

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

Doctoral Committee:

Professor Thomas B.A. Senior, Chairman
Professor Chiao-Min Chu
Professor Ward D. Getty
Professor Wilfred Kaplan
Professor Chen-To Tai
Professor Herschel Weil

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

	<u>Page</u>
DEDICATION	ii
ACKNOWLEDGMENT	iii
LIST OF FIGURES	v
LIST OF TABLES	vi
CHAPTER	
I. INTRODUCTION	1
II. LOW FREQUENCY SCATTERING	7
2.1 The Low Frequency Expansion	7
2.2 Scattering from a Plate	10
III. STATIC SCATTERING FROM A DIELECTRIC PLATE	18
3.1 The Dipole Moment	18
3.2 Static Scattering from a Thin Dielectric Plate	18
3.3 An Overview of Numerical Methods	24
3.4 Functional Description of the Program	27
IV. THE DIPOLE MOMENT: NUMERICAL RESULTS	48
4.1 Accuracy of the Results	48
4.2 A Linear Predictor Model	53
4.3 Shape Effects	69
V. SCATTERING BY DISTRIBUTIONS OF PARTICLES	73
5.1 The Clausius-Mosotti-Lorentz-Lorenz Equation	73
5.2 Cubical Distributions of Particles	80
5.3 A Cubical Exclusion Volume	82
5.4 Restriction of the Clausius-Mosotti Equation to Distributions of Plates	87
VI. CONCLUSIONS	91
APPENDIX	94
BIBLIOGRAPHY	172

LIST OF FIGURES

<u>Figure</u>		<u>Page</u>
3.1	Diagram of a Plate.	20
3.2(a)	Definition of Elements for a Circular Plate.	28
3.2(b)	Input to Computer Program to Define a Circular Plate.	29
3.3(a)	Definition of Elements for a Square Plate.	30
3.3(b)	Input to Computer Program to Define a Square Plate.	31
3.4(a)	Definition of Elements for a Rectangular Plate.	32
3.4(b)	Input to Computer Program to Define a Rectangular Plate.	33
3.5	Dipole Moment and Calculated Potentials for the Square Plate Defined in Fig. 3.3. Under x Excitation, $\tau = 2$, $\ell/t = 10$.	42
3.6	Dipole Moment and Calculated Potentials for the Square Plate Defined in Fig. 3.3. Under x Excitation, $\tau = 101$, $\ell/t = 10$.	43
3.7	Perspective Plot of Potential on Top Surface of Square Plate Shown Partially Hidden by $\phi = 0$ Reference Square: x Excitation, $\ell/t = 10$, $\tau = 2$.	44
3.8	Perspective Plot of Potential on Top Surface of Square Plate Imposed on $\phi = 0$ Reference Square: z Excitation, $\ell/t = 10$, $\tau = 2$.	45
3.9	Perspective Plot of Potential on Top Surface of Square Plate Shown Partially Hidden by $\phi = 0$ Reference Square: x Excitation, $\ell/t = 10$, $\tau = 101$.	46
3.10	Perspective Plot of Potential on Top Surface of Square Plate Imposed on $\phi = 0$ Reference Square: z Excitation, $\ell/t = 10$, $\tau = 101$.	47
4.1	Diagram of a Plate Used by Linear Predictor.	55
5.1	Parallel Plate Capacitor with Spherical Exclusion Volume.	75

LIST OF TABLES

<u>Table</u>		<u>Page</u>
4.1	P_{11}/V for Plates with Circular Cross-Section.	50
4.2	P_{11}/V for Plates with Square Cross-Section.	51
4.3	P_{33}/V for Plates with Square Cross-Section	52
4.4	P_{11}/V for Several Convex Plates with Length/Width = 1	59
4.5	P_{33}/V for Several Convex Plates with Length/Width = 1	60
4.6	P_{11}/V for Several Convex Plates with Length/Width = 2	61
4.7	P_{22}/V for Several Convex Plates with Length/Width = 2	62
4.8	P_{33}/V for Several Convex Plates with Length/Width = 2	63
4.9	P_{11}/V for Several Convex Plates with Length/Width = 2 and complex permittivity	64
4.10	P_{22}/V for Several Convex Plates with Length/Width = 2 and complex permittivity	65
4.11	P_{33}/V for Several Convex Plates with Length/Width = 2 and complex permittivity	66
4.12	Eigenvalues Extracted by Linear Predictor	68

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 $\nabla \times \vec{F} = \vec{f}$ 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., Zienkiewicz, 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

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

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,

$$\begin{aligned} \bar{E} &= \sum_{m=0}^{\infty} \bar{E}_m (ik)^m , & \bar{H} &= \sum_{m=0}^{\infty} \bar{H}_m (ik)^m \\ \rho &= \sum_{m=0}^{\infty} \rho_m (ik)^m , & \bar{J} &= \sum_{m=0}^{\infty} \bar{J}_m (ik)^m . \end{aligned}$$

When the expansions are inserted into Maxwell's equations

$$\begin{aligned} \nabla \times \bar{H} &= \bar{J} + \frac{\partial \bar{D}}{\partial t} \\ \nabla \times \bar{E} &= - \frac{\partial \bar{B}}{\partial t} \end{aligned}$$

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

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

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

The constant Z_0 is the free space impedance, $\bar{\mathbf{H}}_1$ is the first order magnetic field, and $\bar{\mathbf{E}}_0$ 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 $\bar{\mathbf{H}}_1$ given $\bar{\mathbf{E}}_0$. Since $\bar{\mathbf{E}}_0$ 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

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

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

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

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 $\bar{\phi}$ 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

$$\nabla_s \cdot \bar{J}_1 = c\rho_0 \quad (2.2)$$

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

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

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

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

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

$$\begin{aligned}\bar{E}_0 &= Z_0 \nabla \int_B \bar{J}_1 \cdot \nabla' G \, ds' \\ &= -Z_0 \nabla \int_B G \nabla'_S \cdot \bar{J}_1 \, ds' + Z_0 \nabla \int_{C_B} (\bar{J}_1 G) \cdot \hat{\tau}' \, d\ell' ,\end{aligned}$$

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

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

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

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

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 convex. 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 $\hat{\tau}_1 \cdot \bar{J}_1 = 0$. 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

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

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

$$\int_{C_B} \nabla_S \psi \cdot \hat{\tau}' d\ell' = - \int_{C_B} \bar{J}^{(1)} \cdot \hat{\tau}' d\ell' \quad (\text{cont.})$$

$$\begin{aligned} &= - \int_B \nabla_s \cdot \bar{J}^{(1)} ds' \\ &= - \int_B c \rho_0 ds' \\ &= 0 \end{aligned}$$

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)

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

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 $\vec{E}^{inc} = \hat{x}$, 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.

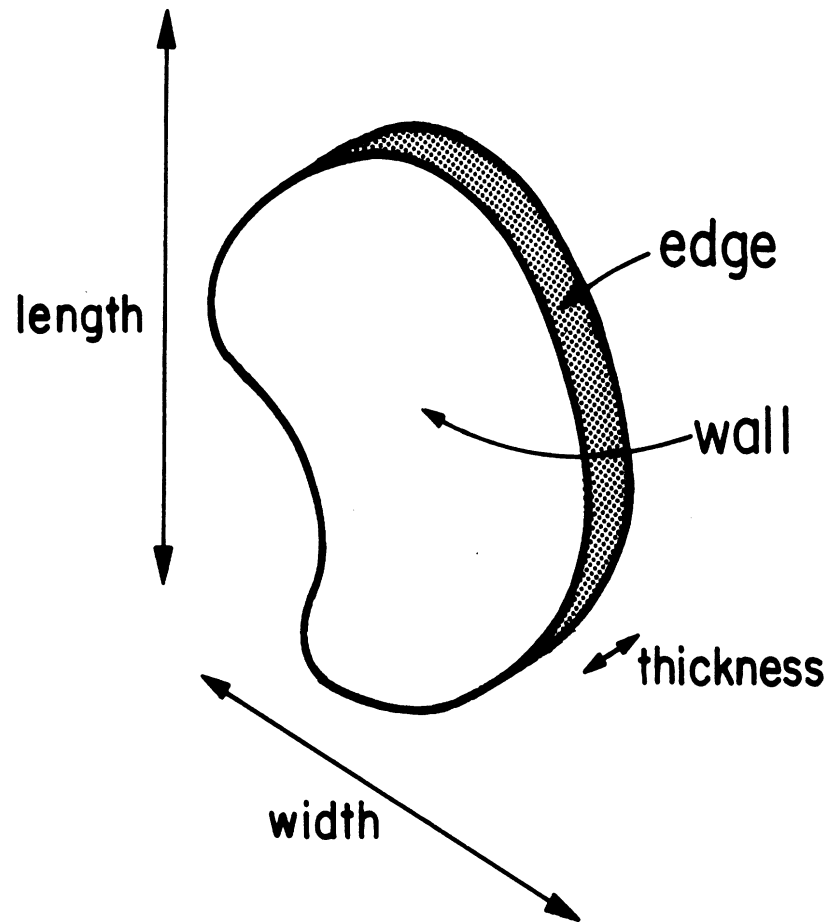


Fig. 3.1: Diagram of a Plate.

$$\phi^t = -\frac{2}{1+\tau} x + \frac{1-\tau}{1+\tau} 2 \left\{ \int_c \int_{-t/2}^{t/2} \phi^t \frac{\partial G}{\partial n'} dz' d\mathbf{l}' + \int_s \int_{-t/2}^{t/2} \frac{\partial}{\partial x'} \left(\phi^t \frac{\partial G}{\partial z'} \right) dz' ds' \right\}$$

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 $z = t/2$.

$$\begin{aligned} \phi^t &= -\frac{2}{1+\tau} x + \frac{1-\tau}{1+\tau} 2 \int_{-t/2}^{t/2} \int_s \left\{ \nabla'_s \cdot (\phi^t \nabla'_s G) + \frac{\partial}{\partial z'} \left(\phi^t \frac{\partial G}{\partial z'} \right) \right\} ds' dz' \\ &= -\frac{2}{1+\tau} x + \frac{1-\tau}{1+\tau} 2 \int_{-t/2}^{t/2} \int_s \left\{ \nabla'_s \phi^t \cdot \nabla'_s G + \phi^t \nabla_s^2 G + \frac{\partial \phi^t}{\partial z'} \frac{\partial G}{\partial z'} + \phi^t \frac{\partial^2 G}{\partial z'^2} \right\} ds' dz' \\ &= -\frac{2}{1+\tau} x + \frac{1-\tau}{1+\tau} 2 \int_{-t/2}^{t/2} \int_s \left\{ -\phi^t \delta(\bar{R}-\bar{R}') + \nabla'_s \phi^t \cdot \nabla'_s G + \frac{\partial \phi^t}{\partial z'} \frac{\partial G}{\partial z'} \right\} ds' dz' \end{aligned}$$

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,

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

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

$$\phi^t = -x + (1-\tau) \int_{-t/2}^{t/2} \int_S \left\{ \nabla'_S \phi^t \cdot \nabla'_S G + \frac{\partial \phi^t}{\partial z'} \frac{\partial G}{\partial z'} \right\} ds' dz' . \quad (3.2)$$

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

$$\phi^t = \sum_{i=0}^{\infty} \phi_{2i}^t (z/t)^{2i} \quad (3.3)$$

i.e., odd powers of (z/t) are unneeded. To simplify Eq. (3.2) only ϕ_0^t will be kept in which case the first equation to be solved is

$$\phi_0^t = -x + (1-\tau) \int_{-t/2}^{t/2} \int_S \nabla'_S \phi_0^t \cdot \nabla'_S G ds' dz' . \quad (3.4)$$

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

$$\begin{aligned}
 \phi^t &= -\frac{2}{1+\tau} x + \frac{1-\tau}{1+\tau} 2 \int_V \nabla' \cdot (\phi^t \nabla' G) dv' \\
 &= -\frac{2}{1+\tau} x + \frac{1-\tau}{1+\tau} 2 \int_V \left\{ \nabla' \phi^t \cdot \nabla' G + \phi^t \nabla'^2 G \right\} dv' \\
 &= -\frac{2}{1+\tau} x - \frac{1-\tau}{1+\tau} \phi^t + \frac{1-\tau}{1+\tau} 2 \int_{-t/2}^{t/2} \int_S \nabla' \phi^t \cdot \nabla' G ds' dz'
 \end{aligned}$$

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

$$\phi^t = -z + (1-\tau) \int_{-t/2}^{t/2} \int_S \left\{ \nabla'_S \phi^t \cdot \nabla'_S G + \frac{\partial \phi^t}{\partial z'} \frac{\partial G}{\partial z'} \right\} ds' dz' \quad (3.5)$$

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

$$\phi^t = \sum_{i=0}^{\infty} \phi_{2i+1}^t (z/t)^{2i+1}$$

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

$$\begin{aligned} \phi_1^t \left(\frac{z}{t} \right) &= -z + (1 - \tau) \int_{-t/2}^{t/2} \int_S \phi_1^t \frac{\partial G}{\partial z'} ds' dz' \\ &= -z + (1 - \tau) \int_S \frac{1}{t} \phi_1^t G \Big|_{-t/2}^{t/2} ds' \end{aligned}$$

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

$$\begin{aligned} \frac{1}{2} \phi_1^t &= -\frac{t}{2} + \frac{1 - \tau}{t} \int_S \phi_1^t \left[G(z = t/2 | z' = t/2) \right. \\ &\quad \left. - G(z = t/2 | z' = -t/2) \right] ds' \end{aligned} \quad (3.6)$$

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 \vec{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 (Zienkiewicz, 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

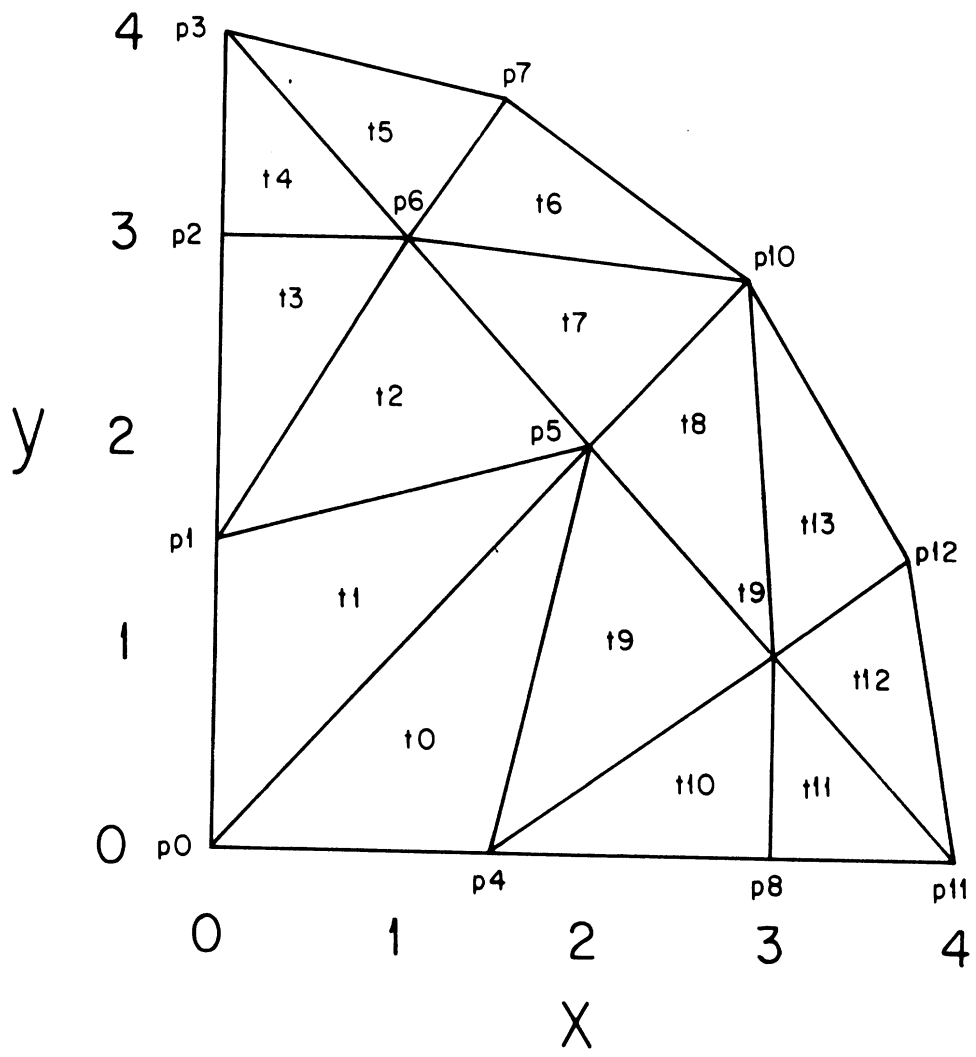


Fig. 3.2(a): Definition of Elements for a Circular Plate.

```
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.

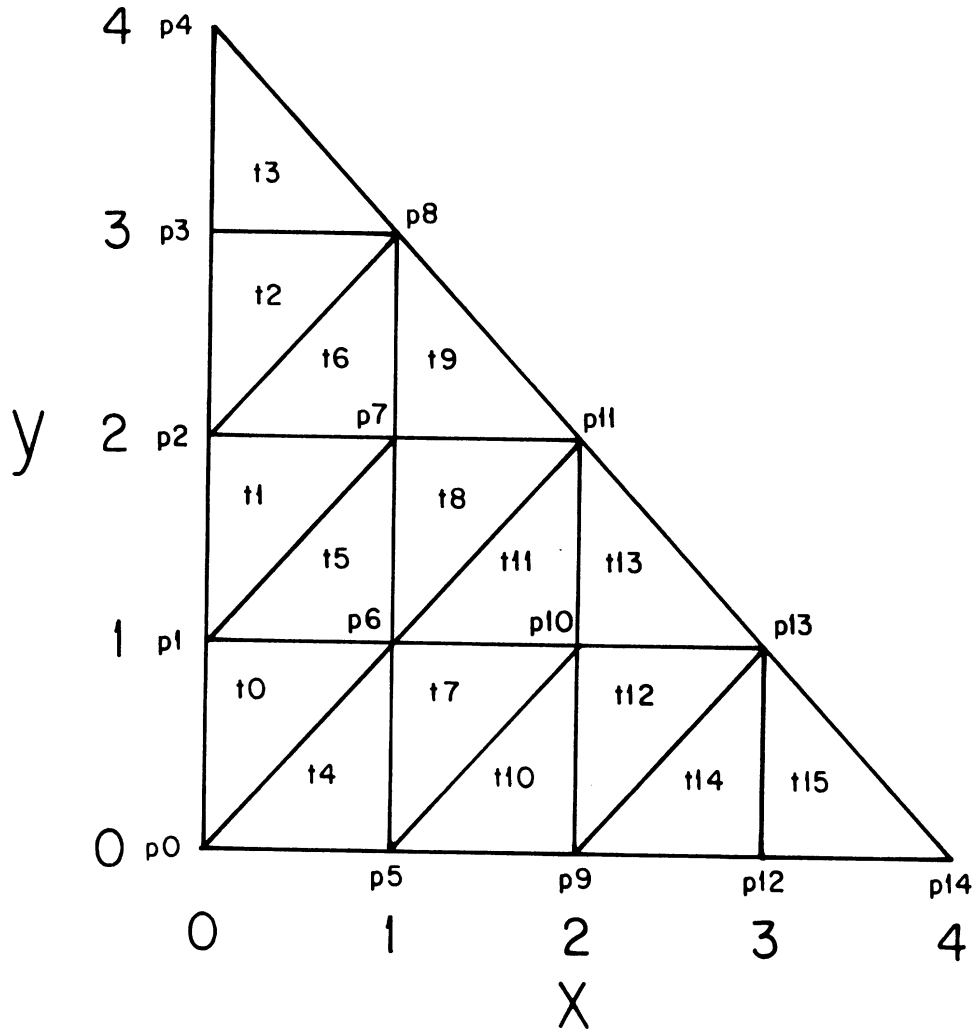


Fig. 3.3(a): Definition of Elements for a Square Plate.

```
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.

