$

By choosing $\bar{J}_1$ to lie in the same direction as the incident electric field, a solution is obtained by simply integrating the charge distribution. The integral specified above proceeds along the surface of the plate in the x direction. If the plate is symmetric and convex, then the normal component of $\bar{J}_1$ will vanish along the edge. The requirement that the plate be symmetric and convex is a loose description of a geometrical constraint which can be stated precisely. Specifically, the plate must have two non-collinear axes of symmetry such that the intersection of the plate with any line parallel to either axis is simply connected. The satisfaction of this requirement will then permit the determination of the field scattered by the plate when it is illuminated by an arbitrary uniform electrostatic field. However, if the plate is not symmetric and convex (in the sense defined above), it is impossible to force the normal component of the current to vanish along the edge, even with an arbitrary function of y which may be added to the integral.

The third solution is intended as a correction to the preceding solution when the plate is not both symmetric and complex. The current is broken into two components,

$$\bar{J}_1 = \bar{J}^{(1)} + \bar{J}^{(2)}$$

with $\bar{J}^{(1)}$ chosen as above, and $\bar{J}^{(2)}$ represented as the gradient of a scalar. Specifically,

$$\bar{J}^{(1)} = J_x^{(1)}\hat{x}$$

$$\bar{J}^{(2)} = \nabla_s\psi$$

$$\nabla_s^2\psi = 0$$

with $\hat{\tau} \cdot \bar{J}_1 = 0$. The validity of $\bar{J}^{(1)}$ is thus preserved from the preceding solution, while $\bar{J}^{(2)}$ as a homogeneous solution to (2.2) is added to satisfy the boundary condition. The solution for $\bar{J}^{(2)}$ is obtained by solving a two dimensional Neumann problem, with boundary conditions which are finite since they arise from the normal component of $\bar{J}^{(1)}$along the edge of the plate. The normal component of $\bar{J}^{(1)}$ along the edge of the plate is finite since it is defined as

$$\bar{J}^{(1)} = \hat{x}c\int\rho_0\, dx \quad ,$$

and the singularities in ρ_0 are of the order $x^{-1/2}$. Finally, the two dimensional Neumann problem does have a solution since

$$\int_{C_B} \nabla_s\psi \cdot \hat{\tau}'\, d\ell' = - \int_{C_B} \bar{J}^{(1)} \cdot \hat{\tau}'\, d\ell'$$

(cont.)

$$= - \int_B \nabla_s \cdot \bar{J}^{(1)} \, ds'$$

$$= - \int_B c\rho_0 \, ds'$$

$$= 0$$

by the zero net charge condition.

This last solution is valid for any flat plate, and thus the low frequency expansion is now repaired. The solution which has been obtained was intended to be complementary to a solution for solid perfectly conducting bodies obtained by Senior (1982). This goal has been achieved (Senior, 1983), and the solution has also proven useful for the problem of scattering from a resistive plate (Senior and Naor, 1984), as well as providing a solution for a problem from classical physics, the construction of a vector potential (Senior and Ksienski, 1984).

The above formulation is necessary in analyzing the scattering by a thin plate via the low frequency expansion, and first and higher order terms in the expansion may now be obtained through the methods presented in this chapter. The low frequency expansion is valid when the dimensions of the particle are small compared to the wavelength of the illuminating radiation. To the extent that this requirement is satisfied, only a few terms in the low frequency expansion may be necessary to accurately characterize the scattered field. For a sufficiently small particle only the zeroth order term is necessary to accurately characterize the scattered field, and this is considered in the remainder of the dissertation.

CHAPTER III. STATIC SCATTERING FROM A DIELECTRIC PLATE

3.1 The Dipole Moment

In Chapter II, the problem considered was that of finding a first order magnetic field given a zeroth order electric field. The zeroth order fields are important not only as the first term in a low frequency expansion upon which the higher order terms depend, but also in their own right. The zeroth order fields determine the electric dipole moment $\bar{p}$ (and magnetic dipole moment $\bar{m}$), which often is all the information that is needed. For example, in sparse distributions of particles, discussed in Chapter V, the distribution will be characterized by the density of the distribution and the dipole moments of the individual particles. Although the zeroth order term and hence the dipole moment are most easily visualized in a static field, the concept of a low frequency analysis permits the use of a dielectric with complex permittivity which has physical significance in a dynamic field. Thus the dipole moment calculated for complex permittivities can provide much information about the far field scattered from a lossy dielectric particle, and for the analysis contained in Chapter V it is all the information that is required.

3.2 Static Scattering from a Thin Dielectric Plate

In the analysis of scattering from a thin flat dielectric plate the usual surface integral formulation (Senior, 1976)

$$\varphi^t = \frac{2}{1 + \tau} x_j + \frac{1 - \tau}{1 + \tau} 2 \int_B \varphi^t \frac{\partial G}{\partial n'} dx' \tag{3.1}$$

which is valid for any solid dielectric body with permittivity (permeability) τ becomes highly unstable. This difficulty is believed to arise from the highly singular kernel and the problem of correctly evaluating contributions from the integration on the edge of the plate. In a previous numerical investigation (Willis, 1982), the instability was found to be worst for very thin plates and for large permittivities. As the current investigation is directed at computing the scattered fields specifically, though not exclusively, for the case of plate thickness approaching zero, coupled with permittivities approaching infinity, it was felt that an alternate formulation was needed.

An integral equation was sought which involves integration only over a single flat surface. Although this was not obtained, an integro-differential equation was found which involves integration and differentiation over a single flat surface. The associated kernel has a very mild singularity, and this new integro-differential equation resulted in greater numerical stability than Eq. 3.1.

Without loss in generality, the plate is assumed to lie centered in the z = 0 plane. The plate is of thickness t, and the surface of the plate comprises the edge and the two parallel walls, as shown in Fig. 3.1. The general scattering problem may be solved by decomposing the incident electric field into $\hat{x}, \hat{y}$, and $\hat{z}$ components and then superposing the associated scattered fields. If $\bar{E}^{inc} = \hat{x}$, then (3.1) becomes

$$\phi^t = -\frac{2}{1+\tau}x + \frac{1-\tau}{1+\tau}2\left\{\int_e \phi^t \frac{\partial G}{\partial n'}ds' + \int_w \phi^t \frac{\partial G}{\partial n'}ds'\right\}$$

where e represents the edge and w represents the two parallel walls.

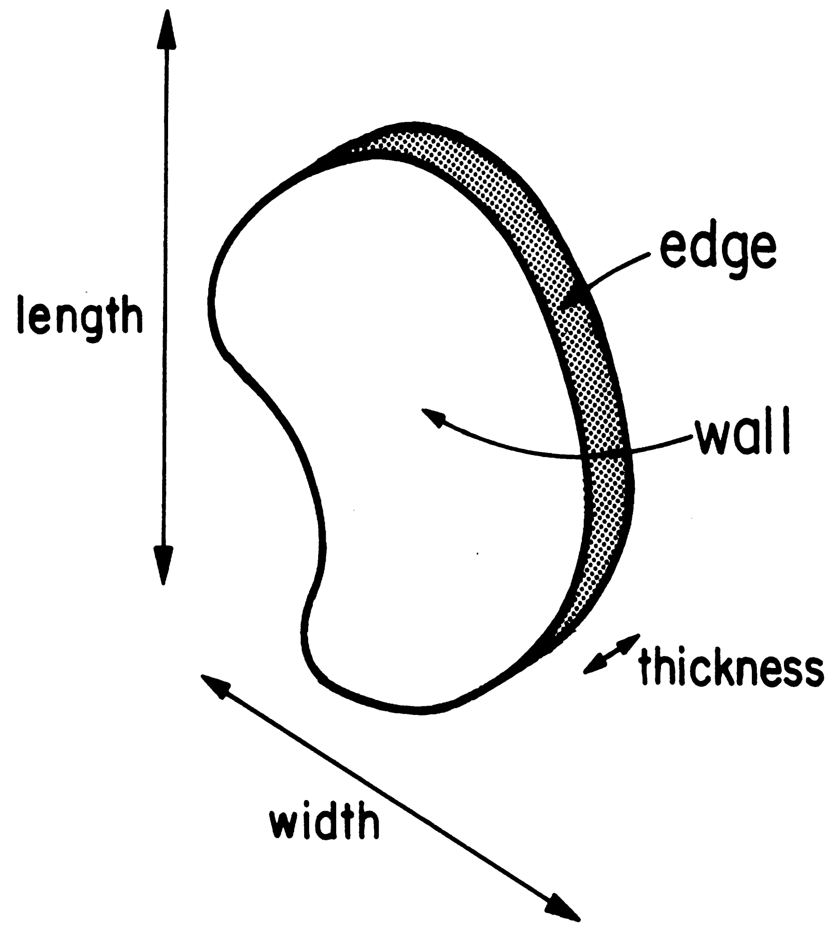

Fig. 3.1: Diagram of a Plate.

$$\phi^t = -\frac{2}{1+\tau}x + \frac{1-\tau}{1+\tau}2\left\{\int_c\int_{-t/2}^{t/2}\phi^t\frac{\partial G}{\partial n'}dz'\,d\ell'\right.$$

$$\left.+\int_s\int_{-t/2}^{t/2}\frac{\partial}{\partial x'}\left(\phi^t\frac{\partial G}{\partial z'}\right)dz'\,ds'\right\}$$

where c is the intersection of the edge with the plane z = z' and s is the intersection of the plate with the plane z = z'. The plate is of thickness t, and extends from z = -t/2 to z = t/2.

$$\phi^t = -\frac{2}{1+\tau}x + \frac{1-\tau}{1+\tau}2\int_{-t/2}^{t/2}\int_s\left\{\nabla_s'\cdot(\phi^t\nabla_s G) + \frac{\partial}{\partial z'}\left(\phi^t\frac{\partial G}{\partial z'}\right)\right\}ds'\,dz'$$

$$= -\frac{2}{1+\tau}x + \frac{1-\tau}{1+\tau}2\int_{-t/2}^{t/2}\int_s\left\{\nabla_s'\phi^t\cdot\nabla_s'G + \phi^t\nabla_s^2G\right.$$

$$\left.+\frac{\partial\phi^t}{\partial z'}\frac{\partial G}{\partial z'} + \phi^t\frac{\partial^2G}{\partial z^2}\right\}ds'\,dz'$$

$$= -\frac{2}{1+\tau}x + \frac{1-\tau}{1+\tau}2\int_{-t/2}^{t/2}\int_s\left\{-\phi^t\delta(\bar{R}-\bar{R}') + \nabla_s'\phi^t\cdot\nabla_s'G\right.$$

$$\left.+\frac{\partial\phi^t}{\partial z'}\frac{\partial G}{\partial z'}\right\}ds'\,dz'$$

where $\bar{R}$ is the field point and $\bar{R}'$ is the source point. Noting that $\bar{R}$ is on the boundary of the volume of integration,

$$\phi^t = -\frac{2}{1+\tau}x - \frac{1-\tau}{1+\tau}\phi^t + \frac{1-\tau}{1+\tau}2\int_{-t/2}^{t/2}\int_s \left\{\nabla_s'\phi^t \cdot \nabla_s' G + \frac{\partial\phi^t}{\partial z'}\frac{\partial G}{\partial z'}\right\} ds'\, dz'$$

$$\phi^t(1+\tau+1-\tau) = -2x + (1-\tau)2\int_{-t/2}^{t/2}\int_s \left\{\nabla_s'\phi^t \cdot \nabla_s' G + \frac{\partial\phi^t}{\partial z'}\frac{\partial G}{\partial z'}\right\} ds'\, dz'$$

$$\phi^t = -x + (1-\tau)\int_{-t/2}^{t/2}\int_s \left\{\nabla_s'\phi^t \cdot \nabla_s' G + \frac{\partial\phi^t}{\partial z'}\frac{\partial G}{\partial z'}\right\} ds'\, dz' \cdot \tag{3.2}$$

Since no approximations have been made yet, Eqs. (3.1) and (3.2) have identical solutions. If ϕ^t is now expanded in powers of (z/t), it is apparent from the symmetry of the problem that

$$\phi^t = \sum_{i=o}^{\infty} \phi_{2i}^t\, (z/t)^{2i} \tag{3.3}$$

i.e., odd powers of (z/t) are unneeded. To simplify Eq. (3.2) only ϕ_o^t will be kept in which case the first equation to be solved is

$$\phi_o^t = -x + (1-\tau)\int_{-t/2}^{t/2}\int_s \nabla_s'\phi_o^t \cdot \nabla_s' G\, ds'\, dz' . \tag{3.4}$$

An alternate derivation of Eq. (3.4) can be obtained by using the divergence theorem. Starting with Eq. (3.1)

$$\phi^t = -\frac{2}{1+\tau}x + \frac{1-\tau}{1+\tau}2\int_V \nabla' \cdot (\phi^t \nabla' G)\, dv'$$

$$= -\frac{2}{1+\tau}x + \frac{1-\tau}{1+\tau}2\int_V \left\{\nabla'\phi^t \cdot \nabla' G + \phi^t \nabla'^2 G\right\} dv'$$

$$= -\frac{2}{1+\tau}x - \frac{1-\tau}{1+\tau}\phi^t + \frac{1-\tau}{1+\tau}2\int_{-t/2}^{t/2}\int_s \nabla'\phi^t \cdot \nabla' G\, ds'\, dz'$$

which may be reduced to Eq. (3.2).

For $\bar{E}^{inc} = \hat{z}$, Eq. (3.1) becomes

$$\phi^t = -\frac{2}{1+\tau}z + \frac{1-\tau}{1+\tau}2\left\{\int_e \phi^t \frac{\partial G}{\partial n'} ds' + \int_w \phi^t \frac{\partial G}{\partial n'} ds'\right\}$$

and following the same procedure as before we obtain

$$\phi^t = -z + (1-\tau)\int_{-t/2}^{t/2}\int_s \left\{\nabla_s'\phi^t \cdot \nabla_s' G + \frac{\partial\phi^t}{\partial z'}\frac{\partial G}{\partial z'}\right\} ds'\, dz' \qquad (3.5)$$

without making any approximations. For $\bar{E}^{inc} = \hat{z}$, ϕ^t may be expanded in powers of (z/t),

$$\phi^t = \sum_{i=o}^{\infty} \phi_{2i+1}^t (z/t)^{2i+1}$$

and this time the even powers of (z/t) are unneeded. Keeping the first term in the expansion, and neglecting the contribution of $\nabla_s' \phi^t \cdot \nabla_s' G$ since it results in only quadrupole and higher order terms

$$\phi_1^t\left(\frac{z}{t}\right) = -z + (1-\tau)\int_{-t/2}^{t/2}\int_S \phi_1^t \frac{\partial G}{\partial z'}\, ds'\, dz'$$

$$= -z + (1-\tau)\int_S \frac{1}{t}\phi_1^t\, G\Big|_{-t/2}^{t/2}\, ds'$$

Restricting $\bar{R}$ to lie on the top surface of the plate $(z = t/2)$

$$\frac{1}{2}\phi_1^t = -\frac{t}{2} + \frac{1-\tau}{t}\int_S \phi_1^t \Big[G(z = t/2 | z' = t/2) - G(z = t/2 | z' = -t/2)\Big]\, ds' \qquad (3.6)$$

The static scattering problem has now been formulated for a thin dielectric plate with the incident electric field either tangential (Eq. 3.4) or normal (Eq. 3.6) to the plane of the plate. The numerical solution of these integral equations occupies the remainder of this chapter.

3.3 An Overview of Numerical Methods

Before discussing the numerical implementation of the integro-differential equation developed in Section 3.2, a comment about numerical methods as encountered in electromagnetic problems is perhaps appropriate. First, the variable for which the solution is sought often constrains the accuracy which can be obtained. If solutions are expressed in terms of the variable ρ, the charge density, then problems may be encountered near edges, where ρ tends to infinity.

Many solutions are posited in terms of $\bar{J}$, the current density, which as the surface integral of ρ is somewhat better behaved. The accuracy that a particular numerical method can achieve is dictated by the degree to which any approximations made in the discretization are justified. For example, in finite difference codes a derivative of a function is often approximated by taking the difference of the value of the function at two nearby points and normalizing by the distance between the two points. This is a reasonable approximation for a derivative if the function is nearly linear, however if the function is not well behaved, such as a Green's function near the point of singularity, the approximation degenerates. In finite element analysis, the domain of the problem is divided into elements, upon which basis functions are then individually imposed. The two most common basis functions are pulse functions, which are constant over individual elements and generate discontinuities between elements, and linear functions, which permit linear variation over the individual elements and can guarantee C_0 continuity over the entire domain. Thus, an integral equation which can be expressed in terms of a variable which is approximately linearly varying should be amenable to solutions using linear basis functions. The use of pulse functions is not an indication of belief that the answer contains step functions, but rather a concession to the numerical difficulties of using more complicated basis functions. Higher order basis functions exist (Zienkowicz, 1982), however only C_0 continuity may be enforced in general, and numerical difficulties seem to have precluded their popularity in electromagnetics. On the other hand, instead of partitioning the object into separate domains, another technique is to define basis functions over the entire object. Although this technique is useful (Harrington, 1982), it is predicated upon at least some a priori knowledge about the nature of the solution.

After the problem has been discretized, there often remain problems of numerical integration, if an integral is involved, or numerical differentiation if a derivative is needed. The numerical differentiation is usually handled as part of the discretization, and any attendant errors can be analyzed in terms of the discretization. The numerical integration often involves a substantial amount of additional computation time as the integrals over each element must be evaluated individually. The program which was developed over the course of this investigation analytically evaluates the integrals which are defined over the individual elements so that no approximations are required after discretization. The basis functions used are linear and are defined over triangular elements. The variable which is solved for is the potential, ϕ, which is even more stable than the current, $\bar{J}$ since $\bar{J}$ is the derivative of ϕ. Additionally, in most cases ϕ does indeed appear to be effectively linear, thus justifying the use of linear basis functions and permitting accurate solutions with a minimal number of subdomain divisions. Further, this leads to a simplified model of the program which can produce fairly accurate predictions of the dipole moment associated with a particular particle given gross parameters such as the height, length, width and permittivity. The range of applicability of the model is any convex plate, and the model and its results are discussed in Chapter IV.

3.4 Functional Description of the Problem

The first operation required in the numerical solution of the static scattering problem must be performed by the user. The plate must be divided into a set of triangular elements. This process can also be performed by automatic mesh generating programs, but this refinement was not felt necessary for the present investigation. The triangular elements are defined in terms of the vertices which delimit the triangles, and the vertices are defined by their coordinates. Figures 3.2(a), 3.3(a), and 3.4(a) show subdivisions of the circular, square, and rectangular plates into triangles with the triangles and points numbered. Figures 3.2(b), 3.3(b), and 3.4(b) are the associated inputs to the program which then define these subdivided plates. The lines beginning with the letter "p" indicate the point number, followed by the x and y coordinates. The lines beginning with the letter "t' indicate triangle number, followed by the numbers of the three points which delimit the triangle. The s3 line at the bottom indicates that the definition is only for the first quadrant and that the actual plate consists of this definition mirrored about both the line x = 0 and the line y = 0. Other options are s0, indicating no mirroring; s1, indicating mirroring about the line x = 0; and s2, indicating mirroring about y = 0. These options are used with Figs. 3.1 and 3.2 to generate a half circle and a triangle which are used in the next chapter. There are effectively only two restrictions about how the plate can be defined. Point numbers and triangle numbers must be given in increasing order starting with zero, but the actual numbering of the plates and triangles is arbitrary. The plate need not be simply connected, can be made up of any number of triangles and

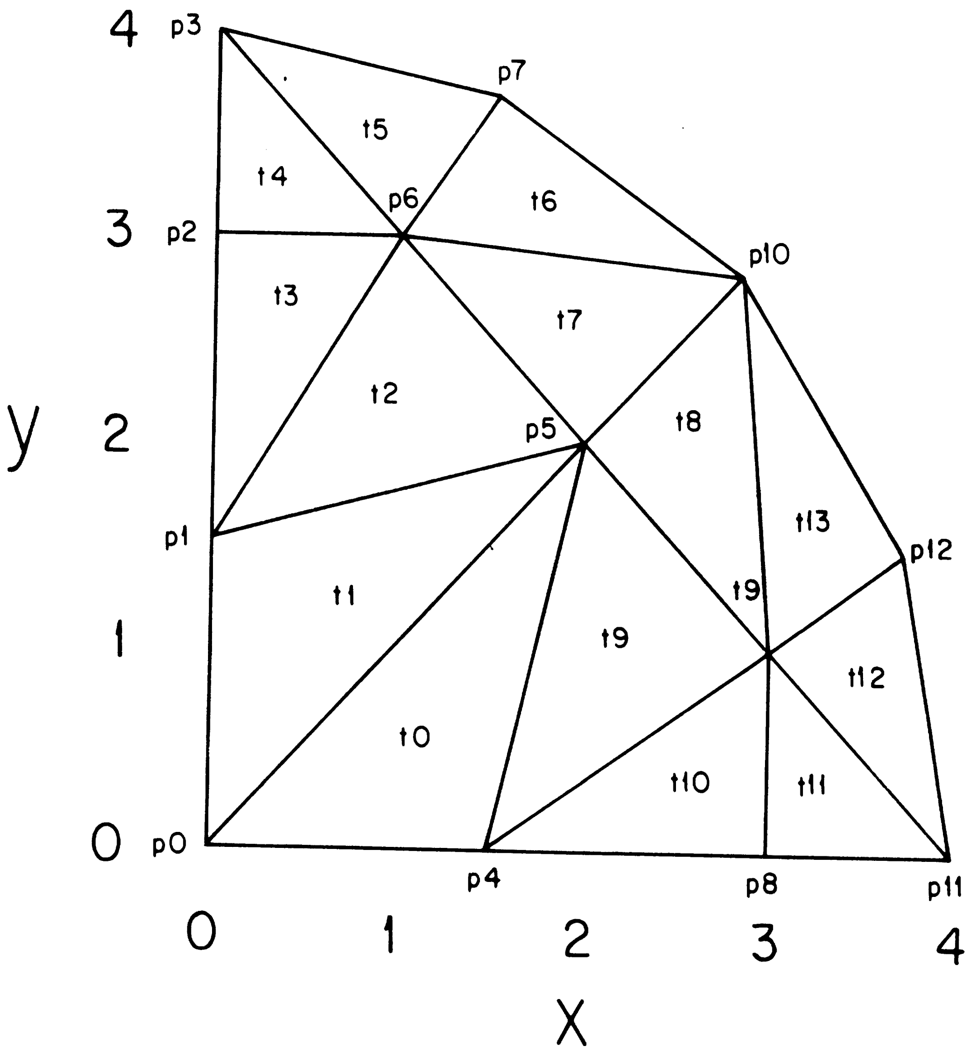

Fig. 3.2(a): Definition of Elements for a Circular Plate.

```
h16 sided approximation to circle, using 12 point definition
p0 0 0
p1 0 1.5
p2 0 3
p3 0 4
p4 1.5 0
p5 2 2
p6 1 3
p7 1.5307337 3.69551813
p8 3 0
p9 3 1
p10 2.828427125 2.828427125
p11 4 0
p12 3.69551813 1.5307337
t0 0 4 5
t1 0 1 5
t2 1 5 6
t3 1 2 6
t4 2 6 3
t5 3 6 7
t6 6 7 10
t7 6 5 10
t8 5 9 10
t9 4 5 9
t10 4 8 9
t11 8 9 11
t12 9 11 12
t13 9 10 12
s3
```

Fig. 3.2(b): Input to Computer Program to Define a Circular Plate.

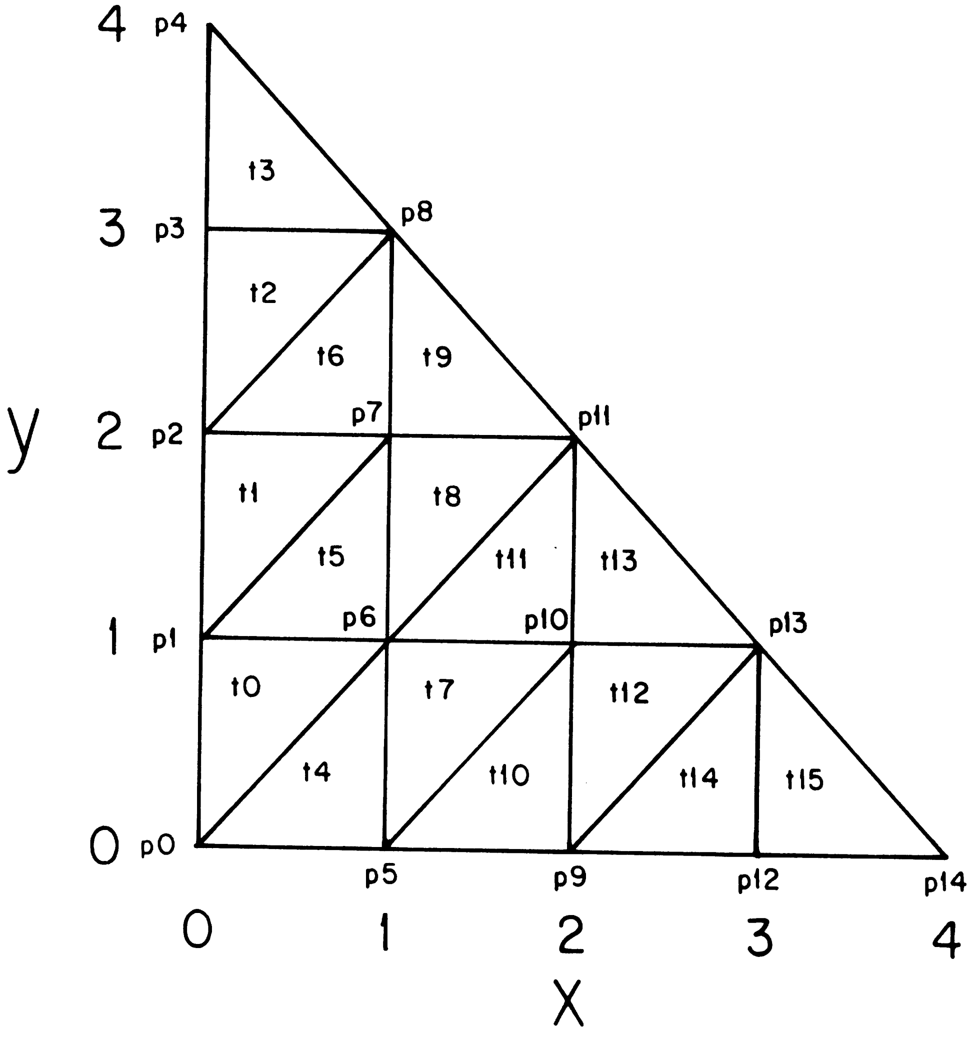

Fig. 3.3(a): Definition of Elements for a Square Plate.

```
h14 point definition of square using 2 planes of symmetry
p0 0 0
p1 0 1
p2 0 2
p3 0 3
p4 0 4
p5 1 0
p6 1 1
p7 1 2
p8 1 3
p9 2 0
p10 2 1
p11 2 2
p12 3 0
p13 3 1
p14 4 0
t0 0 1 6
t1 1 2 7
t2 2 3 8
t3 3 4 8
t4 0 5 6
t5 1 6 7
t6 2 7 8
t7 5 6 10
t8 6 7 11
t9 7 8 11
t10 5 9 10
t11 6 10 11
t12 9 10 13
t13 10 11 13
t14 9 12 13
t15 12 13 14
s3
```

Fig. 3.3(b): Input to Computer Program to Define a Square Plate.

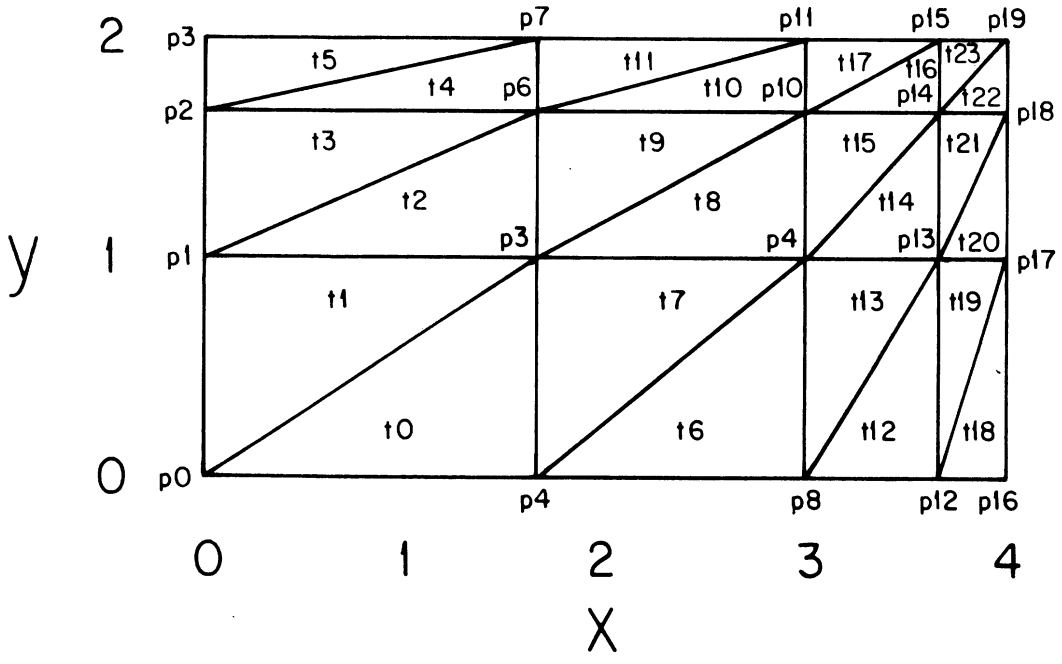

Fig. 3.4(a): Definition of Elements for a Rectangular Plate.

```
h20 point definition of rectangle
p0 0 0
p1 0 1
p2 0 1.66666666
p3 0 2
p4 1.66666666 0
p5 1.66666666 1
p6 1.66666666 1.66666666
p7 1.66666666 2
p8 3 0
p9 3 1
p10 3 1.66666666
p11 3 2
p12 3.66666666 0
p13 3.66666666 1
p14 3.66666666 1.66666666
p15 3.66666666 2
p16 4 0
p17 4 1
p18 4 1.66666666
p19 4 2
t0 0 4 5
t1 0 1 5
t2 1 5 6
t3 1 2 6
t4 2 6 7
t5 2 3 7
t6 4 8 9
t7 4 5 9
t8 5 9 10
t9 5 6 10
t10 6 10 11
t11 6 7 11
t12 8 12 13
t13 8 9 13
t14 9 13 14
t15 9 10 14
t16 10 14 15
t17 10 11 15
t18 12 16 17
t19 12 13 17
t20 13 17 18
t21 13 14 18
t22 14 18 19
t23 14 15 19
s3
```

Fig. 3.4(b): Input to Computer Program to Define a Rectangular Plate.

plates, and the triangles can be of any shape desired. Parameters such as thickness, permittivity, and the direction of the incident field are specified after the elements of the plate are defined to permit respecification of these quantities while retaining the definition of the elements. The program generates all linking information such as which triangles have which points in common, which points are connected to other points, as well as which points delimit the perimeter of the plate, the last being needed in calculating the dipole moment.

The static scattering problem can be solved assuming an incident field in either the x or y directions (tangential to the plane of the plate) or in the z direction (normal to the plane of the plate).

