SANDIA REPORT

SAND97-2191 • UC-706

Unlimited Release

Printed September 1997

Inverse Scattering Code

RECEIVED

OCT 10 1997

OSTI

Arthur Hale, Adrian King

MASTER

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

Approved for public release; distribution is unlimited.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Prices available from (615) 576-8401, FTS 626-8401

Available to the public from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

NTIS price codes
Printed copy: A03
Microfiche copy: A01

SAND 97-2191
Unlimited Release
Printed September 1997

Distribution
Category UC-706

Inverse Scattering Code

Arthur Hale
Computer Sciences Department
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-1109

Adrian King
Intel Corporation
15201 NW Greenbrier Parkway
Beaverton, OR 97006

Abstract

A methodology for the evaluation of complex electromagnetics problems is presented. The methodology reduces the computational requirements for the analysis of large scale computational electromagnetics problems by hybridizing the method of moments and physical optics techniques. The target model is based on triangular facets and the incident field source by its system response function. Data which can be obtained from the analysis are radar cross section, power spectral density, and range profiles.

COMPLEX TARGET SCATTERING CODE

ABSTRACT

A methodology for the evaluation of complex electromagnetics problems is presented. The methodology reduces the computational requirements for the analysis of large scale computational electromagnetics problems by hybridizing the method of moments and physical optics techniques. The target model is based on triangular facets and the incident field source by its system response function. Data which can be obtained from the analysis are radar cross section, power spectral density, and range profiles.

INTRODUCTION

The modeling of radar sensors and targets is of enormous significance because of their importance in all types and phases of military scenarios. The ability to predict the return power spectral density and radar cross section of a given target is a necessary part of stealth vehicle and sensor design. Furthermore, designers of modern systems where features such as polarization and statistical characteristics of the fields scattered from targets are exploited for target identification/discrimination and/or performance enhancement require a detailed understanding of sensor-target interaction. Although electromagnetic performance characterization is critical, practical and accurate sensor modeling against targets such as ships, tanks, and aircraft is still in its infancy.

Many methods are available to solve Maxwell's Equations formulations for large scale targets. These methods include the method of moments (MOM), finite differences, finite elements, and asymptotic methods such as the Uniform Theory of Diffraction and physical optics (PO). Each of these methods have their own advantages and disadvantages, but it is well known that no single analysis technique can accurately and efficiently predict the electromagnetic performance parameters of electrically large and complex structures. For instance, consider a target comprised of electrically large structures with small-to-medium sized electrical perturbations such as the helicopter shown in Figure 1. Modeling such a target with MOM alone is intractable because of the overwhelmingly large impedance matrix. The use of asymptotic methods to model the target sacrifices the detail required for the small-to-medium sized electrical perturbations. Accurate yet efficient methods to model targets comprised of electrically small to large regions are obtained by combining two or more solution techniques. These methods are referred to as hybrid techniques.

Hybrid methods have been the subject of prior investigations, but these investigations were limited to simple geometrical shapes due to the limitations of then existing computer programs. The investigations of [1] and [2] are representative. These prior investigations suggested that current based hybrid methods are more robust than field based hybrid techniques.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

This report is confined to current based method, but extends its generality to arbitrary 3-D perfectly conducting geometries. The current based hybrid described here utilizes the MOM for regions requiring rigorous analysis and PO for less demanding regions.

The report is organized as follows. The technical approach is described in the next section followed by a discussion of the software structure. The report is concluded with a summary.

TECHNICAL APPROACH

The formulation is based on the standard MOM electric field integral equation (EFIE). The MOM procedure involves two phases. The first phase consists of filling a system of equations by computing the reaction of each unknown's (current coefficients) currents and field with one another. The size of the left hand side of the system of equations is therefore N^2 where N is the number of unknown current coefficients. The next phase is to solve the system of equations. Utilizing direct solver technology the solution of the system of equations is quite time consuming. In fact, the direct solver computational requirement is $O(N^3)$ thus limiting the problem size which can be solved. Figure 2 shows the magnitude of the solver limitation for MOM formulations. It shows the amount of time required to solve the system of equations alone (fill time not included) assuming $(8/3)N^3$ operations for solution at a sustained TeraFLOP rate. Clearly, the solve phase must be done faster or the number of unknowns to be solved for must be reduced.