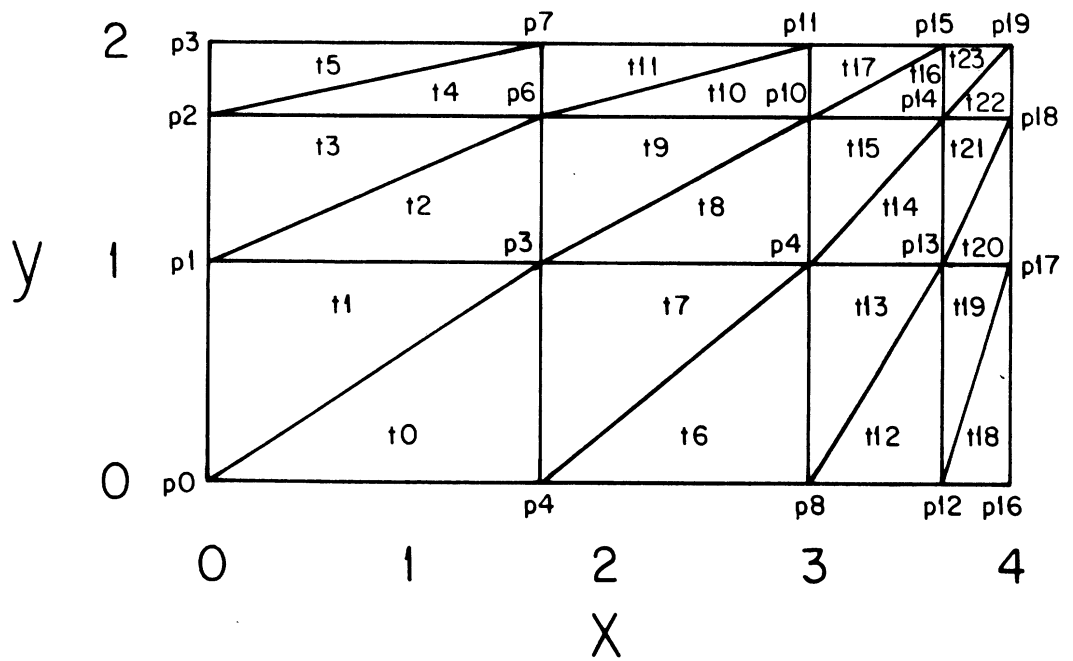


Fig. 3.4(a): Definition of Elements for a Rectangular Plate.


```
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_0^t is restricted to be linearly varying, so that for each element

$$\phi_0^t = C_x x + C_y y + C_0 .$$

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

$$\phi_0^t = -x + (1 - \tau) \int_{-t/2}^{t/2} \int_{S_i} |\bar{c}| \frac{\partial G}{\partial u'} ds' dz' .$$

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

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

From Gradshteyn and Ryzhik (1980, Sec. 2.261)

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

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

and

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

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

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

$$I_1 = \frac{1}{\sqrt{1+a^2}} \ln(2\sqrt{1+a^2} \{(1+a^2)v'^2 + 2(a(b-u) - v)v' + (b-u)^2 + v^2 + (z' - z)^2\}^{1/2} + 2(1+a^2)v' + 2(a(b-u) - v)) \Big|_{v'=V_1}^{v'=V_2}$$

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

$$I = \int_{-t}^0 \frac{1}{\sqrt{1+a^2}} \ln(2\sqrt{1+a^2} \{(1+a^2)v'^2 + 2(a(b-u) - v)v' + (b-u)^2 + v^2 + \eta^2\}^{1/2} + 2(1+a^2)v' + 2(a(b-u) - v)) \Big|_{v'=V_1}^{v'=V_2} d\eta$$

then, change parameters

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

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

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

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

Then, integrating by parts

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

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

To evaluate the second term, call it I_2 ;

$$\begin{aligned} I_2 &= \int_{-t}^0 2\eta^2 \frac{(B + \eta^2)^{-1/2}}{2A(B + \eta^2)^{1/2} + C} \Big|_{V_1}^{V_2} d\eta \\ &= \int_0^t 2\eta^2 \frac{(B + \eta^2)^{-1/2}}{2A(B + \eta^2)^{1/2} + C} \Big|_{V_1}^{V_2} d\eta . \end{aligned}$$

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

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

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

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

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

Changing parameters again,

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

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

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

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

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);

$$\begin{aligned}
 I_3 &= \sqrt{R} \begin{vmatrix} z_2 \\ z_1 \end{vmatrix} \begin{vmatrix} V_2 \\ V_1 \end{vmatrix} + a \left\{ -\frac{1}{\sqrt{a}} \ln \frac{2a + b\gamma + 2\sqrt{a}\sqrt{R}}{\gamma} \right\} \begin{vmatrix} V_2 \\ V_1 \end{vmatrix} \begin{vmatrix} z_2 \\ z_1 \end{vmatrix} \\
 &+ \frac{b}{2} \{ \ln(2\sqrt{R} + 2\gamma + b) \} \begin{vmatrix} V_2 \\ V_1 \end{vmatrix} \begin{vmatrix} z_2 \\ z_1 \end{vmatrix} \\
 I &= \frac{\eta}{A} \ln (2A\{B + \eta^2\}^{1/2} + C) \begin{vmatrix} V_2 \\ V_1 \end{vmatrix} \begin{vmatrix} 0 \\ -t \end{vmatrix} - \frac{1}{2A^2} I_3 .
 \end{aligned}$$

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 (Zienkiewicz, 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 definition 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 definition 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, $\tau = 101$, $\ell/t = 10$.

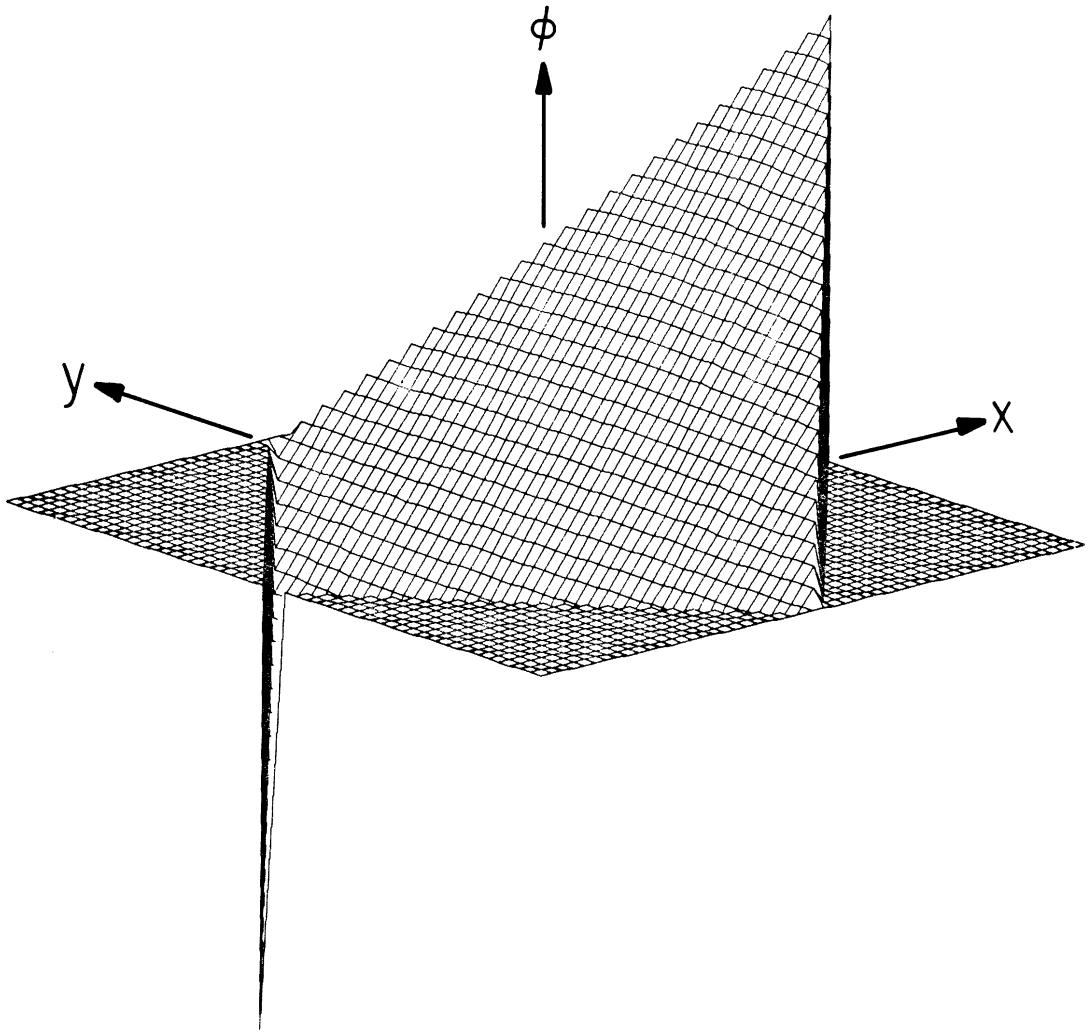


Fig. 3.7: Perspective Plot of Potential on Top Surface of Square Plate Shown Partially Hidden by $\phi = 0$ Reference Square: x Excitation, $l/t = 10$, $\tau = 2$.

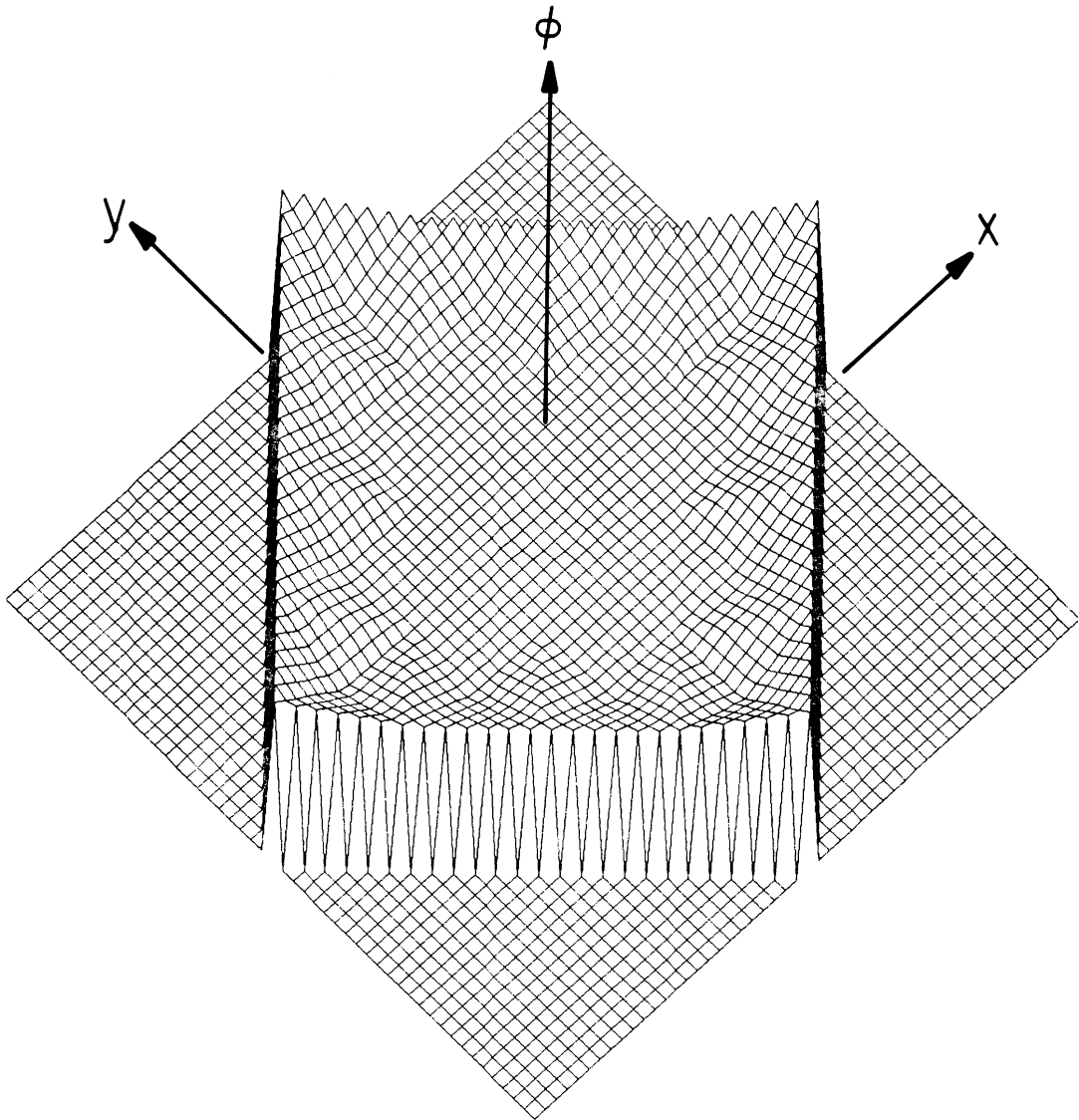


Fig. 3.8: Perspective Plot of Potential on Top Surface of Square Plate Imposed on $\phi = 0$ Reference Square: z Excitation, $\omega/t = 10$, $\tau = 2$.

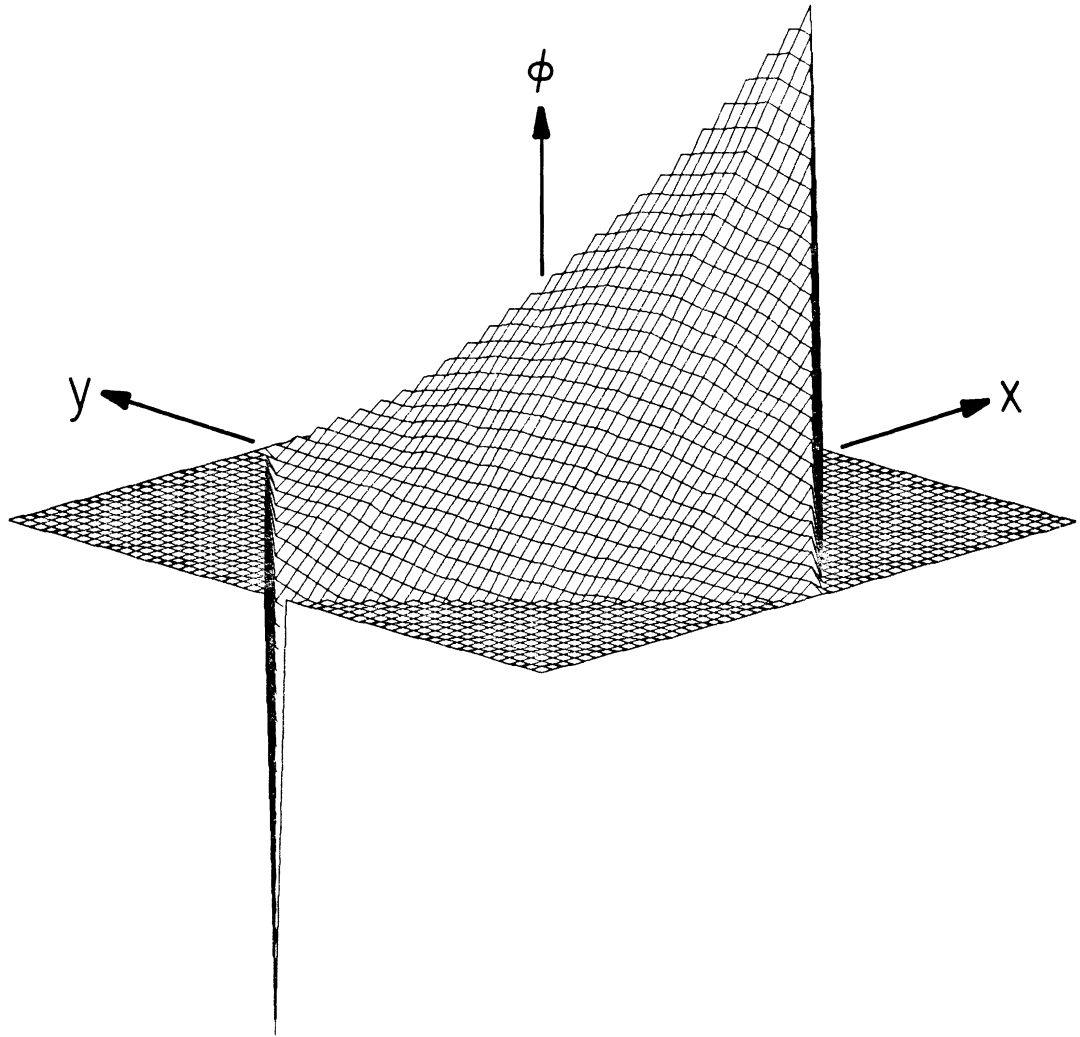


Fig. 3.9: Perspective Plot of Potential on Top Surface of Square Plate Shown Partially Hidden by $\phi = 0$ Reference Square: x Excitation, $\ell/t = 10$, $\tau = 101$.

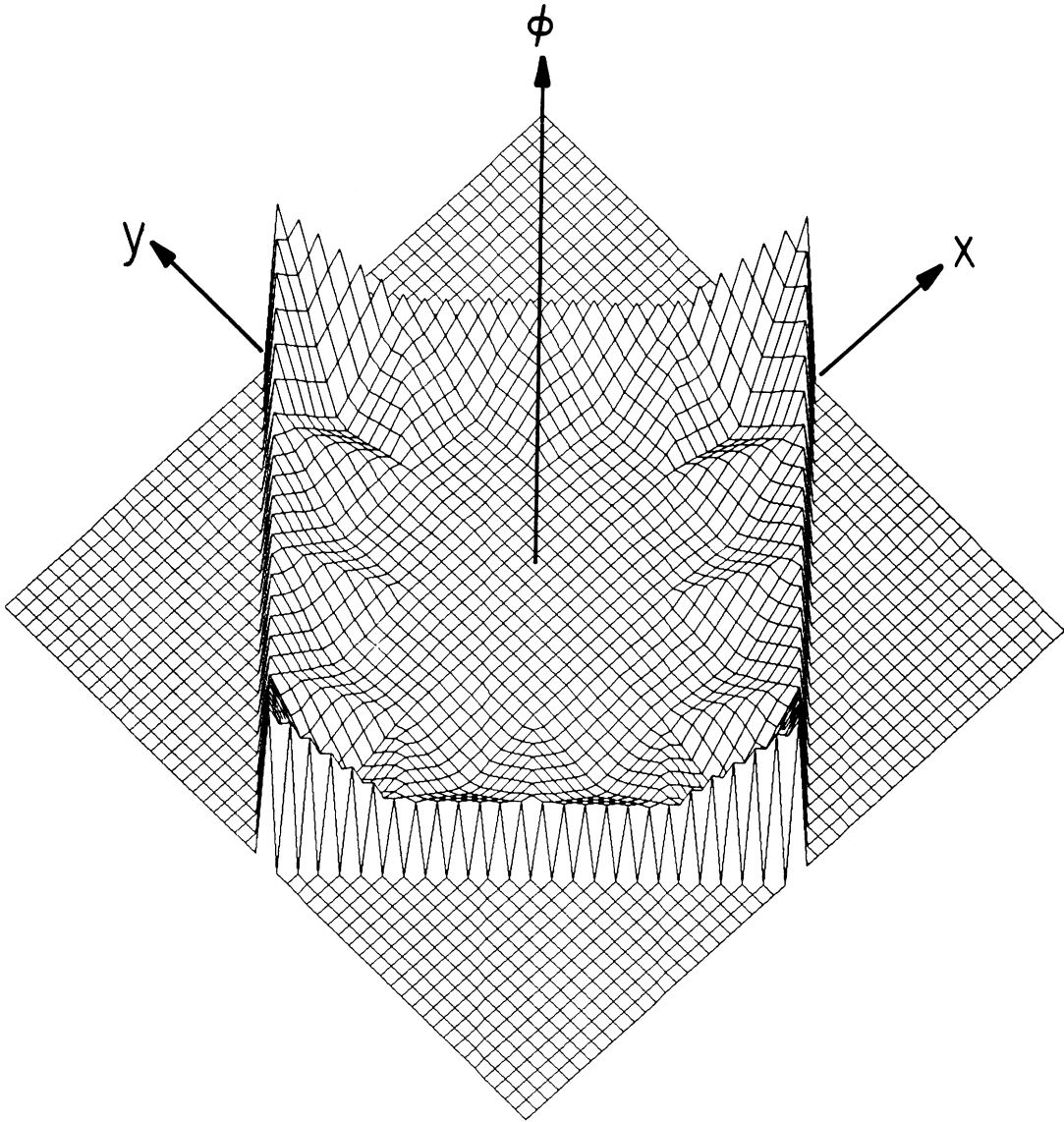


Fig. 3.10: Perspective Plot of Potential on Top Surface of Square Plate Imposed on $\phi = 0$ Reference Square: z Excitation, $\ell/t = 10$, $\tau = 101$.

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

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

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	$(\tau - 1)/\tau \rightarrow \infty$
	-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)

$$\phi_0^t = -x + (1 - \tau) \int_{-t/2}^{t/2} \int_S \nabla'_S \phi_0^t \cdot \nabla'_S G \, ds' \, dz'$$

and the particle is assumed to be a rhombus as shown in Fig. 4.1. If ϕ_0^t is then restricted to be linear, from symmetry consideration ϕ_0^t may be written as

$$\phi_0^t = ax$$

and Eq. (3.4) reduces to

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

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

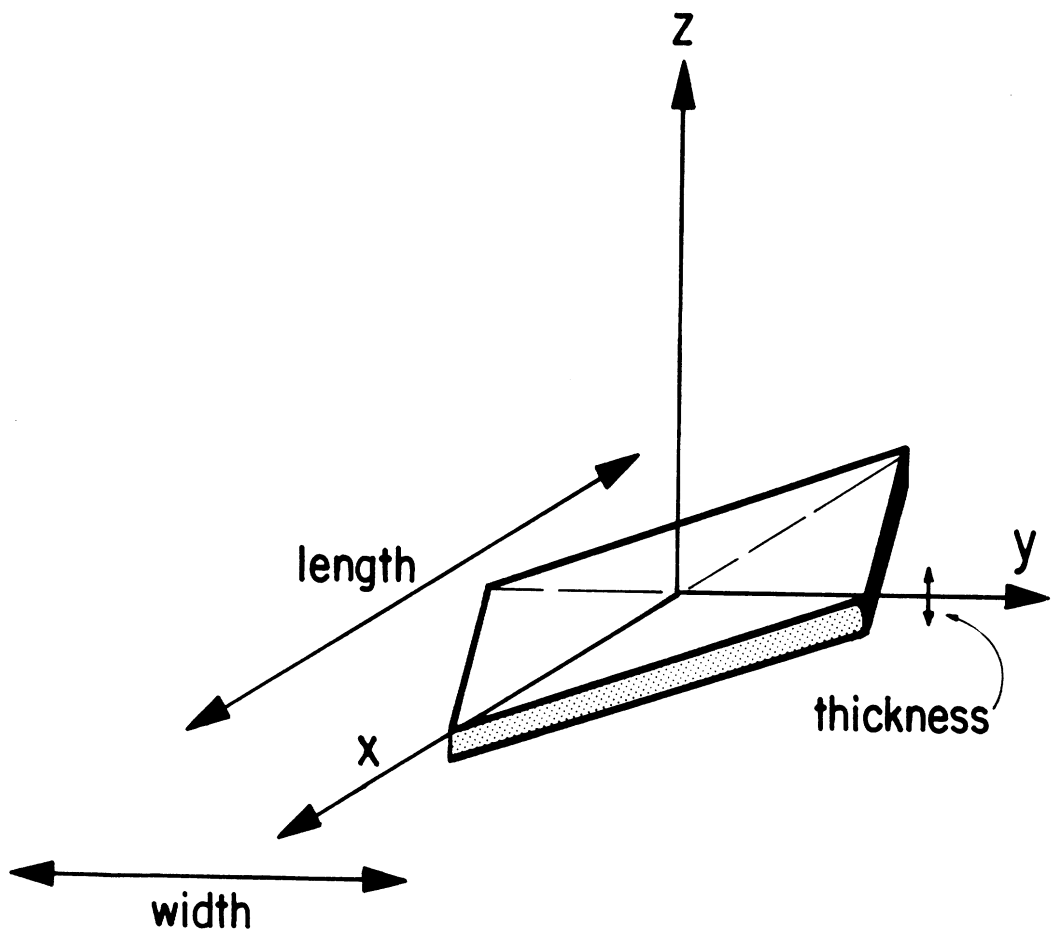


Fig. 4.1: Diagram of a Plate Used by Linear Predictor.

$$a\lambda = -\lambda + (1 - \tau) aI$$

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

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

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

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

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

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

and since

$$\phi_0^t = ax$$

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)

$$\phi_0^t = -z + (1 - \tau) \int_{-t/2}^{t/2} \int_s \left(\nabla_s' \phi_0^t \cdot \nabla_s' G + \frac{\partial \phi_0^t}{\partial z'} \frac{\partial G}{\partial z'} \right) ds' dz' .$$

Again restricting ϕ_0^t to be linear and consistent with the symmetry of the plate the equation yields

$$\phi_0^t = a \frac{z}{t}$$

so Eq. (3.5) becomes

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

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

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

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

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

$$\phi_0^t = a \frac{z}{t}$$

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 $\tau = 2, 11, \text{ and } 101$. 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 $\tau = 2, 11, \text{ and } 101$ (Tables 4.6, 4.7, and 4.8) and $\tau = 2 + i2, 11 + i2, \text{ and } 101 + i2$ (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 $\tau - 1$ for the tangential excitations and $(\tau - 1)/\tau$ 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
		<u>$\tau-1 = 1$</u>	<u>$\tau-1 = 10$</u>	<u>$\tau-1 = 100$</u>
$l/t = 1$				
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

	$\tau =$	2	11	101
		<u>$(\tau-1)/\tau=1/2$</u>	<u>$(\tau-1)/\tau=0.909$</u>	<u>$(\tau-1)/\tau=0.990$</u>
$l/t = 1$				
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
$l/t = 10$				
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
$l/t = 100$				
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

	$\tau =$	2	11	101
		<u>$\tau-1 = 1$</u>	<u>$\tau-1 = 10$</u>	<u>$\tau-1 = 100$</u>
$l/t = 1$				
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

$\tau =$	2	11	101
	<u>$\tau-1 = 1$</u>	<u>$\tau-1 = 10$</u>	<u>$\tau-1 = 100$</u>
$\ell/t = 1$			
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
$\ell/t = 10$			
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
$\ell/t = 100$			
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

	$\tau =$	2	11	101
		$(\tau-1)/\tau=1/2$	$(\tau-1)/\tau=0.909$	$(\tau-1)/\tau=0.990$
$l/t = 1$				
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
 $l/t = 10$				
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
 $l/t = 100$				
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

	$\tau =$	$2+i2$	$11+i2$	$101+i2$
	$\tau - 1 =$	<u>$1+i2$</u>	<u>$10+i2$</u>	<u>$100+i2$</u>
$l/t = 1$				
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

	$\tau =$	$2+i2$	$11+i2$	$101+i2$
	$\tau-1 =$	<u>$1+i2$</u>	<u>$10+i2$</u>	<u>$100+i2$</u>
$\ell/t = 1$				
Linear predictor		1.23+i1.07	2.81+i.152	3.72+i.002
14 pt triangle		1.18+i1.00	2.75+i1.67	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
$\ell/t = 10$				
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
$\ell/t = 100$				
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		
$\tau =$	2+i2	11+i2	101+i2
$(\tau-1)/\tau =$	<u>0.75+i.25</u>	<u>0.912+i.016</u>	<u>0.990+i.0002</u>
$l/t = 1$			
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
$t/\ell = 1.0$	-5.07	-2.12
$t/\ell = 0.1$	-15.02	-0.14
$t/\ell = 0.01$	-86.62	-0.01

length/width = 2

	x excitation	y excitation	z excitation
$t/\ell = 1.0$	-9.45	-2.86	-3.25
$t/\ell = 0.1$	-25.32	-6.80	-0.22
$t/\ell = 0.01$	-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 $(\tau t/l)$ 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)

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

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

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

$$\begin{aligned} X_{11} &= (\tau - 1) \int_B \hat{n} \cdot \bar{x} ds' + (1 - \tau) \int_B \hat{n} \cdot \hat{x} \phi_1^s ds' \\ &= (\tau - 1) \int_{V_B} \nabla \cdot \bar{x} dv' + (1 - \tau) \int_{V_B} \nabla \cdot (\hat{x} \phi_1^s) dv' \quad (\text{cont.}) \end{aligned}$$

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

So, for

$$\left| \frac{\partial \phi_1^S}{\partial x^i} \right| \ll 1$$

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

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

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

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

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

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

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

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

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 constitute 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

associated 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}_ℓ is the local electric field acting on the particle. \bar{E}_ℓ differs from \bar{E} as a result of near-field disturbances of nearby particles. As the distribution of particles becomes sparse, \bar{E}_ℓ 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} ,

$$\bar{P} = \bar{D} - \epsilon_0\bar{E} = \bar{E}(\epsilon - \epsilon_0) = \bar{E}\epsilon_0(\epsilon_r - 1) .$$

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"

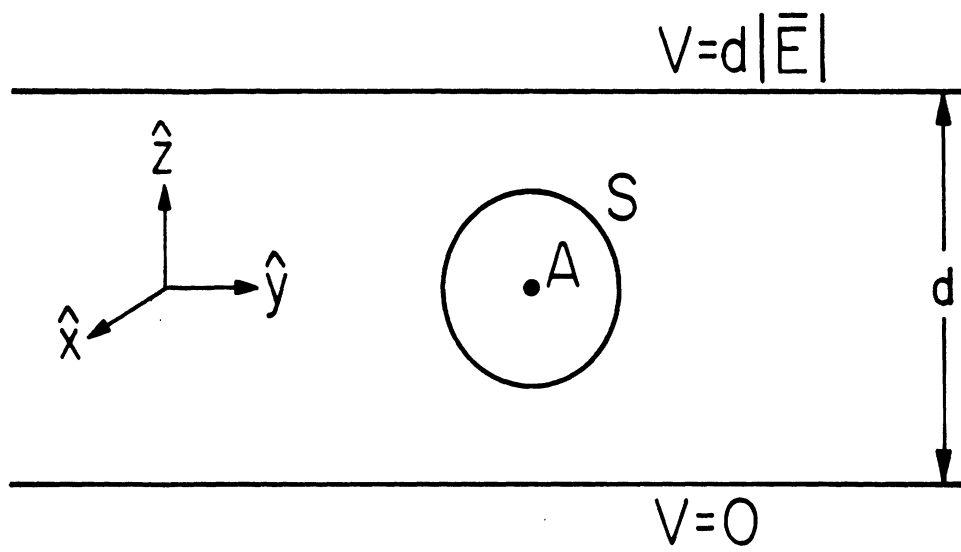


Fig. 5.1: Parallel Plate Capacitor with Spherical Exclusion Volume.

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 .

$$\bar{E}_1 = \frac{\rho}{\epsilon_0} (-\hat{z}) \quad , \quad \rho \equiv \text{charge density on top plate} \quad .$$

\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

$$\bar{E}_2 = \frac{\rho b_1}{\epsilon_0} (-\hat{z}) - \oint_S \frac{\rho b_2}{4\pi\epsilon_0 R^2} \bar{ds} \quad .$$

Then, by linear superposition,

$$\begin{aligned}
 \vec{E}_\ell &= \vec{E}_1 + \vec{E}_2 \\
 &= \frac{\rho}{\epsilon_0} (-\hat{z}) + \frac{\rho b_1}{\epsilon_0} (-\hat{z}) - \oint_S \frac{\rho b_2}{4\pi\epsilon_0 R^2} \vec{ds} \\
 &= \frac{\rho + \rho b_1}{\epsilon_0} (-\hat{z}) + \oint_S \frac{\vec{p} \cdot \hat{n}}{4\pi\epsilon_0 R^2} \vec{ds} .
 \end{aligned}$$

Since $\nabla \cdot \vec{P} = -\rho_{b1}$, and further, since the effects of the polarized particles within the sphere are accounted for with E_1 , the dielectric representation of the polarized particles external to the sphere may be assumed to terminate on the surface of the sphere. Hence, $\vec{P} \equiv 0$ within the sphere.

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

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

and

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

so

$$\vec{E}_\ell = \frac{\rho + \rho b_1}{\epsilon_0} (-\hat{z}) - \oint_S \frac{|\vec{P}| \cos \theta}{4\pi\epsilon_0 R^2} d\vec{s} .$$

But, by the symmetries of the problem, \vec{E}_ℓ has only a z component.

Therefore

$$\begin{aligned}
 \bar{E}_\ell &= \frac{\rho + \rho_{b1}}{\epsilon_0} (-\hat{z}) - \oint_s \frac{|\bar{P}| \cos^2 \theta}{4\pi\epsilon_0 R^2} ds \hat{z} \\
 &= \frac{\rho + \rho_{b1}}{\epsilon_0} (-\hat{z}) - \hat{z} \frac{|\bar{P}|}{4\pi\epsilon_0} \int_0^{2\pi} \int_0^\pi \frac{\cos^2 \theta}{R^2} R^2 \sin \theta d\theta d\phi \\
 &= \frac{\rho + \rho_{b1}}{\epsilon_0} (-\hat{z}) - \hat{z} \frac{|\bar{P}|}{4\pi\epsilon_0} (2\pi) \left[-\frac{1}{3} \cos^2 \theta \right] \Big|_0^\pi \\
 &= \frac{\rho + \rho_{b1}}{\epsilon_0} (-\hat{z}) - \hat{z} \frac{|\bar{P}|}{2\epsilon_0} \frac{2}{3} \\
 &= -\hat{z} \frac{\rho + \rho_{b1}}{\epsilon_0} + \frac{|\bar{P}|}{3\epsilon_0}
 \end{aligned}$$

and

$$\frac{\rho + \rho_{b1}}{\epsilon_0} = |\bar{E}| ,$$

so

$$\begin{aligned}
 \bar{E}_\ell &= \bar{E} + \frac{\bar{P}}{3\epsilon_0} \\
 &= \bar{E} + \frac{\bar{D} - \epsilon_0 \bar{E}}{3\epsilon_0} = \frac{\bar{D} - \epsilon_0 \bar{E} + 3\epsilon_0 \bar{E}}{3\epsilon_0} \\
 &= \frac{\bar{D} + 2\epsilon_0 \bar{E}}{3\epsilon_0} = \bar{E} \frac{\epsilon_0 \epsilon_r + 2\epsilon_0}{3\epsilon_0} \\
 &= \bar{E} \frac{\epsilon_r + 2}{3} .
 \end{aligned}$$

Thus

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

but

$$\bar{P} = \bar{D} - \epsilon_0 \bar{E} = \bar{E} (\epsilon_0 \epsilon_r - \epsilon_0) = \bar{E} \epsilon_0 (\epsilon_r - 1) .$$

So

$$\begin{aligned}\bar{E}\epsilon_0(\epsilon_r - 1) &= N\alpha\bar{E}\frac{\epsilon_r + 2}{3} \\ \epsilon_0(\epsilon_r - 1) &+ N\alpha\frac{\epsilon_r + 2}{3}\end{aligned}$$

and rearranging

$$N\alpha = 3\epsilon_0\frac{\epsilon_r - 1}{\epsilon_r + 2}$$

which is the relation desired.

$$\frac{N\alpha}{3\epsilon_0} = \frac{\epsilon_r - 1}{\epsilon_r + 2}$$

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

Solving for ϵ_r ,

$$\begin{aligned}(\epsilon_r + 2)\frac{N\alpha}{3\epsilon_0} &= \epsilon_r - 1 \\ \epsilon_r\left(\frac{N\alpha}{3\epsilon_0} - 1\right) &= -1 - 2\frac{N\alpha}{3\epsilon_0} \\ \epsilon_r &= \frac{1 + 2\frac{N\alpha}{3\epsilon_0}}{1 - \frac{N\alpha}{3\epsilon_0}} \\ &= 1 + \frac{3\frac{N\alpha}{3\epsilon_0}}{1 - \frac{N\alpha}{3\epsilon_0}}\end{aligned}$$

$$= 1 + \frac{N\alpha}{\epsilon_0 \left(1 - \frac{N\alpha}{3\epsilon_0} \right)}$$

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

$$\vec{E} = \frac{3\hat{n}(\vec{p} \cdot \hat{n}) - \vec{p}}{|\vec{x} - \vec{x}_0|^3}$$

where \vec{p} = the dipole moment,

\vec{x}_0 = the location of the dipole,

\vec{x} = the point of field measurement and

\hat{n} = points from \vec{x}_0 to \vec{x} .

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

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

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

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

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.

$$E_x = \sum_{ijk} \frac{3x_i^2 p_x + 3x_i y_j p_y + 3x_i z_k p_z - p_x |\vec{x}|^2}{|\vec{x}|^5} \stackrel{?}{=} 0$$

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

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

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 A Cubical Exclusion Volume

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

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

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.

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

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

$$\begin{aligned} \bar{E} &= \int_S \frac{\nabla \cdot \bar{P}}{4\pi\epsilon R^2} \hat{R} ds \\ &= \int_{\text{top}} + \frac{\bar{P}_m \cdot \hat{z}}{4\pi\epsilon R^2} \hat{R} ds + \int_{\text{bottom}} - \frac{\bar{P}_m \cdot \hat{z}}{4\pi\epsilon R^2} \hat{R} ds \end{aligned}$$

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

$$= \int_{\text{top}} - \frac{|\bar{P}_m|}{2\pi\epsilon R^2} \hat{R} \, ds \quad .$$

Further, by symmetry

$$\begin{aligned} \bar{E}_\ell &= E_{\ell z} \hat{z} = \hat{z} \int_{\text{top}} - \frac{|P_m|}{2\pi\epsilon R^2} \hat{R} \cdot \hat{z} \, ds = - \frac{|P_m|}{2\pi\epsilon} \hat{z} \int_{\text{top}} \frac{z}{R^3} \, ds \\ &= + \frac{\epsilon_r - 1}{2\pi} \bar{E} \cdot \int_{\text{top}} \frac{z}{R^3} \, ds \quad . \end{aligned}$$

For a sphere,

$$\bar{E}_\ell = \frac{\epsilon_r - 1}{3} \bar{E} \quad .$$

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

$$\begin{aligned} \int_{\text{top}} \frac{z}{R^3} \, ds &= \int_{-\ell}^{\ell} \int_{-\ell}^{\ell} \frac{z}{R^3} \, dx \, dy \Big|_{z=\ell} \\ &= \int_{-\ell}^{\ell} \int_{-\ell}^{\ell} \frac{z}{(x^2 + y^2 + z^2)^{3/2}} \, dx \, dy \Big|_{z=\ell} \quad . \end{aligned}$$

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

$$\begin{aligned}
 &= \int_{-l}^l \frac{z}{y^2 + z^2} \frac{x}{\sqrt{x^2 + y^2 + z^2}} dy \Big|_{z=l} \\
 &= \int_{-l}^l \frac{2l^2}{y^2 + l^2} \frac{1}{\sqrt{y^2 + 2l^2}} dy = \int_0^l \frac{4l^2}{y^2 + l^2} \frac{1}{\sqrt{y^2 + 2l^2}} dy
 \end{aligned}$$

Let

$$u^2 = \frac{y^2 + 2l^2}{l^2} \quad u = \frac{1}{l} \sqrt{y^2 + 2l^2}$$

$$y^2 = u^2 l^2 - 2l^2$$

$$y = l\sqrt{u^2 - 2} \quad dy = l(u^2 - 2)^{-1/2} u du$$

Then

$$\begin{aligned}
 \int_{\text{top}} \frac{z}{R^3} ds &= \int_{\sqrt{2}}^{\sqrt{3}} \frac{4l^2}{u^2 l^2 - l^2} \frac{1}{\sqrt{u^2 l^2}} u l \frac{1}{\sqrt{u^2 - 2}} du \\
 &= \int_{\sqrt{2}}^{\sqrt{3}} \frac{4}{u^2 - 1} \frac{1}{\sqrt{u^2 - 2}} du .
 \end{aligned}$$

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 $v = x - 1$

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

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

$$\begin{aligned} &= 2 \left\{ -\sin^{-1} \left(\frac{-2-2x}{x\sqrt{8}} \right) \Big|_{\sqrt{2}+1}^{\sqrt{3}+1} + \sin^{-1} \left(\frac{-2+2x}{x\sqrt{8}} \right) \Big|_{\sqrt{2}-1}^{\sqrt{3}-1} \right\} \\ &= 2 \left\{ -\sin^{-1} \left(\frac{-2-2(\sqrt{3}+1)}{(\sqrt{3}+1)\sqrt{8}} \right) + \sin^{-1} \left(\frac{-2-2(\sqrt{2}+1)}{(\sqrt{2}+1)\sqrt{8}} \right) \right. \\ &\quad \left. + \sin^{-1} \left(\frac{-2+2(\sqrt{3}-1)}{(\sqrt{3}-1)\sqrt{8}} \right) - \sin^{-1} \left(\frac{-2+2(\sqrt{2}-1)}{(\sqrt{2}-1)\sqrt{8}} \right) \right\} \\ &= 2 \left\{ -\sin^{-1} \left(\frac{-2-\sqrt{3}}{\sqrt{2}+\sqrt{6}} \right) + \sin^{-1} \left(\frac{-2+\sqrt{3}}{-\sqrt{2}+\sqrt{6}} \right) + \sin^{-1}(-1) - \sin^{-1}(-1) \right\} \\ &= 2 \left\{ +\frac{5\pi}{12} + -\frac{\pi}{12} \right\} = 2 \left(\frac{\pi}{3} \right) = \left(\frac{2\pi}{3} \right) . \end{aligned}$$

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

$$\epsilon_r = 1 + \frac{N\alpha}{\epsilon_0 \left(1 - \frac{N\alpha}{3}\right)}$$

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 $\ell \times \ell \times t$. The above formulae were obtained in Chapter IV with a small scattered field approximation which is satisfied as long as $\tau t/\ell \ll 1$. 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

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

and for normal excitation

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

For parallel excitation,

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

$$\begin{aligned} \epsilon_r &= 1 + \frac{3(\tau - 1)t/\ell}{3 - (\tau - 1)t/\ell} \approx 1 + (\tau - 1)t/\ell \\ &= (1 - t/\ell) + \tau(t/\ell) \end{aligned}$$

as expected. For normal excitation

$$\begin{aligned} \epsilon_r &= 1 + \frac{3 \frac{\tau - 1}{\tau} \frac{t}{\ell}}{3 - \frac{\tau - 1}{\tau} \frac{t}{\ell}} \\ &= \frac{3 - \frac{\tau - 1}{\tau} \frac{t}{\ell} + 3 \frac{\tau - 1}{\tau} \frac{t}{\ell}}{3 - \frac{\tau - 1}{\tau} \frac{t}{\ell}} \\ &= \left[\frac{3 - \frac{\tau - 1}{\tau} \frac{t}{\ell}}{3 + 2 \frac{\tau - 1}{\tau} \frac{t}{\ell}} \right]^{-1} \end{aligned}$$

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

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 \vec{F} given $\nabla \times \vec{F} = \vec{f}$. 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