The evaluation of Eq. (3.4) is facilitated by first breaking S, the domain of integration, into triangular subdomains. In these regions, ϕ_o^t is restricted to be linearly varying, so that for each element

$$\phi_o^t = C_x x + C_y y + C_o \quad .$$

Then define a u-v coordinate system, with $\hat{u} = \hat{c}$, where $\bar{c} = c_x \hat{x} + c_y \hat{y}$

$$\phi_o^t = -x + (1 - \tau) \int_{-t/2}^{t/2} \int_{S_i} |\bar{c}| \frac{\partial G}{\partial u'} ds' \, dz' \quad .$$

Since the boundaries of S_i are line segments, the integral in the above equation can be evaluated as

$$I = \int_{-t/2}^{t/2} dz' \int_{V_1}^{V_2} dv' \{(av' + b - u)^2 + (v' - v)^2 + (z' - z)^2\}^{-1/2}$$

$$= \int_{-t/2}^{t/2} I_1 \, dz'$$

First evaluate

$$I_1 = \int_{V_1}^{V_2} \Big\{(1 + a^2)v'^2 + 2(a(b - u) - v)v' + (b - u)^2 + v^2 + (z' - z)^2\Big\}^{-1/2} dv'$$

From Gradshteyn and Ryzhik (1980, Sec. 2.261)

$$\int_{V_1}^{V_2} \{\tilde{A} + \tilde{B}v' + \tilde{C}v'^2\}^{-1/2} dv'$$

$$= \frac{1}{\sqrt{c}} \ln(2\sqrt{\tilde{C}R} + 2\tilde{C}v' + \tilde{B})\Big|_{V_1}^{V_2}$$

and

$$\tilde{A} = (b - u)^2 + v^2 + (z' - z)^2$$

$$\tilde{B} = 2(a(b - u) - v)$$

$$\tilde{C} = (1 + a^2)$$

$$I_1 = \frac{1}{\sqrt{1+a^2}} \ln \left(2\sqrt{1+a^2}\,\{(1+a^2)v'^2 + 2(a(b-u)-v)v'\right.$$

$$\left. + (b-u)^2 + v^2 + (z'-z)^2\}^{1/2} + 2(1+a^2)v' + 2(a(b-u)-v)\right)\Bigg|_{v'=V_1}^{v'=V_2}$$

Take the field point on the top surface, $z = t/2$. To find I, let $\eta = z' - t/2$.

$$I = \int_{-t}^{0} \frac{1}{\sqrt{1+a^2}} \ln\left(2\sqrt{1+a^2}\,\{(1+a^2)v'^2 + 2(a(b-u)-v)v'\right.$$

$$\left. + (b-u)^2 + v^2 + \eta^2\}^{1/2} + 2(1+a^2)v' + 2(a(b-u)-v)\right)\Bigg|_{v'=V_1}^{v'=V_2} d\eta$$

then, change parameters

$$A = (1+a^2)^{1/2}$$

$$B = (1+a^2)v'^2 + 2(a(b-u)-v)v' + (b-u)^2 + v^2 = (av'+b-u)^2 + (v-v')^2$$

$$C = 2(1+a^2)v' + 2(a(b-u)-v)$$

$$I = \int_{-t}^{0} \frac{1}{A} \ln(2A\{B+\eta^2\}^{1/2} + C)\Bigg|_{v'=V_1}^{v'=V_2} d\eta \quad .$$

Then, integrating by parts

$$I = \eta \frac{1}{A} \ln(2A\{B+\eta^2\}^{1/2} + C)\Bigg|_{V_1}^{V_2}\Bigg|_{-t}^{0} \qquad \text{(cont.)}$$

$$- \int_{-t}^{o} \frac{\eta}{A} \frac{1}{2A\{B + \eta^2\}^{1/2} + C} 2A \frac{1}{2} \{B + \eta^2\}^{-1/2} 2\eta \Big|_{V_1}^{V_2} d\eta \quad .$$

To evaluate the second term, call it I_2;

$$I_2 = \int_{-t}^{o} 2\eta^2 \frac{(B + \eta^2)^{-1/2}}{2A(B + \eta^2)^{1/2} + C} \Big|_{V_1}^{V_2} d\eta$$

$$= \int_{0}^{t} 2\eta^2 \frac{(B + \eta^2)^{-1/2}}{2A(B + \eta^2)^{1/2} + C} \Big|_{V_1}^{V_2} d\eta \quad .$$

Let $\xi = (B + \eta^2)^{1/2}$, $\eta = (\xi^2 - B)^{1/2}$, $d\eta = \xi/\sqrt{\xi^2 - B}\, d\xi$

$$I_2 = \int_{\sqrt{B}}^{\sqrt{t^2+B}} 2(\xi^2 - B) \frac{1/\xi}{2A\xi + C} \frac{\xi}{\sqrt{\xi^2 - B}} \Big|_{V_1}^{V_2} d\xi$$

$$= \int_{\sqrt{B}}^{\sqrt{t^2+B}} 2 \frac{\sqrt{\xi^2 - B}}{2A\xi + C} \Big|_{V_1}^{V_2} d\xi$$

Let $\gamma = 2A\xi + C$, $\xi = (\gamma - C)/2A$, $d\xi = 1/2A\, d\gamma$

$$I_2 = \int_{2A\sqrt{B}+C}^{2A\sqrt{t^2+B}+C} 2 \frac{\frac{1}{2A}\sqrt{(\gamma - C)^2 - B4A^2}}{\gamma} \frac{1}{2A} \Big|_{V_1}^{V_2} d\gamma$$

$$I_2 = \int_{2A\sqrt{B}+C}^{2A\sqrt{t^2+B}+C} \frac{1}{2A^2} \left. \frac{\sqrt{\gamma^2 - 2C\gamma + C^2 - 4BA^2}}{\gamma} \right|_{V_1}^{V_2} d\gamma$$

Changing parameters again,

$$z_1 = 2A\sqrt{B} + C \qquad a = C - 4BA$$

$$z_2 = 2A\sqrt{t^2 + B} + C \qquad b = -2C$$

$$I_2 = \int_{z_1}^{z_2} \frac{1}{2A^2} \left. \frac{\sqrt{a + b\gamma + \gamma^2}}{\gamma} \right|_{V_1}^{V_2} d\gamma = \frac{1}{2A^2} I_3$$

$$I_3 = \int_{z_1}^{z_2} \left. \frac{\sqrt{a + b\gamma + \gamma^2}}{\gamma} \right|_{V_1}^{V_2} d\gamma .$$

Then, from Gradshteyn and Ryzhik (1980, sec. 2.267.1) (with $R = a+b\gamma+\gamma^2$).

$$I_3 = \sqrt{R}\Big|_{z_1}^{z_2}\Big|_{V_1}^{V_2} + a \int_{z_1}^{z_2} \frac{\partial\gamma}{\gamma\sqrt{R}}\Big|_{V_1}^{V_2} + \frac{b}{2}\int \frac{\partial\gamma}{\sqrt{R}}\Big|_{V_1}^{V_2} .$$

This may be further reduced by using Gradshteyn and Ryzhik (1980, Secs. 2.266 and 2.261);

$$I_3 = \sqrt{R}\Bigg|_{z_1}^{z_2}\Bigg|_{V_1}^{V_2} + a\left\{-\frac{1}{\sqrt{a}}\ln\frac{2a + b\gamma + 2\sqrt{a}\sqrt{R}}{\gamma}\right\}\Bigg|_{V_1}^{V_2}\Bigg|_{z_1}^{z_2}$$

$$+ \frac{b}{2}\{\ln(2\sqrt{R} + 2\gamma + b)\}\Bigg|_{V_1}^{V_2}\Bigg|_{z_1}^{z_2}$$

$$I = \frac{\eta}{A}\ln(2A\{B + \eta^2\}^{1/2} + C)\Bigg|_{V_1}^{V_2}\Bigg|_{-t}^{0} - \frac{1}{2A^2} I_3 \quad .$$

For the electric field polarized parallel to the plane of the plate the problem is described by Eq. (3.6). This equation, with its integral over the surface of a triangle, can be evaluated with the aid of the techniques described in Wilton et al (1984).

In the numerical solutions of Eqs. (3.4) and (3.6) the integrals occasionally degenerate into simpler forms depending on the shape of the triangular subdomain and its position relative to the field point. The program tests for these problems and uses alternate expressions as appropriate. For x or z excitation the contributions of the individual elements are determined analytically. The problem associated with the electric field polarized in the y direction is entirely analogous to the problem associated with the electric field polarized in the x direction, which is discussed above.

The matrix problem is then generated by relating the elemental contributions to area coordinates (Zienkowicz, 1982) which are in turn defined by the vertices of the triangles. Thus, the problem is formulated in terms of potentials at vertices. Further, the program will set to zero the potentials of points lying on the axes, if warranted by the symmetry of the plate and the direction of excitation. The formulation yields a very stable matrix formulation for most values of τ (at least for Real $\tau > 0$), since the self cell terms of the matrix formulation of the integro-differential equation are usually the largest elements of the matrix, resulting in extremely low condition numbers for the matrix.

After the potentials are obtained, the dipole moment normalized by the volume is obtained using the formulae for a symmetric dielectric body given in Senior (1976). A sample output for the square plate subjected to an incident electric field polarized in the x direction is shown in Fig. 3.5 with $\tau = 2$ and length-to-height ratio equal to ten. Figure 3.6 shows the output obtained from the same problem but with τ changed to 101.

In the solution of any complex numerical problem, it is important to examine the extent to which the initial approximations are justified. Since the basis functions are linear the solution should be approximately linear for the solution to be considered accurate. To determine this, perspective plots were generated showing the variation of the potential across the top surface of the plate. For the case of a square disk with length-to-height ratio of ten, the potential has been plotted for $\tau = 2$ with x excitation in Fig. 3.7, for $\tau = 2$ with z excitation in Fig. 3.8, for $\tau = 101$ with x excitation in Fig. 3.9, and for $\tau = 101$ with z excitation in Fig. 3.10. For the electric field in the x direction the computed potential does indeed appear approximately linear. The electric field in the z direction produces some "buckling" near the edges, particularly for $\tau = 101$, which causes the elemental divisions to become visible. This might indicate that perhaps another formulation might be appropriate, although the solution is quite adequate for the current investigation. The problem could be overcome by shrinking the element size toward the edges, as was done for the rectangular shape, or simply using more elements.

In Figs. 3.7 through 3.10, the middle structure shows the variation of the potential across the top surface of the square plate. The $\phi = 0$ reference square is not part of the solution and is included only to lend perspective. The structure is described by quadrilaterals, and the variation in the shape and size of these quadrilaterals illustrates the variation in the gradient of the potential. In Fig. 3.7 the potential is approximately linear and this should be contrasted to Figs. 3.8 through 3.10 where the potential exhibits some nonlinearities.

```
condition number is 1.504066e+00
14 point defintion of square using 2 planes of symmetry
t = 0.800000, tau = 2.000000 + i 0.000000
potential has been computed assuming x excitation
plate has mirror symmetry about x=0 and y=0
The dipole moment is 0.913873 + i 0.000000
point # 0 is located at x = 0.000000, y = 0.000000
      and has potential 0.000000e+00
point 0 is associated with points and triangles (tri,P1,P2)
(0,1,6),  (4,5,6),
point # 1 is located at x = 0.000000, y = 1.000000
      and has potential 0.000000e+00
point 1 is associated with points and triangles (tri,P1,P2)
(0,6,0),  (1,2,7),  (5,6,7),
point # 2 is located at x = 0.000000, y = 2.000000
      and has potential 0.000000e+00
point 2 is associated with points and triangles (tri,P1,P2)
(1,7,1),  (2,3,8),  (6,7,8),
point # 3 is located at x = 0.000000, y = 3.000000
      and has potential 0.000000e+00
point 3 is associated with points and triangles (tri,P1,P2)
(2,8,2),  (3,4,8),
point # 4 is located at x = 0.000000, y = 4.000000
      and has potential 0.000000e+00
point 4 is associated with points and triangles (tri,P1,P2)
(3,8,3),
point # 5 is located at x = 1.000000, y = 0.000000
      and has potential 9.372243e-01
point 5 is associated with points and triangles (tri,P1,P2)
(4,6,0),  (7,6,10),  (10,9,10),
point # 6 is located at x = 1.000000, y = 1.000000
      and has potential 9.305898e-01
point 6 is associated with points and triangles (tri,P1,P2)
(0,0,1),  (4,0,5),  (5,7,1),  (7,10,5),  (8,7,11),  (11,10,11),
point # 7 is located at x = 1.000000, y = 2.000000
      and has potential 9.083354e-01
point 7 is associated with points and triangles (tri,P1,P2)
(1,1,2),  (5,1,6),  (6,8,2),  (8,11,6),  (9,8,11),
point # 8 is located at x = 1.000000, y = 3.000000
      and has potential 8.580674e-01
point 8 is associated with points and triangles (tri,P1,P2)
(2,2,3),  (3,3,4),  (6,2,7),  (9,11,7),
point # 9 is located at x = 2.000000, y = 0.000000
      and has potential 1.872970e+00
point 9 is associated with points and triangles (tri,P1,P2)
(10,10,5),  (12,10,13),  (14,12,13),
point # 10 is located at x = 2.000000, y = 1.000000
      and has potential 1.856799e+00
point 10 is associated with points and triangles (tri,P1,P2)
(7,5,6),  (10,5,9),  (11,11,6),  (12,13,9),  (13,11,13),
point # 11 is located at x = 2.000000, y = 2.000000
      and has potential 1.803127e+00
point 11 is associated with points and triangles (tri,P1,P2)
(8,6,7),  (9,7,8),  (11,6,10),  (13,13,10),
point # 12 is located at x = 3.000000, y = 0.000000
      and has potential 2.805956e+00
point 12 is associated with points and triangles (tri,P1,P2)
(14,13,9),  (15,13,14),
point # 13 is located at x = 3.000000, y = 1.000000
      and has potential 2.767921e+00
point 13 is associated with points and triangles (tri,P1,P2)
(12,9,10),  (13,10,11),  (14,9,12),  (15,14,12),
point # 14 is located at x = 4.000000, y = 0.000000
      and has potential 3.763731e+00
point 14 is associated with points and triangles (tri,P1,P2)
(15,12,13),
```

Fig. 3.5: Dipole Moment and Calculated Potentials for the Square Plate Defined in Fig. 3.3. Under x Excitation, $\tau = 2$, $\ell/t = 10$.

```
condition number is 9.026534e+00
14 point defintion of square using 2 planes of symmetry
t = 0.800000, tau = 101.000000 + i 0.000000
potential has been computed assuming x excitation
plate has mirror symmetry about x=0 and y=0
The dipole moment is 10.727467 + i 0.000000
point # 0 is located at x = 0.000000, y = 0.000000
      and has potential 0.000000e+00
point 0 is associated with points and triangles (tri,P1,P2)
(0,1,6),  (4,5,6),
point # 1 is located at x = 0.000000, y = 1.000000
      and has potential 0.000000e+00
point 1 is associated with points and triangles (tri,P1,P2)
(0,6,0),  (1,2,7),  (5,6,7),
point # 2 is located at x = 0.000000, y = 2.000000
      and has potential 0.000000e+00
point 2 is associated with points and triangles (tri,P1,P2)
(1,7,1),  (2,3,8),  (6,7,8),
point # 3 is located at x = 0.000000, y = 3.000000
      and has potential 0.000000e+00
point 3 is associated with points and triangles (tri,P1,P2)
(2,8,2),  (3,4,8),
point # 4 is located at x = 0.000000, y = 4.000000
      and has potential 0.000000e+00
point 4 is associated with points and triangles (tri,P1,P2)
(3,8,3),
point # 5 is located at x = 1.000000, y = 0.000000
      and has potential 1.159421e-01
point 5 is associated with points and triangles (tri,P1,P2)
(4,6,0),  (7,6,10),  (10,9,10),
point # 6 is located at x = 1.000000, y = 1.000000
      and has potential 1.106113e-01
point 6 is associated with points and triangles (tri,P1,P2)
(0,0,1),  (4,0,5),  (5,7,1),  (7,10,5),  (8,7,11),  (11,10,11),
point # 7 is located at x = 1.000000, y = 2.000000
      and has potential 9.526785e-02
point 7 is associated with points and triangles (tri,P1,P2)
(1,1,2),  (5,1,6),  (6,8,2),  (8,11,6),  (9,8,11),
point # 8 is located at x = 1.000000, y = 3.000000
      and has potential 6.743818e-02
point 8 is associated with points and triangles (tri,P1,P2)
(2,2,3),  (3,3,4),  (6,2,7),  (9,11,7),
point # 9 is located at x = 2.000000, y = 0.000000
      and has potential 2.354477e-01
point 9 is associated with points and triangles (tri,P1,P2)
(10,10,5),  (12,10,13),  (14,12,13),
point # 10 is located at x = 2.000000, y = 1.000000
      and has potential 2.238533e-01
point 10 is associated with points and triangles (tri,P1,P2)
(7,5,6),  (10,5,9),  (11,11,6),  (12,13,9),  (13,11,13),
point # 11 is located at x = 2.000000, y = 2.000000
      and has potential 1.906126e-01
point 11 is associated with points and triangles (tri,P1,P2)
(8,6,7),  (9,7,8),  (11,6,10),  (13,13,10),
point # 12 is located at x = 3.000000, y = 0.000000
      and has potential 3.639483e-01
point 12 is associated with points and triangles (tri,P1,P2)
(14,13,9),  (15,13,14),
point # 13 is located at x = 3.000000, y = 1.000000
      and has potential 3.383455e-01
point 13 is associated with points and triangles (tri,P1,P2)
(12,9,10),  (13,10,11),  (14,9,12),  (15,14,12),
point # 14 is located at x = 4.000000, y = 0.000000
      and has potential 5.236022e-01
point 14 is associated with points and triangles (tri,P1,P2)
(15,12,13),
```

Fig. 3.6: Dipole Moment and Calculated Potentials for the Square Plate Defined in Fig. 3.3. Under x Excitation, $\tau = 101$, $\ell/t = 10$.

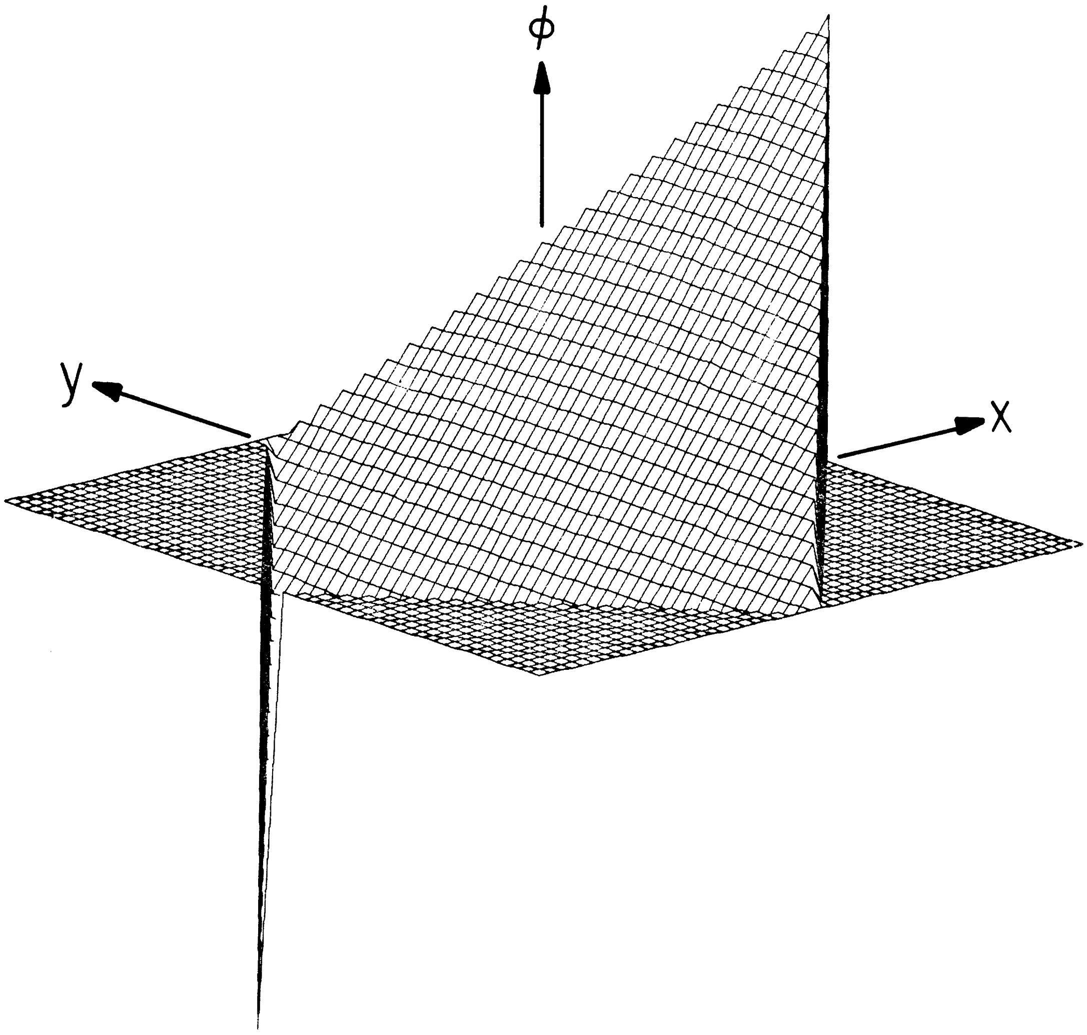

Fig. 3.7: Perspective Plot of Potential on Top Surface of Square Plate Shown Partially Hidden by $\phi = 0$ Reference Square: x Excitation, $\ell/t = 10$, $\tau = 2$.

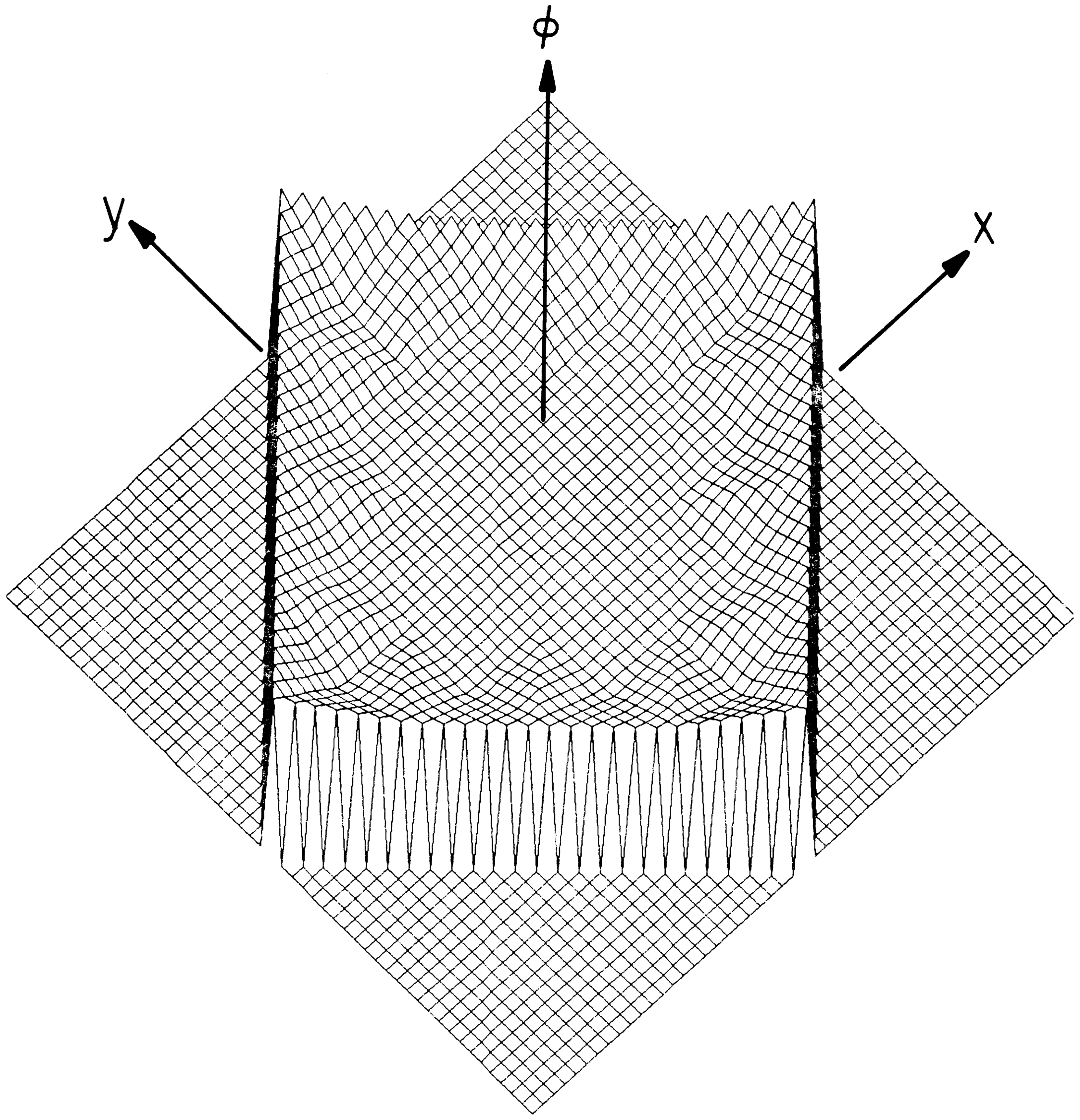

Fig. 3.8: Perspective Plot of Potential on Top Surface of Square Plate Imposed on $\phi = 0$ Reference Square: z Excitation, $\ell/t = 10$, $\tau = 2$.

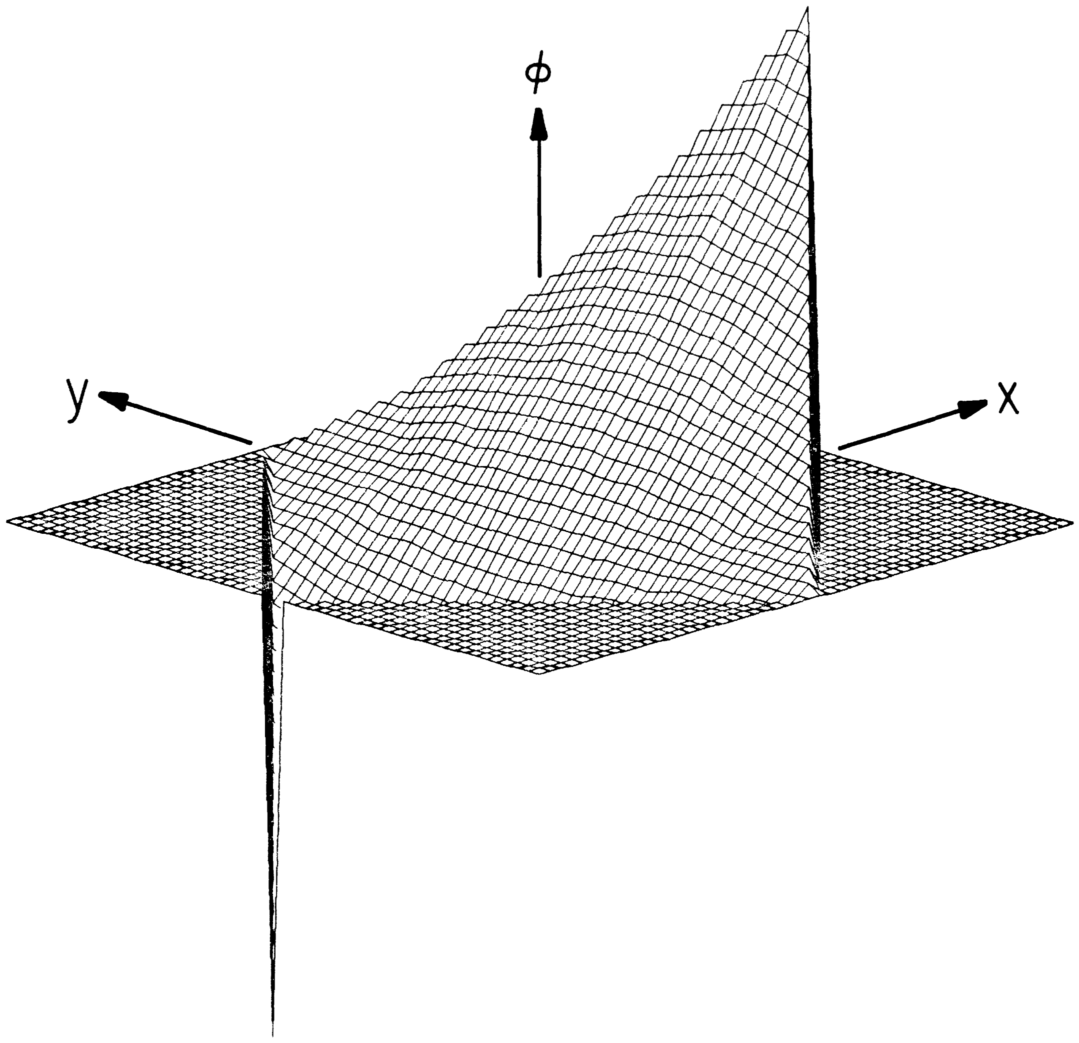

Fig. 3.9: Perspective Plot of Potential on Top Surface of Square Plate Shown Partially Hidden by $\phi = 0$ Reference Square: x Excitation, $\ell/t = 10$, $\tau = 101$.

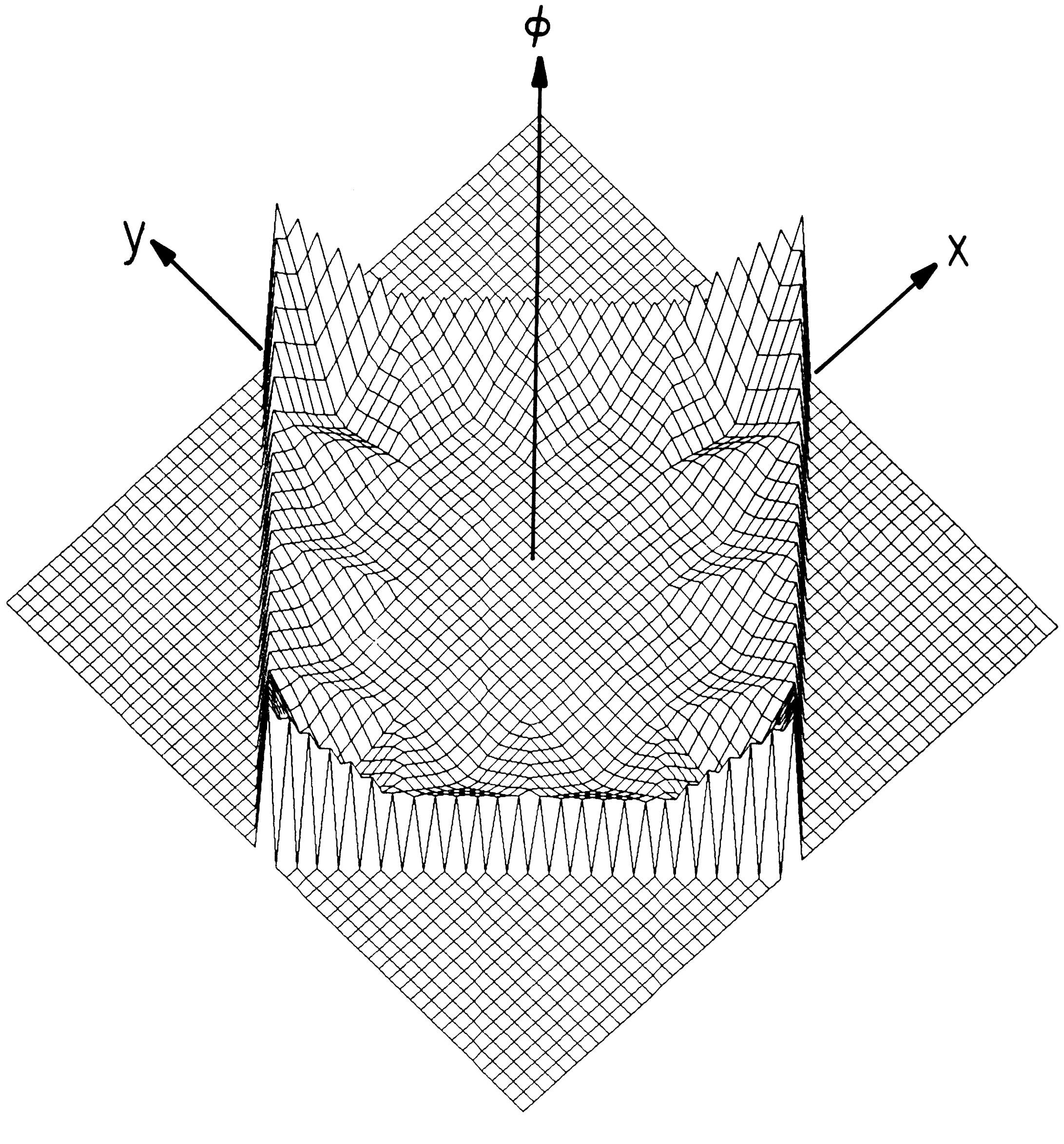

Fig. 3.10: Perspective Plot of Potential on Top Surface of Square Plate Imposed on $\phi = 0$ Reference Square: z Excitation, $\ell/t = 10$, $\tau = 101$.

CHAPTER IV. THE DIPOLE MOMENT: NUMERICAL RESULTS

4.1 Accuracy of the Results

The formulations developed in Chapter III, Eqs. (3.4) and (3.6), achieve the solution for scattering from a dielectric plate by assuming that the plate is thin. The internal field is represented as

$$\phi = f(x,y)(a + bz) \qquad (4.1)$$

where a or b is zero depending on whether the excitation is parallel or normal to the plane of the plate. The function f(x,y) is approximated by a set of continuous and piecewise linear triangular elements. The extent to which the function f(x,y) is indeed approximately linear determines the number of elements needed to accurately characterize f(x,y), which in turn determines the matrix size and the amount of computation time needed to solve the problem. It was felt that the approximation represented by Eq. (4.1) would be valid for thickness-to-length ratios of 0.1 or smaller. This was the region for which the thin plate formulation was developed, since other solid body formulations develop numerical difficulties in this region (e.g., Willis, 1982). As the program was developed to work in a region where existing programs do not work, there is no reliable data to which the results may be compared. Verification of the program is thus limited to searching for inconsistencies in results for limiting cases (none were found), and then comparing specific results with existing calculations (which are of limited accuracy) for approximate agreement. Beyond indicating approximate agreement, the reference data should not be considered as a basis for determining the accuracy of the present program, since in general the results of the present program are believed to be more accurate than the reference data. Of the data available (thickness to length ratios greater than 0.1), ratios of 0.1, 0.2, and 0.5 were selected for comparison with the program. The data are only believed accurate to within a few percent, and are included as reference merely to verify general trends in the dipole moment such as variations with thickness and permittivity. The reference data for the square cylinder was obtained from Herrick (1976) and the reference data for the circular cylinder was obtained from Senior (1975). The case of tangential excitation produced reasonable agreement with the reference data. For both the circular and square cylinders (Tables 4.1 and 4.2), the largest difference at the thickness to length ratio of 0.1 was 16 percent which occurred for the square at $\tau = 10^6$. The next largest error was only eight percent. For thickness to length ratios of 0.5 and 0.2 the largest error was 32 percent and occurred for a thickness to length ratio of 0.5 and $\tau = 10^6$. The data for normal excitation was also in reasonable agreement (Table 4.3). The fact that for the z excitation the dipole moment was overestimated relative to the reference data may be indicative of insufficiently fine elements. As shown in Fig. 3.10, the z problem does produce "buckling" near the edges which exhibits the elemental nature of the plate formulation and

Table 4.1

P_{11}/V for Plates with Circular Cross-Section

	thickness/length			
τ	0.5	0.2	0.1	10^{-6}
0	-1.418	-1.234	-1.150	$\tau - 1 = -1$
	-1.233	-1.147	-1.094	-1.000004
2	0.796	0.867	0.911	$\tau - 1 = 1$
	0.8407	0.887	0.922	0.999996
5	2.012	2.525	2.924	$\tau - 1 = 4$
	2.275	2.657	3.005	3.999929
10	2.821	3.930	4.987	$\tau - 1 = 9$
	3.327	4.218	5.174	8.999640
10^6	4.187	7.131	11.578	
	5.280	7.960	12.301	228,223

Top line of each row is from Senior (1975). The quantity P_{11} denotes a diagonal element of the polarization tensor.

Table 4.2

P_{11}/V for Plates with Square Cross-Section

	thickness/length			
τ	0.5	0.2	0.1	10^6
0	-1.470	-1.271	-1.190	$\tau - 1 = -1$
	-1.226	-1.139	-1.088	-1.000004
2	0.805	0.871	0.908	$\tau - 1 = 1$
	0.850	0.896	0.929	0.999996
5	2.104	2.593	2.944	$\tau - 1 = 4$
	2.381	2.770	3.099	3.999936
10	3.029	4.146	5.125	$\tau - 1 = 9$
	3.608	4.566	5.514	8.999678
10^6	4.763	8.164	13.127	
	6.300	9.793	15.303	261,829

Top line of each row is from Herrick (1976). The quantity P_{11} denotes a diagonal element of the polarization tensor.