The strategy utilized here is to reduce the size of the system of equations to be solved for by utilizing PO as an Ansatz solution where possible. The formulation begins with the Stratton-Chu integral representation of Maxwell's Equations. For the generic geometry shown in Figure 3a, the integral equation for a perfectly conducting body is

$$E^s|_{\tan} = -E^i|_{\tan} = -LJ \quad (1)$$

where the integro-differential operator L is given as

$$LJ = jk\eta \int_S (J + \nabla \nabla' \cdot J / k^2) \Phi ds'. \quad (2)$$

In the above Φ is the free space Green's function, η is the free space impedance, and ∇ is the surface gradient on the body with respect to the unprimed (source) variables, and k is the wavenumber. In the scattering problem, the incident electric field is the known and the induced current is the unknown to be determined.

The original problem of Figure 3a is partitioned into a two region equivalent problem shown in Figure 3b. Region 1 is the MOM region and region 2 is the large PO region. Rewriting equation 1 in a partitioned form yields

$$LJ_1 = E^i|_{\tan} - LJ_2 \quad (3)$$

where the subscripts 1 and 2 represent the MOM and PO regions respectively. The MOM part of the body is that part of the surfaces for which no optic-derived forms are available. These include surface and/or material discontinuities, edges, apertures, and penetrable regions. PO is utilized for the remainder of the body.

Numerical Implementation of Formulation

Equation 3 is numerically solved by the following process. Firstly, the currents, both MOM and PO, are expanded in terms of a set of basis functions and tested utilizing Galerkin's method. Next the PO currents are calculated. Finally, the MOM current coefficients are computed by solving the resulting system of equations utilizing LU decomposition.

The MOM and PO currents are expanded utilizing Rao Wilton Glisson basis functions as follows

$$J_1 = \sum_m I_{1m} v_{1m}$$

$$J_2 = \sum_m I_{2m} v_{1m}$$

where v_{1m} is the Rao Wilton Glisson expansion function of the m^{th} unknown [3], I_1 is the MOM expansion coefficients, and I_2 is the PO expansion coefficients. Note that I_2 is known from a simple nxH calculation on the surface of the target while I_1 is determined by solving the final system of equations.

Next Galerkin testing is used yielding

$$V_1 - \langle w_{1m}, L_2 J_2 \rangle = Z_{11} I_1 \quad (6)$$

where w_{1m} is the weighting function, Z_{11} represents the MOM impedance matrix, and V_1 represents the field incident on the MOM surface.

Finally the system of equations given by equation 6 is solved using LU decomposition techniques. Note that the system of equations is reduced in size relative to a MOM only formulation. Figure 4 summarizes the procedure.

After the currents have been determined by solving equation 6, the fields at the sensor are computed. A convolution between the fields at the sensor and the sensor's system response is performed. This is shown in Figure 5. The range of sensor's which can be modeled is determined by the ability to determine the system response of the sensor and the computation and memory requirements necessary to perform the convolution. Typical sensors which could be considered include synthetic aperture radars, real aperture radars,

and linear frequency modulated radars. A typical radar cross section result is shown in Figure 6.

SOFTWARE STRUCTURE

The software is written in a series of independent modules based upon the McDonnell Douglas CARLOS code. CARLOS was rewritten to implement the hybrid solution and for speed. The rewritten version only implements the electric field integral form of Maxwell's equations. The individual modules allow for expansion of the code to include additional solution techniques such as finite differences and finite elements. The current modules are the input geometry model, input system module, MOM module, PO module, sensor module, and output module.