```
1  /* function definitions */
2  #include <math.h>
3  double ppcn(), ptcon(), u(), v(), vr(), intdvz(), intdz(), msqrt(), mlog(),
4  area(), sumang();
5  /* macro definitions */
6  #define Darray(Ara,I1,I2,L1,L2) Ara[(I1)+(I2)*(L1)] /* Macro to generate other\
7  macros to mimic fortran\
8  arrays */
9  #define Mnumpoi 100 /* maximum number of points */
10 #define Mnumtri 100 /* maximum number of triangular patches */
11 #define Mnumatri 10 /* maximum number of triangles which may share a \
12 common point, 6 is average for internal points, 8 accounts for\
13 most schemes, thus 10 is a reasonable upper bound */
14 #define True 1 /* value of logical true */
15 #define False 0 /* value of logical false */
16 #define Tri(J1,J2) Darray(tri,J1,J2,Mnumatri,Mnumpoi) /* set up tri as two\
17 dimensional array with limits (Mnumatri, Mnumpoi) */
18 #define Poi1(J1,J2) Darray(poi1,J1,J2,Mnumatri,Mnumpoi) /* set up poi1 as two\
19 dimensional array with limits (Mnumatri, Mnumpoi) */
20 #define Poi2(J1,J2) Darray(poi2,J1,J2,Mnumatri,Mnumpoi) /* set up poi2 as two\
21 dimensional array with limits (Mnumatri, Mnumpoi) */
22 #define Pol(J1,J2) Darray(pol,J1,J2,Mnumtri,3) /* set up pol as two dimensional\
23 array with limits (Mnumtri,3) */
24 #define Matrix(J1,J2) Darray(matrix,J1,J2,Mnumpoi,Mnumpoi) /* set up matrix as\
25 two dimensional array with limits (Mnumpoi,Mnumpoi) */
26 #define Cmatrix(J1,J2) Darray(cmatrix,J1,J2,Mnumpoi,Mnumpoi) /* complex\
27 version of Matrix, used with the complex\
28 structure */
29 struct complex {
30     double real;
31     double imag;
32 };
33 /*
34 /*
35 /*
36 /*
37 /*
38 /*
39 /*
40 /*
41 /*
42 /*
43 /*
44 /*
45 /*
46 /*
47 /*
48 /*
49 /*
50 /*
51 /*
52 /*
53 /*
54 /*
55 /*
56 /*
57 /*
58 /*
59 /*
60 /*
61 /*
62 /*
63 /*
64 /*
65 /*
66 /*
67 /*
68 /*
69 /*
70 /*
71 /*
72 /*
73 /*
74 /*
75 /*
76 /*
77 /*
78 /*
79 /*
80 /*
81 /*
82 /*
83 /*
84 /*
85 /*
86 /*
87 /*
88 /*
89 /*
90 /*
91 /*
92 /*
93 /*
94 /*
95 /*
96 /*
97 /*
98 /*
99 /*
100 /*
101 /*
102 /*
103 /*
104 /*
105 /*
106 /*
107 /*
108 /*
109 /*
110 /*
111 /*
112 /*
113 /*
114 /*
115 /*
116 /*
117 /*
118 /*
119 /*
120 /*
121 /*
122 /*
123 /*
124 /*
125 /*
126 /*
127 /*
128 /*
129 /*
130 /*
131 /*
132 /*
133 /*
134 /*
135 /*
136 /*
137 /*
138 /*
139 /*
140 /*
141 /*
142 /*
143 /*
144 /*
145 /*
146 /*
147 /*
148 /*
149 /*
150 /*
151 /*
152 /*
153 /*
154 /*
155 /*
156 /*
157 /*
158 /*
159 /*
160 /*
161 /*
162 /*
163 /*
164 /*
165 /*
166 /*
167 /*
168 /*
169 /*
170 /*
171 /*
172 /*
173 /*
174 /*
175 /*
176 /*
177 /*
178 /*
179 /*
180 /*
181 /*
182 /*
183 /*
184 /*
185 /*
186 /*
187 /*
188 /*
189 /*
190 /*
191 /*
192 /*
193 /*
194 /*
195 /*
196 /*
197 /*
198 /*
199 /*
200 /*
201 /*
202 /*
203 /*
204 /*
205 /*
206 /*
207 /*
208 /*
209 /*
210 /*
211 /*
212 /*
213 /*
214 /*
215 /*
216 /*
217 /*
218 /*
219 /*
220 /*
221 /*
222 /*
223 /*
224 /*
225 /*
226 /*
227 /*
228 /*
229 /*
230 /*
231 /*
232 /*
233 /*
234 /*
235 /*
236 /*
237 /*
238 /*
239 /*
240 /*
241 /*
242 /*
243 /*
244 /*
245 /*
246 /*
247 /*
248 /*
249 /*
250 /*
251 /*
252 /*
253 /*
254 /*
255 /*
256 /*
257 /*
258 /*
259 /*
260 /*
261 /*
262 /*
263 /*
264 /*
265 /*
266 /*
267 /*
268 /*
269 /*
270 /*
271 /*
272 /*
273 /*
274 /*
275 /*
276 /*
277 /*
278 /*
279 /*
280 /*
281 /*
282 /*
283 /*
284 /*
285 /*
286 /*
287 /*
288 /*
289 /*
290 /*
291 /*
292 /*
293 /*
294 /*
295 /*
296 /*
297 /*
298 /*
299 /*
300 /*
301 /*
302 /*
303 /*
304 /*
305 /*
306 /*
307 /*
308 /*
309 /*
310 /*
311 /*
312 /*
313 /*
314 /*
315 /*
316 /*
317 /*
318 /*
319 /*
320 /*
321 /*
322 /*
323 /*
324 /*
325 /*
326 /*
327 /*
328 /*
329 /*
330 /*
331 /*
332 /*
333 /*
334 /*
335 /*
336 /*
337 /*
338 /*
339 /*
340 /*
341 /*
342 /*
343 /*
344 /*
345 /*
346 /*
347 /*
348 /*
349 /*
350 /*
351 /*
352 /*
353 /*
354 /*
355 /*
356 /*
357 /*
358 /*
359 /*
360 /*
361 /*
362 /*
363 /*
364 /*
365 /*
366 /*
367 /*
368 /*
369 /*
370 /*
371 /*
372 /*
373 /*
374 /*
375 /*
376 /*
377 /*
378 /*
379 /*
380 /*
381 /*
382 /*
383 /*
384 /*
385 /*
386 /*
387 /*
388 /*
389 /*
390 /*
391 /*
392 /*
393 /*
394 /*
395 /*
396 /*
397 /*
398 /*
399 /*
400 /*
401 /*
402 /*
403 /*
404 /*
405 /*
406 /*
407 /*
408 /*
409 /*
410 /*
411 /*
412 /*
413 /*
414 /*
415 /*
416 /*
417 /*
418 /*
419 /*
420 /*
421 /*
422 /*
423 /*
424 /*
425 /*
426 /*
427 /*
428 /*
429 /*
430 /*
431 /*
432 /*
433 /*
434 /*
435 /*
436 /*
437 /*
438 /*
439 /*
440 /*
441 /*
442 /*
443 /*
444 /*
445 /*
446 /*
447 /*
448 /*
449 /*
450 /*
451 /*
452 /*
453 /*
454 /*
455 /*
456 /*
457 /*
458 /*
459 /*
460 /*
461 /*
462 /*
463 /*
464 /*
465 /*
466 /*
467 /*
468 /*
469 /*
470 /*
471 /*
472 /*
473 /*
474 /*
475 /*
476 /*
477 /*
478 /*
479 /*
480 /*
481 /*
482 /*
483 /*
484 /*
485 /*
486 /*
487 /*
488 /*
489 /*
490 /*
491 /*
492 /*
493 /*
494 /*
495 /*
496 /*
497 /*
498 /*
499 /*
500 /*
501 /*
502 /*
503 /*
504 /*
505 /*
506 /*
507 /*
508 /*
509 /*
510 /*
511 /*
512 /*
513 /*
514 /*
515 /*
516 /*
517 /*
518 /*
519 /*
520 /*
521 /*
522 /*
523 /*
524 /*
525 /*
526 /*
527 /*
528 /*
529 /*
530 /*
531 /*
532 /*
533 /*
534 /*
535 /*
536 /*
537 /*
538 /*
539 /*
540 /*
541 /*
542 /*
543 /*
544 /*
545 /*
546 /*
547 /*
548 /*
549 /*
550 /*
551 /*
552 /*
553 /*
554 /*
555 /*
556 /*
557 /*
558 /*
559 /*
560 /*
561 /*
562 /*
563 /*
564 /*
565 /*
566 /*
567 /*
568 /*
569 /*
570 /*
571 /*
572 /*
573 /*
574 /*
575 /*
576 /*
577 /*
578 /*
579 /*
580 /*
581 /*
582 /*
583 /*
584 /*
585 /*
586 /*
587 /*
588 /*
589 /*
590 /*
591 /*
592 /*
593 /*
594 /*
595 /*
596 /*
597 /*
598 /*
599 /*
600 /*
601 /*
602 /*
603 /*
604 /*
605 /*
606 /*
607 /*
608 /*
609 /*
610 /*
611 /*
612 /*
613 /*
614 /*
615 /*
616 /*
617 /*
618 /*
619 /*
620 /*
621 /*
622 /*
623 /*
624 /*
625 /*
626 /*
627 /*
628 /*
629 /*
630 /*
631 /*
632 /*
633 /*
634 /*
635 /*
636 /*
637 /*
638 /*
639 /*
640 /*
641 /*
642 /*
643 /*
644 /*
645 /*
646 /*
647 /*
648 /*
649 /*
650 /*
651 /*
652 /*
653 /*
654 /*
655 /*
656 /*
657 /*
658 /*
659 /*
660 /*
661 /*
662 /*
663 /*
664 /*
665 /*
666 /*
667 /*
668 /*
669 /*
670 /*
671 /*
672 /*
673 /*
674 /*
675 /*
676 /*
677 /*
678 /*
679 /*
680 /*
681 /*
682 /*
683 /*
684 /*
685 /*
686 /*
687 /*
688 /*
689 /*
690 /*
691 /*
692 /*
693 /*
694 /*
695 /*
696 /*
697 /*
698 /*
699 /*
700 /*
701 /*
702 /*
703 /*
704 /*
705 /*
706 /*
707 /*
708 /*
709 /*
710 /*
711 /*
712 /*
713 /*
714 /*
715 /*
716 /*
717 /*
718 /*
719 /*
720 /*
721 /*
722 /*
723 /*
724 /*
725 /*
726 /*
727 /*
728 /*
729 /*
730 /*
731 /*
732 /*
733 /*
734 /*
735 /*
736 /*
737 /*
738 /*
739 /*
740 /*
741 /*
742 /*
743 /*
744 /*
745 /*
746 /*
747 /*
748 /*
749 /*
750 /*
751 /*
752 /*
753 /*
754 /*
755 /*
756 /*
757 /*
758 /*
759 /*
760 /*
761 /*
762 /*
763 /*
764 /*
765 /*
766 /*
767 /*
768 /*
769 /*
770 /*
771 /*
772 /*
773 /*
774 /*
775 /*
776 /*
777 /*
778 /*
779 /*
780 /*
781 /*
782 /*
783 /*
784 /*
785 /*
786 /*
787 /*
788 /*
789 /*
790 /*
791 /*
792 /*
793 /*
794 /*
795 /*
796 /*
797 /*
798 /*
799 /*
800 /*
801 /*
802 /*
803 /*
804 /*
805 /*
806 /*
807 /*
808 /*
809 /*
810 /*
811 /*
812 /*
813 /*
814 /*
815 /*
816 /*
817 /*
818 /*
819 /*
820 /*
821 /*
822 /*
823 /*
824 /*
825 /*
826 /*
827 /*
828 /*
829 /*
830 /*
831 /*
832 /*
833 /*
834 /*
835 /*
836 /*
837 /*
838 /*
839 /*
840 /*
841 /*
842 /*
843 /*
844 /*
845 /*
846 /*
847 /*
848 /*
849 /*
850 /*
851 /*
852 /*
853 /*
854 /*
855 /*
856 /*
857 /*
858 /*
859 /*
860 /*
861 /*
862 /*
863 /*
864 /*
865 /*
866 /*
867 /*
868 /*
869 /*
870 /*
871 /*
872 /*
873 /*
874 /*
875 /*
876 /*
877 /*
878 /*
879 /*
880 /*
881 /*
882 /*
883 /*
884 /*
885 /*
886 /*
887 /*
888 /*
889 /*
890 /*
891 /*
892 /*
893 /*
894 /*
895 /*
896 /*
897 /*
898 /*
899 /*
900 /*
901 /*
902 /*
903 /*
904 /*
905 /*
906 /*
907 /*
908 /*
909 /*
910 /*
911 /*
912 /*
913 /*
914 /*
915 /*
916 /*
917 /*
918 /*
919 /*
920 /*
921 /*
922 /*
923 /*
924 /*
925 /*
926 /*
927 /*
928 /*
929 /*
930 /*
931 /*
932 /*
933 /*
934 /*
935 /*
936 /*
937 /*
938 /*
939 /*
940 /*
941 /*
942 /*
943 /*
944 /*
945 /*
946 /*
947 /*
948 /*
949 /*
950 /*
951 /*
952 /*
953 /*
954 /*
955 /*
956 /*
957 /*
958 /*
959 /*
960 /*
961 /*
962 /*
963 /*
964 /*
965 /*
966 /*
967 /*
968 /*
969 /*
970 /*
971 /*
972 /*
973 /*
974 /*
975 /*
976 /*
977 /*
978 /*
979 /*
980 /*
981 /*
982 /*
983 /*
984 /*
985 /*
986 /*
987 /*
988 /*
989 /*
990 /*
991 /*
992 /*
993 /*
994 /*
995 /*
996 /*
997 /*
998 /*
999 /*
1000 /*
```

```

37 /*
38 int flagsym; /* flagsym may assume values of 0,1,2, or 3. Flagsym is used
39 by low level routines to incorporate symmetry in a manner
40 invisible to the rest of the program.
41 0 - indicates no symmetry
42 1 - indicates object possesses mirror symmetry about x=0
43 2 - indicates object possesses mirror symmetry about y=0
44 3 - indicates object possesses mirror symmetry about x=0 and y=0
45 Flagsym is set at the beginning of the main program and after
46 that is never changed */
47 int flagsyms; /* flagsyms is used in conjunction with flagsym to generate
48 mirror images of the source patches while calculating
49 contributions to the field point. The initial object
50 description is assumed to lie in the first quadrant, however
51 if flagsym is 0, this is not necessary. Flagsyms is always
52 less than or equal to flagsym, and may assume the values of
53 0,1,2, or 3. These values are given the following meanings:
54 0 - source patch is original patch (presumably first quadrant)
55 1 - source patch is in second quadrant
56 2 - source patch is in fourth quadrant
57 3 - source patch is in third quadrant */
58 int plotsym; /* plotsym is used to denote what type of mirroring is desired
59 in the perspective plot.
60 0 - no mirroring
61 1 - mirror about x=0
62 2 - mirror about y=0
63 3 - mirror about x=0 and y=0
64 Any mirroring that is done must be consistent with the
65 mirroring specified in generating the plate. plotsym
66 is specified in rgrph. */
67 int eof; /* end of file variable (=0 means end of file) */
68 int potflag; /* flag marks whether or not potentials have been computed
69 0 - potential has not been computed
70 1 - potential has been computed assuming x excitation
71 2 - potential has been computed assuming y excitation
72 3 - potential has been computed assuming z excitation */
73 struct {
74 double x[Mnumpoi];
75 double y[Mnumpoi];
76 double poten[Mnumpoi];
77 struct complex cpoten[Mnumpoi]; /* complex version of poten */
78 }point; /* point is a structure which contains the locations of all
79 of the points used in defining the triangular patches. A point
80 number 0 is permitted as C defines arrays beginning with 0.

```

```

81 X and y are the coordinates of the points, and
82 poten is the potential of each point, as determined from the
83 matrix problem. */
84 struct {
85     int numatri[Mnumpoi];
86     int Tri(Mnumatri,Mnumpoi-1);
87     int Poi1(Mnumatri,Mnumpoi-1);
88     int Poi2(Mnumatri,Mnumpoi-1);
89     } apoint; /* apoint is a structure which contains lists of triangles
90                associated with each point. numatri contains the number of
91                triangles associated with each point; tri contains the list of
92                triangle numbers associated with each point; and poi1 and poi2
93                contain the other two vertices used in defining the triangle.
94                poi1 and poi2 are indices to point. tri is an index to
95                triang. */
96 struct {
97     struct {
98         double x[Mnumtri];
99         double y[Mnumtri];
100     } centroid;
101     int Poi(Mnumtri,3-1);
102     double poten[Mnumtri];
103     struct complex cpoten[Mnumtri]; /* complex version of poten */
104     } triang; /* triang is used to solve the scattering problem with z
105                excitation. centroid .x and .y contain the coordinates
106                of the centroid of each triangle. poi contains the list of
107                points (vertices) associated with each triangle, and is an
108                index to point. poten is the potential of each triangle as
109                determined from solution of the matrix problem. */
110     double Matrix(Mnumpoi,Mnumpoi-1); /* This is "THE" matrix. Trivial points
111                (ie, those which have zero potential from
112                symmetry considerations) are skipped when
113                reading or filling the matrix, which is
114                always done sequentially. */
115     struct complex Cmatrix(Mnumpoi,Mnumpoi-1); /* complex version of Matrix */
116     double fvect[Mnumpoi]; /* fvect is the forcing vector for the matrix problem
117                and after solution of the matrix problem contains
118                the solution vector. */
119     struct complex cfvect[Mnumpoi]; /* complex version of fvect */
120     int mnumpoi; /* This is the total number of points. Points are assumed to be
121                numbered sequentially from 0. mnumpoi <= Mnumpoi. */
122     int mnumtri; /* This is the total number of triangles. Triangles are assumed
123                to be numbered sequentially from 0. mnumtri <= Mnumtri. */
124     union {

```



```

125 char str[2];
126 char let;
127 } com;
128 /* This is the first character of each input line, and is
129 interpreted as a command. Valid commands are:
130 d - display linkup of points and triangles
131 and display potentials if defined
132 e - specify direction of exciting field,
133 and solve the resultant matrix problem.
134 g - graph potentials of points/triangles
135 h - enter heading, ie, a descriptive one line title.
136 m - enter material parameters of plate: t and tau.
137 p - enter coordinates of next point
138 r - regenerate matrix problem using finer mesh
139 s - define symmetry to be assumed in solving problem
140 t - enter definition of next triangle */
141 char hedstr[82]; /* contains one line heading, description of data */
142 double t; /* Thickness of the plate */
143 double tau; /* permittivity of the plate */
144 double tau1; /* imaginary part of the permittivity. This variable is also
145 used as a flag: if tau1 is zero, computations are performed
146 assuming "tau" is purely real; if tau1 is non-zero, the routines
147 appropriate for a complex tau are invoked */
148 double dipmom; /* contains the real part of the computed dipole moment */
149 double dipmomi; /* contains the imaginary part of the computed dipole moment */
150 double Epsilon = 1e-10; /* A very small number, used for approximate equality */
151 double Pi = 3.1415926535; /* Pi */
152 /*****
153 /* M A I N P R O G R A M
154 /*
155 /*
156 /* This is a program which calculates scattering by an arbitrarily shaped,
157 /* thin, flat, dielectric plate. Excitation may be specified in the x, y,
158 /* or z directions. The plate is made up of an arbitrary number (nominally
159 /* less than 50) of triangular patches, upon which a method of moments
160 /* solution is obtained. Contributions from each patch is calculated via
161 /* surface integrals which are evaluated analytically, thus contributions
162 /* from each patch is obtained exactly (at least to 10 plus digits). The
163 /* only approximations which are made are that the potential varies linearly
164 /* with z inside the plate, and the division of the plate into triangular
165 /* patches, inside of which the field varies linearly with x and y. The
166 /* shape and size of the triangular patches are completely arbitrary, and it
167 /* is noted that the patches need not be contiguous. To speed computations,
168 /* and to enhance that accuracy obtainable with a given number of patches, */

```

```
169 /* the user may specify that the plate possesses mirror symmetry about x=0. */
170 /* y=0, or both x=0 and y=0. */
171 /*
172 /*****
173 main() {
174   hedstr[0] = '\0';
175   for(eof=scanf("%is",com.str);eof == 1;) {
176     switch (com.let) {
177       case 'p':
178       case 'p':
179       case 't':
180       case 't':
181         rdata(); /* read in the data */
182         continue;
183       case 'D':
184       case 'd':
185         ddata(); /* display the data */
186         break;
187       case 'G':
188       case 'g':
189         rgrph(); /* read in mirroring specification */
190         break;
191       case 'H':
192       case 'h':
193         gethed();
194         break;
195       case 'M':
196       case 'm':
197         rmp(); /* read in material parameters */
198         break;
199       case 'S':
200       case 's':
201         rsym(); /* read in the symmetry of the plate */
202         break;
203       case 'E':
204         eigen(); /* solve eigenvalue problem */
205         break;
206       case 'e':
207         rexcit(); /* read in direction of excitation
208                    and solve the resulting matrix problem */
209         switch (potflag) {
210           case 1:
211             xsolv();
212             pexc();

```

```
213 break;
214
215 case 2:
216     ysolv();
217     pexc();
218     break;
219
220 case 3:
221     zsolv();
222     pexc();
223     break;
224
225 default:
226     printf("invalid excitation");
227     break;
228
229 }
230
231 default:
232     break;
233
234 printf("%c is an invalid command\n", com.let);
235 while(scanf("%c", com.str), com.let != '\n');
236 /* flush out bad command */
237 break;
238
239 eof=scanf("%1s", com.str);
240 }
241
242 /*****
243 /* ROUTINE: ddata
244 /*
245 /* ddata is called by the main program to display data. ddata display
246 /* locations of points, connections of points, and potentials of points if
247 /* they have been computed. ddata also displays status of various flags
248 /* which indicate symmetries of plate and direction of excitation.
249 /*
250 ddata() {
251     int i,j; /* loop variables */
252     if (t != 0) printf("t = %lf, tau = %lf + i %lf \n", t, tau, tau);
253     switch (potflag) {
254     case 0:
255         printf("potential has not been computed\n");
256         break;
257     case 1:
258         printf("potential has been computed assuming x excitation\n");
259         break;
260     case 2:
261         break;
262     }
```

