(NASA-TM-86488) A NUMERICAL METHOD FOR INTERFACE PROBLEMS IN ELASTODYNAMICS (NASA) 105 p HC A06/MF A01 CSCL 20K

N85-17391

Unclas
G3/39 14067

NASA Technical Memorandum

NASA TM -86488

A NUMERICAL METHOD FOR INTERFACE PROBLEMS IN ELASTODYNAMICS

By David S. McGhee

Systems Dynamics Laboratory

December 1984

National Aeronautics and Space Administration

George C. Marshall Space Flight Center

TECHNICAL REPORT STANDARD TITLE PAGE

1. REPORT NO. NASA TM -86488	2. GOVERNMENT ACCESSION NO.	3. RECIPIENT'S CATALOG NO.
4. TITLE AND SUBTITLE A Numerical Method for Interface Problems in Elastodynamics		5. REPORT DATE December 1984
		6. PERFORMING ORGANIZATION CODE
7. AUTHOR(S) David S. McGhee		8. PERFORMING ORGANIZATION REPORT #
9. PERFORMING ORGANIZATION NAME AND ADDRESS George C. Marshall Space Flight Center Marshall Space Flight Center, Alabama 35812		10. WORK UNIT NO.
		11. CONTRACT OR GRANT NO.
12. SPONSORING AGENCY NAME AND ADDRESS National Aeronautics and Space Administration Washington, D.C. 20546		13. TYPE OF REPORT & PERIOD COVERED Technical Memorandum
		14. SPONSORING AGENCY CODE

15. SUPPLEMENTARY NOTES

Prepared by Systems Dynamics Laboratory, Science and Engineering.

16. ABSTRACT

This study deals with the numerical implementation of a formulation for a class of interface problems in elastodynamics. This formulation combines the use of the finite element and boundary integral methods to represent the interior and the exterior regions, respectively. In particular, the response of a semicylindrical alluvial valley in a homogeneous halfspace to incident antiplane SH waves is considered to determine the accuracy and convergence of the numerical procedure. Numerical results are obtained for several combinations of the incidence angle, frequency of excitation and relative stiffness between the inclusion and the surrounding halfspace. The results tend to confirm the theoretical estimates, that the convergence is of the order h^2 for the piecewise linear elements used. It is also observed that the accuracy decreases as the frequency of excitation increases or as the relative stiffness of the inclusion decreases.

17. KEY WORDS Boundary Integral Finite Element Elastodynamics Interface Problem	18. DISTRIBUTION STATEMENT Unclassified — Unlimited

19. SECURITY CLASSIF. (of this report) Unclassified	20. SECURITY CLASSIF. (of this page) Unclassified	21. NO. OF PAGES 105	22. PRICE NTIS

MSFC - Form 3292 (May 1969)

For sale by National Technical Information Service, Springfield, Virginia 22151

TABLE OF CONTENTS

PRECEDING PAGE BLANK NOT FILMED

ACKNOWLEDGMENT

I would like to thank my advisor Dr. Jacobo Bielak for his guidance and encouragement and the Department of Civil Engineering of Carnegie-Mellon University for its cooperation.

TECHNICAL MEMORANDUM

A NUMERICAL METHOD FOR INTERFACE PROBLEMS IN ELASTODYNAMICS

I. INTRODUCTION

In ground motion studies of earthquakes, the principal interest is in the effect on a localized region of the Earth. This region can take the form of a structure foundation, a region behaving nonlinearly near a structure, or a region that has significantly different soil properties from its surroundings. The difficulty in dealing with this type of problem analytically, is the infinite nature of the Earth, in relation to the region of interest, and its energy absorbing characteristics. In order to address these problems, either very large finite element models are required or absorbing boundaries must be used. Large finite element models are computationally inefficient and standard absorbing boundaries often do not accurately represent the energy dissipation of the Earth.

This problem can be defined in terms of an interior finite region and an exterior infinite region. This type of problem is called an "Interface Problem." A method, being investigated recently, to study these problems combines finite element and boundary integral methods [1,3]. The exterior region is handled by means of a boundary integral formulation and the interior is handled with finite element methods. Continuity conditions are placed across the boundary involving the displacement field and the corresponding tractions.

In this study, this technique is applied to a specific problem. The goal is to develop a computer implementation of the method and determine its accuracy and convergence. The problem chosen is that of an anti-plane SH shear wave incident upon a semi-cylindrical alluvial valley. A closed form analytical solution for this problem is available [2] and will be the basis for the comparisons made.

The specific problem presented here is a simple one. The study, however, has two significant goals: (1) to compare the approximate method with exact analytical results, and (2) to establish a firm baseline for further extensions of the method.

II. MATHEMATICAL FORMULATION

A. Problem Formulation

In order to formulate the problem in question, the region of interest must first be defined. Consider the geometry of Figure 1. The region Ω^+ is to represent homogeneous elastic material with mass density ρ^+ and Lame's constants λ^+, μ^+. The region Ω represents a possibly inhomogeneous cylinder with properties ρ^-, λ^-, and μ^- depending on the spatial coordinates x_1 and x_2. C is a traction-free surface and Γ is the interface between Ω and Ω^+.

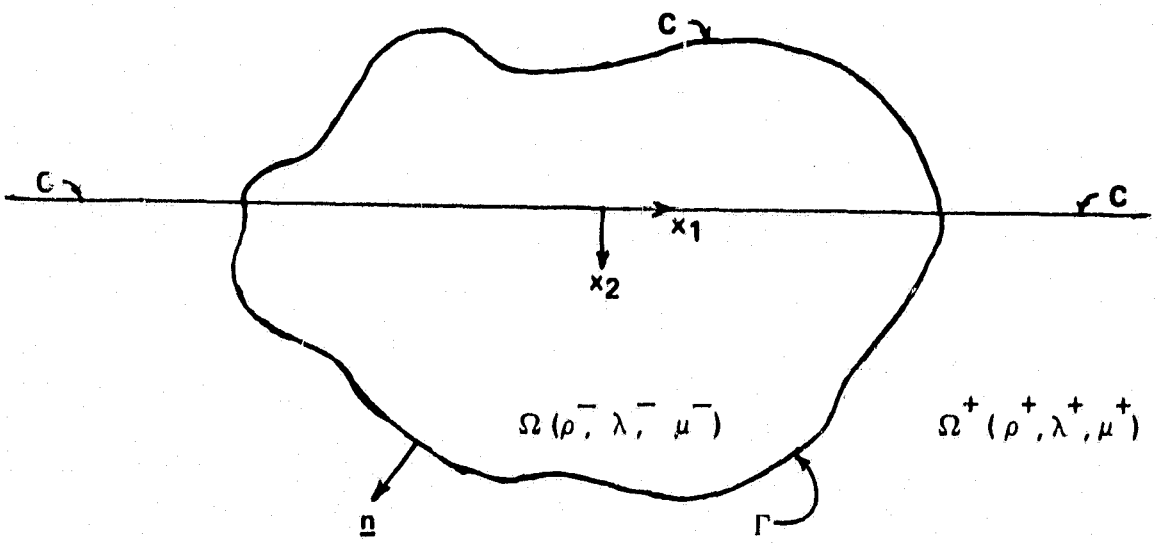

Figure 1.

A displacement field must be defined next. The case of antiplane strain due to an incoming antiplane shear wave shall be studied. While this represents a rather restricted type of excitation, the corresponding response will have qualitative similarities with that due to more general types of plane wave excitation. This approach can then be useful in gaining physical insight into the problem of evaluating the effects of local geology on ground motion.

The single non-vanishing, antiplane displacement is assumed to be time-periodic of the form:

$$W(x_1,x_2,t) = \mathrm{Re}[w^t(x_1,x_2)\, e^{i\omega t}] \tag{1}$$

Also, $w^\circ(x_1,x_2)$ corresponds to the driving field. w° will consist of an incoming wave defined in the halfspace $x_2 > 0$ and its reflection from the surface $x_2 = 0$, taken to be traction free. For specifying w° it will be assumed that the entire halfspace is homogeneous, with properties equal to those of Ω^+. That is, it will be assumed that the excitation displacement field, or free-field displacement, w° is known in the absence of the obstacle Ω.

For convenience, a displacement w is defined such that:

$$w = \begin{cases} w^t & \text{in } \Omega \\ w^t - w^\circ & \text{in } \Omega^+ \end{cases} \tag{2}$$

Thus, in Ω, w represents the total displacement; that is, the displacement due to the free-field excitation, w°, and the effects of the obstacle Ω. In Ω^+, w represents only the effects of the obstacle Ω without the free-field excitation. In other words, in Ω^+, w represents the scattered field only.

The problem then is to find the time periodic displacement field w, inside and outside the obstacle Ω, such that it satisfies the three following conditions. First, since both regions Ω and Ω^+ are linearly elastic, the displacement field w must satisfy a generalized, reduced wave equation in both of these regions. Second, w must satisfy several boundary conditions; namely, the displacements and tractions must be continuous across Γ and C is traction free. And third, w must satisfy a radiation condition in Ω^+. These conditions can be expressed as follows:

$$(\mu^- w_{x_1})_{x_1} + (\mu^- w_{x_2})_{x_2} + \rho^- \omega^2 w = 0 \quad \text{in } \Omega \tag{3}$$

$$\mu^+ \nabla^2 w + \rho^+ \omega^2 w = 0 \quad \text{in } \Omega^+ \tag{4}$$

$$w^{t+} - w^{t-} = 0 \quad ; \quad \mu^+ w_n^{t+} - \mu^- w_n^{t-} = 0 \quad \text{on } \Gamma \tag{5,6}$$

$$\mu\, w_n = 0 \quad \text{on } C \tag{7}$$

$$w \sim r^{-1/2} e^{-ir} \quad \text{as} \quad r = x_1^2 + x_2^2 \to \infty \tag{8}$$

where

ω = frequency of free-field motion, w°

w^{t+} = total displacement on Ω^+ side of boundary Γ

w^{t-} = total displacement on Ω side of boundary Γ

w_n^{t+} = normal derivative of total displacement on Ω^+ side of boundary Γ

w_n^{t-} = normal derivative of total displacement on Ω side of boundary Γ .

Taking equation (2) into consideration, equations (5) and (6) can be rewritten as:

$$w^- = w^+ + w^\circ \tag{9}$$

$$\mu^- w_n^- = \mu^+ w_n^+ + \mu^+ w_n^\circ \tag{10}$$

respectively, where:

w_n° = normal derivative of free-field displacement.

Equations (3), (4), (7), (8), (10), and (11) define a well-posed problem for w in Ω and Ω^+.

It is desirable, from a computational point of view, to reformulate the problem in a way that eliminates the need to deal with an infinite exterior region, which is both time and space consuming. This is done at the expense of introducing two boundary functions on Γ. It begins with some potential theory for the Helmholtz equation:

$$\mu^+ \nabla^2 v + \rho^+ \omega^2 v = 0 \quad , \quad x_2 > 0 \tag{11}$$

subject to the radiation condition (8) and the traction-free boundary condition,

$$v_{x_2} = 0 \quad \text{on} \quad x_2 = 0 \quad .$$

G is defined by the formula:

$$G(\underline{x},\underline{y}) = \frac{i}{H} \left\{ H_o^{(2)} (k|\underline{x}-\underline{y}|) + H_o^{(2)} (k|\underline{x}-\underline{z}^*|) \right\} \tag{12}$$

where

$k^2 = \omega^2 \rho^+/\mu^+$

$\underline{y}^* = (y_1, -y_2)$

$H_o^2(\cdot)$ = the Hankel function of the second kind and zeroth order.

Physically, G can be interpreted as the displacement produced at $\underline{x}$ due to a steady-state point load of frequency ω and unit amplitude at $\underline{y}$.

It is assumed that the solution v in Ω^+ can be represented in terms of a simple layer with density ϕ; that is, it is assumed that a function ϕ exists, defined over Γ, such that:

$$v(\underline{x}) = \int_\Gamma G(\underline{x},\underline{y})\ \phi(\underline{y})\ dS\underline{y} \qquad \text{in} \quad \Omega^+ \quad . \tag{13}$$

Thus, the displacement v at every point within Ω^+ can be represented in terms of a density function ϕ defined only on Γ. It can be shown [5] that G is a continuous function and, therefore, the limiting displacement $v^\pm$ on Γ is given by

$$v^\pm(\underline{x}) = \int_\Gamma G(\underline{x},\underline{y})\ \phi(\underline{y})\ dS\underline{y} \quad \text{on} \quad \Gamma \quad . \tag{14}$$

In addition, under the assumption that Γ is smooth, the limiting normal derivative is [5]

$$v_n^{\pm}(\underline{x}) = \pm \frac{1}{2}\,\phi(\underline{x}) + \int_\Gamma \frac{\partial G(\underline{x},\underline{y})}{\partial n_{\underline{x}}}\,\phi(\underline{y})\,dS\underline{y} \qquad \text{on } \Gamma \quad . \tag{15}$$

The plus and minus denote limits from Ω^+ and Ω, respectively, and $n_{\underline{x}}$ is the unit normal vector on Γ at point $\underline{x}$ (Fig. 1). Then equations (4), (5), and (6) can be replaced in the original problem statement by a combination of equations (9), (10), (14), and (15). This yields

$$w^- = \int_\Gamma G(\underline{x},\underline{y})\,\phi(\underline{y})\,dS\underline{y} + w^\circ \qquad \text{on } \Gamma$$

$$\mu^-\,w_n^- = \frac{1}{2}\,\mu^+\,\phi(\underline{x}) + \mu^+ \int_\Gamma \frac{\partial G(\underline{x},\underline{y})}{\partial n_{\underline{x}}}\,\phi(\underline{y})\,dS\underline{y} + \mu^+\,w_n^\circ \qquad \text{on } \Gamma \quad .$$

In order to arrive later at a symmetric discretized formulation of the problem, the following analogues of the Helmholtz formulas are introduced. If v satisfies equations (11) and (12) in Ω then,

$$1/2\;v^- = \int_\Gamma \frac{\partial G(\underline{x},\underline{y})}{\partial n\underline{y}}\,v^-(\underline{y})\,dS\underline{y} - \int_\Gamma G(\underline{x},\underline{u})\,v_n^-(\underline{y})\,dS\underline{y} \qquad \text{in } \Omega \quad . \tag{16}$$

If v satisfies equations (11) and (8) in Ω^+ then,

$$1/2\;v^+ = \int_\Gamma G(\underline{x},\underline{y})\,v_n^+(\underline{y})\,dS\underline{y} - \int_\Gamma \frac{\partial G(\underline{x},\underline{y})}{\partial n\underline{y}}\,v^+(\underline{y})\,dS\underline{y} \qquad \text{in } \Omega^+ \quad . \tag{17}$$

By equations (9) and (10) $w^+ = w^- - w^\circ$ and $w_n^+ = \mu^-/\mu^+\,w_n^- - w^\circ$, respectively. By substituting these results into equation (17),

$$1/2\;w^- + \int_\Gamma \frac{\partial G(\underline{x},\underline{y})}{\partial n\underline{y}}\,w^-(\underline{y})\,dS\underline{y} - \int_\Gamma G(\underline{x},\underline{y})\left(\frac{\mu^-}{\mu^+}\,w_n^-\right)(\underline{y})\,dS\underline{y}$$

$$= 1/2\;w^\circ - \int_\Gamma G(\underline{x},\underline{y})\,w_n^\circ(\underline{y})\,dS\underline{y} + \int_\Gamma \frac{\partial G(\underline{x},\underline{y})}{\partial n\underline{y}}\,w^\circ(\underline{y})\,dS\underline{y} \quad , \tag{18}$$

is obtained. Now w° is a solution of equation (11) in Ω, hence by equation (16),

$$1/2\ w^\circ = \int_\Gamma \frac{\partial G(\underline{x},\underline{y})}{\partial n\underline{y}}\ w^\circ(\underline{y})\ dS\underline{y} - \int_\Gamma G(\underline{x},\underline{y}\_\ w_n{}^\circ(\underline{y})\ dS\underline{y}\ .$$

So the right hand side of equation (18) becomes w° and can be rewritten,

$$1/2\ w^- + \int_\Gamma \frac{\partial G(\underline{x},\underline{y})}{\partial n\underline{y}}\ w^-(\underline{y})\ dS\underline{y} - \int_\Gamma G(\underline{x},\underline{y}) \left(\frac{\mu^-}{\mu^+} w_n{}^-\right) (\underline{y})\ dS\underline{y} = w^\circ\ . \tag{19}$$

This leads to the following problem. Find $(w,\ \mu^-\ w_n{}^-,\ \phi)$ such that

$$(\mu^-\ w_{x_1})_{x_1} + (\mu^-\ w_{x_2})_{x_2} + \rho^-\ \omega^2\ w = 0 \quad \text{in} \quad \Omega \tag{20a}$$

$$w^- = \int_\Gamma G(\underline{x},\underline{y})\ \phi(\underline{y})\ dS\underline{y} + w^\circ \quad \text{on} \quad \Gamma \tag{20b}$$

$$\mu^-\ w_n{}^- = 1/2\ \mu^+\ \phi(\underline{x}) + \mu^+ \int_\Gamma \frac{\partial G(\underline{x},\underline{y})}{\partial n\underline{x}}\ \phi(\underline{y})\ dS\underline{y} + \mu^+\ w_n{}^\circ \quad \text{on} \quad \Gamma \tag{20c}$$

$$\mu^-\ w_n = 0 \quad \text{on} \quad C \tag{20d}$$

$$1/2\ w^- + \int_\Gamma \frac{\partial G(\underline{x},\underline{y})}{\partial n\underline{y}}\ w^-(\underline{y})\ dS\underline{y} - \int_\Gamma G(\underline{x},\underline{y}) \left(\frac{\mu^-}{\mu^+} w_n{}^-\right) (\underline{y})\ dS\underline{y} = w^\circ \quad \text{on} \quad \Gamma\ . \tag{20e}$$

The only set of points now used are those in Ω and Γ. The set of points in Ω^+ is completely described by points in Γ by the use of the Green's function. It has been shown in Reference 1 that this problem has, in general, a unique solution provided certain critical values of ω are excluded.

B. Variational Principle

Now standard Galerkins' ideas are used to obtain a variational formulation which will be suitable for discretization by the finite element method. First, multiply equation (20a) by a test function δw and integrate over Ω using the divergence theorem and equation (20d). The result is,

$$-\int_\Omega \{\mu^-(w_{x_1}\ \delta w_{x_1} + w_{x_2}\ \delta w_{x_2}) - \omega^2\ \rho^-\ w\ \delta w\}\ dx + \int_\Gamma \mu^-\ w_n{}^-\ \delta w\ dS = 0 \tag{21a}$$

Next, multiply equation (20b) by a test function $\delta\ \mu^-\ w_n^-$ and integrate over Γ;

$$\int_\Gamma \left[\int_\Gamma G(\underline{x},\underline{y})\ \phi(\underline{y})\ dS\underline{y} - w^-(\underline{x}) + w^\circ(\underline{x}) \right] \delta\ \mu^-\ w_n^-(\underline{x})\ d\underline{x} \quad . \tag{21b}$$

Perform similar calculations for equations (20c) and (20e) and combine the resulting equations linearly with (21a) and (21b) to obtain:

$$\left[- \int_\Omega \{\mu^-(w_{x_1}\ \delta w_{x_1} + w_{x_2}\ \delta w_{x_2}) - \omega^2\ \rho^-\ w\ \delta\ w\}\ d\underline{x} + \int_\Gamma \mu^-\ w_n^-\ \delta w\ dS \right]$$

$$- 1/2 \left[\int_\Gamma \left\{ \int_\Gamma G(\underline{x},\underline{y})\ \phi(\underline{y})\ dS\underline{y} - w^-(\underline{x}) + w^\circ(\underline{x}) \right\} \delta\ \mu^-\ w_n^-(\underline{x})\ dS \right]$$

$$+ 1/2 \left[\int_\Gamma \left\{ 1/2\ \mu^+\ \phi(\underline{x}) + \mu^+ \int_\Gamma \frac{\partial G(\underline{x},\underline{y})}{\partial n\underline{x}}\ \phi(\underline{y})\ dS\underline{y} + \mu^+\ w_n^\circ - \mu^-\ w_n^- \right\} \delta\ w^-\ dS \right]$$

$$+ 1/2 \left[\int_\Gamma \left\{ 1/2\ \mu^+\ w^-(\underline{x}) + \mu^+ \int_\Gamma \frac{\partial G(\underline{x},\underline{y})}{\partial n\underline{y}}\ w^-(\underline{y})\ dS\underline{y} - \int_\Gamma G(\underline{x},\underline{y})\ (\mu^-\ w_n^-)(\underline{y})\ dS\underline{y} \right.\right.$$

$$\left.\left. - \mu^+\ w^\circ \right\} \delta\ \phi(x)\ dS \right] = 0 \quad . \tag{22}$$

The problem is now in a form suitable for finite element approximation. This is done by defining the various functions in equation (22) using finite element shape functions as follows:

$$w(\underline{x}) = [N(\underline{x})]^T\ \{w\} = [N_\Omega(\underline{x})^T\ :\ N_\Gamma(\underline{z})^T]\ \begin{Bmatrix} w_\Omega \\ w_\Gamma \end{Bmatrix} \quad . \tag{23a}$$

$$\phi(\underline{x}) = [Q(\underline{x})]^T\ \{\phi\} \tag{23b}$$

$$\mu^-\ w_n^-(\underline{x}) = \lambda(\underline{x}) = [Q(\underline{x})]^T\ \{\lambda\} \quad . \tag{23c}$$

The corresponding test functions are defined using the same shape functions.

Using these shape functions equation (22) leads to the matrix equation:

$$\begin{bmatrix} K_{\Omega\Omega} & K_{\Omega\Gamma} & 0 & 0 \\ K_{\Gamma\Omega} & K_{\Gamma\Gamma} & A^T & B^T \\ 0 & A & 0 & C^T \\ 0 & B & C & 0 \end{bmatrix} \begin{Bmatrix} w_\Omega \\ w_\Gamma \\ \lambda \\ \phi \end{Bmatrix} = \begin{Bmatrix} 0 \\ f_\Gamma \\ f_\lambda \\ f_\phi \end{Bmatrix} \tag{24}$$

where

$$[K] = -\int_\Omega \{[N_{x_1}]\ \mu^-[N_{x_1}]^T + [N_{x_2}]\ \mu^-[N_{x_2}]^T - \omega^2[N]^T\ \rho^-[N]\}\ d\underline{x}$$

$$[A] = 1/2 \int_\Gamma [Q]\ [N_\Gamma]^T\ dS$$

$$[B] = 1/2\ \mu^+ \left\{1/2 \int_\Gamma [Q]\ [N_\Gamma]^T\ dS + \int_\Gamma \int_\Gamma [Q] \left[\frac{\partial G(\underline{x},\underline{y})}{\partial n\underline{y}}\right] [N_\Gamma]^T\ dS\underline{y}\ dS\right\}$$

$$[C] = -\ 1/2 \int_\Gamma \int_\Gamma [Q]\ [G(\underline{x},\underline{y})]\ [Q]^T\ dS\underline{y}\ dS$$

$$\{f_\Gamma\} = -\ 1/2\ \mu^+ \int_\Gamma [N_\Gamma]\ w_n{}^\circ\ dS$$

$$\{f_\lambda\} = 1/2 \int_\Gamma [Q]\ w^\circ\ dS$$

$$\{f_\phi\} = 1/2\ \mu^+ \int_\Gamma [Q]\ w^\circ\ dS \quad .$$

The complete development of equation (24) from equation (22) is shown in Appendix A.

For convenience $\{\lambda\}$ and $\{\phi\}$ are now condensed out, so the equation is in terms of $\{w\}$ only.

$$\begin{bmatrix} K_{\Omega\Omega} & K_{\Omega\Gamma} \\ K_{\Gamma\Omega} & K_{\Gamma\Gamma} + D_{\Gamma\Gamma} \end{bmatrix} \begin{Bmatrix} w_\Omega \\ w_\Gamma \end{Bmatrix} = \begin{Bmatrix} 0 \\ F_\Gamma \end{Bmatrix} \tag{25}$$

is obtained where

$$[D_{\Gamma\Gamma}] = -[A]^T\ [C]^{-1}\ [B] - [B]^T\ [C^T]^{-1}\ [A]$$

$$\{F_\Gamma\} = \{f_\Gamma\} - [A]^T\ [C]^{-1}\ \{f\phi\} - [B]^T\ [C^T]^{-1}\ \{f_\lambda\} \quad .$$

From equation (24), it can be seen that [K] is essentially the stiffness matrix of the cylinder; that is, the force at some point in the cylinder due to a unit displacement at some other point in the cylinder. [C] represents the discretized compliance of the halfspace. [A] and [B] are matrices which couple the cylinder boundary and the halfspace boundary and guarantee the continuity of displacements and tractions across Γ.

From equation (25), $[D_{\Gamma\Gamma}]$ is the force at a point on the boundary due to a force or traction applied at some other point on the boundary allowing for the presence of the halfspace medium. $[D_{\Gamma\Gamma}]$ is essentially a representation of a nonlocal absorbing boundary developed from a Green's function rather than a spring and dashpot model.

III. APPLICATIONS

The primary objective of this study was to develop a computer code to implement the previously described formulation and to check its accuracy. In order to do this, a baseline reference was needed. The most commonly cited problem of this type deals with a semicircular deposit with homogeneous material properties, for which an exact solution is available [2]. The program developed, therefore, used these characteristics. Extensions to more general problems can be made once the method has been checked.

The free-field antiplane shear wave, w^o, was taken as

$$w^o(x_1,x_2) = e^{-i(\omega t - kx_1 \sin \theta_0)} \cos(kx_2 \cos \theta_0) \tag{26}$$

where

θ_0 = angle of incidence

$k^2 = \omega^2 \rho^+/\mu^+$.

Figure 2 illustrates x_1, x_2, and θ_0 and their sign conventions.

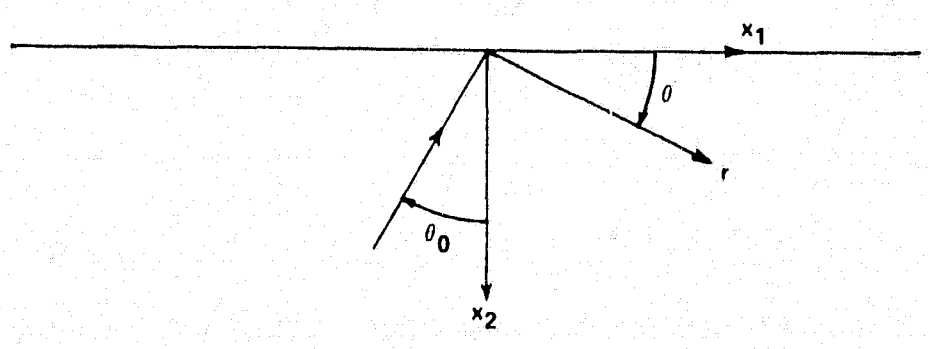

Figure 2.

With the geometry being circular, polar coordinates were used for their convenience. By substituting polar coordinates and simplifying, the incident wave becomes:

$$w^{o} = e^{-i\omega t}\ 1/2\ [e^{i\ kr\ \sin(\theta+\theta_o)} + e^{-i\ kr\ \sin(\theta-\theta_o)}] \tag{27}$$

and the normal derivative:

$$\frac{\partial w^{o}}{\partial n} = \frac{\partial w^{o}}{\partial r} = e^{i\omega t}\ 1/2\ [i\ k\ \sin(\theta+\theta_o)\ e^{i\ kr\ \sin(\theta+\theta_o)} - i\ k\ \sin(\theta-\theta_o)\ e^{i\ kr\ \sin(\theta-\theta_o)}] \tag{28}$$

The Green's function (12) can be expressed as

$$G(\underline{x},\underline{y}) = \frac{i}{4}\ H_o^{(2)}\ [2\ kr\ |\sin\frac{\Delta\theta}{2}|] \tag{29}$$

where

$\Delta\theta$ = angular distance between $\underline{x}$ and $\underline{y}$.

Note that this form of the Green's function satisfies the wave equation in Ω^+ and the radiation condition. The normal derivative of this function is:

$$\frac{\partial G(\underline{x},\underline{y})}{\partial n} = \frac{\partial G(\underline{x},\underline{y})}{\partial r} = \frac{-ik}{2}\ H_1^{(2)}\ (2kr\ |\sin\frac{\Delta\theta}{2}|)\ |\sin\frac{\Delta\theta}{2}| \tag{30}$$

which is bounded and continuous.

For the interior region the elements used for the finite element analysis represent a simple polar grid as shown in Figure 3. This necessitated two basic elements; a "triangular" element and a "rectangular" element. For this interior region the displacement was approximated by shape functions which are piecewise linear in both r and θ. This choice of shape functions provided the necessary accuracy but kept the computations simple.

The boundary elements of the halfspace for the functions ϕ and λ are "line" elements. To preserve symmetry of the resulting matrices, these two functions used the same shape function. A piecewise constant shape function was chosen here, again because this provided the necessary accuracy while keeping computations simple. These elements were chosen to coincide with the boundary elements resulting from the interiors division. This is not necessary to the method but is more convenient computationally.

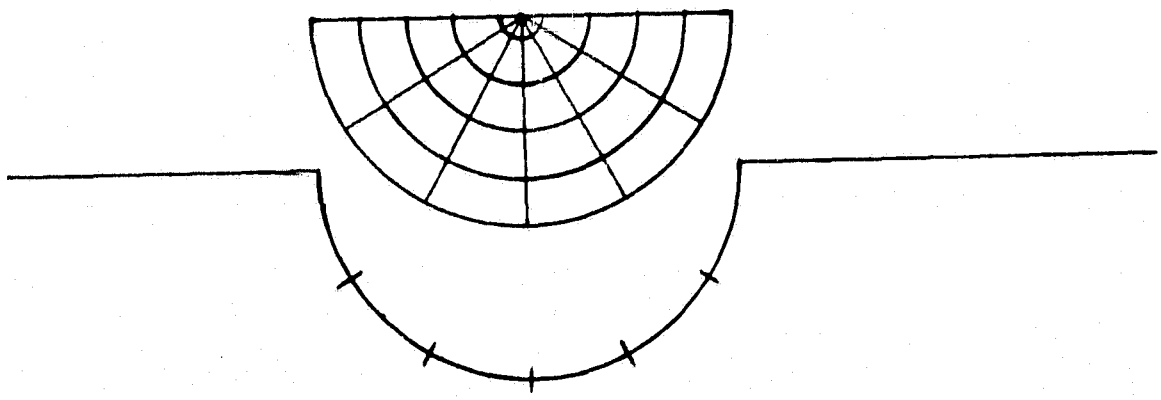

Figure 3.

In the interior, the number of elements in the radial and angular directions was varied to provide a check of the convergence rate of the method. Although it resulted in elements of significantly different areas, the radial divisions were made uniform. This resulted in a natural placing of more elements where needed, near the center. The maximum number of elements used for this study was dictated by the limits of the computer used.

The most computationally expensive parts of this method are the integrals involving the Green's function and its normal derivative. Because of their complexity, these integrals must be evaluated numerically. Gaussian quadrature was selected for this purpose. Because the integrand contains a logarithmic singularity, a modified Gaussian method, to take directly into effect this singularity, was used as presented by Harris and Evans [4]. In addition, care was taken to split the integral, at the singularity, into two integrals. The integration schemes used were tested on the natural logarithm function to determine the number of points necessary for accuracy. By using these techniques the amount of computation for each integral was kept to a minimum while returning the necessary accuracy.

The symmetry of the full space problem was also used to reduce the number of computations. By constructing the matrices for the full space the actual number of integrals necessary to perform was reduced because of symmetry. Further symmetry considerations then allowed the matrices to be reduced to the halfspace problem.

For the various matrix operations, such as solving equations and inverting matrices, the appropriate IMSL routines were used.

IV. NUMERICAL RESULTS

As stated previously, the primary objectives of this study were to determine the accuracy and convergence of the proposed formulation. Given the applications discussed in Section III a number of cases were considered.

The first consideration was the number of elements needed to achieve a desired accuracy and the rate of convergence as the number of elements is increased. For this study, results were computed for five radial and five angular elements; 10 radial and 10 angular elements; and 20 radial and 20 angular elements.

The second consideration was the frequency of the free-field motion. As the frequency of the free-field motion increases, it was expected that the smoothness of the response would decrease. This indicates that more elements are needed to achieve a fixed accuracy for higher frequencies. The circular dimensionless frequencies considered in this study were π, 0.50π, and 0.25π.

The third consideration was the angle of incidence of the free-field wave front. It was of interest to determine whether the angle of incidence has any effect on the solution. To do this, this study computed results for incidences of 0° and 60°. This represented wave fronts coming from directly below and from the left, respectively.