The input geometry module accepts various forms of geometry representation including PATRAN and ACAD. Sensor information is also processed by the input module. Sensor input includes center frequency, bandwidth, beamwidth, polarization, position relative to target, and trajectory.

The MOM module consists of two submodules: matrix fill and matrix solve. The matrix fill submodule calculates the interaction matrix while the solve module determines the MOM currents. The matrix fill module utilizes Gaussian integration of various quadratures depending on the distance between reacting facets to fill the impedance matrix. The solve module utilized is the LAPACK dense symmetric solver.

The PO module calculates the PO currents on the large smooth surfaces of the target. This module simply implements the cross product of the incident field with the outwardly directed surface normal from each PO region facet.

The sensor module combines the fields generated by the MOM and PO modules as well as perturbs them to account for the MOM/PO interactions. In addition, the sensor module performs the convolution between the returned fields to the sensor and the sensor system response. This can be a very computationally intensive operation due to the large sizes of the fast Fourier transforms required for many sensors.

The last module is the output module which organizes and presents the sensor performance parameters to the user.

A listing of the software is provided in Appendix A.

SUMMARY

A description of a hybrid electromagnetic formulation has been presented. The field presented to the sensor from the target are determined using MOM and PO. These fields

are convolved with the sensor system response to determine the performance parameters of the sensor.

REFERENCES

[1] L.N. Medgyesi-Mitschang and D.S. Wang, "Hybrid solutions for scattering from perfectly conducting bodies of revolution," IEEE Trans. Antennas Propagat., vol. AP-31, pp. 570-583, 1983.

[2] L.N. Medgyesi-Mitschang and D.S. Wang, "Hybrid solutions for scattering from large bodies of revolution with material discontinuities and coatings," IEEE Trans. Antennas Propagat., vol. AP-32, pp. 717-723, 1984.

[3] S.M. Rao, D.R. Wilton, and A.W. Glisson, "Electromagnetic scattering by surfaces of any shape," IEEE Trans. Antennas Propagat., vol. AP-30, pp. 409-418, May 1982.

Figure 1. Hybrid Decomposition of a Helicopter

Figure 2. Amount of Time to Solution (Hours) Versus Number of Unknowns for Direct Solver Assuming Sustained TeraFLOP Rate

3(a)

3(b)

Figure 3. Example Hybrid Decomposition (a) Original Problem; (b) Equivalent Problem

Illuminated
MM Surface

I_1, I_2 = MM and PO Current Coefficients Respectively

Z_{11} = MM Interaction Matrix

Z_{12} = MM/PO Interaction Matrix

E_1, E_2 = MM and PO Excitation Matrices

Original Exact System:
$$\begin{bmatrix} Z_{11} & Z_{12} \\ Z_{21} & Z_{22} \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \end{bmatrix}$$