```
257 printf("potential has been computed assuming y excitation\n");
258 break;
259
260 case 3: printf("potential has been computed assuming z excitation\n");
261 break;
262
263 switch (flagsym) {
264 case 0: printf("plate is not symmetric\n");
265 break;
266
267 case 1: printf("plate has mirror symmetry about x=0\n");
268 break;
269
270 case 2: printf("plate has mirror symmetry about y=0\n");
271 break;
272
273 case 3: printf("plate has mirror symmetry about x=0 and y=0\n");
274 break;
275
276 }
277 for (i=0; i<numpoi; i++) {
278 printf("point # %d is located at x = %lf, y = %lf\n", i, point.x[i],
279 point.y[i]);
280 if (potflag) {
281 if (itau) printf(" and has potential %e\n", point.poten[i]);
282 else printf(" and has potential %lf + i %lf\n",
283 point.cpoten[i].real, point.cpoten[i].imag);
284 }
285 printf("point %d is associated with points and triangles (tri,P1,P2)\n",
286 i);
287 for (j=0; j<apoint.numatri[i]; j++) {
288 printf(" (%d,%d,%d), ", apoint.Tri(j,i), apoint.Poi1(j,i),
289 apoint.Poi2(j,i));
290 if (j%6 == 5 || j == apoint.numatri[i]-1) printf("\n");
291 }
292 }
293
294 /*****
295 /*
296 /* ROUTINE: rsym
297 /*
298 /* rsym is called by the main program to read in the symmetry of the plate.
299 /* rsym assumes no symmetry if response is inappropriate.
300 /*
```

```
301 /*****  
302 rsym() {  
303 scanf("%d", &flagsym);  
304 switch(flagsym) {  
305     default: flagsym=0;  
306     case 1:  
307     case 2:  
308     case 3:  
309     ;  
310 }  
311 }  
312 /*****  
313 /* ROUTINE: rgrph  
314 /*  
315 /* rgrph is called by the main program to read in the mirroring desired.  
316 /* rgrph then calls graph to produce the graph. plotsym is enforced to  
317 /* be consistent with flagsym.  
318 /*  
319 /*****  
320 rgrph() {  
321 scanf("%d", &plotsym);  
322 switch(plotsym) {  
323     default: plotsym=0;  
324     case 1:  
325     case 2:  
326     case 3:  
327     ;  
328 }  
329 plotsym = plotsym & flagsym;  
330 graph();  
331 }  
332 /*****  
333 /* ROUTINE: rmp  
334 /*  
335 /* rmp is called by the main program to read in the material parameters of  
336 /* the plate, specifically thickness and permittivity.  
337 /*  
338 /*****  
339 rmp() {  
340 scanf("%f %f %f", &t, &tau, &taui);  
341 dipmomi = 0;  
342 }  
343 }  
344 }
```

```
345 /******  
346 /* ROUTINE:  rexcit  
347 /*  
348 /* rexcit is called by the main program to read in the direction of  
349 /* excitation.  rexcit assumes x excitation in the event of an invalid  
350 /* response.  
351 /*  
352 /******  
353 rexcit() {  
354 char charsr[2]; /* scratch variable, contains direction of excitation */  
355 scanf("%is",charsr);  
356 switch (*charsr) {  
357 default:  
358     potflag = 0;  
359     break;  
360 case 'X':  
361     case 'x':  
362     potflag = 1;  
363     break;  
364 case 'Y':  
365     case 'y':  
366     potflag = 2;  
367     break;  
368 case 'Z':  
369     case 'z':  
370     potflag = 3;  
371     break;  
372 }  
373 }  
374 }  
375 /******  
376 /* ROUTINE:  xsolv  
377 /*  
378 /* xsolv is called by main to fill THE matrix.  xsolv assumes excitation is  
379 /* in the x direction.  Points which lie on x=0 are assumed to have zero  
380 /* potential and are skipped in both the row and column loops.  The outside  
381 /* loop is the column loop and is associated with the field point.  The  
382 /* inside loop is associated with the source point.  Images of the source  
383 /* point are not apparent at this level and are handled in ppcn.  
384 /* After the matrix and forcing vector are filled, solvem is called to  
385 /* solve the matrix problem.  The solution vector is then read in the same manner  
386 /* as it was generated, ie, trivial points are zeroed and skipped in reading  
387 /*  
388 /*
```

```

389 /* the solution vector.
390 /*
391 /******
392 xsolv() {
393   int flagtriv, flagtrif; /* flagtriv and flagtrif are used by the source loop
394                          and the field loop respectively to flag trivial
395                          points. Values are either True (defined above as
396                          1) or False: 0. */
397   int ipois, ipoif; /* ipois and ipoif denote the source and field points
398                    respectively. This is in contradistinction to irow
399                    and icol. */
400   int irow, icol; /* icol and irow denote positions within THE matrix.
401                  If x symmetry is not being used, these variables
402                  will contain the same values as ipois and ipoif.
403                  If x symmetry is being used, then icol and irow will
404                  gradually fall behind ipois and ipoif as trivial
405                  points are encountered. */
406   for ( flagtrif = False, ipoif = 0, irow = 0; ipoif < mnumpoi; ipoif++, irow +=
407         !flagtrif, flagtrif = False ) {
408     if (flagsym & 1 && !point.x[ipoif]) {flagtrif = True; continue;}
409     for ( flagtriv=False,ipois=0,icol=0;ipois < mnumpoi; ipois++, icol +=
410           !flagtriv, flagtriv=False) {
411       if (flagsym & 1 && !point.x[ipois]) {flagtriv=True; continue;}
412       if (!taui) Matrix(irow,icol)=(tau-1)*ppcon(ipoif,ipois);
413     }
414     else {
415       Cmatrix(irow,icol).real=(tau-1)*ppcon(ipoif,ipois);
416       Cmatrix(irow,icol).imag= tau*ppcon(ipoif,ipois);
417     }
418     if (irow == icol) {
419       if (!taui) Matrix(irow,icol) += 1;
420       else Cmatrix(irow,icol).real += 1;
421     }
422   }
423   if (!taui) fvect[irow]=point.x[ipoif];
424   else {
425     cfvect[irow].real=point.x[ipoif];
426     cfvect[irow].imag=0;
427   }
428 }
429 if (!taui) {
430   pmat();
431   pvec();
432 }
433 solvem(irow);

```

```

433 if (itau) {
434   pmat();
435   pvec();
436 }
437 for ( flagtrif=False, ipoif=0, irow=0; ipoif < mnumpoint; ipoif++, irow +=
438   !flagtrif, flagtrif = False ) {
439   if (flagsym & 1 && !point.x[ipoif]) flagtrif = True;
440   if (itau) point.poten[ipoif] = (flagtrif) ? 0. : fvect[irow];
441   else {
442     point.cpoten[ipoif].real = (flagtrif)?0.:cfvect[irow].real;
443     point.cpoten[ipoif].imag = (flagtrif)?0.:cfvect[irow].imag;
444   }
445 }
446 }
447 /*****
448 */
449 /* ROUTINE:   ysolv
450 */
451 /* ysolv is called by main to fill THE matrix. ysolv assumes excitation is
452 /* in the y direction. Points which lie on y=0 are assumed to have zero
453 /* potential and are skipped in both the row and column loops. The outside
454 /* loop is the column loop and is associated with the field point. The
455 /* inside loop is associated with the source point. Images of the source
456 /* point are not apparent at this level and are handled in ppcon.
457 /* After the matrix and forcing vector are filled, solvem is called to
458 /* the matrix problem. The solution vector is then read in the same manner
459 /* it was generated, ie, trivial points are zeroed and skipped in reading
460 /* the solution vector.
461 */
462 /*****
463 */
464 int flagtriv, flagtrif; /* flagtriv and flagtrif are used by the source loop
465 /* and the field loop respectively to flag trivial
466 /* points. Values are either True
467 /* 1) or False: 0
468 int ipoils, ipoif; /* ipoils and ipoif denote the source and field points
469 /* respectively. This is in contradistinction to irow
470 /* and icol.
471 int irow, icol; /* icol and irow denote positions within THE matrix.
472 /* If y symmetry is not being used, these variables
473 /* will contain the same values as ipoils and ipoif.
474 /* If y symmetry is being used, then icol and irow will
475 /* gradually fall behind ipoils and ipoif as trivial
476 /* points are encountered.

```



```

477 for ( flagtrif=False, ipoif=0, irow=0; ipoif < mnumpoi; ipoif++, irow +=
478 !flagtrif, flagtrif = False ) {
479   if ( flagsym & 2 && !point.y[ipoif] ) {flagtrif = True; continue;}
480   for ( flagtriv=False, ipois=0, icol=0; ipois < mnumpoi; ipois++, icol +=
481 !flagtriv, flagtriv=False ) {
482   if ( flagsym & 2 && !point.y[ipois] ) {flagtriv=True; continue;}
483   if ( !taui ) Matrix(irow,icol)=(tau-i)*ppcon(ipoif,ipois);
484   else {
485     Cmatrix(irow,icol).real=(tau-i)*ppcon(ipoif,ipois);
486     Cmatrix(irow,icol).imag=tau*ppcon(ipoif,ipois);
487   }
488   if (irow == icol) {
489     if ( !taui ) Matrix(irow,icol) += 1;
490     else Cmatrix(irow,icol).real += 1;
491   }
492 }
493 if ( !taui ) fvect[irow]=point.y[ipoif];
494 else {
495   cfvect[irow].real=point.y[ipoif];
496   cfvect[irow].imag=0;
497 }
498 }
499 solvem(irow);
500 for ( flagtrif=False, ipoif=0, irow=0; ipoif < mnumpoi; ipoif++, irow +=
501 !flagtrif, flagtrif = False ) {
502   if ( flagsym & 2 && !point.y[ipoif] ) flagtrif = True;
503   if ( !taui ) point.poten[ipoif] = (flagtrif) ? 0. : fvect[irow];
504   else {
505     point.cpoten[ipoif].real=(flagtrif)?0.:cfvect[irow].real;
506     point.cpoten[ipoif].imag=(flagtrif)?0.:cfvect[irow].imag;
507   }
508 }
509 }
510 /*****
511 /*
512 /* ROUTINE: ppcon(ipoif,ipois)
513 /*
514 /* ppcon is called by xsolv and ysolv to calculate the point to point
515 /* contribution. ppcon processes all triangles associated with the source
516 /* point. Imaging of the source patches due to symmetry is not explicitly
517 /* performed by pppcon but is handled by lower level routines. All
518 /* processing is done by calling ptcon
519 /*
520 /*****

```

```

521 double ppcn(ipof,ipois)
522 int ipof, ipois; /* field point index, source point index */
523 {
524 int i; /* loop variable */
525 double sum; /* scratch variable, holds sum of contributions */
526 for (i=0,sum=0; i<apoint.nmatr[ipois]; i++)
527   sum += ptcon(ipof, ipois, apoint.Poi1(i,ipois),
528               apoint.Poi2(i,ipois));
529 return(sum/(4*Pi));
530 }
531 /*****
532 /* ROUTINE: ptcon(ipof,ipois,ipoi1,ipoi2)
533 /*
534 /* ptcon calculates the contribution from a source triangle to a field
535 /* point. The source triangle is delimited by the vertices associated with
536 /* ipois, ipoi1, and ipoi2. Depending upon the shape of the triangle, the
537 /* field contribution is calculated using one of three formulas.
538 /* The coordinates of all points are transformed to the u-v coordinate
539 /* system using the functions u, v, ur, and vr. ur and vr are used to
540 /* calculate coordinates of patches in reflected quadrants. traninit is
541 /* used to initialize the translation.
542 /*
543 /*
544 /*****
545 double ptcon(ipof,ipois,ipoi1,ipoi2)
546 int ipof, ipois, ipoi1, ipoi2; /* indices for: field point, source point, and
547 /* two delimiting points of source triangle */
548 {
549 int i; /* scratch variable */
550 double bsum; /* stores sum of contribution from first quadrant and all reflected
551 /* quadrants */
552 double sum; /* stores line integral contributions to surface integral */
553 for (flagsyms=0,bsum=0; flagsyms<=flagsym; flagsyms += (flagsym&1)?1:2) {
554   traninit(ipoi1,ipoi2);
555   if (ur(ipois) < 0) {
556     i=ipoi1;
557     ipoi1=ipoi2;
558     ipoi2=i;
559     traninit(ipoi1,ipoi2);
560   }
561   sum = (fabs(ur(ipois)) * Epsilon > fabs(vr(ipois)))?0.:
562         intdvz(ur(ipois)/vr(ipois),0,0, vr(ipois),ipof);
563   sum += (fabs(ur(ipois)) * Epsilon > fabs(vr(ipoi2)-vr(ipois)))?0.:
564         intdvz(-ur(ipois)/(vr(ipoi2)-vr(ipois)),ur(ipois)*vr(ipoi2)

```

```

565         /((vr(ipoi2)-vr(ipois)),vr(ipois),vr(ipoi2),ipoi2);
566     sum -= intdvz(0.,0.,0.,vr(ipoi2),ipoi2);
567     sum /= fabs(vr(ipois));
568     bsum += (potflag != 3 && potflag & flagsyms) ? -sum : sum;
569 }
570 return(bsum);
571 }
572 /*****
573 /* Additional External Variables
574 /*
575 /* Used in routines: traninit,u,v,ur,vr
576 /*
577 /*
578 /*****
579 double x0,my0,calp,salp; /* variables used in mapping to u-v coordinate system.
580                          x0, and my0 are the coordinates of the origin, and
581                          calp and salp are the cosine and sine of the angle
582                          alpha used to rotate the x-y coordinate system.
583                          /* my0 is used instead of y0 so as to not be
584                          confused with the system supplied besel
585                          function which is accessed by y0. */
586 /*****
587 /*
588 /* ROUTINE: traninit(ipoi1,ipoi2)
589 /*
590 /* traninit initializes the translation routines u,v,ur, and vr. Initial
591 /* calculations are made to obtain the appropriate displacements and
592 /* rotations which will be needed in the translation routines.
593 /*
594 /*
595 /*****
596 traninit(ipoi1,ipoi2)
597 int ipoi1,ipoi2; /* indices to two auxiliary points used in defining source
598                  source triangle. u-v coordinate system will be centered
599                  on the point associated with ipoi1, with the point
600                  associated with ipoi2 lying on line u=0.
601 {
602 double alp; /* this is the angle of rotation */
603 switch (flagsyms) {
604     case 0: x0=point.x[ipoi1];
605             my0=point.y[ipoi1];
606             alp=atan2(point.y[ipoi2]-my0,point.x[ipoi2]-x0);
607             calp=cos(alp);
608             salp=sin(alp);

```

```
609 break;
610 case 1: x0= -point.x[ipoi1];
611         my0=point.y[ipoi1];
612         alp=atan2(point.y[ipoi2]-my0,-point.x[ipoi2]-x0);
613         calp=cos(alp);
614         salp=sin(alp);
615         break;
616 case 2: x0=point.x[ipoi1];
617         my0= -point.y[ipoi1];
618         alp=atan2(-point.y[ipoi2]-my0,point.x[ipoi2]-x0);
619         calp=cos(alp);
620         salp=sin(alp);
621         break;
622 case 3: x0= -point.x[ipoi1];
623         my0= -point.y[ipoi1];
624         alp=atan2(-point.y[ipoi2]-my0,-point.x[ipoi2]-x0);
625         calp=cos(alp);
626         salp=sin(alp);
627         break;
628 }
629 return;
630 }
631 /*****
632 /* ROUTINES: u(i),v(i),ur(i),vr(i)
633 /*
634 /*
635 /* These routines are used to calculate coordinates of points in the u-v
636 /* coordinate system. i is an index to point. u and v return the
637 /* coordinates of the point, and ur and vr return the coordinates of an
638 /* image of the point, the location of the image depending on the value of
639 /* flagsyms.
640 /*
641 /*****
642 double u(i)
643 int i; /* index to point */
644 {
645     return((point.x[i]-x0)*(-salp)+(point.y[i]-my0)*calp);
646 }
647 double v(i)
648 int i; /* index to point */
649 {
650     return((point.x[i]-x0)*calp+(point.y[i]-my0)*salp);
651 }
652 double ur(i)
```

```

653 int i; /* index to point */
654 {
655     switch (flagsyms) {
656         case 0: return((point.x[i]-x0)*(-salp)+(point.y[i]-my0)*calp);
657         case 1: return((-point.x[i]-x0)*(-salp)+(point.y[i]-my0)*calp);
658         case 2: return((point.x[i]-x0)*(-salp)+(-point.y[i]-my0)*calp);
659         case 3: return((-point.x[i]-x0)*(-salp)+(-point.y[i]-my0)*calp);
660     }
661 }
662 double vr(i)
663 int i; /* index to point */
664 {
665     switch (flagsyms) {
666         case 0: return((point.x[i]-x0)*calp+(point.y[i]-my0)*salp);
667         case 1: return((-point.x[i]-x0)*calp+(point.y[i]-my0)*salp);
668         case 2: return((point.x[i]-x0)*calp+(-point.y[i]-my0)*salp);
669         case 3: return((-point.x[i]-x0)*calp+(-point.y[i]-my0)*salp);
670     }
671 }
672 /*****
673 /* ROUTINE: rdata
674 /*
675 /* rdata is called by the main program to read in point and triangle
676 /* definitions. Upon encountering a non- "p" or "t" command, rdata calls
677 /* ldata to link the data, and then returns to the main program.
678 /*
679 /*
680 /*****
681 rdata() {
682 int i; /* loop variable */
683 int inumb; /* number of next point or triangle */
684 do {
685     switch (com.let) {
686         case 'p':
687             case 'p':
688                 scanf("%d",&inumb);
689                 scanf("%le %le",&inumb);
690                 mnumpoi=inumb+1;
691                 continue;
692             case 't':
693                 case 't':
694                     scanf("%d",&inumb);
695                     for (i=0;i<3;i++) scanf("%d",&triang.Poi(inumb,i));
696                     mnumtri=inumb+1;

```

```

697         continue;
698     default:
699         ldata();
700         return;
701     }
702     while (scanf("%is", com_str) == 1);
703 }
704
705 /*****
706 */
707 /* ROUTINE:  ldata
708 */
709 /* ldata links all information concerning points and triangles. This
710 /* information is needed to precisely define each patch on the plate.
711 */
712 /*****
713 ldata() {
714     int i,j; /* loop variables */
715     int isi; /* scratch variable */
716     for (i=0;i<mnumpoi;i++) apoint.numatri[i]=0;
717     for (i=0;i<mnumtri;i++) {
718         for (triang.centroid.x[i]=triang.centroid.y[i]=0,j=0;j<3;j++) {
719             isi=triang.Poi(i,j);
720             triang.centroid.x[i] += point.x[isi]/3;
721             triang.centroid.y[i] += point.y[isi]/3;
722             apoint.Tri(apoint.numatri[isi], isi) = i;
723             apoint.Poi1(apoint.numatri[isi], isi) = triang.Poi(i, (j+1)%3);
724             apoint.Poi2(apoint.numatri[isi]++, isi) = triang.Poi(i, (j+2)%3);
725         }
726     }
727     return;
728 }
729 /*****
730 */
731 /* ROUTINE:  intdvz(a,b,v1,v2,ipointf)
732 */
733 /* intdvz performs integration in the v and z directions. The limits of
734 /* integration in the v direction are v1 and v2. v1 is nominally less than
735 /* v2 however this may be reversed depending on the geometry of the triangle
736 /* The integration in the z direction is from -t/2 to t/2, the observation
737 /* point lying on the z=t/2 plane. The u,v coordinates of the observation
738 /* point are given by the index ipointf to the structure point. a and b denote
739 /* the variation of the u' variable with v'. Specifically, u'=av'+b.
740 /*

```

```

741 /*****
742 double intdvz(a,b,v1,v2,ipoint)
743 double a,b,v1,v2; /* parameters of integration, see above */
744 int ipoint; /* index to point, denotes observation point */
745 {
746     double sum; /* contain sum of integrations in the z direction, corresponding
747                 to the integral evaluated for v'=v1 and v'=v2 */
748     sum=intdz(msqrt(1+a*a),(1+a*a)*v2*v2+(-2*v(ipoint)+2*a*(b-u(ipoint))))*
749           v2+v(ipoint)*v(ipoint)*(b-u(ipoint)),2*(1+a*a)*v2+(-2*v(ipoint)+
750           2*a*(b-u(ipoint)))));
751     sum -= intdz(msqrt(1+a*a),(1+a*a)*v1*v1+(-2*v(ipoint)+2*a*(b-u(ipoint))))*
752           v1+v(ipoint)*v(ipoint)*(b-u(ipoint)),2*(1+a*a)*v1+(-2*v(ipoint)+
753           2*a*(b-u(ipoint)))));
754     return(sum);
755 }
756 /*****
757 /* ROUTINE:   intdz(A,B,C)
758 /*
759 /* intdz performs the z integration. The integration is done analytically.
760 /* following Gradshteyn and Ryzhik, formulae #2.267-1,#2.261,#2.266.
761 /* The integral which is evaluated is
762 /*  $1/A \cdot \ln(2A(b+z^2)^{.5+C})$  from -t to 0.
763 /*
764 /*
765 /*****
766 double intdz(A,B,C)
767 double A,B,C; /* parameters of integration */
768 {
769     double a,b,z1,z2; /* new parameters of integration */
770     int lan0; /* False (=0) if a is effectively zero */
771     double sum; /* scratch variable, holds sum of contributions */
772     double troot; /* scratch variable, holds root of R=a+b*x+c*z**2 */
773     if (fabs(C) < Epsilon * fabs(A)) {
774         /* short cut for C = 0 */
775         sum = (fabs(B) < Epsilon * fabs(A)) ? (2*t*mlog(t)-2*t);
776             (t*mlog(B+t*t)-2*t+2*msqrt(B)*atan(t/msqrt(B)));
777         sum /= 2*A;
778         sum += t*mlog(2*A)/A;
779         return(sum);
780     }
781     if (fabs(B) < Epsilon * fabs(A)) {
782         /* short cut for B = 0 */
783         z2 = 2*A*t+C;
784         z1 = C;

```

```

785      sum = z2*mlog(z2)-z2;
786      sum -= z1*mlog(z1)-z1;
787      sum /= 2*A*A;
788      return(sum);
789    }
790    /* no short cuts */
791    a=C*C-4*B*A*A;
792    lan0 = fabs(a)>20*Epsilon*(fabs(C*C)+fabs(4*B*A*A));
793    b= -2*C;
794    z1=2*A*msqrt(B)+C;
795    z2=2*A*msqrt(t*t+B)+C;
796    sum=troot=msqrt(a+b*z2+z2*z2);
797    if (lan0) sum += (a>0)?
798      -msqrt(a)*mlog((2*a+b*z2+2*msqrt(a)*troot)/z2):
799      -msqrt(-a)*((fabs(troot)<fabs(a)*Epsilon)?
800        (((2*a+b*z2)>0)?Pi/2: -Pi/2):
801        atan((2*a+b*z2)/(2*msqrt(-a)*troot)));
802    sum += b/2*mlog(2*troot+2*z2+b);
803    sum -= troot=msqrt(a+b*z1+z1*z1);
804    if (lan0) sum -= (a>0)?
805      -msqrt(a)*mlog((2*a+b*z1+2*msqrt(a)*troot)/z1):
806      -msqrt(-a)*((fabs(troot)<fabs(a)*Epsilon)?
807        (((2*a+b*z1)>0)?Pi/2: -Pi/2):
808        atan((2*a+b*z1)/(2*msqrt(-a)*troot)));
809    sum -= b/2*mlog(2*troot+2*z1+b);
810    sum /= -2*A*A;
811    sum += t/A*mlog(2*A*msqrt(B+t*t)+C);
812    return(sum);
813  }
814  /*****
815  /* ROUTINE:  solvem(N)
816  /*
817  /*
818  /* solvem is used to solve the matrix problem.  The matrix and forcing vector*/
819  /* are created in xsolv or ysolv.  N is the dimension of the vector.  solvem */
820  /* solves the matrix problem by calling the two fortran routines dgeco and */
821  /* dgesl which perform a decomposition and back substitution.  solvem is the */
822  /* only c routine which calls a fortran routine.
823  /*
824  /*****
825  solvem(N)
826  int N; /* N is the order of the matrix stored in the array matrix.  N is less
827         than or equal to isize */
828  {

```



```

829 int isize,job; /* arguments for fortran routines, isize is the size of the
830 array matrix, job (=0) indicates type of matrix problem */
831 int ipvt[Mnumpoi]; /* an array used by the fortran routines to store the
832 pivoting vector */
833 double rcond; /* the condition number of the matrix. */
834 double z[Mnumpoi*2]; /* scratch vector, lengthened for complex case */
835 isize=Mnumpoi;
836 job=0;
837 if (!tau) dgeco (matrix,&isize,&N,ipvt,&rcond,z);
838 else cgeco (cmatrix,&isize,&N,ipvt,&rcond,z);
839 printf("condition number is %e\n",1/rcond);
840 if (!tau) dgesl (matrix,&isize,&N,ipvt,fvect,&job);
841 else cgesl_(cmatrix,&isize,&N,ipvt,cfvect,&job);
842 }
843 /*****
844 /* ROUTINE: dipole
845 /*
846 /* dipole is called by the main program after the potentials over the plate
847 /* have been computed. dipole examines each point to determine which points
848 /* are exterior points and then uses these points to define the "edge" of
849 /* of the plate. This is needed to perform the surface integral, see Senior
850 /* 1976, Low-frequency scattering by a dielectric body. The surface integral
851 /* gives the dipole moment. This is normalized by the volume of the body
852 /* which is obtained by summing the area of the individual triangles and
853 /* multiplying by the thickness t. In the process of isolating the exterior
854 /* points, any point which is incorrectly linked will be identified.
855 /*
856 /*
857 /*****
858 dipole() {
859 double dip,vol; /* dip contains the dipole moment and vol contains the
860 volume of the plate */
861 double scrat,sum; /* scrat and sum are scratch variables */
862 double sdip; /* sdip is a scratch variable */
863 double dipi,sdipi; /* imaginary part of dip and sdip */
864 double ycom,xcom,ycomp,xcomp,ycomn,xcomn; /* more scratch variables, used
865 to determine sign of dipole contributions */
866 int plisti[Mnumpoi],plist2[Mnumpoi],plist3[Mnumpoi];
867 /* plisti and plist2 are used to hold the linked
868 list of exterior points, plist3 hold the list
869 of associated triangles. The contents of these
870 arrays are pointers to point and, for plist3, to
871 triang. */
872 int elist[Mnumpoi]; /* elist contains the original, unlinked, list of

```



```

917 for (i=0,dip=0,dipi=0; i<m; i++) {
918     if (i*tau) sdip = ((potflag == 1)?
919         fabs(point.y[pl1st1[i]]-point.y[pl1st2[i]]) :
920         fabs(point.x[pl1st1[i]]-point.x[pl1st2[i]]))
921         *(point.poten[pl1st1[i]]+point.poten[pl1st2[i]])/2;
922     else {
923         sdip = ((potflag == 1)?
924             fabs(point.y[pl1st1[i]]-point.y[pl1st2[i]]) :
925             fabs(point.x[pl1st1[i]]-point.x[pl1st2[i]]))
926             *(point.cpoten[pl1st1[i]].real
927             +point.cpoten[pl1st2[i]].real)/2;
928         sdipi = ((potflag == 1)?
929             fabs(point.y[pl1st1[i]]-point.y[pl1st2[i]]) :
930             fabs(point.x[pl1st1[i]]-point.x[pl1st2[i]]))
931             *(point.cpoten[pl1st1[i]].imag
932             +point.cpoten[pl1st2[i]].imag)/2;
933     }
934 }
935 /* now determine correct sign
936 */
937 ycom = point.y[pl1st1[i]]-point.y[pl1st2[i]];
938 xcom = point.x[pl1st1[i]]-point.x[pl1st2[i]];
939 ycomp = point.y[pl1st1[i]]-triang.centroid.y[pl1st3[i]];
940 xcomp = point.x[pl1st1[i]]-triang.centroid.x[pl1st3[i]];
941 ycomn = -xcom;
942 xcomn = ycom;
943 scrat = (ycomp*ycomn+xcomp*xcomn)*((potflag == 1)?xcomn:ycomn);
944 dip += sdip*((scrat>0)?1:-1);
945 if (tau) dipi += sdipi*((scrat>0)?1:-1);
946 }
947
948 /* calculate volume of plate
949 */
950 for (i=0,vol=0;i<mnumtri;i++)
951     vol += area(point.x[triang.Poi(1,0)],point.y[triang.Poi(1,0)],
952               point.x[triang.Poi(1,1)],point.y[triang.Poi(1,1)],
953               point.x[triang.Poi(1,2)],point.y[triang.Poi(1,2)]);
954 dipmom = (tau-1)*dip/vol-tau*dipi/vol;
955 dipmomi = (i*tau)?0:((tau-1)*dipi/vol+tau*dip/vol);
956 return;
957 /*****
958          end of x or y dipole computation
959          beginning of z dipole computation
960 */

```

```

961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004

label zcase is only branched to from
one spot, immediately following error
checking routine.