Table 4.3

P_{33}/V for Plates with Square Cross-Section

	thickness/length			
τ	0.5	0.2	0.1	10^6
0	-2.256	-3.872	-6.439	$(\tau - 1)/\tau \to \infty$
	-1.945	-3.141	-4.738	1,096,342
2	0.673	0.592	0.552	$(\tau-1)/\tau = 0.5$
	0.682	0.612	0.582	0.554
5	1.402	1.086	0.951	$(\tau-1)/\tau = 0.8$
	1.428	1.164	1.067	0.983
10	1.763	1.287	1.101	$(\tau-1)/\tau = 0.9$
	1.825	1.423	1.282	1.161
10	2.258	1.524	1.262	$(\tau-1)/\tau = 1$
	2.522	1.804	1.569	1.375

Top line of each row is from Herrick (1976). The quantity P_{33} denotes a diagonal element of the polarization tensor.

indicates that the elements may not be sufficiently fine. This hypothesis is supported by the results for the rectangle (presented in the following section), which was generated using a graded mesh which is most fine near the edges.

4.2 A Linear Predictor Model

A quick inspection of the data for the circular and square cylinders presented in Tables 4.1 and 4.2 show that the dipole moment appears to be largely determined by the permittivity, τ, and the thickness to length ratio. It was felt that a model based only on gross parameters such as thickness, width, length, and τ would be able to generate a rough value for the dipole moment associated with a particle. Such a model would be useful for design purposes to get an approximate idea of the dipole moment associated with a particular particle. The model would be restricted to convex shapes, since an object consisting of needle like protrusions would obviously have a much different dipole moment than a convex object with similar thickness, length, width, and τ parameterization. The model was constructed by restricting f(x,y) in Eq. (4.1) to be linear. For the z excitation this is equivalent to restricting f(x,y) to be constant. For either tangential or normal excitation this then permits the parameterization of the problem in terms of one unknown. The problem solved is that of a rhombus symmetric about both axes. The reduction of the problem to a linearly varying potential permits much information to be obtained from recasting the problem as an eigenvalue problem in τ. After the eigenvalue is obtained, the dipole moment is then known for a given shape for all values of τ. For the x excitation the problem is solved by first examining Eq. (3.4)

$$\phi_o^t = -x + (1 - \tau) \int_{-t/2}^{t/2} \int_s \nabla_s' \phi_o^t \cdot \nabla_s' G \, ds' \, dz'$$

and the particle is assumed to be a rhombus as shown in Fig. 4.1. If ϕ_o^t is then restricted to be linear, from symmetry consideration ϕ_o^t may be written as

$$\phi_o^t = ax$$

and Eq. (3.4) reduces to

$$ax = -x + (1 - \tau) \int_{-t/2}^{t/2} \int_s a \frac{\partial G}{\partial x'} \, ds' \, dz' . \qquad (4.2)$$

The choice of testing function now becomes more important than in the previous multiple element formulation of the problem. For simplicity and to accommodate the existing program the testing function was chosen as a single delta function, centered at $x = \ell$, $y = 0$, and $z = t/2$. Then Eq. (4.2) reduces to

$$a\ell = -\ell + (1 - \tau) \int_{-t/2}^{t/2} \int_s a \frac{\partial G}{\partial x'} \, ds' \, dz' .$$

Letting

$$I = \int_{-t/2}^{t/2} \int_s \frac{\partial G}{\partial x'} \, ds' \, dz'$$

yields

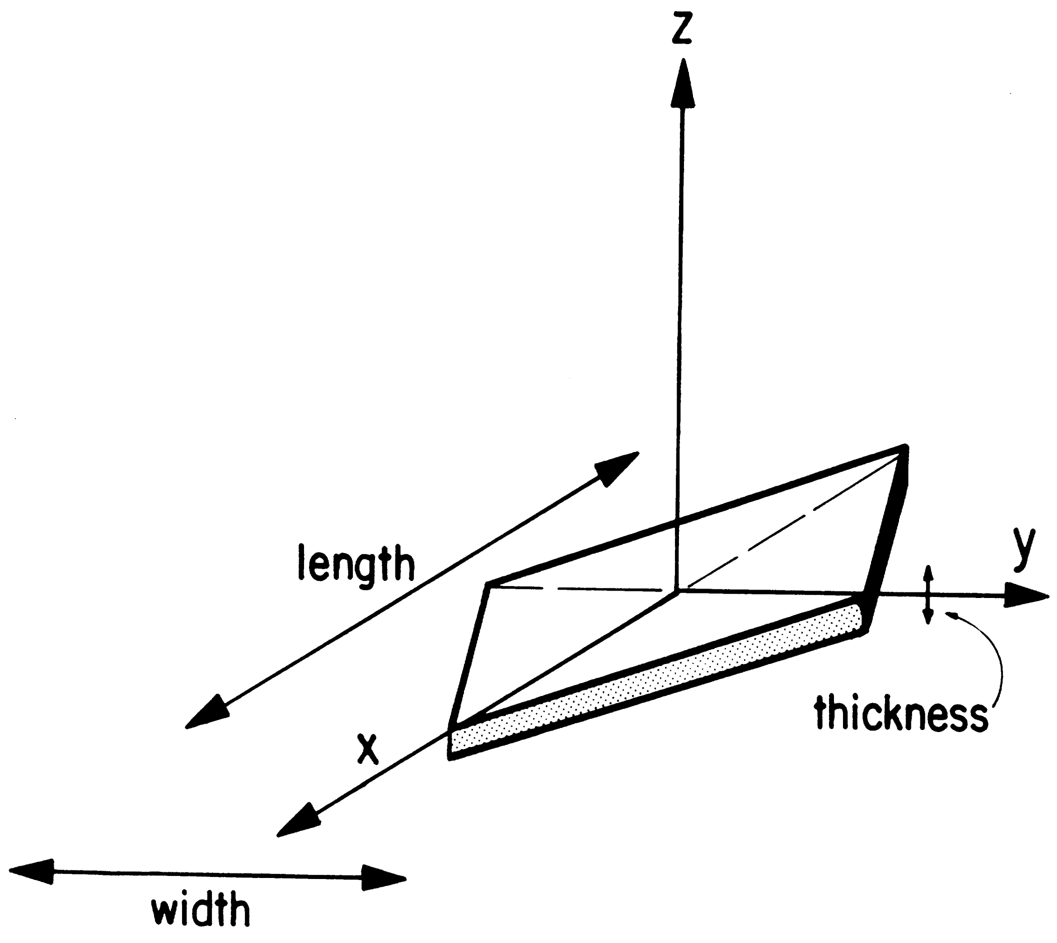

Fig. 4.1: Diagram of a Plate Used by Linear Predictor.

$$a\ell = -\ell + (1 - \tau)\, aI$$

$$a(\ell + (\tau - 1)I) = -\ell$$

$$\left(\left(\frac{\ell}{I} - 1\right) + \tau\right) a = -\frac{\ell}{I}$$

and if the above equation is considered as an eigenvalue problem in τ,

$$\lambda = 1 - \frac{\ell}{I}$$

$$(\tau - \lambda)a = -\frac{\ell}{I}$$

$$a = -\frac{\ell}{I}\,\frac{1}{\tau - \lambda}$$

and since

$$\phi_o^t = ax$$

the potential on the surface of the plate is known for all values of τ after λ and I have been computed. For a z excitation, the simplification begins with Eq. (3.5)

$$\phi_o^t = -z + (1 - \tau) \int_{-t/2}^{t/2} \int_s \left(\nabla_s' \phi_o^t \cdot \nabla_s' G + \frac{\partial \phi_o^t}{\partial z'} \frac{\partial G}{\partial z'}\right) ds'\, dz' \,.$$

Again restricting ϕ_o^t to be linear and consistent with the symmetry of the plate the equation yields

$$\phi_o^t = a\,\frac{z}{t}$$

so Eq. (3.5) becomes

$$a\,\frac{z}{t} = -z + (1 - \tau) \int_{-t/2}^{t/2} \int_{s} \frac{a}{t}\,\frac{\partial G}{\partial z'}\, ds'\, dz' \,. \qquad (4.3)$$

This time the testing function is chosen as a delta function centered at z = t/2, x = y = 0. Let

$$I = \int_{-t/2}^{t/2} \int_{s} \frac{\partial G}{\partial z'}\, ds'\, dz'$$

then Eq. (4.3) becomes

$$\frac{a}{2} = -\frac{t}{2} + (1 - \tau)\,\frac{a}{t}\, I$$

and

$$a\left(\frac{1}{2} + \frac{\tau}{t}\, I - \frac{I}{t}\right) = -\frac{t}{2}$$

$$a\left(\left(\frac{t}{2I} - 1\right) + \tau\right) = -\frac{t^2}{2I}$$

so

$$\lambda = 1 - \frac{t}{2I}$$

$$a(\tau - \lambda) = -\frac{t^2}{2I}$$

$$a = -\frac{t^2}{2I}\,\frac{1}{\tau - \lambda}$$

$$\phi_o^t = a\,\frac{z}{t}$$

and the potential is known for all values of τ after λ and I have been calculated.

The linear predictor described above was implemented and tested on particles with length to width ratios of one to one and two to one, and with length to thickness ratios of 1, 10, and 100, for x, y and z excitation. For length to width ratios of one to one, the particles tested were the circular and square cylinders (shown in Table 4.4 and 4.5), and they were tested at τ = 2, 11, and 101. For the two to one length to width ratio the particles tested were cylinders with cross sections of a triangle, semicircle, and rectangle. These shapes were tested at τ = 2, 11, and 101 (Tables 4.6, 4.7, and 4.8) and τ = 2 + i2, 11 + i2, and 101 + i2 (Tables 4.9, 4.10, and 4.11). One difference in notation between Tables 4.2 and 4.3 and Tables 4.4 and 4.5 must be noted. For Tables 4.2 and 4.3 the length (= width) was measured along the edge of the square while for Tables 4.4 and 4.5 the length (= width) was measured along the diagonal. This difference in notation was required to match the reference data available for the square in Table 4.2 and 4.3, and to be consistent with the linear predictor model presented in Tables 4.4 and 4.5. Although the effect is a change in the thickness by a factor of $\sqrt{2}$, this difference had a negligible effect on the resultant dipole moments.

As can be seen, the results obtained from the linear predictor are in fairly good agreement with those obtained from the multi-element formulation of Chapter III. The results are particularly accurate for dipole moments near τ - 1 for the tangential excitations and (τ - 1)/τ for the normal excitation. This is as expected and is discussed in detail in the next section. The linear predictor

Table 4.4

P_{11}/V for Several Convex Plates with Length/Width = 1

$\tau =$	2	11	101
	τ-1 = 1	τ-1 = 10	τ-1 = 100
ℓ/t = 1			
Linear predictor	0.858	3.77	5.72
14-pt square	0.817	3.21	4.65
12-pt circle	0.817	3.09	4.28
Sphere-Exact	0.750	2.31	2.91
ℓ/t = 10			
Linear predictor	0.941	6.15	13.8
14 pt. square	0.913	5.33	10.7
12 pt circle	0.922	5.49	10.9
ℓ/t = 100			
Linear predictor	0.988	8.97	46.7
14 pt square	0.982	8.54	39.9
12 pt circle	0.985	8.73	42.3

Table 4.5

P_{33}/V for Several Convex Plates with Length/Width = 1

τ =	2	11	101
	$(\tau-1)/\tau=1/2$	$(\tau-1)/\tau=0.909$	$(\tau-1)/\tau=0.990$
$\ell/t = 1$			
Linear predictor	0.757	2.37	3.02
14-pt square	0.800	2.94	4.31
12-pt circle	0.771	2.59	3.59
Sphere-Exact	0.750	2.31	2.91
$\ell/t = 10$			
Linear predictor	0.533	1.02	1.13
14 pt. square	0.595	1.36	1.62
12 pt circle	0.577	1.25	1.45
$\ell/t = 100$			
Linear predictor	0.503	0.919	1.002
14 pt square	0.557	1.192	1.371
12 pt circle	0.544	1.121	1.267

Table 4.6

P_{11}/V for Several Convex Plates with Length/Width = 2

τ =	2	11	101
	τ-1 = 1	τ-1 = 10	τ-1 = 100
ℓ/t = 1			
Linear predictor	0.912	5.11	9.46
14 pt triangle	0.865	4.33	7.96
12 pt half-circle	0.866	4.06	6.59
20 pt rectangle	0.864	4.03	6.51
ℓ/t = 10			
Linear predictor	0.963	7.24	20.8
14 pt triangle	0.930	6.21	16.3
12 pt half circle	0.939	6.26	15.0
20 pt rectangle	0.939	6.36	15.8
ℓ/t = 100			
Linear predictor	0.992	9.33	58.8
14 pt triangle	0.984	8.78	48.6
12 pt half circle	0.987	8.95	49.5
20 pt rectangle	0.987	9.00	53.4

Table 4.7

P_{22}/V for Several Convex Plates with Length/Width = 2

τ =	2	11	101
	τ-1 = 1	τ-1 = 10	τ-1 = 100
ℓ/t = 1			
Linear predictor	0.794	2.78	3.72
14 pt triangle	0.772	2.73	3.90
12 pt half-circle	0.768	2.55	3.38
20 pt rectangle	0.768	2.57	3.43
ℓ/t = 10			
Linear predictor	0.886	4.38	7.23
14 pt triangle	0.858	4.09	7.29
12 pt half circle	0.870	4.13	6.80
20 pt rectangle	0.879	4.34	7.40
ℓ/t = 100			
Linear predictor	0.974	7.94	27.8
14 pt triangle	0.963	7.44	26.5
12 pt half circle	0.970	7.73	26.8
20 pt rectangle	0.973	8.03	31.8

Table 4.8

P_{33}/V for Several Convex Plates with Length/Width = 2

τ =	2	11	101
	$(\tau-1)/\tau=1/2$	$(\tau-1)/\tau=0.909$	$(\tau-1)/\tau=0.990$
$\ell/t = 1$			
Linear predictor	0.809	2.98	4.07
14 pt triangle	0.851	3.74	6.15
12 pt half circle	0.826	3.29	5.01
20 pt rectangle	0.802	2.94	4.20
$\ell/t = 10$			
Linear predictor	0.550	1.09	1.21
14 pt triangle	0.641	1.63	2.04
12 pt half circle	0.621	1.50	1.81
20 pt rectangle	0.582	1.26	1.46
$\ell/t = 100$			
Linear predictor	0.504	0.925	1.01
14 pt triangle	0.594	1.38	1.66
12 pt half circle	0.583	1.32	1.54
20 pt rectangle	0.526	1.02	1.13

Table 4.9

P_{11}/V for Several Convex Plates with Length/Width = 2 and Complex Permittivity

τ =	2+i2	11+i2	101+i2
τ-1 =	1+i2	10+i2	100+i2
ℓ/t = 1			
Linear predictor	1.19+i1.61	5.16+i.517	9.46+i.017
14,14 pt triangle	1.19+i1.41	4.37+i.413	7.96+i.015
12,12 pt half circle	1.22+i1.40	4.10+i.337	6.59+i.009
20 pt rectangle	1.22+i1.39	4.07+i.333	6.51+i.009
ℓ/t = 10			
Linear predictor	1.09+i1.84	7.30+i1.04	20.8+i.086
14 pt triangle	1.13+i1.69	6.26+i.820	16.3+i.063
12 pt half circle	1.13+i1.73	6.31+i.798	15.0+i.047
20 pt rectangle	1.13+i1.73	6.42+i.833	15.8+i.053
ℓ/t = 100			
Linear predictor	1.02+i1.97	9.36+i1.71	58.5+i.686
14 pt triangle	1.04+i1.93	8.81+i1.56	48.6+i.521
12 pt half circle	1.03+i1.94	8.98+i1.61	49.5+i.510
20 pt rectangle	1.03+i1.94	9.03+i1.64	53.4+i.601

Table 4.10

P_{22}/V for Several Convex Plates with Length/Width = 2

and Complex Permittivity

τ =	2+i2	11+i2	101+i2
τ-1 =	1+i2	10+i2	100+i2
ℓ/t = 1			
Linear predictor	1.23+i1.07	2.81+i.152	3.72+i.002
14 pt triangle	1.18+i1.00	2.75+i167	3.90+i.004
12 pt half circle	1.20+i.977	2.57+i.133	3.38+i.002
20 pt rectangle	1.20+i.982	2.59+i.137	3.43+i.002
ℓ/t = 10			
Linear predictor	1.22+i1.49	4.42+i.379	7.23+i.010
14 pt triangle	1.21+i1.37	4.13+i.368	7.29+i.014
12 pt half circle	1.22+i1.42	4.17+i.348	6.80+i.010
20 pt rectangle	1.21+i1.46	4.29+i.386	7.40+i.011
ℓ/t = 100			
Linear predictor	1.07+i1.89	7.99+i1.26	27.8+i.155
14 pt triangle	1.09+i1.84	7.49+i1.13	26.5+i.170
12 pt half circle	1.07+i1.87	7.78+i1.20	26.8+i.153
20 pt rectangle	1.06+i1.88	8.08+i1.31	31.8+i.216

Table 4.11

P_{33}/V for Several Convex Plates with Length/Width = 2 and Complex Permittivity

length/width

τ =	2+i2	11+i2	101+i2
$(\tau-1)/\tau$ =	0.75+i.25	0.912+i.016	0.990+i.0002
ℓ/t = 1			
Linear predictor	1.24+i1.14	3.00+i.174	4.07+i.003
14 pt triangle	1.23+i.133	3.77+i.291	6.15+i.011
12 pt half circle	1.24+i1.21	3.32+i.224	5.01+i.007
20 pt rectangle	1.23+i1.11	2.96+i.178	4.20+i.005
ℓ/t = 10			
Linear predictor	0.852+i.335	1.09+i.023	1.21+i.000
14 pt triangle	1.009+i.571	1.64+i.063	2.04+i.001
12 pt half circle	0.978+i.517	1.51+i.052	1.81+i.001
20 pt rectangle	0.911+i.411	1.26+i.034	1.46+i.000
ℓ/t = 100			
Linear predictor	0.759+i.257	0.928+i.016	1.01+i.000
14 pt triangle	0.926+i.468	1.39+i.045	1.65+i.001
12 pt half circle	0.907+i.442	1.32+i.039	1.54+i.000
20 pt rectangle	0.801+i.303	1.02+i.021	1.13+i.000

uniformly underestimates the dipole moment calculated for z excitation (for Real $\tau > 1$). This is a result of choosing the testing function as a delta function centered on the middle of the plate thus eliminating any effects of the "buckling" near the edges of the plate. This difference between the linear predictor and the multi-element algorithm is in the opposite direction of the difference between the multi-element algorithm and the reference data, thus the result of the linear predictor for the z excitation might be better than Tables 4.5, 4.8, and 4.11 would indicate. Also, the dipole moments calculated for the rectangle for z excitation are lower than those calculated for the triangle and semicircle with the same thickness and τ. This supports the idea that the multi-element formulation for the z excitation might be using elements which are too coarse to accurately represent the buckling near the edges.

Finally, it is interesting to examine the eigenvalues extracted by the linear predictor algorithm, and these are shown in Table 4.12. These seem to parallel the path of the eigenvalues associated with the spheroid (e.g., Senior and Weil, 1982) and would seem to have physical significance. Since there is only a single eigenvalue the interpretation is simply as an estimate for the location of the resonance region. The most striking result is that the eigenvalue for the z excitation, though negative as expected, is extremely close to zero, while the x and y components are much farther from zero. From the perspective of designing particles with high absorption (operating in the resonance region), the important part of the dipole moment may be the z component. This is in contradistinction to the region

Table 4.12

Eigenvalues Extracted by Linear Predictor

length/width = 1

	x excitation	z excitation
t/ℓ = 1.0	-5.07	-2.12
t/ℓ = 0.1	-15.02	-0.14
t/ℓ = 0.01	-86.62	-0.01

length/width = 2

	x excitation	y excitation	z excitation
t/ℓ = 1.0	-9.45	-2.86	-3.25
t/ℓ = 0.1	-25.32	-6.80	-0.22
t/ℓ = 0.01	-140.44	-37.66	-0.01

of Real $\tau > 0$, where the normal component is usually negligible compared to the tangential components.

4.3 Shape Effects

As may be seen from Tables 4.4 through 4.11, the shape of a plate does have an effect on its associated dipole moments. Shape is important when the product of thickness/length (t/ℓ) and τ is large. For $(\tau t/\ell)$ small the tangential components of the dipole moment may be approximated by $\tau-1$, and the normal component may be approximated by $(\tau-1)/\tau$. The physical significance of these approximations for the tangential excitation is a total electric field $\bar{E}^t$ which equals the incident electric field $\bar{E}^i$. For the normal excitation, the total electric field $\bar{D}^t$ is assumed to equal the incident electric field $\bar{D}^i$. These results may be obtained by examining the formulae for the dipole moment. From Senior (1976)

$$X_{ii} = (1 - \tau) \int_B \hat{n} \cdot \hat{x}_i \, \phi_i^t \, ds' \qquad (4.4)$$

starting with $\hat{x}_i = \hat{x}$ for tangential excitation.

$$\phi_i^t = \phi_i^i + \phi_i^s = -x + \phi_1^s \quad \text{(on the plate)}$$

$$X_{11} = (\tau - 1) \int_B \hat{n} \cdot \bar{x} \, ds' + (1 - \tau) \int_B \hat{n} \cdot \hat{x} \, \phi_1^s \, ds'$$

$$= (\tau - 1) \int_{V_B} \nabla \cdot \bar{x} \, dv' + (1 - \tau) \int_{V_B} \nabla \cdot (\hat{x}\phi_1^s) \, dv' \qquad \text{(cont.)}$$

$$= (\tau - 1)V_B + (1 - \tau) \int_{V_B} \frac{\partial \phi_1^s}{\partial x'} dv'$$

So, for

$$\left|\frac{\partial \phi_1^s}{\partial x'}\right| << 1$$

$$X_{11} \approx (\tau - 1)V_B \quad .$$

For normal excitation, $\hat{x}_i = \hat{z}$

$$\phi_i^t = \frac{1}{\tau} \phi_i^i + \phi_i^s = -\frac{1}{\tau} \bar{z} + \phi_3^s \quad \text{on the plate from (4.4)}$$

$$X_{33} = (\tau-1) \int_B \frac{1}{\tau} \hat{n} \quad \bar{z} \, ds' + (1-\tau) \int_B \hat{n} \cdot \hat{z} \, \phi_3^s \, ds'$$

$$= \frac{\tau-1}{\tau} \int_{V_B} \nabla \cdot \bar{z} \, dv' + (1-\tau) \int_{V_B} \nabla \cdot (\hat{z} \, \phi_3^s) \, dv'$$

$$= \frac{\tau-1}{\tau} V_B + (1-\tau) \int_{V_B} \frac{\partial \phi_3^s}{\partial z'} dv'$$

So, for $\left|\tau \frac{\partial \phi_3^s}{\partial z'}\right| << 1$

$$X_{33} \approx \frac{\tau-1}{\tau} V_B$$

Thus both approximations assume the scattered field on the plate is negligible compared to the incident field. The motivation for the linear predictor model was to account for small but non-zero scattered fields. Since the incident field would still be the dominant component, the total potential on the plate should still be effectively linear. The linear predictor provides a next order approximation to the scattered field. Instead of assuming the scattered field is negligible, it is linear. This permits an increase in accuracy over the small scattered field approximation and incorporates gross shape effects. However, as the scattered field increases the importance of the non-linear components of the scattered field also increases. An indication of the size of the scattered field is the amount by which the computed dipole moment (using the linear predictor model) differs from the small scattered field model (i.e., $\tau - 1$ or $(\tau-1)/\tau$). The primary benefits of the linear predictor model then are to give a rough estimate of the dipole moment, and when such estimates differ from the small field estimates of the dipole moment to indicate that use of the multi-element formulation is appropriate.

The region where the use of the multi-element formulation is appropriate constitutes a very important region, particularly in regard to the analysis of aerosols. Particles which remain suspended in the air generally have a large surface area relative to their weight. Thin particles consitute a common realization of this characteristic. In the case of aerosols which efficiently absorb or reflect radio waves, the constituent particles must either have a relatively large permittivity or a permittivity in the resonance region of the particle. The dipole moments assocated with both of these cases are strongly influenced by the shape of the particle, and this may be further investigated through the use of the multi-element program developed in Chapter III.

CHAPTER V. SCATTERING BY DISTRIBUTIONS OF PARTICLES

5.1 The Clausius-Mosotti-Lorentz-Lorenz Equation

The problem of scattering by sparse distributions of particles is now considered. Although the formulae used are classical (Mosotti, 1850), various modifications to the standard form have surfaced to describe experimental results for distributions with high densities. In this chapter the Clausius-Mosotti-Lorentz-Lorenz equation will be derived, and it will be shown that although modifications to this formula may approximately describe experimental results, such extensions are not rigorous and hence should not be expected to work in cases other than those for which they were developed. The final section discusses scattering by plates as formulated in the two preceding chapters, and the validity of a limit case is shown.

Assume a uniform applied field, $\bar{E}$, in a distribution of particles. For each particle, a dipole moment $\bar{\mu}$ is generated, where

$$\bar{\mu} = \alpha \bar{E}_\ell ,$$

where α is the polarizability of the particle, and $\bar{E}_\ell$ is the local electric field acting on the particle. $\bar{E}_\ell$ differs from $\bar{E}$ as a result of near-field disturbances of nearby particles. As the distribution of particles becomes sparse, $\bar{E}_\ell$ will converge to $\bar{E}$.

Now, if the density of particles is N per unit volume, then

$$\bar{P} = N\bar{\mu} = N\alpha\bar{E}_{\ell} \ .$$

However, from the definition of $\bar{P}$,

$$\bar{P} = \bar{D} - \varepsilon_0\bar{E} = \bar{E}(\varepsilon - \varepsilon_0) = \bar{E}\varepsilon_0(\varepsilon_r - 1) \cdot$$

To find a relation between the polarizability per unit volume, $N\alpha$, and the relative permittivity, ε_r, assume a capacitor as shown in Fig. 5.1.

The distribution of particles is now located between a pair of infinite parallel plates. The region between the plates is uniformly filled by the distribution of particles. The voltage between the plates is

$$V = d|\bar{E}| \ , \quad \bar{E} = E_0(-\hat{z}) \ .$$

And,

$$\bar{E}_{\ell}(A) = \bar{E}_1(A) + \bar{E}_2(A) + \bar{E}_3(A) \ ,$$

where, $\bar{E}_{\ell}(A)$ is the local electric field acting upon the particle located at "A".

$\bar{E}_1(A)$ is the electric field resulting from the charges located on the metal plates.

$\bar{E}_2(A)$ is the electric field resulting from the polarized particle distribution exterior to the sphere "S". The sphere "S"

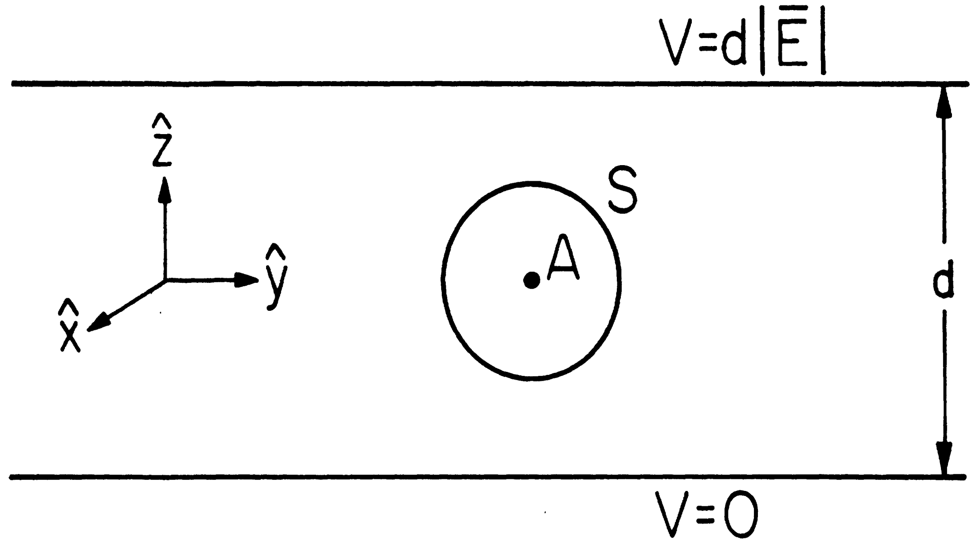

Fig. 5.1: Parallel Plate Capacitor with Spherical Exclusion Volume.

is taken large enough so that the near field effect of the individual polarized particles is not apparent when viewed from "A".

$\bar{E}_3(A)$ is the electric field resulting from the polarized particle distribution interior to the sphere "S". In general, it is not possible, or at least not easy, to calculate $\bar{E}_3$. For a cubical array of particles (i.e., particles lying on the vertices of adjoining cubes) $\bar{E}_3 = 0$, as shown in the following section. The Mosotti (1850) approximation is to simply set $\bar{E}_3 = 0$.

The next step is to calculate $\bar{E}_1$ and $\bar{E}_2$.

$$\bar{E}_1 = \frac{\rho}{\varepsilon_o}(-\hat{z}) \quad , \quad \rho \equiv \text{charge density on top plate} \quad .$$

$\bar{E}_2$ has two components. The first component is due to a bound charge distribution in the vicinity of the plates. The second is the bound charge distribution surrounding the sphere. Since the sphere is large enough to hide individual particle effects, it is permissible to consider the distribution of particles exterior to the sphere as a dielectric continuum and use terms such as "bound charge". So

$$\bar{E}_2 = \frac{\rho_{b1}}{\varepsilon_o}(-\hat{z}) - \oint_S \frac{\rho_{b2}}{4\pi\varepsilon_o R^2}\overline{ds} \quad .$$

Then, by linear superposition,

$$\bar{E}_\ell = \bar{E}_1 + \bar{E}_2$$

$$= \frac{\rho}{\varepsilon_o}(-\hat{z}) + \frac{\rho_{b1}}{\varepsilon_o}(-\hat{z}) - \oint_S \frac{\rho_{b2}}{4\pi\varepsilon_0 R^2}\,\bar{ds}$$

$$= \frac{\rho + \rho_{b1}}{\varepsilon_o}(-\hat{z}) + \oint_S \frac{\bar{P}\cdot\hat{n}}{4\pi\varepsilon_0 R^2}\,\bar{ds} \quad .$$

Since $\nabla \cdot \bar{P} = -\rho_{b1}$, and further, since the effects of the polarized particles within the sphere are accounted for with E_3, the dielectric representation of the polarized particles external to the sphere may be assumed to terminate on the surface of the sphere. Hence, $\bar{P} \equiv 0$ within the sphere.

Now, despite the segmentation of the particle distribution with the sphere, the distribution is, in fact, uniform. Therefore,

$$\bar{P} = |\bar{P}|(-\hat{z})$$

and

$$\bar{P} \cdot \hat{n} = -\,|\bar{P}|\cos\theta$$

so

$$\bar{E}_\ell = \frac{\rho + \rho_{b1}}{\varepsilon_o}(-\hat{z}) - \oint_S \frac{|\bar{P}|\cos\theta}{4\pi\varepsilon_0 R^2}\,d\bar{s} \quad .$$

But, by the symmetries of the problem, $\bar{E}_\ell$ has only a z component. Therefore

$$\bar{E}_\ell = \frac{\rho + \rho_{b1}}{\varepsilon_o}(-\hat{z}) - \oint_s \frac{|\bar{P}| \cos^2\theta}{4\pi\varepsilon_o R^2} ds\, \hat{z}$$

$$= \frac{\rho + \rho_{b1}}{\varepsilon_o}(-\hat{z}) - \hat{z}\frac{|\bar{P}|}{4\pi\varepsilon_o}\int_0^{2\pi}\int_0^{\pi}\frac{\cos^2\theta}{R^2} R^2 \sin\theta\, d\theta\, d\phi$$

$$= \frac{\rho + \rho_{b1}}{\varepsilon_o}(-\hat{z}) - \hat{z}\frac{|\bar{P}|}{4\pi\varepsilon_o}(2\pi)\left[-\frac{1}{3}\cos^2\theta\right]\Big|_0^{\pi}$$

$$= \frac{\rho + \rho_{b1}}{\varepsilon_o}(-\hat{z}) - \hat{z}\frac{|\bar{P}|}{2\varepsilon_o}\frac{2}{3}$$

$$= -\hat{z}\;\frac{\rho + \rho_{b1}}{\varepsilon_o} + \frac{|\bar{P}|}{3\varepsilon_o}$$

and

$$\frac{\rho + \rho_{b1}}{\varepsilon_o} = |\bar{E}| \quad ,$$

so

$$\bar{E}_\ell = \bar{E} + \frac{\bar{P}}{3\varepsilon_o}$$

$$= \bar{E} + \frac{\bar{D} - \varepsilon_o E}{3\varepsilon_o} = \frac{\bar{D} - \varepsilon_o E + 3\varepsilon_o E}{3\varepsilon_o}$$

$$= \frac{\bar{D} + 2\varepsilon_o E}{3\varepsilon_o} = \bar{E}\,\frac{\varepsilon_o\varepsilon_r + 2\varepsilon_o}{3\varepsilon_o}$$

$$= \bar{E}\,\frac{\varepsilon_r + 2}{3} \quad .$$

Thus

$$\bar{P} = N\alpha\bar{E}_\ell = N\alpha\,\bar{E}\,\frac{\varepsilon_r + 2}{3}$$

but

$$\bar{P} = \bar{D} - \varepsilon_o\bar{E} = \bar{E}(\varepsilon_o\varepsilon_r - \varepsilon_o) = \bar{E}\varepsilon_o(\varepsilon_r - 1) \quad .$$

So

$$\bar{E}\varepsilon_0(\varepsilon_r - 1) = N\alpha\bar{E}\,\frac{\varepsilon_r + 2}{3}$$

$$\varepsilon_0(\varepsilon_r - 1) + N\alpha\,\frac{\varepsilon_r + 2}{3}$$

and rearranging

$$N\alpha = 3\varepsilon_0\,\frac{\varepsilon_r - 1}{\varepsilon_r + 2}$$

which is the relation desired.

$$\frac{N\alpha}{3\varepsilon_0} = \frac{\varepsilon_r - 1}{\varepsilon_r + 2}$$

is the standard form of writing the Clausius-Mosotti-Lorentz-Lorenz equation.