Asymptomatic Solution: $Z_{11}I_1 = E_1 - Z_{12}I_2$

Note: Hybrid Solution Has Reduced Size Linear System

Figure 4. Simplified Hybrid Formulation

Figure 5. Convolution Stage

RCS of T-39

VV

HH

Cross Pol

Figure 6. Sample RCS Calculation

Appendix A
Source Code Listing

```

      subroutine acad(nunit,nvrt,nface,xvert,yvert,zvert,idxvf)
c
c read acad geometry files
c
      implicit none
      integer idxvf(3,*),nvrt,nfct,nunit,nsolids,i,nvrtemp,j,numpoly
      +,nfacet,k,idx1,idx2,idx3,nface
      real xvert(*),yvert(*),zvert(*)
      nvrt=0
      nfct=0
c
c title line must be skipped
c
      read(nunit,*)
      read(nunit,*)nsolids
c
c loop over solids defining geometry
c
      do 500 i=1,nsolids
        read(nunit,*)
        read(nunit,*)
        read(nunit,*)nvrtemp
c
c read vertices
c
        do 100 j=nvrt+1,nvrt+nvrtemp
          read(nunit,*)xvert(j),yvert(j),zvert(j)
100      continue
          read(nunit,*)numpoly
c
c loop over polygons
c
          do 300 j=1,numpoly
            read(nunit,*)
            read(nunit,*)nfacet
            do 200 k=nfct+1,nfct+nfacet
c
c read and set facet indices
c
              read(nunit,*)idx1,idx2,idx3
              idxvf(1,k)=idx1+nvrt
              idxvf(2,k)=idx2+nvrt
              idxvf(3,k)=idx3+nvrt
              if (idxvf(1,k).lt.nvrt+1.or.idxvf(1,k).gt.nvrt+nvrtemp.or.
&              idxvf(2,k).lt.nvrt+1.or.idxvf(2,k).gt.nvrt+nvrtemp.or.
&              idxvf(3,k).lt.nvrt+1.or.idxvf(3,k).gt.nvrt+nvrtemp) then
                write(6,*)' error: Face vertex index out of range'
                stop
              endif
200          continue
c
c increment facet index counter
c
            nfct=nfct+nfacet
300        continue
c
c increment vertex index counter
c
          nvrt=nvrt+nvrtemp
500      continue
c
c total facets and edges
c
      nface=nfct
      return
      end

```

```

      subroutine agm(nunit,numpat,nface,nedge,xvert,yvert,zvert,idxvf)
c
c read agm file
c
      implicit none
      real xvert(*),yvert(*),zvert(*),x1,y1,z1,x2,y2,z2,x3,y3,z3
      integer idxvf(3,*),nvrt,nfct,nunit,i,numpat,nface,nedge
      nvrt=0
      nfct=0
c
c read vertices forming facets
c
100  read(nunit,*,end=300)
      read(nunit,10)x1,y1,z1
      read(nunit,10)x2,y2,z2
      read(nunit,10)x3,y3,z3
10   format(3g10.5)
      nfct=nfct+1
      do 110 i=1,nvrt
         if(x1.eq.xvert(i).and.y1.eq.yvert(i).and.z1.eq.zvert(i))then
            idxvf(1,nfct)=i
            goto 120
         endif
110  continue
      nvrt=nvrt+1
      xvert(nvrt)=x1
      yvert(nvrt)=y1
      zvert(nvrt)=z1
      idxvf(1,nfct)=nvrt
120  continue
      do 200 i=1,nvrt
         if(x2.eq.xvert(i).and.y2.eq.yvert(i).and.z2.eq.zvert(i))then
            idxvf(2,nfct)=i
            goto 210
         endif
200  continue
      nvrt=nvrt+1
      xvert(nvrt)=x2
      yvert(nvrt)=y2
      zvert(nvrt)=z2
      idxvf(2,nfct)=nvrt
210  continue
      do 220 i=1,nvrt
         if(x3.eq.xvert(i).and.y3.eq.yvert(i).and.z3.eq.zvert(i))then
            idxvf(3,nfct)=i
            goto 230
         endif
220  continue
      nvrt=nvrt+1
      xvert(nvrt)=x3
      yvert(nvrt)=y3
      zvert(nvrt)=z3
      idxvf(3,nfct)=nvrt
230  continue
      goto 100
300  continue
      numpat=nvrt
      nface=nfct
      nedge=0
      return
      end