*****/
zcase:
for (i=0,vol=0,dip=0,i<mmumtri;i++) {
  scrat = area(point.x[triang.Poi(1,0)],point.y[triang.Poi(1,0)],
point.x[triang.Poi(1,1)],point.y[triang.Poi(1,1)],
point.x[triang.Poi(1,2)],point.y[triang.Poi(1,2)]);
  vol += scrat;
  if (itau) {
    sdip = point.poten[triang.Poi(1,0)];
    sdip += point.poten[triang.Poi(1,1)];
    sdip += point.poten[triang.Poi(1,2)];
    sdip *= scrat/3;
    dip += sdip;
  }
  else {
    sdip = point.cpoten[triang.Poi(1,0)].real;
    sdip1 = point.cpoten[triang.Poi(1,0)].imag;
    sdip += point.cpoten[triang.Poi(1,1)].real;
    sdip1 += point.cpoten[triang.Poi(1,1)].imag;
    sdip += point.cpoten[triang.Poi(1,2)].real;
    sdip1 += point.cpoten[triang.Poi(1,2)].imag;
    sdip *= scrat/3;
    sdip1 *= scrat/3;
    dip += sdip;
    dip1 += sdip1;
  }
}
/* note that vol actually contains area */
dipmom = -2*(tau-1)*dip/(vol*t)+2*tau*dip1/(vol*t);
dipmom1 = (!tau)?0:-2*(tau-1)*dip1/(vol*t)-2*tau*dip/(vol*t);
return;
}
*****/
/* ROUTINE: sumang(1point)
/*
/* sumang is called by dipole to produce the sum of the angles associated
/* with the point 1point. 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
/*
/*

```

```

1005 /*****
1006 #define Px1 (point.x[apoint.Poi1(i.ipoint)]-point.x[ipoint]) /* shorthand */
1007 #define Px2 (point.x[apoint.Poi2(i.ipoint)]-point.x[ipoint]) /* shorthand */
1008 #define Py1 (point.y[apoint.Poi1(i.ipoint)]-point.y[ipoint]) /* shorthand */
1009 #define Py2 (point.y[apoint.Poi2(i.ipoint)]-point.y[ipoint]) /* shorthand */
1010 double sumang(ipoint)
1011 int ipoint; /* pointer to the structure point */
1012 {
1013 int i; /* loop variable */
1014 double scale,prod,sum; /* scale contains the product of the lengths of the
1015 two edges, prod contains the dot product of the
1016 two edges, and sum contains a running total of the
1017 angles. */
1018 for (i=0,sum=0;i<apoint.numatri[ipoint];i++) {
1019     scale = (Px1*Px1+Py1*Py1);
1020     scale *= (Px2*Px2+Py2*Py2);
1021     scale = sqrt(scale);
1022     prod = Px1*Px2+Py1*Py2;
1023     sum += acos(prod/scale);
1024 }
1025 if (ipoint.x[ipoint] && flagsym & 1) sum *= 2;
1026 if (ipoint.y[ipoint] && flagsym & 2) sum *= 2;
1027 sum *= 180/Pi;
1028 return(sum);
1029 }
1030 /*****
1031 /* ROUTINE: gethed
1032 /*
1033 /*
1034 /* gethed is called by the main program to get the heading of the data.
1035 /* Gethed fills hedstr with the remainder of the line (the entire line
1036 /* except for the letter H which must be in column 1.
1037 /*
1038 /*****
1039 gethed() {
1040 int i; /* loop variable */
1041 for (i=0; (hedstr[i]=getchar()) != '\n'; i++);
1042 hedstr[i] = '\0';
1043 return;
1044 }
1045 /*****
1046 /* ROUTINE: pexc
1047 /*
1048 /*

```

```
1049 /* pexc is called by the main program immediately after calling xsolv or */
1050 /* ysolv. pexc calls dipole to calculate the dipole moment, and prints the */
1051 /* result, along with information such as the heading, the material */
1052 /* parameters, the direction of excitation and the implicit symmetry of the */
1053 /* plate. */
1054 /*
1055 /******
1056 pexc() {
1057     printf("%s \n",hedstr);
1058     printf("t = %lf, tau = %lf + i %lf\n",t,tau,taui);
1059     switch (potflag) {
1060     case 0:
1061         printf("potential has not been computed\n");
1062         break;
1063     case 1:
1064         printf("potential has been computed assuming x excitation\n");
1065         break;
1066     case 2:
1067         printf("potential has been computed assuming y excitation\n");
1068         break;
1069     case 3:
1070         printf("potential has been computed assuming z excitation\n");
1071         break;
1072     }
1073     switch (flagsym) {
1074     case 0:
1075         printf("plate is not symmetric\n");
1076         break;
1077     case 1:
1078         printf("plate has mirror symmetry about x=0\n");
1079         break;
1080     case 2:
1081         printf("plate has mirror symmetry about y=0\n");
1082         break;
1083     case 3:
1084         printf("plate has mirror symmetry about x=0 and y=0\n");
1085         break;
1086     }
1087     dipole();
1088     printf("The dipole moment is %lf + i %lf\n",dipmom,dipmomi);
1089 }
1090 pmat() {
1091     int i,j;
```

```

1093 for(i=0;i<10;printf("\n"),i++)
1094     for(j=0;j<10;j++) printf("%lf ",Matrix(i,j));
1095 }
1096 pvec(){
1097     int i;
1098     for(i=0;i<10;i++) printf("%lf ",fvect[i]);
1099     printf("\n");
1100 }
1101 double msqrt(x)
1102 double x;
1103 {
1104     if (x> -1e-15) return(sqrt(x));
1105     return(sqrt(x));
1106 }
1107 double mlog(x)
1108 double x;
1109 {
1110     if (x> -1e-15) return(log(x));
1111     return(log(x));
1112 }
1113 /* function definitions */
1114 #include <math.h>
1115 double zppcon(),zptcon(),zplcon(),zint1(),zint2(),p(0,10),alpo();
1116 /* macro definitions */
1117 #define Darray(Ara,I1,I2,L1,L2) Ara[(I1)+(I2)*(L1)] /* Macro to generate other\
1118 macros to mimic fortran\
1119 arrays */
1120 #define Mnumpoi 100 /* maximum number of points */
1121 #define Mnumtri 100 /* maximum number of triangular patches */
1122 #define Mnumatri 10 /* maximum number of triangles which may share a \
1123 common point, 6 is average for internal points, 8 accounts for\
1124 most schemes, thus 10 is a reasonable upper bound */
1125 #define True 1 /* value of logical true */
1126 #define False 0 /* value of logical false */
1127 #define Tri(J1,J2) Darray(tri,J1,J2,Mnumatri,Mnumpoi) /* set up tri as two\
1128 dimensional array with limits (Mnumatri,Mnumpoi) */
1129 #define Poi1(J1,J2) Darray(poi1,J1,J2,Mnumatri,Mnumpoi) /* set up poi1 as two\
1130 dimensional array with limits (Mnumatri,Mnumpoi) */
1131 #define Poi2(J1,J2) Darray(poi2,J1,J2,Mnumatri,Mnumpoi) /* set up poi2 as two\
1132 dimensional array with limits (Mnumatri,Mnumpoi) */
1133 #define Poi(J1,J2) Darray(poi,J1,J2,Mnumtri,3) /* set up poi as two dimensional\
1134 array with limits (Mnumtri,3) */
1135 #define Matrix(J1,J2) Darray(matrix,J1,J2,Mnumpoi,Mnumpoi) /* set up matrix as\
1136 two dimensional array with limits (Mnumpoi,Mnumpoi) */

```

```

1137 #define Cmatrix(J1,J2) Darray(cmatrix,J1,J2,Mnumpoi,Mnumpoi) /* complex
1138 version of Matrix, used with the complex
1139 structure */
1140 struct complex {
1141     double real;
1142     double imag;
1143 };
1144 /*
1145 /*
1146 /* End of Definitions / Beginning of external variable declarations
1147 /*
1148 /*
1149 extern
1150 int flagsym; /* flagsym may assume values of 0,1,2, or 3. Flagsym is used
1151 by low level routines to incorporate symmetry in a manner
1152 invisible to the rest of the program.
1153 0 - indicates no symmetry
1154 1 - indicates object possesses mirror symmetry about x=0
1155 2 - indicates object possesses mirror symmetry about y=0
1156 3 - indicates object possesses mirror symmetry about x=0 and y=0
1157 Flagsym is set at the beginning of the main program and after
1158 that is never changed
1159 extern
1160 int flagsyms; /* flagsyms is used in conjunction with flagsym to generate
1161 mirror images of the source patches while calculating
1162 contributions to the field point. The initial object
1163 description is assumed to lie in the first quadrant, however
1164 if flagsym is 0, this is not necessary. Flagsyms is always
1165 less than or equal to flagsym, and may assume the values of
1166 0,1,2, or 3. These values are given the following meanings:
1167 0 - source patch is original patch (presumably first quadrant)
1168 1 - source patch is in second quadrant
1169 2 - source patch is in fourth quadrant
1170 3 - source patch is in third quadrant */
1171 extern
1172 struct {
1173     double x[Mnumpoi];
1174     double y[Mnumpoi];
1175     double poten[Mnumpoi];
1176 }point; /* complex cpoten[Mnumpoi]; /* complex version of poten */
1177 /* point is a structure which contains the locations of all
1178 of the points used in defining the triangular patches. A point
1179 number 0 is permitted as C defines arrays beginning with 0.
1180 X and y are the coordinates of the points, and

```



```

1181     poten is the potential of each point, as determined from the
1182     matrix problem. */
1183 extern
1184 struct {
1185     int numatri[Mnumpoi];
1186     int Tri(Mnumatri,Mnumpoi-1);
1187     int Poi1(Mnumatri,Mnumpoi-1);
1188     int Poi2(Mnumatri,Mnumpoi-1);
1189 } apoint; /* apoint is a structure which contains lists of triangles
1190 associated with each point. numatri contains the number of
1191 triangles associated with each point; tri contains the list of
1192 triangle numbers associated with each point; and poi1 and poi2
1193 contain the other two vertices used in defining the triangle.
1194 poi1 and poi2 are indices to point. tri is an index to
1195 triang. */
1196 extern
1197 struct {
1198     struct {
1199         double x[Mnumtri];
1200         double y[Mnumtri];
1201     } centroid;
1202     int Poi(Mnumtri,3-i);
1203     double poten[Mnumtri];
1204     struct complex cpoten[Mnumtri]; /* complex version of poten */
1205 } triang; /* triang is used to solve the scattering problem with z
1206 excitation. centroid .x and .y contain the coordinates
1207 of the centroid of each triangle. poi contains the list of
1208 points (vertices) associated with each triangle, and is an
1209 index to point. poten is the potential of each triangle as
1210 determined from solution of the matrix problem. */
1211 extern
1212 double Matrix(Mnumpoi,Mnumpoi-1); /* This is "THE" matrix. Trivial points
1213 (ie, those which have zero potential from
1214 symmetry considerations) are skipped when
1215 reading or filling the matrix, which is
1216 always done sequentially. */
1217 extern
1218 struct complex Cmatrix(Mnumpoi,Mnumpoi-1); /* complex version of Matrix */
1219 extern
1220 double fvect[Mnumpoi]; /* fvect is the forcing vector for the matrix problem
1221 and after solution of the matrix problem contains
1222 the solution vector. */
1223 extern
1224 struct complex cfvect[Mnumpoi]; /* complex version of fvect */

```

```
1225 extern
1226 int mnumpoi; /* This is the total number of points. Points are assumed to be
1227 numbered sequentially from 0. mnumpoi <= Mnumpoi. */
1228 extern
1229 int mnumtri; /* This is the total number of triangles. Triangles are assumed
1230 to be numbered sequentially from 0. mnumtri <= Mnumtri. */
1231 extern
1232 double t; /* Thickness of the plate */
1233 extern
1234 double tau; /* permittivity of the plate */
1235 extern
1236 double tau1; /* imaginary part of the permittivity. This variable is also
1237 used as a flag: if tau1 is zero, computations are performed
1238 assuming "tau" is purely real; if tau1 is non-zero, the routines
1239 appropriate for a complex tau are invoked */
1240 extern
1241 double dipmom; /* contains the real part of the computed dipole moment */
1242 extern
1243 double dipmomi; /* contains the imaginary part of the computed dipole moment */
1244 extern
1245 double Epsilon; /* A very small number, used for approximate equality */
1246 extern
1247 double P1; /* P1 */
1248 /*
1249 local global definitions
1250 */
1251 #define Pointx(I) (point.x[I]*(flagsyms & 1?-1:1)) /* reflect x coordinate */
1252 #define Pointy(I) (point.y[I]*(flagsyms & 2?-1:1)) /* reflect y coordinate */
1253 /*
1254 variable declarations
1255 */
1256 double d; /* set to either t or 0, depending on which side of the
1257 surface is being integrated */
1258 double zx0,zy0,alp,calp,salp;
1259 /* these variables are set by zrotat when the p-1 coordinate
1260 system is defined. They are used by p() and l(). */
1261 /*****
1262 */
1263 /* ROUTINE: zsolve()
1264 /* zsolve is called by the main program to calculate the potential on a
1265 /* resulting from z excitation. zsolve fills Matrix using zppcon (point-
1266 /* point) contributions. The matrix problem is solved by solve.
1267 /*
1268 /*****
```

```

1269 zsolve() {
1270   int ipoint, ipois; /* pointers to field point and source point, respectively */
1271   for (ipoint=0; ipoint < mnumpoi; ipoint++) {
1272     for(ipois=0; ipois<mnumpoi; ipois++) {
1273       if (!itau) Matrix(ipoint, ipois) = (tau-1)/t*zppcon(ipoint, ipois);
1274       else {
1275         Cmatrix(ipoint, ipois).real=(tau-1)/t*zppcon(ipoint, ipois);
1276         Cmatrix(ipoint, ipois).imag=tau/t*zppcon(ipoint, ipois);
1277       }
1278       if (ipoint == ipois) {
1279         if (!itau) Matrix(ipoint, ipois) += .5;
1280         else Cmatrix(ipoint, ipois).real += .5;
1281       }
1282     }
1283     if (!itau) fvect[ipoint] = -t/2;
1284     else {
1285       cfvect[ipoint].real = -t/2;
1286       cfvect[ipoint].imag = 0;
1287     }
1288   }
1289   if (!itau) {
1290     pmat();
1291     pvec();
1292   }
1293   solvem(ipoint);
1294   if (!itau) {
1295     pmat();
1296     pvec();
1297   }
1298   for (ipoint=0; ipoint < mnumpoi; ipoint++)
1299     if (!itau) point.poten[ipoint] = fvect[ipoint]/2;
1300     else {
1301       point.cpoten[ipoint].real = cfvect[ipoint].real/2;
1302       point.cpoten[ipoint].imag = cfvect[ipoint].imag/2;
1303     }
1304 }
1305 /*****
1306 */
1307 /* ROUTINE: zppcon(ipoint, ipois)
1308 /* zppcon is called by zsolve to calculate the point to point contribution
1309 /* to the integral. zppcon evaluates the surface integral on the top and
1310 /* the bottom by setting d=0,t and calling zptcon.
1311 /*
1312 /*****

```

```

1313 double zppcon(ipoif,ipois)
1314 int ipoif,ipois; /* pointers to field and source points, respectively */
1315 {
1316 int i; /* loop variable */
1317 double sum; /* holds sum of contributions from both sides of all
1318             relevant triangles */
1319 for (i=0,sum=0; i<apoint.numatri[ipois];i++) {
1320     d=0;
1321     sum += zptcon(ipoif,ipois,apoint.Poi1(1,ipois),apoint.Poi2(1,ipois));
1322     d=t;
1323     sum -= zptcon(ipoif,ipois,apoint.Poi1(1,ipois),apoint.Poi2(1,ipois));
1324 }
1325 return(sum/(4*Pi));
1326 }
1327 /*****
1328 /* ROUTINE: zptcon(ipoif,ipois,ipoe1,ipoe2)
1329 /* zptcon is called by zppcon to calculate the point-triangle contribution
1330 /* to the surface integral. zptcon accomplishes this by calling zplcon
1331 /* to evaluate the line integral about the triangle (the surface integral
1332 /* is converted to a line integral via Wilton's method).
1333 /*
1334 /*
1335 /*****
1336 double zptcon(ipoif,ipois,ipoe1,ipoe2)
1337 int ipoif,ipois,ipoe1,ipoe2; /* pointers to the field point, source point,
1338                               first point of edge opposing ipois in desired triangle, and
1339                               second point of edge opposing ipois in desired triangle. */
1340 {
1341 double sum; /* holds sum of integration on three edges of triangle */
1342 for (sum=0,flagsyms=0; flagsyms <= flagsym; flagsyms += (flagsym & 1)?1:2) {
1343     sum += zplcon(ipoif,ipois,ipoe1,ipoe2,ipois,ipoe1);
1344     sum += zplcon(ipoif,ipois,ipoe1,ipoe2,ipoe1,ipoe2);
1345     sum += zplcon(ipoif,ipois,ipoe1,ipoe2,ipoe2,ipois);
1346 }
1347 return(sum);
1348 }
1349 /*****
1350 /* ROUTINE: zplcon(ipoif,ipois,ipoe1,ipoe2,ipoi1,ipoi2)
1351 /* zplcon is called by zptcon to calculate the contribution of one line of
1352 /* a triangle to a point. zplcon performs this by evaluating two line
1353 /* integrals via the routines zint1 and zint2.
1354 /*
1355 /*
1356 /*****

```

```

1357 #define Dot(X1,Y1,X2,Y2) (X1*X2+Y1*Y2) /* define dot product */
1358 #define Vx(I2,I1) (Pointx(I2)-Pointx(I1)) /* x component of vector */
1359 #define Vy(I2,I1) (Pointy(I2)-Pointy(I1)) /* y component of vector */
1360 #define Vfx(I1) (Pointx(I1)-point.x[ipoi1]) /* x component relative to ipoi1 */
1361 #define Vfy(I1) (Pointy(I1)-point.y[ipoi1]) /* y component relative to ipoi1 */
1362 #define Unit(X,Y) temp=sqrt(X*X+Y*Y); \
1363 X /= temp; \
1364 Y /= temp; /* normalize the vector pair X,Y */
1365 double zplcon(ipoi1,ipoi2,ipoe1,ipoe2,ipoi1,ipoi2)
1366 int ipoi1,ipoi2,ipoe1,ipoe2,ipoi1,ipoi2;
1367 /* ipoi1 is the pointer to the field point,
1368 ipoi2 is the pointer to the source point,
1369 ipoe1 is the pointer to the first point associated with the
1370 edge which opposes the source point in the current triangle,
1371 ipoe2 is the pointer to the second point associated with the
1372 edge which opposes the source point in the current triangle,
1373 ipoi1 is the pointer to the first point associated with the
1374 line over which the integration is to be performed in the
1375 current triangle,
1376 ipoi2 is the pointer to the second point associated with the
1377 line over which the integration is to be performed in the
1378 current triangle.
1379 */
1380 {
1381 int ipoi3; /* ipoi3 is the third point in the triangle ipoi1,ipoi2,ipoi3 */
1382 double x0,y0; /* will contain coordinates of the point on line ipoi1-
1383 ipoi2 which is closest to ipoi1 */
1384 double a,b,c; /* contains the parameters which describe line ipoi1-ipoi2 */
1385 int iscr; /* scratch variable */
1386 double sum; /* holds result of integrations */
1387 double tx,ty; /* temporary variables, used to hold x and y components of
1388 a vector */
1389 double temp; /* used only in the macro Unit, hold intermediate computation */
1390 double ux,uy; /* holds x and y components of the unit vector u */
1391 double qx,qy; /* holds x and y components of the unit vector q */
1392 double scale1, scale2; /* these are the factors by which the two integrations,
1393 zint1 and zint2, are scaled */
1394 double mq; /* holds the magnitude of the q vector */
1395 double tl; /* scratch variable */
1396 /* calculate scaling factors */
1397 tx = Vx(ipoi2,ipoi1);
1398 ty = Vy(ipoi2,ipoi1);
1399 ux = tx;
1400 uy = -tx;

```

```
1401 Unit(ux, uy);
1402 ipo13 = (i((ipo11-ipoe1)*(ipo12-ipoe1))?
1403 ((ipo11-ipoe2)*(ipo12-ipoe2)?ipoe2:ipo1s):ipoe1);
1404 if (Dot(Vx(ipo11, ipo13), Vy(ipo11, ipo13), ux, uy)<0) {
1405     ux = -ux;
1406     uy = -uy;
1407 }
1408 /* now find q, first find qhat */
1409 tx = Vx(ipoe2, ipoe1);
1410 ty = Vy(ipoe2, ipoe1);
1411 qx = ty;
1412 qy = -tx;
1413 Unit(qx, qy);
1414 if (Dot(Vx(ipoe1, ipo1s), Vy(ipoe1, ipo1s), qx, qy)>0) {
1415     qx = -qx;
1416     qy = -qy;
1417 }
1418 /* next find mag of q, mq */
1419 t1 = -Dot(Vx(ipoe1, ipo1s), Vy(ipoe1, ipo1s), qx, qy);
1420 mq = 1/t1;
1421 /* now have enough to get u dot q, next get offset */
1422 t1 = Dot(Vfx(ipoe1), Vfy(ipoe1), qx, qy)*mq;
1423 scale1 = -t1;
1424 scale2 = mq*Dot(ux, uy, qx, qy);
1425 /* find x0, y0: needed to set up 1-p coordinate system to facilitate
1426 integration, cy=ax+b, describes line ipo11 - ipo12 */
1427 if (fabs(Pointx(ipo11)-Pointx(ipo12)) < Epsilon) {
1428     c=0;
1429     a=1;
1430     b = -Pointx(ipo11);
1431 }
1432 else {
1433     c=1;
1434     a=(Pointy(ipo11)-Pointy(ipo12))/(Pointx(ipo11)-Pointx(ipo12));
1435     b=Pointy(ipo11)-a*Pointx(ipo11);
1436 }
1437 if (c==0) {
1438     x0=Pointx(ipo11);
1439     y0=point.y[ipo1f];
1440 }
1441 else { /* find closest point on line to ipo1f */
1442     x0 = (point.x[ipo1f]+a*point.y[ipo1f]-a*b)/(1+a*a);
1443     y0 = a*x0+b;
1444 }
```

```

1445 /* generate (p,l) coordinate system with ipof at center and x0,y0 on p axis */
1446 zrotat(point.x[ipof],point.y[ipof],x0,y0,ipol1,ipol2);
1447 if (l(ipol1) > l(ipol2)) {
1448   iscr = ipol1;
1449   ipol1=ipol2;
1450   ipol2=iscr;
1451 }
1452 if (p(ipol1)>Epsilon)
1453   sum = scale1*zint1(ipof,ipol,ipol,ipol,ipol,ipol,ipol,ipol,ipol,ipol);
1454   * ((p(ipol1)>p(ipol3))?-1);
1455   else sum = 0;
1456   sum += scale2*zint2(ipof,ipol,ipol,ipol,ipol,ipol,ipol,ipol,ipol,ipol);
1457   if (point.x[ipof] == Pointx(ipol) && point.y[ipof] == Pointy(ipol))
1458     sum -= scale1*d*alpo(ipol,ipol,ipol,ipol,ipol,ipol,ipol,ipol,ipol,ipol)/2;
1459   if (point.x[ipof] == Pointx(ipol) && point.y[ipof] == Pointy(ipol))
1460     sum -= scale1*d*alpo(ipol,ipol,ipol,ipol,ipol,ipol,ipol,ipol,ipol,ipol)/2;
1461   return(sum);
1462 }
1463 /******
1464 */
1465 /* ROUTINE: zint1(ipof,ipol,ipol,ipol,ipol,ipol,ipol,ipol,ipol,ipol)
1466 /* zint1 is called by zplcon to calculate the line integral resulting from
1467 /* the surface integral 1/R. The line integral is evaluated over the line
1468 /* segment ipol1 - ipol2.
1469 */
1470 /******
1471 #define R(I) (sqrt(l(I)*l(I)+p(I)*p(I)+d*d))
1472 double zint1(ipof,ipol,ipol,ipol,ipol,ipol,ipol,ipol,ipol,ipol)
1473 int ipof,ipol,ipol,ipol,ipol,ipol,ipol,ipol,ipol,ipol;
1474 /* ipof is the pointer to the field point,
1475 ipol is the pointer to the source point,
1476 ipol is the pointer to the first point associated with the
1477 edge which opposes the source point in the current triangle,
1478 ipol is the pointer to the second point associated with the
1479 edge which opposes the source point in the current triangle,
1480 ipol is the pointer to the first point associated with the
1481 line over which the integration is to be performed in the
1482 current triangle,
1483 ipol2 is the pointer to the second point associated with the
1484 line over which the integration is to be performed in the
1485 current triangle.
1486 */
1487 {
1488   double p0; /* distance from ipof to line ipol1-ipol2 */

```

```

1489 double sum; /* holds contributions to integral */
1490 sum=0;
1491 p0=p(ipoi1);
1492 if (p0 < Epsilon) return(sum);
1493 sum += log(1(ipoi2))+sqrt(p0*p0+d*d+1(ipoi2)*1(ipoi2));
1494 sum -= log(1(ipoi1))+sqrt(p0*p0+d*d+1(ipoi1)*1(ipoi1));
1495 sum *= p0;
1496 if (d > Epsilon) {
1497     sum += d*atan(d*1(ipoi2))/(p0*R(ipoi2));
1498     sum -= d*atan(d*1(ipoi1))/(p0*R(ipoi1));
1499 }
1500 return(sum);
1501 }
1502 /*****
1503 */
1504 /* ROUTINE: zint2(ipoi1,ipoi2,ipoe1,ipoe2,ipoi1,ipoi2)
1505 /* zint2 is called by zplcon to calculate the line integral resulting from
1506 /* the surface integral x/R. The line integral is evaluated over the line
1507 /* segment ipoi1 - ipoi2.
1508 */
1509 /*****
1510 double zint2(ipoi1,ipoi2,ipoe1,ipoe2,ipoi1,ipoi2)
1511 int ipoi1,ipoi2,ipoe1,ipoe2,ipoi1,ipoi2;
1512 /* ipoi1 is the pointer to the field point,
1513 /* ipoi2 is the pointer to the source point,
1514 /* ipoe1 is the pointer to the first point associated with the
1515 /* edge which opposes the source point in the current triangle,
1516 /* ipoe2 is the pointer to the second point associated with the
1517 /* edge which opposes the source point in the current triangle,
1518 /* ipoi1 is the pointer to the first point associated with the
1519 /* line over which the integration is to be performed in the
1520 /* current triangle,
1521 /* ipoi2 is the pointer to the second point associated with the
1522 /* line over which the integration is to be performed in the
1523 /* current triangle.
1524 */
1525 {
1526     double p0; /* distance from ipoi1 to line ipoi1-ipoi2 */
1527     double sum; /* holds contributions to integral */
1528     double r02; /* may equal p0**2 or p0**2+t**2, depending on d */
1529     sum=0;
1530     p0=p(ipoi1);
1531     r02=p0*p0+d*d;
1532     if(r02<Epsilon){

```