Solving for ε_r,

$$(\varepsilon_r + 2)\,\frac{N\alpha}{3\varepsilon_0} = \varepsilon_r - 1$$

$$\varepsilon_r\left(\frac{N\alpha}{3\varepsilon_0} - 1\right) = -1 - 2\,\frac{N\alpha}{3\varepsilon_0}$$

$$\varepsilon_r = \frac{1+2\,\frac{N\alpha}{3\varepsilon_0}}{1 - \frac{N\alpha}{3\varepsilon_0}}$$

$$= 1 + \frac{3\,\frac{N\alpha}{3\varepsilon_0}}{1 - \frac{N\alpha}{3\varepsilon_0}}$$

$$= 1 + \frac{N\alpha}{\varepsilon_0 \left(1 - \frac{N\alpha}{3\varepsilon_0}\right)}$$

5.2 Cubical Distributions of Particles

As noted in the previous section the exclusion volume is a virtual cavity which has no physical significance, and is merely a computational artifice. The exclusion volume must satisfy two constraints. First, all particles outside the exclusion volume must be far enough away to be accurately represented by an equivalent, homogeneous, dielectric. Second, the combined effect of the particles within the exclusion volume should not alter the electric field at the point of computation. The first constraint may be satisfied immediately by letting the exclusion volume be arbitrarily large. The second constraint is not usually possible to satisfy exactly for an arbitrary particle distribution, so that an approximate solution is used which works reasonably well.

Since the task of determining the field contributions of an arbitrary particle distribution is in general very difficult, a highly symmetric particle distribution is used. The standard reasoning is that if the field contribution of a symmetric particle distribution is zero, then the field contribution of a random distribution of particles should also be zero. The derivation for a cubical distribution is given below and follows the approach of Jackson (1975).

The field resulting from a dipole is

$$\bar{E} = \frac{3\hat{n}(\bar{p} \cdot \hat{n}) - \bar{p}}{|x - x_0|^3}$$

where $\bar{p}$ = the dipole moment,

$\bar{x}_0$ = the location of the dipole,

$\bar{x}$ = the point of field measurement and

$\hat{n}$ = points from x_0 to x.

Then the total $\bar{E}$ field due to a cubical array of dipoles, each with identical $\bar{p}$, is

$$\bar{E} = \sum_{ijk} \frac{3(x_i,y_j,z_k)[\bar{p} \cdot (x_i,y_j,z_k)] - \bar{p}|\bar{x}|^2}{|\bar{x}|^5}$$

To show $E_x = E_y = E_z = 0$, write

$$E_x = \sum_{ijk} \frac{3x_i[p \cdot (x_i,y_j,z_k)] - p_i|\bar{x}|^2}{|\bar{x}|^5}$$

The summation is taken over all vertices of the array within the exclusion volume, and (x_i,y_j,z_k) are the coordinates of each dipole.

$$E_x = \sum_{ijk} \frac{3x_i^2p_x + 3x_iy_jp_y + 3x_iz_kp_z - p_x|\bar{x}|^2}{|\bar{x}|^5} \overset{?}{=} 0$$

Since this equation must hold for all choices of p_x, p_y, p_z, then

$$\sum_{ijk} \frac{x_i y_j}{|\bar{x}|^5} = \sum_{ijk} \frac{x_i z_k}{|\bar{x}|^5} = 0$$

and from similar formulae for $E_y = E_z = 0$, we get

$$\sum_{ijk} \frac{y_i z_k}{|\bar{x}|^5} = 0 \quad .$$

This represents a rather strong symmetry constraint, and is satisfied by the sphere, the cube, and an infinity of other symmetrical shapes. However, since the only purpose of the exclusion volume is to yield the Clausius-Mosotti equation, which is shape independent, there is no point in finding all the acceptable shapes.

In the following section the computations are performed using a cubical exclusion volume.

5.3 <u>A Cubical Exclusion Volume</u>

The standard derivation of the Clausius-Mosotti relation used a spherical virtual cavity (see Section 5.1). However, the only reason for using the sphere is a general symmetry argument, and a cubical cavity should work equally well.

If the spherical cavity is replaced by the cubical cavity in the Clausius-Mosotti derivation, two of the field components may be affected. The first is the field due to the particles within the cavity. However, due to symmetry considerations (see the preceding section) this field component is zero for both the spherical cavity and the cubical cavity. The second field component is due to the bound charge on the virtual cavity surface. For a sphere, the resulting field was shown in Section 5.1 to be

$$\bar{E}_\ell = \frac{\varepsilon_r - 1}{3}\bar{E}$$

where $\bar{E}$ is the uniform macroscopic field. It will now be shown that the cube yields the same result. Assume a virtual cube in a uniform electric field $\bar{E} = E_0(-\hat{z})$ similar to Fig. 5.1. Then the bound charge, $\rho_b = -\nabla \cdot \bar{P}$, will accumulate on the top and bottom surfaces. We wish to find the E field due to this charge distribution at the center of the cube.

$$\bar{E}_\ell = \int_s - \frac{-\rho}{4\pi\varepsilon R^2}\hat{R}\, ds \quad ,$$

where $\bar{R}$ is measured from the center of the cube.

$$\bar{E} = \int_s \frac{\nabla \cdot \bar{P}}{4\pi\varepsilon R^2}\hat{R}\, ds$$

$$= \int_{top} + \frac{\bar{P}_m \cdot \hat{z}}{4\pi\varepsilon R^2}\hat{R}\, ds + \int_{bottom} - \frac{\bar{P}_m \hat{z}}{4\pi\varepsilon R^2}\hat{R}\, ds$$

where P_m denotes a "macroscopic" P. Since $\bar{P} \equiv 0$ inside the cube,

$$= \int_{\text{top}} - \frac{|\bar{P}_m|}{2\pi\varepsilon R^2} \hat{R}\, ds \;\cdot$$

Further, by symmetry

$$\bar{E}_\ell = E_{\ell z}\hat{z} = \hat{z} \int_{\text{top}} - \frac{|P_m|}{2\pi\varepsilon R^2} \hat{R}\cdot\hat{z}\, ds = -\frac{|P_m|}{2\pi\varepsilon} \hat{z} \int_{\text{top}} \frac{z}{R^3}\, ds$$

$$= + \frac{\varepsilon_r - 1}{2\pi} \bar{E} \cdot \int_{\text{top}} \frac{z}{R^3}\, ds \;\cdot$$

For a sphere,

$$\bar{E}_\ell = \frac{\varepsilon_r - 1}{3} \bar{E} \;.$$

So, if

$$\frac{2\pi}{3} = \int_{\text{top}} \frac{z}{R^3}\, ds$$

then the field in a cubical virtual cavity is the same as that in a spherical virtual cavity.

Let the dimensions of the cube be $2\ell \times 2\ell \times 2\ell$. Then

$$\int_{\text{top}} \frac{z}{R^3}\, ds = \int_{-\ell}^{\ell} \int_{-\ell}^{\ell} \frac{z}{R^3}\, dx\, dy \Big|_{z=\ell}$$

$$= \int_{-\ell}^{\ell} \int_{-\ell}^{\ell} \frac{z}{(x^2 + y^2 + z^2)^{3/2}}\, dx\, dy \Big|_{z=\ell} \;.$$

Then, from Gradshteyn and Ryzhik (1980, Sec. 2.271.5)

$$= \int_{-\ell}^{\ell} \frac{z}{y^2 + z^2} \frac{x}{\sqrt{x^2 + y^2 + z^2}} dy \Big|_{z=\ell}$$

$$= \int_{-\ell}^{\ell} \frac{2\ell^2}{y^2 + \ell^2} \frac{1}{\sqrt{y^2 + 2\ell^2}} dy = \int_{0}^{\ell} \frac{4\ell^2}{y^2 + \ell^2} \frac{1}{\sqrt{y^2+2\ell^2}} dy$$

Let

$$u^2 = \frac{y^2 + 2\ell^2}{\ell^2} \qquad u = \frac{1}{\ell} \sqrt{y^2 + 2\ell^2}$$

$$y^2 = u^2\ell^2 - 2\ell^2$$

$$y = \ell\sqrt{u^2 - 2} \qquad dy = \ell(u^2 - 2)^{-1/2} u \, du$$

Then

$$\int_{top} \frac{z}{R^3} ds = \int_{\sqrt{2}}^{\sqrt{3}} \frac{4\ell^2}{u^2\ell^2 - \ell^2} \frac{1}{\sqrt{u^2\ell^2}} u\ell \frac{1}{\sqrt{u^2 - 2}} du$$

$$= \int_{\sqrt{2}}^{\sqrt{3}} \frac{4}{u^2 - 1} \frac{1}{\sqrt{u^2 - 2}} du \, .$$

Expanding in partial fractions yields

$$= 2 \int_{\sqrt{2}}^{\sqrt{3}} \left[\frac{-1}{(x + 1)\sqrt{x^2 - 2}} + \frac{1}{(x - 1)\sqrt{x^2 - 2}} \right] dx$$

let $u = x + 1$ and $v = x - 1$

$$= 2\left\{ \int_{\sqrt{2}+1}^{\sqrt{3}+1} \frac{-1}{u\sqrt{u^2 - 2u - 1}}\, du + \int_{\sqrt{2}-1}^{\sqrt{3}-1} \frac{1}{v\sqrt{v^2 + 2v - 1}} \right\} dv \cdot$$

Then, again from Gradshteyn and Ryzhik (1980, Sec. 2.266)

$$= 2\left\{ -\sin^{-1}\left(\frac{-2 - 2x}{x\sqrt{8}}\right)\Bigg|_{\sqrt{2}+1}^{\sqrt{3}+1} + \sin^{-1}\left(\frac{-2 + 2x}{x\sqrt{8}}\right)\Bigg|_{\sqrt{2}-1}^{\sqrt{3}-1} \right\}$$

$$= 2\left\{ -\sin^{-1}\left(\frac{-2 - 2(\sqrt{3} + 1}{(\sqrt{3} + 1)\sqrt{8}}\right) + \sin^{-1}\left(\frac{-2 - 2(\sqrt{2} + 1))}{(\sqrt{2} + 1)\sqrt{8}}\right) \right.$$

$$\left. + \sin^{-1}\left(\frac{-2 + 2(\sqrt{3} - 1}{(\sqrt{3} - 1)\sqrt{8}}\right) - \sin^{-1}\left(\frac{-2 + 2(\sqrt{2} - 1))}{(\sqrt{2} - 1)\sqrt{8}}\right) \right\}$$

$$= 2\left\{ -\sin^{-1}\left|\frac{-2 - \sqrt{3}}{\sqrt{2} + \sqrt{6}}\right| + \sin^{-1}\left|\frac{-2 + \sqrt{3}}{-\sqrt{2} + \sqrt{6}}\right| + \sin^{-1}(-1) - \sin^{-1}(-1) \right\}$$

$$= 2\left\{ + \frac{5\pi}{12} + - \frac{\pi}{12} \right\} = 2\left(\frac{\pi}{3}\right) = \left(\frac{2\pi}{3}\right) .$$

Thus the spherical and cubical exclusion volumes yield the same result.

5.4 Restriction of the Clausius-Mosotti Equation to Distribution of Plates

The Clausius-Mosotti relation is valid for any sparse distribution of particles. Thus

$$\varepsilon_r = 1 + \frac{N\alpha}{\varepsilon_0 \left(1 - \frac{N\alpha}{3}\right)}$$

may be used with α obtained from the algorithms presented in Chapters III and IV. The validity of the formula can be verified for very thin plates in a dense cubical distribution.

$$\alpha = (\tau - 1)v = (\tau - 1)\, t\ell^2$$

for excitation parallel to the plate and

$$\alpha = \frac{\tau - 1}{\tau} v = \frac{\tau - 1}{\tau}\, t\ell^2$$

for excitation normal to the plate, where the particle is assumed square with dimensions ℓ x ℓ x t. The above formulae were obtained in Chapter IV with a small scattered field approximation which is satisfied as long as $\tau t/\ell << 1$. If the cubical distribution of plates is now made dense so that the plates form a multiple layer dielectric, then ε_r as seen under tangential excitation would be

$$\varepsilon_r = \tau \frac{t}{\ell} + \left(1 - \frac{t}{\ell}\right)$$

and for normal excitation

$$\varepsilon_r = \frac{1}{\left(1 - \frac{t}{\ell}\right) + \frac{t}{\ell}\frac{1}{\tau}} .$$

For parallel excitation,

$$N\alpha = N(\tau - 1)t\ell^2 = (\tau - 1)\frac{t}{\ell}$$

$$\varepsilon_r = 1 + \frac{3(\tau - 1)t/\ell}{3-(\tau - 1)t/\ell} \approx 1 + (\tau - 1)t/\ell$$

$$= (1 - t/\ell) + \tau(t/\ell)$$

as expected. For normal excitation

$$\varepsilon_r = 1 + \frac{3\frac{\tau - 1}{\tau}\frac{t}{\ell}}{3 - \frac{\tau - 1}{\tau}\frac{t}{\ell}}$$

$$= \frac{3 - \frac{\tau - 1}{\tau}\frac{t}{\ell} + 3\frac{\tau - 1}{\tau}\frac{t}{\ell}}{3 - \frac{\tau - 1}{\tau}\frac{t}{\ell}}$$

$$= \left[\frac{3 - \frac{\tau - 1}{\tau}\frac{t}{\ell}}{3 + 2\frac{\tau - 1}{\tau}\frac{t}{\ell}}\right]^{-1}$$

$$= \left[\frac{\left(1 - \frac{t}{\ell}\right)\left(3 + 2\frac{\tau - 1}{\tau}\frac{t}{\ell}\right) + \frac{t}{\ell}\left(3 + 2\frac{\tau - 1}{\tau}\frac{t}{\ell}\right) - 3\frac{\tau - 1}{\tau}\frac{t}{\ell}}{3 + 2\frac{\tau - 1}{\tau}\frac{t}{\ell}}\right]^{-1}$$

$$= \left[\left(1 - \frac{t}{\ell}\right) + \frac{t}{\ell}\,\frac{3 + 2\frac{\tau - 1}{\tau}\frac{t}{\ell} - 3\frac{\tau - 1}{\tau}}{3 + 2\frac{\tau - 1}{\tau}\frac{t}{\ell}}\right]^{-1}$$

$$= \left[\left(1 - \frac{t}{\ell}\right) + \frac{t}{\ell}\,\frac{1}{\tau}\,\frac{3\tau - 3\tau + 3 + 2(\tau - 1)t/\ell}{3 + 2\frac{\tau - 1}{\tau}\frac{t}{\ell}}\right]^{-1}$$

$$= \left[\left(1 - \frac{t}{\ell}\right) + \frac{t}{\ell}\,\frac{1}{\tau}\,\frac{3 + 2(\tau - 1)t/\ell}{3 + 2\left(\frac{\tau - 1}{\tau}\right)\frac{t}{\ell}}\right]^{-1} \approx \frac{1}{\left(1 - \frac{t}{\ell}\right) + \frac{t}{\ell}\,\frac{1}{\tau}}$$

as expected.

Thus the effective permittivity of a dense cubical distribution of thin plates as predicted by the Clausius-Mosotti theory is in agreement with the exact average permittivity of a layered dielectric when excited either normally or tangentially to the plane of the layers. Further, the Clausius-Mosotti theory is entirely adequate for describing scattering by distributions of particles as long as the electrical interaction between the particles may be accurately described with only the dipole term. The low frequency expansion permits the Clausius-Mosotti formula to be used with dynamic electromagnetic radiation as well as for the electrostatic and magnetostatic problems. For dynamic radiation, the dipole moment may contain an imaginary component which then results in an imaginary component in the effective permittivity. On a macroscopic level this corresponds to a lossy medium. Since these results can also be applied to magnetic scattering, the entire distribution can then be characterized by complex electric and magnetic polarization tensors. Although the analysis of electromagnetic propagation through a complex tensor medium is not particularly simple, it is trivial compared to the problem of analyzing propagation through a distribution of particles by computing the scattering of the electromagnetic wave from each individual particle in the distribution.

CHAPTER VI. CONCLUSIONS

The problem of constructing the low frequency expansion which is valid for scattering by an open surface has been solved with the simultaneous solution of a problem from classical physics, find $\bar{F}$ given $\nabla \times \bar{F} = \bar{f}$. A highly efficient and accurate numerical program has been developed to solve the problem of static scattering from a thin plate. The potential on flat plates with thickness to length ratios less than 0.1 and arbitrary shape may be solved and the resulting dipole moments computed, with the plate being described via an arbitrary number of triangular elements with arbitrary shape and size. This flexibility permits the description of an arbitrary plate with a minimum number of elements. Varying the size of the elements is important near the edges of a plate, primarily when computing the dipole moments associated with normal excitation of the plate. The range of operation of the program is very broad. The permittivity can assume a wide range of values, real and complex, small and large. The result is that the scattering from a particle composed of any homogeneous material can be analyzed, including perfect conductors. The thickness of the material can also assume a wide range of values, and the program has been successfully tested for length to thickness ratios ranging from 1:1 to in excess of 10^6:1.

Finally, distributions of particles have been discussed, and the completeness of the Clausius-Mosotti-Lorentz-Lorenz formulation has been shown. The validity of this formulation for a cubical dense distribution of thin plates has also been shown. Although the content of Chapter V was restricted to sparse (particle volume as a percentage of total volume) distributions of particles, an area of considerable interest is dense distributions of particles. The primary difference between dense and non-dense scattering theories is that sparse scattering theories assume the particle interaction is limited to far-field interaction. In Section 5.4 the thin plate approximation for the dipole moment guaranteed that the scattered field was negligible and that only far field interactions were being considered. There have been several attempts to modify the Clausius-Mosotti-Lorentz-Lorenz formula to account for non-sparse distributions. The theories are designed to account for experimental results on dense distributions of particles and in general they may be characterized as attempting to describe higher order interactions (quadrupole, etc.) without specifically calculating these components. Discussions of the relative merits of these methods are contained in Granqvist and Hunderi (1977) and Bohren and Battan (1980).

Future work may use the program developed for this dissertation to investigate the resonance region associated with thin plates. To this end, the matrix problem solved might be decomposed into an eigenfunction expansion. This would permit the simultaneous calculation of the dipole moments associated with a particular shape for all values of τ. The linear predictor described in Chapter IV, although quite satisfactory for the purpose of determining when to use the full multi-element program, could perhaps be further refined by modifying the testing function. Specifically, it might be desirable to use a Galerkin's method approach (e.g., Harrington, 1982). Although the results presented in Chapter IV are believed accurate and show qualitative agreement with the results of Senior (1975) and Herrick (1976), it would be useful to verify some of the thin plate calculations for which the program was developed (length to thickness ratios greater than 10), either experimentally or numerically. Efforts are currently underway to obtain verification of the results using a version of the program developed by Senior and Willis (1982) specifically modified to permit the accurate analysis of thin rotationally symmetric plates (Weil, 1984).

It is hoped that the developed program (a listing of which is provided in the Appendix will be incorporated into emerging theories on low frequency scattering by distributions of particles. Since the shape of a particle affects the dipole moments which are associated with a particle which in turn determines parameters such as the effective dielectric constant of the distribution, the developed program should provide a valuable tool in determining the electrical properties of particle distributions. As shape becomes extremely important in the resonance region, the shape of the particle may aid in the identification of different particles in cases where the particles are illuminated with relatively low frequency electromagnetic radiation. This may be particularly useful in regions where identification through high frequency electromagnetic radiation, which relies primarily on shape effects which become apparent only when the particle size is comparable to a wavelength, is either not possible or inconvenient.