The fourth consideration was the stiffness of the obstacle. The ability to vary the stiffness of the obstacle in relation to the stiffness of the halfspace is important to the usefulness of the formulation and any further extensions. For this study two cases were considered. An obstacle stiffer than the halfspace was studied using a ratio of mass densities (ρ^-/ρ_+) equal to 1.50 and a ratio of shear wave velocities (β^-/β_+) equal to 2.00. A softer obstacle used a density ratio of 0.67 and a velocity ratio of 0.50.

For all of the above applications and cases a closed form solution is available in terms of a series involving Bessel functions [2]. All of these cases and their exact solutions were computed. The results for various points on the surface ($\theta = 0°$ and $\theta = 180°$) were compiled into Tables 1 through 3. Both the real and imaginary parts of the solution are shown. For an incidence angle of 0° only those points where $\theta = 0°$ are shown since the response is symmetric.

The relative errors for three surface points were computed and plotted against the number of elements for the various cases. This error is defined as:

$$\text{error} = \left| \frac{\text{exact} - \text{computed}}{\text{exact}} \right| .$$

The graphs of the errors are shown in Figures 4 through 9.

As can be seen from the graphs, the convergence and accuracy of the formulation is very good. The rate of convergence is on the order of the square of the number of elements or better. The relative error itself is small even when only using 20 elements. The effect of frequency generally follows the expected trend, that is, as the frequency increases more elements are needed to provide accurate answers. There is little effect on the accuracy of the formulation from the angle of incidence of the free-field motion or the relative stiffness of the obstacle and halfspace.

Another concern was the viability of the formulation at or near the natural frequency of the obstacle when having the same material properties as the exterior region. At this natural frequency, the matrix [C] defined in equation (24) becomes ill-conditioned and difficult to invert. This is a particular problem with this study since inversion of the [C] matrix was required for the reduction done in equation (25). This inversion did, in fact, fail at the natural frequency of 0.7655π. However, good results were obtained at a frequency of 0.7632π. Results for this frequency and

another close frequency are shown in Table 4. It is believed that this problem can be avoided by solving the full problem [equation (24)]. This could not be confirmed in this study, however, because of computer limitations.

V. CONCLUSIONS

The numerical results obtained indicate the formulation is very good. Accuracy is good even with few elements and the convergence rates are on the order of the number of elements squared or better. The formulation can be used for a wide range of incidence angles and frequencies. A variety of obstacle densities can be accommodated and the natural frequency of the halfspace can be closely approached.

Now that the formulation has been numerically proven, a number of extensions for further study suggest themselves. An obstacle of inhomogeneous properties, a region of irregular shape, and a region of nonlinear properties are some possible extensions. This study provides a firm baseline for this further study.

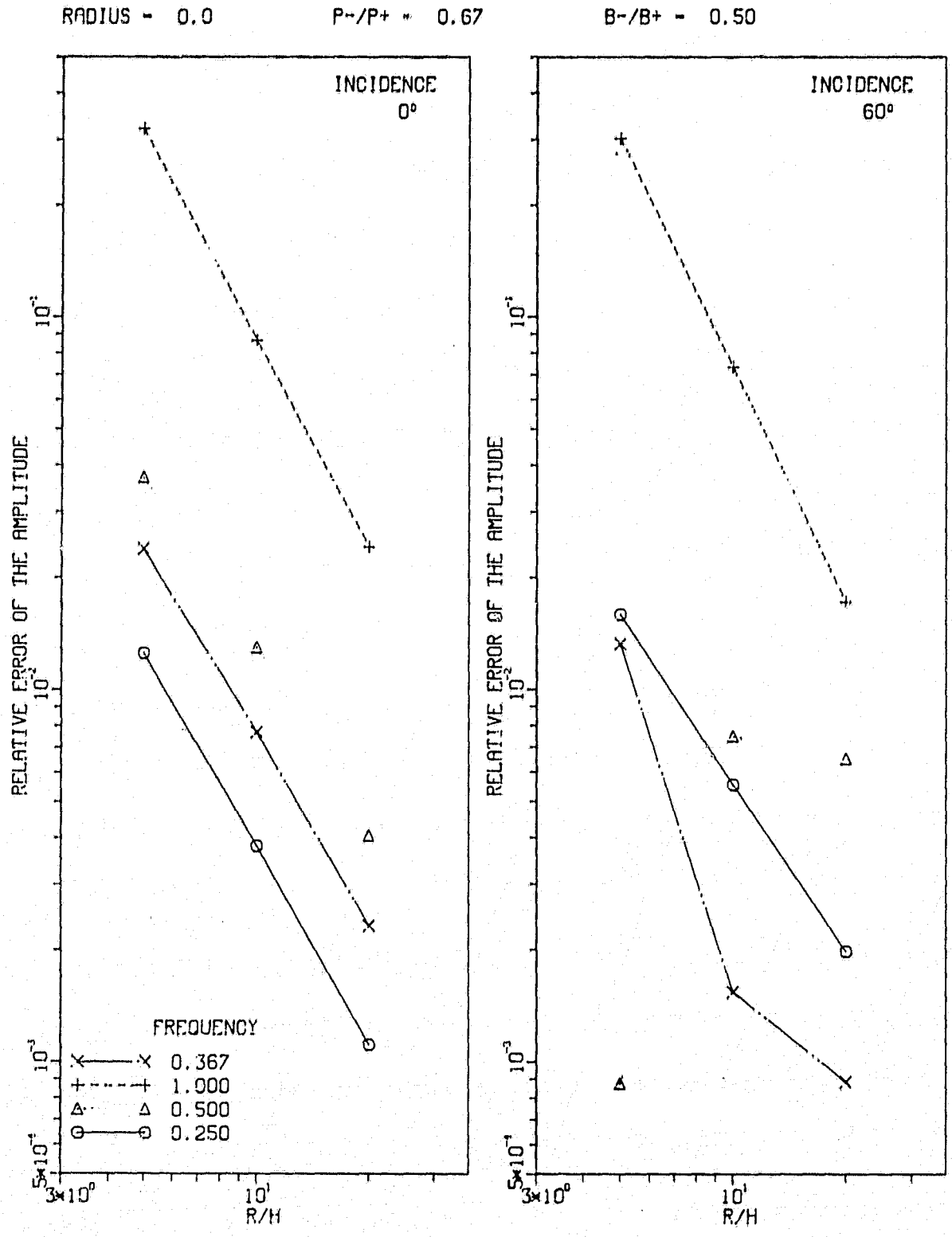
RADIUS = 0.0
P-/P+ = 0.67
B-/B+ = 0.50
INCIDENCE
0°
INCIDENCE
60°
RELATIVE ERROR OF THE AMPLITUDE
R/H
FREQUENCY
0.367
1.000
0.500
0.250

Figure 4.

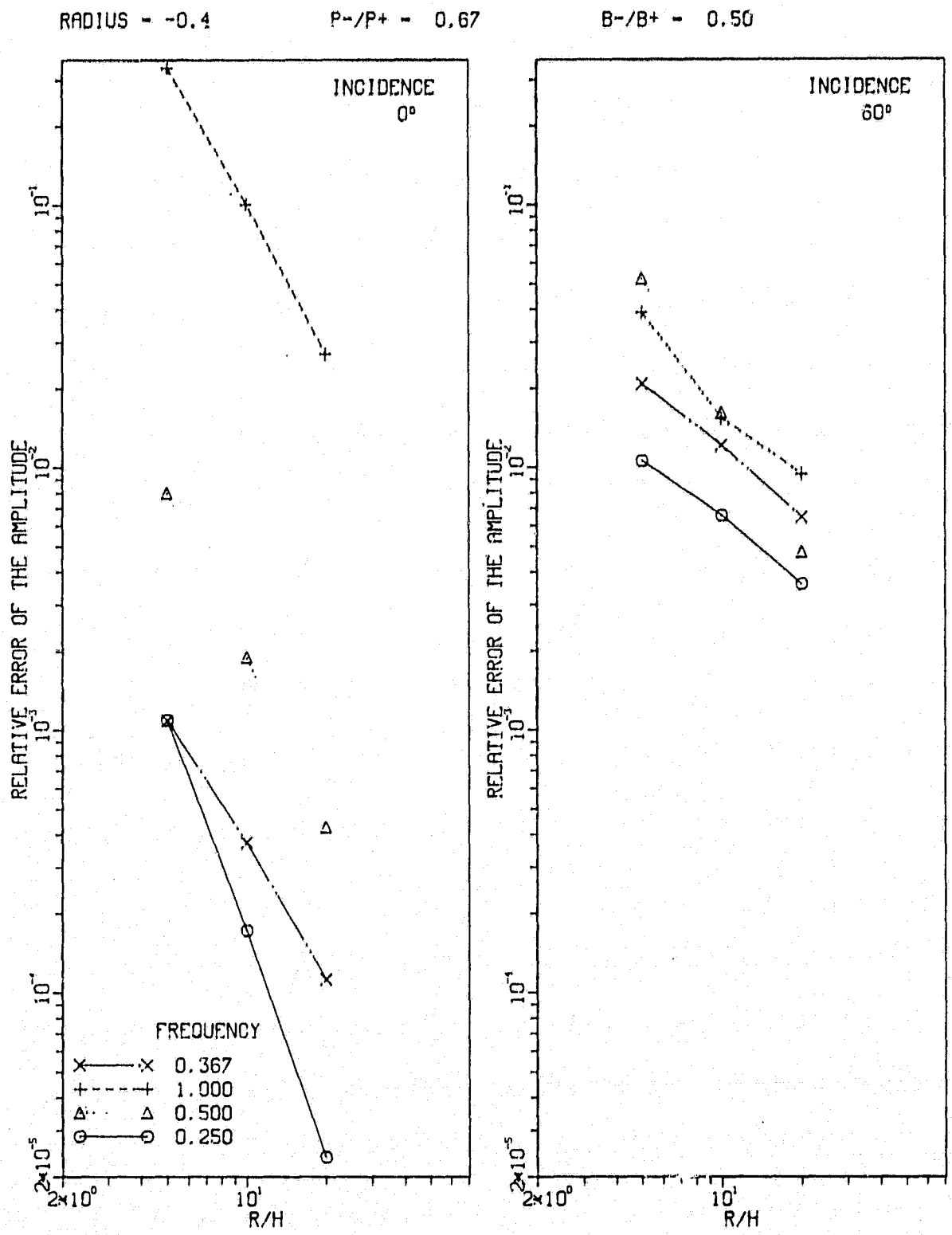
RADIUS = -0.4
P-/P+ = 0.67
B-/B+ = 0.50
INCIDENCE
0°
INCIDENCE
60°
RELATIVE ERROR OF THE AMPLITUDE
R/H
FREQUENCY
0.367
1.000
0.500
0.250

Figure 5.

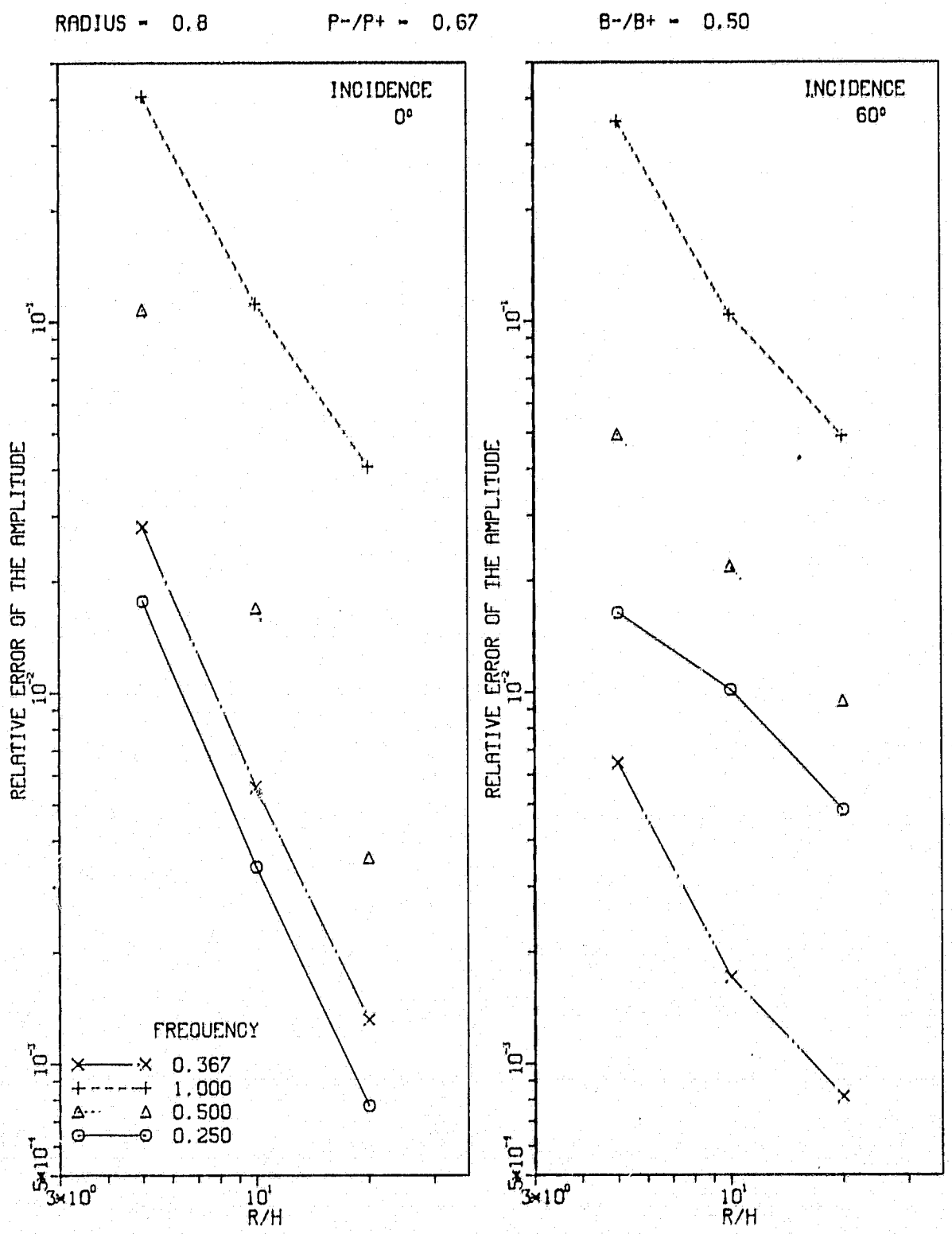
RADIUS = 0.8
P-/P+ = 0.67
B-/B+ = 0.50
INCIDENCE 0°
INCIDENCE 60°
RELATIVE ERROR OF THE AMPLITUDE
R/H
FREQUENCY
0.367
1.000
0.500
0.250

Figure 6.

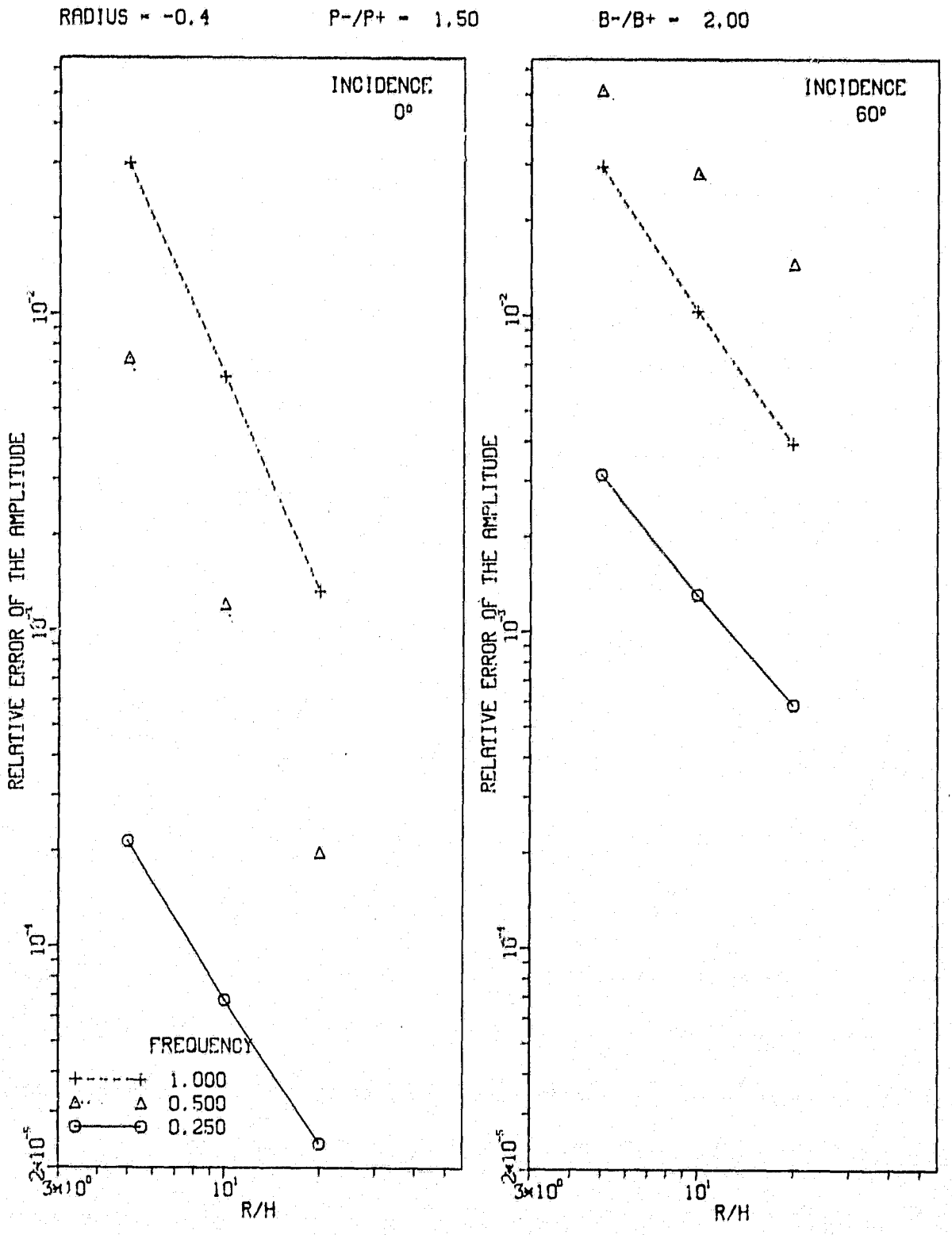
RADIUS = -0.4
P-/P+ = 1.50
B-/B+ = 2.00
INCIDENCE
0°
INCIDENCE
60°
RELATIVE ERROR OF THE AMPLITUDE
R/H
FREQUENCY
1.000
0.500
0.250

Figure 7.

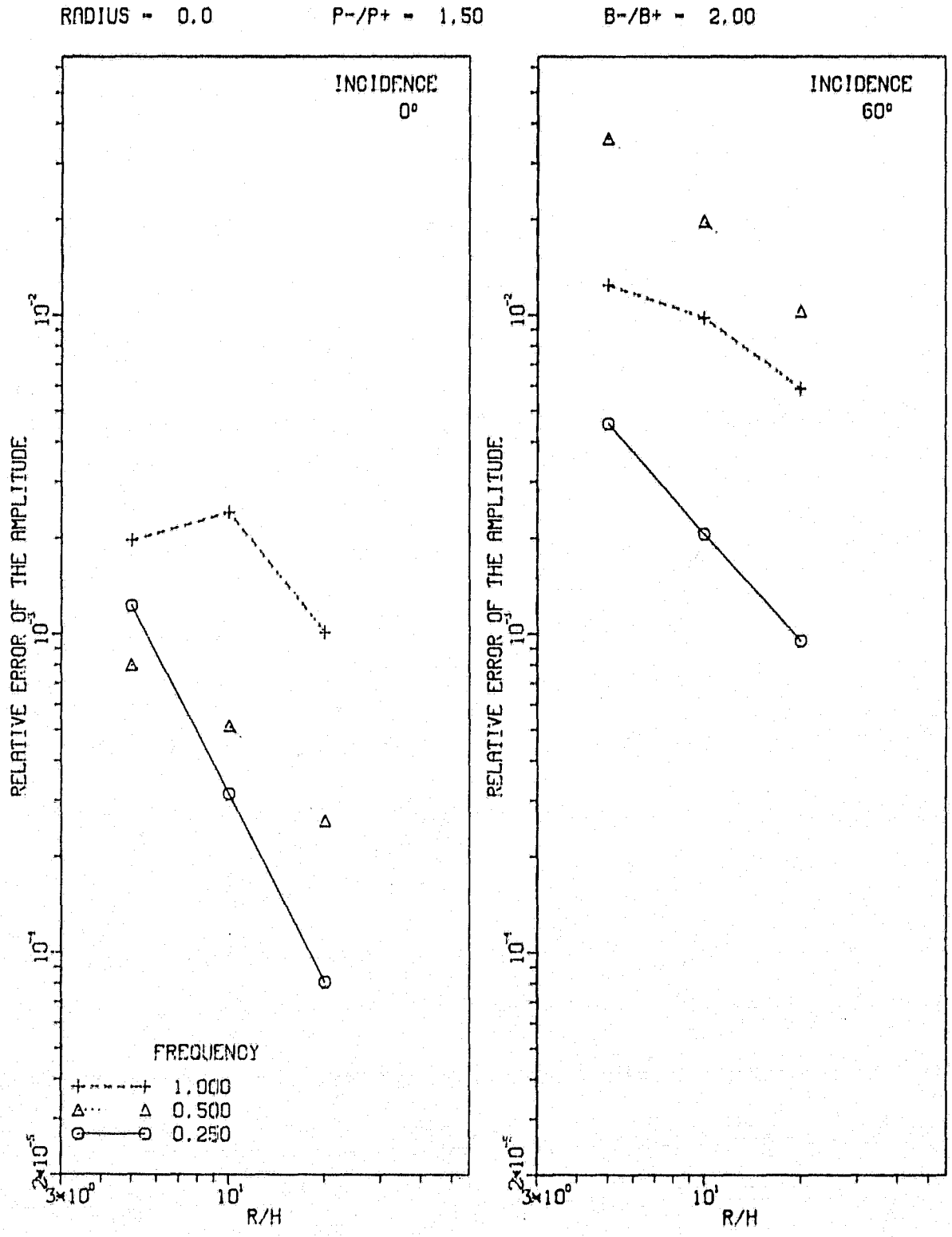
RADIUS = 0.0
P-/P+ = 1.50
B-/B+ = 2.00
INCIDENCE
0°
INCIDENCE
60°
RELATIVE ERROR OF THE AMPLITUDE
R/H
FREQUENCY
1.000
0.500
0.250

Figure 8.

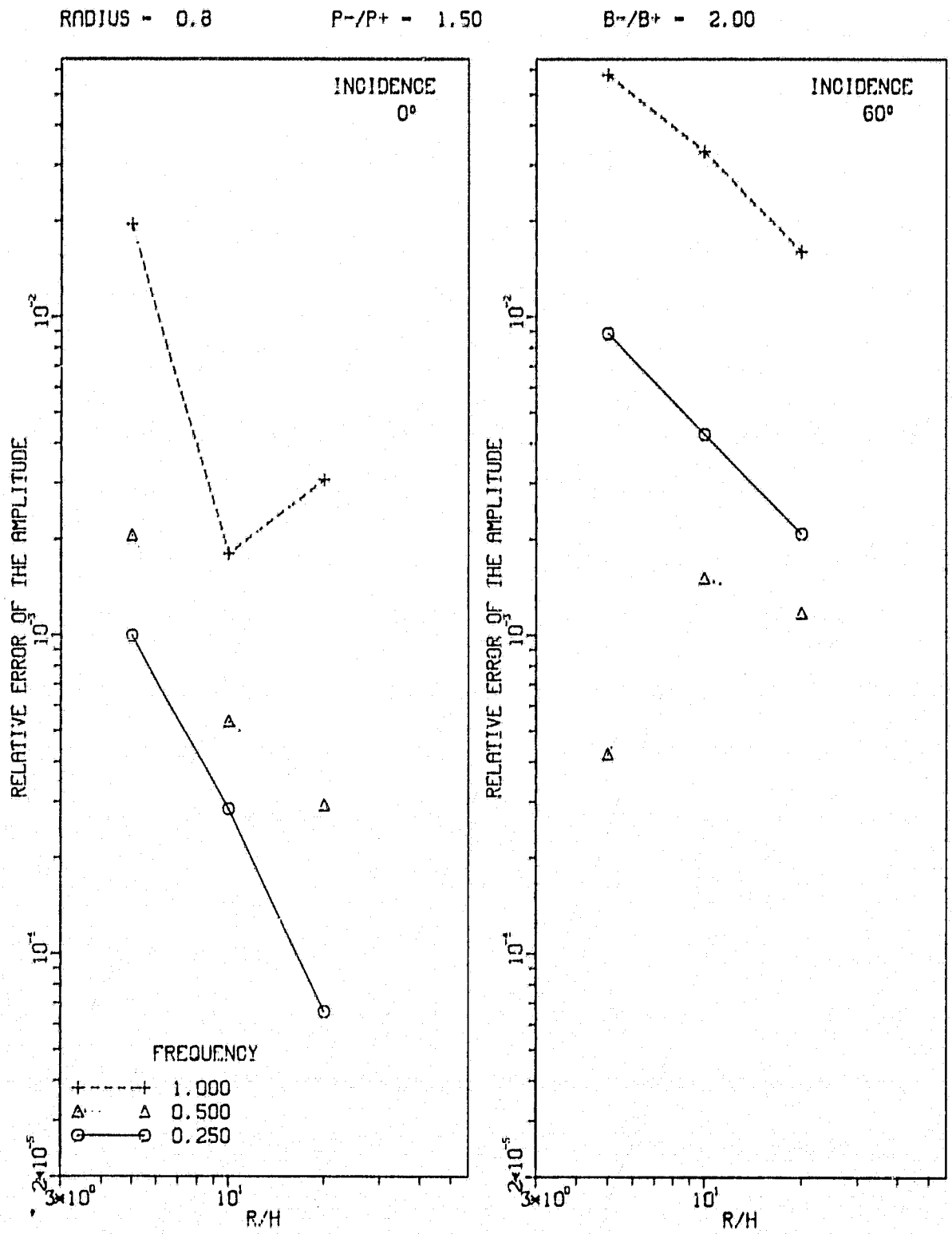
RADIUS - 0.8
P-/P+ - 1.50
B-/B+ - 2.00
INCIDENCE
0°
INCIDENCE
60°
RELATIVE ERROR OF THE AMPLITUDE
R/H
FREQUENCY
1.000
0.500
0.250

Figure 9.

TABLE 1

FREQUENCY = $.25\pi$ INCIDENCE ANGLE = 0°
ρ_-/ρ_+ = .67 β_-/β_+ = .50

RADIUS	5 ELEMENTS		10 ELEMENTS		20 ELEMENTS		EXACT	
1.00	1.1717	-0.0106	1.1320	-0.0127	1.1241	-0.0129	1.1218	-0.0130
0.80	1.3671	-0.0198	1.3479	-0.0220	1.3444	-0.0222	1.3434	-0.0222
0.60	1.5405	-0.0278	1.5338	-0.0302	1.5327	-0.0304	1.5324	-0.0303
0.40	1.6787	-0.0339	1.6771	-0.0365	1.6769	-0.0367	1.6768	-0.0366
0.20	1.7713	-0.0379	1.7685	-0.0405	1.7677	-0.0407	1.7674	-0.0406
0.00	1.8209	-0.0394	1.8051	-0.0419	1.8003	-0.0421	1.7983	-0.0420

FREQUENCY = $.50\pi$ INCIDENCE ANGLE = 0°
ρ_-/ρ_+ = .67 β_-/β_+ = .50

RADIUS	5 ELEMENTS		10 ELEMENTS		20 ELEMENTS		EXACT	
1.00	1.4287	-0.1334	1.2486	-0.1654	1.2172	-0.1699	1.2084	-0.1710
0.80	1.0713	0.2401	0.9829	0.2230	0.9697	0.2214	0.9661	0.2211
0.60	0.5380	0.7157	0.5122	0.7145	0.5099	0.7156	0.5094	0.7161
0.40	-0.0179	1.1896	-0.0109	1.1970	-0.0083	1.1988	-0.0074	1.1993
0.20	-0.4351	1.5505	-0.4144	1.5520	-0.4105	1.5509	-0.4094	1.5503
0.00	-0.5755	1.7429	-0.5650	1.7012	-0.5619	1.6856	-0.5605	1.6785

FREQUENCY = $.25\pi$ INCIDENCE ANGLE = 60°
ρ_-/ρ_+ = .67 β_-/β_+ = .50

RADIUS	5 ELEMENTS		10 ELEMENTS		20 ELEMENTS		EXACT	
1.00	0.4901	0.9179	0.4675	0.9666	0.4386	0.9894	0.4010	1.0249
0.80	0.8362	0.8943	0.8087	0.9093	0.7939	0.9137	0.7802	0.9179
0.60	1.1733	0.7559	1.1533	0.7486	1.1436	0.7454	1.1336	0.7437
0.40	1.4662	0.5419	1.4500	0.5268	1.4424	0.5202	1.4352	0.5143
0.20	1.6883	0.2828	1.6732	0.2647	1.6672	0.2553	1.6623	0.2450
0.00	1.8274	0.0057	1.8087	-0.0171	1.8021	-0.0293	1.7983	-0.042
-0.20	1.8405	-0.2720	1.8345	-0.2982	1.8331	-0.3128	1.8328	-0.3272
-0.40	1.7617	-0.5325	1.7614	-0.5586	1.7622	-0.5743	1.7635	-0.5915
-0.60	1.5915	-0.7488	1.5923	-0.7778	1.5939	-0.7943	1.5959	-0.8129
-0.80	1.3473	-0.8902	1.3435	-0.9350	1.3422	-0.9557	1.3428	-0.9766
-1.00	1.0580	-0.9172	1.0600	-0.9884	1.0455	-1.0233	1.0237	-1.0714

FREQUENCY = $.50\pi$ INCIDENCE ANGLE = 60°
ρ_-/ρ_+ = .67 β_-/β_+ = .50

RADIUS	5 ELEMENTS		10 ELEMENTS		20 ELEMENTS		EXACT	
1.00	-0.8350	0.2944	-0.8821	0.2526	-0.9628	0.2343	-1.0670	0.2269
0.80	-1.4689	1.4214	-1.5942	1.3720	-1.6468	1.3506	-1.6858	1.3345
0.60	-1.7754	2.2891	-1.8995	2.2567	-1.9418	2.2458	-1.9703	2.2405
0.40	-1.6925	2.6565	-1.7884	2.6518	-1.8177	2.6551	-1.8326	2.6631
0.20	-1.2614	2.4187	-1.3040	2.4297	-1.3125	2.4424	-1.3115	2.4603
0.00	-0.6277	1.6562	-0.5936	1.6530	-0.5765	1.6609	-0.5605	1.6785
-0.20	0.1049	0.4709	0.1593	0.5012	0.1824	0.5196	0.2013	0.5401
-0.40	0.6459	-0.6739	0.7182	-0.6506	0.7436	-0.6387	0.7595	-0.6270
-0.60	0.8930	-1.4961	0.9498	-1.4930	0.9670	-1.4922	0.9724	-1.4935
-0.80	0.8273	-1.7845	0.8337	-1.8073	0.8253	-1.8183	0.8131	-1.8324
-1.00	0.5360	-1.4826	0.5303	-1.5278	0.4653	-1.5522	0.3734	-1.5880

TABLE 2

FREQUENCY = .25π INCIDENCE ANGLE = 0°
ρ_-/ρ_+ = 1.50 β_-/β_+ = 2.00

RADIUS	5 ELEMENTS		10 ELEMENTS		20 ELEMENTS		EXACT	
1.00	0.8880	0.1357	0.8863	0.1339	0.8858	0.1336	0.8856	0.1336
0.80	0.8840	0.1380	0.8837	0.1361	0.8835	0.1358	0.8835	0.1358
0.60	0.8816	0.1397	0.8819	0.1378	0.8819	0.1375	0.8819	0.1375
0.40	0.8803	0.1410	0.8807	0.1391	0.8808	0.1388	0.8808	0.1388
0.20	0.8799	0.1418	0.8801	0.1398	0.8802	0.1395	0.8803	0.1395
0.00	0.8807	0.1421	0.8802	0.1401	0.8800	0.1398	0.8800	0.1398