```

1533 sum += l(ipoi2)*l(ipoi2);
1534 sum -= l(ipoi1)*l(ipoi1)*(l(ipoi1)*l(ipoi2)>0?1:-1);
1535 sum = fabs(sum);
1536 sum /= 2;
1537 }
1538 else {
1539     sum += r02*log(R(ipoi2)+l(ipoi2))+l(ipoi2)*R(ipoi2);
1540     sum -= r02*log(R(ipoi1)+l(ipoi1))+l(ipoi1)*R(ipoi1);
1541     sum /= 2;
1542 }
1543 return(sum);
1544 }
1545 /*****
1546 */
1547 /* ROUTINE: zrotat(x1,y1,x2,y2,ipoi1,ipoi2)
1548 /* zrotat is called by zpicon to initialize the (p,l) coordinate system.
1549 /* zrotat initializes several external variables which are used by p() and
1550 /* l().
1551 /*
1552 /*****
1553 zrotat(x1,y1,x2,y2,ipoi1,ipoi2)
1554 double xi,y1,x2,y2; /* xi,y1 are the coordinates of the origin of the (p,l)
1555 coordinate system. x2,y2 are the coordinate of a point
1556 which should lie on the p axis */
1557 int ipoi1,ipoi2; /* pointers, used if xi=x2 and yi=y2 */
1558 {
1559     x0=xi;
1560     y0=yi;
1561     if (fabs(x2-x1)+fabs(y2-y1)<Epsilon)
1562         /* line up with perpendicular to ipoi1-ipoi2 */
1563         alp=atan2(Pointx(ipoi2)-Pointx(ipoi1),-(Pointy(ipoi2)-Pointy(ipoi1)));
1564     else
1565         /* use standard orientation */
1566         alp=atan2(y2-y1,x2-x1);
1567     calp=cos(alp);
1568     salp=sin(alp);
1569     return;
1570 /*****
1571 */
1572 /* ROUTINES: l(i1) and p(i1)
1573 /* l(i1) and p(i1) return, respectively, the reflected coordinates of the
1574 /* point i1. Since l() and p() are never used with ipoi1, it is safe to
1575 /* reflect all points.
1576 /*

```



```

1621 #define Mnumpoi 100 /* maximum number of points */
1622 #define Mnumtri 100 /* maximum number of triangular patches */
1623 #define Mnumatri 10 /* maximum number of triangles which may share a \
1624 common point, 6 is average for internal points, 8 accounts for \
1625 most schemes, thus 10 is a reasonable upper bound */
1626 #define True 1 /* value of logical true */
1627 #define False 0 /* value of logical false */
1628 #define Tri(J1,J2) Darray(tri,J1,J2,Mnumatri,Mnumpoi) /* set up tri as two\
1629 dimensional array with limits (Mnumatri, Mnumpoi) */
1630 #define Poi1(J1,J2) Darray(poi1,J1,J2,Mnumatri,Mnumpoi) /* set up poi1 as two\
1631 dimensional array with limits (Mnumatri, Mnumpoi) */
1632 #define Poi2(J1,J2) Darray(poi2,J1,J2,Mnumatri,Mnumpoi) /* set up poi2 as two\
1633 dimensional array with limits (Mnumatri, Mnumpoi) */
1634 #define Poi(J1,J2) Darray(poi,J1,J2,Mnumtri,3) /* set up poi as two dimensional\
1635 array with limits (Mnumtri,3) */
1636 #define Matrix(J1,J2) Darray(matrix,J1,J2,Mnumpoi,Mnumpoi) /* set up matrix as\
1637 two dimensional array with limits (Mnumpoi,Mnumpoi) */
1638 #define Cmatrix(J1,J2) Darray(cmatrix,J1,J2,Mnumpoi,Mnumpoi) /* complex\
1639 version of Matrix, used with the complex\
1640 structure */
1641 struct complex {
1642 double real;
1643 double imag;
1644 };
1645 /*
1646 /*
1647 /*
1648 /*
1649 /*
1650 extern
1651 int potflag; /* flag marks whether or not potentials have been computed
1652 0 - potential has not been computed
1653 1 - potential has been computed assuming x excitation
1654 2 - potential has been computed assuming y excitation
1655 3 - potential has been computed assuming z excitation */
1656 extern
1657 struct {
1658 double x[Mnumpoi];
1659 double y[Mnumpoi];
1660 double poten[Mnumpoi];
1661 struct complex cpoten[Mnumpoi];
1662 }point; /* point is a structure which contains the locations of all
1663 of the points used in defining the triangular patches. A point
1664 number 0 is permitted as C defines arrays beginning with 0.

```

```

1665 X and Y are the coordinates of the points, and
1666 poten is the potential of each point, as determined from the
1667 matrix problem. */
1668 extern
1669 double t; /* Thickness of the plate */
1670 extern
1671 double tau; /* permittivity of the plate */
1672 extern
1673 double tau1; /* imaginary part of the permittivity. This variable is also
1674 used as a flag: if tau1 is zero, computations are performed
1675 assuming "tau" is purely real; if tau1 is non-zero, the routines
1676 appropriate for a complex tau are invoked */
1677 extern
1678 double dipmom; /* contains the real part of the computed dipole moment */
1679 extern
1680 double dipmomi; /* contains the imaginary part of the computed dipole moment */
1681 extern
1682 double Epsilon; /* A very small number, used for approximate equality */
1683 extern
1684 double Pi; /* Pi */
1685 /*****
1686 */
1687 /* ROUTINE: eigen
1688 */
1689 /* eigen is called by the main program to calculate eigenvalues. eigen is
1690 is designed to function on a simplified version of the general plate
1691 problem, so as to affect a linear predictor model of the dipole moment.
1692 The plate must have full x-y symmetry (ie, s3), and should be described
1693 via a single triangle, with point 0 falling at the origin, point 1 lying
1694 on the y axis, and point 2 lying on the x axis. The material parameters
1695 should be specified before invoking eigen() via the E command. eigen
1696 uses the thickness specified in the m command, but ignores the
1697 permittivity. eigen examines x, y, and z excitation, and will print out
1698 the eigenvalue for each case along with a dipole moment which would result
1699 if tau=2 and the field variation within the plate was unity, or rather,
1700 the parameter a (see write up) equals 1.
1701 */
1702 /*****
1703 eigen() {
1704 double integra; /* holds result of integration */
1705 double leng; /* holds x or y position of test point */
1706 double lambda; /* holds eigenvalue */
1707 /* first do x excitation */
1708 potflag=1;

```

```

1709 integra = ppcon(2,2)*point.x[2];          /* normalize for unit slope */
1710 leng = point.x[2];
1711 lambda = 1-leng/integra;
1712 point.poten[0]=point.poten[1]=0;
1713 point.poten[2]=point.x[2];
1714 tau=2;
1715 tau1=0;
1716 dipole();
1717 printf("thickness = %lf\n",t);
1718 printf(" x excitation, eigenvalue = %lf, I = %lf, punit = %lf\n",lambda,integra,dipmom);
1719 /* now for y excitation */
1720 potflag=2;
1721 integra = ppcon(1,1)*point.y[1];
1722 leng = point.y[1];
1723 lambda = 1-leng/integra;
1724 point.poten[0]=point.poten[2]=0;
1725 point.poten[1]=point.y[1];
1726 dipole();
1727 printf(" y excitation, eigenvalue = %lf, I = %lf, punit = %lf\n",lambda,integra,dipmom);
1728 /* and now for z excitation */
1729 potflag=3;
1730 integra = zppcon(0,0)+zppcon(0,1)+zppcon(0,2);
1731 lambda = 1 - t/(2*integra);
1732 point.poten[0]=point.poten[1]=point.poten[2]=t/2;
1733 dipole();
1734 printf(" z excitation, eigenvalue = %lf, I = %lf, punit = %lf\n",lambda,integra,dipmom);
1735 return;
1736 }
1737 /* function definitions */
1738 #include <math.h>
1739 #include <stdio.h>
1740 double area();
1741 /* macro definitions */
1742 #define Darray(Ara,I1,I2,L1,L2) Ara[(I1)+(I2)*(L1)] /* Macro to generate other
1743 macros to mimic fortran
arrays */
1744 #define Nnumpoi 100 /* maximum number of points */
1745 #define Nnumtri 100 /* maximum number of triangular patches */
1746 #define Nnumatri 10 /* maximum number of triangles which may share a \
1747 common point, 6 is average for internal points, 8 accounts for\
1748 most schemes, thus 10 is a reasonable upper bound */
1749 #define Nnumplo 51 /* number of lines to plot in graph, must be odd, < 60 */
1750 #define True 1 /* value of logical true */
1751 #define False 0 /* value of logical false */
1752

```

```

1753 #define Para(J1,J2) Darray(para,J1,J2,Mnumplo,Mnumplo) /* set up para as
1754 two dimensional array with limits (Mnumplo, Mnumplo) */
1755 #define Tri(J1,J2) Darray(tri,J1,J2,Mnumatri,Mnumpoi) /* set up tri as two\
1756 dimensional array with limits (Mnumatri, Mnumpoi) */
1757 #define Poi(J1,J2) Darray(poi,J1,J2,Mnumtri,3) /* set up poi as two dimensional\
1758 array with limits (Mnumtri,3) */
1759 #define X0 point.x[triang.Poi(1,0)] /* set up shorthand notations */
1760 #define X1 point.x[triang.Poi(1,1)]
1761 #define X2 point.x[triang.Poi(1,2)]
1762 #define Y0 point.y[triang.Poi(1,0)]
1763 #define Y1 point.y[triang.Poi(1,1)]
1764 #define Y2 point.y[triang.Poi(1,2)]
1765 /*
1766 /*
1767 /* End of Definitions / Beginning of external variable declarations
1768 /*
1769 /*
1770 extern
1771 int flagsym; /* flagsym may assume values of 0,1,2, or 3. Flagsym is used
1772 by low level routines to incorporate symmetry in a manner
1773 invisible to the rest of the program.
1774 0 - indicates no symmetry
1775 1 - indicates object possesses mirror symmetry about x=0
1776 2 - indicates object possesses mirror symmetry about y=0
1777 3 - indicates object possesses mirror symmetry about x=0 and y=0
1778 Flagsym is set at the beginning of the main program and after
1779 that is never changed
1780 extern
1781 int flagsyms; /* flagsyms is used in conjunction with flagsym to generate
1782 mirror images of the source patches while calculating
1783 contributions to the field point. The initial object
1784 description is assumed to lie in the first quadrant, however
1785 if flagsym is 0, this is not necessary. Flagsyms is always
1786 less than or equal to flagsym, and may assume the values of
1787 0,1,2, or 3. These values are given the following meanings:
1788 0 - source patch is original patch (presumably first quadrant)
1789 1 - source patch is in second quadrant
1790 2 - source patch is in fourth quadrant
1791 3 - source patch is in third quadrant */
1792 extern
1793 int plotsym; /* plotsym indicates what mirroring is desired for plotting
1794 0 - no mirroring
1795 1 - mirrored about x=0
1796 2 - mirrored about y=0

```

```

1797 3 - mirrored about x=0 and y=0
1798 Any mirroring specified must also have been previously
1799 specified in the generation of the object (see flagsym). */
1800 extern
1801 int potriag; /* flag marks whether or not potentials have been computed
1802 0 - potential has not been computed
1803 1 - potential has been computed assuming x excitation
1804 2 - potential has been computed assuming y excitation
1805 3 - potential has been computed assuming z excitation */
1806 float Para(Mnumplo,Mnumplo-1); /* Para contains the potentials which will
1807 be plotted */
1808 extern
1809 struct {
1810     double x[Mnumpoi];
1811     double y[Mnumpoi];
1812     double poten[Mnumpoi];
1813 }point; /* point is a structure which contains the locations of all
1814 of the points used in defining the triangular patches. A point
1815 number 0 is permitted as C defines arrays beginning with 0.
1816 X and y are the coordinates of the points, and
1817 poten is the potential of each point, as determined from the
1818 matrix problem. */
1819 extern
1820 struct {
1821     struct {
1822         double x[Mnumtri];
1823         double y[Mnumtri];
1824     } centroid;
1825     int Poi(Mnumtri,3-1);
1826     double poten[Mnumtri];
1827 } triang; /* triang is used to solve the scattering problem with z
1828 excitation. centroid .x and .y contain the coordinates
1829 of the centroid of each triangle. poi contains the list of
1830 points (vertices) associated with each triangle, and is an
1831 index to point. poten is the potential of each triangle as
1832 determined from solution of the matrix problem. */
1833 extern
1834 int mnumpoi; /* This is the total number of points. Points are assumed to be
1835 numbered sequentially from 0. mnumpoi <= Mnumpoi. */
1836 extern
1837 int mnumtri; /* This is the total number of triangles. Triangles are assumed
1838 to be numbered sequentially from 0. mnumtri <= Mnumtri. */
1839 extern
1840 double Epsilon; /* A very small number, used for approximate equality */

```

```
1841 extern
1842 double P1; /* P1 */
1843 /*****
1844 /* ROUTINE: graph
1845 /*
1846 /* graph is called by the main program to graph the calculated potentials.
1847 /* Graph accomplishes this by rerouting the standard output to the input
1848 /* Of a call to imprint. imprint in turn produces a graph and queues it for
1849 /* printing. All variables which must be transferred to graph and its
1850 /* subordinate routines are passed from the main program via external
1851 /* variables.
1852 /*
1853 /*
1854 /*****
1855 Graph() {
1856 FILE *p_imprint;
1857 FILE *popen();
1858 int ret1, oldstdout;
1859 printf("This shouldn't go to the file\n");
1860 fflush(stdout);
1861 oldstdout=dup(fileno(stdout));
1862 if (oldstdout<0) perror("dup failed");
1863 /* save stdout pointer */
1864 p_imprint=popen("imprint -n -I -Ltekrnix");
1865 if (p_imprint == NULL) perror("popen died");
1866 ret1=dup2(fileno(p_imprint),fileno(stdout)); /*make stdout the same as imprint*/
1867 if (ret1<0) perror("dup2 failed");
1868 graphp();
1869 fflush(stdout);
1870 ret1=close(fileno(stdout));
1871 if (ret1<0) perror("close of stdout file");
1872 ret1=dup2(oldstdout,fileno(stdout));
1873 if (ret1<0) perror("restore of stdout failed");
1874 ret1=pclose(p_imprint);
1875 if (ret1<0) perror("pclose failed");
1876 ret1=close(oldstdout);
1877 if (ret1<0) perror("close failed");
1878 printf("This shouldn't go to the file either.\n");
1879 }
1880 Graphy() {
1881 int i,j;
1882 float nara[2500],angle,zmax,zscale;
1883 double P1=3.14159265;
1884 P1 *= 2./5.;
```



```

1885 zscale = 1.;
1886 for (i=0;i<50;i++) {
1887   for (j=0;j<50;j++) {
1888     nara[i+50*j]=zscale*sin(i*Pi/10)*sin(j*Pi/10)+zscale;
1889   }
1890 }
1891 angle=30.;
1892 zmax=2.;
1893 dgsurf(nara,50,angle,zmax,1.0);
1894 }
1895 /*****
1896 */
1897 /* ROUTINE: graphp
1898 */
1899 /* graphp is called by graph. graphp performs the necessary calculations to
1900 /* fill para in a manner acceptable to dgsurf.
1901 */
1902 /*****
1903 graphp() {
1904 double m12,m20,m01; /* these variables contain the slope of the lines
1905 connecting points 12, 20, and 01, respectively. */
1906 double my12,my20,my01; /* These variables are used if the line is vertical,
1907 implying that m** should be infinity. Since this
1908 is not possible, that formulation of the line is
1909 changed to my*=m*x+b, with my=0. Except for
1910 this case of a vertical line, my=1. */
1911 double b12,b20,b01; /* these variables contain the y offset of the lines
1912 connecting points 12, 20, and 01, respectively. */
1913 double mby12,mby20,mby01; /* these variables contain the evaluation of the
1914 expression mx+b-y, where x and y are the
1915 coordinates of the centroid of the triangle,
1916 and m and b are defined above by point pairs
1917 using the formulation for the line as y=mx+b.
1918 The only importance of the mby** variables is
1919 the sign of the number they contain, since if
1920 an arbitrary point (x*,y*) substituted into
1921 the expression mx+b-y produces a similar sign,
1922 then both the centroid and the point (x*,y*)
1923 are on the same side of the line. By making
1924 this comparison for all three lines, it is
1925 determined whether or not an arbitrary point
1926 lies within a given triangle. */
1927 float zofset; /* zofset contains the offset which is added to all
1928 potentials as they are stored in para. This is

```

```

1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972

```

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. zpmx is the maximum value
to be plotted (usually zmax+zofset). zscrat is
a scratch variable used in performing the above
described calculations. */
parameters which are passed to dgsurf. zpmx
is the maximum value to be graphed. angle is
the angle at which the perspective plot is to
be made. */
/* These variables delimit the region
occupied by the plate */
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 */

```

```

1973 double l1,l2,l0,delta; /* These are the area coordinates: L1,L2, and L3,
1974 see, for example Zienkiewicz, The Finite Element
1975 Method for an explanation. delta is the area of
1976 the triangle and l1, l2, and l0 are the area
1977 coordinates associated with points 1, 2, and 0,
1978 respectively. */
1979 int i,j,ix,iy; /* loop variables. ix and iy are used specifically to index
1980 para */
1981 if (potflag == 3) for (i=0;i<mnumplo;i++) point.poten[i] = -point.poten[i];
1982 mirx = (i(plotsym & 1)) ? 0 : ((potflag == 1) ? -1 : 1);
1983 miry = (i(plotsym & 2)) ? 0 : ((potflag == 2) ? -1 : 1);
1984 zmax = -1e20;
1985 zmin = 1e20;
1986 xmax = -1e20;
1987 xmin = 1e20;
1988 ymax = -1e20;
1989 ymin = 1e20;
1990 for (i=0; i < mnumplo; i++) {
1991     if (zmax < point.poten[i]) zmax = point.poten[i];
1992     if (zmin > point.poten[i]) zmin = point.poten[i];
1993     if (xmax < point.x[i]) xmax = point.x[i];
1994     if (xmin > point.x[i]) xmin = point.x[i];
1995     if (ymax < point.y[i]) ymax = point.y[i];
1996     if (ymin > point.y[i]) ymin = point.y[i];
1997 }
1998 zscrat=(fabs(zmax) > fabs(zmin)) ? fabs(zmax) : fabs(zmin);
1999 if (mirx == -1 || miry == -1) {
2000     zoffset = zscrat;
2001     zpmax = 2*zoffset;
2002 }
2003 else {
2004     zoffset = (zmin>0)?0:-zmin;
2005     zpmax = zoffset+zmax;
2006 }
2007 for (i=0; i < Mnumplo; i++) {
2008     for (j=0; j < Mnumplo; j++) {
2009         Para(i,j) = zoffset;
2010     }
2011 }
2012 for (i=0; i<mnumtri; i++) {
2013     for (txmin=tymin=1e20,txmax=tymax= -1e20,j=0; j<3; j++) {
2014         if (txmin > point.x[triang.Poi(i,j)])
2015             txmin = point.x[triang.Poi(i,j)];
2016         if (tymin > point.y[triang.Poi(i,j)])

```

```
2017     tymin = point.y[triang.Poi(1,j)];
2018     if (txmax < point.x[triang.Poi(1,j)])
2019         txmax = point.x[triang.Poi(1,j)];
2020     if (tymax < point.y[triang.Poi(1,j)])
2021         tymax = point.y[triang.Poi(1,j)];
2022 }
2023 zscrat= X1-X0;
2024 if (fabs(zscrat) > Epsilon) {
2025     m01 = (Y1-Y0)/zscrat;
2026     my01 = 1;
2027 }
2028 else {
2029     m01 = 1;
2030     my01 = 0;
2031 }
2032 zscrat = X2-X1;
2033 if (fabs(zscrat) > Epsilon) {
2034     m12 = (Y2-Y1)/zscrat;
2035     my12 = 1;
2036 }
2037 else {
2038     m12 = 1;
2039     my12 = 0;
2040 }
2041 zscrat = X0-X2;
2042 if (fabs(zscrat) > Epsilon) {
2043     m20 = (Y0-Y2)/zscrat;
2044     my20 = 1;
2045 }
2046 else {
2047     m20 = 1;
2048     my20 = 0;
2049 }
2050 b01 = my01*Y0-m01*X0;
2051 b12 = my12*Y1-m12*X1;
2052 b20 = my20*Y2-m20*X2;
2053 mby01 = m01*triang.centroid.x[i]+b01-my01*triang.centroid.y[i];
2054 mby12 = m12*triang.centroid.x[i]+b12-my12*triang.centroid.y[i];
2055 mby20 = m20*triang.centroid.x[i]+b20-my20*triang.centroid.y[i];
2056 delta = area(X0,Y0,X1,Y1,X2,Y2);
2057 txmin *= (txmin < 0)? (1+Epsilon) : (1-Epsilon);
2058 tymin *= (tymin < 0)? (1+Epsilon) : (1-Epsilon);
2059 txmax *= (txmax > 0)? (1+Epsilon) : (1-Epsilon);
2060 tymax *= (tymax > 0)? (1+Epsilon) : (1-Epsilon);
```

```

2061 ixmin = (mirx)? ceil((txmin-xmin)*(Mnumplo-1)/2/(xmax-xmin)):
2062   ceil((txmin-xmin)*(Mnumplo-1)/(xmax-xmin));
2063 ixmax = (mirx)? floor((txmax-xmin)*(Mnumplo-1)/2/(xmax-xmin)):
2064   floor((txmax-xmin)*(Mnumplo-1)/(xmax-xmin));
2065 iymin = (miry)? ceil((tymin-ymin)*(Mnumplo-1)/2/(ymax-ymin)):
2066   ceil((tymin-ymin)*(Mnumplo-1)/(ymax-ymin));
2067 iymax = (miry)? floor((tymax-ymin)*(Mnumplo-1)/2/(ymax-ymin)):
2068   floor((tymax-ymin)*(Mnumplo-1)/(ymax-ymin));
2069 for (ix=ixmin; ix <= ixmax; ix++) {
2070   xi = (mirx)? (ix*(xmax-xmin)*2/(Mnumplo-1)+xmin):
2071     (ix*(xmax-xmin)/(Mnumplo-1)+xmin);
2072   for (iy=iymin; iy <= iymax; iy++) {
2073     yi = (miry)? (iy*(ymax-ymin)*2/(Mnumplo-1)+ymin):
2074       (iy*(ymax-ymin)/(Mnumplo-1)+ymin);
2075     if (Para(ix,iy) != zofset ||
2076         mby01*(m01*xi+b01-my01*yi) < -Epsilon ||
2077         mby12*(m12*xi+b12-my12*yi) < -Epsilon ||
2078         mby20*(m20*xi+b20-my20*yi) < -Epsilon ) continue;
2079     l0 = area(xi, yi, X1, Y1, X2, Y2)/delta;
2080     l1 = area(X0, Y0, xi, yi, X2, Y2)/delta;
2081     l2 = area(X0, Y0, X1, Y1, xi, yi)/delta;
2082     Para(ix,iy) = l0*point.poten[triang.Poi(1,0)]
2083       + l1*point.poten[triang.Poi(1,1)]
2084       + l2*point.poten[triang.Poi(1,2)]
2085       + zofset;
2086   }
2087 }
2088
2089 if (mirx) {
2090   for (ix=(Mnumplo-1)/2; ix >= 0; ix--)
2091     for (iy=0; iy<Mnumplo:iy++)
2092       Para(ix+(Mnumplo-1)/2,iy)=Para(ix,iy);
2093   for (ix=0; ix<(Mnumplo-1)/2:ix++)
2094     for (iy=0; iy<Mnumplo:iy++)
2095       Para(ix,iy) = (mirx == 1)? Para(Mnumplo-1-ix,iy):
2096         -Para(Mnumplo-1-ix,iy)+2*zofset;
2097 }
2098
2099 if (miry) {
2100   for (iy=(Mnumplo-1)/2; iy >= 0; iy--)
2101     for (ix=0; ix<Mnumplo:ix++)
2102       Para(ix,iy+(Mnumplo-1)/2)=Para(ix,iy);
2103   for (iy=0; iy<(Mnumplo-1)/2:iy++)
2104     for (ix=0; ix<Mnumplo:ix++)
2105       Para(ix,iy) = (miry == 1)? Para(ix,Mnumplo-1-iy):

```

```

2105 }
2106 }
2107 }
2108 }
2109 }
2110 }
2111 }
2112 }
2113 }
2114 }
2115 }
2116 }
2117 }
2118 }
2119 }
2120 }
2121 }
2122 }
2123 }
2124 }
2125 }
2126 }
2127 }
2128 }
2129 }
2130 }
2131 }
2132 }
2133 }
2134 }
2135 }
2136 }
2137 }
2138 }
2139 }
2140 }
2141 }
2142 }
2143 }
2144 }
2145 }
2146 }
2147 }
2148 }

-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) /* These are the coordinates of the three
double x1,y1,x2,y2,x3,y3; /* points which delimit the triangle */
{
return(fabs(x2*y3-y2*x3-x1*y3+x1*y2-y1*x2)/2);
}
/*****
ROUTINES: DSGSURF and DGZPLT
These routines are proprietary and source listings are not available.
/*****
C NAASA 2.1.028 DGECC FTN-A 05-02-78 THE UNIV OF MICH COMP CTR
SUBROUTINE DGECC(A,LDA,N,IPVT,RCOND,Z)
INTEGER LDA,N,IPVT(1)
DOUBLE PRECISION A(LDA,1),Z(1)
DOUBLE PRECISION RCOND
C
DGECC FACTORS A DOUBLE PRECISION MATRIX BY GAUSSIAN ELIMINATION
AND ESTIMATES THE CONDITION OF THE MATRIX.
IF RCOND IS NOT NEEDED, DGEFFA IS SLIGHTLY FASTER.
TO SOLVE A*X = B , FOLLOW DGECC BY DGEFL.
TO COMPUTE INVERSE(A)*C , FOLLOW DGECC BY DGEFL.
TO COMPUTE DETERMINANT(A) , FOLLOW DGECC BY DGEFL.
TO COMPUTE INVERSE(A) , FOLLOW DGECC BY DGEFL.
ON ENTRY
A DOUBLE PRECISION(LDA, N)

```

2149 C THE MATRIX TO BE FACTORED.
2150 C
2151 C LDA INTEGER
2152 C THE LEADING DIMENSION OF THE ARRAY A .
2153 C
2154 C N INTEGER
2155 C THE ORDER OF THE MATRIX A .
2156 C
2157 C ON RETURN
2158 C
2159 C A AN UPPER TRIANGULAR MATRIX AND THE MULTIPLIERS
2160 C WHICH WERE USED TO OBTAIN IT.
2161 C THE FACTORIZATION CAN BE WRITTEN $A = L*U$ WHERE
2162 C L IS A PRODUCT OF PERMUTATION AND UNIT LOWER
2163 C TRIANGULAR MATRICES AND U IS UPPER TRIANGULAR.
2164 C
2165 C IPVT INTEGER(N)
2166 C AN INTEGER VECTOR OF PIVOT INDICES.
2167 C
2168 C RCOND DOUBLE PRECISION
2169 C AN ESTIMATE OF THE RECIPROCAL CONDITION OF A .
2170 C FOR THE SYSTEM $A*X = B$, RELATIVE PERTURBATIONS
2171 C IN A AND B OF SIZE EPSILON MAY CAUSE
2172 C RELATIVE PERTURBATIONS IN X OF SIZE EPSILON/RCOND .
2173 C IF RCOND IS SO SMALL THAT THE LOGICAL EXPRESSION
2174 C $1.0 + RCOND .EQ. 1.0$
2175 C IS TRUE, THEN A MAY BE SINGULAR TO WORKING
2176 C PRECISION. IN PARTICULAR, RCOND IS ZERO IF
2177 C EXACT SINGULARITY IS DETECTED OR THE ESTIMATE
2178 C UNDERFLOWS.
2179 C
2180 C Z DOUBLE PRECISION(N)
2181 C A WORK VECTOR WHOSE CONTENTS ARE USUALLY UNIMPORTANT .
2182 C IF A IS CLOSE TO A SINGULAR MATRIX, THEN Z IS
2183 C AN APPROXIMATE NULL VECTOR IN THE SENSE THAT
2184 C $NORM(A*Z) = RCOND*NORM(A)*NORM(Z)$.
2185 C
2186 C LINPACK. THIS VERSION DATED 07/14/77 .
2187 C CLEVE MOLEK, UNIVERSITY OF NEW MEXICO, ARGONNE NATIONAL LABS.
2188 C
2189 C SUBROUTINES AND FUNCTIONS
2190 C
2191 C LINPACK DGEFA
2192 C BLAS DAXPY, DDOT, DSCAL, DASUM

```

2193 C FORTRAN DABS, DMAX1, DSIGN
2194 C
2195 C INTERNAL VARIABLES
2196 C
2197 C DOUBLE PRECISION DDOT, EK, I, WK, WKM
2198 C DOUBLE PRECISION ANORM, S, DASUM, SM, YNORM
2199 C INTEGER INFO, J, K, KB, KP1, L
2200 C
2201 C DOUBLE PRECISION DSIGN
2202 C
2203 C COMPUTE 1-NORM OF A
2204 C
2205 C ANORM = 0.0D0
2206 C DO 10 J = 1, N
2207 C   ANORM = DMAX1 (ANORM, DASUM(N, A(1, J), 1))
2208 C 10 CONTINUE
2209 C
2210 C FACTOR
2211 C
2212 C CALL DGEFA(A, LDA, N, IPVT, INFO)
2213 C
2214 C RCOND = 1/(NORM(A) * (ESTIMATE OF NORM(INVERSE(A))))
2215 C ESTIMATE = NORM(Z)/NORM(Y) WHERE A*Z = Y AND TRANS(A)*Y = E
2216 C TRANS(A) IS THE TRANSPOSE OF A. THE COMPONENTS OF E ARE
2217 C CHOSEN TO CAUSE MAXIMUM LOCAL GROWTH IN THE ELEMENTS OF W WHERE
2218 C TRANS(U)*W = E. THE VECTORS ARE FREQUENTLY RESCALED TO AVOID
2219 C OVERFLOW.
2220 C
2221 C SOLVE TRANS(U)*W = E
2222 C
2223 C EK = 1.0D0
2224 C DO 20 J = 1, N
2225 C   Z(J) = 0.0D0
2226 C 20 CONTINUE
2227 C DO 100 K = 1, N
2228 C   IF (Z(K) .NE. 0.0D0) EK = DSIGN(EK, -Z(K))
2229 C   IF (DABS(EK-Z(K)) .LE. DABS(A(K,K))) GO TO 30
2230 C   S = DABS(A(K,K))/DABS(EK-Z(K))
2231 C   CALL DSCAL(N, S, Z, 1)
2232 C   EK = S*EK
2233 C 30 CONTINUE
2234 C WK = EK - Z(K)
2235 C WKM = -EK - Z(K)
2236 C S = DABS(WK)