APPENDIX

PROGRAM LISTING

```
/* function definitions */
#include <math.h>
double ppcon(), ptcon(),u(),v(),ur(),vr(),intdvz(),intdz(),msqrt(),mlog(),
        area(),sumang();
/* macro definitions */
#define Darray(Ara,I1,I2,L1,L2) Ara[(I1)+(I2)*(L1)]  /* Macro to generate other\
                                                      macros to mimic fortran\
                                                      arrays   */
#define Mnumpoi 100 /* maximum number of points */
#define Mnumtri 100 /* maximum number of triangular patches */
#define Mnumatri 10 /* maximum number of triangles which may share a \
        common point, 6 is average for internal points, 8 accounts for\
        most schemes, thus 10 is a reasonable upper bound */
#define True 1 /* value of logical true */
#define False 0 /* value of logical false */
#define Tri(J1,J2) Darray(tri,J1,J2,Mnumatri,Mnumpoi)  /* set up tri as two\
                     dimensional array with limits (Mnumatri, Mnumpoi) */
#define Poi1(J1,J2) Darray(poi1,J1,J2,Mnumatri,Mnumpoi)  /* set up poi1 as two\
                     dimensional array with limits (Mnumatri, Mnumpoi) */
#define Poi2(J1,J2) Darray(poi2,J1,J2,Mnumatri,Mnumpoi)  /* set up poi2 as two\
                     dimensional array with limits (Mnumatri, Mnumpoi) */
#define Poi(J1,J2) Darray(poi,J1,J2,Mnumtri,3) /* set up poi as two dimensional\
                     array with limits (Mnumtri,3)  */
#define Matrix(J1,J2) Darray(matrix,J1,J2,Mnumpoi,Mnumpoi)  /* set up matrix as\
                     two dimensional array with limits (Mnumpoi,Mnumpoi) */
#define Cmatrix(J1,J2) Darray(cmatrix,J1,J2,Mnumpoi,Mnumpoi)  /* complex
                           version of Matrix, used with the complex
                           structure */
struct complex {
        double real;
        double imag;
        };
/*                                                                        */
/*                                                                        */
/*  End of Definitions / Beginning of external variable declarations     */
/*                                                                        */
/*                                                                       */
int flagsym; /* flagsym may assume values of 0,1,2, or 3.  Flagsym is used
                by low level routines to incorporate symmetry in a manner
                invisible to the rest of the program.
                0 - indicates no symmetry
                1 - indicates object possesses mirror symmetry about x=0
                2 - indicates object possesses mirror symmetry about y=0
                3 - indicates object possesses mirror symmetry about x=0 and y=0
                Flagsym is set at the beginning of the main program and after
                that is never changed                                    */
int flagsyms; /* flagsyms is used in conjunction with flagsym to generate
                 mirror images of the source patches while calculating
                 contributions to the field point.  The initial object
                 description is assumed to lie in the first quadrant, however
                 if flagsym is 0, this is not necessary.  Flagsyms is always
                 less than or equal to flagsym, and may assume the values of
                 0,1,2, or 3.  These values are given the following meanings:
                 0 - source patch is original patch (presumably first quadrant)
                 1 - source patch is in second quadrant
                 2 - source patch is in fourth quadrant
                 3 - source patch is in third quadrant   */
int plotsym;  /* plotsym is used to denote what type of mirroring is desired
                 in the perspective plot.
                 0 - no mirroring
                 1 - mirror about x=0
                 2 - mirror about y=0
                 3 - mirror about x=0 and y=0
                 Any mirroring that is done must be consistent with the
                 mirroring specified in generating the plate.  plotsym
                 is specified in rgrph.   */
int eof; /* end of file variable (=0 means end of file) */
int potflag; /* flag marks whether or not potentials have been computed
                0 - potential has not been computed
                1 - potential has been computed assuming x excitation
                2 - potential has been computed assuming y excitation
                3 - potential has been computed assuming z excitation  */
struct {
        double x[Mnumpoi];
        double y[Mnumpoi];
        double poten[Mnumpoi];
        struct complex cpoten[Mnumpoi]; /* complex version of poten */
        }point ;    /* point is a structure which contains the locations of all
                 of the points used in defining the triangular patches.  A point
                 number 0 is permitted as C defines arrays beginning with 0.
                 X and y are the coordinates of the points, and
                 poten is the potential of each point, as determined from the
                 matrix problem.    */
struct {
        int numatri[Mnumpoi];
        int Tri(Mnumatri,Mnumpoi-1);
        int Poi1(Mnumatri,Mnumpoi-1);
        int Poi2(Mnumatri,Mnumpoi-1);
        } apoint;  /* apoint is a structure which contains lists of triangles
                 associated with each point.  numatri contains the number of
                 triangles associated with each point; tri contains the list of
                 triangle numbers associated with each point; and poi1 and poi2
                 contain the other two vertices used in defining the triangle.
                 poi1 and poi2 are indices to point.  tri is an index to
                 triang.      */
struct {
        struct {
                double x[Mnumtri];
                double y[Mnumtri];
                } centroid;
        int Poi(Mnumtri,3-1);
        double poten[Mnumtri];
        struct complex cpoten[Mnumtri]; /* complex version of poten */
        } triang;   /*  triang is used to solve the scattering problem with z
                    excitation.  centroid .x and .y contain the coordinates
                    of the centroid of each triangle.  poi contains the list of
                    points (vertices) associated with each triangle, and is an
                    index to point.  poten is the potential of each triangle as
                    determined from solution of the matrix problem.    */
double Matrix(Mnumpoi,Mnumpoi-1);  /* This is "THE"  matrix.  Trivial points
                                      (ie, those which have zero potential from
                                      symmetry considerations) are skipped when
                                      reading or filling the matrix, which is
                                      always done sequentially.    */
struct complex Cmatrix(Mnumpoi,Mnumpoi-1);  /* complex version of Matrix */
double fvect[Mnumpoi];   /* fvect is the forcing vector for the matrix problem
                            and after solution of the matrix problem contains
                            the solution vector.   */
struct complex cfvect[Mnumpoi];  /* complex version of fvect */
int mnumpoi;  /* This is the total number of points.  Points are assumed to be
                 numbered sequentially from 0.  mnumpoi <= Mnumpoi. */
int mnumtri;  /* This is the total number of triangles.  Triangles are assumed
                 to be numbered sequentially from 0.  mnumtri <= Mnumtri. */
union {
        char str[2];
        char let;
        } com;
              /* This is the first character of each input line, and is
                 interpreted as a command.  Valid commands are:
                        d - display linkup of points and triangles
                                and display potentials if defined
                        e - specify direction of exciting field,
                                and solve the resultant matrix problem.
                        g - graph potentials of points/triangles
                        h - enter heading, ie, a descriptive one line title.
                        m - enter material parameters of plate: t and tau.
                        p - enter coordinates of next point
                        r - regenerate matrix problem using finer mesh
                        s - define symmetry to be assumed in solving problem
                        t - enter definition of next triangle       */
char hedstr[82];  /*  contains one line heading, description of data */
double t; /* Thickness of the plate */
double tau; /* permittivity of the plate */
double taui;  /* imaginary part of the permittivity.  This variable is also
                used as a flag: if taui is zero, computations are performed
                assuming "tau" is purely real; if taui is non-zero, the routines
                appropriate for a complex tau are invoked  */
double dipmom;  /* contains the real part of the computed dipole moment */
double dipmomi;  /* contains the imaginary part of the computed dipole moment */
double Epsilon = 1e-10; /* A very small number, used for approximate equality */
double Pi = 3.1415926535; /* Pi */
/******************************************************************************/
/*                                                                            */
/*  M A I N     P R O G R A M                                                 */
/*                                                                            */
/*  This is a program which calculates scattering by an arbitrarily shaped,   */
/*  thin, flat, dielectric plate.  Excitation may be specified in the x, y,   */
/*  or z directions.  The plate is made up of an arbitrary number (nominally  */
/*  less than 50) of triangular patches, upon which a method of moments       */
/*  solution is obtained.  Contributions from each patch is calculated via    */
/*  surface integrals which are evaluated analytically, thus contributions    */
/*  from each patch is obtained exactly (at least to 10 plus digits).  The    */
/*  only approximations which are made are that the potential varies linearly */
/*  with z inside the plate, and the division of the plate into triangular    */
/*  patches, inside of which the field varies linearly with x and y.  The     */
/*  shape and size of the triangular patches are completely arbitrary, and it */
/*  is noted that the patches need not be contiguous.  To speed computations, */
/*  and to enhance that accuracy obtainable with a given number of patches,   */
/*  the user may specify that the plate possesses mirror symmetry about x=0,  */
/*  y=0, or both x=0 and y=0.                                                 */
/*                                                                            */
/*******************************************************************************/
main() {
hedstr[0] = '\0';
for(eof=scanf("%1s",com.str);eof == 1;) {
        switch (com.let) {
                case 'P':
                case 'p':
                case 'T':
                case 't':
                        rdata();  /* read in the data */
                        continue;
                case 'D':
                case 'd':
                        ddata();  /* display the data */
                        break;
                case 'G':
                case 'g':
                        rgrph();  /* read in mirroring specification */
                        break;
                case 'H':
                case 'h':
                        gethed();
                        break;
                case 'M':
                case 'm':
                        rmp();  /* read in material parameters */
                        break;
                case 'S':
                case 's':
                        rsym();   /* read in the symmetry of the plate */
                        break;
                case 'E':
                        eigen();  /* solve eigenvalue problem */
                        break;
                case 'e':
                        rexcit(); /* read in direction of excitation
                                     and solve the resulting matrix problem */
                        switch (potflag) {
                                case 1:
                                        xsolv();
                                        pexc();
                                        break;
                                case 2:
                                        ysolv();
                                        pexc();
                                        break;
                                case 3:
                                        zsolv();
                                        pexc();
                                        break;
                                default:
                                        printf("invalid excitation");
                                        break;
                                }
                        break;
                default:
                        printf("%c is an invalid command\n",com.let);
                        while(scanf("%c",com.str),com.let != '\n');
                                  /* flush out bad command */
                        break;
                }
        eof=scanf("%1s",com.str);
        }
}
/*****************************************************************************/
/*                                                                           */
/*  ROUTINE:   ddata                                                         */
/*                                                                           */
/*  ddata is called by the main program to display data.  ddata display      */
/*  locations of points, connections of points, and potentials of points if  */
/*  they have been computed.  ddata also displays status of various flags    */
/*  which indicate symmetries of plate and direction of excitation.          */
/*                                                                           */
/*****************************************************************************/
ddata() {
int i,j; /* loop variables */
if (t != 0) printf("t = %lf, tau = %lf + i %lf \n",t,tau,taui);
switch (potflag) {
        case 0:
                printf("potential has not been computed\n");
                break;
        case 1:
                printf("potential has been computed assuming x excitation\n");
                break;
        case 2:
                printf("potential has been computed assuming y excitation\n");
                break;
        case 3:
                printf("potential has been computed assuming z excitation\n");
                break;
        }
switch (flagsym) {
        case 0:
                printf("plate is not symmetric\n");
                break;
        case 1:
                printf("plate has mirror symmetry about x=0\n");
                break;
        case 2:
                printf("plate has mirror symmetry about y=0\n");
                break;
        case 3:
                printf("plate has mirror symmetry about x=0 and y=0\n");
                break;
        }
for (i=0; i<mnumpoi; i++) {
        printf("point # %d is located at x = %lf, y = %lf\n",i,point.x[i],
          point.y[i]);
        if (potflag) {
                if(!taui) printf("      and has potential %e\n",point.poten[i]);
                else printf("      and has potential %lf + i %lf\n",
                        point.cpoten[i].real,point.cpoten[i].imag);
                }
        printf("point %d is associated with points and triangles (tri,P1,P2)\n"
          ,i);
        for (j=0; j<apoint.numatri[i];j++) {
                printf("(%d,%d,%d),  ",apoint.Tri(j,i), apoint.Poi1(j,i),
                  apoint.Poi2(j,i));
                if (j%6 == 5 || j == apoint.numatri[i]-1) printf("\n");
                }
        }
}
/************************************************************************/
/*                                                                      */
/*  ROUTINE:   rsym                                                     */
/*                                                                      */
/*  rsym is called by the main program to read in the symmetry of the plate.  */
/*  rsym assumes no symmetry if response is inappropriate.              */
/*                                                                      */
```

```
/*******************************************************************************/
rsym() {
scanf("%d",&flagsym);
switch(flagsym) {
        default: flagsym=0;
        case 1:
        case 2:
        case 3:
                ;
        }
}
/*******************************************************************************/
/*                                                                             */
/*  ROUTINE:   rgrph                                                           */
/*                                                                             */
/*  rgraph is called by the main program to read in the mirroring desired.    */
/*  rgrph then calls graph to produce the graph.  plotsym is enforced to      */
/*  be consistent with flagsym.                                                */
/*                                                                             */
/*******************************************************************************/
rgrph() {
scanf("%d",&plotsym);
switch(plotsym) {
        default: plotsym=0;
        case 1:
        case 2:
        case 3:
                ;
        }
        plotsym = plotsym & flagsym;
        graph();
}
/*******************************************************************************/
/*                                                                             */
/*  ROUTINE:   rmp                                                             */
/*                                                                             */
/*  rmp is called by the main program to read in the material parameters of   */
/*  the plate, specifically thickness and permittivity.                        */
/*                                                                             */
/*******************************************************************************/
rmp() {
scanf("%lf %lf %lf",&t,&tau,&taui);
dipmomi = 0;
}
```

```
/*****************************************************************************/
/*                                                                           */
/*  ROUTINE:   rexcit                                                        */
/*                                                                           */
/*  rexcit is called by the main program to read in the direction of        */
/*  excitation.  rexcit assumes x excitation in the event of an invalid     */
/*  response.                                                                */
/*                                                                           */
/*****************************************************************************/
rexcit() {
char charscr[2]; /* scratch variable, contains direction of excitation */
scanf("%1s",charscr);
switch (*charscr) {
        default:
                potflag = 0;
                break;
        case 'X':
        case 'x':
                potflag = 1;
                break;
        case 'Y':
        case 'y':
                potflag = 2;
                break;
        case 'Z':
        case 'z':
                potflag = 3;
                break;
        }
}

/*****************************************************************************/
/*                                                                           */
/*  ROUTINE:   xsolv                                                         */
/*                                                                           */
/*  xsolv is called by main to fill THE matrix.  xsolv assumes excitation is */
/*  in the x direction.  Points which lie on x=0 are assumed to have zero    */
/*  potential and are skipped in both the row and column loops.  The outside */
/*  loop is the column loop and is associated with the field point.  The     */
/*  inside loop is associated with the source point.  Images of the source   */
/*  point are not apparent at this level and are handled in ppcon.          */
/*      After the matrix and forcing vector are filled, solvem is called to */
/*  the matrix problem.  The solution vector is then read in the same manner */
/*  it was generated, ie, trivial points are zeroed and skipped in reading   */
/*  the solution vector.
/*                                                                             */
/*******************************************************************************/
xsolv() {
int flagtriv, flagtrif;  /* flagtriv and flagtrif are used by the source loop
                            and the field loop respectively to flag trivial
                            points.  Values are either True (defined above as
                            1) or False: 0.     */
int ipois, ipoif;        /* ipois and ipoif denote the source and field points
                            respectively.  This is in contradistinction to irow
                            and icol.    */
int irow, icol;          /* icol and irow denote positions within THE matrix.
                            If x symmetry is not being used, these variables
                            will contain the same values as ipois and ipoif.
                            If x symmetry is being used, then icol and irow will
                            gradually fall behind ipois and ipoif as trivial
                            points are encountered.    */
for ( flagtrif = False, ipoif = 0, irow = 0; ipoif < mnumpoi; ipoif++, irow +=
  !flagtrif, flagtrif = False ) {
        if (flagsym & 1 && !point.x[ipoif]) {flagtrif = True; continue;}
        for ( flagtriv=False,ipois=0,icol=0;ipois < mnumpoi; ipois++, icol +=
          !flagtriv, flagtriv=False) {
                if (flagsym & 1 && !point.x[ipois]) {flagtriv=True; continue;}
                if (!taui) Matrix(irow,icol)=(tau-1)*ppcon(ipoif,ipois);
                else {
                        Cmatrix(irow,icol).real=(tau-1)*ppcon(ipoif,ipois);
                        Cmatrix(irow,icol).imag= taui*ppcon(ipoif,ipois);
                        }
                if (irow == icol) {
                        if (!taui) Matrix(irow,icol) += 1;
                        else Cmatrix(irow,icol).real += 1;
                        }
                }
        if (!taui) fvect[irow]=point.x[ipoif];
        else {
                cfvect[irow].real=point.x[ipoif];
                cfvect[irow].imag=0;
                }
        }
if (!taui) {
        pmat();
        pvec();
        }
solvem(irow);
```

```
if (!taui) {
        pmat();
        pvec();
        }
for ( flagtrif=False, ipoif=0, irow=0; ipoif < mnumpoi; ipoif++, irow +=
  !flagtrif, flagtrif = False ) {
        if (flagsym & 1 && !point.x[ipoif]) flagtrif = True;
        if (!taui) point.poten[ipoif] = (flagtrif) ? 0. : fvect[irow];
        else {
                point.cpoten[ipoif].real = (flagtrif)?0.:cfvect[irow].real;
                point.cpoten[ipoif].imag = (flagtrif)?0.:cfvect[irow].imag;
                }
        }
}
/***************************************************************************/
/*                                                                         */
/*  ROUTINE:   ysolv                                                       */
/*                                                                         */
/*  ysolv is called by main to fill THE matrix.  ysolv assumes excitation is  */
/*  in the y direction.  Points which lie on y=0 are assumed to have zero     */
/*  potential and are skipped in both the row and column loops.  The outside  */
/*  loop is the column loop and is associated with the field point.  The      */
/*  inside loop is associated with the source point.  Images of the source    */
/*  point are not apparent at this level and are handled in ppcon.            */
/*      After the matrix and forcing vector are filled, solvem is called to   */
/*  the matrix problem.  The solution vector is then read in the same manner  */
/*  it was generated, ie, trivial points are zeroed and skipped in reading    */
/*  the solution vector.
/*                                                                         */
/***************************************************************************/
ysolv() {
int flagtriv, flagtrif;  /* flagtriv and flagtrif are used by the source loop
                            and the field loop respectively to flag trivial
                            points.  Values are either True (defined above as
                            1) or False: 0.     */
int ipois, ipoif;        /* ipois and ipoif denote the source and field points
                            respectively.  This is in contradistinction to irow
                            and icol.    */
int irow, icol;          /* icol and irow denote positions within THE matrix.
                            If y symmetry is not being used, these variables
                            will contain the same values as ipois and ipoif.
                            If y symmetry is being used, then icol and irow will
                            gradually fall behind ipois and ipoif as trivial
                            points are encountered.    */
```

```
for ( flagtrif=False, ipoif=0, irow=0; ipoif < mnumpoi; ipoif++, irow +=
  !flagtrif, flagtrif = False ) {
        if (flagsym & 2 && !point.y[ipoif]) {flagtrif = True; continue;}
        for ( flagtriv=False,ipois=0,icol=0;ipois < mnumpoi; ipois++, icol +=
          !flagtriv, flagtriv=False) {
                if (flagsym & 2 && !point.y[ipois]) {flagtriv=True; continue;}
                if (!taui) Matrix(irow,icol)=(tau-1)*ppcon(ipoif,ipois);
                else {
                        Cmatrix(irow,icol).real=(tau-1)*ppcon(ipoif,ipois);
                        Cmatrix(irow,icol).imag=taui*ppcon(ipoif,ipois);
                        }
                if (irow == icol) {
                        if (!taui) Matrix(irow,icol) += 1;
                        else Cmatrix(irow,icol).real += 1;
                        }
                }
        if (!taui) fvect[irow]=point.y[ipoif];
        else {
                cfvect[irow].real=point.y[ipoif];
                cfvect[irow].imag=0;
                }
        }
solvem(irow);
for ( flagtrif=False, ipoif=0, irow=0; ipoif < mnumpoi; ipoif++, irow +=
  !flagtrif, flagtrif = False ) {
        if (flagsym & 2 && !point.y[ipoif]) flagtrif = True;
        if (!taui) point.poten[ipoif] = (flagtrif) ? 0. : fvect[irow];
        else {
                point.cpoten[ipoif].real=(flagtrif)?0.:cfvect[irow].real;
                point.cpoten[ipoif].imag=(flagtrif)?0.:cfvect[irow].imag;
                }
        }
}
/******************************************************************************/
/*                                                                            */
/*  ROUTINE:   ppcon(ipoif,ipois)                                             */
/*                                                                            */
/*  ppcon is called by xsolv and ysolv to calculate the point to point       */
/*  contribution.  ppcon processes all triangles associated with the source  */
/*  point.  Imaging of the source patches due to symmetry is not explicitly  */
/*  performed by pppcon but is handled by lower level routines.  All         */
/*  processing is done by calling ptcon                                       */
/*                                                                            */
/******************************************************************************/
```

```
double ppcon(ipoif,ipois)
int ipoif, ipois; /* field point index, source point index */
{
int i; /* loop variable */
double sum; /* scratch variable, holds sum of contributions */
for (i=0,sum=0; i<apoint.numatri[ipois]; i++)
        sum += ptcon(ipoif, ipois, apoint.Poi1(i,ipois),
          apoint.Poi2(i,ipois));
return(sum/(4*Pi));
}
/********************************************************************************/
/*                                                                              */
/*  ROUTINE:   ptcon(ipoif,ipois,ipoi1,ipoi2)                                   */
/*                                                                              */
/*  ptcon calculates the contribution from a source triangle to a field         */
/*  point.  The source triangle is delimited by the vertices associated with    */
/*  ipois, ipoi1, and ipoi2.  Depending upon the shape of the triangle, the     */
/*  field contribution is calculated using one of three formulas.               */
/*  The coordinates of all points are transformed to the u-v coordinate         */
/*  system using the functions u, v, ur, and vr.  ur and vr are used to         */
/*  calculate coordinates of patches in reflected quadrants.  traninit is       */
/*  used to initialize the translation.                                         */
/*                                                                              */
/********************************************************************************/
double ptcon(ipoif,ipois,ipoi1,ipoi2)
int ipoif, ipois, ipoi1, ipoi2;  /* indices for: field point, source point, and
                                    two delimiting points of source triangle */
{
int i; /* scratch variable */
double bsum; /* stores sum of contribution from first quadrant and all reflected
           quadrants */
double sum;  /* stores line integral contributions to surface integral */
for (flagsyms=0,bsum=0; flagsyms<=flagsym; flagsyms += (flagsym&1)?1:2) {
        traninit(ipoi1,ipoi2);
        if (ur(ipois) < 0) {
                i=ipoi1;
                ipoi1=ipoi2;
                ipoi2=i;
                traninit(ipoi1,ipoi2);
                }
        sum = (fabs(ur(ipois)) * Epsilon > fabs(vr(ipois)))?0.:
                 intdvz(ur(ipois)/vr(ipois),0.,0.,vr(ipois),ipoif);
        sum += (fabs(ur(ipois)) * Epsilon > fabs(vr(ipoi2)-vr(ipois)))?0.:
                 intdvz(-ur(ipois)/(vr(ipoi2)-vr(ipois)),ur(ipois)*vr(ipoi2)
                        /(vr(ipoi2)-vr(ipois)),vr(ipois),vr(ipoi2),ipoif);
        sum -=   intdvz(0.,0.,0.,vr(ipoi2),ipoif);
        sum /= fabs(ur(ipois));
        bsum += (potflag != 3 && potflag & flagsyms) ? -sum : sum;
        }
return(bsum);
}
/*******************************************************************************/
/*                                                                             */
/*   Additional External Variables                                             */
/*                                                                             */
/*   Used in routines: traninit,u,v,ur,vr                                      */
/*                                                                             */
/*******************************************************************************/
double x0,my0,calp,salp; /* variables used in mapping to u-v coordinate system,
                           x0, and my0 are the coordinates of the origin, and
                           calp and salp are the cosine and sine of the angle
                           alpha used to rotate the x-y coordinate system.     */
                           /*  myo is used instead of y0 so as to not be
                               confused with the system supplied bessel
                               function which is accessed by y0.    */

/*******************************************************************************/
/*                                                                             */
/*  ROUTINE:   traninit(ipoi1,ipoi2)                                           */
/*                                                                             */
/*  traninit initializes the translation routines u,v,ur, and vr.  Initial     */
/*  calculations are made to obtain the appropriate displacements and          */
/*  rotations which will be needed in the translation routines.                */
/*                                                                             */
/*******************************************************************************/
traninit(ipoi1,ipoi2)
int ipoi1,ipoi2; /* indices to two auxiliary points used in defining source
                    source triangle.  u-v coordinate system will be centered
                    on the point associated with ipoi1, with the point
                    associated with ipoi2 lying on line u=0.     */
{
double alp;  /* this is the angle of rotation */
switch (flagsyms) {
        case 0: x0=point.x[ipoi1];
                my0=point.y[ipoi1];
                alp=atan2(point.y[ipoi2]-my0,point.x[ipoi2]-x0);
                calp=cos(alp);
                salp=sin(alp);
                break;
        case 1: x0= -point.x[ipoi1];
                my0=point.y[ipoi1];
                alp=atan2(point.y[ipoi2]-my0,-point.x[ipoi2]-x0);
                calp=cos(alp);
                salp=sin(alp);
                break;
        case 2: x0=point.x[ipoi1];
                my0= -point.y[ipoi1];
                alp=atan2(-point.y[ipoi2]-my0,point.x[ipoi2]-x0);
                calp=cos(alp);
                salp=sin(alp);
                break;
        case 3: x0= -point.x[ipoi1];
                my0= -point.y[ipoi1];
                alp=atan2(-point.y[ipoi2]-my0,-point.x[ipoi2]-x0);
                calp=cos(alp);
                salp=sin(alp);
                break;
        }
return;
}
/******************************************************************************/
/*                                                                            */
/*  ROUTINES:   u(i),v(i),ur(i),vr(i)                                         */
/*                                                                            */
/*  These routines are used to calculate coordinates of points in the u-v     */
/*  coordinate system.  i is an index to point.  u and v return the           */
/*  coordinates of the point, and ur and vr return the coordinates of an      */
/*  image of the point, the location of the image depending on the value of   */
/*  flagsyms.                                                                 */
/*                                                                            */
/******************************************************************************/
double u(i)
int i; /* index to point */
{
return((point.x[i]-x0)*(-salp)+(point.y[i]-my0)*calp);
}
double v(i)
int i; /* index to point */
{
return((point.x[i]-x0)*calp+(point.y[i]-my0)*salp);
}
double ur(i)
```

```
int i; /* index to point */
{
switch (flagsyms) {
        case 0: return((point.x[i]-x0)*(-salp)+(point.y[i]-my0)*calp);
        case 1: return((-point.x[i]-x0)*(-salp)+(point.y[i]-my0)*calp);
        case 2: return((point.x[i]-x0)*(-salp)+(-point.y[i]-my0)*calp);
        case 3: return((-point.x[i]-x0)*(-salp)+(-point.y[i]-my0)*calp);
        }
}
double vr(i)
int i; /* index to point */
{
switch (flagsyms) {
        case 0: return((point.x[i]-x0)*calp+(point.y[i]-my0)*salp);
        case 1: return((-point.x[i]-x0)*calp+(point.y[i]-my0)*salp);
        case 2: return((point.x[i]-x0)*calp+(-point.y[i]-my0)*salp);
        case 3: return((-point.x[i]-x0)*calp+(-point.y[i]-my0)*salp);
        }
}
/****************************************************************************/
/*                                                                          */
/*  ROUTINE:   rdata                                                        */
/*                                                                          */
/*  rdata is called by the main program to read in point and triangle       */
/*  definitions.  Upon encountering a non- "p" or "t" command, rdata calls  */
/*  ldata to link the data, and then returns to the main program.           */
/*                                                                          */
/****************************************************************************/
rdata() {
int i; /* loop variable */
int inumb; /* number of next point or triangle */
do {
        switch (com.let) {
                case 'P':
                case 'p':
                        scanf("%d",&inumb);
                        scanf("%le %le",point.x+inumb,point.y+inumb);
                        mnumpoi=inumb+1;
                        continue;
                case 'T':
                case 't':
                        scanf("%d",&inumb);
                        for (i=0;i<3;i++) scanf("%d",&triang.Poi(inumb,i));
                        mnumtri=inumb+1;
                        continue;
                default:
                        ldata();
                        return;
                }
        }
        while (scanf("%1s",com.str) == 1);
}
/******************************************************************************/
/*                                                                            */
/*  ROUTINE:   ldata                                                          */
/*                                                                            */
/*  ldata links all information concerning points and triangles.  This      */
/*  information is needed to precisely define each patch on the plate.       */
/*                                                                            */
/******************************************************************************/
ldata() {
int i,j; /* loop variables */
int isl; /* scratch variable */
for (i=0;i<mnumpoi;i++) apoint.numatri[i]=0;
for (i=0;i<mnumtri;i++) {
        for (triang.centroid.x[i]=triang.centroid.y[i]=0,j=0;j<3;j++) {
                isl=triang.Poi(i,j);
                triang.centroid.x[i] += point.x[isl]/3;
                triang.centroid.y[i] += point.y[isl]/3;
                apoint.Tri(apoint.numatri[isl],isl) = i;
                apoint.Poi1(apoint.numatri[isl],isl) = triang.Poi(i,(j+1)%3);
                apoint.Poi2(apoint.numatri[isl]++,isl) = triang.Poi(i,(j+2)%3);
                }
        }
return;
}
/******************************************************************************/
/*                                                                            */
/*  ROUTINE:   intdvz(a,b,v1,v2,ipoif)                                        */
/*                                                                            */
/*  intdvz performs integration in the v and z directions.  The limits of     */
/*  integration in the v direction are v1 and v2.  v1 is nominally less than  */
/*  v2 however this may be reversed depending on the geometry of the triangle */
/*  The integration in the z direction is from -t/2 to t/2, the observation   */
/*  point lying on the z=t/2 plane.  The u,v coordinates of the observation   */
/*  point are given by the index ipoif to the structure point.  a and b denote*/
/*  the variation of the u' variable with v'.  Specifically, u'=av'+b.        */
/*                                                                            */
```

```
/******************************************************************************/
double intdvz(a,b,v1,v2,ipoif)
double a,b,v1,v2; /* parameters of integration, see above */
int ipoif; /* index to point, denotes observation point */
{
double sum; /* contain sum of integrations in the z direction, corresponding
               to the integral evaluated for v'=v1 and v'=v2  */
sum=intdz(msqrt(1+a*a),(1+a*a)*v2*v2+(-2*v(ipoif)+2*a*(b-u(ipoif)))*
  v2+v(ipoif)*v(ipoif)+(b-u(ipoif))*(b-u(ipoif)),2*(1+a*a)*v2+(-2*v(ipoif)+
  2*a*(b-u(ipoif))));
sum -= intdz(msqrt(1+a*a),(1+a*a)*v1*v1+(-2*v(ipoif)+2*a*(b-u(ipoif)))*
  v1+v(ipoif)*v(ipoif)+(b-u(ipoif))*(b-u(ipoif)),2*(1+a*a)*v1+(-2*v(ipoif)+
  2*a*(b-u(ipoif))));
return(sum);
}
/******************************************************************************/
/*                                                                            */
/*  ROUTINE:   intdz(A,B,C)                                                   */
/*                                                                            */
/*  intdz performs the z integration.  The integration is done analytically,  */
/*  following Gradshteyn and Ryzhik, formulae #2.267-1,#2.261,#2.266.         */
/*  The integral which is evaluated is                                        */
/*  1/A*ln(2A(b+z**2)**.5+C) from -t to 0.                                    */
/*                                                                            */
/******************************************************************************/
double intdz(A,B,C)
double A,B,C; /* parameters of integration */
{
double a,b,z1,z2; /* new parameters of integration */
int ian0;  /* False (=0) if a is effectively zero */
double sum; /* scratch variable, holds sum of contributions */
double troot; /* scratch variable, holds root of R=a+b*x+c*z**2  */
if (fabs(C) < Epsilon * fabs(A)) {
        /* short cut for C = 0 */
        sum = (fabs(B) < Epsilon * fabs(A))? (2*t*mlog(t)-2*t):
              (t*mlog(B+t*t)-2*t+2*msqrt(B)*atan(t/msqrt(B)));
        sum /= 2*A;
        sum += t*mlog(2*A)/A;
        return(sum);
        }
if (fabs(B) < Epsilon * fabs(A)) {
        /* short cut for B = 0 */
        z2 = 2*A*t+C;
        z1 = C;
        sum = z2*mlog(z2)-z2;
        sum -= z1*mlog(z1)-z1;
        sum /= 2*A*A;
        return(sum);
        }
/* no short cuts */
a=C*C-4*B*A*A;
ian0 = fabs(a)>20*Epsilon*(fabs(C*C)+fabs(4*B*A*A));
b= -2*C;
z1=2*A*msqrt(B)+C;
z2=2*A*msqrt(t*t+B)+C;
sum=troot=msqrt(a+b*z2+z2*z2);
if (ian0) sum += (a>0)?
        -msqrt(a)*mlog((2*a+b*z2+2*msqrt(a)*troot)/z2):
        -msqrt(-a)*((fabs(troot)<fabs(a)*Epsilon)?
                (((2*a+b*z2)>0)?Pi/2: -Pi/2):
                atan((2*a+b*z2)/(2*msqrt(-a)*troot)));
sum += b/2*mlog(2*troot+2*z2+b);
sum -= troot=msqrt(a+b*z1+z1*z1);
if (ian0) sum -= (a>0)?
        -msqrt(a)*mlog((2*a+b*z1+2*msqrt(a)*troot)/z1):
        -msqrt(-a)*((fabs(troot)<fabs(a)*Epsilon)?
                (((2*a+b*z1)>0)?Pi/2: -Pi/2):
                atan((2*a+b*z1)/(2*msqrt(-a)*troot)));
sum -= b/2*mlog(2*troot+2*z1+b);
sum /= -2*A*A;
sum += t/A*mlog(2*A*msqrt(B+t*t)+C);
return(sum);
}
/******************************************************************************/
/*                                                                            */
/*  ROUTINE:   solvem(N)                                                      */
/*                                                                            */
/*  solvem is used to solve the matrix problem.  The matrix and forcing vector*/
/*  are created in xsolv or ysolv.  N is the dimension of the vector.  solvem */
/*  solves the matrix problem by calling the two fortran routines dgeco and   */
/*  dgesl which preform a decomposition and back substitution.  solvem is the */
/*  only c routine which calls a fortran routine.                             */
/*                                                                            */
/******************************************************************************/
solvem(N)
int N; /* N is the order of the matrix stored in the array matrix.  N is less
          than or equal to isize */
{
```

```
int isize,job;  /* arguments for fortran routines, isize is the size of the
                array matrix, job (=0) indicates type of matrix problem */
int ipvt[Mnumpoi];  /* an array used by the fortran routines to store the
                        pivoting vector   */
double rcond; /* the condition number of the matrix. */
double z[Mnumpoi*2]; /* scratch vector, lengthened for complex case */
isize=Mnumpoi;
job=0;
if (!taui) dgeco_(matrix,&isize,&N,ipvt,&rcond,z);
else cgeco_(cmatrix,&isize,&N,ipvt,&rcond,z);
printf("condition number is %e\n",1/rcond);
if (!taui) dgesl_(matrix,&isize,&N,ipvt,fvect,&job);
else cgesl_(cmatrix,&isize,&N,ipvt,cfvect,&job);
}
/**************************************************************************/
/*                                                                        */
/*  ROUTINE:   dipole                                                     */
/*                                                                        */
/*  dipole is called by the main program after the potentials over the plate  */
/*  have been computed.  dipole examines each point to determine which points */
/*  are exterior points and then uses these points to define the "edge" of    */
/*  of the plate.  This is needed to perform the surface integral, see Senior */
/*  1976, Low-frequency scattering by a dielectric body.  The surface integral*/
/*  gives the dipole moment.  This is normalized by the volume of the body    */
/*  which is obtained by summing the area of the individual triangles and     */
/*  multiplying by the thickness t.  In the process of isolating the exterior */
/*  points, any point which is incorrectly linked will be identified.         */
/*                                                                        */
/**************************************************************************/
dipole() {
double dip,vol;         /*  dip contains the dipole moment and vol contains the
                            volume of the plate   */
double scrat,sum;       /*  scrat and sum are scratch variables */
double sdip;            /*  sdip is a scratch variable */
double dipi,sdipi;      /*  imaginary part of dip and sdip */
double ycom,xcom,ycomp,xcomp,ycomn,xcomn;  /* more scratch variables, used
                                to determine sign of dipole contributions */
int plist1[Mnumpoi],plist2[Mnumpoi],plist3[Mnumpoi];
                        /*  plist1 and plist2 are used to hold the linked
                            list of exterior points, plist3 hold the list
                            of associated triangles.  The contents of these
                            arrays are pointers to point and, for plist3, to
                            triang.   */
int elist[Mnumpoi];     /*  elist contains the original, unlinked, list of
```

```
                            exterior points.  */
int i,j,k,l,m;          /*  loop variables */
/*
     fill elist
*/
for (i=0,k=0;i<mnumpoi;i++) {
        scrat=sumang(i);
        if (scrat>360.01) {
                printf(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>");
                printf("E R R O R  <<<<<<<<<<<<<<<<<<<<<<<<<<<<<< \n");
                printf(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>");
                printf("point # %d is defined incorrectly:\n",i);
                printf(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>");
                printf("total associated angle is %lf \n",scrat);
                continue;
                }
        if (scrat<359.99) elist[k++] = i;
        }
/*
        if z excitation, skip the following stuff and goto zcase
*/
if (potflag == 3) goto zcase;
/*
    create linked list
*/
for (i=0,m=0;i<k-1;i++) {
        for (j=i+1; j<k; j++) {
                for (l=0;l<apoint.numatri[elist[i]];l++) {
                        if (apoint.Poi1(l,elist[i]) == elist[j]) {
                                plist1[m] = elist[i];
                                plist2[m] = elist[j];
                                plist3[m++] = apoint.Tri(l,elist[i]);
                                }
                        if (apoint.Poi2(l,elist[i]) == elist[j]) {
                                plist1[m] = elist[i];
                                plist2[m] = elist[j];