FREQUENCY = .50π INCIDENCE ANGLE = 0°
ρ_-/ρ_+ = 1.50 β_-/β_+ = 2.00

RADIUS	5 ELEMENTS		10 ELEMENTS		20 ELEMENTS		EXACT	
1.00	0.5750	-0.0041	0.5729	-0.0052	0.5714	-0.0052	0.5706	-0.0051
0.80	0.5549	0.0065	0.5564	0.0046	0.5563	0.0044	0.5561	0.0044
0.60	0.5421	0.0148	0.5451	0.0124	0.5456	0.0121	0.5456	0.0120
0.40	0.5346	0.0206	0.5379	0.0181	0.5385	0.0177	0.5386	0.0176
0.20	0.5313	0.0240	0.5341	0.0215	0.5345	0.0211	0.5346	0.0210
0.00	0.5327	0.0249	0.5335	0.0225	0.5334	0.0222	0.5333	0.0222

FREQUENCY = .25π INCIDENCE ANGLE = 60°
ρ_-/ρ_+ = 1.50 β_-/β_+ = 2.00

RADIUS	5 ELEMENTS		10 ELEMENTS		20 ELEMENTS		EXACT	
1.00	0.8750	0.3010	0.8728	0.2936	0.8711	0.2907	0.8683	0.2894
0.80	0.8828	0.2775	0.8809	0.2694	0.8800	0.2652	0.8793	0.2610
0.60	0.8876	0.2510	0.8866	0.2418	0.8862	0.2368	0.8860	0.2315
0.40	0.8888	0.2228	0.8884	0.2127	0.8883	0.2073	0.8883	0.2016
0.20	0.8861	0.1936	0.8861	0.1828	0.8861	0.1770	0.8862	0.1637
0.00	0.8798	0.1644	0.8797	0.1525	0.8798	0.1461	0.8800	0.1398
-0.20	0.8691	0.1346	0.8690	0.1217	0.8691	0.1149	0.8693	0.1155
-0.40	0.8543	0.1039	0.8542	0.0905	0.8544	0.0833	0.8545	0.0763
-0.60	0.8357	0.0733	0.8355	0.0593	0.8355	0.0517	0.8355	0.0445
-0.80	0.8136	0.0436	0.8131	0.0286	0.8127	0.0205	0.8123	0.0122
-1.00	0.7889	0.0159	0.7885	0.0006	0.7875	-0.0087	0.7852	-0.0197

FREQUENCY = .50π INCIDENCE ANGLE = 60°
ρ_-/ρ_+ = 1.50 β_-/β_+ = 2.00

RADIUS	5 ELEMENTS		10 ELEMENTS		20 ELEMENTS		EXACT	
1.00	0.5944	0.2505	0.5941	0.2412	0.5930	0.2374	0.5883	0.2345
0.80	0.5973	0.2141	0.5991	0.2052	0.6009	0.2006	0.6032	0.1960
0.60	0.5914	0.1740	0.5969	0.1648	0.6006	0.1597	0.6047	0.1544
0.40	0.5753	0.1318	0.5834	0.1221	0.5881	0.1167	0.5932	0.1114
0.20	0.5486	0.0886	0.5582	0.0784	0.5635	0.0726	0.5692	0.0676
0.00	0.5126	0.0462	0.5221	0.0347	0.5275	0.0285	0.5333	0.0222
-0.20	0.4659	0.0038	0.4751	-0.0085	0.4805	-0.0150	0.4862	-0.0225
-0.40	0.4095	-0.0380	0.4183	-0.0505	0.4234	-0.0571	0.4288	-0.0641
-0.60	0.3450	-0.0777	0.3528	-0.0901	0.3573	-0.0968	0.3619	-0.1035
-0.80	0.2744	-0.1140	0.2803	-0.1264	0.2834	-0.1332	0.2865	-0.1403
-1.00	0.2006	-0.1454	0.2063	-0.1572	0.2071	-0.1645	0.2035	-0.1728

TABLE 3

FREQUENCY = 1.00π INCIDENCE ANGLE = 0°
ρ_-/ρ_+ = .67 β_-/β_+ = .50

RADIUS	5 ELEMENTS		10 ELEMENTS		20 ELEMENTS		EXACT	
1.00	2.6999	0.4648	1.5254	-0.0488	1.4244	-0.0224	1.4034	-0.0037
0.80	1.6630	0.9319	1.1053	0.4798	1.1905	0.5251	1.2366	0.5571
0.60	-0.1468	1.0585	-0.1258	0.7668	0.0042	0.7883	0.0607	0.8082
0.40	-1.4253	0.6158	-1.2088	0.4260	-1.1236	0.4102	-1.0900	0.4099
0.20	-1.8934	-0.1374	-1.6905	-0.2832	-1.6331	-0.3128	-1.6127	-0.3216
0.00	-2.3700	-0.5224	-1.8809	-0.6694	-1.7531	-0.6850	-1.7050	-0.6859

FREQUENCY = 1.00π INCIDENCE ANGLE = 60°
ρ_-/ρ_+ = .67 β_-/β_+ = .50

RADIUS	5 ELEMENTS		10 ELEMENTS		20 ELEMENTS		EXACT	
1.00	-1.1846	0.8262	-0.5685	0.4656	-0.6292	0.3569	-0.6971	0.1962
0.80	0.2819	0.5360	0.6327	0.5367	0.6107	0.6343	0.6021	0.7031
0.60	0.8912	-0.4740	0.9871	-0.3300	0.9713	-0.2135	0.9546	-0.1176
0.40	-0.3372	-1.4100	-0.3591	-1.5494	-0.3842	-1.5829	-0.4230	-1.5687
0.20	-2.1465	-1.4241	-1.9791	-1.7901	-1.9613	-1.9122	-1.9885	-1.9634
0.00	-2.3429	-0.4825	-1.8655	-0.6399	-1.7454	-0.6691	-1.7050	-0.6859
-0.20	-0.1101	0.6409	0.1856	0.8438	0.2620	0.9304	0.3148	0.9629
-0.40	1.9918	0.7143	1.8290	1.1634	1.7830	1.2555	1.7968	1.2719
-0.60	2.0383	-0.0450	1.3603	0.2858	1.1410	0.2157	1.0631	0.1366
-0.80	0.0976	-0.8284	-0.5279	-0.6316	-0.7781	-0.7939	-0.8945	-0.9124
-1.00	-1.7652	-0.9353	-1.4090	-0.7225	-1.5759	-0.7304	-1.6672	-0.6452

FREQUENCY = 1.00π INCIDENCE ANGLE = 0°
ρ_-/ρ_+ = 1.50 β_-/β_+ = 2.00

RADIUS	5 ELEMENTS		10 ELEMENTS		20 ELEMENTS		EXACT	
1.00	0.2325	-0.3428	0.2463	-0.3417	0.2431	-0.3396	0.2340	-0.3387
0.80	0.1802	-0.3015	0.1897	-0.3047	0.1908	-0.3045	0.1890	-0.3044
0.60	0.1525	-0.2679	0.1596	-0.2729	0.1612	-0.2736	0.1614	-0.2738
0.40	0.1390	-0.2440	0.1445	-0.2486	0.1458	-0.2496	0.1461	-0.2498
0.20	0.1333	-0.2307	0.1379	-0.2338	0.1387	-0.2344	0.1389	-0.2345
0.00	0.1343	-0.2314	0.1367	-0.2301	0.1369	-0.2295	0.1368	-0.2293

FREQUENCY = 1.00π INCIDENCE ANGLE = 60°
ρ_-/ρ_+ = 1.50 β_-/β_+ = 2.00

RADIUS	5 ELEMENTS		10 ELEMENTS		20 ELEMENTS		EXACT	
1.00	0.5797	0.0607	0.5966	0.0309	0.6050	0.0153	0.6036	-0.0107
0.80	0.5697	-0.0346	0.5835	-0.0500	0.5929	-0.0604	0.6013	-0.0729
0.60	0.5107	-0.1136	0.5260	-0.1194	0.5352	-0.1257	0.5448	-0.1340
0.40	0.4107	-0.1735	0.4253	-0.1752	0.4333	-0.1789	0.4416	-0.1843
0.20	0.2794	-0.2126	0.2902	-0.2126	0.2959	-0.2143	0.3016	-0.2172
0.00	0.1292	-0.2299	0.1324	-0.2288	0.1345	-0.2288	0.1368	-0.2293
-0.20	-0.0319	-0.2252	-0.0368	-0.2225	-0.0387	-0.2210	-0.0401	-0.2192
-0.40	-0.1937	-0.1995	-0.2058	-0.1951	-0.2115	-0.1917	-0.2167	-0.1874
-0.60	-0.3441	-0.1555	-0.3634	-0.1478	-0.3726	-0.1424	-0.3819	-0.1355
-0.80	-0.4743	-0.0975	-0.4994	-0.0835	-0.5119	-0.0751	-0.5265	-0.0646
-1.00	-0.5787	-0.0319	-0.6017	-0.0096	-0.6170	0.0017	-0.6438	0.0248

TABLE 4

FREQUENCY = .7639π INCIDENCE ANGLE = 0°
ρ_-/ρ_+ = .67 β_-/β_+ = .50

RADIUS	5 ELEMENTS		10 ELEMENTS		20 ELEMENTS		EXACT	
1.00	1.2885	0.7557	0.6008	0.5329	0.4699	0.4502	0.4322	0.4191
0.80	1.7459	2.1061	0.8836	2.2977	0.6170	2.2960	0.5240	2.2857
0.60	1.5807	2.6912	0.7885	3.1921	0.4843	3.2716	0.3731	3.2874
0.40	0.8648	2.4518	0.3467	2.9921	0.1232	3.1032	0.0393	3.1337
0.20	0.0481	1.8992	-0.1567	2.2912	-0.2559	2.3843	-0.2947	2.4139
0.00	-0.2416	1.9265	-0.3773	2.0175	-0.4188	2.0315	-0.4336	2.0327

FREQUENCY = .7639π INCIDENCE ANGLE = 60°
ρ_-/ρ_+ = .67 β_-/β_+ = .50

RADIUS	5 ELEMENTS		10 ELEMENTS		20 ELEMENTS		EXACT	
1.00	0.1824	-0.0520	0.5011	-0.1965	0.5007	-0.2320	0.4419	-0.3117
0.80	-0.7646	-1.7509	-0.2770	-2.1918	-0.0889	-2.3035	0.0121	-2.3884
0.60	-1.5071	-2.4883	-1.0001	-2.9706	-0.7766	-3.1009	-0.6631	-3.1864
0.40	-1.6157	-1.7282	-1.3282	-1.9749	-1.1970	-2.0368	-1.1350	-2.0708
0.20	-1.0528	0.0848	-1.0578	0.1435	-1.0509	0.1761	-1.0525	0.2039
0.00	-0.2289	1.8186	-0.3652	1.9493	-0.4119	1.9956	-0.4336	2.0327
-0.20	0.4553	2.2009	0.3285	2.1436	0.3190	2.1440	0.3292	2.1542
-0.40	0.4445	1.1071	0.5636	0.7313	0.6704	0.6352	0.7445	0.5848
-0.60	-0.0427	-0.6358	0.2801	-1.1881	0.4672	-1.3377	0.5750	-1.4226
-0.80	-0.5517	-1.8617	-0.2191	-2.2071	-0.0723	-2.2967	0.0011	-2.3533
-1.00	-0.6603	-1.8214	-0.4074	-1.6962	-0.4329	-1.6356	-0.5248	-1.5736

FREQUENCY = .7480π INCIDENCE ANGLE = 0°
ρ_-/ρ_+ = .67 β_-/β_+ = .50

RADIUS	5 ELEMENTS		10 ELEMENTS		20 ELEMENTS		EXACT	
1.00	1.4014	0.7395	0.7765	0.6111	0.6452	0.5666	0.6046	0.5495
0.80	1.8842	1.8155	1.2719	2.0665	1.0756	2.1275	1.0040	2.1448
0.60	1.6919	2.2815	1.2144	2.7801	1.0143	2.9148	0.9374	2.9573
0.40	0.9262	2.1212	0.6445	2.6158	0.5058	2.7528	0.4511	2.7974
0.20	0.0697	1.7310	-0.0322	2.0636	-0.0911	2.1530	-0.1149	2.1824
0.00	-0.2267	1.7944	-0.3242	1.8570	-0.3524	1.8639	-0.3617	1.8621

FREQUENCY = .7480π INCIDENCE ANGLE = 60°
ρ_-/ρ_+ = .67 β_-/β_+ = .50

RADIUS	5 ELEMENTS		10 ELEMENTS		20 ELEMENTS		EXACT	
1.00	0.1479	-0.0826	0.4103	-0.2591	0.4024	-0.3148	0.3392	-0.4012
0.80	-0.8543	-1.6355	-0.5289	-2.0535	-0.3988	-2.1883	-0.3282	-2.2870
0.60	-1.6098	-2.3271	-1.2809	-2.7524	-1.1303	-2.8988	-1.0544	-2.9958
0.40	-1.6969	-1.6638	-1.5085	-1.8667	-1.4222	-1.9385	-1.3828	-1.9807
0.20	-1.0934	-0.0005	-1.0945	0.0607	-1.0901	0.0850	-1.0925	0.1067
0.00	-0.2196	1.6905	-0.3166	1.7924	-0.3480	1.8299	-0.3617	1.8621
-0.20	0.5098	2.2102	0.3617	2.1399	0.3431	2.1275	0.3491	2.1296
-0.40	0.5258	1.3568	0.5114	0.9991	0.5652	0.8854	0.6136	0.8226
-0.60	0.0252	-0.2368	0.1475	-0.7638	0.2579	-0.9382	0.3291	-1.0405
-0.80	-0.5396	-1.5412	-0.3682	-1.9033	-0.2769	-2.0252	-0.2283	-2.1019
-1.00	-0.7173	-1.7779	-0.5017	-1.7239	-0.5288	-1.6946	-0.6191	-1.6529

REFERENCES

1. Bielak, J. and MacCamy, R. C.: An Exterior Interface Problem in Two-Dimensional Elastodynamics. Quarterly of Applied Mathematics, Vol. 41, 1983, pp. 143-160.

2. Trifunac, M. D.: Surface Motion of a Semi-Cylindrical Alluvial Valley for Incident Plane SH Waves. Bulletin of the Seismological Society of America, Vol. 61, 1971, pp. 1755-1770.

3. Zienkiewicz, O. C., Kelly, D. W., and Bettess, P.: Marriage a'la mode — The Best of Both Worlds (Finite Elements and Boundary Integrals). Energy Methods in Finite Element Analyses, Edited by Glowinski, R., Rodin, E. Y., and Zienkiewicz, O. C., pp. 81-107.

4. Harris, C. G. and Evans, W. A. B.: Extension of Numerical Quadrature Formulae to Cater for End Point Singular Behaviours over Finite Intervals. International Journal of Computer Methods, Vol. 6, 1977, pp. 219-227.

5. Pao, Y. H. and Mow, C. C.: Diffraction of Elastic Waves and Dynamic Stress Concentrations. Crane, Russak, New York, 1971.

APPENDIX A

Substituting the shape functions [equation (23)] into equation (22) is done after grouping certain terms.

1st Term

$$-\int_{\Omega} \{\mu^{-}(w_{x_1}\ \delta w_{x_1} + w_{x_2}\ \delta w_{x_2}) - \omega^2\ \rho^{-}\ w\ \delta w\}\ d\underline{x}$$

$$= -\int_{\Omega} \Big\{\{\delta w\}^T\ [N_{x_1}]\ \mu^{-}\ [N_{x_1}]^T\ \{w\} + \{\delta w\}^T\ [N_{x_2}]\ \mu^{-}[N_{x_2}]^T\{w\}$$

$$- \omega^2\ \{\delta w\}^T\ [N]\ \rho^{-}[N]^T\ \{w\}\Big\}\ d\underline{x}$$

Let $[K] = -\int_{\Omega} \{[N_{x_1}]\ \mu^{-}[N_{x_1}]^T + [N_{x_2}]\ \mu^{-}[N_{x_2}]^T - \omega^2\ [N]^T\ \rho^{-}[N]\}\ d\underline{x}$

then the 1st term becomes

$$\{\delta w\}^T\ [K]\ \{w\} \quad .$$

2nd Term

$$1/2 \int_{\Gamma} w^{-}(\underline{x})\ \delta\mu^{-}\ w_n^{-}(\underline{x})\ dS = 1/2 \int_{\Gamma} \{\delta\lambda\}^T\ [Q]\ [N_{\Gamma}]^T\ \{w_{\Gamma}\}\ dS$$

Let $[A] = 1/2 \int_{\Gamma} [Q]\ [N_{\Gamma}]^T\ dS$

then the 2nd term becomes

$$\{\delta\lambda\}^T\ [A]\ \{w_{\Gamma}\} \quad .$$

3rd Term

$$1/2 \int_\Gamma \left\{ 1/2\ \mu^+ w^-(\underline{x}) + \mu^+ \int_\Gamma \frac{\partial G(\underline{x},\underline{y})}{\partial n_{\underline{y}}} w^-(\underline{y})\, dS\underline{y} \right\} \delta\phi(\underline{x})\, dS$$

$$= 1/2 \int_\Gamma \left\{ 1/2\ \mu^+ \{\delta\phi\}^T [Q]\, [N_\Gamma]^T \{w_\Gamma\}\, dS \right.$$

$$\left. + \mu^+ \int_\Gamma \{\delta\phi\}^T [Q] \frac{\partial G(\underline{x},\underline{y})}{\partial n\underline{y}} [N_\Gamma]^T \{w_\Gamma\}\, dS\underline{y} \right\} dS$$

Let $[B] = 1/2\ \mu^+ \left\{ 1/2 \int_\Gamma [Q]\, [N_\Gamma]^T\, dS + \int_\Gamma \int_\Gamma [Q] \left[\frac{\partial G(\underline{x},\underline{y})}{\partial n\underline{y}} \right] [N_\Gamma]^T\, dS_y\, dS \right\}$

then the 3rd term becomes

$$\{\delta\phi\}^T [B]\, \{w_\Gamma\} \quad .$$

4th Term

$$- 1/2 \int_\Gamma \int_\Gamma G(\underline{x},\underline{y})\ \mu^-\ w_n^-(\underline{y})\ \delta\phi\ dS\underline{y}\ dS$$

$$= 1/2 \int_\Gamma \int_\Gamma \{\delta\phi\}\, [Q]\, [G(\underline{x},\underline{y})]\, [Q]^T \{\lambda\}\, dS\underline{y}\ dS$$

Let $[C] = - 1/2 \int_\Gamma \int_\Gamma [Q]\, [G(\underline{x},\underline{y})]\, [Q]^T\, dS\underline{y}\ dS$

then the 4th term becomes

$$\{\delta\phi\}\, [C]\, \{\lambda\} \quad .$$

5th Term

$$\int_\Gamma \mu^-\ w_n^-\ \delta w\ dS - 1/2 \int_\Gamma \mu^-\ w_n^-\ \delta w^-\ dS = \int_\Gamma \{\delta w_\Gamma\}^T [N_\Gamma]\, [Q]^T \{\lambda\}\, dS$$

$$- 1/2 \int_\Gamma \{\delta w_\Gamma\}^T [N_\Gamma]\, [Q]^T \{\lambda\}\, dS$$

Let $[A]^T = 1/2 \int_\Gamma [N_\Gamma]\ [Q]^T\ dS$

then the 5th term becomes

$$\{\delta w_\Gamma\}^T\ [A]^T\ \{\lambda\} \quad .$$

<u>6th Term</u>

$$1/2 \int_\Gamma \left\{ 1/2\ \mu^+\ \phi(\underline{x}) + \mu^+ \int_\Gamma \frac{\partial G(\underline{x},\underline{y})}{\partial n\underline{x}}\ \phi(\underline{y})\ dS\underline{y} \right\}\ \delta w^-\ dS$$

$$= 1/2 \int_\Gamma \left\{ 1/2\ \mu^+\ \{\delta w_\Gamma\}^+\ [N_\Gamma]\ [Q]^T\ \{\phi\} \right.$$

$$\left. + \mu^+ \int_\Gamma \{\delta w_\Gamma\}^T\ [N_\Gamma] \left[\frac{\partial G(\underline{x},\underline{y})}{\partial n\underline{x}} \right] [Q]^T\ \{\phi\}\ dS\underline{y} \right\} dS$$

Let $[B]^T = 1/2\ \mu^+ \left\{ 1/2 \int_\Gamma [N_\Gamma]\ [Q]^T\ dS + \int_\Gamma\ {}_\Gamma\ [N_\Gamma] \left[\frac{\partial G(\underline{x},\underline{y})}{\partial n\underline{x}} \right] [Q]^T\ dS\underline{y}\ dS \right\}$

then the 6th term becomes

$$\{\delta w_\Gamma\}^T\ [B]^T\ \{\phi\} \quad .$$

<u>7th Term</u>

$$- 1/2 \int_\Gamma \int_\Gamma G(\underline{x},\underline{y})\ \phi(\underline{y})\ \delta\mu^-\ w_n^-(\underline{x})\ dS\underline{y}\ dS$$

$$= - 1/2 \int_\Gamma \int_\Gamma \{\delta\lambda\}^T\ [Q]\ [G(\underline{x},\underline{y})]\ [Q]^T\ \{\phi\}\ dS\underline{y}\ dS$$

Let $[C]^T = - 1/2 \int_\Gamma \int_\Gamma [Q]\ [G(\underline{x},\underline{y})]\ [Q]^T\ dS\underline{y}\ dS$

then the 4th term becomes

$$\{\delta\lambda\}^T\ [C]^T\ \{\phi\} \quad .$$

8th Term

$$- 1/2 \int_\Gamma w^\circ(x) \; \delta\mu^- \; w_n^-(\underline{x}) \; dS = - 1/2 \int_\Gamma \{\delta\lambda\}^T \; [Q] \; w^\circ \; dS$$

Let $\{f\lambda\} = 1/2 \int_\Gamma [Q] \; w^\circ \; dS$

then the 8th term becomes

$$- \{\delta\lambda\}^T \; \{f\lambda\} \quad .$$

9th Term

$$1/2 \int_\Gamma \mu^+ \; w_n^\circ \; \delta w^- \; dS = 1/2 \; \mu^+ \int_\Gamma \{\delta w_\Gamma\}^T \; [N_\Gamma] \; w_n^\circ \; dS$$

Let $\{f_\Gamma\} = - 1/2 \; \mu^+ \int_\Gamma [N_\Gamma] \; w_n^\circ \; dS$

then the 9th term becomes

$$- \{\delta w_\Gamma\}^T \; \{f_\Gamma\} \quad .$$

10th Term

$$- 1/2 \; \mu^+ \int_\Gamma w^\circ \; \delta\phi \; dS = - 1/2 \; \mu^+ \int_\Gamma \{\delta\phi\}^T \; [Q] \; w^\circ \; dS$$

Let $\{f_\phi\} = \mu^+ \; 1/2 \int_\Gamma [Q] \; w^\circ \; dS$

then the 10th term becomes

$$- \{\delta\phi\}^T \; \{f_\phi\} \quad .$$

Equation (22) now becomes:

$$\{\delta w\}^T ([K]\ \{w\}) + \{\delta w_\Gamma\}^T ([B]\ \{\phi\} + [A]^T\ \{\lambda\} - \{f_\Gamma\})$$

$$+ \{\delta\lambda\}^T ([A]\ \{w_\Gamma\} + [C]^T\ \{\phi\} - \{f\lambda\}) + \{\delta\phi\}^T ([B]\ \{w_\Gamma\}$$

$$+ [C]\ \{\lambda\} - \{f\phi\}) = 0 \quad .$$

For arbitrary $\{\delta w\}$, $\{\delta w_\Gamma\}$, $\{\delta\lambda\}$, and $\{\delta\phi\}$; quantities multiplying them must each be zero.

$$\begin{bmatrix} K_{\Omega\Omega} & K_{\Omega\Gamma} & 0 & 0 \\ K_{\Gamma\Omega} & K_{\Gamma\Gamma} & A^T & B^T \\ 0 & A & 0 & C^T \\ 0 & B & C & 0 \end{bmatrix} \begin{Bmatrix} w_\Omega \\ w_\Gamma \\ \lambda \\ \phi \end{Bmatrix} = \begin{Bmatrix} 0 \\ f_\Gamma \\ f_\lambda \\ f_\phi \end{Bmatrix}$$