```



```
2237 SM = DABS(WKM)
2238 IF (A(K,K) .EQ. 0.0D0) GO TO 40
2239 WK = WK/A(K,K)
2240 WKM = WKM/A(K,K)
2241 GO TO 50
2242 CONTINUE
2243 WK = 1.0D0
2244 WKM = 1.0D0
2245 CONTINUE
2246 KP1 = K + 1
2247 IF (KP1 .GT. N) GO TO 90
2248 DO 60 J = KP1, N
2249 SM = SM + DABS(Z(J)+WKM*A(K,J))
2250 Z(J) = Z(J) + WK*A(K,J)
2251 S = S + DABS(Z(J))
2252 CONTINUE
2253 IF (S .GE. SM) GO TO 80
2254 T = WKM - WK
2255 WK = WKM
2256 DO 70 J = KP1, N
2257 Z(J) = Z(J) + T*A(K,J)
2258 CONTINUE
2259 CONTINUE
2260 Z(K) = WK
2261 CONTINUE
2262 S = 1.0D0/DASUM(N,Z,1)
2263 CALL DSCAL(N,S,Z,1)
2264 C
2265 C SOLVE TRANS(L)*Y = V
2266 C
2267 C DO 120 KB = 1, N
2268 K = N + 1 - KB
2269 IF (K .LT. N) Z(K) = Z(K) + DDOT(N-K,A(K+1,K),1,Z(K+1),1)
2270 IF (DABS(Z(K)) .LE. 1.0D0) GO TO 110
2271 S = 1.0D0/DABS(Z(K))
2272 CALL DSCAL(N,S,Z,1)
2273 CONTINUE
2274 L = IPVT(K)
2275 T = Z(L)
2276 Z(L) = Z(K)
2277 Z(K) = T
2278 CONTINUE
2279 S = 1.0D0/DASUM(N,Z,1)
2280
```

```
2281 CALL DSCAL(N,S,Z,1)
2282 C
2283 YNORM = 1.0D0
2284 C
2285 SOLVE L*V = Y
2286 C
2287 DO 140 K = 1, N
2288 L = IPVT(K)
2289 T = Z(L)
2290 Z(L) = Z(K)
2291 Z(K) = T
2292 IF (K.LT. N) CALL DAXPY(N-K,T,A(K+1,K),1,Z(K+1),1)
2293 IF (DABS(Z(K)) .LE. 1.0D0) GO TO 130
2294 S = 1.0D0/DABS(Z(K))
2295 CALL DSCAL(N,S,Z,1)
2296 YNORM = S*YNORM
2297 CONTINUE
2298 130 CONTINUE
2299 140 CONTINUE
2300 S = 1.0D0/DASUM(N,Z,1)
2301 CALL DSCAL(N,S,Z,1)
2302 YNORM = S*YNORM
2303 C
2304 SOLVE U*Z = V
2305 C
2306 DO 160 KB = 1, N
2307 K = N + 1 - KB
2308 IF (DABS(Z(K)) .LE. DABS(A(K,K))) GO TO 150
2309 S = DABS(A(K,K))/DABS(Z(K))
2310 CALL DSCAL(N,S,Z,1)
2311 YNORM = S*YNORM
2312 CONTINUE
2313 IF (A(K,K) .NE. 0.0D0) Z(K) = Z(K)/A(K,K)
2314 IF (A(K,K) .EQ. 0.0D0) Z(K) = 1.0D0
2315 T = -Z(K)
2316 CALL DAXPY(K-1,T,A(1,K),1,Z(1),1)
2317 150 CONTINUE
2318 160 CONTINUE
2319 MAKE ZNORM = 1.0
2320 S = 1.0D0/DASUM(N,Z,1)
2321 CALL DSCAL(N,S,Z,1)
2322 YNORM = S*YNORM
2323 C
2324 IF (ANORM .NE. 0.0D0) RCOND = YNORM/ANORM
2325 IF (ANORM .EQ. 0.0D0) RCOND = 0.0D0
2326 RETURN
```

2325 END
2326 C NAASA 2.1.029 DGEFA FTN-A 05-02-78 THE UNIV OF MICH COMP CTR
2327 SUBROUTINE DGEFA(A,LDA,N,IPVT,INFO)
2328 INTEGER LDA,N,IPVT(1),INFO
2329 DOUBLE PRECISION A(LDA,1)
2330 C
2331 C DGEFA FACTORS A DOUBLE PRECISION MATRIX BY GAUSSIAN ELIMINATION.
2332 C
2333 C DGEFA IS USUALLY CALLED BY DGECC, BUT IT CAN BE CALLED
2334 C DIRECTLY WITH A SAVING IN TIME IF RCOND IS NOT NEEDED.
2335 C (TIME FOR DGECC) = (1 + 9/N)*(TIME FOR DGEFA).
2336 C
2337 C ON ENTRY
2338 C
2339 C A DOUBLE PRECISION(LDA, N)
2340 C THE MATRIX TO BE FACTORED.
2341 C
2342 C LDA INTEGER
2343 C THE LEADING DIMENSION OF THE ARRAY A.
2344 C
2345 C N INTEGER
2346 C THE ORDER OF THE MATRIX A.
2347 C
2348 C ON RETURN
2349 C
2350 C A AN UPPER TRIANGULAR MATRIX AND THE MULTIPLIERS
2351 C WHICH WERE USED TO OBTAIN IT.
2352 C THE FACTORIZATION CAN BE WRITTEN A = L*U WHERE
2353 C L IS A PRODUCT OF PERMUTATION AND UNIT LOWER
2354 C TRIANGULAR MATRICES AND U IS UPPER TRIANGULAR.
2355 C
2356 C IPVT INTEGER(N)
2357 C AN INTEGER VECTOR OF PIVOT INDICES.
2358 C
2359 C INFO INTEGER
2360 C = 0 NORMAL VALUE.
2361 C = K IF U(K,K).EQ. 0.0. THIS IS NOT AN ERROR
2362 C CONDITION FOR THIS SUBROUTINE, BUT IT DOES
2363 C INDICATE THAT DGECC OR DGECC WILL DIVIDE BY ZERO
2364 C IF CALLED. USE RCOND IN DGECC FOR A RELIABLE
2365 C INDICATION OF SINGULARITY.
2366 C
2367 C LINUXPACK. THIS VERSION DATED 07/14/77.
2368 C CLEVE MOLER, UNIVERSITY OF NEW MEXICO, ARGONNE NATIONAL LABS.

```
2369 C SUBROUTINES AND FUNCTIONS
2370 C
2371 C BLAS DAXPY,DSCAL, IDAMAX
2372 C
2373 C INTERNAL VARIABLES
2374 C
2375 C DOUBLE PRECISION T
2376 C INTEGER IDAMAX,J,K,KP1,L,NM1
2377 C
2378 C
2379 C
2380 C GAUSSIAN ELIMINATION WITH PARTIAL PIVOTING
2381 C
2382 C INFO = 0
2383 C NM1 = N - 1
2384 C IF (NM1 .LT. 1) GO TO 70
2385 C DO 60 K = 1, NM1
2386 C KP1 = K + 1
2387 C
2388 C FIND L = PIVOT INDEX
2389 C
2390 C L = IDAMAX(N-K+1,A(K,K),1) + K - 1
2391 C IPVT(K) = L
2392 C
2393 C ZERO PIVOT IMPLIES THIS COLUMN ALREADY TRIANGULARIZED
2394 C
2395 C IF (A(L,K) .EQ. 0.0D0) GO TO 40
2396 C
2397 C INTERCHANGE IF NECESSARY
2398 C
2399 C IF (L .EQ. K) GO TO 10
2400 C T = A(L,K)
2401 C A(L,K) = A(K,K)
2402 C A(K,K) = T
2403 C CONTINUE
2404 C
2405 C COMPUTE MULTIPLIERS
2406 C
2407 C T = -1.0D0/A(K,K)
2408 C CALL DSCAL(N-K,T,A(K+1,K),1)
2409 C
2410 C ROW ELIMINATION WITH COLUMN INDEXING
2411 C
2412 C DO 30 J = KP1, N
```

```
2413 T = A(L,J)
2414 IF (L.EQ. K) GO TO 20
2415 A(L,J) = A(K,J)
2416 A(K,J) = T
2417 CONTINUE
2418 CALL DAXPY(N-K,T,A(K+1,K),1,A(K+1,J),1)
2419 CONTINUE
2420 GO TO 50
2421 CONTINUE
2422 INFO = K
2423 CONTINUE
2424 CONTINUE
2425 CONTINUE
2426 IPVT(N) = N
2427 IF (A(N,N).EQ. 0.0D0) INFO = N
2428 RETURN
2429 END
2430 C NAASA 2.1.030 DGESL FTM-A 05-02-78 THE UNIV OF MICH COMP CTR
2431 SUBROUTINE DGESL(A,LDA,N,IPVT,B,JOB)
2432 INTEGER LDA,N,IPVT(1),JOB
2433 DOUBLE PRECISION A(LDA,1),B(1)
2434
2435 C DGESL SOLVES THE DOUBLE PRECISION SYSTEM
2436 A * X = B OR TRANS(A) * X = B
2437 USING THE FACTORS COMPUTED BY DGECC OR DGEFA.
2438
2439 C ON ENTRY
2440
2441 C A DOUBLE PRECISION(LDA, N)
2442 THE OUTPUT FROM DGECC OR DGEFA.
2443
2444 C LDA INTEGER
2445 THE LEADING DIMENSION OF THE ARRAY A .
2446
2447 C N INTEGER
2448 THE ORDER OF THE MATRIX A .
2449
2450 C IPVT INTEGER(N)
2451 THE PIVOT VECTOR FROM DGECC OR DGEFA.
2452
2453 C B DOUBLE PRECISION(N)
2454 THE RIGHT HAND SIDE VECTOR.
2455
2456 C JOB INTEGER
```

```

2457 C      = 0      TO SOLVE A*X = B
2458 C      = NONZERO TO SOLVE TRANS(A)*X = B WHERE
2459 C      TRANS(A) IS THE TRANSPOSE.
2460 C
2461 C
2462 C
2463 C      B      THE SOLUTION VECTOR X
2464 C
2465 C      ERROR CONDITION
2466 C
2467 C      A DIVISION BY ZERO WILL OCCUR IF THE INPUT FACTOR CONTAINS A
2468 C      ZERO ON THE DIAGONAL. TECHNICALLY THIS INDICATES SINGULARITY
2469 C      BUT IT IS OFTEN CAUSED BY IMPROPER ARGUMENTS OR IMPROPER
2470 C      SETTING OF LDA. IT WILL NOT OCCUR IF THE SUBROUTINES ARE
2471 C      CALLED CORRECTLY AND IF DGECCO HAS SET RCOND .GT. 0.0
2472 C      OR DGEFA HAS SET INFO .EQ. 0
2473 C
2474 C      TO COMPUTE INVERSE(A) * C WHERE C IS A MATRIX
2475 C      WITH P COLUMNS
2476 C      CALL DGECCO(A, LDA, N, IPVT, RCOND, Z)
2477 C      IF (RCOND IS TOO SMALL) GO TO ...
2478 C      DO 10 J = 1, P
2479 C      CALL DGESL(A, LDA, N, IPVT, C(1, J), 0)
2480 C      10 CONTINUE
2481 C
2482 C      LINPACK. THIS VERSION DATED 07/14/77
2483 C      CLEVE MOLER, UNIVERSITY OF NEW MEXICO, ARGONNE NATIONAL LABS.
2484 C
2485 C      SUBROUTINES AND FUNCTIONS
2486 C
2487 C      BLAS DAXPY, DDOT
2488 C
2489 C      INTERNAL VARIABLES
2490 C
2491 C      DOUBLE PRECISION DDOT, T
2492 C      INTEGER K, KB, L, NM1
2493 C
2494 C      NM1 = N - 1
2495 C      IF (JOB .NE. 0) GO TO 50
2496 C
2497 C      JOB = 0, SOLVE A * X = B
2498 C      FIRST SOLVE L*Y = B
2499 C
2500 C      IF (NM1 .LT. 1) GO TO 30

```

```
2501 DO 20 K = 1, NM1
2502 L = IPVT(K)
2503 T = B(L)
2504 IF (L.EQ. K) GO TO 10
2505 B(L) = B(K)
2506 B(K) = T
2507 CONTINUE
2508 CALL DAXPY(N-K, T, A(K+1, K), 1, B(K+1), 1)
2509 CONTINUE
2510 CONTINUE
2511 C
2512 C
2513 C
2514 DO 40 KB = 1, N
2515 K = N + 1 - KB
2516 B(K) = B(K)/A(K, K)
2517 T = -B(K)
2518 CALL DAXPY(K-1, T, A(1, K), 1, B(1), 1)
2519 CONTINUE
2520 GO TO 100
2521 CONTINUE
2522 C
2523 JOB = NONZERO, SOLVE TRANS(A) * X = B
2524 FIRST SOLVE TRANS(U)*Y = B
2525 C
2526 DO 60 K = 1, N
2527 T = DDOT(K-1, A(1, K), 1, B(1), 1)
2528 B(K) = (B(K) - T)/A(K, K)
2529 CONTINUE
2530 C
2531 C
2532 C
2533 NOW SOLVE TRANS(L)*X = Y
2534 IF (NM1.LT. 1) GO TO 90
2535 DO 80 KB = 1, NM1
2536 K = N - KB
2537 B(K) = B(K) + DDOT(N-K, A(K+1, K), 1, B(K+1), 1)
2538 L = IPVT(K)
2539 IF (L.EQ. K) GO TO 70
2540 T = B(L)
2541 B(L) = B(K)
2542 B(K) = T
2543 CONTINUE
2544 CONTINUE
2545 CONTINUE
2546 CONTINUE
2547 CONTINUE
2548 CONTINUE
2549 CONTINUE
2550 CONTINUE
2551 CONTINUE
2552 CONTINUE
2553 CONTINUE
2554 CONTINUE
2555 CONTINUE
2556 CONTINUE
2557 CONTINUE
2558 CONTINUE
2559 CONTINUE
2560 CONTINUE
2561 CONTINUE
2562 CONTINUE
2563 CONTINUE
2564 CONTINUE
2565 CONTINUE
2566 CONTINUE
2567 CONTINUE
2568 CONTINUE
2569 CONTINUE
2570 CONTINUE
2571 CONTINUE
2572 CONTINUE
2573 CONTINUE
2574 CONTINUE
2575 CONTINUE
2576 CONTINUE
2577 CONTINUE
2578 CONTINUE
2579 CONTINUE
2580 CONTINUE
2581 CONTINUE
2582 CONTINUE
2583 CONTINUE
2584 CONTINUE
2585 CONTINUE
2586 CONTINUE
2587 CONTINUE
2588 CONTINUE
2589 CONTINUE
2590 CONTINUE
2591 CONTINUE
2592 CONTINUE
2593 CONTINUE
2594 CONTINUE
2595 CONTINUE
2596 CONTINUE
2597 CONTINUE
2598 CONTINUE
2599 CONTINUE
2600 CONTINUE
```

```
2545 100 CONTINUE
2546 RETURN
2547 END
2548 C NAASA 1.1.001 DASUM FTN-A 05-02-78 THE UNIV OF MICH COMP CTR
2549 DOUBLE PRECISION FUNCTION DASUM(N,DX,INCX)
2550 C
2551 C TAKES THE SUM OF THE ABSOLUTE VALUES.
2552 C JACK DONGARRA, LINPACK, 6/17/77.
2553 C
2554 C DOUBLE PRECISION DX(1),DTEMP
2555 C INTEGER I,INCX,M,MP1,N,NINCX
2556 C
2557 DASUM = 0.0D0
2558 DTEMP = 0.0D0
2559 IF(N.LE.0)RETURN
2560 IF(INCX.EQ.1)GOTO 20
2561 C
2562 C CODE FOR INCREMENT NOT EQUAL TO 1
2563 C
2564 NINCX = N*INCX
2565 DO 10 I = 1,NINCX,INCX
2566 DTEMP = DTEMP + DABS(DX(I))
2567 10 CONTINUE
2568 DASUM = DTEMP
2569 RETURN
2570 C
2571 C CODE FOR INCREMENT EQUAL TO 1
2572 C
2573 C
2574 C CLEAN-UP LOOP
2575 C
2576 20 M = MOD(N,6)
2577 IF(M.EQ.0) GO TO 40
2578 DO 30 I = 1,M
2579 DTEMP = DTEMP + DABS(DX(I))
2580 30 CONTINUE
2581 IF(N.LI.6) GO TO 60
2582 40 MP1 = M + 1
2583 DO 50 I = MP1,N,6
2584 DTEMP = DTEMP + DABS(DX(I)) + DABS(DX(I + 1)) + DABS(DX(I + 2))
2585 * + DABS(DX(I + 3)) + DABS(DX(I + 4)) + DABS(DX(I + 5))
2586 50 CONTINUE
2587 60 DASUM = DTEMP
2588 RETURN
```



```
2589      END
2590      C  NAASA 1.1.004 DAXPY  FTN-A 05-02-78  THE UNIV OF MICH COMP CTR
2591      SUBROUTINE DAXPY(N,DA,DX,INCX,DY,INCY)
2592      C
2593      C  CONSTANT TIMES A VECTOR PLUS A VECTOR.
2594      C  USES UNROLLED LOOPS FOR INCREMENTS EQUAL TO ONE.
2595      C  JACK DONGARRA, LINPACK, 6/17/77.
2596      C
2597      DOUBLE PRECISION DX(1),DY(1),DA
2598      INTEGER I,INCX,INCY,IXIY,M,MPI,N
2599      C
2600      IF(N.LE.0)RETURN
2601      IF(DA.EQ.0.0D0) RETURN
2602      IF(INCX.EQ.1.AND.INCY.EQ.1)GOTO 20
2603      C
2604      C  CODE FOR UNEQUAL INCREMENTS OR EQUAL INCREMENTS
2605      C  NOT EQUAL TO 1
2606      C
2607      IX = 1
2608      IY = 1
2609      IF(INCX.LT.0)IX = (-N+1)*INCX + 1
2610      IF(INCY.LT.0)IY = (-N+1)*INCY + 1
2611      DO 10 I = 1,N
2612      DY(IY) = DY(IY) + DA*DX(IX)
2613      IX = IX + INCX
2614      IY = IY + INCY
2615      10 CONTINUE
2616      RETURN
2617      C
2618      C  CODE FOR BOTH INCREMENTS EQUAL TO 1
2619      C
2620      C
2621      C  CLEAN-UP LOOP
2622      C
2623      20 M = MOD(N,4)
2624      IF(M.EQ.0) GO TO 40
2625      DO 30 I = 1,M
2626      DY(I) = DY(I) + DA*DX(I)
2627      30 CONTINUE
2628      IF(N.LT.4) RETURN
2629      40 MP1 = M + 1
2630      DO 50 I = MP1,N,4
2631      DY(I) = DY(I) + DA*DX(I)
2632      DY(I + 1) = DY(I + 1) + DA*DX(I + 1)
```

```
2633      DY(I + 2) = DY(I + 2) + DA*DX(I + 2)
2634      DY(I + 3) = DY(I + 3) + DA*DX(I + 3)
2635 50 CONTINUE
2636 RETURN
2637 END
2638 C NAASA 1.1.003 DDOT FTN-A 05-02-78 THE UNIV OF MICH COMP CTR
2639 DOUBLE PRECISION FUNCTION DDOT(N,DX,INCX,DY,INCY)
2640 C
2641 C FORMS THE DOT PRODUCT OF A VECTOR.
2642 C USES UNROLLED LOOPS FOR INCREMENTS EQUAL TO ONE.
2643 C JACK DONGARRA, LINPACK, 6/17/77.
2644 C
2645 C DOUBLE PRECISION DX(1),DY(1),DTEMP
2646 C INTEGER I,INCX,INCY,IX,IY,M,MP1,N
2647 C
2648 DDOT = 0.0D0
2649 DTEMP = 0.0D0
2650 IF(N.LE.0)RETURN
2651 IF(INCX.EQ.1.AND.INCY.EQ.1)GOTO 20
2652 C
2653 C CODE FOR UNEQUAL INCREMENTS OR EQUAL INCREMENTS
2654 C NOT EQUAL TO 1
2655 C
2656 IX = 1
2657 IY = 1
2658 IF(INCX.LT.0)IX = (-N+1)*INCX + 1
2659 IF(INCY.LT.0)IY = (-N+1)*INCY + 1
2660 DO 10 I = 1,N
2661 DTEMP = DTEMP + DX(IX)*DY(IY)
2662 IX = IX + INCX
2663 IY = IY + INCY
2664 10 CONTINUE
2665 DDOT = DTEMP
2666 RETURN
2667 C
2668 C CODE FOR BOTH INCREMENTS EQUAL TO 1
2669 C
2670 C CLEAN-UP LOOP
2671 C
2672 20 M = MOD(N,5)
2673 IF(M.EQ.0)GO TO 40
2674 DO 30 I = 1,M
2675 DTEMP = DTEMP + DX(I)*DY(I)
2676
```

```
2677 30 CONTINUE
2678 IF( N .LT. 5 ) GO TO 60
2679 40 MP1 = M + 1
2680 DO 50 I = MP1,N,5
2681 DTEMP = DTEMP + DX(I)*DY(I) + DX(I + 1)*DY(I + 1) +
2682 * DX(I + 2)*DY(I + 2) + DX(I + 3)*DY(I + 3) + DX(I + 4)*DY(I + 4)
2683 50 CONTINUE
2684 60 DDOT = DTEMP
2685 RETURN
2686 END
2687 C NAASA 1.1.009 DSCAL FTN-A 05-02-78 THE UNIV OF MICH COMP CTR
2688 SUBROUTINE DSCAL(N,DA,DX,INCX)
2689 C
2690 C SCALES A VECTOR BY A CONSTANT.
2691 C USES UNROLLED LOOPS FOR INCREMENT EQUAL TO ONE.
2692 C JACK DONGARRA, LINPACK, 6/17/77.
2693 C
2694 C DOUBLE PRECISION DA,DX(1)
2695 C INTEGER I, INCX,M,MP1,N,NINCX
2696 C
2697 IF(N.LE.0)RETURN
2698 IF(INCX.EQ.1)GOTO 20
2699 C
2700 C CODE FOR INCREMENT NOT EQUAL TO 1
2701 C
2702 NINCX = N*INCX
2703 DO 10 I = 1,NINCX,INCX
2704 DX(I) = DA*DX(I)
2705 10 CONTINUE
2706 RETURN
2707 C
2708 C CODE FOR INCREMENT EQUAL TO 1
2709 C
2710 C CLEAN-UP LOOP
2711 C
2712 C
2713 20 M = MOD(N,5)
2714 IF( M .EQ. 0 ) GO TO 40
2715 DO 30 I = 1,M
2716 DX(I) = DA*DX(I)
2717 30 CONTINUE
2718 IF( N .LT. 5 ) RETURN
2719 40 MP1 = M + 1
2720 DO 50 I = MP1,N,5
```

```
2721 DX(I) = DA*DX(I)
2722 DX(I + 1) = DA*DX(I + 1)
2723 DX(I + 2) = DA*DX(I + 2)
2724 DX(I + 3) = DA*DX(I + 3)
2725 DX(I + 4) = DA*DX(I + 4)
2726 50 CONTINUE
2727 RETURN
2728 END
2729 C NAASA 1.1.020 IDAMAX FTM-A 05-02-78 THE UNIV OF MICH COMP CTR
2730 INTEGER FUNCTION IDAMAX(N,DX,INCX)
2731 C
2732 C FINDS THE INDEX OF ELEMENT HAVING MAX. ABSOLUTE VALUE.
2733 C JACK DONGARRA, LINPACK, 6/17/77.
2734 C
2735 C DOUBLE PRECISION DX(1),DMAX
2736 C INTEGER I, INCX, IX, N
2737 C
2738 C IDAMAX = 1
2739 C IF(N.LE.1)RETURN
2740 C IF(INCX.EQ.1)GOTO 20
2741 C
2742 C CODE FOR INCREMENT NOT EQUAL TO 1
2743 C
2744 C IX = 1
2745 C DMAX = DABS(DX(1))
2746 C IX = IX + INCX
2747 C DO 10 I = 2,N
2748 C IF(DABS(DX(IX)) .LE. DMAX) GO TO 5
2749 C IDAMAX = I
2750 C DMAX = DABS(DX(IX))
2751 C IX = IX + INCX
2752 C 5 CONTINUE
2753 C 10 RETURN
2754 C
2755 C CODE FOR INCREMENT EQUAL TO 1
2756 C
2757 C 20 DMAX = DABS(DX(1))
2758 C DO 30 I = 2,N
2759 C IF(DABS(DX(I)) .LE. DMAX) GO TO 30
2760 C IDAMAX = I
2761 C DMAX = DABS(DX(I))
2762 C 30 CONTINUE
2763 C RETURN
2764 C END
```

```

2765 C NAASA 2.1.042 CGECO FTN-A 05-02-78 THE UNIV OF MICH COMP CTR
2766 SUBROUTINE CGECO(A,LDA,N,IPVT,RCOND,Z)
2767 IMPLICIT REAL*8(A-H,O-Z)
2768 INTEGER LDA,N,IPVT(1)
2769 COMPLEX*16 A(LDA,1),Z(1)
2770 REAL*8 RCOND
2771
2772 C CGECO FACTORS A COMPLEX MATRIX BY GAUSSIAN ELIMINATION
2773 C AND ESTIMATES THE CONDITION OF THE MATRIX.
2774 C
2775 C IF RCOND IS NOT NEEDED, CGEFA IS SLIGHTLY FASTER.
2776 C TO SOLVE A*X = B, FOLLOW CGECO BY CGESL.
2777 C TO COMPUTE INVERSE(A)*C, FOLLOW CGECO BY CGESL.
2778 C TO COMPUTE DETERMINANT(A), FOLLOW CGECO BY CGEDI.
2779 C TO COMPUTE INVERSE(A), FOLLOW CGECO BY CGEDI.
2780 C
2781 C ON ENTRY
2782 C
2783 C A COMPLEX(LDA, N)
2784 C THE MATRIX TO BE FACTORED.
2785 C
2786 C LDA INTEGER
2787 C THE LEADING DIMENSION OF THE ARRAY A .
2788 C
2789 C N INTEGER
2790 C THE ORDER OF THE MATRIX A .
2791 C
2792 C ON RETURN
2793 C
2794 C A AN UPPER TRIANGULAR MATRIX AND THE MULTIPLIERS
2795 C WHICH WERE USED TO OBTAIN IT.
2796 C THE FACTORIZATION CAN BE WRITTEN A = L*U WHERE
2797 C L IS A PRODUCT OF PERMUTATION AND UNIT LOWER
2798 C TRIANGULAR MATRICES AND U IS UPPER TRIANGULAR.
2799 C
2800 C IPVT INTEGER(N)
2801 C AN INTEGER VECTOR OF PIVOT INDICES.
2802 C
2803 C RCOND REAL
2804 C AN ESTIMATE OF THE RECIPROCAL CONDITION OF A .
2805 C FOR THE SYSTEM A*X = B, RELATIVE PERTURBATIONS
2806 C IN A AND B OF SIZE EPSILON MAY CAUSE
2807 C RELATIVE PERTURBATIONS IN X OF SIZE EPSILON/RCOND .
2808 C IF RCOND IS SO SMALL THAT THE LOGICAL EXPRESSION