                                plist3[m++] = apoint.Tri(l,elist[i]);
                                }
                        }
                }
        }
/*
    calculate dipole moment, assuming potflag=1 or 2.
*/
```

```
for (i=0,dip=0,dipi=0; i<m; i++) {
        if (!taui) sdip = ((potflag == 1)?
                fabs(point.y[plist1[i]]-point.y[plist2[i]]):
                fabs(point.x[plist1[i]]-point.x[plist2[i]]))
                        *(point.poten[plist1[i]]+point.poten[plist2[i]])/2;
        else {
                sdip = ((potflag == 1)?
                  fabs(point.y[plist1[i]]-point.y[plist2[i]]):
                  fabs(point.x[plist1[i]]-point.x[plist2[i]]))
                        *(point.cpoten[plist1[i]].real
                          +point.cpoten[plist2[i]].real)/2;
                sdipi = ((potflag == 1)?
                  fabs(point.y[plist1[i]]-point.y[plist2[i]]):
                  fabs(point.x[plist1[i]]-point.x[plist2[i]]))
                        *(point.cpoten[plist1[i]].imag
                          +point.cpoten[plist2[i]].imag)/2;
                }
/*
now determine correct sign
*/
        ycom = point.y[plist1[i]]-point.y[plist2[i]];
        xcom = point.x[plist1[i]]-point.x[plist2[i]];
        ycomp = point.y[plist1[i]]-triang.centroid.y[plist3[i]];
        xcomp = point.x[plist1[i]]-triang.centroid.x[plist3[i]];
        ycomn = -xcom;
        xcomn = ycom;
        scrat = (ycomp*ycomn+xcomp*xcomn)*((potflag == 1)?xcomn:ycomn);
        dip += sdip*((scrat>0)?1:-1);
        if (taui) dipi += sdipi*((scrat>0)?1:-1);
        }

/*
   calculate volume of plate
*/
for (i=0,vol=0;i<mnumtri;i++)
        vol += area(point.x[triang.Poi(i,0)],point.y[triang.Poi(i,0)],
                point.x[triang.Poi(i,1)],point.y[triang.Poi(i,1)],
                point.x[triang.Poi(i,2)],point.y[triang.Poi(i,2)]);
dipmom = (tau-1)*dip/vol-taui*dipi/vol;
dipmomi = (!taui)?0.:(tau-1)*dipi/vol+taui*dip/vol;
return;
/*********************
                        end of x or y dipole computation
                        beginning of z dipole computation
```

```
                                label zcase is only branched to from
                                one spot, immediately following error
                                checking routine.
********************/
zcase:
for (i=0,vol=0,dip=0,dipi=0;i<mnumtri;i++) {
        scrat = area(point.x[triang.Poi(i,0)],point.y[triang.Poi(i,0)],
                point.x[triang.Poi(i,1)],point.y[triang.Poi(i,1)],
                point.x[triang.Poi(i,2)],point.y[triang.Poi(i,2)]);
        vol += scrat;
        if (!taui) {
                sdip = point.poten[triang.Poi(i,0)];
                sdip += point.poten[triang.Poi(i,1)];
                sdip += point.poten[triang.Poi(i,2)];
                sdip *= scrat/3;
                dip += sdip;
                }
        else {
                sdip = point.cpoten[triang.Poi(i,0)].real;
                sdipi = point.cpoten[triang.Poi(i,0)].imag;
                sdip += point.cpoten[triang.Poi(i,1)].real;
                sdipi += point.cpoten[triang.Poi(i,1)].imag;
                sdip += point.cpoten[triang.Poi(i,2)].real;
                sdipi += point.cpoten[triang.Poi(i,2)].imag;
                sdip *= scrat/3;
                sdipi *= scrat/3;
                dip += sdip;
                dipi += sdipi;
                }
        }
/* note that vol actually contains area */
dipmom = -2*(tau-1)*dip/(vol*t)+2*taui*dipi/(vol*t);
dipmomi = (!taui)?0:-2*(tau-1)*dipi/(vol*t)-2*taui*dip/(vol*t);
return;
}
/**************************************************************************/
/*                                                                        */
/*  ROUTINE:   sumang(ipoint)                                             */
/*                                                                        */
/*  sumang is called by dipole to produce the sum of the angles associated  */
/*  with the point ipoint.  If the sum is 360, then the point is an interior */
/*  point, if the sum is less than 360, then the point is an exterior point  */
/*  and defines the edge of the plate                                     */
/*                                                                        */
```

```
/******************************************************************************/
#define Px1 (point.x[apoint.Poi1(i,ipoint)]-point.x[ipoint])  /* shorthand */
#define Px2 (point.x[apoint.Poi2(i,ipoint)]-point.x[ipoint])  /* shorthand */
#define Py1 (point.y[apoint.Poi1(i,ipoint)]-point.y[ipoint])  /* shorthand */
#define Py2 (point.y[apoint.Poi2(i,ipoint)]-point.y[ipoint])  /* shorthand */
double sumang(ipoint)
int ipoint;  /* pointer to the structure point */
{
int i; /* loop variable */
double scale,prod,sum;  /* scale contains the product of the lengths of the
                           two edges, prod contains the dot product of the
                           two edges, and sum contains a running total of the
                           angles.   */
for (i=0,sum=0;i<apoint.numatri[ipoint];i++) {
        scale = (Px1*Px1+Py1*Py1);
        scale *= (Px2*Px2+Py2*Py2);
        scale = sqrt(scale);
        prod = Px1*Px2+Py1*Py2;
        sum += acos(prod/scale);
        }
if (!point.x[ipoint] && flagsym & 1) sum *= 2;
if (!point.y[ipoint] && flagsym & 2) sum *= 2;
sum *= 180/Pi;
return(sum);
}
/******************************************************************************/
/*                                                                            */
/*  ROUTINE:   gethed                                                         */
/*                                                                            */
/*  gethed is called by the main program to get the heading of the data.      */
/*  gethed fills hedstr with the remainder of the line (the entire line       */
/*  except for the letter H which must be in column 1.                        */
/*                                                                            */
/******************************************************************************/
gethed() {
int i;  /* loop variable */
for (i=0; (hedstr[i]=getchar()) != '\n'; i++);
hedstr[i] = '\0';
return;
}
/******************************************************************************/
/*                                                                            */
/*  ROUTINE:   pexc                                                           */
/*                                                                            */
/*  pexc is called by the main program immediately after calling xsolv or    */
/*  ysolv.  pexc calls dipole to calculate the dipole moment, and prints the  */
/*  result, along with information such as the heading, the material          */
/*  parameters, the direction of excitation and the implicit symmetry of the  */
/*  plate.
/*                                                                            */
/******************************************************************************/
pexc() {
printf("%s \n",hedstr);
printf("t = %lf, tau = %lf + i %lf\n",t,tau,taui);
switch (potflag) {
        case 0:
                printf("potential has not been computed\n");
                break;
        case 1:
                printf("potential has been computed assuming x excitation\n");
                break;
        case 2:
                printf("potential has been computed assuming y excitation\n");
                break;
        case 3:
                printf("potential has been computed assuming z excitation\n");
                break;
        }
switch (flagsym) {
        case 0:
                printf("plate is not symmetric\n");
                break;
        case 1:
                printf("plate has mirror symmetry about x=0\n");
                break;
        case 2:
                printf("plate has mirror symmetry about y=0\n");
                break;
        case 3:
                printf("plate has mirror symmetry about x=0 and y=0\n");
                break;
        }
dipole();
printf("The dipole moment is %lf + i %lf\n",dipmom,dipmomi);
}

pmat(){
int i,j;
for(i=0;i<10;printf("\n"),i++)
        for(j=0;j<10;j++) printf("%lf ",Matrix(i,j));
}
pvec(){
int i;
for(i=0;i<10;i++) printf("%lf ",fvect[i]);
printf("\n");
}
double msqrt(x)
double x;
{
if (x> -1e-15) return(sqrt(x));
return(sqrt(x));
}
double mlog(x)
double x;
{
if (x> -1e-15) return(log(x));
return(log(x));
}
/* function definitions */
#include <math.h>
double zppcon(),zptcon(),zplcon(),zint1(),zint2(),p(),l(),alpo();
/* macro definitions */
#define Darray(Ara,I1,I2,L1,L2) Ara[(I1)+(I2)*(L1)]  /* Macro to generate other\
                                                      macros to mimic fortran\
                                                      arrays   */
#define Mnumpoi 100 /* maximum number of points */
#define Mnumtri 100 /* maximum number of triangular patches */
#define Mnumatri 10 /* maximum number of triangles which may share a \
        common point, 6 is average for internal points, 8 accounts for\
        most schemes, thus 10 is a reasonable upper bound */
#define True 1 /* value of logical true */
#define False 0 /* value of logical false */
#define Tri(J1,J2) Darray(tri,J1,J2,Mnumatri,Mnumpoi)  /* set up tri as two\
                           dimensional array with limits (Mnumatri, Mnumpoi) */
#define Poi1(J1,J2) Darray(poi1,J1,J2,Mnumatri,Mnumpoi)  /* set up poi1 as two\
                           dimensional array with limits (Mnumatri, Mnumpoi) */
#define Poi2(J1,J2) Darray(poi2,J1,J2,Mnumatri,Mnumpoi)  /* set up poi2 as two\
                           dimensional array with limits (Mnumatri, Mnumpoi) */
#define Poi(J1,J2) Darray(poi,J1,J2,Mnumtri,3) /* set up poi as two dimensional\
                           array with limits (Mnumtri,3)  */
#define Matrix(J1,J2) Darray(matrix,J1,J2,Mnumpoi,Mnumpoi)  /* set up matrix as\
                           two dimensional array with limits (Mnumpoi,Mnumpoi) */
```

```
#define Cmatrix(J1,J2) Darray(cmatrix,J1,J2,Mnumpoi,Mnumpoi)  /* complex
                                version of Matrix, used with the complex
                                structure */
struct complex {
        double real;
        double imag;
        };
/*                                                                            */
/*                                                                            */
/*  End of Definitions / Beginning of external variable declarations          */
/*                                                                            */
/*                                                                            */
extern
int flagsym; /* flagsym may assume values of 0,1,2, or 3.  Flagsym is used
                by low level routines to incorporate symmetry in a manner
                invisible to the rest of the program.
                0 - indicates no symmetry
                1 - indicates object possesses mirror symmetry about x=0
                2 - indicates object possesses mirror symmetry about y=0
                3 - indicates object possesses mirror symmetry about x=0 and y=0
                Flagsym is set at the beginning of the main program and after
                that is never changed                                         */
extern
int flagsyms; /* flagsyms is used in conjunction with flagsym to generate
                 mirror images of the source patches while calculating
                 contributions to the field point.  The initial object
                 description is assumed to lie in the first quadrant, however
                 if flagsym is 0, this is not necessary.  Flagsyms is always
                 less than or equal to flagsym, and may assume the values of
                 0,1,2, or 3.  These values are given the following meanings:
                 0 - source patch is original patch (presumably first quadrant)
                 1 - source patch is in second quadrant
                 2 - source patch is in fourth quadrant
                 3 - source patch is in third quadrant   */
extern
struct {
        double x[Mnumpoi];
        double y[Mnumpoi];
        double poten[Mnumpoi];
        struct complex cpoten[Mnumpoi]; /* complex version of poten */
        }point ;    /* point is a structure which contains the locations of all
                 of the points used in defining the triangular patches.  A point
                 number 0 is permitted as C defines arrays beginning with 0.
                 X and y are the coordinates of the points, and
                poten is the potential of each point, as determined from the
                matrix problem.    */
extern
struct {
        int numatri[Mnumpoi];
        int Tri(Mnumatri,Mnumpoi-1);
        int Poi1(Mnumatri,Mnumpoi-1);
        int Poi2(Mnumatri,Mnumpoi-1);
        } apoint;  /* apoint is a structure which contains lists of triangles
                associated with each point.  numatri contains the number of
                triangles associated with each point; tri contains the list of
                triangle numbers associated with each point; and poi1 and poi2
                contain the other two vertices used in defining the triangle.
                poi1 and poi2 are indices to point.  tri is an index to
                triang.     */
extern
struct {
        struct {
                double x[Mnumtri];
                double y[Mnumtri];
                } centroid;
        int Poi(Mnumtri,3-1);
        double poten[Mnumtri];
        struct complex cpoten[Mnumtri]; /* complex version of poten */
        } triang;   /*  triang is used to solve the scattering problem with z
                   excitation.  centroid .x and .y contain the coordinates
                   of the centroid of each triangle.  poi contains the list of
                   points (vertices) associated with each triangle, and is an
                   index to point.  poten is the potential of each triangle as
                   determined from solution of the matrix problem.    */
extern
double Matrix(Mnumpoi,Mnumpoi-1);  /* This is "THE"  matrix.  Trivial points
                                      (ie, those which have zero potential from
                                      symmetry considerations) are skipped when
                                      reading or filling the matrix, which is
                                      always done sequentially.    */
extern
struct complex Cmatrix(Mnumpoi,Mnumpoi-1);  /* complex version of Matrix */
extern
double fvect[Mnumpoi];     /* fvect is the forcing vector for the matrix problem
                              and after solution of the matrix problem contains
                              the solution vector.   */
extern
struct complex cfvect[Mnumpoi];  /* complex version of fvect */
```

```
extern
int mnumpoi;  /* This is the total number of points.  Points are assumed to be
                 numbered sequentially from 0.  mnumpoi <= Mnumpoi. */
extern
int mnumtri;  /* This is the total number of triangles.  Triangles are assumed
                 to be numbered sequentially from 0.  mnumtri <= Mnumtri. */
extern
double t; /* Thickness of the plate */
extern
double tau; /* permittivity of the plate */
extern
double taui;  /* imaginary part of the permittivity.  This variable is also
                used as a flag: if taui is zero, computations are performed
                assuming "tau" is purely real; if taui is non-zero, the routines
                appropriate for a complex tau are invoked  */
extern
double dipmom;  /* contains the real part of the computed dipole moment */
extern
double dipmomi;  /* contains the imaginary part of the computed dipole moment */
extern
double Epsilon; /* A very small number, used for approximate equality */
extern
double Pi; /* Pi */
/*
        local global definitions
*/
#define Pointx(I) (point.x[I]*(flagsyms & 1?-1:1)) /* reflect x coordinate */
#define Pointy(I) (point.y[I]*(flagsyms & 2?-1:1)) /* reflect y coordinate */
/*
        variable declarations
*/
double d;       /* set to either t or 0, depending on which side of the
                        surface is being integrated */
double zx0,zy0,alp,calp,salp;
                /* these variables are set by zrotat when the p-l coordinate
                        system is defined.  They are used by p() and l(). */
/******************************************************************************/
/*                                                                            */
/*  ROUTINE:  zsolv()                                                         */
/*  zsolv is called by the main program to calculate the potential on a       */
/*  resulting from z excitation.  zsolv fills Matrix using zppcon (point-     */
/*  point) contributions.  The matrix problem is solved by solvem.            */
/*                                                                            */
/******************************************************************************/
```

```
zsolv() {
int ipoif,ipois; /* pointers to field point and source point, respectively */
for (ipoif=0; ipoif < mnumpoi; ipoif++) {
        for(ipois=0; ipois<mnumpoi; ipois++) {
                if (!taui) Matrix(ipoif,ipois) = (tau-1)/t*zppcon(ipoif,ipois);
                else {
                        Cmatrix(ipoif,ipois).real=(tau-1)/t*zppcon(ipoif,ipois);
                        Cmatrix(ipoif,ipois).imag=taui/t*zppcon(ipoif,ipois);
                        }
                if (ipoif == ipois) {
                        if (!taui) Matrix(ipoif,ipois) += .5;
                        else Cmatrix(ipoif,ipois).real += .5;
                        }
                }
        if (!taui) fvect[ipoif] = -t/2;
        else {
                cfvect[ipoif].real = -t/2;
                cfvect[ipoif].imag = 0;
                }
        }
if (!taui) {
        pmat();
        pvec();
        }
solvem(ipoif);
if (!taui) {
        pmat();
        pvec();
        }
for (ipoif=0; ipoif < mnumpoi; ipoif++)
        if (!taui) point.poten[ipoif] = fvect[ipoif]/2;
        else {
                point.cpoten[ipoif].real = cfvect[ipoif].real/2;
                point.cpoten[ipoif].imag = cfvect[ipoif].imag/2;
                }
}
/*****************************************************************************/
/*                                                                           */
/*  ROUTINE:  zppcon(ipoif,ipois)                                            */
/*  zppcon is called by zsolv to calculate the point to point contribution   */
/*  to the integral.  zppcon evaluates the surface integral on the top and   */
/*  the bottom by setting d=0,t and calling zptcon.                          */
/*                                                                           */
/*****************************************************************************/
```

```
double zppcon(ipoif,ipois)
int ipoif,ipois; /* pointers to field and source points, respectively */
{
int i;  /* loop variable */
double sum;     /* holds sum of contributions from both sides of all
                        relevant triangles */
for (i=0,sum=0; i<apoint.numatri[ipois];i++) {
        d=0;
        sum += zptcon(ipoif,ipois,apoint.Poi1(i,ipois),apoint.Poi2(i,ipois));
        d=t;
        sum -= zptcon(ipoif,ipois,apoint.Poi1(i,ipois),apoint.Poi2(i,ipois));
        }
return(sum/(4*Pi));
}
/******************************************************************************/
/*                                                                            */
/*  ROUTINE:  zptcon(ipoif,ipois,ipoe1,ipoe2)                                 */
/*  zptcon is called by zppcon to calculate the point-triangle contribution   */
/*  to the surface integral.  zptcon accomplishes this by calling zplcon      */
/*  to evaluate the line integral about the triangle (the surface integral    */
/*  is converted to a line integral via Wilton's method).                     */
/*                                                                            */
/******************************************************************************/
double zptcon(ipoif,ipois,ipoe1,ipoe2)
int ipoif,ipois,ipoe1,ipoe2; /* pointers to the field point, source point,
                first point of edge opposing ipois in desired triangle, and
                second point of edge opposing ipois in desired triangle. */
{
double sum;     /* holds sum of integration on three edges of triangle */
for (sum=0,flagsyms=0; flagsyms <= flagsym; flagsyms += (flagsym & 1)?1:2) {
        sum += zplcon(ipoif,ipois,ipoe1,ipoe2,ipois,ipoe1);
        sum += zplcon(ipoif,ipois,ipoe1,ipoe2,ipoe1,ipoe2);
        sum += zplcon(ipoif,ipois,ipoe1,ipoe2,ipoe2,ipois);
        }
return(sum);
}
/******************************************************************************/
/*                                                                            */
/*  ROUTINE:  zplcon(ipoif,ipois,ipoe1,ipoe2,ipoi1,ipoi2)                     */
/*  zplcon is called by zptcon to calculate the contribution of one line of   */
/*  a triangle to a point.  zplcon performs this by evaluating two line       */
/*  integrals via the routines zint1 and zint2.                               */
/*                                                                            */
/******************************************************************************/
```

```
#define Dot(X1,Y1,X2,Y2) (X1*X2+Y1*Y2)  /* define dot product */
#define Vx(I2,I1) (Pointx(I2)-Pointx(I1)) /* x component of vector */
#define Vy(I2,I1) (Pointy(I2)-Pointy(I1)) /* y component of vector */
#define Vfx(I1) (Pointx(I1)-point.x[ipoif])  /* x component relative to ipoif */
#define Vfy(I1) (Pointy(I1)-point.y[ipoif])  /* y component relative to ipoif */
#define Unit(X,Y) temp=sqrt(X*X+Y*Y); \
                  X /= temp;  \
                  Y /= temp; /* normalize the vector pair X,Y */
double zplcon(ipoif,ipois,ipoe1,ipoe2,ipoi1,ipoi2)
int ipoif,ipois,ipoe1,ipoe2,ipoi1,ipoi2;
        /*  ipoif is the pointer to the field point,
            ipois is the pointer to the source point,
            ipoe1 is the pointer to the first point associated with the
                edge which opposes the source point in the current triangle,
            ipoe2 is the pointer to the second point associated with the
                edge which opposes the source point in the current triangle,
            ipoi1 is the pointer to the first point associated with the
                line over which the integration is to be performed in the
                current triangle,
            ipoi2 is the pointer to the second point associated with the
                line over which the integration is to be performed in the
                current triangle,
        */
{
int ipoi3;      /* ipoi3 is the third point in the triangle ipoi1,ipoi2,ipoi3 */
double x0,y0;   /* will contain coordinates of the point on line ipoi1-
                        ipoi2 which is closest to ipoif */
double a,b,c;   /* contains the parameters which describe line ipoi1-ipoi2 */
int iscr;       /* scratch variable */
double sum;     /* holds result of integrations */
double tx,ty;   /* temporary variables, used to hold x and y components of
                        a vector */
double temp;    /* used only in the macro Unit, hold intermediate computation */
double ux,uy;   /* holds x and y components of the unit vector u */
double qx,qy;   /* holds x and y components of the unit vector q */
double scale1,scale2;  /* these are the factors by which the two integrations,
                                zint1 and zint2, are scaled */
double mq;      /* holds the magnitude of the q vector */
double t1;      /* scratch variable */
/*  calculate scaling factors */
tx = Vx(ipoi2,ipoi1);
ty = Vy(ipoi2,ipoi1);
ux = ty;
uy = -tx;
```

```
Unit(ux,uy);
ipoi3 = (!((ipoi1-ipoe1)*(ipoi2-ipoe1))?
        ((ipoi1-ipoe2)*(ipoi2-ipoe2)?ipoe2:ipois):ipoe1);
if (Dot(Vx(ipoi1,ipoi3),Vy(ipoi1,ipoi3),ux,uy)<0) {
        ux = -ux;
        uy = -uy;
        }
/* now find q, first find qhat */
tx = Vx(ipoe2,ipoe1);
ty = Vy(ipoe2,ipoe1);
qx = ty;
qy = -tx;
Unit(qx,qy);
if (Dot(Vx(ipoe1,ipois),Vy(ipoe1,ipois),qx,qy)>0) {
        qx = -qx;
        qy = -qy;
        }
/* next find mag of q, mq */
tl = -Dot(Vx(ipoe1,ipois),Vy(ipoe1,ipois),qx,qy);
mq = 1/tl;
/* now have enough to get u dot q, next get offset */
tl = Dot(Vfx(ipoe1),Vfy(ipoe1),qx,qy)*mq;
scale1 = -tl;
scale2 = mq*Dot(ux,uy,qx,qy);
/*  find x0,y0:  needed to set up l-p coordinate system to facilitate
    integration, cy=ax+b, describes line ipoi1 - ipoi2  */
if (fabs(Pointx(ipoi1)-Pointx(ipoi2)) < Epsilon) {
        c=0;
        a=1;
        b = -Pointx(ipoi1);
        }
else {
        c=1;
        a=(Pointy(ipoi1)-Pointy(ipoi2))/(Pointx(ipoi1)-Pointx(ipoi2));
        b=Pointy(ipoi1)-a*Pointx(ipoi1);
        }
if (c==0) {
        x0=Pointx(ipoi1);
        y0=point.y[ipoif];
        }
else { /* find closest point on line to ipoif */
        x0 = (point.x[ipoif]+a*point.y[ipoif]-a*b)/(1+a*a);
        y0 = a*x0+b;
        }
```

```c
/*  generate (p,l) coordinate system with ipoif at center and x0,y0 on p axis */
zrotat(point.x[ipoif],point.y[ipoif],x0,y0,ipoi1,ipoi2);
if (l(ipoi1) > l(ipoi2)) {
        iscr = ipoi1;
        ipoi1=ipoi2;
        ipoi2=iscr;
        }
if (p(ipoi1)>Epsilon)
        sum = scale1*zint1(ipoif,ipois,ipoe1,ipoe2,ipoi1,ipoi2)
          * ((p(ipoi1)>p(ipoi3))?1:-1);
else    sum = 0;
sum +=  scale2*zint2(ipoif,ipois,ipoe1,ipoe2,ipoi1,ipoi2);
if (point.x[ipoif] == Pointx(ipoi1) && point.y[ipoif] == Pointy(ipoi1))
        sum -= scale1*d*alpo(ipoi1,ipois,ipoe1,ipoe2)/2;
if (point.x[ipoif] == Pointx(ipoi2) && point.y[ipoif] == Pointy(ipoi2))
        sum -= scale1*d*alpo(ipoi2,ipois,ipoe1,ipoe2)/2;
return(sum);
}
/****************************************************************************/
/*                                                                          */
/*  ROUTINE:  zint1(ipoif,ipois,ipoe1,ipoe2,ipoi1,ipoi2)                    */
/*  zint1 is called by zplcon to calculate the line integral resulting from */
/*  the surface integral 1/R.  The line integral is evaluated over the line */
/*  segment ipoi1 - ipoi2.                                                  */
/*                                                                          */
/****************************************************************************/
#define R(I) (sqrt(l(I)*l(I)+p(I)*p(I)+d*d))
double zint1(ipoif,ipois,ipoe1,ipoe2,ipoi1,ipoi2)
int ipoif,ipois,ipoe1,ipoe2,ipoi1,ipoi2;
        /*  ipoif is the pointer to the field point,
            ipois is the pointer to the source point,
            ipoe1 is the pointer to the first point associated with the
                edge which opposes the source point in the current triangle,
            ipoe2 is the pointer to the second point associated with the
                edge which opposes the source point in the current triangle,
            ipoi1 is the pointer to the first point associated with the
                line over which the integration is to be performed in the
                current triangle,
            ipoi2 is the pointer to the second point associated with the
                line over which the integration is to be performed in the
                current triangle,
        */
{
double p0;      /* distance from ipoif to line ipoi1-ipoi2 */
double sum;      /* holds contributions to integral */
sum=0;
p0=p(ipoi1);
if (p0 < Epsilon) return(sum);
sum += log(l(ipoi2)+sqrt(p0*p0+d*d+l(ipoi2)*l(ipoi2)));
sum -= log(l(ipoi1)+sqrt(p0*p0+d*d+l(ipoi1)*l(ipoi1)));
sum *= p0;
if (d > Epsilon) {
        sum += d*atan(d*l(ipoi2)/(p0*R(ipoi2)));
        sum -= d*atan(d*l(ipoi1)/(p0*R(ipoi1)));
        }
return(sum);
}
/***************************************************************************/
/*                                                                         */
/*  ROUTINE:  zint2(ipoif,ipois,ipoe1,ipoe2,ipoi1,ipoi2)                   */
/*  zint2 is called by zplcon to calculate the line integral resulting from */
/*  the surface integral x/R.  The line integral is evaluated over the line */
/*  segment ipoi1 - ipoi2.                                                 */
/*                                                                         */
/***************************************************************************/
double zint2(ipoif,ipois,ipoe1,ipoe2,ipoi1,ipoi2)
int ipoif,ipois,ipoe1,ipoe2,ipoi1,ipoi2;
        /*  ipoif is the pointer to the field point,
            ipois is the pointer to the source point,
            ipoe1 is the pointer to the first point associated with the
                edge which opposes the source point in the current triangle,
            ipoe2 is the pointer to the second point associated with the
                edge which opposes the source point in the current triangle,
            ipoi1 is the pointer to the first point associated with the
                line over which the integration is to be performed in the
                current triangle,
            ipoi2 is the pointer to the second point associated with the
                line over which the integration is to be performed in the
                current triangle,
        */
{
double p0;       /* distance from ipoif to line ipoi1-ipoi2 */
double sum;      /* holds contributions to integral */
double r02;      /* may equal p0**2 or p0**2+t**2, depending on d */
sum=0;
p0=p(ipoi1);
r02=p0*p0+d*d;
if(r02<Epsilon){
        sum += l(ipoi2)*l(ipoi2);
        sum -= l(ipoi1)*l(ipoi1)*(l(ipoi1)*l(ipoi2)>0?1:-1);
        sum = fabs(sum);
        sum /= 2;
        }
else {
        sum += r02*log(R(ipoi2)+l(ipoi2))+l(ipoi2)*R(ipoi2);
        sum -= r02*log(R(ipoi1)+l(ipoi1))+l(ipoi1)*R(ipoi1);
        sum /= 2;
        }
return(sum);
}
/***************************************************************************/
/*                                                                         */
/*  ROUTINE:  zrotat(x1,y1,x2,y2,ipoi1,ipoi2)                              */
/*  zrotat is called by zplcon to initialize the (p,l) coordinate system.  */
/*  zrotat initializes several external variables which are used by p() and */
/*  l().                                                                   */
/*                                                                         */
/***************************************************************************/
zrotat(x1,y1,x2,y2,ipoi1,ipoi2)
double x1,y1,x2,y2; /* x1,y1 are the coordinates of the origin of the (p,l)
                coordinate system.  x2,y2 are the coordinate of a point
                which should lie on the p axis  */
int ipoi1,ipoi2;        /* pointers, used if x1=x2 and y1=y2 */
{
zx0=x1;
zy0=y1;
if (fabs(x2-x1)+fabs(y2-y1)<Epsilon)
        /* line up with perpendicular to ipoi1-ipoi2 */
        alp=atan2(Pointx(ipoi2)-Pointx(ipoi1),-(Pointy(ipoi2)-Pointy(ipoi1)));
else    /* use standard orientation */
        alp=atan2(y2-y1,x2-x1);
calp=cos(alp);
salp=sin(alp);
return;
}
/***************************************************************************/
/*                                                                         */
/*  ROUTINES:  l(i1) and p(i1)                                             */
/*  l(i1) and p(i1) return, respectively, the reflected coordinates of the */
/*  point i1.  Since l() and p() are never used with ipoif, it is safe to  */
/*  reflect all points.                                                    */
/*                                                                         */
```

```
/****************************************************************************/
double l(i1)
int i1;
{
return((Pointx(i1)-zx0)*(-salp)+(Pointy(i1)-zy0)*calp);
}
double p(i1)
int i1;
{
return((Pointx(i1)-zx0)*calp+(Pointy(i1)-zy0)*salp);
}
/****************************************************************************/
/*                                                                          */
/*  ROUTINE:  alpo(i,ipois,ie1,ie2)                                         */
/*  alpo returns the angle associated with point i.  The angle is delimited */
/*  by ipois,ie1,ie2, of which one equals i.                                */
/*                                                                          */
/****************************************************************************/
#define Spx(I) (point.x[I]-point.x[i])  /* shorthand */
#define Spy(I) (point.y[I]-point.y[i])  /* shorthand */
double alpo(i,ipois,ie1,ie2)
int i,ipois,ie1,ie2;    /* see above, ipois, ie1 and ie2 delimit the triangle,
                                of which i is one of the vertices */
{
double scale,prod,ans;  /* scale contains the product of the lengths,
                                prod contains the dot product,
                                and ans contains the angle, ie the answer */
if (ie1 == i) ie1=ipois;
if (ie2 == i) ie2=ipois;
scale = Spx(ie1)*Spx(ie1)+Spy(ie1)*Spy(ie1);
scale *= Spx(ie2)*Spx(ie2)+Spy(ie2)*Spy(ie2);
scale = sqrt(scale);
prod = Spx(ie1)*Spx(ie2)+Spy(ie1)*Spy(ie2);
ans = acos(prod/scale);
return(ans);
}
/* function definitions */
#include <math.h>
double ppcon(), ptcon(),u(),v(),ur(),vr(),intdvz(),intdz(),msqrt(),mlog(),
        area(),sumang(),zppcon();
/* macro definitions */
#define Darray(Ara,I1,I2,L1,L2) Ara[(I1)+(I2)*(L1)]  /* Macro to generate other\
                                                         macros to mimic fortran\
                                                         arrays   */
```

```
#define Mnumpoi 100 /* maximum number of points */
#define Mnumtri 100 /* maximum number of triangular patches */
#define Mnumatri 10 /* maximum number of triangles which may share a \
        common point, 6 is average for internal points, 8 accounts for\
        most schemes, thus 10 is a reasonable upper bound */
#define True 1 /* value of logical true */
#define False 0 /* value of logical false */
#define Tri(J1,J2) Darray(tri,J1,J2,Mnumatri,Mnumpoi)  /* set up tri as two\
                        dimensional array with limits (Mnumatri, Mnumpoi) */
#define Poi1(J1,J2) Darray(poi1,J1,J2,Mnumatri,Mnumpoi)  /* set up poi1 as two\
                        dimensional array with limits (Mnumatri, Mnumpoi) */
#define Poi2(J1,J2) Darray(poi2,J1,J2,Mnumatri,Mnumpoi)  /* set up poi2 as two\
                        dimensional array with limits (Mnumatri, Mnumpoi) */
#define Poi(J1,J2) Darray(poi,J1,J2,Mnumtri,3) /* set up poi as two dimensional\
                        array with limits (Mnumtri,3)  */
#define Matrix(J1,J2) Darray(matrix,J1,J2,Mnumpoi,Mnumpoi)  /* set up matrix as\
                        two dimensional array with limits (Mnumpoi,Mnumpoi) */
#define Cmatrix(J1,J2) Darray(cmatrix,J1,J2,Mnumpoi,Mnumpoi)  /* complex
                                version of Matrix, used with the complex
                                structure */
struct complex {
        double real;
        double imag;
        };
/*                                                                      */
/*                                                                      */
/*  End of Definitions / Beginning of external variable declarations    */
/*                                                                      */
/*                                                                      */
extern
int potflag; /* flag marks whether or not potentials have been computed
                0 - potential has not been computed
                1 - potential has been computed assuming x excitation
                2 - potential has been computed assuming y excitation
                3 - potential has been computed assuming z excitation  */
extern
struct {
        double x[Mnumpoi];
        double y[Mnumpoi];
        double poten[Mnumpoi];
        struct complex cpoten[Mnumpoi]; /* complex version of poten */
        }point ;    /* point is a structure which contains the locations of all
                 of the points used in defining the triangular patches.  A point
                 number 0 is permitted as C defines arrays beginning with 0.
                X and y are the coordinates of the points, and
                poten is the potential of each point, as determined from the
                matrix problem.    */
extern
double t; /* Thickness of the plate */
extern
double tau; /* permittivity of the plate */
extern
double taui;  /* imaginary part of the permittivity.  This variable is also
               used as a flag: if taui is zero, computations are performed
               assuming "tau" is purely real; if taui is non-zero, the routines
               appropriate for a complex tau are invoked  */
extern
double dipmom;  /* contains the real part of the computed dipole moment */
extern
double dipmomi;  /* contains the imaginary part of the computed dipole moment */
extern
double Epsilon;  /* A very small number, used for approximate equality */
extern
double Pi;  /* Pi */
/****************************************************************************/
/*                                                                          */
/*  ROUTINE:    eigen                                                       */
/*                                                                          */
/*  eigen is called by the main program to calculate eigenvalues.  eigen is */
/*  is designed to function on a simplified version of the general plate    */
/*  problem, so as to affect a linear predictor model of the dipole moment. */
/*  The plate must have full x-y symmetry (ie, s3), and should be described */
/*  via a single triangle, with point 0 falling at the origin, point 1 lying */
/*  on the y axis, and point 2 lying on the x axis.  The material parameters */
/*  should be specified before invoking eigen() via the E command.  eigen   */
/*  uses the thickness specified in the m command, but ignores the          */
/*  permittivity.  eigen examines x, y, and z excitation; and will print out */
/*  the eigenvalue for each case along with a dipole moment which would result*/
/*  if tau=2 and the field variation within the plate was unity, or rather, */
/*  the parameter a (see write up) equals 1.                                */
/*                                                                          */
/****************************************************************************/
eigen() {
double integra; /* holds result of integration */
double leng;    /* holds x or y position of test point */
double lambda;  /* holds eigenvalue */
/*  first do x excitation */
potflag=1;
integra = ppcon(2,2)*point.x[2];        /* normalize for unit slope */
leng = point.x[2];
lambda = 1-leng/integra;
point.poten[0]=point.poten[1]=0;
point.poten[2]=point.x[2];
tau=2;
taui=0;
dipole();
printf("thickness = %lf\n",t);
printf(" x excitation, eigenvalue = %lf, I = %lf, punit = %lf\n",lambda,integra,dipmom);
/*  now for y excitation */
potflag=2;
integra = ppcon(1,1)*point.y[1];
leng = point.y[1];
lambda = 1-leng/integra;
point.poten[0]=point.poten[2]=0;
point.poten[1]=point.y[1];
dipole();
printf(" y excitation, eigenvalue = %lf, I = %lf, punit = %lf\n",lambda,integra,dipmom);
/*   and now for z excitation */
potflag=3;
integra = zppcon(0,0)+zppcon(0,1)+zppcon(0,2);
lambda = 1 - t/(2*integra);
point.poten[0]=point.poten[1]=point.poten[2]=t/2;
dipole();
printf(" z excitation, eigenvalue = %lf, I = %lf, punit = %lf\n",lambda,integra,dipmom);
return;
}
/* function definitions */
#include <math.h>
#include <stdio.h>
double area();
/* macro definitions */
#define Darray(Ara,I1,I2,L1,L2) Ara[(I1)+(I2)*(L1)]  /* Macro to generate other\
                                                       macros to mimic fortran\
                                                       arrays   */
#define Mnumpoi 100 /* maximum number of points */
#define Mnumtri 100 /* maximum number of triangular patches */
#define Mnumatri 10 /* maximum number of triangles which may share a \
        common point, 6 is average for internal points, 8 accounts for\
        most schemes, thus 10 is a reasonable upper bound */
#define Mnumplo 51 /* number of lines to plot in graph, must be odd, < 60 */
#define True 1 /* value of logical true */
#define False 0 /* value of logical false */
```

```
#define Para(J1,J2) Darray(para,J1,J2,Mnumplo,Mnumplo)  /* set up para as
                two dimensional array with limits (Mnumplo, Mnumplo) */
#define Tri(J1,J2) Darray(tri,J1,J2,Mnumatri,Mnumpoi)  /* set up tri as two\
                          dimensional array with limits (Mnumatri, Mnumpoi) */
#define Poi(J1,J2) Darray(poi,J1,J2,Mnumtri,3) /* set up poi as two dimensional\
                          array with limits (Mnumtri,3)  */
#define X0 point.x[triang.Poi(i,0)]  /* set up shorthand notations */
#define X1 point.x[triang.Poi(i,1)]
#define X2 point.x[triang.Poi(i,2)]
#define Y0 point.y[triang.Poi(i,0)]
#define Y1 point.y[triang.Poi(i,1)]
#define Y2 point.y[triang.Poi(i,2)]
/*                                                                    */
/*                                                                    */
/*  End of Definitions / Beginning of external variable declarations  */
/*                                                                    */
/*                                                                    */
extern
int flagsym; /* flagsym may assume values of 0,1,2, or 3.  Flagsym is used
                by low level routines to incorporate symmetry in a manner
                invisible to the rest of the program.
                0 - indicates no symmetry
                1 - indicates object possesses mirror symmetry about x=0
                2 - indicates object possesses mirror symmetry about y=0
                3 - indicates object possesses mirror symmetry about x=0 and y=0
                Flagsym is set at the beginning of the main program and after
                that is never changed                                         */
extern
int flagsyms; /* flagsyms is used in conjunction with flagsym to generate
                 mirror images of the source patches while calculating
                 contributions to the field point.  The initial object
                 description is assumed to lie in the first quadrant, however
                 if flagsym is 0, this is not necessary.  Flagsyms is always
                 less than or equal to flagsym, and may assume the values of
                 0,1,2, or 3.  These values are given the following meanings:
                 0 - source patch is original patch (presumably first quadrant)
                 1 - source patch is in second quadrant
                 2 - source patch is in fourth quadrant
                 3 - source patch is in third quadrant   */
extern
int plotsym;  /* plotsym indicates what mirroring is desired for plotting
                 0 - no mirroring
                 1 - mirrored about x=0
                 2 - mirrored about y=0
                 3 - mirrored about x=0 and y=0
                 Any mirroring specified must also have been previously
                 specified in the generation of the object (see flagsym). */
extern
int potflag; /* flag marks whether or not potentials have been computed
                0 - potential has not been computed
                1 - potential has been computed assuming x excitation
                2 - potential has been computed assuming y excitation
                3 - potential has been computed assuming z excitation  */
float Para(Mnumplo,Mnumplo-1);  /* Para contains the potentials which will
                                        be plotted */
extern
struct {
        double x[Mnumpoi];
        double y[Mnumpoi];
        double poten[Mnumpoi];
        }point ;    /* point is a structure which contains the locations of all
                 of the points used in defining the triangular patches.  A point
                 number 0 is permitted as C defines arrays beginning with 0.
                 X and y are the coordinates of the points, and
                 poten is the potential of each point, as determined from the
                 matrix problem.    */
extern
struct {
        struct {
                double x[Mnumtri];
                double y[Mnumtri];
                } centroid;
        int Poi(Mnumtri,3-1);
        double poten[Mnumtri];
        } triang;   /*  triang is used to solve the scattering problem with z
                    excitation.  centroid .x and .y contain the coordinates
                    of the centroid of each triangle.  poi contains the list of
                    points (vertices) associated with each triangle, and is an
                    index to point.  poten is the potential of each triangle as
                    determined from solution of the matrix problem.    */
extern
int mnumpoi;  /* This is the total number of points.  Points are assumed to be
                 numbered sequentially from 0.  mnumpoi <= Mnumpoi. */
extern
int mnumtri;  /* This is the total number of triangles.  Triangles are assumed
                 to be numbered sequentially from 0.  mnumtri <= Mnumtri. */
extern
double Epsilon; /* A very small number, used for approximate equality */
```

```
extern
double Pi; /* Pi */
/******************************************************************************/
/*                                                                            */
/*  ROUTINE:  graph                                                           */
/*                                                                            */
/*  graph is called by the main program to graph the calculated potentials.   */
/*  graph accomplishes this by rerouting the standard output to the input     */
/*  of a call to imprint.  imprint in turn produces a graph and queues it for */
/*  printing.  All variables which must be transferred to graph and its      */
/*  subordinate routines are passed from the main program via external       */
/*  variables.                                                                */
/*                                                                            */
/******************************************************************************/
graph() {
FILE *p_imprint;
FILE *popen();
int ret1, oldstdout;
printf("This shouldn't go to the file\n");
fflush(stdout);
oldstdout=dup(fileno(stdout));
if (oldstdout<0) perror("dup failed");
/* save stdout pointer */
p_imprint=popen("imprint -n -I -Ltektronix");
if (p_imprint == NULL) perror("popen died");
ret1=dup2(fileno(p_imprint),fileno(stdout)); /*make stdout the same as imprint*/
if (ret1<0) perror("dup2 failed");
graphp();
fflush(stdout);
ret1=close(fileno(stdout));
if (ret1<0) perror("close of stdout file");
ret1=dup2(oldstdout,fileno(stdout));
if (ret1<0) perror("restore of stdio  failed");
ret1=pclose(p_imprint);
if (ret1<0) perror("pclose failed");
ret1=close(oldstdout);
if (ret1<0) perror("close failed");
printf("This shouldn't go to the file either.\n");
}
graphy() {
int i,j;
float nara[2500],angle,zmax,zscale;
double Pi=3.14159265;
Pi *= 2./5.;
```

```
zscale = 1.;
for (i=0;i<50;i++) {
        for (j=0;j<50;j++) {
                nara[i+50*j]=zscale*sin(i*Pi/10)*sin(j*Pi/10)+zscale;
                }
        }
angle=30.;
zmax=2.;
dgsurf(nara,50,angle,zmax,1,0);
}
/*****************************************************************************/
/*                                                                           */
/*  ROUTINE: graphp                                                          */
/*                                                                           */