APPENDIX B. PROGRAM LISTING

```
C INTRODUCTION
C ------------
C
C This program solves, using finite element techniques, a soil structure
C interaction problem defined in polar coordinates. The structure is a
C semicylinderical deposit of soil at the surface of an elastic halfspace.
C The properties of this deposit can be diferent from those of the
C halfspace and can also represent a linear change of shear modulus with
C depth. The forcing function of this problem is an incoming antiplane
C shear wave. Th computed results are the steady state antiplane
C displacements. The interior deposit is represented by finite element
C techniques while the exterior halfspace is represented at the boundary
C of the deposit bye the use of Green's functions. Thus only finite
C element storage requirements are for the deposite and no additional
C finite element manipulations are necessary.
C
C PROGRAM STRUCTURE
C -----------------
C
C The program is stuctured with a main driving program with various
C levels of subroutines. A schematic of this structure is shown here.
C
C    LEVEL 0      LEVEL 1      LEVEL 2      LEVEL 3      LEVEL 4
C    -------      -------      -------      -------      -------
C
C            |--- MESH
C            |              |-- RECT
C            |--- ASSM ----|
C            |              |-- TRI
C            |
C            |              |-- GAMMA
C            |--- FORCE ---|
C            |              |-- PHILAM
C            |
C    MAIN ---|              |-- BMAT
C            |              |             |--- NORM  --|
C            |              |-- AMAT ----|             |--- DERIV
C            |              |             |--- NORM2 --|
C            |--- HSPACE --|-- ATEST
C            |              |             |--- DIAG  --|
C            |              |-- CMAT ----|             |--- FUNC
C            |              |             |--- DIAG2 --|
C            |              |-- CTEST
C            |--- KTEST
C
C A description of what each subroutine does is presented at the
C beginning of that subroutine.
C
C INPUT DATA
C ----- ----
C
C The input data for this program is contained in two data files.
C FOR24.DAT contains the Gauss integration information and should not be
C changed without a series of tests to check convergence. FOR25.DAT
C contains all the other information necessary for the operation of this
C program. The unit numbers for these files are 24 and 25 respectively.
C An example of the data contained in this file is shown here:
C
C
C AMAT         |
```

```
C CMAT          |
C HSPACE        |--- debug switches (subroutine names)
C MESH _________|
C START ---------- signals end of debug switches (must always be used)
C UNIFORM -------- radial division method (UNIFORM or GRADIENT)
C 5.76 0.0 0.5 0.5 1.0 1.0 0.5 -------------------|-frequency squared
C 10 10 ------------------------------------|      | incidence angle (radians)
C 3.14159 1.0 1 ----------------------|     |      | shear modulus at depth
C 1.0 1.0 1.0 1.0 1.0 -----------|    |     |      | shear modulus at surface
C                               |    |     |      | density of deposit
C                               |    |     |      | shear modulus of halfspace
C                               |    |     |      |_geometric shape factor (.5)
C                               |    |     |
C                               |    |     |_____|-number of radial divisions
C                               |    |           |_number of angular divisions
C                               |    |
C                               |    |           |-angular sweep (radians)
C                               |    |-----------| radial range
C                               |                |-number of boundary elements
C                               |                   per angular element
C                               |
C                               |_______________ boudary node displacements
C                                                  if KTEST is to be run.
C
C In use this file would look like this:
C
C AMAT
C CMAT
C HSPACE
C MESH
C START
C UNIFORM
C 5.76 0.0 0.5 0.5 1.0 1.0 0.5
C 10 10
C 3.14159 1.0 1
C 1.0 1.0 1.0 1.0 1.0
C
C
C OUTPUT DATA
C ------ ----
C All output is directed to the unit number 26.
C
C
C MAIN PROGRAM
C ------------
C
C The main or driving program has four major duties. The first is to
C read in the data which must be supplied by the user. A more detailed
C description of this data is presented in the "Input Data" section. The
C second duty of this subprogram is to set the bollion values for the
C various debug and method options available in this program. The
C third duty is the definition of matrix sizes. Each matrix is defined
C here using an actual number which represents the maximum size
C allowed. This limits the size of structure whcih can be analysed.
C Using the data which is read in the actual sizes of the matrices are
C calculated. These values are then used throughout the program.
C Computed matrix sizes are passed in COMMON to all subroutines while
C the matrices themselves are passed as needed through the arguments of
C the various subroutines. An  expansion of the size of structure the
C program can handle can be done
```

```
C from this subroutine alone since this is the only place where these
C matrices are defined using actual numbers.
C However, if the size of the individual elements used is changed then
C the subroutines ASSM, RECT, and TRI must also be changed and their
C accompanying matrices.
C The final duty of this subprogram is to control the execution of
C the entire program. This is done by calling various subroutines which
C do specific jobs such as constructing particular matrices or solving
C the equations.
C
C Declare in COMMON all variables required by more
C than one C subroutine.  These variables are divided into four
C catagories:
C       1. SIZE This common block contains variables which define
C               array dimensions.
C       2. PROB This common block contains variables which define
C               problem parameters.
C       3. MAP  This common block contains variables which define
C               the finite element mesh.
C       4. dbug This common block contains variables which define
C               which debug switches are desired.
C Arrays are not passed through COMMON but passed as arguments to the
C subroutines.
C
      COMMON  /SIZE/ NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,
     1        NUMEQN,LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
      INTEGER NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,NUMEQN,
     1        LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
      COMMON  /PROB/ W2,GA,GB,RO,EXTG,ALPHA,THETAO
      REAL    W2,GA,GB,RO,EXTG,ALPHA,THETAO
      COMMON  /MAP/ SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2,RATIO,GRADNT
      INTEGER RATIO
      REAL    SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2
      LOGICAL GRADNT
      common  /dbug/ dmain,dmesh,dassm,dktest,dforce,
     1        dhspac,drect,dtri,dgamma,dfilam,damat,dbmat,
     1        dcmat,dnorm,ddiag,ddiag2,dfunc,dderiv,dctest,
     1        datest
      logical  dmain,dmesh,dassm,dktest,dforce,dhspac,
     1        drect,dtri,dgamma,dfilam,damat,dbmat,dcmat,dnorm,
     1        ddiag,ddiag2,dfunc,dderiv,dctest,datest
C
C Declare and dimension arrays necessary. Only those arrays which are
C dependent on structure size are dimensioned here. This is the only
C location where actual numbers are used for these arrays. An increase
C of structure size can be handled from here alone. Arrays which are
C dependent on element size are dimensioned in RECT, TRI, and ASSM.
C
      INTEGER GEOM(400,4)
      REAL    PROP(400,5),K(421,45),OUTPUT(421),KOG(400,21),
     1        B(400,21),KOO(400,45),XL(400,23),WG(21),WO(400),
     2        EXACT(400),BMATRX(20,2)
      COMPLEX FG(21),FPHI(21),C(20,20),CTEMP(20),FD(21),W(21),
     1        CINVER(20,20),ATRANS(21,20),ATEMP(2,21),ATEMP2(80,40),
     1        ATEMP3(80,40),D(21,21),TEMPD(21,21),WHOLE(21,21),
     1        LAM(20)
C
C Declare variables required in this main program.
C
      LOGICAL OK
```

```
      INTEGER debug,METHOD
      REAL    FREQ,G
C
C Define whether debug is required or not.
C
C    Initialize.
C
      dmain = .false.
      dmesh = .false.
      dassm = .false.
      dktest = .false.
      dforce = .false.
      dhspac = .false.
      dbmat  = .false.
      drect  = .false.
      dtri   = .false.
      dgamma = .false.
      dfilam = .false.
      damat  = .false.
      dbmat  = .false.
      dcmat  = .false.
      dnorm  = .false.
      ddiag  = .false.
      ddiag2 = .false.
      dfunc  = .false.
      dderiv = .false.
      dctest = .false.
      datest = .false.
C
C    Read in debugs desired.
C
    5 read(25,7)debug
    7 format(a5)
C
C    Echo debugs desired.
C
      write(5,8)debug
    8 format(2X,a5)
C
C    Set switches for debugs desired. 'START' indicates no more
C    switches will be requested.
C
      if(debug.eq.'MAIN')dmain = .true.
      if(debug.eq.'MESH')dmesh = .true.
      if(debug.eq.'ASSM')dassm = .true.
      if(debug.eq.'KTEST')dktest = .true.
      if(debug.eq.'FORCE')dforce = .true.
      if(debug.eq.'HSPAC')dhspac = .true.
      if(debug.eq.'RECT')drect = .true.
      if(debug.eq.'TRI')dtri = .true.
      if(debug.eq.'GAMMA')dgamma = .true.
      if(debug.eq.'PHILA')dfilam = .true.
      if(debug.eq.'AMAT')damat = .true.
      if(debug.eq.'BMAT')dbmat = .true.
      if(debug.eq.'CMAT')dcmat = .true.
      if(debug.eq.'NORM')dnorm = .true.
      if(debug.eq.'DIAG')ddiag = .true.
      if(debug.eq.'DIAG2')ddiag2 = .true.
      if(debug.eq.'FUNC')dfunc = .true.
      if(debug.eq.'DERIV')dderiv = .true.
```

```
      if(debug.eq.'CTEST')dctest = .true.
      if(debug.eq.'ATEST')datest = .true.
      if(debug.eq.'START')goto 10
      goto 5
   10 continue
C
C Define desired radial division method. If GRADIENT is desired radial
C divisions are chosen to balance element length and width. If UNIFORM
C is requested radial divisions are equal segments.
C
C    Initialize.
C
      GRADNT = .FALSE.
C
C    Read in desired method.
C
      READ(25,11)METHOD
   11 FORMAT(1A5)
C
C    Echo method desired.
C
      WRITE(5,12)METHOD
   12 FORMAT(2X,1A5)
C
C    Set switch.
C
      IF(METHOD.EQ.'GRADI')GRADNT = .TRUE.
C
C Read in incident wave information and soil parameters.
C
      READ(25,*)W2,THETAO,G,GB,RO,EXTG,ALPHA
C
C Compute frequency from input squared frequency.
C
      FREQ = SQRT(W2)
C
C Read in desired number of angular and radial divisions.
C
      READ(25,*)NUMANG,NUMRAD
C
C Read in angle sweep of the structure (0-?radians) and the radius
C range of the structure (0-?).
C
      READ(25,*)SWEEP,RANGE,RATIO
C
C Calculate the rate of change of soil stiffness.
C
      GA = (G-GB)/RANGE
C
C Calculate array dimensions and make checks for program capacity.
C
      OK = .TRUE.
C#    IF(RATIO.GE.1.AND.RATIO.LE.5) GOTO 15
C#    WRITE(5,800)
C#    OK = .FALSE.
   15 BNDELM = NUMANG/RATIO
C#    I = MOD(NUMANG,RATIO)
C#    IF(I.EQ.0.AND.BNDELM.LE.15) GOTO 16
C#    WRITE(5,150)
C#    OK = .FALSE.
```

```
   16   BCOLS = RATIO + 1
C#      IF(BCOLS.LE.6) GOTO 17
C#      WRITE(5,150)
C#      OK = .FALSE.
   17   AWIDE = BNDELM*2
C#      IF(AWIDE.LE.30) GOTO 18
C#      WRITE(5,150)
C#      OK = .FALSE.
   18   ALENG = BNDELM*RATIO*4
C#      IF(ALENG.LE.300)GOTO 19
C#      WRITE(5,150)
C#      OK = .FALSE.
   19   ATMPL = RATIO*2
C#      IF(ATMPL.LE.10)GOTO 20
C#      WRITE(5,150)
C#      OK = .FALSE.
   20  NUMELM = NUMANG*NUMRAD
C#      IF(NUMELM.LE.225) GOTO 21
C#      WRITE(5,100)
C#      OK = .FALSE.
   21   INTNOD = NUMANG+1
C#      IF(INTNOD.LE.16) GOTO 30
C#      WRITE(5,200)
C#      OK = .FALSE.
   30   NUMNOD = (INTNOD)*NUMRAD+1
C#      IF(NUMNOD.LE.241) GOTO 40
C#      WRITE(5,300)
C#      OK = .FALSE.
   40   CODIAG = NUMANG + 2
C#      IF(CODIAG.LE.17) GOTO 50
C#      WRITE(5,400)
C#      OK = .FALSE.
   50   NUMEQN = NUMNOD-INTNOD
C#      IF(NUMEQN.LE.225) GOTO 60
C#      WRITE(5,500)
C#      OK = .FALSE.
   60   LENGTH = NUMNOD*(NUMNOD+1)/2
C#      IF(LENGTH.LE.29161) GOTO 70
C#      WRITE(5,600)
C#      OK = .FALSE.
   70   WIDTH  = 2*CODIAG+1
        SPACE  = CODIAG+1
C
C If checks for program capacity are allright continue otherwise abort
C program execution.
C
        IF(OK) GOTO 80
        WRITE(5,700)
        STOP
C
C Generate finite element mesh.
C
   80   CALL MESH(PROP,GEOM)
C
C Assemble structure stiffness matrix.
C
        CALL ASSM(PROP,GEOM,K,OUTPUT)
C
C Determine applied forces.
C
```

```
      CALL FORCE(FG,FPHI)
C
C Generate matrix defining halfspace.
C
      CALL HSPACE(ATRANS,ATEMP,ATEMP2,ATEMP3,BMATRX,
     1 C,CTEMP,FG,FPHI,FD,OUTPUT,CINVER,TEMPD,D)
C
C I test of K matrics is desired call KTEST subroutine.
C
      IF(DKTEST)CALL KTEST(K,WO,WG,KOG,B,KOO,XL,OUTPUT,EXACT)
C
C Solve for displacements.
C
      CALL SOLVE(K,D,FG,FD,KOG,KOO,B,WHOLE,SLTN,W,XL,OUTPUT,
     1       LAM,TEMPD,CINVER,FPHI)
C
C Output computed values.
C
      CALL OUT(SLTN,W,FG,FD,B,LAM)
C
C Output information supplied to program by user.
C
      WRITE(26,1000) SWEEP,NUMANG,RANGE,NUMRAD,ALPHA,RATIO
      IF(GRADNT)WRITE(26,1500)
      IF(.NOT.GRADNT)WRITE(26,1600)
      WRITE(26,2000) W2,FREQ,THETAO,GA,GB,RO,EXTG
C
C Format statements for error messages.
C
  100 FORMAT(2X,'Number of elements is more than can be handled.')
  150 FORMAT(2X,'Number of boundary elements is more than
     1 can be handled. Ratio must be corrected.')
  200 FORMAT(2X,'Number of boundary nodes is more than can be
     1 handled.')
  300 FORMAT(2X,'Number of nodes is more than can be handled.')
  400 FORMAT(2X,'Band width is greater than can be handled.')
  500 FORMAT(2X,'Number of interior nodes is more than can be
     1 handled.')
  600 FORMAT(2X,'Size of structure stiffness matrix is greater than
     1 can be handled.')
  700 FORMAT(//,2X,'Execution aborted. Storage capacity exceeded.')
  800 FORMAT(2X,'Ratio must be greater than or equal to 1 and less
     1 than or equal to 5.',
     1 /,2x,'Execution aborted.')
C
C Formats for output of information supplied by user.
C
 1000 FORMAT(//2X,'************************* INPUT INFORMATION',
     1 ' *************************',///,
     1 2X,'MESH INFORMATION',/,2X,'---- -----------',//,
     1 2X,'SWEEP = ',F11.6,2X,'NUMBER OF ANGLE DIVISIONS  = ',I5,//,
     1 2X,'RANGE = ',F11.6,2X,'NUMBER OF RADIAL DIVISIONS = ',I5,//,
     1 2X,6X,'SHAPE CONSTANT = ',F11.6,//,
     1 2X,16X,'INTERNAL BOUNDARY ELEMENTS',/,
     1 2X,6X,'RATIO OF  -------------------------  =',I5,/,
     1 2X,16X,'EXTERNAL BOUNDARY ELEMENTS',//)
 1500 FORMAT(2X,6X,'RADIAL DIVISIONS ARE GRADUATED',//)
 1600 FORMAT(2X,6X,'RADIAL DIVISIONS ARE UNIFORM',/,29X,'---------',/)
 2000 FORMAT(2X,'FREE FIELD WAVE INFORMATION',/,2X,'---- ----- ----',
     1 ' -----------',//,
```

```
     1  2X,'FREQUENCY SQUARED = ',F11.6,2X,'FREQUENCY = ',F11.6,//,
     1  2X,'ANGLE OF INCIDENCE FROM DOWNWARD VERTICAL = ',F11.6,/,
     1  2X,'    (+ CLOCKWISE,- COUNTERCOLOCKWISE)       ',///,
     1  2X,'MATERIAL PROPERTIES',/,2X,'-------- ----------',//,
     1  2X,'INTERNAL SHEAR MODULUS    = ',F11.6,' Y + ',F11.6,//,
     1  2X,'INTERNAL MATERIAL DENSITY = ',F11.6,//,
     1  2X,'EXTERNAL SHEAR MODULUS    = ',F11.6,//)
C
C Halt program execution.
C
      STOP
      END
C*****************************************************************************
C
C*****************************************************************************
C SUBROUTINE MESH
C ---------------
C
C The purpose of the MESH subprogram is to generate the finite element
C mesh required with a minimum of input by the user. It will generate a
C mesh for a circular structure defined in polar coordinates.
C There are only four
C required inputs. The first is the angular sweep of the entire
C structure in radians. The second is the maximum radius of the
C structure. The two remaining inputs are simply the number of divisions
C of these two structure dimensions that are desired. MESH will generate
C two tables. The first is an incidence table for each element. The
C second is a table of element coordinates and element type. These two
C tables, GEOM and PROP are passed back to the main program as they are
C used by a number of other subroutines in defining the stiffness
C matrix.
C
      SUBROUTINE MESH(PROP,GEOM)
C
C Declare in COMMON all variables required by more than one
C subroutine.  These variables are divided into four catagories:
C       1. SIZE This common block contains variables which define
C               array dimensions.
C       2. PROB This common block contains variables which define
C               problem parameters.
C       3. MAP  This common block contains variables which define
C               the finite element mesh.
C       4. dbug This common block contains variables which define
C               which debug switches are desired.
C Arrays are not passed through COMMON but passed as arguments to the
C subroutines.
C
      COMMON  /SIZE/ NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,
     1        NUMEQN,LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
      INTEGER NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,NUMEQN,
     1        LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
      COMMON  /PROB/ W2,GA,GB,RO,EXTG,ALPHA,THETAO
      REAL    W2,GA,GB,RO,EXTG,ALPHA,THETAO
      COMMON  /MAP/ SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2,RATIO,GRADNT
      INTEGER RATIO
      REAL    SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2
      LOGICAL GRADNT
      common  /dbug/ dmain,dmesh,dassm,dktest,dforce,
     1        dhspac,drect,dtri,dgamma,dfilam,damat,dbmat,
     1        dcmat,dnorm,ddiag,ddiag2,dfunc,dderiv,dctest,datest
```

```
      logical  dmain,dmesh,dassm,dktest,dforce,dhspac,
     1         drect,dtri,dgamma,dfilam,damat,dbmat,dcmat,dnorm,
     1         ddiag,ddiag2,dfunc,dderiv,dctest,datest
C
C Declare and dimension arrays required in this subroutine.
C
      INTEGER GEOM(NUMELM,4)
      REAL    PROP(NUMELM,5)
C
C Declare variables required only within this subroutine.
C
      INTEGER I,J,ELM
      REAL    RAD,THETA
C
C Calculate mesh sizes.
C
      DANG = SWEEP/FLOAT(NUMANG)
      DRAD = RANGE/FLOAT(NUMRAD)
      BDANG1 = DANG*RATIO
      BDANG2 = SWEEP - (BNDELM-1)*BDANG1
C
C Iterate over structure defining element coordinates and element-node
C incidences.
C
C     Iterate over number of angle divisions.
C
      DO 20 I=0,NUMANG-1
C
C             Define angle: 0 to sweep minus one angle mesh size.
C
              THETA = I*DANG
C
C             Iterate over number of radius divisions.
C
              DO 10 J=0,NUMRAD-1
C
C                     Label element number.
C
                      ELM=I*NUMRAD+J+1
C
C                     Define element coordinates and types:
C
C                     1. if radial divisions are to be uniform or
C
                      IF(GRADNT) GOTO 7
                      PROP(ELM,1)=(RANGE-J*DRAD)-DRAD
                      PROP(ELM,2)=RANGE-J*DRAD
                      GOTO 8
C
C                     2. if radial divisions are to be graduated.
C
    7                 PROP(ELM,1)=((1-DANG/2.0)/(1+DANG/2.0))**(J+1)
     1                *RANGE
                      IF(J.EQ.NUMRAD-1)PROP(ELM,1)=0.0
                      PROP(ELM,2)=((1-DANG/2.0)/(1+DANG/2.0))**J
     1                *RANGE
    8                 PROP(ELM,3)=THETA
                      PROP(ELM,4)=THETA+DANG
                      PROP(ELM,5)=4.0
                      IF(J.EQ.NUMRAD-1)PROP(ELM,5)=3.0
```

```
C
C                         Define element node incidences.
C
                          GEOM(ELM,1)=(J+1)*(NUMANG+1)+I+1
                          IF(J.EQ.NUMRAD-1)GEOM(ELM,1)=(NUMANG+1)*NUMRAD+1
                          GEOM(ELM,2)=J*(NUMANG+1)+I+1
                          GEOM(ELM,3)=GEOM(ELM,2)+1
                          GEOM(ELM,4)=GEOM(ELM,1)+1
                          IF(J.EQ.NUMRAD-1)GEOM(ELM,4)=0
   10           CONTINUE
C
C Check to see if mesh defines a complete circle. If so impose
C constraint that coordinates and incidences along angle equal 0 and
C 2Pi are the same.
C
                IF(THETA.NE.(6.28319-DANG)) GOTO 20
                PROP(ELM,3)=PROP(1,2)
                PROP(ELM,4)=PROP(1,1)
                GEOM(ELM,3)=GEOM(1,2)
                GEOM(ELM.4)=GEOM(1,1)
   20   CONTINUE
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C If debug required output coordinate and incidence information.
C
        if(.not.dmesh)return
        write(26,110)
        do 30 i=1,numelm
        write(26,100)(geom(i,j),j=1,4),(prop(i,j),j=1,5)
   30   continue
  100   format(2x,4(i6),2x,5(f10.3))
  110   format(2x,'node1 ,node2,node3 ,node4 ',2x,'  ra',8x,' rb',8x,
     1  ' qa',8x,' qb')
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C Return to main program.
C
        RETURN
        END
C*************************************************************************

C*************************************************************************
C SUBROUTINE ASSM
C ---------------
C
C The purpose of the ASSM subprogram is to assemble the structure
C stiffness matrix. It does this by looking at each element, obtaining
C its element stiffness matrix and then placing it term by term into the
C structure stiffness matrix. ASSM calls on the two subroutines RECT
C and TRI to calculate element stiffness matrices. Once the
C structure stiffness matrix is assembled ASSM returns it and the
C back to the main program for use in other subroutines.
C Another vector passed form the main program to ASSM is the OUTPUT
C vector. There is no information passed here other than the size of the
C vector. The size of the output vector is dependent on the structure
C size and therefore dimensioned in the main program for convenience.
C ASSM will output the structure stiffness matrix if debug diagnostics
C are requested.
C
        SUBROUTINE ASSM(PROP,GEOM,K,OUTPUT)
C
```

```
C Declare in COMMON all variables required by more than one
C subroutine.  These variables are divided into four catagories:
C       1. SIZE This common block contains variables which define
C               array dimensions.
C       2. PROB This common block contains variables which define
C               problem parameters.
C       3. MAP  This common block contains variables which define
C               the finite element mesh.
C       4. dbug This common block contains variables which define
C               which debug switches are desired.
C Arrays are not passed through COMMON but passed as arguments to the
C subroutines.
C
        COMMON  /SIZE/ NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,
     1          NUMEQN,LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
        INTEGER NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,NUMEQN,
     1          LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
        COMMON  /PROB/ W2,GA,GB,RO,EXTG,ALPHA,THETAO
        REAL    W2,GA,GB,RO,EXTG,ALPHA,THETAO
        COMMON  /MAP/ SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2,RATIO,GRADNT
        INTEGER RATIO
        REAL    SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2
        LOGICAL GRADNT
        common  /dbug/ dmain,dmesh,dassm,dktest,dforce,
     1          dhspac,drect,dtri,dgamma,dfilam,damat,dbmat,
     1          dcmat,dnorm,ddiag,ddiag2,dfunc,dderiv,dctest,datest
        logical  dmain,dmesh,dassm,dktest,dforce,dhspac,
     1          drect,dtri,dgamma,dfilam,damat,dbmat,dcmat,dnorm,
     1          ddiag,ddiag2,dfunc,dderiv,dctest,datest
C
C Declare and dimension arrays required in this subroutine.
C
        INTEGER GEOM(NUMELM,4)
        REAL    PROP(NUMELM,5),K(NUMNOD,WIDTH),OUTPUT(NUMNOD),KELM(10)
C
C Declare variables required only in this subroutine.
C
        INTEGER I,N,M,ELM,JTREF1,JTREF2,JTREF3,JTREF4,PULL,ROW,
     1          COL,PUSH
C
C Initialize structure stiffness matrix to zero.
C
        DO 15 I=1,NUMNOD
                DO 16 J=1,WIDTH
                        K(I,J)=0.0
   16           CONTINUE
   15   CONTINUE
C
C Iterate through each element calculating stiffness matrix and then
C loading into structure stifffness matrix.
C
        DO 50 ELM=1,NUMELM
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C       If debug required output element number.
C
                if(dassm)write(26,120)elm
  120           format(2x,'element=',i4)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C       Determine element type and calculate proper element
C       stiffness matrix.
```

```
C
                I=ELM
                IF(PROP(ELM,5).EQ.4.0) CALL RECT(PROP,KELM,I)
                IF(PROP(ELM,5).EQ.3.0) CALL TRI(PROP,KELM,I)
C
C       Determine structure-element node reference points.
C
                JTREF1 = GEOM(ELM,1)-1
                JTREF2 = GEOM(ELM,2)-2
                JTREF3 = GEOM(ELM,3)-3
                JTREF4 = GEOM(ELM,4)-4
C
C       Iterate through each term of element stiffness matrix and
C       position it in structure stiffness matrix. Only one term of
C       each symmetric pair is considered.
C
                DO 40 N=1,int(PROP(ELM,5))
                        DO 30 M=1,int(PROP(ELM,5))
                                IF(M.GT.N) GOTO 40
                                PULL = M +N*(N-1)/2
                                IF(N.LE.1)ROW = JTREF1 + N
                                IF(N.GT.1)ROW = JTREF2 + N
                                IF(N.GT.2)ROW = JTREF3 + N
                                IF(N.GT.3)ROW = JTREF4 + N
                                IF(M.LE.1)COL = JTREF1 + M
                                IF(M.GT.1)COL = JTREF2 + M
                                IF(M.GT.2)COL = JTREF3 + M
                                IF(M.GT.3)COL = JTREF4 + M
        I=ROW-(1+CODIAG)
        PUSH=COL-I
        K(ROW,PUSH)=K(ROW,PUSH)+KELM(PULL)
        IF(ROW.EQ.COL)GOTO 30
        I=COL-(1+CODIAG)
        PUSH=ROW-I
        K(COL,PUSH)=K(COL,PUSH)+KELM(PULL)
   30                   CONTINUE
   40           CONTINUE
   50   CONTINUE
C
C  Multiply each stiffness term by 2 to represent half circle problem.
C
        DO 51 I=1,NUMNOD
                DO 52 J=1,WIDTH
                        K(I,J)=2.0*K(I,J)
   52           CONTINUE
   51   CONTINUE
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C If debug required output structure stiffness matrix.
C
        if(.not.dassm)return
        write(26,130)
  130   format(/,2x,'**** structure stiffness matrix ****',/,
     1  2X,'          band storage mode            ')
        write(26,100) (i,i=0,width)
        do 53 i=1,numnod
                write(26,200) i,(k(i,j),j=1,width)
   53   continue
  100   format(/,i7,100(i8))
  200   format(2x,i5,2x,100(f6.2,2x))
```

```
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C Return to main program.
C
      RETURN
      END
C*****************************************************************************
C
C*****************************************************************************
C SUBROUTINE RECT
C ---------------
C
C The purpose of this subprogram is to calculate the element stiffness
C matrix for a rectangular sector of a circle. The shape functions used
C for the representation are linear in both radius and angle. The
C details of the equations used here can be seen in the accompanying
C thesis. The inputs required are the element coordinate information,
C shear modulus and density of the structure, and the frequency of the
C incoming wave. This data is supplied through the ASSM subprogram and
C the element stiffness matrix KELM is returned.
C
      SUBROUTINE RECT(PROP,KELM,ELM)
C
C Declare in COMMON all variables required by more than one
C subroutine.  These variables are divided into four catagories:
C       1. SIZE This common block contains variables which define
C               array dimensions.
C       2. PROB This common block contains variables which define
C               problem parameters.
C       3. MAP  This common block contains variables which define
C               the finite element mesh.
C       4. dbug This common block contains variables which define
C               which debug switches are desired.
C Arrays are not passed through COMMON but passed as arguments to the
C subroutines.
C
      COMMON  /SIZE/ NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,
     1        NUMEQN,LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
      INTEGER NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,NUMEQN,
     1        LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
      COMMON  /PROB/ W2,GA,GB,RO,EXTG,ALPHA,THETAO
      REAL    W2,GA,GB,RO,EXTG,ALPHA,THETAO
      COMMON  /MAP/ SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2,RATIO,GRADNT
      INTEGER RATIO
      REAL    SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2
      LOGICAL GRADNT
      common  /dbug/ dmain,dmesh,dassm,dktest,dforce,
     1        dhspac,drect,dtri,dgamma,dfilam,damat,dbmat,
     1        dcmat,dnorm,ddiag,ddiag2,dfunc,dderiv,dctest,datest
      logical  dmain,dmesh,dassm,dktest,dforce,dhspac,
     1        drect,dtri,dgamma,dfilam,damat,dbmat,dcmat,dnorm,
     1        ddiag,ddiag2,dfunc,dderiv,dctest,datest
C
C  Declare and dimension arrays required in this subroutine.
C  KELM, KSTIF, and KMASS are element stiffness dependent and are therfore
C  dimensioned with numbers here as they would have to be changed if the
C  type of element were to be changed. They are independent of the
C  structure size. KELM must also be dimensioned in ASSM.
C
      REAL    PROP(NUMELM,5),KELM(10),KSTIFF(10),KMASS(10),KSTIF2(10)
```

```
C
C Declare variables required only by this subroutine.
C
      INTEGER ELM
      REAL    RA,RB,QA,QB,DIFFQ,DIFFR,RDIFSQ,DENOM
C
C Assign simplified variable names.
C
      RA = PROP(ELM,1)
      RB = PROP(ELM,2)
      QA = PROP(ELM,3)
      QB = PROP(ELM,4)
      DIFFQ = QB-QA
      DIFFR = RB-RA
      RDIFSQ = RB**2 - RA**2
      DENOM = 6.0*DIFFQ*DIFFR**2
C$    IF(GA.EQ.0.0)GOTO 17
C
C Calculate element stiffness properties. Equations are derived in the
C text. Stiffness is composed of a variable stiffness term, a constant
C stiffness term, and a mass term. These
C are calculated seperately and then combined later in this subroutine.
C If debug is required each term is output.
C
C-----------------------------------------------------------------------
      KSTIF2(1) =((RB**3-RA**3)/(3.0*DIFFQ*DIFFR)*(DIFFQ**2*COS(QA)
     1            +2.0*DIFFQ*SIN(QA)-2.0*(COS(QA)-COS(QB))))
     1            -((COS(QB)-COS(QA))/(3.0*DIFFQ)*DIFFR**2)
C     ++++++++++++++++++++++++++++++++
      if(drect)write(26,141)
  141 format(2x,'kstif2(1-10)')
      if(drect)write(26,*)kstif2(1)
C     ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KSTIF2(2) =((RB**3-RA**3)/(3.0*DIFFQ*DIFFR)*(-(DIFFQ**2)*COS(QA)
     1            -2.0*DIFFQ*SIN(QA)+2.0*(COS(QA)-COS(QB))))
     1            -((COS(QB)-COS(QA))/(6.0*DIFFQ)*DIFFR**2)
C     ++++++++++++++++++++++++++++++++
      if(drect)write(26,*)kstif2(2)
C     ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KSTIF2(3) =((RB**3-RA**3)/(3.0*DIFFQ*DIFFR)*(DIFFQ**2*COS(QA)
     1            +2.0*DIFFQ*SIN(QA)-2.0*(COS(QA)-COS(QB))))
     1            -((COS(QB)-COS(QA))/(3.0*DIFFQ)*DIFFR**2)
C     ++++++++++++++++++++++++++++++++
      if(drect)write(26,*)kstif2(3)
C     ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KSTIF2(4) =((RB**3-RA**3)/(3.0*DIFFQ*DIFFR)*(DIFFQ*(SIN(QA)
     1            +SIN(QB))-2.0*(COS(QA)-COS(QB))))
     1            +((COS(QB)-COS(QA))/(6.0*DIFFQ)*DIFFR**2)
C     ++++++++++++++++++++++++++++++++
      if(drect)write(26,*)kstif2(4)
C     ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KSTIF2(5) =((RB**3-RA**3)/(3.0*DIFFQ*DIFFR)*(-DIFFQ*(SIN(QA)
     1            +SIN(QB))+2.0*(COS(QA)-COS(QB))))
     1            +((COS(QB)-COS(QA))/(3.0*DIFFQ)*DIFFR**2)
C     ++++++++++++++++++++++++++++++++
      if(drect)write(26,*)kstif2(5)
```

```
C       ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
        KSTIF2(6) =((RB**3-RA**3)/(3.0*DIFFQ*DIFFR)*(-(DIFFQ**2)*COS(QB)
     1              +2.0*DIFFQ*SIN(QB)-2.0*(COS(QA)-COS(QB))))
     1              -((COS(QB)-COS(QA))/(3.0*DIFFQ)*DIFFR**2)
C       ++++++++++++++++++++++++++++++++
        if(drect)write(26,*)kstif2(6)
C       ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
        KSTIF2(7) =((RB**3-RA**3)/(3.0*DIFFQ*DIFFR)*(-DIFFQ*(SIN(QA)
     1              +SIN(QB))+2.0*(COS(QA)-COS(QB))))
     1              +((COS(QB)-COS(QA))/(3.0*DIFFQ)*DIFFR**2)
C       ++++++++++++++++++++++++++++++++
        if(drect)write(26,*)kstif2(7)
C       ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
        KSTIF2(8) =((RB**3-RA**3)/(3.0*DIFFQ*DIFFR)*(DIFFQ*(SIN(QA)
     1              +SIN(QB))-2.0*(COS(QA)-COS(QB))))
     1              +((COS(QB)-COS(QA))/(6.0*DIFFQ)*DIFFR**2)
C       ++++++++++++++++++++++++++++++++
        if(drect)write(26,*)kstif2(8)
C       ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
        KSTIF2(9) =((RB**3-RA**3)/(3.0*DIFFQ*DIFFR)*(DIFFQ**2*COS(QB)
     1              -2.0*DIFFQ*SIN(QB)+2.0*(COS(QA)-COS(QB))))
     1              -((COS(QB)-COS(QA))/(6.0*DIFFQ)*DIFFR**2)
C       ++++++++++++++++++++++++++++++++
        if(drect)write(26,*)kstif2(9)
C       ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
        KSTIF2(10) =((RB**3-RA**3)/(3.0*DIFFQ*DIFFR)*(-(DIFFQ**2)*COS(QB)
     1              +2.0*DIFFQ*SIN(QB)-2.0*(COS(QA)-COS(QB))))
     1              -((COS(QB)-COS(QA))/(3.0*DIFFQ)*DIFFR**2)
C       ++++++++++++++++++++++++++++++++
        if(drect)write(26,*)kstif2(10)
C       ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
   17 KSTIFF(1)= (DIFFQ**2*RDIFSQ + (-9.0*RB**2+12.0*RA*RB-3.0*RA**2) +
     1 6.0*RB**2*ALOG(RB/RA))/DENOM
C       ++++++++++++++++++++++++++++++++
        if(drect)write(26,140)
  140   format(2x,'kstiff(1-10)')
        if(drect)write(26,*)kstiff(1)
C       ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KSTIFF(2)= (-(DIFFQ**2)*RDIFSQ + 3.0*RDIFSQ -
     1 6.0*RA*RB*ALOG(RB/RA))/DENOM
C       ++++++++++++++++++++++++++++++++
        if(drect)write(26,*)kstiff(2)
C       ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KSTIFF(3)= (DIFFQ**2*RDIFSQ + (3.0*RB**2-12.0*RA*RB+9.0*RA**2) +
     1 6.0*RA**2*ALOG(RB/RA))/DENOM
C       ++++++++++++++++++++++++++++++++
        if(drect)write(26,*)kstiff(3)
C       ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KSTIFF(4)= (-0.5*DIFFQ**2*RDIFSQ - 3.0*RDIFSQ +
     1 6.0*RA*RB*ALOG(RB/RA))/DENOM
C       ++++++++++++++++++++++++++++++++
```

```
        if(drect)write(26,*)kstiff(4)
C       ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KSTIFF(5)=(0.5*DIFFQ**2*RDIFSQ + (-3.0*RB**2+12.0*RA*RB-9.0*RA**2)
     1  -6.0*RA**2*ALOG(RB/RA))/DENOM
C       ++++++++++++++++++++++++++++++++
        if(drect)write(26,*)kstiff(5)
C       ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KSTIFF(6)=(DIFFQ**2*RDIFSQ + (3.0*RB**2-12.0*RA*RB+9.0*RA**2) +
     1  6.0*RA**2*ALOG(RB/RA))/DENOM
C       ++++++++++++++++++++++++++++++++
        if(drect)write(26,*)kstiff(6)
C       ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KSTIFF(7)=(0.5*DIFFQ**2*RDIFSQ + (9.0*RB**2-12.0*RA*RB+3.0*RA**2)
     1  - 6.0*RB**2*ALOG(RB/RA))/DENOM
C       ++++++++++++++++++++++++++++++++
        if(drect)write(26,*)kstiff(7)
C       ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KSTIFF(8)=(-0.5*DIFFQ**2*RDIFSQ - 3.0*RDIFSQ +
     1  6.0*RA*RB*ALOG(RB/RA))/DENOM
C       ++++++++++++++++++++++++++++++++
        if(drect)write(26,*)kstiff(8)
C       ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KSTIFF(9)=(-(DIFFQ**2)*RDIFSQ + 3.0*RDIFSQ -
     1  6.0*RB*RA*ALOG(RB/RA))/DENOM
C       ++++++++++++++++++++++++++++++++
        if(drect)write(26,*)kstiff(9)
C       ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KSTIFF(10)=(DIFFQ**2*RDIFSQ + (-9.0*RB**2+12.0*RA*RB-3.0*RA**2) +
     1  6.0 *RB**2*ALOG(RB/RA))/DENOM
C       ++++++++++++++++++++++++++++++++
        if(drect)write(26,*)kstiff(10)
C       ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------------
      KMASS(1)=(DIFFQ)*(RB**2+2.0*RA*RB-3.0*RA**2)/36.0
C       ++++++++++++++++++++++++++++++++
        if(drect)write(26,145)
  145   format(2x,'kmass(1-10)')
        if(drect)write(26,*)kmass(1)