```

```
2809 C          1.0 + RCOND .EQ. 1.0
2810 C      IS TRUE, THEN A MAY BE SINGULAR TO WORKING
2811 C      PRECISION. IN PARTICULAR, RCOND IS ZERO IF
2812 C      EXACT SINGULARITY IS DETECTED OR THE ESTIMATE
2813 C      UNDERFLOWS.
2814 C
2815 C      Z      COMPLEX(N)
2816 C      A WORK VECTOR WHOSE CONTENTS ARE USUALLY UNIMPORTANT.
2817 C      IF A IS CLOSE TO A SINGULAR MATRIX, THEN Z IS
2818 C      AN APPROXIMATE NULL VECTOR IN THE SENSE THAT
2819 C       $NORM(A*Z) = RCOND * NORM(A) * NORM(Z)$  .
2820 C
2821 C      LINPACK. THIS VERSION DATED 07/14/77 .
2822 C      CLEVE MOLER, UNIVERSITY OF NEW MEXICO, ARGONNE NATIONAL LABS.
2823 C
2824 C      SUBROUTINES AND FUNCTIONS
2825 C
2826 C      LINPACK CGEFA
2827 C      BLAS CAXPY, CDOTC, CSSCAL, SCASUM
2828 C      FORTRAN DABS, DIMAG, DMAX1, DCMPLX, DCONJG, REAL
2829 C
2830 C      INTERNAL VARIABLES
2831 C
2832 C      IMPLICIT REAL*8(A-H,O-Z)
2833 C      COMPLEX*16 CDOTC, EK, T, WK, WKM
2834 C      REAL*8 ANORM, S, SCASUM, SM, YNORM
2835 C      INTEGER INFO, J, K, KB, KP1, L
2836 C
2837 C      COMPLEX*16 ZDUM, ZDUM1, ZDUM2, CSIGN1
2838 C      REAL*8 CABS1
2839 C      CABS1(ZDUM) = DABS(DREAL(ZDUM)) + DABS(DIMAG(ZDUM))
2840 C      CSIGN1(ZDUM1, ZDUM2) = CABS1(ZDUM1) * (ZDUM2 / CABS1(ZDUM2))
2841 C
2842 C      COMPUTE 1-NORM OF A
2843 C
2844 C      ANORM = 0.0D0
2845 C      DO 10 J = 1, N
2846 C          ANORM = DMAX1(ANORM, SCASUM(N, A(1, J), 1))
2847 C      10 CONTINUE
2848 C
2849 C      FACTOR
2850 C
2851 C      CALL CGEFA(A, LDA, N, IPVT, INFO)
2852 C
```

```

2853 C RCOND = 1/(NORM(A)*(ESTIMATE OF NORM(INVERSE(A))))
2854 C ESTIMATE = NORM(Z)/NORM(Y) WHERE A*Z = Y AND CTRANS(A)*Y = E .
2855 C CTRANS(A) IS THE CONJUGATE TRANSPOSE OF A .
2856 C THE COMPONENTS OF E ARE CHOSEN TO CAUSE MAXIMUM LOCAL
2857 C GROWTH IN THE ELEMENTS OF W WHERE CTRANS(U)*W = E .
2858 C THE VECTORS ARE FREQUENTLY RESCALED TO AVOID OVERFLOW.
2859 C
2860 C SOLVE CTRANS(U)*W = E
2861 C
2862 C EK = DCMPLEX(1.0D0,0.0D0)
2863 C DO 20 J = 1, N
2864 C Z(J) = DCMPLEX(0.0D0,0.0D0)
2865 C
2866 C 20 CONTINUE
2867 C DO 100 K = 1, N
2868 C IF (CABS1(Z(K)) .NE. 0.0D0) EK = CSIGN1(EK,-Z(K))
2869 C IF (CABS1(EK-Z(K)) .LE. CABS1(A(K,K))) GO TO 30
2870 C S = CABS1(A(K,K))/CABS1(EK-Z(K))
2871 C CALL CSSCAL(N,S,Z,1)
2872 C EK = DCMPLEX(S,0.0D0)*EK
2873 C CONTINUE
2874 C WK = EK - Z(K)
2875 C WKM = -EK - Z(K)
2876 C S = CABS1(WK)
2877 C SM = CABS1(WKM)
2878 C IF (CABS1(A(K,K)) .EQ. 0.0D0) GO TO 40
2879 C WK = WK/DCONJG(A(K,K))
2880 C WKM = WKM/DCONJG(A(K,K))
2881 C GO TO 50
2882 C CONTINUE
2883 C WK = DCMPLEX(1.0D0,0.0D0)
2884 C WKM = DCMPLEX(1.0D0,0.0D0)
2885 C 50 CONTINUE
2886 C KP1 = K + 1
2887 C IF (KP1 .GT. N) GO TO 90
2888 C DO 60 J = KP1, N
2889 C SM = SM + CABS1(Z(J)+WKM*DCONJG(A(K,J)))
2890 C Z(J) = Z(J) + WK*DCONJG(A(K,J))
2891 C S = S + CABS1(Z(J))
2892 C CONTINUE
2893 C IF (S .GE. SM) GO TO 80
2894 C T = WKM - WK
2895 C WK = WKM
2896 C DO 70 J = KP1, N
2897 C Z(J) = Z(J) + T*DCONJG(A(K,J))

```

```
2897 70 CONTINUE
2898 80 CONTINUE
2899 90 CONTINUE
2900 Z(K) = WK
2901 100 CONTINUE
2902 S = 1.0D0/SCASUM(N,Z,1)
2903 CALL CSSCAL(N,S,Z,1)
2904 C
2905 C SOLVE CTRANS(L)*Y = V
2906 C
2907 DO 120 KB = 1, N
2908 K = N + 1 - KB
2909 IF (K .LT. N) Z(K) = Z(K) + CDOTC(N-K,A(K+1,K),1,Z(K+1),1)
2910 IF (CABS1(Z(K)) .LE. 1.0D0) GO TO 110
2911 S = 1.0D0/CABS1(Z(K))
2912 CALL CSSCAL(N,S,Z,1)
2913 110 CONTINUE
2914 L = IPVT(K)
2915 T = Z(L)
2916 Z(L) = Z(K)
2917 Z(K) = T
2918 120 CONTINUE
2919 S = 1.0D0/SCASUM(N,Z,1)
2920 CALL CSSCAL(N,S,Z,1)
2921 C
2922 YNORM = 1.0D0
2923 C
2924 C SOLVE L*V = Y
2925 C
2926 DO 140 K = 1, N
2927 L = IPVT(K)
2928 T = Z(L)
2929 Z(L) = Z(K)
2930 Z(K) = T
2931 IF (K .LT. N) CALL CAXPY(N-K,T,A(K+1,K),1,Z(K+1),1)
2932 IF (CABS1(Z(K)) .LE. 1.0D0) GO TO 130
2933 S = 1.0D0/CABS1(Z(K))
2934 CALL CSSCAL(N,S,Z,1)
2935 YNORM = S*YNORM
2936 130 CONTINUE
2937 140 CONTINUE
2938 S = 1.0D0/SCASUM(N,Z,1)
2939 CALL CSSCAL(N,S,Z,1)
2940 YNORM = S*YNORM
```



```
2941 C SOLVE U*Z = V
2942 C
2943 C
2944 DO 160 KB = 1, N
2945 K = N + 1 - KB
2946 IF (CABS1(Z(K)) .LE. CABS1(A(K,K))) GO TO 150
2947 S = CABS1(A(K,K))/CABS1(Z(K))
2948 CALL CSSCAL(N,S,Z,1)
2949 YNORM = S*YNORM
2950 CONTINUE
2951 IF (CABS1(A(K,K)) .NE. 0.0D0) Z(K) = Z(K)/A(K,K)
2952 IF (CABS1(A(K,K)) .EQ. 0.0D0) Z(K) = DCMPLEX(1.0D0, 0.0D0)
2953 T = -Z(K)
2954 CALL CAXPY(K-1, T, A(1,K), 1, Z(1), 1)
2955 CONTINUE
2956 MAKE ZNORM = 1.0
2957 S = 1.0D0/SCASUM(N,Z,1)
2958 CALL CSSCAL(N,S,Z,1)
2959 YNORM = S*YNORM
2960 C
2961 IF (ANORM .NE. 0.0D0) RCOND = YNORM/ANORM
2962 IF (ANORM .EQ. 0.0D0) RCOND = 0.0D0
2963 RETURN
2964 END
2965 C NAASA 2.1.044 CGESL FTN-A 05-02-78 THE UNIV OF MICH COMP CTR
2966 SUBROUTINE CGESL(A,LDA,N,IPVT,B,JOB)
2967 IMPLICIT REAL*8(A-H,O-Z)
2968 INTEGER LDA,N,IPVT(1),JOB
2969 COMPLEX*16 A(LDA,1),B(1)
2970 C
2971 CGESL SOLVES THE COMPLEX SYSTEM
2972 A * X = B OR CTRANS(A) * X = B
2973 USING THE FACTORS COMPUTED BY CGECO OR CGEFA.
2974 C
2975 ON ENTRY
2976 C
2977 A COMPLEX(LDA, N)
2978 THE OUTPUT FROM CGECO OR CGEFA.
2979 C
2980 LDA INTEGER
2981 THE LEADING DIMENSION OF THE ARRAY A .
2982 C
2983 N INTEGER
2984 THE ORDER OF THE MATRIX A .
C
```

```

2985 C
2986 C
2987 C
2988 C
2989 C
2990 C
2991 C
2992 C
2993 C
2994 C
2995 C
2996 C
2997 C
2998 C
2999 C
3000 C
3001 C
3002 C
3003 C
3004 C
3005 C
3006 C
3007 C
3008 C
3009 C
3010 C
3011 C
3012 C
3013 C
3014 C
3015 C
3016 C
3017 C
3018 C
3019 C
3020 C
3021 C
3022 C
3023 C
3024 C
3025 C
3026 C
3027 C
3028 C

IPVT      INTEGER(N)
          THE PIVOT VECTOR FROM CGECO OR CGEFA.

B         COMPLEX(N)
          THE RIGHT HAND SIDE VECTOR.

JOB       INTEGER
          = 0      TO SOLVE  $A \cdot X = B$ 
          = NONZERO TO SOLVE  $C \cdot TRANS(A) \cdot X = B$  WHERE
          CTRANS(A) IS THE CONJUGATE TRANSPOSE.

ON RETURN

B         THE SOLUTION VECTOR X.

ERROR CONDITION

A DIVISION BY ZERO WILL OCCUR IF THE INPUT FACTOR CONTAINS A
ZERO ON THE DIAGONAL.  TECHNICALLY THIS INDICATES SINGULARITY
BUT IT IS OFTEN CAUSED BY IMPROPER ARGUMENTS OR IMPROPER
SETTING OF LDA.  IT WILL NOT OCCUR IF THE SUBROUTINES ARE
CALLED CORRECTLY AND IF CGECO HAS SET RCOND .GT. 0.0
OR CGEFA HAS SET INFO .EQ. 0.

      TO COMPUTE INVERSE(A) * C WHERE C IS A MATRIX
      WITH P COLUMNS
      CALL CGECO(A,LDA,N,IPVT,RCOND,Z)
      IF (RCOND IS TOO SMALL) GO TO ...
      DO 10 J = 1, P
         CALL CGESL(A,LDA,N,IPVT,C(1,J),0)
      10 CONTINUE

LINPACK. THIS VERSION DATED 07/14/77.
CLEVE MOLER, UNIVERSITY OF NEW MEXICO, ARGONNE NATIONAL LABS.

SUBROUTINES AND FUNCTIONS

BLAS CAXPY,CDDTC
FORTRAN DCONJG

INTERNAL VARIABLES

COMPLEX*16 CDDTC,I

```

```
3029 C INTEGER K,KB,L,NM1
3030 C
3031 NM1 = N - 1
3032 IF (JOB .NE. 0) GO TO 50
3033 C
3034 C JOB = 0 , SOLVE A * X = B
3035 C FIRST SOLVE L*Y = B
3036 C
3037 IF (NM1 .LT. 1) GO TO 30
3038 DO 20 K = 1, NM1
3039 L = IPVT(K)
3040 T = B(L)
3041 IF (L .EQ. K) GO TO 10
3042 B(L) = B(K)
3043 B(K) = T
3044 CONTINUE
3045 CALL CAXPY(N-K,T,A(K+1,K),1,B(K+1),1)
3046 CONTINUE
3047 C
3048 C NOW SOLVE U*X = Y
3049 C
3050 C
3051 DO 40 KB = 1, N
3052 K = N + 1 - KB
3053 B(K) = B(K)/A(K,K)
3054 T = -B(K)
3055 CALL CAXPY(K-1,T,A(1,K),1,B(1),1)
3056 CONTINUE
3057 GO TO 100
3058 C
3059 C JOB = NONZERO, SOLVE CTRANS(A) * X = B
3060 C FIRST SOLVE CTRANS(U)*Y = B
3061 C
3062 C
3063 DO 60 K = 1, N
3064 T = CDOTC(K-1,A(1,K),1,B(1),1)
3065 B(K) = (B(K) - T)/DCONJG(A(K,K))
3066 CONTINUE
3067 C
3068 C NOW SOLVE CTRANS(L)*X = Y
3069 C
3070 IF (NM1 .LT. 1) GO TO 90
3071 DO 80 KB = 1, NM1
3072 K = N - KB
```

```

3073 B(K) = B(K) + CDDTC(N-K,A(K+1,K),1,B(K+1),1)
3074 L = IPVT(K)
3075 IF (L.EQ.K) GO TO 70
3076 T = B(L)
3077 B(L) = B(K)
3078 B(K) = T
3079 CONTINUE
3080 CONTINUE
3081 CONTINUE
3082 CONTINUE
3083 RETURN
3084 END
3085 C NAASA 2.1.043 CGEFA FTN-A 05-02-78 THE UNIV OF MICH COMP CTR
3086 SUBROUTINE CGEFA(A,LDA,N,IPVT,INFO)
3087 IMPLICIT REAL*8(A-H,O-Z)
3088 INTEGER LDA,N,IPVT(1),INFO
3089 COMPLEX*16 A(LDA,1)
3090
3091 C
3092 C
3093 C CGEFA IS USUALLY CALLED BY CGECO, BUT IT CAN BE CALLED
3094 C DIRECTLY WITH A SAVING IN TIME IF RCOND IS NOT NEEDED.
3095 C (TIME FOR CGECO) = (1 + 9/N)*(TIME FOR CGEFA)
3096 C
3097 C ON ENTRY
3098 C
3099 C A COMPLEX(LDA, N)
3100 C THE MATRIX TO BE FACTORED.
3101 C
3102 C LDA INTEGER
3103 C THE LEADING DIMENSION OF THE ARRAY A
3104 C
3105 C N INTEGER
3106 C THE ORDER OF THE MATRIX A
3107 C
3108 C ON RETURN
3109 C
3110 C A AN UPPER TRIANGULAR MATRIX AND THE MULTIPLIERS
3111 C WHICH WERE USED TO OBTAIN IT.
3112 C THE FACTORIZATION CAN BE WRITTEN A = L*U WHERE
3113 C L IS A PRODUCT OF PERMUTATION AND UNIT LOWER
3114 C TRIANGULAR MATRICES AND U IS UPPER TRIANGULAR.
3115 C
3116 C IPVT INTEGER(N)

```

```

3117 C AN INTEGER VECTOR OF PIVOT INDICES.
3118 C
3119 C INTEGER
3120 C = 0 NORMAL VALUE.
3121 C = K IF U(K,K) .EQ. 0.0 . THIS IS NOT AN ERROR
3122 C CONDITION FOR THIS SUBROUTINE, BUT IT DOES
3123 C INDICATE THAT CGESL OR CGEDI WILL DIVIDE BY ZERO
3124 C IF CALLED. USE RCOND IN CGECO FOR A RELIABLE
3125 C INDICATION OF SINGULARITY.
3126 C
3127 C LINPACK. THIS VERSION DATED 07/14/77 .
3128 C CLEVE MOLER, UNIVERSITY OF NEW MEXICO, ARGONNE NATIONAL LABS.
3129 C
3130 C SUBROUTINES AND FUNCTIONS
3131 C
3132 C BLAS CAXPY, CSCAL, ICAMAX
3133 C FORTRAN DABS, DIMAG, DCMPLX, DREAL
3134 C
3135 C INTERNAL VARIABLES
3136 C
3137 C COMPLEX*16 T
3138 C INTEGER ICAMAX, J,K,KP1,L,NM1
3139 C
3140 C COMPLEX*16 ZDUM
3141 C REAL*8 CABS1
3142 C CABS1(ZDUM) = DABS(DREAL(ZDUM)) + DABS(DIMAG(ZDUM))
3143 C
3144 C GAUSSIAN ELIMINATION WITH PARTIAL PIVOTING
3145 C
3146 C INFO = 0
3147 C NM1 = N - 1
3148 C IF (NM1 .LT. 1) GO TO 70
3149 C DO 60 K = 1, NM1
3150 C KP1 = K + 1
3151 C
3152 C FIND L = PIVOT INDEX
3153 C
3154 C L = ICAMAX(N-K+1,A(K,K),1) + K - 1
3155 C IPVT(K) = L
3156 C
3157 C ZERO PIVOT IMPLIES THIS COLUMN ALREADY TRIANGULARIZED
3158 C
3159 C IF (CABS1(A(L,K)) .EQ. 0.0D0) GO TO 40
3160 C

```

```

3161 C INTERCHANGE IF NECESSARY
3162 C
3163 IF (L .EQ. K) GO TO 10
3164 T = A(L,K)
3165 A(L,K) = A(K,K)
3166 A(K,K) = T
3167 CONTINUE
3168 C
3169 C COMPUTE MULTIPLIERS
3170 C
3171 T = -DCMPLX(1.0D0,0.0D0)/A(K,K)
3172 CALL CSCAL(N-K,T,A(K+1,K),1)
3173 C
3174 C ROW ELIMINATION WITH COLUMN INDEXING
3175 C
3176 DO 30 J = KP1, N
3177 T = A(L,J)
3178 IF (L .EQ. K) GO TO 20
3179 A(L,J) = A(K,J)
3180 A(K,J) = T
3181 CONTINUE
3182 CALL CAXPY(N-K,T,A(K+1,K),1,A(K+1,J),1)
3183 CONTINUE
3184 GO TO 50
3185 CONTINUE
3186 INFO = K
3187 CONTINUE
3188 CONTINUE
3189 CONTINUE
3190 IPVT(N) = N
3191 IF (CABS1(A(N,N)) .EQ. 0.0D0) INFO = N
3192 RETURN
3193 END
3194 C NAASA 1.1.014 CAXPY FTN-A 05-02-78 THE UNIV OF MICH COMP CTR
3195 SUBROUTINE CAXPY(N,CA,CX,INCX,CY,INCY)
3196 C
3197 C CONSTANT TIMES A VECTOR PLUS A VECTOR.
3198 C JACK DONGARRA, LINPACK, 6/17/77.
3199 C
3200 C IMPLICIT REAL*8(A-H,O-Z)
3201 C COMPLEX*16 CX(1),CY(1),CA
3202 C INTEGER I, INCX, INCY, IX, IY, N
3203 C
3204 C IF (N.LE.0) RETURN

```

```

3205 IF (DABS(DREAL(CA)) + DABS(DIMAG(CA)) .EQ. 0.0D0 ) RETURN
3206 IF (INCX.EQ.1.AND.INCY.EQ.1)GOTO 20
3207 C
3208 C CODE FOR UNEQUAL INCREMENTS OR EQUAL INCREMENTS
3209 C NOT EQUAL TO 1
3210 C
3211 IX = 1
3212 IY = 1
3213 IF (INCX.LT.0)IX = (-N+1)*INCX + 1
3214 IF (INCY.LT.0)IY = (-N+1)*INCY + 1
3215 DO 10 I = 1,N
3216 CY(IY) = CY(IY) + CA*CX(IX)
3217 IX = IX + INCX
3218 IY = IY + INCY
3219
3220 10 CONTINUE
3221 RETURN
3222 C
3223 C CODE FOR BOTH INCREMENTS EQUAL TO 1
3224 C
3225 20 DO 30 I = 1,N
3226 CY(I) = CY(I) + CA*CX(I)
3227 30 CONTINUE
3228 RETURN
3229 END
3229 C NAASA 1.1.012 CDOTC FTN-A 05-02-78 THE UNIV OF MICH COMP CTR
3230 C COMPLEX*16 FUNCTION CDOTC(N,CX,INCX,CY,INCY)
3231 C
3232 C FORMS THE DOT PRODUCT OF TWO VECTORS, CONJUGATING THE FIRST
3233 C VECTOR.
3234 C JACK DONGARRA, LINPACK, 6/17/77.
3235 C
3236 C IMPLICIT REAL*8(A-H,O-Z)
3237 C COMPLEX*16 CX(1),CY(1),CTEMP
3238 C INTEGER I,INCX,INCY,IX,IY,N
3239 C
3240 CTEMP = DCMLX(0.0D0,0.0D0)
3241 CDOTC = DCMLX(0.0D0,0.0D0)
3242 IF (N.LE.0)RETURN
3243 IF (INCX.EQ.1.AND.INCY.EQ.1)GOTO 20
3244 C
3245 C CODE FOR UNEQUAL INCREMENTS OR EQUAL INCREMENTS
3246 C NOT EQUAL TO 1
3247 C
3248 IX = 1

```

```
3249      IY = 1
3250      IF(INCX.LT.0) IX = (-N+1)*INCX + 1
3251      IF(INCY.LT.0) IY = (-N+1)*INCY + 1
3252      DO 10 I = 1,N
3253          CTEMP = CTEMP + DCONJG(CX(IX))*CY(IY)
3254          IX = IX + INCX
3255          IY = IY + INCY
3256      10 CONTINUE
3257      CDOTC = CTEMP
3258      RETURN
3259      C
3260      C      CODE FOR BOTH INCREMENTS EQUAL TO 1
3261      C
3262      20 DO 30 I = 1,N
3263          CTEMP = CTEMP + DCONJG(CX(I))*CY(I)
3264      30 CONTINUE
3265      CDOTC = CTEMP
3266      RETURN
3267      END
3268      C NAASA 1.1.018 CSSCAL FTN-A 05-02-78 THE UNIV OF MICH COMP CTR
3269      SUBROUTINE CSSCAL(N,SA,CX,INCX)
3270      C
3271      C      SCALES A COMPLEX VECTOR BY A REAL CONSTANT.
3272      C      JACK DONGARRA, LINPACK, 6/17/77.
3273      C
3274      IMPLICIT REAL*8(A-H,O-Z)
3275      COMPLEX*16 CX(1)
3276      REAL*8 SA
3277      INTEGER I,INCX,N,NINCX
3278      C
3279      IF(N.LE.0) RETURN
3280      IF(INCX.EQ.1) GOTO 20
3281      C
3282      C      CODE FOR INCREMENT NOT EQUAL TO 1
3283      C
3284      NINCX = N*INCX
3285      DO 10 I = 1,NINCX,INCX
3286          CX(I) = DCMLX(SA*DREAL(CX(I)),SA*DIMAG(CX(I)))
3287      10 CONTINUE
3288      RETURN
3289      C
3290      C      CODE FOR INCREMENT EQUAL TO 1
3291      C
3292      20 DO 30 I = 1,N
```



```
3293 CX(I) = DCMPLX(SA*DREAL(CX(I)), SA*DIMAG(CX(I)))
3294 CONTINUE
3295 RETURN
3296 END
3297 C NAASA 1.1.010 SCASUM FTN-A 05-02-78 THE UNIV OF MICH COMP CTR
3298 REAL*8 FUNCTION SCASUM(N,CX,INCX)
3299 C
3300 C TAKES THE SUM OF THE ABSOLUTE VALUES OF A COMPLEX VECTOR AND
3301 C RETURNS A SINGLE PRECISION RESULT.
3302 C JACK DONGARRA, LINPACK, 6/17/77.
3303 C
3304 C IMPLICIT REAL*8(A-H,O-Z)
3305 C COMPLEX*16 CX(1)
3306 C REAL*8 STEMP
3307 C INTEGER I,INCX,N,NINCX
3308 C
3309 C SCASUM = 0.0D0
3310 C STEMP = 0.0D0
3311 C IF(N.LE.0)RETURN
3312 C IF(INCX.EQ.1)GOTO 20
3313 C
3314 C CODE FOR INCREMENT NOT EQUAL TO 1
3315 C
3316 C NINCX = N*INCX
3317 C DO 10 I = 1,NINCX,INCX
3318 C STEMP = STEMP + DABS(DREAL(CX(I))) + DABS(DIMAG(CX(I)))
3319 C
3320 C SCASUM = STEMP
3321 C RETURN
3322 C
3323 C CODE FOR INCREMENT EQUAL TO 1
3324 C
3325 C DO 30 I = 1,N
3326 C STEMP = STEMP + DABS(DREAL(CX(I))) + DABS(DIMAG(CX(I)))
3327 C
3328 C SCASUM = STEMP
3329 C RETURN
3330 C END
3331 C NAASA 1.1.019 CSCAL FTN-A 05-02-78 THE UNIV OF MICH COMP CTR
3332 C SUBROUTINE CSCAL(N,CA,CX,INCX)
3333 C
3334 C SCALES A VECTOR BY A CONSTANT.
3335 C JACK DONGARRA, LINPACK, 6/17/77.
3336 C
```

```

3337      IMPLICIT REAL*8 (A-H,O-Z)
3338      COMPLEX*16 CA,CX(1)
3339      INTEGER I,INCX,N,NINCX
3340      C
3341      IF(N.LE.0)RETURN
3342      IF(INCX.EQ.1)GOTO 20
3343      C
3344      C      CODE FOR INCREMENT NOT EQUAL TO 1
3345      C
3346      NINCX = N*INCX
3347      DO 10 I = 1,NINCX,INCX
3348      CX(I) = CA*CX(I)
3349      10 CONTINUE
3350      RETURN
3351      C
3352      C      CODE FOR INCREMENT EQUAL TO 1
3353      C
3354      20 DO 30 I = 1,N
3355      CX(I) = CA*CX(I)
3356      30 CONTINUE
3357      RETURN
3358      END
3359      C      NAASA 1.1.021 ICAMAX FTN-A 05-02-78      THE UNIV OF MICH COMP CTR
3360      C      INTEGER FUNCTION ICAMAX(N,CX,INCX)
3361      C
3362      C      FINDS THE INDEX OF ELEMENT HAVING MAX. ABSOLUTE VALUE.
3363      C      JACK DONGARRA, LINPACK, 6/17/77.
3364      C
3365      IMPLICIT REAL*8 (A-H,O-Z)
3366      COMPLEX*16 CX(1)
3367      REAL*8 SMAX
3368      INTEGER I,INCX,IX,N
3369      COMPLEX*16 ZDUM
3370      REAL*8 CABS1
3371      CABS1(ZDUM) = DABS(DREAL(ZDUM)) + DABS(DIMAG(ZDUM))
3372      C
3373      ICAMAX = 1
3374      IF(N.LE.1)RETURN
3375      IF(INCX.EQ.1)GOTO 20
3376      C
3377      C      CODE FOR INCREMENT NOT EQUAL TO 1
3378      C
3379      IX = 1
3380      SMAX = CABS1(CX(1))

```

```
3381 IX = IX + INCX
3382 DO 10 I = 2,N
3383 IF(CABS1(CX(IX)) .LE. SMAX) GO TO 5
3384 ICAMAX = I
3385 SMAX = CABS1(CX(IX))
3386 IX = IX + INCX
3387 10 CONTINUE
3388 RETURN
3389 C
3390 C CODE FOR INCREMENT EQUAL TO 1
3391 C
3392 20 SMAX = CABS1(CX(1))
3393 DO 30 I = 2,N
3394 IF(CABS1(CX(I)) .LE. SMAX) GO TO 30
3395 ICAMAX = I
3396 SMAX = CABS1(CX(I))
3397 30 CONTINUE
3398 RETURN
3399 END
3400 CCCCCCCCCCCCCCCCCCCCCCCCCC
3401 C DREAL DOESN'T SEEM TO WORK, SO THIS FUNCTION IS A SUBSTITUTE
3402 REAL*8 FUNCTION DREAL(X)
3403 COMPLEX*16 X,X2
3404 REAL*8 XA(2)
3405 EQUIVALENCE (X2,XA(1))
3406 X2=X
3407 DREAL=XA(1)
3408 RETURN
3409 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-SiO₂ 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-32, 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.