/*  graphp is called by graph.  graphp performs the necessary calculations to */
/*  fill para in a manner acceptable to dgsurf.                              */
/*                                                                           */
/*****************************************************************************/
graphp() {
double m12,m20,m01;  /* these variables contain the slope of the lines
                        connecting points 12, 20, and 01, respectively.  */
double my12,my20,my01;  /*  These variables are used if the line is vertical,
                            implying that m** should be infinity.  Since this
                            is not possible, that formulation of the line is
                            changed to my*y=m*x+b, with my=0.  Except for
                            this case of a vertical line, my=1.   */
double b12,b20,b01;  /* these variables contain the y offset of the lines
                        connecting points 12, 20, and 01, respectively.  */
double mby12,mby20,mby01;  /*  these variables contain the evaluation of the
                                expression  mx+b-y, where x and y are the
                                coordinates of the centroid of the triangle,
                                and m and b are defined above by point pairs
                                using the formulation for the line as y=mx+b.
                                The only importance of the mby** variables is
                                the sign of the number they contain, since if
                                an arbitrary point (x*,y*) substituted into
                                the expression mx+b-y produces a similar sign,
                                then both the centroid and the point (x*,y*)
                                are on the same side of the line.  By making
                                this comparison for all three lines, it is
                                determined whether or not an arbitrary point
                                lies within a given triangle. */
float zofset;           /*  zofset contains the offset which is added to all
                            potentials as they are stored in para.  This is
```

```
                                necessary because dgsurf requires all points to
                                be positive.  zofset is defined as float so that
                                it may be compared with elements of para to
                                determine if they have been filled.  An exact
                                match indicates that they have not been filled */
double zmax,zmin,zscrat;
                             /* zmax and zmin are the maximum and
                                minimum potentials and are, of course, needed to
                                determine what is the proper offset to be added
                                to the potentials.  zpmax is the maximum value
                                to be plotted (usually zmax+zofset).  zscrat is
                                a scratch variable used in performing the above
                                described calculations.  */
float zpmax,angle;           /* parameters which are passed to dgsurf.  zpmax
                                is the maximum value to be graphed.  angle is
                                the angle at which the perspective plot is to
                                be made.    */
double xmin,xmax,ymin,ymax;     /*  These variables delimit the region
                                     occupied by the plate */
int mirx,miry;               /* mirx and miry are flags used to note whether a
                                mirror image of calculated potentials is to be
                                plotted.  This may be used in conjunctions with
                                the symmetry specification (see main program).
                                Thus, using symmetry, only, say, 1/4 of the
                                potentials need be computed, and using mirror
                                plotting, the additional points may be generated.
                                mirx and miry may assumed values of 1,0, or -1.
                                    0  indicates no mirroring is desired.
                                    1  indicates exact mirroring
                                    -1 indicates that the mirrored image should
                                            be inverted
                                The mirroring may be specified independently
                                for the x and y directions.  The type of mirroring
                                is determined by the program by considering
                                direction of mirroring and direction of excitation.
                                                                                */
int ixmin,iymin,ixmax,iymax;/* These variables are used to delimit the region of
                                para which is currently being filled with values
                                from a triangular patch.  */
double txmin,txmax,tymin,tymax; /*  These variables are used to delimit the
                                     triangular patch which is currently being
                                     mapped to para */
double xi,yi;                /* These variables specify coordinates within the
                                triangular patch which are being mapped to para */
```

```c
double l1,l2,l0,delta;  /*  These are the area coordinates: L1,L2, and L3,
                            see, for example Zienkiewicz, The Finite Element
                            Method for an explanation.  delta is the area of
                            the triangle and l1, l2, and l0 are the area
                            coordinates associated with points 1, 2, and 0,
                            respectively.   */
int i,j,ix,iy;   /*  loop variables.  ix and iy are used specifically to index
                          para  */
if (potflag == 3) for (i=0;i<mnumpoi;i++) point.poten[i] = -point.poten[i];
mirx = (!(plotsym & 1))?0:((potflag == 1)? -1:1);
miry = (!(plotsym & 2))?0:((potflag == 2)? -1:1);
zmax = -1e20;
zmin = 1e20;
xmax = -1e20;
xmin = 1e20;
ymax = -1e20;
ymin = 1e20;
for (i=0; i < mnumpoi; i++) {
        if (zmax < point.poten[i]) zmax = point.poten[i];
        if (zmin > point.poten[i]) zmin = point.poten[i];
        if (xmax < point.x[i]) xmax = point.x[i];
        if (xmin > point.x[i]) xmin = point.x[i];
        if (ymax < point.y[i]) ymax = point.y[i];
        if (ymin > point.y[i]) ymin = point.y[i];
        }
zscrat=(fabs(zmax) > fabs(zmin))?fabs(zmax):fabs(zmin);
if (mirx == -1 || miry == -1) {
        zofset = zscrat;
        zpmax = 2*zofset;
        }
else {
        zofset = (zmin>0)?0:-zmin;
        zpmax = zofset+zmax;
        }
for (i=0; i < Mnumplo; i++) {
        for (j=0; j < Mnumplo; j++) {
                Para(i,j) = zofset;
                }
        }
for (i=0; i<mnumtri; i++) {
        for (txmin=tymin=1e20,txmax=tymax= -1e20,j=0; j<3; j++) {
                if (txmin > point.x[triang.Poi(i,j)])
                        txmin = point.x[triang.Poi(i,j)];
                if (tymin > point.y[triang.Poi(i,j)])
                tymin = point.y[triang.Poi(i,j)];
        if (txmax < point.x[triang.Poi(i,j)])
                txmax = point.x[triang.Poi(i,j)];
        if (tymax < point.y[triang.Poi(i,j)])
                tymax = point.y[triang.Poi(i,j)];
        }
zscrat= X1-X0;
if (fabs(zscrat) > Epsilon) {
        m01 = (Y1-Y0)/zscrat;
        my01 = 1;
        }
else {
        m01 = 1;
        my01 = 0;
        }
zscrat = X2-X1;
if (fabs(zscrat) > Epsilon) {
        m12 = (Y2-Y1)/zscrat;
        my12 = 1;
        }
else {
        m12 = 1;
        my12 = 0;
        }
zscrat = X0-X2;
if (fabs(zscrat) > Epsilon) {
        m20 = (Y0-Y2)/zscrat;
        my20 = 1;
        }
else {
        m20 = 1;
        my20 = 0;
        }
b01 = my01*Y0-m01*X0;
b12 = my12*Y1-m12*X1;
b20 = my20*Y2-m20*X2;
mby01 = m01*triang.centroid.x[i]+b01-my01*triang.centroid.y[i];
mby12 = m12*triang.centroid.x[i]+b12-my12*triang.centroid.y[i];
mby20 = m20*triang.centroid.x[i]+b20-my20*triang.centroid.y[i];
delta = area(X0,Y0,X1,Y1,X2,Y2);
txmin *= (txmin < 0)? (1+Epsilon): (1-Epsilon);
tymin *= (tymin < 0)? (1+Epsilon): (1-Epsilon);
txmax *= (txmax > 0)? (1+Epsilon): (1-Epsilon);
tymax *= (tymax > 0)? (1+Epsilon): (1-Epsilon);
```

```
        ixmin = (mirx)? ceil((txmin-xmin)*(Mnumplo-1)/2/(xmax-xmin)):
                ceil((txmin-xmin)*(Mnumplo-1)/(xmax-xmin));
        ixmax = (mirx)? floor((txmax-xmin)*(Mnumplo-1)/2/(xmax-xmin)):
                floor((txmax-xmin)*(Mnumplo-1)/(xmax-xmin));
        iymin = (miry)? ceil((tymin-ymin)*(Mnumplo-1)/2/(ymax-ymin)):
                ceil((tymin-ymin)*(Mnumplo-1)/(ymax-ymin));
        iymax = (miry)? floor((tymax-ymin)*(Mnumplo-1)/2/(ymax-ymin)):
                floor((tymax-ymin)*(Mnumplo-1)/(ymax-ymin));
        for (ix=ixmin; ix <= ixmax; ix++) {
                xi = (mirx)? (ix*(xmax-xmin)*2/(Mnumplo-1)+xmin):
                        (ix*(xmax-xmin)/(Mnumplo-1)+xmin);
                for (iy=iymin; iy <= iymax; iy++) {
                        yi = (miry)? (iy*(ymax-ymin)*2/(Mnumplo-1)+ymin):
                                (iy*(ymax-ymin)/(Mnumplo-1)+ymin);
                        if (Para(ix,iy) != zofset ||
                                mby01*(m01*xi+b01-my01*yi) < -Epsilon ||
                                mby12*(m12*xi+b12-my12*yi) < -Epsilon ||
                                mby20*(m20*xi+b20-my20*yi)< -Epsilon ) continue;
                        l0 = area(xi,yi,X1,Y1,X2,Y2)/delta;
                        l1 = area(X0,Y0,xi,yi,X2,Y2)/delta;
                        l2 = area(X0,Y0,X1,Y1,xi,yi)/delta;
                        Para(ix,iy) = l0*point.poten[triang.Poi(i,0)]
                                + l1*point.poten[triang.Poi(i,1)]
                                + l2*point.poten[triang.Poi(i,2)]
                                + zofset;
                        }
                }
        }
if (mirx) {
        for (ix=(Mnumplo-1)/2; ix >= 0; ix--)
                for (iy=0;iy<Mnumplo;iy++)
                        Para(ix+(Mnumplo-1)/2,iy)=Para(ix,iy);
        for (ix=0;ix<(Mnumplo-1)/2;ix++)
                for (iy=0;iy<Mnumplo;iy++)
                        Para(ix,iy) = (mirx == 1)? Para(Mnumplo-1-ix,iy):
                                -Para(Mnumplo-1-ix,iy)+2*zofset;
        }
if (miry) {
        for (iy=(Mnumplo-1)/2; iy >= 0; iy--)
                for (ix=0;ix<Mnumplo;ix++)
                        Para(ix,iy+(Mnumplo-1)/2)=Para(ix,iy);
        for (iy=0;iy<(Mnumplo-1)/2;iy++)
                for (ix=0;ix<Mnumplo;ix++)
                        Para(ix,iy) = (miry == 1)? Para(ix,Mnumplo-1-iy):
```

```
                                  -Para(ix,Mnumplo-1-iy)+2*zofset;
        }
angle=(potflag == 3)?30:75;
dgsurf(para,Mnumplo,angle,zpmax,1,0);
}
/****************************************************************************/
/*                                                                          */
/*  ROUTINE: area                                                           */
/*                                                                          */
/*  area is called by graphp to calculate the area of a triangle.  This is  */
/*  needed in the computation of the area coordinates.                      */
/*                                                                          */
/****************************************************************************/
double area(x1,y1,x2,y2,x3,y3)
double x1,y1,x2,y2,x3,y3;       /*  These are the coordinates of the three
                                    points which delimit the triangle  */
{
return(fabs(x2*y3-y2*x3-x1*y3+y1*x3+x1*y2-y1*x2)/2);
}
/****************************************************************************/
/*                                                                          */
/*  ROUTINES: DSGSURF and DGZPLT                                            */
/*                                                                          */
/*  These routines are proprietary and source listings are not available.   */
/*                                                                          */
/****************************************************************************/
C NAASA  2.1.028 DGECO    FTN-A 05-02-78     THE UNIV OF MICH COMP CTR
      SUBROUTINE DGECO(A,LDA,N,IPVT,RCOND,Z)
      INTEGER LDA,N,IPVT(1)
      DOUBLE PRECISION A(LDA,1),Z(1)
      DOUBLE PRECISION RCOND
C
C     DGECO FACTORS A DOUBLE PRECISION MATRIX BY GAUSSIAN ELIMINATION
C     AND ESTIMATES THE CONDITION OF THE MATRIX.
C
C     IF  RCOND  IS NOT NEEDED, DGEFA IS SLIGHTLY FASTER.
C     TO SOLVE  A*X = B , FOLLOW DGECO BY DGESL.
C     TO COMPUTE  INVERSE(A)*C , FOLLOW DGECO BY DGESL.
C     TO COMPUTE  DETERMINANT(A) , FOLLOW DGECO BY DGEDI.
C     TO COMPUTE  INVERSE(A) , FOLLOW DGECO BY DGEDI.
C
C     ON ENTRY
C
C        A       DOUBLE PRECISION(LDA, N)
```

```
C                THE MATRIX TO BE FACTORED.
C
C        LDA     INTEGER
C                THE LEADING DIMENSION OF THE ARRAY  A .
C
C        N       INTEGER
C                THE ORDER OF THE MATRIX  A .
C
C     ON RETURN
C
C        A       AN UPPER TRIANGULAR MATRIX AND THE MULTIPLIERS
C                WHICH WERE USED TO OBTAIN IT.
C                THE FACTORIZATION CAN BE WRITTEN  A = L*U  WHERE
C                L  IS A PRODUCT OF PERMUTATION AND UNIT LOWER
C                TRIANGULAR MATRICES AND  U  IS UPPER TRIANGULAR.
C
C        IPVT    INTEGER(N)
C                AN INTEGER VECTOR OF PIVOT INDICES.
C
C        RCOND   DOUBLE PRECISION
C                AN ESTIMATE OF THE RECIPROCAL CONDITION OF  A .
C                FOR THE SYSTEM  A*X = B , RELATIVE PERTURBATIONS
C                IN  A  AND  B  OF SIZE  EPSILON  MAY CAUSE
C                RELATIVE PERTURBATIONS IN  X  OF SIZE  EPSILON/RCOND .
C                IF  RCOND  IS SO SMALL THAT THE LOGICAL EXPRESSION
C                           1.0 + RCOND .EQ. 1.0
C                IS TRUE, THEN  A  MAY BE SINGULAR TO WORKING
C                PRECISION.  IN PARTICULAR,  RCOND  IS ZERO  IF
C                EXACT SINGULARITY IS DETECTED OR THE ESTIMATE
C                UNDERFLOWS.
C
C        Z       DOUBLE PRECISION(N)
C                A WORK VECTOR WHOSE CONTENTS ARE USUALLY UNIMPORTANT.
C                IF  A  IS CLOSE TO A SINGULAR MATRIX, THEN  Z  IS
C                AN APPROXIMATE NULL VECTOR IN THE SENSE THAT
C                NORM(A*Z) = RCOND*NORM(A)*NORM(Z) .
C
C     LINPACK. THIS VERSION DATED 07/14/77 .
C     CLEVE MOLER, UNIVERSITY OF NEW MEXICO, ARGONNE NATIONAL LABS.
C
C     SUBROUTINES AND FUNCTIONS
C
C     LINPACK DGEFA
C     BLAS DAXPY,DDOT,DSCAL,DASUM
```

```
C     FORTRAN DABS,DMAX1,DSIGN
C
C     INTERNAL VARIABLES
C
      DOUBLE PRECISION DDOT,EK,T,WK,WKM
      DOUBLE PRECISION ANORM,S,DASUM,SM,YNORM
      INTEGER INFO,J,K,KB,KP1,L
C
      DOUBLE PRECISION DSIGN
C
C     COMPUTE 1-NORM OF A
C
      ANORM = 0.0D0
      DO 10 J = 1, N
         ANORM = DMAX1(ANORM,DASUM(N,A(1,J),1))
   10 CONTINUE
C
C     FACTOR
C
      CALL DGEFA(A,LDA,N,IPVT,INFO)
C
C     RCOND = 1/(NORM(A)*(ESTIMATE OF NORM(INVERSE(A)))) .
C     ESTIMATE = NORM(Z)/NORM(Y) WHERE  A*Z = Y  AND  TRANS(A)*Y = E .
C     TRANS(A)  IS THE TRANSPOSE OF A .  THE COMPONENTS OF  E  ARE
C     CHOSEN TO CAUSE MAXIMUM LOCAL GROWTH IN THE ELEMENTS OF W  WHERE
C     TRANS(U)*W = E .  THE VECTORS ARE FREQUENTLY RESCALED TO AVOID
C     OVERFLOW.
C
C     SOLVE TRANS(U)*W = E
C
      EK = 1.0D0
      DO 20 J = 1, N
         Z(J) = 0.0D0
   20 CONTINUE
      DO 100 K = 1, N
         IF (Z(K) .NE. 0.0D0) EK = DSIGN(EK,-Z(K))
         IF (DABS(EK-Z(K)) .LE. DABS(A(K,K))) GO TO 30
            S = DABS(A(K,K))/DABS(EK-Z(K))
            CALL DSCAL(N,S,Z,1)
            EK = S*EK
   30    CONTINUE
         WK = EK - Z(K)
         WKM = -EK - Z(K)
         S = DABS(WK)
```

```
          SM = DABS(WKM)
          IF (A(K,K) .EQ. 0.0D0) GO TO 40
             WK = WK/A(K,K)
             WKM = WKM/A(K,K)
          GO TO 50
   40     CONTINUE
             WK = 1.0D0
             WKM = 1.0D0
   50     CONTINUE
          KP1 = K + 1
          IF (KP1 .GT. N) GO TO 90
             DO 60 J = KP1, N
                SM = SM + DABS(Z(J)+WKM*A(K,J))
                Z(J) = Z(J) + WK*A(K,J)
                S = S + DABS(Z(J))
   60        CONTINUE
             IF (S .GE. SM) GO TO 80
                T = WKM - WK
                WK = WKM
                DO 70 J = KP1, N
                   Z(J) = Z(J) + T*A(K,J)
   70           CONTINUE
   80        CONTINUE
   90     CONTINUE
          Z(K) = WK
  100 CONTINUE
      S = 1.0D0/DASUM(N,Z,1)
      CALL DSCAL(N,S,Z,1)
C
C     SOLVE TRANS(L)*Y = V
C
      DO 120 KB = 1, N
         K = N + 1 - KB
         IF (K .LT. N) Z(K) = Z(K) + DDOT(N-K,A(K+1,K),1,Z(K+1),1)
         IF (DABS(Z(K)) .LE. 1.0D0) GO TO 110
            S = 1.0D0/DABS(Z(K))
            CALL DSCAL(N,S,Z,1)
  110    CONTINUE
         L = IPVT(K)
         T = Z(L)
         Z(L) = Z(K)
         Z(K) = T
  120 CONTINUE
      S = 1.0D0/DASUM(N,Z,1)
      CALL DSCAL(N,S,Z,1)
C
      YNORM = 1.0D0
C
C     SOLVE L*V = Y
C
      DO 140 K = 1, N
         L = IPVT(K)
         T = Z(L)
         Z(L) = Z(K)
         Z(K) = T
         IF (K .LT. N) CALL DAXPY(N-K,T,A(K+1,K),1,Z(K+1),1)
         IF (DABS(Z(K)) .LE. 1.0D0) GO TO 130
            S = 1.0D0/DABS(Z(K))
            CALL DSCAL(N,S,Z,1)
            YNORM = S*YNORM
  130    CONTINUE
  140 CONTINUE
      S = 1.0D0/DASUM(N,Z,1)
      CALL DSCAL(N,S,Z,1)
      YNORM = S*YNORM
C
C     SOLVE  U*Z = V
C
      DO 160 KB = 1, N
         K = N + 1 - KB
         IF (DABS(Z(K)) .LE. DABS(A(K,K))) GO TO 150
            S = DABS(A(K,K))/DABS(Z(K))
            CALL DSCAL(N,S,Z,1)
            YNORM = S*YNORM
  150    CONTINUE
         IF (A(K,K) .NE. 0.0D0) Z(K) = Z(K)/A(K,K)
         IF (A(K,K) .EQ. 0.0D0) Z(K) = 1.0D0
         T = -Z(K)
         CALL DAXPY(K-1,T,A(1,K),1,Z(1),1)
  160 CONTINUE
C     MAKE ZNORM = 1.0
      S = 1.0D0/DASUM(N,Z,1)
      CALL DSCAL(N,S,Z,1)
      YNORM = S*YNORM
C
      IF (ANORM .NE. 0.0D0) RCOND = YNORM/ANORM
      IF (ANORM .EQ. 0.0D0) RCOND = 0.0D0
      RETURN
      END
C NAASA  2.1.029 DGEFA    FTN-A 05-02-78     THE UNIV OF MICH COMP CTR
      SUBROUTINE DGEFA(A,LDA,N,IPVT,INFO)
      INTEGER LDA,N,IPVT(1),INFO
      DOUBLE PRECISION A(LDA,1)
C
C     DGEFA FACTORS A DOUBLE PRECISION MATRIX BY GAUSSIAN ELIMINATION.
C
C     DGEFA IS USUALLY CALLED BY DGECO, BUT IT CAN BE CALLED
C     DIRECTLY WITH A SAVING IN TIME IF  RCOND  IS NOT NEEDED.
C     (TIME FOR DGECO) = (1 + 9/N)*(TIME FOR DGEFA) .
C
C     ON ENTRY
C
C        A       DOUBLE PRECISION(LDA, N)
C                THE MATRIX TO BE FACTORED.
C
C        LDA     INTEGER
C                THE LEADING DIMENSION OF THE ARRAY  A .
C
C        N       INTEGER
C                THE ORDER OF THE MATRIX  A .
C
C     ON RETURN
C
C        A       AN UPPER TRIANGULAR MATRIX AND THE MULTIPLIERS
C                WHICH WERE USED TO OBTAIN IT.
C                THE FACTORIZATION CAN BE WRITTEN  A = L*U  WHERE
C                L  IS A PRODUCT OF PERMUTATION AND UNIT LOWER
C                TRIANGULAR MATRICES AND  U  IS UPPER TRIANGULAR.
C
C        IPVT    INTEGER(N)
C                AN INTEGER VECTOR OF PIVOT INDICES.
C
C        INFO    INTEGER
C                = 0  NORMAL VALUE.
C                = K  IF  U(K,K) .EQ. 0.0 .  THIS IS NOT AN ERROR
C                     CONDITION FOR THIS SUBROUTINE, BUT IT DOES
C                     INDICATE THAT DGESL OR DGEDI WILL DIVIDE BY ZERO
C                     IF CALLED.  USE  RCOND  IN DGECO FOR A RELIABLE
C                     INDICATION OF SINGULARITY.
C
C     LINPACK. THIS VERSION DATED 07/14/77 .
C     CLEVE MOLER, UNIVERSITY OF NEW MEXICO, ARGONNE NATIONAL LABS.
```

```
C
C     SUBROUTINES AND FUNCTIONS
C
C     BLAS DAXPY,DSCAL,IDAMAX
C
C     INTERNAL VARIABLES
C
      DOUBLE PRECISION T
      INTEGER IDAMAX,J,K,KP1,L,NM1
C
C
C     GAUSSIAN ELIMINATION WITH PARTIAL PIVOTING
C
      INFO = 0
      NM1 = N - 1
      IF (NM1 .LT. 1) GO TO 70
      DO 60 K = 1, NM1
         KP1 = K + 1
C
C        FIND L = PIVOT INDEX
C
         L = IDAMAX(N-K+1,A(K,K),1) + K - 1
         IPVT(K) = L
C
C        ZERO PIVOT IMPLIES THIS COLUMN ALREADY TRIANGULARIZED
C
         IF (A(L,K) .EQ. 0.0D0) GO TO 40
C
C           INTERCHANGE IF NECESSARY
C
            IF (L .EQ. K) GO TO 10
               T = A(L,K)
               A(L,K) = A(K,K)
               A(K,K) = T
   10       CONTINUE
C
C           COMPUTE MULTIPLIERS
C
            T = -1.0D0/A(K,K)
            CALL DSCAL(N-K,T,A(K+1,K),1)
C
C           ROW ELIMINATION WITH COLUMN INDEXING
C
            DO 30 J = KP1, N
```

```
               T = A(L,J)
               IF (L .EQ. K) GO TO 20
                  A(L,J) = A(K,J)
                  A(K,J) = T
   20          CONTINUE
               CALL DAXPY(N-K,T,A(K+1,K),1,A(K+1,J),1)
   30       CONTINUE
         GO TO 50
   40    CONTINUE
            INFO = K
   50    CONTINUE
   60 CONTINUE
   70 CONTINUE
      IPVT(N) = N
      IF (A(N,N) .EQ. 0.0D0) INFO = N
      RETURN
      END
C NAASA  2.1.030 DGESL    FTN-A 05-02-78      THE UNIV OF MICH COMP CTR
      SUBROUTINE DGESL(A,LDA,N,IPVT,B,JOB)
      INTEGER LDA,N,IPVT(1),JOB
      DOUBLE PRECISION A(LDA,1),B(1)
C
C     DGESL SOLVES THE DOUBLE PRECISION SYSTEM
C     A * X = B  OR  TRANS(A) * X = B
C     USING THE FACTORS COMPUTED BY DGECO OR DGEFA.
C
C     ON ENTRY
C
C        A       DOUBLE PRECISION(LDA, N)
C                THE OUTPUT FROM DGECO OR DGEFA.
C
C        LDA     INTEGER
C                THE LEADING DIMENSION OF THE ARRAY  A .
C
C        N       INTEGER
C                THE ORDER OF THE MATRIX  A .
C
C        IPVT    INTEGER(N)
C                THE PIVOT VECTOR FROM DGECO OR DGEFA.
C
C        B       DOUBLE PRECISION(N)
C                THE RIGHT HAND SIDE VECTOR.
C
C        JOB     INTEGER
```

```
C                 = 0          TO SOLVE  A*X = B ,
C                 = NONZERO    TO SOLVE  TRANS(A)*X = B  WHERE
C                              TRANS(A)  IS THE TRANSPOSE.
C
C     ON RETURN
C
C        B       THE SOLUTION VECTOR  X .
C
C     ERROR CONDITION
C
C        A DIVISION BY ZERO WILL OCCUR IF THE INPUT FACTOR CONTAINS A
C        ZERO ON THE DIAGONAL.  TECHNICALLY THIS INDICATES SINGULARITY
C        BUT IT IS OFTEN CAUSED BY IMPROPER ARGUMENTS OR IMPROPER
C        SETTING OF LDA .  IT WILL NOT OCCUR IF THE SUBROUTINES ARE
C        CALLED CORRECTLY AND IF DGECO HAS SET RCOND .GT. 0.0
C        OR DGEFA HAS SET INFO .EQ. 0 .
C
C     TO COMPUTE  INVERSE(A) * C  WHERE  C  IS A MATRIX
C     WITH  P  COLUMNS
C           CALL DGECO(A,LDA,N,IPVT,RCOND,Z)
C           IF (RCOND IS TOO SMALL) GO TO ...
C           DO 10 J = 1, P
C              CALL DGESL(A,LDA,N,IPVT,C(1,J),0)
C        10 CONTINUE
C
C     LINPACK. THIS VERSION DATED 07/14/77 .
C     CLEVE MOLER, UNIVERSITY OF NEW MEXICO, ARGONNE NATIONAL LABS.
C
C     SUBROUTINES AND FUNCTIONS
C
C     BLAS DAXPY,DDOT
C
C     INTERNAL VARIABLES
C
      DOUBLE PRECISION DDOT,T
      INTEGER K,KB,L,NM1
C
      NM1 = N - 1
      IF (JOB .NE. 0) GO TO 50
C
C        JOB = 0 , SOLVE  A * X = B
C        FIRST SOLVE  L*Y = B
C
         IF (NM1 .LT. 1) GO TO 30
      DO 20 K = 1, NM1
         L = IPVT(K)
         T = B(L)
         IF (L .EQ. K) GO TO 10
            B(L) = B(K)
            B(K) = T
   10    CONTINUE
         CALL DAXPY(N-K,T,A(K+1,K),1,B(K+1),1)
   20 CONTINUE
   30 CONTINUE
C
C     NOW SOLVE  U*X = Y
C
      DO 40 KB = 1, N
         K = N + 1 - KB
         B(K) = B(K)/A(K,K)
         T = -B(K)
         CALL DAXPY(K-1,T,A(1,K),1,B(1),1)
   40 CONTINUE
      GO TO 100
   50 CONTINUE
C
C     JOB = NONZERO, SOLVE  TRANS(A) * X = B
C     FIRST SOLVE  TRANS(U)*Y = B
C
      DO 60 K = 1, N
         T = DDOT(K-1,A(1,K),1,B(1),1)
         B(K) = (B(K) - T)/A(K,K)
   60 CONTINUE
C
C     NOW SOLVE TRANS(L)*X = Y
C
      IF (NM1 .LT. 1) GO TO 90
      DO 80 KB = 1, NM1
         K = N - KB
         B(K) = B(K) + DDOT(N-K,A(K+1,K),1,B(K+1),1)
         L = IPVT(K)
         IF (L .EQ. K) GO TO 70
            T = B(L)
            B(L) = B(K)
            B(K) = T
   70    CONTINUE
   80 CONTINUE
   90 CONTINUE
```

```
  100 CONTINUE
      RETURN
      END
C NAASA  1.1.001 DASUM     FTN-A 05-02-78      THE UNIV OF MICH COMP CTR
      DOUBLE PRECISION FUNCTION DASUM(N,DX,INCX)
C
C     TAKES THE SUM OF THE ABSOLUTE VALUES.
C     JACK DONGARRA, LINPACK, 6/17/77.
C
      DOUBLE PRECISION DX(1),DTEMP
      INTEGER I,INCX,M,MP1,N,NINCX
C
      DASUM = 0.0D0
      DTEMP = 0.0D0
      IF(N.LE.0)RETURN
      IF(INCX.EQ.1)GOTO 20
C
C        CODE FOR INCREMENT NOT EQUAL TO 1
C
      NINCX = N*INCX
      DO 10 I = 1,NINCX,INCX
        DTEMP = DTEMP + DABS(DX(I))
   10 CONTINUE
      DASUM = DTEMP
      RETURN
C
C        CODE FOR INCREMENT EQUAL TO 1
C
C
C        CLEAN-UP LOOP
C
   20 M = MOD(N,6)
      IF( M .EQ. 0 ) GO TO 40
      DO 30 I = 1,M
        DTEMP = DTEMP + DABS(DX(I))
   30 CONTINUE
      IF( N .LT. 6 ) GO TO 60
   40 MP1 = M + 1
      DO 50 I = MP1,N,6
        DTEMP = DTEMP + DABS(DX(I)) + DABS(DX(I + 1)) + DABS(DX(I + 2))
     *  + DABS(DX(I + 3)) + DABS(DX(I + 4)) + DABS(DX(I + 5))
   50 CONTINUE
   60 DASUM = DTEMP
      RETURN
      END
C NAASA  1.1.004 DAXPY    FTN-A 05-02-78     THE UNIV OF MICH COMP CTR
      SUBROUTINE DAXPY(N,DA,DX,INCX,DY,INCY)
C
C     CONSTANT TIMES A VECTOR PLUS A VECTOR.
C     USES UNROLLED LOOPS FOR INCREMENTS EQUAL TO ONE.
C     JACK DONGARRA, LINPACK, 6/17/77.
C
      DOUBLE PRECISION DX(1),DY(1),DA
      INTEGER I,INCX,INCY,IXIY,M,MP1,N
C
      IF(N.LE.0)RETURN
      IF (DA .EQ. 0.0D0) RETURN
      IF(INCX.EQ.1.AND.INCY.EQ.1)GOTO 20
C
C        CODE FOR UNEQUAL INCREMENTS OR EQUAL INCREMENTS
C          NOT EQUAL TO 1
C
      IX = 1
      IY = 1
      IF(INCX.LT.0)IX = (-N+1)*INCX + 1
      IF(INCY.LT.0)IY = (-N+1)*INCY + 1
      DO 10 I = 1,N
        DY(IY) = DY(IY) + DA*DX(IX)
        IX = IX + INCX
        IY = IY + INCY
   10 CONTINUE
      RETURN
C
C        CODE FOR BOTH INCREMENTS EQUAL TO 1
C
C
C        CLEAN-UP LOOP
C
   20 M = MOD(N,4)
      IF( M .EQ. 0 ) GO TO 40
      DO 30 I = 1,M
        DY(I) = DY(I) + DA*DX(I)
   30 CONTINUE
      IF( N .LT. 4 ) RETURN
   40 MP1 = M + 1
      DO 50 I = MP1,N,4
        DY(I) = DY(I) + DA*DX(I)
        DY(I + 1) = DY(I + 1) + DA*DX(I + 1)
```

```
         DY(I + 2) = DY(I + 2) + DA*DX(I + 2)
         DY(I + 3) = DY(I + 3) + DA*DX(I + 3)
   50 CONTINUE
      RETURN
      END
C NAASA  1.1.003 DDOT     FTN-A 05-02-78     THE UNIV OF MICH COMP CTR
      DOUBLE PRECISION FUNCTION DDOT(N,DX,INCX,DY,INCY)
C
C     FORMS THE DOT PRODUCT OF A VECTOR.
C     USES UNROLLED LOOPS FOR INCREMENTS EQUAL TO ONE.
C     JACK DONGARRA, LINPACK, 6/17/77.
C
      DOUBLE PRECISION DX(1),DY(1),DTEMP
      INTEGER I,INCX,INCY,IX,IY,M,MP1,N
C
      DDOT = 0.0D0
      DTEMP = 0.0D0
      IF(N.LE.0)RETURN
      IF(INCX.EQ.1.AND.INCY.EQ.1)GOTO 20
C
C        CODE FOR UNEQUAL INCREMENTS OR EQUAL INCREMENTS
C          NOT EQUAL TO 1
C
      IX = 1
      IY = 1
      IF(INCX.LT.0)IX = (-N+1)*INCX + 1
      IF(INCY.LT.0)IY = (-N+1)*INCY + 1
      DO 10 I = 1,N
        DTEMP = DTEMP + DX(IX)*DY(IY)
        IX = IX + INCX
        IY = IY + INCY
   10 CONTINUE
      DDOT = DTEMP
      RETURN
C
C        CODE FOR BOTH INCREMENTS EQUAL TO 1
C
C
C        CLEAN-UP LOOP
C
   20 M = MOD(N,5)
      IF( M .EQ. 0 ) GO TO 40
      DO 30 I = 1,M
        DTEMP = DTEMP + DX(I)*DY(I)
   30 CONTINUE
      IF( N .LT. 5 ) GO TO 60
   40 MP1 = M + 1
      DO 50 I = MP1,N,5
        DTEMP = DTEMP + DX(I)*DY(I) + DX(I + 1)*DY(I + 1) +
     *   DX(I + 2)*DY(I + 2) + DX(I + 3)*DY(I + 3) + DX(I + 4)*DY(I + 4)
   50 CONTINUE
   60 DDOT = DTEMP
      RETURN
      END
C NAASA  1.1.009 DSCAL    FTN-A 05-02-78     THE UNIV OF MICH COMP CTR
      SUBROUTINE  DSCAL(N,DA,DX,INCX)
C
C     SCALES A VECTOR BY A CONSTANT.
C     USES UNROLLED LOOPS FOR INCREMENT EQUAL TO ONE.
C     JACK DONGARRA, LINPACK, 6/17/77.
C
      DOUBLE PRECISION DA,DX(1)
      INTEGER I,INCX,M,MP1,N,NINCX
C
      IF(N.LE.0)RETURN
      IF(INCX.EQ.1)GOTO 20
C
C        CODE FOR INCREMENT NOT EQUAL TO 1
C
      NINCX = N*INCX
      DO 10 I = 1,NINCX,INCX
        DX(I) = DA*DX(I)
   10 CONTINUE
      RETURN
C
C        CODE FOR INCREMENT EQUAL TO 1
C
C
C        CLEAN-UP LOOP
C
   20 M = MOD(N,5)
      IF( M .EQ. 0 ) GO TO 40
      DO 30 I = 1,M
        DX(I) = DA*DX(I)
   30 CONTINUE
      IF( N .LT. 5 ) RETURN
   40 MP1 = M + 1
      DO 50 I = MP1,N,5
```

```
          DX(I) = DA*DX(I)
          DX(I + 1) = DA*DX(I + 1)
          DX(I + 2) = DA*DX(I + 2)
          DX(I + 3) = DA*DX(I + 3)
          DX(I + 4) = DA*DX(I + 4)
   50 CONTINUE
      RETURN
      END
C NAASA  1.1.020 IDAMAX   FTN-A 05-02-78      THE UNIV OF MICH COMP CTR
      INTEGER FUNCTION IDAMAX(N,DX,INCX)
C
C     FINDS THE INDEX OF ELEMENT HAVING MAX. ABSOLUTE VALUE.
C     JACK DONGARRA, LINPACK, 6/17/77.
C
      DOUBLE PRECISION DX(1),DMAX
      INTEGER I,INCX,IX,N
C
      IDAMAX = 1
      IF(N.LE.1)RETURN
      IF(INCX.EQ.1)GOTO 20
C
C        CODE FOR INCREMENT NOT EQUAL TO 1
C
      IX = 1
      DMAX = DABS(DX(1))
      IX = IX + INCX
      DO 10 I = 2,N
         IF(DABS(DX(IX)).LE.DMAX) GO TO 5
         IDAMAX = I
         DMAX = DABS(DX(IX))
    5    IX = IX + INCX
   10 CONTINUE
      RETURN
C
C        CODE FOR INCREMENT EQUAL TO 1
C
   20 DMAX = DABS(DX(1))
      DO 30 I = 2,N
         IF(DABS(DX(I)).LE.DMAX) GO TO 30
         IDAMAX = I
         DMAX = DABS(DX(I))
   30 CONTINUE
      RETURN
      END
```

```
C NAASA  2.1.042 CGECO    FTN-A 05-02-78     THE UNIV OF MICH COMP CTR
      SUBROUTINE CGECO(A,LDA,N,IPVT,RCOND,Z)
      IMPLICIT REAL*8(A-H,O-Z)
      INTEGER LDA,N,IPVT(1)
      COMPLEX*16 A(LDA,1),Z(1)
      REAL*8 RCOND
C
C     CGECO FACTORS A COMPLEX MATRIX BY GAUSSIAN ELIMINATION
C     AND ESTIMATES THE CONDITION OF THE MATRIX.
C
C     IF  RCOND  IS NOT NEEDED, CGEFA IS SLIGHTLY FASTER.
C     TO SOLVE  A*X = B , FOLLOW CGECO BY CGESL.
C     TO COMPUTE  INVERSE(A)*C , FOLLOW CGECO BY CGESL.
C     TO COMPUTE  DETERMINANT(A) , FOLLOW CGECO BY CGEDI.
C     TO COMPUTE  INVERSE(A) , FOLLOW CGECO BY CGEDI.
C
C     ON ENTRY
C
C        A       COMPLEX(LDA, N)
C                THE MATRIX TO BE FACTORED.
C
C        LDA     INTEGER
C                THE LEADING DIMENSION OF THE ARRAY  A .
C
C        N       INTEGER
C                THE ORDER OF THE MATRIX  A .
C
C     ON RETURN
C
C        A       AN UPPER TRIANGULAR MATRIX AND THE MULTIPLIERS
C                WHICH WERE USED TO OBTAIN IT.
C                THE FACTORIZATION CAN BE WRITTEN  A = L*U  WHERE
C                L  IS A PRODUCT OF PERMUTATION AND UNIT LOWER
C                TRIANGULAR MATRICES AND  U  IS UPPER TRIANGULAR.
C
C        IPVT    INTEGER(N)
C                AN INTEGER VECTOR OF PIVOT INDICES.
C
C        RCOND   REAL
C                AN ESTIMATE OF THE RECIPROCAL CONDITION OF  A .
C                FOR THE SYSTEM  A*X = B , RELATIVE PERTURBATIONS
C                IN  A  AND  B  OF SIZE  EPSILON  MAY CAUSE
C                RELATIVE PERTURBATIONS IN  X  OF SIZE  EPSILON/RCOND .
C                IF  RCOND  IS SO SMALL THAT THE LOGICAL EXPRESSION
```

```
C                       1.0 + RCOND .EQ. 1.0
C                IS TRUE, THEN  A  MAY BE SINGULAR TO WORKING
C                PRECISION.  IN PARTICULAR,  RCOND  IS ZERO  IF
C                EXACT SINGULARITY IS DETECTED OR THE ESTIMATE
C                UNDERFLOWS.
C
C        Z       COMPLEX(N)
C                A WORK VECTOR WHOSE CONTENTS ARE USUALLY UNIMPORTANT.
C                IF  A  IS CLOSE TO A SINGULAR MATRIX, THEN  Z  IS
C                AN APPROXIMATE NULL VECTOR IN THE SENSE THAT
C                NORM(A*Z) = RCOND*NORM(A)*NORM(Z) .
C
C     LINPACK. THIS VERSION DATED 07/14/77 .
C     CLEVE MOLER, UNIVERSITY OF NEW MEXICO, ARGONNE NATIONAL LABS.
C
C     SUBROUTINES AND FUNCTIONS
C
C     LINPACK CGEFA
C     BLAS CAXPY,CDOTC,CSSCAL,SCASUM
C     FORTRAN DABS,DIMAG,DMAX1,DCMPLX,DCONJG,REAL
C
C     INTERNAL VARIABLES
C
      IMPLICIT REAL*8(A-H,O-Z)
      COMPLEX*16 CDOTC,EK,T,WK,WKM
      REAL*8 ANORM,S,SCASUM,SM,YNORM
      INTEGER INFO,J,K,KB,KP1,L
C
      COMPLEX*16 ZDUM,ZDUM1,ZDUM2,CSIGN1
      REAL*8 CABS1
      CABS1(ZDUM) = DABS(DREAL(ZDUM)) + DABS(DIMAG(ZDUM))
      CSIGN1(ZDUM1,ZDUM2) = CABS1(ZDUM1)*(ZDUM2/CABS1(ZDUM2))
C
C     COMPUTE 1-NORM OF A
C
      ANORM = 0.0D0
      DO 10 J = 1, N
         ANORM = DMAX1(ANORM,SCASUM(N,A(1,J),1))
   10 CONTINUE
C
C     FACTOR
C
      CALL CGEFA(A,LDA,N,IPVT,INFO)
C
```

```
C     RCOND = 1/(NORM(A)*(ESTIMATE OF NORM(INVERSE(A)))) .
C     ESTIMATE = NORM(Z)/NORM(Y) WHERE  A*Z = Y  AND  CTRANS(A)*Y = E .
C     CTRANS(A)  IS THE CONJUGATE TRANSPOSE OF A .
C     THE COMPONENTS OF  E  ARE CHOSEN TO CAUSE MAXIMUM LOCAL
C     GROWTH IN THE ELEMENTS OF W  WHERE  CTRANS(U)*W = E .
C     THE VECTORS ARE FREQUENTLY RESCALED TO AVOID OVERFLOW.
C
C     SOLVE CTRANS(U)*W = E
C
      EK = DCMPLX(1.0D0,0.0D0)
      DO 20 J = 1, N
         Z(J) = DCMPLX(0.0D0,0.0D0)
   20 CONTINUE
      DO 100 K = 1, N
         IF (CABS1(Z(K)) .NE. 0.0D0) EK = CSIGN1(EK,-Z(K))
         IF (CABS1(EK-Z(K)) .LE. CABS1(A(K,K))) GO TO 30
            S = CABS1(A(K,K))/CABS1(EK-Z(K))
            CALL CSSCAL(N,S,Z,1)
            EK = DCMPLX(S,0.0D0)*EK
   30    CONTINUE
         WK = EK - Z(K)
         WKM = -EK - Z(K)
         S = CABS1(WK)
         SM = CABS1(WKM)
         IF (CABS1(A(K,K)) .EQ. 0.0D0) GO TO 40
            WK = WK/DCONJG(A(K,K))
            WKM = WKM/DCONJG(A(K,K))
         GO TO 50
   40    CONTINUE
            WK = DCMPLX(1.0D0,0.0D0)
            WKM = DCMPLX(1.0D0,0.0D0)
   50    CONTINUE
         KP1 = K + 1
         IF (KP1 .GT. N) GO TO 90
            DO 60 J = KP1, N
               SM = SM + CABS1(Z(J)+WKM*DCONJG(A(K,J)))
               Z(J) = Z(J) + WK*DCONJG(A(K,J))
               S = S + CABS1(Z(J))
   60       CONTINUE
            IF (S .GE. SM) GO TO 80
               T = WKM - WK
               WK = WKM
               DO 70 J = KP1, N
                  Z(J) = Z(J) + T*DCONJG(A(K,J))
```

```
 70            CONTINUE
 80         CONTINUE
 90      CONTINUE
         Z(K) = WK
100   CONTINUE
      S = 1.0D0/SCASUM(N,Z,1)
      CALL CSSCAL(N,S,Z,1)
C
C     SOLVE CTRANS(L)*Y = V
C
      DO 120 KB = 1, N
         K = N + 1 - KB
         IF (K .LT. N) Z(K) = Z(K) + CDOTC(N-K,A(K+1,K),1,Z(K+1),1)
         IF (CABS1(Z(K)) .LE. 1.0D0) GO TO 110
            S = 1.0D0/CABS1(Z(K))
            CALL CSSCAL(N,S,Z,1)
110      CONTINUE
         L = IPVT(K)
         T = Z(L)
         Z(L) = Z(K)
         Z(K) = T
120   CONTINUE
      S = 1.0D0/SCASUM(N,Z,1)
      CALL CSSCAL(N,S,Z,1)
C
      YNORM = 1.0D0
C
C     SOLVE L*V = Y
C
      DO 140 K = 1, N
         L = IPVT(K)
         T = Z(L)
         Z(L) = Z(K)
         Z(K) = T
         IF (K .LT. N) CALL CAXPY(N-K,T,A(K+1,K),1,Z(K+1),1)
         IF (CABS1(Z(K)) .LE. 1.0D0) GO TO 130
            S = 1.0D0/CABS1(Z(K))
            CALL CSSCAL(N,S,Z,1)
            YNORM = S*YNORM
130      CONTINUE
140   CONTINUE
      S = 1.0D0/SCASUM(N,Z,1)
      CALL CSSCAL(N,S,Z,1)
      YNORM = S*YNORM
```

```
C
C     SOLVE  U*Z = V
C
      DO 160 KB = 1, N
         K = N + 1 - KB
         IF (CABS1(Z(K)) .LE. CABS1(A(K,K))) GO TO 150
            S = CABS1(A(K,K))/CABS1(Z(K))
            CALL CSSCAL(N,S,Z,1)
            YNORM = S*YNORM
  150    CONTINUE
         IF (CABS1(A(K,K)) .NE. 0.0D0) Z(K) = Z(K)/A(K,K)
         IF (CABS1(A(K,K)) .EQ. 0.0D0) Z(K) = DCMPLX(1.0D0,0.0D0)
         T = -Z(K)
         CALL CAXPY(K-1,T,A(1,K),1,Z(1),1)
  160 CONTINUE
C     MAKE ZNORM = 1.0
      S = 1.0D0/SCASUM(N,Z,1)
      CALL CSSCAL(N,S,Z,1)
      YNORM = S*YNORM
C
      IF (ANORM .NE. 0.0D0) RCOND = YNORM/ANORM
      IF (ANORM .EQ. 0.0D0) RCOND = 0.0D0
      RETURN
      END
C NAASA  2.1.044 CGESL    FTN-A 05-02-78      THE UNIV OF MICH COMP CTR
      SUBROUTINE CGESL(A,LDA,N,IPVT,B,JOB)
      IMPLICIT REAL*8(A-H,O-Z)
      INTEGER LDA,N,IPVT(1),JOB
      COMPLEX*16 A(LDA,1),B(1)
C
C     CGESL SOLVES THE COMPLEX SYSTEM
C     A * X = B  OR  CTRANS(A) * X = B
C     USING THE FACTORS COMPUTED BY CGECO OR CGEFA.
C
C     ON ENTRY
C
C        A       COMPLEX(LDA, N)
C                THE OUTPUT FROM CGECO OR CGEFA.
C
C        LDA     INTEGER
C                THE LEADING DIMENSION OF THE ARRAY  A .
C
C        N       INTEGER
C                THE ORDER OF THE MATRIX  A .
```

```
C
C        IPVT    INTEGER(N)
C                THE PIVOT VECTOR FROM CGECO OR CGEFA.
C
C        B       COMPLEX(N)
C                THE RIGHT HAND SIDE VECTOR.
C
C        JOB     INTEGER
C                = 0         TO SOLVE  A*X = B ,
C                = NONZERO   TO SOLVE  CTRANS(A)*X = B  WHERE
C                            CTRANS(A)  IS THE CONJUGATE TRANSPOSE.
C
C     ON RETURN
C
C        B       THE SOLUTION VECTOR  X .
C
C     ERROR CONDITION
C
C        A DIVISION BY ZERO WILL OCCUR IF THE INPUT FACTOR CONTAINS A
C        ZERO ON THE DIAGONAL.  TECHNICALLY THIS INDICATES SINGULARITY
C        BUT IT IS OFTEN CAUSED BY IMPROPER ARGUMENTS OR IMPROPER
C        SETTING OF LDA .  IT WILL NOT OCCUR IF THE SUBROUTINES ARE
C        CALLED CORRECTLY AND IF CGECO HAS SET RCOND .GT. 0.0
C        OR CGEFA HAS SET INFO .EQ. 0 .
C
C     TO COMPUTE  INVERSE(A) * C  WHERE  C  IS A MATRIX
C     WITH  P  COLUMNS
C           CALL CGECO(A,LDA,N,IPVT,RCOND,Z)
C           IF (RCOND IS TOO SMALL) GO TO ...
C           DO 10 J = 1, P
C              CALL CGESL(A,LDA,N,IPVT,C(1,J),0)
C        10 CONTINUE
C
C     LINPACK. THIS VERSION DATED 07/14/77 .
C     CLEVE MOLER, UNIVERSITY OF NEW MEXICO, ARGONNE NATIONAL LABS.
C
C     SUBROUTINES AND FUNCTIONS
C
C     BLAS CAXPY,CDOTC
C     FORTRAN DCONJG
C
C     INTERNAL VARIABLES
C
      COMPLEX*16 CDOTC,T
      INTEGER K,KB,L,NM1
C
      NM1 = N - 1
      IF (JOB .NE. 0) GO TO 50
C
C        JOB = 0 , SOLVE  A * X = B
C        FIRST SOLVE  L*Y = B
C
         IF (NM1 .LT. 1) GO TO 30
         DO 20 K = 1, NM1
            L = IPVT(K)
            T = B(L)
            IF (L .EQ. K) GO TO 10
               B(L) = B(K)
               B(K) = T
   10       CONTINUE
            CALL CAXPY(N-K,T,A(K+1,K),1,B(K+1),1)
   20    CONTINUE
   30    CONTINUE
C
C        NOW SOLVE  U*X = Y
C
         DO 40 KB = 1, N
            K = N + 1 - KB
            B(K) = B(K)/A(K,K)
            T = -B(K)
            CALL CAXPY(K-1,T,A(1,K),1,B(1),1)
   40    CONTINUE
      GO TO 100
   50 CONTINUE
C
C        JOB = NONZERO, SOLVE  CTRANS(A) * X = B
C        FIRST SOLVE  CTRANS(U)*Y = B
C
         DO 60 K = 1, N
            T = CDOTC(K-1,A(1,K),1,B(1),1)
            B(K) = (B(K) - T)/DCONJG(A(K,K))
   60    CONTINUE
C
C        NOW SOLVE CTRANS(L)*X = Y
C
         IF (NM1 .LT. 1) GO TO 90
         DO 80 KB = 1, NM1
            K = N - KB
            B(K) = B(K) + CDOTC(N-K,A(K+1,K),1,B(K+1),1)
            L = IPVT(K)
            IF (L .EQ. K) GO TO 70
               T = B(L)
               B(L) = B(K)
               B(K) = T
   70       CONTINUE
   80    CONTINUE
   90    CONTINUE
  100 CONTINUE
      RETURN
      END
C NAASA  2.1.043 CGEFA    FTN-A 05-02-78      THE UNIV OF MICH COMP CTR
      SUBROUTINE CGEFA(A,LDA,N,IPVT,INFO)
      IMPLICIT REAL*8(A-H,O-Z)
      INTEGER LDA,N,IPVT(1),INFO
      COMPLEX*16 A(LDA,1)
C
C     CGEFA FACTORS A COMPLEX MATRIX BY GAUSSIAN ELIMINATION.
C
C     CGEFA IS USUALLY CALLED BY CGECO, BUT IT CAN BE CALLED
C     DIRECTLY WITH A SAVING IN TIME IF  RCOND  IS NOT NEEDED.
C     (TIME FOR CGECO) = (1 + 9/N)*(TIME FOR CGEFA) .
C
C     ON ENTRY
C
C        A       COMPLEX(LDA, N)
C                THE MATRIX TO BE FACTORED.
C
C        LDA     INTEGER
C                THE LEADING DIMENSION OF THE ARRAY  A .
C
C        N       INTEGER
C                THE ORDER OF THE MATRIX  A .
C
C     ON RETURN
C
C        A       AN UPPER TRIANGULAR MATRIX AND THE MULTIPLIERS
C                WHICH WERE USED TO OBTAIN IT.
C                THE FACTORIZATION CAN BE WRITTEN  A = L*U  WHERE
C                L  IS A PRODUCT OF PERMUTATION AND UNIT LOWER
C                TRIANGULAR MATRICES AND  U  IS UPPER TRIANGULAR.
C
C        IPVT    INTEGER(N)
```

```
C                AN INTEGER VECTOR OF PIVOT INDICES.
C
C        INFO    INTEGER
C                = 0  NORMAL VALUE.
C                = K  IF  U(K,K) .EQ. 0.0 .  THIS IS NOT AN ERROR
C                     CONDITION FOR THIS SUBROUTINE, BUT IT DOES
C                     INDICATE THAT CGESL OR CGEDI WILL DIVIDE BY ZERO
C                     IF CALLED.  USE  RCOND  IN CGECO FOR A RELIABLE
C                     INDICATION OF SINGULARITY.
C
C     LINPACK. THIS VERSION DATED 07/14/77 .
C     CLEVE MOLER, UNIVERSITY OF NEW MEXICO, ARGONNE NATIONAL LABS.
C
C     SUBROUTINES AND FUNCTIONS
C
C     BLAS CAXPY,CSCAL,ICAMAX
C     FORTRAN DABS,DIMAG,DCMPLX,DREAL
C
C     INTERNAL VARIABLES
C
      COMPLEX*16 T
      INTEGER ICAMAX,J,K,KP1,L,NM1
C
      COMPLEX*16 ZDUM
      REAL*8 CABS1
      CABS1(ZDUM) = DABS(DREAL(ZDUM)) + DABS(DIMAG(ZDUM))
C
C     GAUSSIAN ELIMINATION WITH PARTIAL PIVOTING
C
      INFO = 0
      NM1 = N - 1
      IF (NM1 .LT. 1) GO TO 70
      DO 60 K = 1, NM1
         KP1 = K + 1
C
C        FIND L = PIVOT INDEX
C
         L = ICAMAX(N-K+1,A(K,K),1) + K - 1
         IPVT(K) = L
C
C        ZERO PIVOT IMPLIES THIS COLUMN ALREADY TRIANGULARIZED
C
         IF (CABS1(A(L,K)) .EQ. 0.0D0) GO TO 40
C
```

```
C           INTERCHANGE IF NECESSARY
C
            IF (L .EQ. K) GO TO 10
               T = A(L,K)
               A(L,K) = A(K,K)
               A(K,K) = T
   10       CONTINUE
C
C           COMPUTE MULTIPLIERS
C
            T = -DCMPLX(1.0D0,0.0D0)/A(K,K)
            CALL CSCAL(N-K,T,A(K+1,K),1)
C
C           ROW ELIMINATION WITH COLUMN INDEXING
C
            DO 30 J = KP1, N
               T = A(L,J)
               IF (L .EQ. K) GO TO 20
                  A(L,J) = A(K,J)
                  A(K,J) = T
   20          CONTINUE
               CALL CAXPY(N-K,T,A(K+1,K),1,A(K+1,J),1)
   30       CONTINUE
         GO TO 50
   40    CONTINUE
            INFO = K
   50    CONTINUE
   60 CONTINUE
   70 CONTINUE
      IPVT(N) = N
      IF (CABS1(A(N,N)) .EQ. 0.0D0) INFO = N
      RETURN
      END
C NAASA  1.1.014 CAXPY    FTN-A 05-02-78     THE UNIV OF MICH COMP CTR
      SUBROUTINE CAXPY(N,CA,CX,INCX,CY,INCY)
C
C     CONSTANT TIMES A VECTOR PLUS A VECTOR.
C     JACK DONGARRA, LINPACK, 6/17/77.
C
      IMPLICIT REAL*8(A-H,O-Z)
      COMPLEX*16 CX(1),CY(1),CA
      INTEGER I,INCX,INCY,IX,IY,N
C
      IF(N.LE.0)RETURN
      IF (DABS(DREAL(CA)) + DABS(DIMAG(CA)) .EQ. 0.0D0 ) RETURN
      IF(INCX.EQ.1.AND.INCY.EQ.1)GOTO 20
C
C        CODE FOR UNEQUAL INCREMENTS OR EQUAL INCREMENTS
C          NOT EQUAL TO 1
C
      IX = 1
      IY = 1
      IF(INCX.LT.0)IX = (-N+1)*INCX + 1
      IF(INCY.LT.0)IY = (-N+1)*INCY + 1
      DO 10 I = 1,N
        CY(IY) = CY(IY) + CA*CX(IX)
        IX = IX + INCX
        IY = IY + INCY
   10 CONTINUE
      RETURN
C
C        CODE FOR BOTH INCREMENTS EQUAL TO 1
C
   20 DO 30 I = 1,N
        CY(I) = CY(I) + CA*CX(I)
   30 CONTINUE
      RETURN
      END
C NAASA  1.1.012 CDOTC     FTN-A 05-02-78      THE UNIV OF MICH COMP CTR
      COMPLEX*16 FUNCTION CDOTC(N,CX,INCX,CY,INCY)
C
C     FORMS THE DOT PRODUCT OF TWO VECTORS, CONJUGATING THE FIRST
C     VECTOR.
C     JACK DONGARRA, LINPACK,  6/17/77.
C
      IMPLICIT REAL*8(A-H,O-Z)
      COMPLEX*16 CX(1),CY(1),CTEMP
      INTEGER I,INCX,INCY,IX,IY,N
C
      CTEMP = DCMPLX(0.0D0,0.0D0)
      CDOTC = DCMPLX(0.0D0,0.0D0)
      IF(N.LE.0)RETURN
      IF(INCX.EQ.1.AND.INCY.EQ.1)GOTO 20
C
C        CODE FOR UNEQUAL INCREMENTS OR EQUAL INCREMENTS
C          NOT EQUAL TO 1
C
      IX = 1
      IY = 1
      IF(INCX.LT.0)IX = (-N+1)*INCX + 1
      IF(INCY.LT.0)IY = (-N+1)*INCY + 1
      DO 10 I = 1,N
        CTEMP = CTEMP + DCONJG(CX(IX))*CY(IY)
        IX = IX + INCX
        IY = IY + INCY
   10 CONTINUE
      CDOTC = CTEMP
      RETURN
C
C        CODE FOR BOTH INCREMENTS EQUAL TO 1
C
   20 DO 30 I = 1,N
        CTEMP = CTEMP + DCONJG(CX(I))*CY(I)
   30 CONTINUE
      CDOTC = CTEMP
      RETURN
      END
C NAASA  1.1.018 CSSCAL   FTN-A 05-02-78     THE UNIV OF MICH COMP CTR
      SUBROUTINE  CSSCAL(N,SA,CX,INCX)
C
C     SCALES A COMPLEX VECTOR BY A REAL CONSTANT.
C     JACK DONGARRA, LINPACK, 6/17/77.
C
      IMPLICIT REAL*8(A-H,O-Z)
      COMPLEX*16 CX(1)
      REAL*8 SA
      INTEGER I,INCX,N,NINCX
C
      IF(N.LE.0)RETURN
      IF(INCX.EQ.1)GOTO 20
C
C        CODE FOR INCREMENT NOT EQUAL TO 1
C
      NINCX = N*INCX
      DO 10 I = 1,NINCX,INCX
        CX(I) = DCMPLX(SA*DREAL(CX(I)),SA*DIMAG(CX(I)))
   10 CONTINUE
      RETURN
C
C        CODE FOR INCREMENT EQUAL TO 1
C
   20 DO 30 I = 1,N
```

```
         CX(I) = DCMPLX(SA*DREAL(CX(I)),SA*DIMAG(CX(I)))
   30 CONTINUE
      RETURN
      END
C NAASA  1.1.010 SCASUM   FTN-A 05-02-78     THE UNIV OF MICH COMP CTR
      REAL*8 FUNCTION SCASUM(N,CX,INCX)
C
C     TAKES THE SUM OF THE ABSOLUTE VALUES OF A COMPLEX VECTOR AND
C     RETURNS A SINGLE PRECISION RESULT.
C     JACK DONGARRA, LINPACK, 6/17/77.
C
      IMPLICIT REAL*8(A-H,O-Z)
      COMPLEX*16 CX(1)
      REAL*8 STEMP
      INTEGER I,INCX,N,NINCX
C
      SCASUM = 0.0D0
      STEMP = 0.0D0
      IF(N.LE.0)RETURN
      IF(INCX.EQ.1)GOTO 20
C
C        CODE FOR INCREMENT NOT EQUAL TO 1
C
      NINCX = N*INCX
      DO 10 I = 1,NINCX,INCX
        STEMP = STEMP + DABS(DREAL(CX(I))) + DABS(DIMAG(CX(I)))
   10 CONTINUE
      SCASUM = STEMP
      RETURN
C
C        CODE FOR INCREMENT EQUAL TO 1
C
   20 DO 30 I = 1,N
        STEMP = STEMP + DABS(DREAL(CX(I))) + DABS(DIMAG(CX(I)))
   30 CONTINUE
      SCASUM = STEMP
      RETURN
      END
C NAASA  1.1.019 CSCAL    FTN-A 05-02-78     THE UNIV OF MICH COMP CTR
      SUBROUTINE  CSCAL(N,CA,CX,INCX)
C
C     SCALES A VECTOR BY A CONSTANT.
C     JACK DONGARRA, LINPACK,  6/17/77.
C
```

```
      IMPLICIT REAL*8 (A-H,O-Z)
      COMPLEX*16 CA,CX(1)
      INTEGER I,INCX,N,NINCX
C
      IF(N.LE.0)RETURN
      IF(INCX.EQ.1)GOTO 20
C
C        CODE FOR INCREMENT NOT EQUAL TO 1
C
      NINCX = N*INCX
      DO 10 I = 1,NINCX,INCX
        CX(I) = CA*CX(I)
   10 CONTINUE
      RETURN
C
C        CODE FOR INCREMENT EQUAL TO 1
C
   20 DO 30 I = 1,N
        CX(I) = CA*CX(I)
   30 CONTINUE
      RETURN
      END
C NAASA  1.1.021 ICAMAX   FTN-A 05-02-78      THE UNIV OF MICH COMP CTR
      INTEGER FUNCTION ICAMAX(N,CX,INCX)
C
C     FINDS THE INDEX OF ELEMENT HAVING MAX. ABSOLUTE VALUE.
C     JACK DONGARRA, LINPACK, 6/17/77.
C
      IMPLICIT REAL*8 (A-H,O-Z)
      COMPLEX*16 CX(1)
      REAL*8 SMAX
      INTEGER I,INCX,IX,N
      COMPLEX*16 ZDUM
      REAL*8 CABS1
      CABS1(ZDUM) = DABS(DREAL(ZDUM)) + DABS(DIMAG(ZDUM))
C
      ICAMAX = 1
      IF(N.LE.1)RETURN
      IF(INCX.EQ.1)GOTO 20
C
C        CODE FOR INCREMENT NOT EQUAL TO 1
C
      IX = 1
      SMAX = CABS1(CX(1))
      IX = IX + INCX
      DO 10 I = 2,N
         IF(CABS1(CX(IX)).LE.SMAX) GO TO 5
         ICAMAX = I
         SMAX = CABS1(CX(IX))
    5    IX = IX + INCX
   10 CONTINUE
      RETURN
C
C        CODE FOR INCREMENT EQUAL TO 1
C
   20 SMAX = CABS1(CX(1))
      DO 30 I = 2,N
         IF(CABS1(CX(I)).LE.SMAX) GO TO 30
         ICAMAX = I
         SMAX = CABS1(CX(I))
   30 CONTINUE
      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCC
C     DREAL DOESN'T SEEM TO WORK, SO THIS FUNCTION IS A SUBSTITUTE
      REAL*8 FUNCTION DREAL(X)
      COMPLEX*16 X,X2
      REAL*8 XA(2)
      EQUIVALENCE (X2,XA(1))
      X2=X
      DREAL=XA(1)
      RETURN
      END
```