C       ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------------
      KMASS(2)=(DIFFQ)*(RDIFSQ)/36.0
C       ++++++++++++++++++++++++++++++++
        if(drect)write(26,*)kmass(2)
C       ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------------
      KMASS(3)=(DIFFQ)*(3.0*RB**2-2.0*RA*RB-RA**2)/36.0
C       ++++++++++++++++++++++++++++++++
        if(drect)write(26,*)kmass(3)
C       ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------------
      KMASS(4)=(DIFFQ)*(RDIFSQ)/72.0
C       ++++++++++++++++++++++++++++++++
        if(drect)write(26,*)kmass(4)
C       ++++++++++++++++++++++++++++++++
```

```
C-----------------------------------------------------------------------
      KMASS(5)=(DIFFQ)*(3.0*RB**2-2.0*RA*RB-RA**2)/72.0
C       ++++++++++++++++++++++++++++++++
        if(drect)write(26,*)kmass(5)
C       ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KMASS(6)=(DIFFQ)*(3.0*RB**2-2.0*RA*RB-RA**2)/36.0
C       ++++++++++++++++++++++++++++++++
        if(drect)write(26,*)kmass(6)
C       ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KMASS(7)=(DIFFQ)*(RB**2+2.0*RA*RB-3.0*RA**2)/72.0
C       ++++++++++++++++++++++++++++++++
        if(drect)write(26,*)kmass(7)
C       ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KMASS(8)=(DIFFQ)*(RDIFSQ)/72.0
C       ++++++++++++++++++++++++++++++++
        if(drect)write(26,*)kmass(8)
C       ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KMASS(9)=(DIFFQ)*(RDIFSQ)/36.0
C       ++++++++++++++++++++++++++++++++
        if(drect)write(26,*)kmass(9)
C       ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KMASS(10)=(DIFFQ)*(RB**2+2.0*RA*RB-3.0*RA**2)/36.0
C       ++++++++++++++++++++++++++++++++
        if(drect)write(26,*)kmass(10)
C       ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
C Combine stiffness and mass terms.
C
        KELM(1)=RO*W2*KMASS(1)-GB*KSTIFF(1)-GA*KSTIF2(1)
        KELM(2)=RO*W2*KMASS(2)-GB*KSTIFF(2)-GA*KSTIF2(2)
        KELM(3)=RO*W2*KMASS(3)-GB*KSTIFF(3)-GA*KSTIF2(3)
        KELM(4)=RO*W2*KMASS(4)-GB*KSTIFF(4)-GA*KSTIF2(4)
        KELM(5)=RO*W2*KMASS(5)-GB*KSTIFF(5)-GA*KSTIF2(5)
        KELM(6)=RO*W2*KMASS(6)-GB*KSTIFF(6)-GA*KSTIF2(6)
        KELM(7)=RO*W2*KMASS(7)-GB*KSTIFF(7)-GA*KSTIF2(7)
        KELM(8)=RO*W2*KMASS(8)-GB*KSTIFF(8)-GA*KSTIF2(8)
        KELM(9)=RO*W2*KMASS(9)-GB*KSTIFF(9)-GA*KSTIF2(9)
        KELM(10)=RO*W2*KMASS(10)-GB*KSTIFF(10)-GA*KSTIF2(10)
C
C Return to ASSM subroutine.
C
        RETURN
        END
C***********************************************************************
C
C***********************************************************************
C SUBROUTINE TRI
C ---------------
C
C The purpose of this subprogram is to calculate the element stiffness
C matrix for a triangular sector of a circle. The shape functions used
C for the representation are linear in both radius and angle. The
C details of the equations used here can be seen in the accompanying
C thesis. The inputs required are the element coordinate information,
C shear modulus and density of the structure, and the frequency of the
```

```
C incoming wave. This data is supplied through the ASSM subprogram and
C the element stiffness matrix KELM is returned.
C
      SUBROUTINE TRI(PROP,KELM,ELM)
C
C Declare in COMMON all variables required by more than one
C subroutine.  These variables are divided into four catagories:
C       1. SIZE This common block contains variables which define
C               array dimensions.
C       2. PROB This common block contains variables which define
C               problem parameters.
C       3. MAP  This common block contains variables which define
C               the finite element mesh.
C       4. dbug This common block contains variables which define
C               which debug switches are desired.
C Arrays are not passed through COMMON but passed as arguments to the
C subroutines.
C
      COMMON  /SIZE/ NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,
     1        NUMEQN,LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
      INTEGER NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,NUMEQN,
     1        LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
      COMMON  /PROB/ W2,GA,GB,RO,EXTG,ALPHA,THETAO
      REAL    W2,GA,GB,RO,EXTG,ALPHA,THETAO
      COMMON  /MAP/ SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2,RATIO,GRADNT
      INTEGER RATIO
      REAL    SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2
      LOGICAL GRADNT
      common  /dbug/ dmain,dmesh,dassm,dktest,dforce,
     1        dhspac,drect,dtri,dgamma,dfilam,damat,dbmat,
     1        dcmat,dnorm,ddiag,ddiag2,dfunc,dderiv,dctest,datest
      logical  dmain,dmesh,dassm,dktest,dforce,dhspac,
     1        drect,dtri,dgamma,dfilam,damat,dbmat,dcmat,dnorm,
     1        ddiag,ddiag2,dfunc,dderiv,dctest,datest
C
C  Declare and dimension arrays required in this subroutine.
C  KELM, KSTIF, and KMASS are element stiffness dependent and are therfore
C  dimensioned with numbers here as they would have to be changed if the
C  type of element were to be changed. They are independent of the
C  structure size. KELM must also be dimensioned in ASSM.
C
      REAL    PROP(NUMELM,5),KELM(10),KSTIFF(10),KMASS(10),KSTIF2(10)
C
C Declare variables required only by this subroutine.
C
      INTEGER ELM
      REAL    RA,RB,QA,QB,DIFFQ
C
C Assign simplified variable names.
C
      RA = PROP(ELM,1)
      RB = PROP(ELM,2)
      QA = PROP(ELM,3)
      QB = PROP(ELM,4)
      DIFFQ = QB-QA
C$    IF(GA.EQ.0.0) GOTO 17
C
C Calculate element stiffness properties. Equations are derived in the
C text. Stiffness is composed of a variable stiffness term, a constant
C striffness term, and a mass term. These
```

```
C are calculated seperately and then combined later in this subroutine.
C If debug is required each term is output.
C
C-----------------------------------------------------------------------
      KSTIF2(1)= RB**2/(3.0*DIFFQ)*(-(DIFFQ**2)*(COS(QB)-COS(QA)))
C     ++++++++++++++++++++++++++++++++
      if(dtri)write(26,151)
  151 format(2x,'kstif2(1-6)')
      if(dtri)write(26,*)kstif2(1)
C     ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KSTIF2(2)= RB**2/(3.0*DIFFQ)*(-(DIFFQ**2)*COS(QA)+DIFFQ*
     1           (SIN(QB)-SIN(QA)))
C     ++++++++++++++++++++++++++++++++
      if(dtri)write(26,*)kstif2(2)
C     ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KSTIF2(3)= RB**2/(3.0*DIFFQ)*((DIFFQ**2)*COS(QA)+
     1           2.0*DIFFQ*SIN(QA)+1.0*(COS(QB)-COS(QA)))
C     ++++++++++++++++++++++++++++++++
      if(dtri)write(26,*)kstif2(3)
C     ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KSTIF2(4)= RB**2/(3.0*DIFFQ)*((DIFFQ**2)*COS(QB)-DIFFQ*
     1           (SIN(QB)-SIN(QA)))
C     ++++++++++++++++++++++++++++++++
      if(dtri)write(26,*)kstif2(4)
C     ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KSTIF2(5)= RB**2/(3.0*DIFFQ)*(-(DIFFQ**2)*(SIN(QA)+SIN(QB))
     1           -(COS(QB)-COS(QA)))
C     ++++++++++++++++++++++++++++++++
      if(dtri)write(26,*)kstif2(5)
C     ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KSTIF2(6)= RB**2/(3.0*DIFFQ)*(DIFFQ**2*COS(QB)
     1           +2.0*DIFFQ*SIN(QB)+(COS(QB)-COS(QA)))
C     ++++++++++++++++++++++++++++++++
      if(dtri)write(26,*)kstif2(6)
C     ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
   17 KSTIFF(1)= 0.5*DIFFQ
C     ++++++++++++++++++++++++++++++++
      if(dtri)write(26,150)
  150 format(2x,'kstiff(1-6)')
      if(dtri)write(26,*)kstiff(1)
C     ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KSTIFF(2)= -0.25*DIFFQ
C     ++++++++++++++++++++++++++++++++
      if(dtri)write(26,*)kstiff(2)
C     ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KSTIFF(3)= (DIFFQ**2 + 3.0)/(6.0 * DIFFQ)
C     ++++++++++++++++++++++++++++++++
      if(dtri)write(26,*)kstiff(3)
C     ++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KSTIFF(4)= -0.25*DIFFQ
C     ++++++++++++++++++++++++++++++++
```

```
        if(dtri)write(26,*)kstiff(4)
C       ++++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
       KSTIFF(5)= (DIFFQ**2 - 6.0)/(12.0 * DIFFQ)
C       ++++++++++++++++++++++++++++++++++
        if(dtri)write(26,*)kstiff(5)
C       ++++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
       KSTIFF(6)= (DIFFQ**2 + 3.0)/(6.0 * DIFFQ)
C       ++++++++++++++++++++++++++++++++++
        if(dtri)write(26,*)kstiff(6)
C       ++++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KMASS(1)=(DIFFQ)*RB**2/12.0
C       ++++++++++++++++++++++++++++++++++
        if(dtri)write(26,160)
  160   format(2x,'kmass(1-6)')
        if(dtri)write(26,*)kmass(1)
C       ++++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KMASS(2)=(DIFFQ)*RB**2/24.0
C       ++++++++++++++++++++++++++++++++++
        if(dtri)write(26,*)kmass(2)
C       ++++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KMASS(3)=(DIFFQ)*RB**2/12.0
C       ++++++++++++++++++++++++++++++++++
        if(dtri)write(26,*)kmass(3)
C       ++++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KMASS(4)=(DIFFQ)*RB**2/24.0
C       ++++++++++++++++++++++++++++++++++
        if(dtri)write(26,*)kmass(4)
C       ++++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KMASS(5)=(DIFFQ)*RB**2/24.0
C       ++++++++++++++++++++++++++++++++++
        if(dtri)write(26,*)kmass(5)
C       ++++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
      KMASS(6)=(DIFFQ)*RB**2/12.0
C       ++++++++++++++++++++++++++++++++++
        if(dtri)write(26,*)kmass(6)
C       ++++++++++++++++++++++++++++++++++
C-----------------------------------------------------------------------
C Combine stiffness and mass terms.
C
        KELM(1)=RO*W2*KMASS(1)-GB*KSTIFF(1)-GA*KSTIF2(1)
        KELM(2)=RO*W2*KMASS(2)-GB*KSTIFF(2)-GA*KSTIF2(2)
        KELM(3)=RO*W2*KMASS(3)-GB*KSTIFF(3)-GA*KSTIF2(3)
        KELM(4)=RO*W2*KMASS(4)-GB*KSTIFF(4)-GA*KSTIF2(4)
        KELM(5)=RO*W2*KMASS(5)-GB*KSTIFF(5)-GA*KSTIF2(5)
        KELM(6)=RO*W2*KMASS(6)-GB*KSTIFF(6)-GA*KSTIF2(6)
C
C Return to ASSM subroutine.
C
        RETURN
        END
C***********************************************************************
C
```

```
C***********************************************************************
      SUBROUTINE KTEST(K,WO,WG,KOG,B,KOO,XL,OUTPUT,EXACT)
C
C Declare in COMMON all variables required by more than one
C subroutine.  These variables are divided into four catagories:
C       1. SIZE This common block contains variables which define
C               array dimensions.
C       2. PROB This common block contains variables which define
C               problem parameters.
C       3. MAP  This common block contains variables which define
C               the finite element mesh.
C       4. dbug This common block contains variables which define
C               which debug switches are desired.
C Arrays are not passed through COMMON but passed as arguments to the
C subroutines.
C
      COMMON  /SIZE/ NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,
     1        NUMEQN,LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
      INTEGER NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,NUMEQN,
     1        LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
      COMMON  /PROB/ W2,GA,GB,RO,EXTG,ALPHA,THETAO
      REAL    W2,GA,GB,RO,EXTG,ALPHA,THETAO
      COMMON  /MAP/ SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2,RATIO,GRADNT
      INTEGER RATIO
      REAL    SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2
      LOGICAL GRADNT
      common  /dbug/ dmain,dmesh,dassm,dktest,dforce,
     1        dhspac,drect,dtri,dgamma,dfilam,damat,dbmat,
     1        dcmat,dnorm,ddiag,ddiag2,dfunc,dderiv,dctest,datest
      logical  dmain,dmesh,dassm,dktest,dforce,dhspac,
     1        drect,dtri,dgamma,dfilam,damat,dbmat,dcmat,dnorm,
     1        ddiag,ddiag2,dfunc,dderiv,dctest,datest
C
C Declare and dimension arrays required by this subroutine.
C
      REAL    K(NUMNOD,WIDTH),WO(NUMEQN),WG(INTNOD),KOG(NUMEQN,INTNOD)
     1        ,B(NUMEQN,INTNOD),KOO(NUMEQN,WIDTH),XL(NUMEQN,SPACE),
     2        OUTPUT(NUMNOD),EXACT(NUMEQN)
C
C Declare variables required by this subroutine.
C
      INTEGER N,M,I,J,L,PULL,P,ROW,COL,IER
C
C Read in the values of displacements at the boundry nodes. These
C values are computed from the free field motion at the boundary.
C
      READ(25,*) (WG(I),I=1,INTNOD)
C
C Echo imposed boundary displacements.
C
      WRITE(26,725)
      DO 5 J=1,INTNOD
              WRITE(26,750) J,WG(J)
    5 CONTINUE
C
C Partition K(omega,gamma) from structure stiffness matrix. KOG is
C pulled term by term from K. KOG is stored in full storage mode.
C
      N=0
      DO 20 I=INTNOD+1,NUMNOD
```

```
                N=N+1
                DO 10 J=1,INTNOD
                        KOG(N,J)=0.0
                        B(N,J)=0.0
                        M=I-(1+CODIAG)
                        PULL=J-M
                        IF(PULL.LE.0) GOTO 10
                        IF(PULL.GT.WIDTH) GOTO 20
                        KOG(N,J)=K(I,PULL)
                        B(N,J)=K(I,PULL)*-1.0
   10           CONTINUE
   20   CONTINUE
C
C Partition K(omega,omega) from structure stiffness matrix. KOO is
C pulled term by term from K. KOO is stored in band storage mode rather
C than symmetric storage mode to allow the use of an IMSL solving
C routine.
C
        DO 40 I=INTNOD+1,NUMNOD
                N=I-(INTNOD+1+CODIAG)
                P=N
                IF(N.LT.0)N=0
                M=I+CODIAG
                IF(M.GT.NUMNOD)M=NUMNOD
                DO 30 J=INTNOD+1+N,M
                        L=I-(1+CODIAG)
                        PULL=J-L
                        ROW=I-INTNOD
                        COL=J-INTNOD-P
                        KOO(ROW,COL)=K(I,PULL)
   30           CONTINUE
   40   CONTINUE
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C If debug required output K(omega,gamma) and K(omega,omega).
C
        if(.not.dktest) goto 500
        write(26,300)
  300   format(//,2x,'**** k(omega,gamma) ****',/)
        do 50 i=1,numeqn
        do 60 j=1,INTNOD
        output(j)=kog(i,j)
   60   continue
        write(26,100) (output(n),n=1,INTNOD)
   50   continue
        write(26,310)
  310   format(//,2x,'**** k(omega,omega) ****',/)
        do 70 i=1,numeqn
        do 80 j=1,width
        output(j)=koo(i,j)
   80   continue
        write(26,200)(output(n),n=1,width)
   70   continue
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C Solve equation KOO x X = B = -KOG. X, the solution is written over
C top of B. This solving is done with IMSL routine LEQT1B.
C
  500   CALL LEQT1B(KOO,NUMEQN,CODIAG,CODIAG,NUMEQN,B,INTNOD,NUMEQN,0,
     1  XL,IER)
        IF(IER.EQ.129)write(5,400)
```

```
  400   format(2x,'error')
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C If debug required output INSL altered KOO and solution matrix.
C
        if(.not.dktest)goto 600
        write(26,320)
  320   format(//,2x,'**** modified k(omega,omega) ****',/)
        do 75 i=1,numeqn
        do 85 j=1,width
        output(j)=koo(i,j)
   85   continue
        write(26,200)(output(n),n=1,width)
   75   continue
        write(26,330)
  330   format(//,2x,'**** solution matrix ****')
        do 90 i=1,numeqn
        do 95 j=1,INTNOD
        output(j)=B(i,j)
   95   continue
        write(26,100) (output(n),n=1,INTNOD)
   90   continue
  100   format(2x,111f6.2)
  200   format(2x,111f6.2)
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C Calculate interior node displacements which equals the boundry node
C displacements times the solution matrix.
C
  600   DO 110 I=1,NUMEQN
                WO(I)=0.0
                DO 120 J=1,INTNOD
                        WO(I)=B(I,J)*WG(J)+WO(I)
  120           CONTINUE
  110   CONTINUE
C
C Iterate over nodal points solving for the real portion of the
C displacements from the incident wave equation. This provides a
C partial check on the correctness of the stiffness matrix. This check
C is only valid for a uniform mass and shear modulus equal to those of
C the exterior region. Thus the solved for solution is the free field
C motion.
C
        DO 25 I=0,NUMANG
                THETA = I * DANG
                DO 15 J=1,NUMRAD
                        RAD =((1-DANG/2.0)/(1+DANG/2.0))**J*RANGE
                        IF(.NOT.GRADNT)RAD=RANGE-J*DRAD
                        IF(J.EQ.NUMRAD)RAD=0.0
                        N=(J-1)*INTNOD+I+1
                EXACT(N)=0.5*(COS(SQRT(W2)*RAD*SIN(THETAD+THETA0))+
     1                  COS(SQRT(W2)*RAD*SIN(THETA-THETA0)))
   15           CONTINUE
   25   CONTINUE
C
C Output finite element solution and exact solution side by side for
C comparison.
C
        IF(.NOT.DKTEST)RETURN
        WRITE(26,700)
        DO 35 I=1,NUMEQN
```

```
      J=I+INTNOD
      WRITE(26,750) J,WO(I),EXACT(I)
   35 CONTINUE
  700 FORMAT(//,15X,'SOLVED',15X,'EXACT')
  725 FORMAT(//,10X,'IMPOSED BOUNDARY DISPLACEMENTS PER',/,
     1 10X,'        FREE FIELD MOTION          ')
  750 FORMAT(2X,i5,3x,G15.8,6X,G15.8)
C
C Return to main program.
C
      RETURN
      END
C ***********************************************************************
C
C ***********************************************************************
      SUBROUTINE FORCE(FG,FPHI)
C
C Declare in COMMON all variables required by more than one
C subroutine.  These variables are divided into four catagories:
C       1. SIZE This common block contains variables which define
C               array dimensions.
C       2. PROB This common block contains variables which define
C               problem parameters.
C       3. MAP  This common block contains variables which define
C               the finite element mesh.
C       4. dbug This common block contains variables which define
C               which debug switches are desired.
C Arrays are not passed through COMMON but passed as arguments to the
C subroutines.
C
      COMMON  /SIZE/ NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,
     1        NUMEQN,LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
      INTEGER NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,NUMEQN,
     1        LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
      COMMON  /PROB/ W2,GA,GB,RO,EXTG,ALPHA,THETAO
      REAL    W2,GA,GB,RO,EXTG,ALPHA,THETAO
      COMMON  /MAP/ SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2,RATIO,GRADNT
      INTEGER RATIO
      REAL    SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2
      LOGICAL GRADNT
      common  /dbug/ dmain,dmesh,dassm,dktest,dforce,
     1        dhspac,drect,dtri,dgamma,dfilam,damat,dbmat,
     1        dcmat,dnorm,ddiag,ddiag2,dfunc,dderiv,dctest,datest
      logical  dmain,dmesh,dassm,dktest,dforce,dhspac,
     1        drect,dtri,dgamma,dfilam,damat,dbmat,dcmat,dnorm,
     1        ddiag,ddiag2,dfunc,dderiv,dctest,datest
C
C Declare and dimension arrays used in this subroutine.
C
      COMPLEX FG(INTNOD),FPHI(BNDELM)
      REAL    XI(50),WGT(50)
C
C Declare variable required only by this subroutine.
C
      INTEGER NUMPTS,I,ELM,J
      REAL    KAPPA,KAPPAR,QA,QB,THETA
      COMPLEX FGCNST,FPCNST,FG1,FG2,FPHI1,NFG1,NFG2,NFPHI1
C
C Define required constants.
C
```

```
      KAPPA = SQRT(W2*RANGE/EXTG)
      KAPPAR = KAPPA * RANGE
      FGCNST = CMPLX(0.0,-KAPPA/4.0)
      FPCNST = CMPLX(0.25,0.0)
C ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C If debug required output computed constants.
C
      if(dforce)write(26,1900) fgcnst,fpcnst
C ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C Read in Gauss integration points for numerical integration of force
C expressions.
C
      READ(24,*) NUMPTS
      DO 30 I=1,NUMPTS/2
              READ(24,*) XI(I),WGT(I)
   30 CONTINUE
C-----------------------------------------------------------------------
C
C Compute forces at boundary nodes.
C
C
C    Initialize forces.
C
      DO 10 I=1,INTNOD
              FG(I)=(0.0,0.0)
   10 CONTINUE
C     ++++++++++++++++++++++++
      if(dforce)write(26,1300)
C     ++++++++++++++++++++++++
C
C    Integrate over each element for both terms of shape function.
C    Contributions to common nodes are summed.
C
      DO 20 ELM=1,NUMANG
C             ++++++++++++++++++++++++++++++++++++++
              j=elm
              if(dforce.and.elm.le.1)write(26,300)elm
C             ++++++++++++++++++++++++++++++++++++++
C
C          Define integration limits.
C
              QA = (ELM-1)*DANG
              QB = ELM*DANG
C             +++++++++++++++++++++++++++++++++++++++
              if(dforce.and.elm.le.1)write(26,400)qa,qb
C             +++++++++++++++++++++++++++++++++++++++
C
C          Iterate through each Gauss integration point.
C
              DO 50 I=1,NUMPTS/2
C
C              Transform -1 to 1 positive integration points to qa
C              to qb points.
C
                      THETA = ((QB-QA)*XI(I)+(QB+QA))/2.0
C                     ++++++++++++++++++++++++++++++++++++++
                      if(dforce.and.elm.le.1)write(26,500)theta
C                     ++++++++++++++++++++++++++++++++++++++
C
```

```
C                 Evaluate force function at positive integration
C                 point.
C
                        CALL GAMMA(QA,QB,THETA,KAPPAR,FG1,FG2,j)
C                       ++++++++++++++++++++++++++++++++++++++++
                        if(dforce.and.elm.le.1)write(26,700)fg1,fg2
C                       ++++++++++++++++++++++++++++++++++++++++
C
C               Transform -1 to 1 negative integration points to qa
C               to qb points.
C
                        THETA = ((QA-QB)*XI(I)+(QB+QA))/2.0
C                       +++++++++++++++++++++++++++++++++++++++
                        if(dforce.and.elm.le.1)write(26,500)theta
C                       +++++++++++++++++++++++++++++++++++++++
C
C                 Evaluate force function at negative integration
C                 point.
C
                        CALL GAMMA(QA,QB,THETA,KAPPAR,NFG1,NFG2,j)
C                       ++++++++++++++++++++++++++++++++++++++++++
                        if(dforce.and.elm.le.1)write(26,800)nfg1,nfg2
C                       ++++++++++++++++++++++++++++++++++++++++++
C
C                 Multiply evaluation by integration weight and
C                 function constant and sum.
C
                        FG(ELM)=FG(ELM)+FGCNST*WGT(I)/2.0*(FG1+NFG1)
                        FG(ELM+1)=FG(ELM+1)+FGCNST*WGT(I)/2.0*(FG2+NFG2)
C                       ++++++++++++++++++++++++++++++++++++++++++++++
                        if(dforce.and.elm.le.1)j=elm+1
                        if(dforce.and.elm.le.1)write(26,600)elm,fg(elm),
     1                  j,fg(j)
C                       ++++++++++++++++++++++++++++++++++++++++++++++
   50             CONTINUE
   20   CONTINUE
C-----------------------------------------------------------------------
C
C Compute forces over elements originating from the halfspace.
C
C
C    Initialize forces.
C
        DO 80 I=1,BNDELM
               FPHI(I) = CMPLX(0.0,0.0)
   80   CONTINUE
C       ++++++++++++++++++++++++
        if(dforce)write(26,1400)
C       ++++++++++++++++++++++++
C
C    Integrate over each element.
C
        DO 60 ELM=1,BNDELM
C              ++++++++++++++++++++++++++++++++++++++
               j=elm
               if(dforce.and.elm.le.1)write(26,300)elm
C              ++++++++++++++++++++++++++++++++++++++
C
C          Define integration limits.
C
```

```
                QA = (ELM-1)*BDANG1
                QB = ELM*BDANG1
                IF(ELM.EQ.BNDELM)QA = (ELM-1)*BDANG2
                IF(ELM.EQ.BNDELM)QB = ELM*BDANG2
C               ++++++++++++++++++++++++++++++++++++++++
                if(dforce.and.elm.le.1)write(26,400)qa,qb
C               ++++++++++++++++++++++++++++++++++++++++
C
C          Iterate through each Gauss integration point.
C
                DO 70 I=1,NUMPTS/2
C
C                Transform -1 to 1 positive integration points to qa
C                to qb points.
C
                      THETA = ((QB-QA)*XI(I)+(QB+QA))/2.0
C                     ++++++++++++++++++++++++++++++++++++++++
                      if(dforce.and.elm.le.1)write(26,500)theta
C                     ++++++++++++++++++++++++++++++++++++++++
C
C                 Evaluate force function at positive integration
C                 point.
C
                      CALL PHILAM(THETA,KAPPAR,FPHI1,j)
C                     ++++++++++++++++++++++++++++++++++++++++
                      if(dforce.and.elm.le.1)write(26,1000)fphi1
C                     ++++++++++++++++++++++++++++++++++++++++
C
C                Transform -1 to 1 negative integration points to qa
C                to qb points.
C
                      THETA = ((QA-QB)*XI(I)+(QB+QA))/2.0
C                     ++++++++++++++++++++++++++++++++++++++++
                      if(dforce.and.elm.le.1)write(26,500)theta
C                     ++++++++++++++++++++++++++++++++++++++++
C
C                 Evaluate force function at negative integration
C                 point.
C
                      CALL PHILAM(THETA,KAPPAR,NFPHI1,j)
C                     +++++++++++++++++++++++++++++++++++++++++
                      if(dforce.and.elm.le.1)write(26,1100)nfphi1
C                     +++++++++++++++++++++++++++++++++++++++++
C
C                 Multiply evaluation by integration weight and
C                 function constant and sum.
C
                      FPHI(ELM)=FPHI(ELM)+FPCNST*(QB-QA)*WGT(I)
     1                /2.0*(FPHI1+NFPHI1)
C                     ++++++++++++++++++++++++++++++++++++++++++++
                      if(dforce.and.elm.le.1)write(26,1200)elm,
     1                fphi(elm)
C                     ++++++++++++++++++++++++++++++++++++++++++++
   70           CONTINUE
   60   CONTINUE
C-----------------------------------------------------------------
C
C Multiply forces by 2 to represent half circle.
C
        DO 65 ROW=2,INTNOD-1
```

```
                FG(ROW)=2.0*FG(ROW)
   65   CONTINUE
        DO 66 ROW=1,BNDELM
                FPHI(ROW)=2.0*FPHI(ROW)
   66   CONTINUE
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C If debug required output FG, FPHI
C
        if(.not.dforce)RETURN
        write(26,110) numpts
        write(26,100)
        do 40 i=1,intnod
                write(26,200)i,fg(i)
   40   continue
        write(26,150)
        do 45 i=1,bndelm
                write(26,200)i,fphi(i)
   45   continue
        return
C
C Debug output formats.
C
  100   format(//,2x,' node',14x,'fg',/,13x,'real',12x,'imaginary',/)
  110   format(/,2x,'number of gauss points = ',i5)
  150   format(//,2x,' node',14x,'fphi',/,13x,'real',12x,
     1  'imaginary',/)
  200   format(2x,i5,2x,2g15.7)
  300   format(2x,'element =',i5)
  400   format(2x,'qa ='g15.7,/,2x,'qb=',g15.7)
  500   format(2x,'theta ='g15.7)
  600   format(2x,'fg(',i1,') =',2g15.7,/,2x,'fg(',i1,') =',2g15.7)
  700   format(2x,'fg1 =',2g15.7,/,2x,'fg2 =',2g15.7)
  800   format(2x,'nfg1 =',2g15.7,/,2x,'nfg2 =',2g15.7)
 1000   format(2x,'fphi1 =',2g15.7)
 1100   format(2x,'nfphi1 =',2g15.7)
 1200   format(2x,'fphi(',i1,') =',2g15.7)
 1300   format(2x,'******** fg check *********')
 1400   format(///,2x,'******** fphi check *********')
 1900   format(2x,'fgcnst =',2g15.7,/,
     1  2x,'fpcnst = ',2g15.7)
        END
C*********************************************************************
C
C*********************************************************************
        SUBROUTINE GAMMA(QA,QB,THETA,KAPPAR,FG1,FG2,elm)
C
C Declare in COMMON all variables required by more than one
C subroutine.  These variables are divided into four catagories:
C       1. SIZE This common block contains variables which define
C               array dimensions.
C       2. PROB This common block contains variables which define
C               problem parameters.
C       3. MAP  This common block contains variables which define
C               the finite element mesh.
C       4. dbug This common block contains variables which define
C               which debug switches are desired.
C Arrays are not passed through COMMON but passed as arguments to the
C subroutines.
C
        COMMON  /SIZE/ NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,
```

```
     1          NUMEQN,LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
        INTEGER NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,NUMEQN,
     1          LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
        COMMON  /PROB/ W2,GA,GB,RO,EXTG,ALPHA,THETAO
        REAL    W2,GA,GB,RO,EXTG,ALPHA,THETAO
        COMMON  /MAP/ SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2,RATIO,GRADNT
        INTEGER RATIO
        REAL    SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2
        LOGICAL GRADNT
        common  /dbug/ dmain,dmesh,dassm,dktest,dforce,
     1          dhspac,drect,dtri,dgamma,dfilam,damat,dbmat,
     1          dcmat,dnorm,ddiag,ddiag2,dfunc,dderiv,dctest,datest
        logical  dmain,dmesh,dassm,dktest,dforce,dhspac,
     1          drect,dtri,dgamma,dfilam,damat,dbmat,dcmat,dnorm,
     1          ddiag,ddiag2,dfunc,dderiv,dctest,datest
C
C Declare variables used only in this subroutine.
C
        INTEGER elm
        REAL    QA,QB,THETA,KAPPAR,C,D
        DOUBLE PRECISION SINDIF,SINSUM,CSSUM,SSDIF,SSSUM,A,B
        COMPLEX FG1,FG2,FGAMAA,NGAMA1,NGAMA2
C
C Evaluate force function for the two terms of the shape function.
C
        SINDIF = SIN(THETA-THETAO)
        SINSUM = SIN(THETA+THETAO)
        CSDIF  = COS(KAPPAR*SINDIF)
        CSSUM  = COS(KAPPAR*SINSUM)
        SSDIF  = SIN(KAPPAR*SINDIF)
        SSSUM  = SIN(KAPPAR*SINSUM)
        A = SINSUM*CSSUM-SINDIF*CSDIF
        B = SINSUM*SSSUM+SINDIF*SSDIF
        C = A
        D = B
        FGAMAA = CMPLX(C,D)
        NGAMA1 = CMPLX((QB-THETA),0.0)
        NGAMA2 = CMPLX((THETA-QA),0.0)
        FG1 = NGAMA1*FGAMAA
        FG2 = NGAMA2*FGAMAA
C       ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C If debug required write results of computations.
C
        if(dgamma.and.elm.le.1)write(26,100)sindif,sinsum,csdif,
     1  cssum,ssdif,sssum,fgamaa,ngama1,ngama2,fg1,fg2
C       ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C Format statements for debug output.
C
  100   format(//,2x,'sindif =',g15.7,/,
        2x,'sinsum =',g15.7,/,
        2x,'csdif =',g15.7,/,
        2x,'cssum =',g15.7,/,
        2x,'ssdif =',g15.7,/,
        2x,'sssum =',g15.7,/,
        2x,'fgamaa =',2g15.7,/,
        2x,'ngama1 =',2g15.7,/,
        2x,'ngama2 =',2g15.7,/,
        2x,'fg1 =',2g15.7,/,
        2x,'fg2 =',2g15.7,//)
```

```
C
C Return to FORCE subroutine.
C
      RETURN
      END
C*********************************************************************
C
C*********************************************************************
      SUBROUTINE PHILAM(THETA,KAPPAR,F1,elm)
C
C Declare in COMMON all variables required by more than one
C subroutine.  These variables are divided into four catagories:
C       1. SIZE This common block contains variables which define
C               array dimensions.
C       2. PROB This common block contains variables which define
C               problem parameters.
C       3. MAP  This common block contains variables which define
C               the finite element mesh.
C       4. dbug This common block contains variables which define
C               which debug switches are desired.
C Arrays are not passed through COMMON but passed as arguments to the
C subroutines.
C
      COMMON  /SIZE/ NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,
     1        NUMEQN,LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
      INTEGER NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,NUMEQN,
     1        LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
      COMMON  /PROB/ W2,GA,GB,RO,EXTG,ALPHA,THETAO
      REAL    W2,GA,GB,RO,EXTG,ALPHA,THETAO
      COMMON  /MAP/ SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2,RATIO,GRADNT
      INTEGER RATIO
      REAL    SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2
      LOGICAL GRADNT
      common  /dbug/ dmain,dmesh,dassm,dktest,dforce,
     1        dhspac,drect,dtri,dgamma,dfilam,damat,dbmat,
     1        dcmat,dnorm,ddiag,ddiag2,dfunc,dderiv,dctest,datest
      logical  dmain,dmesh,dassm,dktest,dforce,dhspac,
     1        drect,dtri,dgamma,dfilam,damat,dbmat,dcmat,dnorm,
     1        ddiag,ddiag2,dfunc,dderiv,dctest,datest
C
C Declare variables used only in this subroutine.
C
      INTEGER elm
      REAL    THETA,KAPPAR,C,D
      DOUBLE PRECISION SINDIF,SINSUM,CSSUM,SSDIF,SSSUM,A,B
      COMPLEX F1
C
C Evaluate force function.
C
      SINDIF = SIN(THETA-THETAO)
      SINSUM = SIN(THETA+THETAO)
      CSDIF  = COS(KAPPAR*SINDIF)
      CSSUM  = COS(KAPPAR*SINSUM)
      SSDIF  = SIN(KAPPAR*SINDIF)
      SSSUM  = SIN(KAPPAR*SINSUM)
      A = CSSUM + CSDIF
      B = SSSUM - SSDIF
      C = A
      D = B
      F1  = CMPLX(C,D)
```

```
C       ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C If debug required output computed values.
C
        if(dfilam.and.(elm.le.1.or.elm.eq.3000))write(26,100)sindif,
     1  sinsum,csdif,cssum,ssdif,sssum,fl
C       ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C Format statements for debug output.
C
  100   format(//,2x,'sindif =',g15.7,/,
        2x,'sinsum =',g15.7,/,
        2x,'csdif =',g15.7,/,
        2x,'cssum =',g15.7,/,
        2x,'ssdif =',g15.7,/,
        2x,'sssum =',g15.7,/,
        2x,'fl =',2g15.7,//)
C
C Return to FORCE subroutine.
C
        RETURN
        END
C ****************************************************************
C
C ****************************************************************
        SUBROUTINE HSPACE(ATRANS,ATEMP,ATEMP2,ATEMP3,BMATRX,
     1  C,CTEMP,FG,FPHI,FD,OUTPUT,CINVER,TEMPD,D)
C
C Declare in COMMON all variables required by more than one
C subroutine.  These variables are divided into four catagories:
C       1. SIZE This common block contains variables which define
C               array dimensions.
C       2. PROB This common block contains variables which define
C               problem parameters.
C       3. MAP  This common block contains variables which define
C             , the finite element mesh.
C       4. dbug This common block contains variables which define
C               which debug switches are desired.
C Arrays are not passed through COMMON but passed as arguments to the
C subroutines.
C
        COMMON  /SIZE/ NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,
     1          NUMEQN,LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
        INTEGER NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,NUMEQN,
     1          LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
        COMMON  /PROB/ W2,GA,GB,RO,EXTG,ALPHA,THETAO
        REAL    W2,GA,GB,RO,EXTG,ALPHA,THETAO
        COMMON  /MAP/ SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2,RATIO,GRADNT
        INTEGER RATIO
        REAL    SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2
        LOGICAL GRADNT
        common  /dbug/ dmain,dmesh,dassm,dktest,dforce,
     1          dhspac,drect,dtri,dgamma,dfilam,damat,dbmat,
     1          dcmat,dnorm,ddiag,ddiag2,dfunc,dderiv,dctest,datest
        logical  dmain,dmesh,dassm,dktest,dforce,dhspac,
     1          drect,dtri,dgamma,dfilam,damat,dbmat,dcmat,dnorm,
     1          ddiag,ddiag2,dfunc,dderiv,dctest,datest
C
C Declare and dimension arrays used in this subroutine.
C
        REAL    OUTPUT(INTNOD),BMATRX(BNDELM,BCOLS)
```

```
      COMPLEX ATEMP(ATMPL,AWIDE),CINVER(BNDELM,BNDELM),
     1        TEMPD(INTNOD,INTNOD),D(INTNOD,INTNOD),
     1        ATRANS(INTNOD,BNDELM),ATEMP2(ALENG,AWIDE),
     1        ATEMP3(ALENG,AWIDE),FD(BNDELM),
     1        C(BNDELM,BNDELM),CTEMP(NUMANG),FG(INTNOD),FPHI(BNDELM)
C
C Declare variables used only in this subroutine.
C
      INTEGER ROW,COL,I,PUSH,IER,COL1,COL2
C
C Determine B matrix.
C
      CALL BMAT(BMATRX,OUTPUT)
C
C Determine A matrix.
C
      CALL AMAT(ATRANS,ATEMP,ATEMP2,ATEMP3)
C
C If a test of the amatrix is desired run ATEST.
C
      IF(DATEST)CALL ATEST(ATRANS,FG,BMATRX)
C
C Complete A matrix by addition of term similar to B matrix.
C
C
C    Iterate trough rows and columns of BMATRX, modify constant and add
C    to appropriate term in ATRANS.

      DO 110 ROW=1,BNDELM
              DO 115 COL=1,BCOLS
                      PUSH=RATIO*(ROW-1)+COL
                      ATRANS(PUSH,ROW)=ATRANS(PUSH,ROW) -
     1                        ALPHA*BMATRX(ROW,COL)
 115          CONTINUE