BIBLIOGRAPHY

Bergman, D. J. (1978), "The Dielectric Constant of a Composite Material - A Problem in Classical Physics," Phys. Rep., Vol. 43, 377-407.

Bohren, C. F. and L. J. Battan (1980), "Radar Backscattering by Inhomogeneous Precipitation Particles," J. Atmos. Sci., Vol. 37, 1821-1827.

Gradshteyn, I. S. and I. M. Ryzhik (1980), "Table of Integrals, Series, and Products," Academic Press, New York.

Granqvist, C. G. and O. Hunderi (1978), "Optical Properties of Ag-SiO_2 Cermet Films: A Comparison of Effective-Medium Theories," Phys. Rev. B, Vol. 18, 2897-2906.

Harrington, R. F. and J. R. Mautz (1975), "An Impedance Sheet Approximation for Thin Dielectric Shells," IEEE Trans. Antennas Propag., Vol. AP-23, 531-534.

Harrington, R. F. (1982), "Field Computation by Moment Methods," Krieger, Malabar, FL.

Herrick, D. F. (1976), "Analytical Evaluation of Kernels," Radiation Laboratory Memo No. 013714-512-M, University of Michigan.

Inspektorov, E. M. (1982), "Using Integral Equations of the Second Kind for Analysis of Diffraction of Thin Shields," Izv. Vyssh, Uchebn. Zaved. Radiofiz., Vol. 25, 1099-1101.

Jackson, J. D. (1975), "Classical Electrodynamics," Wiley, New York.

Keller, J. B., R. E. Kleinman, and T.B.A. Senior (1972), "Dipole Moments in Rayleigh Scattering," J. Inst. Math. Its Appl., Vol. 9, 14-22.

Kleinman, R. E. (1965), "Low Frequency Solution of Three-Dimensional Scattering Problems," Radiation Laboratory Report #7133-4-T, University of Michigan.

Kock, W. E. (1948), "Metallic Delay Lenses," Bell Syst. Tech. J., Vol. 27, 58-83.

Maxwell Garnett, J. C. (1904), "Colours in Metal Glasses and in Metallic Films," Philos. Trans. R. Soc. London, Vol. 203, 385-420.

Mosotti, O. F. (1850), Mem. Soc. Ital., Vol. 14, 49.

Polder, D. and J. H. Van Santen (1946), "The Effective Permeability of Mixtures of Solids," Physica, Vol. 12, 257-271.

Senior, T.B.A. (1975), "Low Frequency Scattering Data for Dielectric Bodies," Radiation Laboratory Memo #013714-505-M, University of Michigan.

Senior, T.B.A. (1976), "Low Frequency Scattering by a Dielectric Body," Radio Sci., Vol. 11, 477-482.

Senior, T.B.A. (1982), "Low-Frequency Scattering by a Perfectly Conducting Body," Radio Sci., Vol. 17, 741-746.

Senior, T.B.A. and H. Weil (1982), "On the Validity of Modeling Rayleigh Scatterers by Spheroids," Appl. Phys. B., Vol. 29, 117-124.

Senior, T.B.A. and T. M. Willis (1982), "Rayleigh Scattering by Dielectric Bodies," IEEE Trans. Antennas and Propag., Vol. AP-30, 1271.

Senior, T.B.A. (1983), "Low Frequency Scattering by a Metallic Plate," Electromagnetics, Vol. 3, 131-144.

Senior, T.B.A. and D.A. Ksienski (1984), "Determination of a Vector Potential," Radio Sci., Vol. 19, 603-607.

Senior, T.B.A. and M. Naor (1984), "Low Frequency Scattering by a Resistive Plate," IEEE Trans. Antennas Propag. Vol. AP-23, 272-275.

Stevenson, A. F. (1953), "Solution of Electromagnetic Scattering Problems as Power Series in the Ratio (Dimension of Scatterer)/Wavelength," J. Appl. Phys., Vol. 24, 1134-1142.

Stevenson, A.F. (1954), "Note on the Existence and Determination of a Vector Potential," Quart. Appl. Math., Vol. 12, 194-197.

Von Hippel, A. R. (1954), "Dielectrics and Waves," Wiley, New York.

Weil, H. (1984), private communications.

Willis, T. M. (1982), "Low Frequency Scattering by a Thin Dielectric Plate," Radiation Laboratory Memo #01955-502-M, University of Michigan.

Wilton, D. R., S. M. Rao, A. W. Glisson, D. H. Schaubert, O. M. Al-Bundak, and C. M. Butler (1984), "Potential Integrals for Uniform and Linear Source Distributions on Polygonal and Polyhedral Domains," IEEE Trans. Antennas Propag., Vol. AP-32, 276-281.

Zienkiewicz, O. C. (1982), "The Finite Element Method," McGraw-Hill, London.