 110  CONTINUE
C     ++++++++++++++++++++++++++++++++++++++++++++
C If debug desired output ATRANS.
C
      if(.not.dhspac)goto 113
      write(26,5000)
      do 111 row=1,intnod
              write(26,1000)(atrans(row,col),col=1,bndelm)
 111  continue
C     ++++++++++++++++++++++++++++++++++++++++++++
C
C Determine C matrix.
C
 113  CALL CMAT(C,CTEMP)
C
C If test of C matrix is desired run CTEST.
C
      IF(DCTEST)CALL CTEST(FPHI,C)
C     ++++++++++++++++++++++++++++++++++++++++++++
C If debug desired output C matrix.
C
      if(.not.dhspac)goto 36
      write(26,2000)
      do 35 row=1,bndelm
              write(26,1000)(c(row,col),col=1,bndelm)
  35  continue
```

```
C       ++++++++++++++++++++++++++++++++++++++++++++++
C
C Perform condensation of exterior matrices to one matrix
C representing the exterior at the boundary elements only.
C
C
C Store C matrix in temporary storage so it can be inverted without
C destruction.
C
   36   DO 5 ROW=1,BNDELM
                DO 10 COL=1,BNDELM
                        ATEMP2(ROW,COL)=C(ROW,COL)
   10           CONTINUE
    5   CONTINUE
C
C Invert C matrix according to instuctions in IMSL manual for LEQT1C.
C
        CALL LEQT1C(ATEMP2,BNDELM,ALENG,CTEMP,1,BNDELM,1,OUTPUT,IER)
        CTEMP(1) = CMPLX(1.0,0.0)
        DO 15 ROW=2,BNDELM
                CTEMP(ROW) = CMPLX(0.0,0.0)
   15   CONTINUE
        DO 20 COL=1,BNDELM
            CALL LEQT1C(ATEMP2,BNDELM,ALENG,CTEMP,1,BNDELM,2,OUTPUT,IER)
            DO 25 ROW=1,BNDELM
                        CINVER(ROW,COL) = CTEMP(ROW)
                        CTEMP(ROW) = CMPLX(0.0,0.0)
                        IF(ROW.EQ.COL+1)CTEMP(ROW) = CMPLX(1.0,0.0)
   25       CONTINUE
   20   CONTINUE
C       ++++++++++++++++++++++++++++++++++++++++++++++++++
C If debug required output inverse of C matrix.
C
        if(.not.dhspac)goto 31
        write(26,2500)
        do 30 row=1,bndelm
                write(26,1000)(cinver(row,col),col=1,bndelm)
   30   continue
C       ++++++++++++++++++++++++++++++++++++++++++++++++++
C
C Initialize product of C inverse transpose times B to zero.
C
   31   DO 40 ROW=1,BNDELM
                DO 45 COL=1,INTNOD
                        TEMPD(ROW,COL) = CMPLX(0.0,0.0)
   45           CONTINUE
   40   CONTINUE
C
C Multiply C inverse transpose times B. Actual method is to multiply B
C transpose times C inverse and then transposing the result. This is
C done because of the way B is stored.
C
        DO 50 COL1=1,BNDELM
                DO 55 ROW=1,BNDELM
                        PUSH = RATIO*(ROW-1)
                        DO 60 COL2=1,BCOLS
                                TEMPD(COL1,PUSH+COL2)=
     1                              TEMPD(COL1,PUSH+COL2) +
     1                              BMATRX(ROW,COL2)*CINVER(ROW,COL1)
   60                   CONTINUE
```

```
   55           CONTINUE
   50   CONTINUE
C       ++++++++++++++++++++++++++++++++++++++++++++++++++
C If debug desired output C inverse transpose times B.
C
        if(.not.dhspac)goto 66
        write(26,3000)
        do 65 row=1,bndelm
                write(26,1000)(tempd(row,col),col=1,intnod)
   65   continue
C       ++++++++++++++++++++++++++++++++++++++++++++++++++
C
C Multiply A transpose times C inverse transpose times B.
C
   66   DO 70 COL1=1,INTNOD
                DO 75 ROW=1,INTNOD
                        ATEMP2(ROW,COL1) = CMPLX(0.0,0.0)
                        DO 80 COL2=1,BNDELM
                                ATEMP2(ROW,COL1)=ATEMP2(ROW,COL1) +
     1                             ATRANS(ROW,COL2)*TEMPD(COL2,COL1)
   80                   CONTINUE
   75           CONTINUE
   70   CONTINUE
C       ++++++++++++++++++++++++++++++++++++++++++++++++++
C If debug desired output A transpose times C inverse times B.
C
        if(.not.hspac) goto 86
        write(26,3500)
        do 85 row=1,intnod
                write(26,1000)(ATEMP2(row,col),col=1,intnod)
   85   continue
C       ++++++++++++++++++++++++++++++++++++++++++++++++++
C
C Complete halfspace matrix D by taking negative of the sum of the
C matrix just computed and its transpose.
C
   86   DO 90 ROW=1,INTNOD
                DO 95 COL=1,INTNOD
                        D(ROW,COL)=-ATEMP2(ROW,COL)-ATEMP2(COL,ROW)
   95           CONTINUE
   90   CONTINUE
C       ++++++++++++++++++++++++++++++++++++++++++++++++
C If debug desired output D matrix.
C
        if(.not.dhspac) goto 101
        write(26,4000)
        do 100 row=1,intnod
                write(26,1000)(d(row,col),col=1,intnod)
  100   continue
C       ++++++++++++++++++++++++++++++++++++++++++++++++
C
C Condense external force vectors to a single force vector at the
C boundary.
C
C
C Determine FD.
C
C   Multiply B transpose times C inverse times FPHI.
C
  101   DO 165 ROW=1,INTNOD
```

```
                FD(ROW) = CMPLX(0.0,0.0)
                DO 170 COL=1,BNDELM
                        FD(ROW) = FD(ROW)+TEMPD(COL,ROW)*FPHI(COL)
  170           CONTINUE
  165   CONTINUE
C$      write(26,*)(fd(row),row=1,intnod)
C
C   Multiply A transpose times C inverse transpose.
C
        DO 175 COL1=1,BNDELM
                DO 180 ROW=1,INTNOD
                        TEMPD(ROW,COL1) = CMPLX(0.0,0.0)
                        DO 185 COL2=1,BNDELM
                                TEMPD(ROW,COL1) = TEMPD(ROW,COL1) +
     1                             ATRANS(ROW,COL2)*CINVER(COL1,COL2)
  185                   CONTINUE
  180           CONTINUE
  175   CONTINUE
        if(.not.dhspac) goto 177
        write(26,5500)
        do 176 row=1,intnod
                write(26,1000)(tempd(row,col),col=1,bndelm)
  176   continue
C
C   Multiply A transpose times C inverse transpose times FPHI.
C
  177   DO 190 ROW=1,INTNOD
                DO 195 COL=1,BNDELM
                        FD(ROW) = FD(ROW)-TEMPD(ROW,COL)*FPHI(COL)
  195           CONTINUE
  190   CONTINUE
C$      write(26,*)(fd(row),row=1,intnod)
C
C Debug format statements.
C
 5500   FORMAT(/2X,'********* A TRANSPOSE TIMES C INVERSE TRANSPOSE
     1   ********')
 5000   FORMAT(/2X,'********* ATRANS *********')
 4000   FORMAT(/2X,'********* D MATRIX *********')
 3500   FORMAT(/2X,'********* ATRANS TIMES CINVER TIMES B *********')
 3000   FORMAT(/2X,'********* C INVERS TRANSPOSE TIMES B *********')
 2500   FORMAT(/2X,'********* C INVERSE *********')
 2000   FORMAT(/2X,'********* C MATRIX *********')
 1000   FORMAT(1X,100F7.4)
        return
        END
C ***************************************************************
C
C ***************************************************************
        SUBROUTINE BMAT(BMATRX,OUTPUT)
C
C Declare in COMMON all variables required by more than one
C subroutine.  These variables are divided into four catagories:
C       1. SIZE This common block contains variables which define
C               array dimensions.
C       2. PROB This common block contains variables which define
C               problem parameters.
C       3. MAP  This common block contains variables which define
C               the finite element mesh.
C       4. dbug This common block contains variables which define
```

```
C               which debug switches are desired.
C Arrays are not passed through COMMON but passed as arguments to the
C subroutines.
C
        COMMON  /SIZE/ NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,
     1          NUMEQN,LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
        INTEGER NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,NUMEQN,
     1          LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
        COMMON  /PROB/ W2,GA,GB,RO,EXTG,ALPHA,THETAO
        REAL    W2,GA,GB,RO,EXTG,ALPHA,THETAO
        COMMON  /MAP/ SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2,RATIO,GRADNT
        INTEGER RATIO
        REAL    SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2
        LOGICAL GRADNT
        common  /dbug/ dmain,dmesh,dassm,dktest,dforce,
     1          dhspac,drect,dtri,dgamma,dfilam,damat,dbmat,
     1          dcmat,dnorm,ddiag,ddiag2,dfunc,dderiv,dctest,datest
        logical  dmain,dmesh,dassm,dktest,dforce,dhspac,
     1          drect,dtri,dgamma,dfilam,damat,dbmat,dcmat,dnorm,
     1          ddiag,ddiag2,dfunc,dderiv,dctest,datest
C
C Declare and dimension arrays used in this subroutine.
C
        REAL    BMATRX(BNDELM,BCOLS),OUTPUT(INTNOD)
C
C Declare variables used in this subroutine only.
C
        INTEGER I,J,ELM,ROW,COL
        REAL    ANS
C
C Calculate integral from derived expression. The terms of the integral
C for both shape functions are the same.
C
        ANS=0.25*DANG
C
C Position these values in the proper location of the B matrix. The
C positioning is derived from the use of a circulant matrix which is
C condensed to the one for a half-circle. Only the non-zero terms are
C stored.
C
        DO 60 ROW=1,BNDELM
                BMATRX(ROW,1)=2.0*ANS
                BMATRX(ROW,BCOLS)=2.0*ANS
                IF(BCOLS.EQ.2) GOTO 60
                DO 70 COL=2,BCOLS-1
                        BMATRX(ROW,COL)=2.0*ANS
   70           CONTINUE
   60   CONTINUE
C       ++++++++++++++++++++++++++++++++++++++++++++
C If debug is desired output both the stored version of the B matrix and
C the full representation for the half-circle. Otherwise return to
C HSPACE subroutine.
C
        if(.not.dbmat)RETURN
        write(26,300)
        do 40 i=1,bndelm
                do 50 j=1,bcols
                        output(j)=BMATRX(i,j)
   50           continue
                write(26,200) (output(j),j=1,bcols)
```

```
   40   continue
        write(26,400) (i,i=1,intnod)
        do 10 row=1,bndelm
                do 20 col=1,intnod
                        output(col)=0.0
   20           continue
                push = ratio*(row-1)
                do 30 col=1,ratio+1
                        output(push+col)=bmatrx(row,col)
   30           continue
                write(26,100) row,(output(i),i=1,intnod)
   10   continue
C       ++++++++++++++++++++++++++++++++++++++++++++
C
C Debug format statements
C
  100   format(2x,i2,1x,100f11.6)
  200   format(2x,10f6.3)
  300   format(/2x,'********* compact b matrix *********',/)
  400   format(//2x,'********* full b matrix ***********',//
     1  ,7x,100(i2,4x))
C
C Return to HSPACE subroutine.
C
        return
        END
C*****************************************************************************
C
C*****************************************************************************
        SUBROUTINE AMAT(ATRANS,ATEMP,ATEMP2,ATEMP3)
C
C Declare in COMMON all variables required by more than one
C subroutine.  These variables are divided into four catagories:
C       1. SIZE This common block contains variables which define
C               array dimensions.
C       2. PROB This common block contains variables which define
C               problem parameters.
C       3. MAP  This common block contains variables which define
C               the finite element mesh.
C       4. dbug This common block contains variables which define
C               which debug switches are desired.
C Arrays are not passed through COMMON but passed as arguments to the
C subroutines.
C
        COMMON  /SIZE/ NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,
     1          NUMEQN,LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
        INTEGER NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,NUMEQN,
     1          LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
        COMMON  /PROB/ W2,GA,GB,RO,EXTG,ALPHA,THETAO
        REAL    W2,GA,GB,RO,EXTG,ALPHA,THETAO
        COMMON  /MAP/ SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2,RATIO,GRADNT
        INTEGER RATIO
        REAL    SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2
        LOGICAL GRADNT
        common  /dbug/ dmain,dmesh,dassm,dktest,dforce,
     1          dhspac,drect,dtri,dgamma,dfilam,damat,dbmat,
     1          dcmat,dnorm,ddiag,ddiag2,dfunc,dderiv,dctest,datest
        logical  dmain,dmesh,dassm,dktest,dforce,dhspac,
     1          drect,dtri,dgamma,dfilam,damat,dbmat,dcmat,dnorm,
     1          ddiag,ddiag2,dfunc,dderiv,dctest,datest
```

```
C
C Declare in COMMON the Gauss integration points.
C
      COMMON  /INGRAL/ NUMPTY,YVALUE,WGTY,NUMPTZ,ZVALUE,WGTZ
      INTEGER NUMPTY,NMPTYD,NUMPTZ
C
C Declare and dimension arrays required in this subroutine.
C
      COMPLEX ATEMP(2,ALENG),TEMP,
     1 ATRANS(INTNOD,BNDELM),ATEMP2(ATMPL,AWIDE),ATEMP3(ALENG,AWIDE)
      REAL    YVALUE(100),WGTY(100),ZVALUE(100),WGTZ(100)
C
C Declare variables used only in this subroutine.
C
      INTEGER I,ROW,COL,SOURCE,PULL1,PULL2,OBSERV,PUSH,LIMIT,
     1        PULL,WOR,PUSH1,PUSH2
      REAL    QAOBS,QBOBS,QASRC,QBSRC,TEST
      LOGICAL ODD
C
C Read in Gauss integration points.
C
      READ(24,*) NUMPTY
      DO 5 I=1,NUMPTY/2
              READ(24,*) YVALUE(I),WGTY(I)
    5 CONTINUE
      READ(24,*) NUMPTZ
      DO 10 I=1,NUMPTZ/2
              READ(24,*) ZVALUE(I),WGTZ(I)
   10 CONTINUE
C     ++++++++++++++++++++++++++++++++++++
      if(damat)write(26,7000) numptz,numpty
C     ++++++++++++++++++++++++++++++++++++
C
C Determine if number of interior boundary elements per exterior
C boundary element is odd or even. Set limits on the number of observer
C integrals necessary.
C
      ODD = .FALSE.
      LIMIT = RATIO/2
      TEST = RATIO - 2*LIMIT
      IF(TEST.NE.0) ODD = .TRUE.
      IF(LIMIT.EQ.0) GOTO 16
C
C Perform double integral for A matrix. Outer integrals are the
C observers.
C
C
C    Iterate over necessary observers.
C
      DO 30 OBSERV=1,LIMIT
C             ++++++++++++++++++++++++++
              if(damat)write(26,500) observ
C             ++++++++++++++++++++++++++
C
C    Determine limits of integration for observer.
C
              QAOBS = (OBSERV-1)*DANG
              QBOBS = OBSERV*DANG
C             ++++++++++++++++++++++++++
              if(damat)write(26,600)qaobs,qbobs
```

```
C               ++++++++++++++++++++++++++++
C
C    Iterate over necessary sources.
C
                DO 15 SOURCE=1,BNDELM*2
C                       ++++++++++++++++++++++++++++
                        if(damat)write(26,700) source
C                       ++++++++++++++++++++++++++++
C
C       Determine limits of integration for the source.
C
                        QASRC = (SOURCE-1)*BDANG1
                        QBSRC = SOURCE*BDANG1
C                       ++++++++++++++++++++++++++++
                        if(damat)write(26,800)qasrc,qbsrc
C                       ++++++++++++++++++++++++++++
C
C        Determine temporary storage slot for observer-source
C        integral.
C
                        PUSH = (OBSERV-1)*AWIDE+SOURCE
C
C       Evaluate integral.
C
                        IF(SOURCE.EQ.1) GOTO 11
                        CALL NORM2(QAOBS,QBOBS,QASRC,QBSRC,
     1                          ATEMP(1,PUSH),ATEMP(2,PUSH))
                        GOTO 12
   11                   CALL NORM(QAOBS,QBOBS,QASRC,QBSRC,
     1                          ATEMP(1,PUSH),ATEMP(2,PUSH))
C                       ++++++++++++++++++++++++++++++++++++++++++++
   12                   if(damat)write(26,900)push,atemp(1,push),push,
     1                        atemp(2,push)
C                       ++++++++++++++++++++++++++++++++++++++++++++
   15           CONTINUE
   30   CONTINUE
C
C If number of internal boundary elements per external boundary element
C is odd perform integral for the odd observer. This is done here since
C the number of sources is less than for the other observers due to
C symmetry.
C
                IF(.NOT.ODD) GOTO 20
   16           OBSERV = LIMIT+1
C
C    Determine limits of integration for observer.
C
                QAOBS = (OBSERV-1)*DANG
                QBOBS = OBSERV*DANG
C               ++++++++++++++++++++++++++++++++
                if(damat)write(26,600)qaobs,qbobs
C               ++++++++++++++++++++++++++++++++
C
C    Iterate over necessary sources.
C
                DO 25 SOURCE=1,BNDELM+1
C                       ++++++++++++++++++++++++++++++++
                        if(damat)write(26,700) source
C                       ++++++++++++++++++++++++++++++++
C
```

```
C       Determine limits of integration for source.
C
                        QASRC = (SOURCE-1)*BDANG1
                        QBSRC = SOURCE*BDANG1
C                       ++++++++++++++++++++++++++++++++
                        if(damat)write(26,800)qasrc,qbsrc
C                       ++++++++++++++++++++++++++++++++
C
C        Determine temporary storage slot for observer-source
C        integral.
C
                        PUSH = (OBSERV-1)*AWIDE+SOURCE
                        IF(SOURCE.EQ.1)GOTO 21
C
C       Evaluate integral.
C
                        CALL NORM2(QAOBS,QBOBS,QASRC,QBSRC,
     1                          ATEMP(1,PUSH),ATEMP(2,PUSH))
                        GOTO 22
   21                   CALL NORM(QAOBS,QBOBS,QASRC,QBSRC,
     1                          ATEMP(1,PUSH),ATEMP(2,PUSH))
C                       ++++++++++++++++++++++++++++++++++++++++++++++
   22                   if(damat)write(26,900)push,atemp(1,push),push,
     1                        atemp(2,push)
C                       ++++++++++++++++++++++++++++++++++++++++++++++
   25           CONTINUE

C       ++++++++++++++++++++++++++++++++++++++++++++
C If debug desired output computed integrals.
C
   20   if(.not.damat) goto 36
        write(26,300)
        limit=ratio*bndelm
        if(odd)limit=limit+1
        do 35 col=1,limit
                write(26,400)(atemp(row,col),row=1,2)
   35   continue
C       ++++++++++++++++++++++++++++++++++++++++++++
C
C Determine if number of interior boundary elements per exterior
C boundary element is odd or even. Set limits on the length of A
C matrix and test value.
C
   36   LIMIT = RATIO/2
        IF(ODD) LIMIT = LIMIT+1
        TEST = RATIO*BNDELM
        IF(ODD) TEST=TEST+1
C-----------------------------------------------
C CREATE UNIT MATRIX
        DO 50 ROW=1,LIMIT
                PUSH = ROW*2
                DO 40 COL=1,BNDELM*2
                        PULL=(ROW-1)*BNDELM*2+COL
                        IF(PULL.GT.TEST) GOTO 37
                        ATEMP2(PUSH-1,COL)=ATEMP(1,PULL)
                        ATEMP2(PUSH,COL)=ATEMP(2,PULL)
                        GOTO 40
   37                   PULL = 2*TEST-PULL
                        ATEMP2(PUSH-1,COL)=ATEMP(2,PULL)
                        ATEMP2(PUSH,COL)=ATEMP(1,PULL)
```

```
   40            CONTINUE
                 IF(ROW.EQ.LIMIT.AND.ODD)GOTO 50
                 WOR = RATIO+1-ROW
                 PUSH = WOR*2
                 DO 45 COL=1,BNDELM*2
                         PULL=2+ROW*BNDELM*2-COL
                         IF(PULL.GT.ROW*BNDELM*2)PULL=PULL-BNDELM*2
                         ATEMP2(PUSH-1,COL)=ATEMP(1,PULL)
                         ATEMP2(PUSH,COL)=ATEMP(2,PULL)
   45            CONTINUE
   50    CONTINUE
         if(.not.damat) goto 56
         write(26,2000)
         do 55 row=1,ratio*2
                 write(26,7600)(atemp2(row,col),col=1,bndelm*2)
   55    continue
C SETUP BIG A MATRIX
   56    DO 70 I=1,BNDELM*2
                 DO 65 ROW=1,RATIO*2
                         PUSH1 = ROW+(I-1)*RATIO*2
                         DO 60 COL=1,BNDELM*2
                                 PUSH2 = COL+I-1
                                 IF(PUSH2.GT.BNDELM*2)PUSH2 =
     1                                   PUSH2-2*BNDELM
                                 ATEMP3(PUSH1,PUSH2)=ATEMP2(ROW,COL)
   60                    CONTINUE
   65            CONTINUE
   70    CONTINUE
         if(.not.damat)goto 72
         write(26,7100)
         do 71 row=1,bndelm*ratio*4
                 write(26,7600)(atemp3(row,col),col=1,bndelm*2)
   71    continue
C ADD ROWS OF COMMON NODES
   72    DO 75 COL=1,BNDELM*2
                 ATEMP3(1,COL)= ATEMP3(1,COL)+ATEMP3(BNDELM*RATIO*4,COL)
   75    CONTINUE
         DO 85 ROW=2,BNDELM*RATIO*4-2
                 PULL = 2*ROW-2
                 DO 80 COL=1,BNDELM*2
                         ATEMP3(ROW,COL) = ATEMP3(PULL,COL) +
     1                                   ATEMP3(PULL+1,COL)
   80            CONTINUE
   85    CONTINUE
         if(.not.damat)goto 87
         write(26,7200)
         do 86 row=1,bndelm*ratio*2
                 write(26,7600)(atemp3(row,col),col=1,bndelm*2)
   86    continue
C FLIP RIGHT HAND GROUP OF COLUMNS
   87    DO 95 ROW=1,BNDELM*RATIO*2
                 DO 90 I=1,BNDELM/2
                         PUSH1 = BNDELM+I
                         PUSH2 = 2*BNDELM-(I-1)
                         TEMP  = ATEMP3(ROW,PUSH1)
                         ATEMP3(ROW,PUSH1) = ATEMP3(ROW,PUSH2)
                         ATEMP3(ROW,PUSH2) = TEMP
   90            CONTINUE
   95    CONTINUE
         if(.not.damat)goto 97
```

```
      write(26,7300)
      do 96 row=1,bndelm*ratio*2
              write(26,7600)(atemp3(row,col),col=1,bndelm*2)
   96 continue
C FLIP LOWER GROUP OF ROWS
   97 DO 105 COL=1,BNDELM*2
              LIMIT = (BNDELM*RATIO)/2
              DO 100 I=1,LIMIT
                      PUSH1 = BNDELM*RATIO+I
                      PUSH2 = BNDELM*RATIO*2-(I-1)
                      TEMP  = ATEMP3(PUSH1,COL)
                      ATEMP3(PUSH1,COL) = ATEMP3(PUSH2,COL)
                      ATEMP3(PUSH2,COL) = TEMP
  100         CONTINUE
  105 CONTINUE
      if(.not.damat)goto 107
      write(26,7400)
      do 106 row=1,bndelm*ratio*2
              write(26,7600)(atemp3(row,col),col=1,bndelm*2)
  106 continue
C CONDENSE PHI
  107 DO 115 ROW=1,BNDELM*RATIO*2
              DO 110 COL=1,BNDELM
                      PULL = BNDELM+COL
                      ATEMP3(ROW,COL) = ATEMP3(ROW,COL) +
     1                                  ATEMP3(ROW,PULL)
  110         CONTINUE
  115 CONTINUE
      if(.not.damat)goto 117
      write(26,7500)
      do 116 row=1,bndelm*ratio*2
              write(26,7600)(atemp3(row,col),col=1,bndelm)
  116 continue
C CONDENSE DISPLACEMENT
  117 DO 125 COL=1,BNDELM
              DO 120 ROW=2,BNDELM*RATIO
                      PULL = BNDELM*RATIO+(ROW-1)
                      ATEMP3(ROW,COL) = ATEMP3(ROW,COL) +
     1                                  ATEMP3(PULL,COL)
  120         CONTINUE
  125 CONTINUE
      DO 130 COL=1,BNDELM
              ATEMP3(BNDELM*RATIO+1,COL)=ATEMP3(BNDELM*RATIO*2,COL)
  130 CONTINUE
      DO 132 ROW=1,INTNOD
              DO 131 COL=1,BNDELM
                      ATRANS(ROW,COL)=ATEMP3(ROW,COL)
  131         CONTINUE
  132 CONTINUE
      if(.not.damat)RETURN
      write(26,3000)
      do 135 row=1,intnod
              write(26,7600)(atrans(row,col),col=1,bndelm)
              if(row.eq.1.or.row.eq.bndelm*ratio)write(26,6000)
  135 continue
  300 format(/,2x,'******* atemp transpose ********',/)
  400 format(2x,100('(',2f11.6,')'))
  500 format(2x,'observ =',i5)
  600 format(2x,'  qaobs=',f11.6,' qbobs=',f11.6)
  700 format(2x,'    source =',i5)
```

```
  800   format(2x,'      qasrc=',f11.6,' qbsrc=',f11.6)
  900   format(2x,'      atemp(1,',i3,') =',2f11.6,/,
     1  2X,'      atemp(2,',i3,') =',2f11.6)
 1000   format(2x,8F6.3,' |',8F6.3)
 2000   format(/,2x,'*********** atemp2 ************')
 3000   format(/,2x,'*********** atrans ************')
 5000   format(2x,8F6.3)
 6000   format(2x,'--------------------')
 7000   format(/,2x,'number of points for dz integral =',i5,/,
     1  2x,'number of points for dy integral =',i5)
 7100   format(/,2x,'******** full a matrix ********')
 7200   format(/,2x,'******** adding common nodes ********')
 7300   format(/,2x,'******** flipping righthand columns ********')
 7400   format(/,2x,'******** flipping lower rows ********')
 7500   format(/,2x,'******** condensing out phi ********')
 7600   format(2x,20F6.3)
        return
        END
C*********************************************************************
C
C*********************************************************************
        SUBROUTINE NORM(C,D,E,F,INT1,INT2)
C
C Declare in COMMON all variables required by more than one
C subroutine.  These variables are divided into four catagories:
C       1. SIZE This common block contains variables which define
C               array dimensions.
C       2. PROB This common block contains variables which define
C               problem parameters.
C       3. MAP  This common block contains variables which define
C               the finite element mesh.
C       4. dbug This common block contains variables which define
C               which debug switches are desired.
C Arrays are not passed through COMMON but passed as arguments to the
C subroutines.
C
        COMMON  /SIZE/ NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,
     1          NUMEQN,LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
        INTEGER NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,NUMEQN,
     1          LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
        COMMON  /PROB/ W2,GA,GB,RO,EXTG,ALPHA,THETAO
        REAL    W2,GA,GB,RO,EXTG,ALPHA,THETAO
        COMMON  /MAP/ SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2,RATIO,GRADNT
        INTEGER RATIO
        REAL    SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2
        LOGICAL GRADNT
        common  /dbug/ dmain,dmesh,dassm,dktest,dforce,
     1          dhspac,drect,dtri,dgamma,dfilam,damat,dbmat,
     1          dcmat,dnorm,ddiag,ddiag2,dfunc,dderiv,dctest,datest
        logical  dmain,dmesh,dassm,dktest,dforce,dhspac,
     1          drect,dtri,dgamma,dfilam,damat,dbmat,dcmat,dnorm,
     1          ddiag,ddiag2,dfunc,dderiv,dctest,datest
C
C Declare in COMMON the Gauss integration points.
C
        COMMON  /INGRAL/ NUMPTY,YVALUE,WGTY,NUMPTZ,ZVALUE,WGTZ
        INTEGER NUMPTY,NMPTYD,NUMPTZ
        REAL    YVALUE(100),WGTY(100),ZVALUE(100),WGTZ(100)
C
C Declare variables used only in this subroutine.
```

```
C
      INTEGER I,ZI,YI
      REAL    A,B,C,D,Z,Y,EPSLON,DIST,CONST
      COMPLEX INTGYZ,INTGY,ANSY,INT1,INT2,LN,X2,DERIV
      LOGICAL DONE
C     ++++++++++++++++++++++++++++++++++
C     if debug required output number of integration points.
C
      if(dnorm)write(26,100)numpty,numptz
C     ++++++++++++++++++++++++++++++++++
C
C Define required constants.
C
      CONST = 1.0/(2.0*(D-C))
C
C   Initialize integrals.
C
      INT1 = CMPLX(0.0,0.0)
      INT2 = CMPLX(0.0,0.0)
C
C   Integrate with respect to dz over each element.
C
C
C     Iterate through each Gauss integration point.
C
      DO 50 ZI=1,NUMPTZ/2
C
C        Transform -1 to 1 positive integration points to c to d points.
C
             Z = (ZVALUE(ZI)*(D-C)+D+C)/2.0
             DONE = .FALSE.
             GOTO 30
C        Transform -1 to 1 integration points to c to d points.
C        This is done after the positive point has been integrated
C        with respect to dy.
C
   25        Z= (-ZVALUE(ZI)*(D-C)+D+C)/2.0
             DONE = .TRUE.
C
C        For each value from the dz integral integrate with respect
C        to dy.
C
C            Initialize dy integral to zero.
C
   30        INTGY = CMPLX(0.0,0.0)
C            ++++++++++++++++++++++++++++
             if(dnorm)write(26,500) zi,z
C            ++++++++++++++++++++++++++++
C
C            Iterate through each Gauss integration point.
C
             DO 40 YI=1,NUMPTY/2
C
C            The dy integral is divided into two integrals to
C            handle the logrithmic singularity. The first is from
C            the lower limit of the source to the z value.
C
                     A = E
                     B=Z
C            Transform -1 to 1 positive integration points to e to
```

```
C               z points.
C
                Y = (YVALUE(YI)*(B-A)+B+A)/2.0
C               ++++++++++++++++++++++++++++
                if(dnorm)write(26,550) yi,y
C               ++++++++++++++++++++++++++++
C               Evaluate derivitive of Green's Function at integration
C               points. Multiply by scaling factor and sum for total
C               dy integral.
C
                DIST = ABS(SIN((Z-Y)/2.0))
                ANSY = DERIV(DIST)*(B-A)/2.0
C               ++++++++++++++++++++++++++++
                if(dnorm)write(26,600)ansy
C               ++++++++++++++++++++++++++++
C
C               Transform -1 to 1 negative integration points to e to
C               z points.
C
                Y = (-YVALUE(YI)*(B-A)+B+A)/2.0
C               ++++++++++++++++++++++++++++
                if(dnorm)write(26,550) yi,y
C               ++++++++++++++++++++++++++++
C               Evaluate derivitive of Green's Function at integration
C               points. Multiply by scaling factor and sum for total
C               dy integral.
C
                DIST = ABS(SIN((Z-Y)/2.0))
                ANSY = ANSY + DERIV(DIST)*(B-A)/2.0
C               ++++++++++++++++++++++++++++
                if(dnorm)write(26,600)ansy
C               ++++++++++++++++++++++++++++
C
C               The second portion of the dy integral is from z to f.
C
                A=Z
                B = F
C               ++++++++++++++++++++++++++++
                if(dnorm)write(26,550) yi,y
C               ++++++++++++++++++++++++++++
C
C               Transform -1 to 1 positive integration points to z to
C               f points.
C
                Y = (YVALUE(YI)*(B-A)+B+A)/2.0
C               Evaluate derivitive of Green's Function at integration
C               points. Multiply by scaling factor and sum for total
C               dy integral.
C
                DIST = ABS(SIN((Z-Y)/2.0))
                ANSY = ANSY + DERIV(DIST)*(B-A)/2.0
C               ++++++++++++++++++++++++++++
                if(dnorm)write(26,600)ansy
C               ++++++++++++++++++++++++++++
C               Transform -1 to 1 negative integration points to z to
C               f points.
C
                Y = (-YVALUE(YI)*(B-A)+B+A)/2.0
C               ++++++++++++++++++++++++++++
                if(dnorm)write(26,550) yi,y
```

```
C                       ++++++++++++++++++++++++++
C               Evaluate derivitive of Green's Function at integration
C               points. Multiply by scaling factor and sum for total
C               dy integral.
C
                        DIST = ABS(SIN((Z-Y)/2.0))
                        ANSY = ANSY + DERIV(DIST)*(B-A)/2.0
C                       ++++++++++++++++++++++++++
                        if(dnorm)write(26,600)ansy
C                       ++++++++++++++++++++++++++
C
C               Multiply dy integral by the integration weights.
C
                        INTGY = INTGY + WGTY(YI)*ANSY
C                       ++++++++++++++++++++++++++
                        if(dnorm)write(26,650)intgy
C                       ++++++++++++++++++++++++++
   40           CONTINUE
C
C          Multiply dy evaluation by dz weights and scaling factors
C          and the required constants and sum to total dz integrals.
C
                INTGYZ = CONST*WGTZ(ZI)*(D-C)/2.0*INTGY
                INT1 = INT1 + INTGYZ*(D-Z)
                INT2 = INT2 + INTGYZ*(Z-C)
                IF(DONE) GOTO 50
                GOTO 25
   50   CONTINUE
C       ++++++++++++++++++++++++++
C If debug required output value of integral.
        if(dnorm)write(26,200) intgyz
C       ++++++++++++++++++++++++++
C
C Debug format statements.
C
  100   format(/2x,'number of points for DY integral =',i5,/,
     1  2x,'number of points for DZ integral =',i5)
  200   format(2x,'dz integral = ',2f11.6)
  500   format(2x,'zi = ',i5,/,2x,'z = ',f11.6)
  550   format(/,2x,'yi = ',i5,/,2x,'y = ',f11.6)
  600   format(2x,'ansy = ',2f11.6)
  650   format(2x,'dy integral = ',2f11.6)
C
C Return to AMAT subroutine.
C
        RETURN
        END
C*********************************************************************
C
C*********************************************************************
        SUBROUTINE NORM2(C,D,E,F,INT1,INT2)
C
C Declare in COMMON all variables required by more than one
C subroutine.  These variables are divided into four catagories:
C       1. SIZE This common block contains variables which define
C               array dimensions.
C       2. PROB This common block contains variables which define
C               problem parameters.
C       3. MAP  This common block contains variables which define
C               the finite element mesh.
```

```
C       4. dbug This common block contains variables which define
C               which debug switches are desired.
C Arrays are not passed through COMMON but passed as arguments to the
C subroutines.
C
        COMMON  /SIZE/ NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,
     1          NUMEQN,LENGTH,WIDTH,SPACE,BNDELM,BCOLS
        INTEGER NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,NUMEQN,
     1          LENGTH,WIDTH,SPACE,BNDELM,BCOLS
        COMMON  /PROB/ W2,GA,GB,RO,EXTG,ALPHA,THETAO
        REAL    W2,GA,GB,RO,EXTG,ALPHA,THETAO
        COMMON  /MAP/ SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2,RATIO,GRADNT
        INTEGER RATIO
        REAL    SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2
        LOGICAL GRADNT
        common  /dbug/ dmain,dmesh,dassm,dktest,dforce,
     1          dhspac,drect,dtri,dgamma,dfilam,damat,dbmat,
     1          dcmat,dnorm,ddiag,ddiag2,dfunc,dderiv,dctest,datest
        logical  dmain,dmesh,dassm,dktest,dforce,dhspac,
     1          drect,dtri,dgamma,dfilam,damat,dbmat,dcmat,dnorm,
     1          ddiag,ddiag2,dfunc,dderiv,dctest,datest
C
C Declare in COMMON the Gauss integration points.
C
        COMMON  /INGRAL/ NUMPTY,YVALUE,WGTY,NUMPTZ,ZVALUE,WGTZ
        INTEGER NUMPTY,NMPTYD,NUMPTZ
        REAL    YVALUE(100),WGTY(100),ZVALUE(100),WGTZ(100)
C
C Declare variables used only in this subroutine.
C
        INTEGER I,ZI,YI
        REAL    A,B,C,D,Z,Y,E,F,DIST,CONST
        COMPLEX INTGYZ,INTGY,ANSY,INT1,INT2,LN,X2,DERIV
        LOGICAL DONE
C       ++++++++++++++++++++++++++++++++++++
C If debug required output number of Gauss integration points.
C
        if(dnorm)write(26,100)numpty,numptz
C       ++++++++++++++++++++++++++++++++++++
C
C Define required constant.
C
        CONST = 1.0/(2.0*(D-C))
C
C Compute double Green's function integral for A matrix.
C
C
C   Initialize integrals.
C
        INT1 = CMPLX(0.0,0.0)
        INT2 = CMPLX(0.0,0.0)
C
C   Integrate with respect to dz over each element.
C
C
C       Iterate through each Gauss integration point.
C
        DO 50 ZI=1,NUMPTZ/2
C
C          Transform -1 to 1 positive integration points to c to d points.
```

```
C
            Z = (ZVALUE(ZI)*(D-C)+D+C)/2.0
            DONE = .FALSE.
            GOTO 30
C
C       Transform -1 to 1 negative integration points to c to d
C       points. This is done after the positive point has been
C       integrated with respect to dy.
C
   25       Z= (-ZVALUE(ZI)*(D-C)+D+C)/2.0
            DONE = .TRUE.
C
C       For each value from the dz integral integrate with respect
C       to dy.
C
C
C           Initialize dy integral to zero.
C
   30       INTGY = CMPLX(0.0,0.0)
C
C           Iterate through each Gauss integration point.
C
            DO 40 YI=1,NUMPTY/2
C
C           Define limits of integration.
C
                    A = E
                    B = F
C
C           Transform -1 to 1 positive integration points to e to
C           f points.
C
                    Y = (YVALUE(YI)*(B-A)+B+A)/2.0
C
C           Evaluate derivative of Green's function at integration
C           points. Multiply by scaling factor and sum for total
C           dy integral.
C
                    DIST = ABS(SIN((Z-Y)/2.0))
                    ANSY = DERIV(DIST)*(B-A)/2.0
C
C           Transform -1 to 1 negative integration points to e to
C           f points.
C
                    Y = (-YVALUE(YI)*(B-A)+B+A)/2.0
C
C           Evaluate derivative of Green's function at integration
C           points. Multiply by scaling factor and sum for total
C           dy integral.
C
                    DIST = ABS(SIN((Z-Y)/2.0))
                    ANSY = ANSY + DERIV(DIST)*(B-A)/2.0
C
C           Multiply dy integral by integration weights.
C
                    INTGY = INTGY + WGTY(YI)*ANSY
   40       CONTINUE
C
C       Multiply dy evaluation by dz weights and scaling factors
C       and the required constants and sum to total dz integrals.
```

```
C
                INTGYZ = CONST*WGTZ(ZI)*(D-C)/2.0*INTGY
                INT1 = INT1 + INTGYZ*(D-Z)
                INT2 = INT2 + INTGYZ*(Z-C)
                IF(DONE) GOTO 50
                GOTO 25
   50   CONTINUE
C       ++++++++++++++++++++++++++++++
C If debug required output values of integrals.
C
        if(dnorm)write(26,200) intgyz
        if(dnorm)write(26,1100) int1,int2
C       ++++++++++++++++++++++++++++++
C
C Debug format statements.
C
  100   format(/2x,        number of points for DY integral =',i5,/,
     1  2x,'      number of points for DZ integral =',i5)
  200   format(2x,'       integral = ',2f11.6)
 1100   format(2x,'       int1 =',2f11.6,/,
        2x,'       int2 =',2f11.6)
C
C Return to AMAT subroutine.
C
        RETURN
        END
C*********************************************************************
C
C*********************************************************************
        COMPLEX FUNCTION DERIV(DIST)
C
C Declare in COMMON all variables required by more than one
C subroutine.  These variables are divided into four catagories:
C       1. SIZE This common block contains variables which define
C               array dimensions.
C       2. PROB This common block contains variables which define
C               problem parameters.
C       3. MAP  This common block contains variables which define
C               the finite element mesh.
C       4. dbug This common block contains variables which define
C               which debug switches are desired.
C Arrays are not passed through COMMON but passed as arguments to the
C subroutines.
C
                COMMON  /SIZE/ NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,
     1          NUMEQN,LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
        INTEGER NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,NUMEQN,
     1          LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
        COMMON  /PROB/ W2,GA,GB,RO,EXTG,ALPHA,THETAO
        REAL    W2,GA,GB,RO,EXTG,ALPHA,THETAO
        COMMON  /MAP/ SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2,RATIO,GRADNT
        INTEGER RATIO
        REAL    SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2
        LOGICAL GRADNT
        common  /dbug/ dmain,dmesh,dassm,dktest,dforce,
     1          dhspac,drect,dtri,dgamma,dfilam,damat,dbmat,
     1          dcmat,dnorm,ddiag,ddiag2,dfunc,dderiv,dctest,datest
        logical  dmain,dmesh,dassm,dktest,dforce,dhspac,
     1          drect,dtri,dgamma,dfilam,damat,dbmat,dcmat,dnorm,
     1          ddiag,ddiag2,dfunc,dderiv,dctest,datest
```

```
C
C Declare variables used in this function only.
C
      REAL    KAPPA,X,DIST,ARG,F1,THETA,Y1,J1
C
C Define necessary constant.
C
      KAPPA = SQRT(W2*RANGE/EXTG)
C
C Compute argument of Green's function derivative.
C
      X = 2.0*KAPPA*RANGE*DIST
C
C Test argument to determine proper formula for computation.
C
      IF(X.LT.3.0) GOTO 10
C
C Compute derivative of Green's function.
C
      ARG = 3.0/X
      F1 = (((((-.00020033*ARG+.00113653)*ARG-.00249511)*ARG
     1       +.00017105)*ARG+.01659667)*ARG+.00000156)*ARG
     1       +.79788456
      THETA = (((((-.00029166*ARG+.00079824)*ARG+.00074348)*ARG
     1          -.00637879)*ARG+.00005650)*ARG+.12499612)*ARG
     1          -2.35619449+X
      J1 = F1*COS(THETA)/SQRT(X)*KAPPA*DIST/(-4.0)
      Y1 = F1*SIN(THETA)/SQRT(X)*KAPPA*DIST/(4.0)
C     ++++++++++++++++++++++++++++++++++++++++++
C If debug desired output computed values.
C
      if(dderiv)write(26,500)kappa,dist,x,j1,y1
C     ++++++++++++++++++++++++++++++++++++++++++
      DERIV = CMPLX(Y1,J1)
C
C Return to NORM or NORM2 subroutines.
C
      RETURN
C
C Compute derivative of Green's function.
C
   10 ARG = (X/3.0)**2
      F1 = ((((((.00001109*ARG-.00031761)*ARG+.00443319)*ARG
     1       -.03954289)*ARG+.21093573)*ARG-.56249985)*ARG+0.5)*X
      THETA = ((((((.0027873*ARG-.0400976)*ARG+.3123951)*ARG
     1          -1.3164827)*ARG+2.1682709)*ARG+.2212091)*ARG
     1          -.6366198+2/3.1415927*X*ALOG(0.5*X)*F1)/X
      J1 = F1*KAPPA*DIST/(-4.0)
      Y1 = THETA*KAPPA*DIST/(4.0)
C     ++++++++++++++++++++++++++++++++++++++++++
C If debug desired output computed values.
C
      if(dderiv)write(26,500)kappa,dist,x,j1,y1
C     ++++++++++++++++++++++++++++++++++++++++++
      DERIV = CMPLX(Y1,J1)
C
C Return to NORM or NORM2 subroutines.
C
      RETURN
C
```

```
C Debug format statements.
C
  500   format(2x,'kappa = ',f11.6,/.
     1  2x,'distance = ',f11.6,/,
     1  2x,'x = ',f11.6,/,
     1  2x,'jl = ',f11.6,/,
     1  2x,'yl = ',f11.6,/,
     1  2x,'func = cmplx(yl,jl)')
        END
C***********************************************************************
C
C***********************************************************************
        SUBROUTINE ATEST(ATRANS,FG,BMATRX)
C
C Declare in COMMON all variables required by more than one
C subroutine.  These variables are divided into four catagories:
C       1. SIZE This common block contains variables which define
C               array dimensions.
C       2. PROB This common block contains variables which define
C               problem parameters.
C       3. MAP  This common block contains variables which define
C               the finite element mesh.
C       4. dbug This common block contains variables which define
C               which debug switches are desired.
C Arrays are not passed through COMMON but passed as arguments to the
C subroutines.
C
                COMMON  /SIZE/ NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,
     1          NUMEQN,LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
        INTEGER NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,NUMEQN,
     1          LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
        COMMON  /PROB/ W2,GA,GB,RO,EXTG,ALPHA,THETAO
        REAL    W2,GA,GB,RO,EXTG,ALPHA,THETAO
        COMMON  /MAP/ SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2,RATIO,GRADNT
        INTEGER RATIO
        REAL    SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2
        LOGICAL GRADNT
        common  /dbug/ dmain,dmesh,dassm,dktest,dforce,
     1          dhspac,drect,dtri,dgamma,dfilam,damat,dbmat,
     1          dcmat,dnorm,ddiag,ddiag2,dfunc,dderiv,dctest,datest
        logical  dmain,dmesh,dassm,dktest,dforce,dhspac,
     1          drect,dtri,dgamma,dfilam,damat,dbmat,dcmat,dnorm,
     1          ddiag,ddiag2,dfunc,dderiv,dctest,datest
        INTEGER COL1,COL2,ROW,IER
        REAL    WA(15),BMATRX(BNDELM,BCOLS),TMPB(15,15),TEMPB(15,16)
        COMPLEX ATRANS(INTNOD,BNDELM),
     1          FG(INTNOD),TMPA(15,15),TMPFG(15),ANS(15),TEMPA(15,16)
     1          ,TMPAT(16,15),TMPAAT(15,15)
        WRITE(26,2000)
        IF(INTNOD.GT.16.OR.BNDELM.GT.15) GOTO 100
C Write BMATRX
        WRITE(26,1100)
        DO 5 ROW=1,BNDELM
                WRITE(26,1000)(BMATRX(ROW,COL1),COL1=1,BCOLS)
    5   CONTINUE
C WRITE ATRANS
        WRITE(26,1200)
        DO 10 ROW=1,INTNOD
                WRITE(26,1000)(ATRANS(ROW,COL1),COL1=1,BNDELM)
   10   CONTINUE
```

```
C COPY ATRANS
        DO 12 ROW=1,INTNOD
                DO 11 COL1=1,BNDELM
                        TMPAT(ROW,COL1)=ATRANS(ROW,COL1)
   11           CONTINUE
   12   CONTINUE
C WRITE COPY OF ATRANS
        WRITE(26,1900)
        DO 13 ROW=1,INTNOD
                WRITE(26,1000)(TMPAT(ROW,COL1),COL1=1,BNDELM)
   13   CONTINUE
C ADD BMATRX TERM TO ATRANS
        DO 20 ROW=1,BNDELM
                DO 15 COL1=1,BCOLS
                        COL2=RATIO*(ROW-1)+COL1
                        TMPAT(COL2,ROW)=ATRANS(COL2,ROW)+
     1                          EXTG*ALPHA/G*BMATRX(ROW,COL1)
   15           CONTINUE
   20   CONTINUE
C WRITE ATRANS WITH BMATRX TERM ADDED
        WRITE(26,1300)
        DO 25 ROW=1,INTNOD
                WRITE(26,1000)(TMPAT(ROW,COL1),COL1=1,BNDELM)
   25   CONTINUE
C MULTIPLY A * A TRANSPOSE
        DO 40 COL1=1,BNDELM
                DO 35 ROW=1,BNDELM
                        TMPAAT(ROW,COL1)=CMPLX(0.0,0.0)
                        DO 30 COL2=1,INTNOD
                                TMPAAT(ROW,COL1)=TMPAAT(ROW,COL1)+
     1                                  TMPAT(COL2,ROW)*TMPAT(COL2,COL1)
   30                   CONTINUE
   35           CONTINUE
   40   CONTINUE
C WRITE A * A TRANSPOSE
        WRITE(26,1400)
        DO 45 ROW=1,BNDELM
                WRITE(26,1000)(TMPAAT(ROW,COL1),COL1=1,BNDELM)
   45   CONTINUE
C MULTIPLY A * FG
                DO 55 ROW=1,BNDELM
                        TMPFG(ROW)=CMPLX(0.0,0.0)
                        DO 50 COL2=1,INTNOD
                                TMPFG(ROW)=TMPFG(ROW)+
     1                                  TMPAT(COL2,ROW)*FG(COL2)
   50                   CONTINUE
   55           CONTINUE
C WRITE A * FG
        WRITE(26,1500)
        DO 60 ROW=1,BNDELM
        WRITE(26,1000)TMPFG(ROW)
   60   CONTINUE
C SOLVE FOR PHI
        CALL LEQT1C(TMPAAT,BNDELM,15,TMPFG,1,15,0,WA,IER)
        IF(IER.EQ.129) GOTO 110
C WRITE PHI
        WRITE(26,1600)
        DO 65 ROW=1,BNDELM
                WRITE(26,1000) TMPFG(ROW)
   65   CONTINUE
```

```
      WRITE(26,2000)
      RETURN
  100 WRITE(26,1700)
      STOP
  110 WRITE(26,1800)
      STOP
 2000 FORMAT(/,2X,'ATEST SUBROUTINE--------------------------------')
 1900 FORMAT(/,2X,'********* COPIED ATRANS ********')
 1800 FORMAT(/,2X,'A * A TRANSPOSE IS SINGULAR.')
 1700 FORMAT(/,2X,'MATRICES ARE LARGER THAN THE TEMPORARY STORAGE
     1 IN ATEST.',//,2X,'EXECUTION ABORTED')
 1600 FORMAT(/,2X,'********** PHI *********')
 1500 FORMAT(/,2X,'********** A * FG **********')
 1400 FORMAT(/,2X,'********** A*A TRANSPOSE *********')
 1300 FORMAT(/,2X,'********** ATRANS + B TERM *********')
 1200 FORMAT(/,2X,'********** ATRANS ********')
 1100 FORMAT(/,2X,'********** BMATRX ********')
 1000 FORMAT(2X,100F11.6)
      END
C*****************************************************************
C
C*****************************************************************
      SUBROUTINE CMAT(C,CTEMP)
C
C Declare in COMMON all variables required by more than one
C subroutine.  These variables are divided into four catagories:
C       1. SIZE This common block contains variables which define
C               array dimensions.
C       2. PROB This common block contains variables which define
C               problem parameters.
C       3. MAP  This common block contains variables which define
C               the finite element mesh.
C       4. dbug This common block contains variables which define
C               which debug switches are desired.
C Arrays are not passed through COMMON but passed as arguments to the
C subroutines.
C
      COMMON  /SIZE/ NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,
     1        NUMEQN,LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
      INTEGER NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,NUMEQN,
     1        LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
      COMMON  /PROB/ W2,GA,GB,RO,EXTG,ALPHA,THETAO
      REAL    W2,GA,GB,RO,EXTG,ALPHA,THETAO
      COMMON  /MAP/ SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2,RATIO,GRADNT
      INTEGER RATIO
      REAL    SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2
      LOGICAL GRADNT
      common  /dbug/ dmain,dmesh,dassm,dktest,dforce,
     1        dhspac,drect,dtri,dgamma,dfilam,damat,dbmat,
     1        dcmat,dnorm,ddiag,ddiag2,dfunc,dderiv,dctest,datest
      logical  dmain,dmesh,dassm,dktest,dforce,dhspac,
     1        drect,dtri,dgamma,dfilam,damat,dbmat,dcmat,dnorm,
     1        ddiag,ddiag2,dfunc,dderiv,dctest,datest
C
C Declare in COMMON the Gauss integration points.
C
      COMMON  /INGRAL/ NUMPTY,YVALUE,WGTY,NUMPTZ,ZVALUE,WGTZ
      INTEGER NUMPTY,NMPTYD,NUMPTZ
      REAL    YVALUE(100),WGTY(100),ZVALUE(100),WGTZ(100)
C
```

```
C Declare and dimension arrays required in this subroutine.
C
      COMPLEX C(BNDELM,BNDELM),CTEMP(NUMANG)
C
C Declare variables used only in this subroutine.
C
      INTEGER I,ROW,COL,SOURCE,PULL1,PULL2
      REAL    QAOBS,QBOBS,QASRC,QBSRC
C
C Perform double integral of Green's function.
C
C
C    Determine limits of integration for outer integral. This is the
C    integral over the observer.
C
      QAOBS = 0.0
      QBOBS = BDANG1
C
C    Perform integral over first source.
C
      SOURCE = 1
C
C     Read in the Gauss integration points desired.
C
              READ(24,*)NUMPTY
              DO 10 I=1,NUMPTY/2
                      READ(24,*) YVALUE(I),WGTY(I)
   10         CONTINUE
              READ(24,*)NUMPTZ
              DO 20 I=1,NUMPTZ/2
                      READ(24,*) ZVALUE(I),WGTZ(I)
   20         CONTINUE
C
C     Evaluate integral.
C
              CALL DIAG(QAOBS,QBOBS,CTEMP(SOURCE))
C     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C     If debug desired compute exact integral for ln|z-y| and
C     output.
C
      if(dcmat)exact=(qaobs-qbobs)*(qbobs-qaobs)+(qbobs-qaobs)**2
     1 *(alog(qbobs-qaobs)-0.5)+qbobs*(1.0-alog(qbobs-qaobs))*
     1 (qbobs-qaobs-1.0)
      if(dcmat)write(26,600) exact
      if(dcmat)write(26,500) source,ctemp(source)
C     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C    Perform integral over second source.
C
      SOURCE = 2
C
C     Read in the Gauss integration points desired.
C
              READ(24,*)NUMPTY
              DO 40 I=1,NUMPTY/2
                      READ(24,*) YVALUE(I),WGTY(I)
   40         CONTINUE
              READ(24,*)NUMPTZ
              DO 50 I=1,NUMPTZ/2
                      READ(24,*) ZVALUE(I),WGTZ(I)
```

```
   50          CONTINUE
C
C       Determine limits of integration for second source.
C
        QASRC = (SOURCE-1)*BDANGl
        QBSRC = SOURCE*BDANGl
C
C       Evaluate integral
C
        CALL DIAG2(QAOBS,QBOBS,QASRC,QBSRC,CTEMP(SOURCE))
C       ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C       If debug desired compute exact integral for ln|z-y| and
C       output.
C
        if(dcmat)exact=-.5*(qbsrc-qbobs)**2*alog(qbsrc-qbobs)+.75*
     1  (qbsrc-qbobs)**2+.5*qbsrc**2*alog(qbsrc)-.75*qbsrc**2+
     1  -1.0*(.75*
     1  (qasrc-qbobs)**2+.5*qasrc**2*alog(qasrc)-.75*qasrc**2)
        if(dcmat)write(26,600) exact
        if(dcmat)write(26,500) source,ctemp(source)
C       ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C       Read in Gauss integration points for remaining sources.
C
                READ(24,*)NUMPTY
                DO 60 I=1,NUMPTY/2
                        READ(24,*) YVALUE(I),WGTY(I)
   60           CONTINUE
                READ(24,*)NUMPTZ
                DO 70 I=1,NUMPTZ/2
                        READ(24,*) ZVALUE(I),WGTZ(I)
   70           CONTINUE
C
C    Perform integral over remaining sources.
C
        DO 80 SOURCE=3,BNDELM + 1
C
C          Determine limits of integration.
C
                QASRC = (SOURCE-1)*BDANGl
                QBSRC = SOURCE*BDANGl
C
C          Evaluate integral.
C
                CALL DIAG2(QAOBS,QBOBS,QASRC,QBSRC,CTEMP(SOURCE))
C       ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C       If debug desired compute exact integral for ln|z-y| and
C       output.
C
        if(dcmat)exact=-.5*(qbsrc-qbobs)**2*alog(qbsrc-qbobs)+.75*
     1  (qbsrc-qbobs)**2+.5*qbsrc**2*alog(qbsrc)-.75*qbsrc**2+
     1  -1.0*(-.5*(qasrc-qbobs)**2*alog(qasrc-qbobs)+.75*
     1  (qasrc-qbobs)**2+.5*qasrc**2*alog(qasrc)-.75*qasrc**2)
        if(dcmat)write(26,600) exact
        if(dcmat)write(26,500) source,ctemp(source)
C       ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   80   CONTINUE
C       ++++++++++++++++++++++++++++++++++++
        if(dcmat)write(26,400) (i,i=1,bndelm)
C       ++++++++++++++++++++++++++++++++++++
```

```
C
C Position observer one integrals into proper place in C matrix. This is
C done on the basis of a circulant matrix which is then condensed to
C reflect only a half-circle.
C
        DO 100 ROW=1,BNDELM
                DO 90 COL=1,BNDELM
                        IF(COL.LT.ROW) GOTO 85
                        PULL1 = COL-(ROW-1)
                        PULL2 = COL + ROW
                        IF(PULL2.GT.BNDELM+1)PULL2=2*(BNDELM+1)-PULL2
                        C(ROW,COL) = 2.0*(CTEMP(PULL1) + CTEMP(PULL2))
                        GOTO 90
   85                   C(ROW,COL) = C(COL,ROW)
   90           CONTINUE
C               ++++++++++++++++++++++++++++++++++++++++++++++++++
                if(dcmat)write(26,300) row,(c(row,col),col=1,bndelm)
C               ++++++++++++++++++++++++++++++++++++++++++++++++++
  100   CONTINUE
C
C Debug format statements.
C
  300   format(2x,i2,1x,100(1X,2f6.3))
  400   format(//2x,'********* full c matrix **********',//
     1  ,11x,100(i2,11x))
  500   format(2x,'ctemp(',i3,') = ',2f11.6,/)
  600   format(2x,'exact ln|z-y| integral = ',f11.6)
C
C Return to HSPACE subroutine.
C
        RETURN
        END


C*********************************************************************
        SUBROUTINE DIAG(C,D,INTGYZ)
C
C Declare in COMMON all variables required by more than one
C subroutine.  These variables are divided into four catagories:
C       1. SIZE This common block contains variables which define
C               array dimensions.
C       2. PROB This common block contains variables which define
C               problem parameters.
C       3. MAP  This common block contains variables which define
C               the finite element mesh.
C       4. dbug This common block contains variables which define
C               which debug switches are desired.
C Arrays are not passed through COMMON but passed as arguments to the
C subroutines.
C
        COMMON  /SIZE/ NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,
     1          NUMEQN,LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
        INTEGER NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,NUMEQN,
     1          LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
        COMMON  /PROB/ W2,GA,GB,RO,EXTG,ALPHA,THETAO
        REAL    W2,GA,GB,RO,EXTG,ALPHA,THETAO
        COMMON  /MAP/ SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2,RATIO,GRADNT
        INTEGER RATIO
        REAL    SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2
        LOGICAL GRADNT
```

```
      common  /dbug/ dmain,dmesh,dassm,dktest,dforce,
     1        dhspac,drect,dtri,dgamma,dfilam,damat,dbmat,
     1        dcmat,dnorm,ddiag,ddiag2,dfunc,dderiv,dctest,datest
      logical  dmain,dmesh,dassm,dktest,dforce,dhspac,
     1        drect,dtri,dgamma,dfilam,damat,dbmat,dcmat,dnorm,
     1        ddiag,ddiag2,dfunc,dderiv,dctest,datest
C
C Declare in COMMON the Gauss integration points.
C
      COMMON  /INGRAL/ NUMPTY,YVALUE,WGTY,NUMPTZ,ZVALUE,WGTZ
      INTEGER NUMPTY,NMPTYD,NUMPTZ
      REAL    YVALUE(100),WGTY(100),ZVALUE(100),WGTZ(100)
C
C Declare variable used only int this subroutine.
C
      INTEGER I,ZI,YI
      REAL    A,B,C,D,Z,Y,EPSLON,DIST,CONST
      COMPLEX INTGYZ,INTGY,ANSY,FUNC
      LOGICAL DONE
C     ++++++++++++++++++++++++++++++++
C If debug desired output number of Gauss integration points.
C
      if(ddiag)write(26,100)numpty,numptz
C     ++++++++++++++++++++++++++++++++
C
C Define required constant
C
      CONST = -0.5
C
C Compute double Green's function integral for C matrix.
C
C
C    Initialize integral
C
      INTGYZ = CMPLX(0.0,0.0)
C
C    Integrate with respect to dz over each element.
C
C
C     Iterate through each Gauss integration point.
C
      DO 50 ZI=1,NUMPTZ/2
C
C          Transform -1 to 1 positive integration points to c to d points
C
              Z = (ZVALUE(ZI)*(D-C)+D+C)/2.0
              DONE = .FALSE.
              GOTO 30
C
C            Transform -1 to 1 negative integration points to c to d
C            points. This is done after the positive point has been
C            integrated with respect to dy.
C
   25         Z= (-ZVALUE(ZI)*(D-C)+D+C)/2.0
              DONE = .TRUE.
C
C            For each value from the dz integral integrate with respect to dy.
C
C
C             Initialize dy integral to zero.
```

```
C
   30          INTGY = CMPLX(0.0,0.0)
C
C              Iterate through each Gauss integration point.
C
               DO 40 YI=1,NUMPTY/2
C
C               The dy integral is divide into two integrals to handle
C               the logrithmic singularity. The first is from the lower
C               limit to the z value.
C

                     A = C
                     B=Z
C
C               Transform -1 to 1 positive integration points to c to
C               z points.
C
                     Y = (YVALUE(YI)*(B-A)+B+A)/2.0
C
C               Evaluate Green's function at integration points.
C               Multiply by scaling factor and sum for total dy
C               integral.
C
                     DIST = 2.0*RANGE*ABS(SIN((Z-Y)/2.0))
                     ANSY = FUNC(DIST)*(B-A)/2.0
C
C               Transform -1 to 1 negative integration points to c to z
C               points.
C
                     Y = (-YVALUE(YI)*(B-A)+B+A)/2.0
                     DIST = 2.0*RANGE*ABS(SIN((Z-Y)/2.0))
C
C               Evaluate Green's function at integration points.
C               Multiply by scaling factor and sum for total dy
C               integral.
C
                     ANSY = ANSY + FUNC(DIST)*(B-A)/2.0
C
C               The second portion of the dy integral is from z to d.
C
                     A=Z
                     B = D
C
C               Transform -1 to 1 positive integration points to c to
C               z points.
C
                     Y = (YVALUE(YI)*(B-A)+B+A)/2.0
C
C               Evaluate Green's function at integration points.
C               Multiply by scaling factor and sum for total dy
C               integral.
C
                     DIST = 2.0*RANGE*ABS(SIN((Z-Y)/2.0))
                     ANSY = ANSY + FUNC(DIST)*(B-A)/2.0
C
C               Transform -1 to 1 negative integration points to c to z
C               points.
C
                     Y = (-YVALUE(YI)*(B-A)+B+A)/2.0
```

```
C
C                Evaluate Green's function at integration points.
C                Multiply by scaling factor and sum for total dy
C                integral.
C
                       DIST = 2.0*RANGE*ABS(SIN((Z-Y)/2.0))
                       ANSY = ANSY + FUNC(DIST)*(B-A)/2.0
C
C                Multiply dy integral by the integration weights.
C
                       INTGY = INTGY + WGTY(YI)*ANSY
   40          CONTINUE
C
C             Multiply dy evaluation by dz weights and scaling factors
C             and the required constant and sum to total dz integral.
C
               INTGYZ = INTGYZ + CONST*WGTZ(ZI)*(D-C)/2.0*INTGY
               IF(DONE) GOTO 50
               GOTO 25
   50   CONTINUE
C       ++++++++++++++++++++++++++++
C If debug desired output value of integral.
C
        if(ddiag)write(25,200) intgyz
C       ++++++++++++++++++++++++++++
C
C Debug format statements.
C
  100   format(/2x,'number of points for DY integral =',i5,/,
     1  2x,'number of points for DZ integral =',i5)
  200   format(2x,'integral = ',2f11.6)
C
C Return to CMAT subroutine.
C
        RETURN
        END
C*************************************************************************
        SUBROUTINE DIAG2(C,D,E,F,INTGYZ)
C
C Declare in COMMON all variables required by more than one
C subroutine.  These variables are divided into four catagories:
C       1. SIZE This common block contains variables which define
C               array dimensions.
C       2. PROB This common block contains variables which define
C               problem parameters.
C       3. MAP  This common block contains variables which define
C               the finite element mesh.
C       4. dbug This common block contains variables which define
C               which debug switches are desired.
C Arrays are not passed through COMMON but passed as arguments to the
C subroutines.
C
        COMMON  /SIZE/ NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,
     1          NUMEQN,LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
        INTEGER NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,NUMEQN,
     1          LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
        COMMON  /PROB/ W2,GA,GB,RO,EXTG,ALPHA,THETAO
        REAL    W2,GA,GB,RO,EXTG,ALPHA,THETAO
        COMMON  /MAP/ SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2,RATIO,GRADNT
        INTEGER RATIO
```

```
      REAL    SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2
      LOGICAL GRADNT
      common  /dbug/ dmain,dmesh,dassm,dktest,dforce,
     1        dhspac,drect,dtri,dgamma,dfilam,damat,dbmat,
     1        dcmat,dnorm,ddiag,ddiag2,dfunc,dderiv,dctest,datest
      logical  dmain,dmesh,dassm,dktest,dforce,dhspac,
     1        drect,dtri,dgamma,dfilam,damat,dbmat,dcmat,dnorm,
     1        ddiag,ddiag2,dfunc,dderiv,dctest,datest
C
C Declare in COMMON the Gauss integration points.
C
      COMMON  /INGRAL/ NUMPTY,YVALUE,WGTY,NUMPTZ,ZVALUE,WGTZ
      INTEGER NUMPTY,NMPTYD,NUMPTZ
      REAL    YVALUE(100),WGTY(100),ZVALUE(100),WGTZ(100)
C
C Declare variables used only in this subroutine.
C
      INTEGER I,ZI,YI
      REAL    A,B,C,D,Z,Y,EPSLON,E,F,DIST,CONST
      COMPLEX INTGYZ,INTGY,ANSY,FUNC
      LOGICAL DONE
C     ++++++++++++++++++++++++++++++++++++
C If debug desired output number of Gauss integration points.
C
      if(ddiag2)write(26,100)numpty,numptz
C     ++++++++++++++++++++++++++++++++++++
C
C Define required constant
C
      CONST = -0.5
C
C Compute double Green's function integral for C matrix.
C
C
C    Initialize integral
C
      INTGYZ = CMPLX(0.0,0.0)
C
C    Integrate with respect to dz over each element.
C
C
C     Iterate through each Gauss integration point.
C
      DO 50 ZI=1,NUMPTZ/2
C
C           Transform -1 to 1 positive integration points to c to d points
C
              Z = (ZVALUE(ZI)*(D-C)+D+C)/2.0
              DONE = .FALSE.
              GOTO 30
C
C            Transform -1 to 1 negative integration points to c to d
C            points. This is done after the positive point has been
C            integrated with respect to dy.
C
   25         Z= (-ZVALUE(ZI)*(D-C)+D+C)/2.0
              DONE = .TRUE.
C
C            For each value from the dz integral integrate with respect to dy.
C
```

```
C
C               Initialize dy integral to zero.
C
   30           INTGY = CMPLX(0.0,0.0)
C
C               Iterate through each Gauss integration point.
C
                DO 40 YI=1,NUMPTY/2
C
C               Define limits of integration.
C
                       A = E
                       B = F
C
C                Transform -1 to 1 positive integration points to c to
C                z points.
C
                       Y = (YVALUE(YI)*(B-A)+B+A)/2.0
C
C                Evaluate Green's function at integration points.
C                Multiply by scaling factor and sum for total dy
C                integral.
C
                       DIST = 2.0*RANGE*ABS(SIN((Z-Y)/2.0))
                       ANSY = FUNC(DIST)*(B-A)/2.0
C
C                Transform -1 to 1 negative integration points to c to z
C                points.
C
                       Y = (-YVALUE(YI)*(B-A)+B+A)/2.0
C
C                Evaluate Green's function at integration points.
C                Multiply by scaling factor and sum for total dy
C                integral.
C
                       DIST = 2.0*RANGE*ABS(SIN((Z-Y)/2.0))
                       ANSY = ANSY + FUNC(DIST)*(B-A)/2.0
C
C                Multiply dy integral by the integration weights.
C
                       INTGY = INTGY + WGTY(YI)*ANSY
   40           CONTINUE
C
C              Multiply dy evaluation by dz weights and scaling factors
C              and the required constant and sum to total dz integral.
C
                INTGYZ = INTGYZ + CONST*WGTZ(ZI)*(D-C)/2.0*INTGY
                IF(DONE) GOTO 50
                GOTO 25
   50    CONTINUE
C        ++++++++++++++++++++++++++++
C If debug desired output value of integral.
C
         if(ddiag2)write(26,200) intgyz
C        ++++++++++++++++++++++++++++
C
C Debug format statements.
C
  100    format(/2x,'number of points for DY integral =',i5,/,
     1   2x,'number of points for DZ integral =',i5)
```

```
  200   format(2x,'integral = ',2f11.6)
C
C Return to CMAT subroutine.
C
        RETURN
        END
C*********************************************************************
        COMPLEX FUNCTION FUNC(DIST)
C
C Declare in COMMON all variables required by more than one
C subroutine.  These variables are divided into four catagories:
C       1. SIZE This common block contains variables which define
C               array dimensions.
C       2. PROB This common block contains variables which define
C               problem parameters.
C       3. MAP  This common block contains variables which define
C               the finite element mesh.
C       4. dbug This common block contains variables which define
C               which debug switches are desired.
C Arrays are not passed through COMMON but passed as arguments to the
C subroutines.
C
        COMMON  /PROB/ W2,GA,GB,RO,EXTG,ALPHA,THETAO
        REAL    W2,GA,GB,RO,EXTG,ALPHA,THETAO
        COMMON  /MAP/ SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2,RATIO,GRADNT
        INTEGER RATIO
        REAL    SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2
        LOGICAL GRADNT
        common  /dbug/ dmain,dmesh,dassm,dktest,dforce,
     1          dhspac,drect,dtri,dgamma,dfilam,damat,dbmat,
     1          dcmat,dnorm,ddiag,ddiag2,dfunc,dderiv,dctest,datest
        logical  dmain,dmesh,dassm,dktest,dforce,dhspac,
     1          drect,dtri,dgamma,dfilam,damat,dbmat,dcmat,dnorm,
     1          ddiag,ddiag2,dfunc,dderiv,dctest,datest
C
C Declare variables used in this function only.
C
        REAL    KAPPA,DIST,X,ARG,FO,THETA,YO,JO
C
C Define necessary constant.
C
        KAPPA = SQRT(W2*RANGE/EXTG)
C
C Compute Green's function argument.
C
        X = KAPPA*DIST
C
C Test argument to determine proper formula for computation.
C
        IF(X.LT.3.0)GOTO 10
C
C Compute Green's function
C
        ARG = 3.0/X
        FO=(((((.00014476*ARG-.00072805)*ARG+.0013723)*ARG
     1  -.00009512)*ARG-.00552740)*ARG-.00000077)*ARG+.79788456
        THETA=(((((.00013558*ARG-.00029333)*ARG-.00054125)*ARG
     1  +.00262573)*ARG-.00003954)*ARG-.04166397)*ARG
     1  -.78539816+X
        JO = X**-0.5*FO*COS(THETA)/(4.0)
```

```
      Y0 = X**-0.5*F0*SIN(THETA)/(-4.0)
C     ++++++++++++++++++++++++++++++++++++++
C If debug desired output computed values.
C
      if(dfunc)write(26,500)kappa,dist,x,j0,y0
C     ++++++++++++++++++++++++++++++++++++++
      FUNC = CMPLX(Y0,J0)
C
C Return to DIAG or DIAG2 subroutines.
C
      RETURN
C
C Compute Green's function
C
   10 ARG = (X/3.0)**2
      F0=(((((.0002100*ARG-.0039444)*ARG+.0444479)*ARG
     1 -.3163866)*ARG+1.2656208)*ARG-2.2499997)*ARG+1.0
      THETA=(((((-.00024846*ARG+.00427916)*ARG-.04261214)*ARG
     1 +.25300117)*ARG-.74350384)*ARG+.60559366)*ARG
     1 +.36746691+2.0/3.14159265*ALOG(0.5*X)*F0
      J0 = F0/(4.0)
      Y0 = THETA/(-4.0)
C     ++++++++++++++++++++++++++++++++++++++
C If debug desired output computed values.
C
      if(dfunc)write(26,500)kappa,dist,x,j0,y0
C     ++++++++++++++++++++++++++++++++++++++
      FUNC = CMPLX(Y0,J0)
C
C Return to DIAG or DIAG2 subroutines.
C
      RETURN
C
C Debug format statements.
C
  500 format(/,2x,'kappa = ',f11.6,/,
     1 2x,'distance = ',f11.6,/,
     1 2x,'x = ',f11.6,/,
     1 2x,'j0 = ',f11.6,/,
     1 2x,'y0 = ',f11.6,/,
     1 2x,'func = cmplx(y0,j0)')
      END
C*********************************************************************
C
C*********************************************************************
      SUBROUTINE CTEST(FPHI,C)
C
C Declare in COMMON all variables required by more than one
C subroutine.  These variables are divided into four catagories:
C       1. SIZE This common block contains variables which define
C               array dimensions.
C       2. PROB This common block contains variables which define
C               problem parameters.
C       3. MAP  This common block contains variables which define
C               the finite element mesh.
C       4. dbug This common block contains variables which define
C               which debug switches are desired.
C Arrays are not passed through COMMON but passed as arguments to the
C subroutines.
C
```

```
                COMMON  /SIZE/ NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,
     1          NUMEQN,LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
        INTEGER NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,NUMEQN,
     1          LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
        COMMON  /PROB/ W2,GA,GB,RO,EXTG,ALPHA,THETAO
        REAL    W2,GA,GB,RO,EXTG,ALPHA,THETAO
        COMMON  /MAP/ SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2,RATIO,GRADNT
        INTEGER RATIO
        REAL    SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2
        LOGICAL GRADNT
        common  /dbug/ dmain,dmesh,dassm,dktest,dforce,
     1          dhspac,drect,dtri,dgamma,dfilam,damat,dbmat,
     1          dcmat,dnorm,ddiag,ddiag2,dfunc,dderiv,dctest,datest
        logical  dmain,dmesh,dassm,dktest,dforce,dhspac,
     1          drect,dtri,dgamma,dfilam,damat,dbmat,dcmat,dnorm,
     1          ddiag,ddiag2,dfunc,dderiv,dctest,datest
C
C Declare and dimension arrays used in this subroutine.
C
        REAL    WA(15)
        COMPLEX C(BNDELM,BNDELM),SUM,W,FPHI(BNDELM),
     1  FUNC,TEMPC(15,15),TMPFHI(15),ANS(15)
C
C Declare variables used only in this subroutine.
C
        INTEGER I,IER
        WRITE(26,600)
C
C Copy C matrix and force vector so test can be performed without
C damage. Also initialize result.
C
        DO 4 I=1,BNDELM
                DO 5 J=1,BNDELM
                        TEMPC(I,J)=C(I,J)
    5           CONTINUE
                TMPFHI(I)=FPHI(I)
                ANS(I)=CMPLX(0.0,0.0)
    4   CONTINUE
C
C Solve equation C*PHI= FPHI
C
        CALL LEQT1C(TEMPC,BNDELM,15,TMPFHI,1,15,0,WA,IER)
        IF(IER.EQ.129) GOTO 90
        WRITE(26,150)
C
C Sum phi's. Displacement at origin equals G(0,y)*delta theta*sum of phji's.
C
        SUM = CMPLX(0.0,0.0)
        DO 10 I=1,BNDELM
                WRITE(26,200) I,TMPFHI(I)
                SUM = SUM + TMPFHI(I)
   10   CONTINUE
        WRITE(26,550)
C
C Multiply PHI*C to see if C matrix is well behaved.
C
        DO 20 I=1,BNDELM
                DO 30 J=1,BNDELM
                        ANS(I)=ANS(I)+C(I,J)*TMPFHI(J)
   30           CONTINUE
```

```
              WRITE(26,500) I,ANS(I),FPHI(I)
   20   CONTINUE
C
C Determine G(0,y).
C
        W = FUNC(RANGE)
        WRITE(26,650) W
C
C Determine displacement at origin.
C
        W = 2.0 * BDANG1 * SUM * W
        WRITE(26,250) W
        WRITE(26,600)
C
C Return to HSPACE subroutine.
C
        RETURN
   90   WRITE(5,100)
        STOP
C
C Debug format statements.
C
  100   FORMAT(/,2X,'C MATRIX IS SINGULAR. EXECUTION ABORTED.',/)
  125   FORMAT(/2X,'MATRICES ARE LARGER THAN THE TEMPORARY STORAGE IN
     1  CTEST.',//,2X,'EXECUTION ABORTED.')
  150   FORMAT(/,2X,'ELEMENT',11X,'PHI',/)
  200   FORMAT(2X,I5,5X,2F11.6)
  250   FORMAT(/,2X,'DISPLACEMENT AT ORIGIN = ',2F11.6)
  550   FORMAT(/,2X,'ELEMENT',11X,'ANS',22X,'FPHI',/)
  500   FORMAT(2X,I5,5X,2F11.6,5X,2F11.6)
  600   FORMAT(2X,'SUBROUTINE CTEST ------------------------------',/)
  650   FORMAT(/2X,'G(R) = ',2F11.6)
        END
C*********************************************************************
C
C*********************************************************************

        SUBROUTINE SOLVE(K,D,FG,FD,KOG,KOO,B,WHOLE,SLTN,W,XL,WA,
     1          LAM,TEMPD,CINVER,FPHI)
C
C Declare in COMMON all variables required by more than one
C subroutine.  These variables are divided into four catagories:
C       1. SIZE This common block contains variables which define
C               array dimensions.
C       2. PROB This common block contains variables which define
C               problem parameters.
C       3. MAP  This common block contains variables which define
C               the finite element mesh.
C       4. dbug This common block contains variables which define
C               which debug switches are desired.
C Arrays are not passed through COMMON but passed as arguments to the
C subroutines.
C
        COMMON  /SIZE/ NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,
     1          NUMEQN,LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
        INTEGER NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,NUMEQN,
     1          LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
        COMMON  /PROB/ W2,GA,GB,RO,EXTG,ALPHA,THETAO
        REAL    W2,GA,GB,RO,EXTG,ALPHA,THETAO
        COMMON  /MAP/ SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2,RATIO,GRADNT
```


C-2

```
      INTEGER RATIO
      REAL    SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2
      LOGICAL GRADNT
      common  /dbug/ dmain,dmesh,dassm,dktest,dforce,
     1        dhspac,drect,dtri,dgamma,dfilam,damat,dbmat,
     1        dcmat,dnorm,ddiag,ddiag2,dfunc,dderiv,dctest,datest
      logical dmain,dmesh,dassm,dktest,dforce,dhspac,
     1        drect,dtri,dgamma,dfilam,damat,dbmat,dcmat,dnorm,
     1        ddiag,ddiag2,dfunc,dderiv,dctest,datest
C
C Dimension and declare arrays used within this subroutine.
C
      REAL    K(NUMNOD,WIDTH),KOG(NUMEQN,INTNOD),KOO(NUMEQN,WIDTH),
     1        B(NUMEQN,INTNOD),XL(NUMEQN,SPACE),WA(INTNOD)
      COMPLEX D(INTNOD,INTNOD),FG(INTNOD),FD(INTNOD),SLTN(INTNOD),
     1        WHOLE(INTNOD,INTNOD),W(INTNOD),LAM(BNDELM),
     1        TEMPD(INTNOD,INTNOD),CINVER(BNDELM,BNDELM),FPHI(BNDELM)
C
C Declare variables used only by this subroutine.
C
      INTEGER ROW,COL,COL1,COL2,PULL,IER,N,L,M,I,J,P
      REAL    KAPPA,KAPPAR,THETA

C$    WRITE(26,700)
C
C Partition K(omega,gamma) from structure stiffness matrix. KOG is
C pulled term by term from K. KOG is stored in full storage mode.
C
      N=0
      DO 20 I=INTNOD+1,NUMNOD
              N=N+1
              DO 10 J=1,INTNOD
                      KOG(N,J)=0.0
                      B(N,J)=0.0
                      M=I-(1+CODIAG)
                      PULL=J-M
                      IF(PULL.LE.0) GOTO 10
                      IF(PULL.GT.WIDTH) GOTO 20
                      KOG(N,J)=K(I,PULL)
                      B(N,J)=K(I,PULL)*-1.0
   10         CONTINUE
   20 CONTINUE
C$    write(26,1300)
C$    DO 53 ROW=1,NUMEQN
C$            WRITE(26,1200)(KOG(ROW,COL),COL=1,INTNOD)
C$   53 CONTINUE
C$    write(26,1400)
C$    DO 63 ROW=1,NUMEQN
C$            WRITE(26,1200)(B(ROW,COL),COL=1,INTNOD)
C$   63 CONTINUE
C
C Partition K(omega,omega) from structure stiffness matrix. KOO is
C pulled term by term from K. KOO is stored in band storage mode rather
C than symmetric storage mode to allow the use of an IMSL solving
C routine.
C
      DO 40 I=INTNOD+1,NUMNOD
              N=I-(INTNOD+1+CODIAG)
              P=N
              IF(N.LT.0)N=0
```

```
                M=I+CODIAG
                IF(M.GT.NUMNOD)M=NUMNOD
                DO 30 J=INTNOD+1+N,M
                        L=I-(1+CODIAG)
                        PULL=J-L
                        ROW=I-INTNOD
                        COL=J-INTNOD-P
                        KOO(ROW,COL)=K(I,PULL)
   30           CONTINUE
   40   CONTINUE
C$      write(26,1500)
C$      do 74 i=1,numeqn
C$              write(26,1200)(koo(i,j),j=1,WIDTH)
C$   74 continue
C
C Solve equation KOO x X = B = -KOG. X, the solution is written over
C top of B. This solving is done with IMSL routine LEQT1B.
C
        CALL LEQT1B(KOO,NUMEQN,CODIAG,CODIAG,NUMEQN,B,INTNOD,NUMEQN,0,
     1  XL,IER)
        IF(IER.EQ.129)write(5,400)
  400   format(2x,'error')
C$      write(26,1600)
C$      do 73 i=1,numeqn
C$              write(26,1200)(b(i,j),j=1,intnod)
C$   73 continue
C
C Multiply k(gamma,omega) times -K(omega,omega) inverse times
C K(omega,gamma).
C
        DO 50 COL1=1,INTNOD
                DO 60 ROW=1,INTNOD
                        KOO(ROW,COL1)=0.0
                        DO 70 COL2=1,NUMEQN
                                KOO(ROW,COL1) = KOO(ROW,COL1) +
     1                                  KOG(COL2,ROW)*B(COL2,COL1)
   70                   CONTINUE
   60           CONTINUE
   50   CONTINUE
C$      write(26,1100)
C$      do 51 row=1,intnod
C$              write(26,1200)(koo(row,col),col=1,intnod)
C$   51 continue
C
C Assemble WHOLE matrix.
C
        DO 80 ROW=1,INTNOD
                DO 90 COL=1,INTNOD
C
C    Load K(gamma,gamma)
C
                        L=ROW-(1+CODIAG)
                        PULL=COL-L
                        IF(PULL.LT.0.OR.PULL.GT.WIDTH) GOTO 91
C$                      PULL = COL+ROW*(ROW-1)/2
C$                      IF(COL.GT.ROW)PULL = ROW+COL*(COL-1)/2
                        WHOLE(ROW,COL)=K(ROW,PULL)
                        GOTO 93
   91                   WHOLE(ROW,COL)=CMPLX(0.0,0.0)
C
```

```
  700   FORMAT(2X,'SUBROUTINE SOLVE ----------------------------------')
  800   format(2x,'row = ',I3,2x,'theta = ',f5.3,2x,'kappa =',f5.3)
  900   format(2x,'w(',i3,') =',2f15.6)
 1000   format(1x,'fg =',2f10.6,' fd =',2f10.6,' sltn =',2f10.6)
 1100   format(/,2x,'*********
     1  -K(GAMMA,OMEGA)*K(OMEGA,OMEGA)-1*K(OMEGA,GAMMA) **********',/)
 1200   format(1x,100f7.4)
 1300   format(/,2x,'kog')
 1400   format(/,2x,'b')
 1500   format(/,2x,'koo')
1600    format(/,2x,'solution matrix')
C
C Return to MAIN program.
C
        RETURN
        END
C*********************************************************************
C
C*********************************************************************
      SUBROUTINE OUT(SLTN,W,FG,FD,B,LAM)
C
C Declare in COMMON all variables required by more than one
C subroutine.  These variables are divided into four catagories:
C       1. SIZE This common block contains variables which define
C               array dimensions.
C       2. PROB This common block contains variables which define
C               problem parameters.
C       3. MAP  This common block contains variables which define
C               the finite element mesh.
C       4. dbug This common block contains variables which define
C               which debug switches are desired.
C Arrays are not passed through COMMON but passed as arguments to the
C subroutines.
C
        COMMON  /SIZE/ NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,
     1          NUMEQN,LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
        INTEGER NUMANG,NUMRAD,NUMELM,INTNOD,NUMNOD,CODIAG,NUMEQN,
     1          LENGTH,WIDTH,SPACE,BNDELM,BCOLS,ALENG,AWIDE,ATMPL
        COMMON  /PROB/ W2,GA,GB,RO,EXTG,ALPHA,THETAO
        REAL    W2,GA,GB,RO,EXTG,ALPHA,THETAO
        COMMON  /MAP/ SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2,RATIO,GRADNT
        INTEGER RATIO
        REAL    SWEEP,RANGE,DRAD,DANG,BDANG1,BDANG2
        LOGICAL GRADNT
        common  /dbug/ dmain,dmesh,dassm,dktest,dforce,
     1          dhspac,drect,dtri,dgamma,dfilam,damat,dbmat,
     1          dcmat,dnorm,ddiag,ddiag2,dfunc,dderiv,dctest,datest
        logical  dmain,dmesh,dassm,dktest,dforce,dhspac,
     1          drect,dtri,dgamma,dfilam,damat,dbmat,dcmat,dnorm,
     1          ddiag,ddiag2,dfunc,dderiv,dctest,datest
C
C Dimension and declare arrays used within this subroutine.
C
        REAL    B(NUMEQN,INTNOD)
        COMPLEX FG(INTNOD),FD(INTNOD),SLTN(INTNOD),W(INTNOD),LAM(BNDELM)
C
C Declare variables used only by this subroutine.
C
        INTEGER ROW,COL,COL1,COL2,PULL,IER,N,L,M,I,J,P
        REAL    RAD,RE,IM,AMP,PHASE,AMPC,PHASEC,PHASC
```

```
      COMPLEX X,DUMMY
C
C Output boundary node displacements.
C
      WRITE(26,600)
      DO 120 ROW=1,INTNOD
              WRITE(26,500) ROW,SLTN(ROW),W(ROW)
  120 CONTINUE
C
C Back substitute to determine surface node displacements.
C
      DO 130 I=2,NUMRAD
              ROW=INTNOD*(I-1)
              PUSH = NUMRAD+1-I
              FG(PUSH)=CMPLX(0.0,0.0)
              DO 140 COL=1,INTNOD
                      FG(PUSH)=FG(PUSH)+B(ROW,COL)*SLTN(COL)
  140         CONTINUE
              ROW=ROW-(INTNOD-1)
              PUSH=I-1
              FD(PUSH)=CMPLX(0.0,0.0)
              DO 150 COL=1,INTNOD
                      FD(PUSH)=FD(PUSH)+B(ROW,COL)*SLTN(COL)
  150         CONTINUE
  130 CONTINUE
      ROW=NUMEQN
      PUSH=NUMRAD
      FD(PUSH)=CMPLX(0.0,0.0)
      DO 160 COL=1,INTNOD
              FD(PUSH)=FD(PUSH)+B(ROW,COL)*SLTN(COL)
  160 CONTINUE
C
C Seperate real and imaginary parts of displacement and calculate
c amplitude and phase of origin.
C
      X = CMPLX(0.0,-1.0)
      RE = FD(NUMRAD)
      DUMMY = FD(NUMRAD) - RE
      DUMMY = X*DUMMY
      IM = DUMMY
      AMPC = SQRT(RE**2+IM**2)
      PHASEC = ATAN(IM/RE)
C
C Determine and output displacement, amplitude, phase with respect to
C incoming wave, and phase with respect to origin for the surface nodes.
C
      WRITE(26,800)
      RAD=RANGE
      RE = SLTN(1)
      DUMMY = SLTN(1) - RE
      DUMMY = X*DUMMY
      IM = DUMMY
      AMP = SQRT(RE**2+IM**2)
      PHASE = ATAN(IM/RE)
      PHASC = PHASE - PHASEC
      WRITE(26,700)RAD,SLTN(1),AMP,PHASE,PHASC
      DO 170 J=1,NUMRAD
              RAD=RANGE-DRAD*FLOAT(J)
              RE = FD(J)
              DUMMY = FD(J) - RE
```

```
                DUMMY = X*DUMMY
                IM = DUMMY
                AMP = SQRT(RE**2+IM**2)
                PHASE = ATAN(IM/RE)
                PHASC = PHASE - PHASEC
                WRITE(26,700)RAD,FD(J),AMP,PHASE,PHASC
  170   CONTINUE
        DO 180 J=1,NUMRAD-1
                RAD = -DRAD*FLOAT(J)
                RE = FG(J)
                DUMMY = FG(J) - RE
                DUMMY = X*DUMMY
                IM = DUMMY
                AMP = SQRT(RE**2+IM**2)
                PHASE = ATAN(IM/RE)
                PHASC = PHASE - PHASEC
                WRITE(26,700)RAD,FG(J),AMP,PHASE,PHASC
  180   CONTINUE
        RAD= -RANGE
        RE = SLTN(INTNOD)
        DUMMY = SLTN(INTNOD) - RE
        DUMMY = X*DUMMY
        IM = DUMMY
        AMP = SQRT(RE**2+IM**2)
        PHASE = ATAN(IM/RE)
        PHASC = PHASE - PHASEC
        WRITE(26,700)RAD,SLTN(INTNOD),AMP,PHASEC,PHASC
C
C Output tractions at boundary elements.
C
        WRITE(26,900)
        DO 200 I=1,BNDELM
                WRITE(26,950)I,LAM(I)
  200   CONTINUE
C
C Output format statements.
C
  500   FORMAT(2X,I5,2X,E15.6,2X,E15.6,2X,E15.6,2X,E15.6)
  600   FORMAT(/,2X,'************ BOUNDARY NODE DISPLACEMENTS
     1  **********',//,
     1  2X,18X,'SOLVED',26X,'FREE FIELD',/,
     1  2X,' NODE',7X,'REAL',11X,'IMAGINARY',10X,'REAL',11X,'IMAGINARY')
  700   FORMAT(4X,F5.2,2X,E15.6,2X,E15.6,2X,E15.6,2X,E15.6,2X,E15.6)
  800   FORMAT(//,2X,'************ SURFACE NODE DISPLACEMENTS
     1  **********',//,
     1  3X,'RADIUS',7X,'REAL',13X,'IMAGINARY',8X,'AMPLITUDE',8X,'PHASE',
     1  12X,'PHASE',/,67X,'FREE FIELD',7X,'CENTER')
  950   FORMAT(4X,I5,2X,E15.6,2X,E15.6)
  900   FORMAT(//,2X,'************ BOUNDARY ELEMENT TRACTIONS
     1  **********',//,
     1  2X,'ELEMENT',7X,'REAL',13X,'IMAGINARY')
        RETURN
        END
```

APPROVAL

A NUMERICAL METHOD FOR INTERFACE PROBLEMS IN ELASTODYNAMICS

By David McGhee

The information in this report has been reviewed for technical content. Review of any information concerning Department of Defense or nuclear energy activities or programs has been made by the MSFC Security Classification Officer. This report, in its entirety, has been determined to be unclassified.

G. F. McDONOUGH
Director, Systems Dynamics Laboratory

☆U.S. GOVERNMENT PRINTING OFFICE 1985—544-048/10088