```

```

      subroutine block(ib,jb)
c
c Determine if any of the unknowns from the test and source surfaces fall
c within the system matrix sub-block being filled.
c Eliminate any unknowns from the index arrays
c which are not in the sub-block. Set nincl=0 if none of the test/source
c unknown combinations will contribute to the sub-block being filled.
c
      implicit none
      include 'arraydim.inc'
      include 'edges.inc'
      include 'index.inc'
      include 'block.inc'
      integer ni,ib,nj,jb,itype,i,indii,jtype,j,indjj
      ni=numedg(ib)
      nj=numedg(jb)
c
c itype and jtype variables
c 0 - none of the edges are in the sub-block
c 1 - edges are only in the sub-block
c
      itype=0
c
c loop over edges
c
      do 100 i=1,ni
         indii=iabs(indxi(i))
         if((iblock1.le.indii.and.indii.le.iblock2)
+          .or.(jblock1.le.indii.and.indii.le.jblock2))then
c
c edge is within sub block
c
            itype=1
         else
c
c edge is not within sub block.
c for edge not within sub block set temporary global unknown to 0.
c
            indxi(i)=0
         endif
      100 continue
c
c if no unknowns within subn block, quit on this surface
c
      if(itype.eq.0)then
         nincl=0
         return
      endif
      jtype=0
c
c loop over edges
c
      do 200 j=1,nj
         indjj=iabs(indxj(j))
         if((jblock1.le.indjj.and.indjj.le.jblock2)
+          .or.(iblock1.le.indjj.and.indjj.le.iblock2))then
c
c edge is within sub block
c
            jtype=1
         else
c
c edge is not within sub block.
c for edge not within sub block set temporary global unknown to 0.
c
            indxj(j)=0

```

```
                endif
200  continue
C
C if no unknowns within sub block, quit on this surface
C
    if(jtype.eq.0)then
        nincl=0
        return
    endif
    return
end
```



```

      subroutine block2(ni,indx,nj,jndx,numb)
C
C are test and source unknowns in sub-block?
C
      integer indx(*),jndx(*)
      include 'arraydim.inc'
      include 'index.inc'
      include 'block.inc'
      numb=1
      do 310 i=1,ni
      it=indx(i)
      do 310 j=1,nj
      jt=jndx(j)
      do 300 kloc=1,nincl
      ib=iabs(indxi(it))
      jb=iabs(indxj(jt))
      if(iblck1.le.ib.and.ib.le.iblck2.and.
+      jblck1.le.jb.and.jb.le.jblck2)return
      if(jblck1.le.ib.and.ib.le.jblck2.and.
+      iblck1.le.jb.and.jb.le.iblck2)return
300    continue
310    continue
      numb=0
      return
      end

```

```

      subroutine blocki(ni,indx,numb)
c
c are test unknowns in sub-block?
c
      implicit none
      integer indx(*),numb,i,ni,it,kloc
      include 'arraydim.inc'
      include 'index.inc'
      numb=1
      do 110 i=1,ni
         it=indx(i)
         do 100 kloc=1,nincl
            if(indxi(it).ne.0) return
100      continue
110      continue
      numb=0
      return
      end

```

```

      subroutine blockj (nj, jindx, numb)
c
c source unknowns in sub-block?
c
      implicit none
      integer jindx(*), numb, j, nj, jt, kloc
      include 'arraydim.inc'
      include 'index.inc'
      numb=1
      do 210 j=1, nj
         jt=jindx(j)
         do 200 kloc=1, nincl
            if(indxj(jt).ne.0) return
200      continue
210      continue
      numb=0
      return
      end

```

```

      subroutine build(zxx,it,jt,ism,zl,lzl)
C
C add the element zxx to the system matrix z1 at all locations.
C
      implicit none
      complex*8 zxx,z1(*),sign
C
      include 'arraydim.inc'
C
      include 'index.inc'
      include 'block.inc'
      integer nbi,nbj,kloc,nb,lzl,ib,it,jb,jt,ism,kb,indb
      if(part) then
         nbi=iblock2-iblock1+1
         nbj=jblock2-jblock1+1
      endif
      do 100 kloc=1,nincl
         nb=lzl
         ib=indx1(it)
         jb=indxj(jt)
         if(ib.eq.0 .or. jb.eq.0) go to 100
         sign=curij
         if(ib*jb.lt.0) sign=-sign
         ib=iabs(ib)
         jb=iabs(jb)
C
C build symmetric matrix in packed form.
C
C skip the diagonal elements of a symmetric matrix, if the elements are below
C the main diagonal of z1.
         if(ism.ne.0 .and. it.eq.jt .and. ib.gt.jb) go to 100
C double the elements of a full matrix, if the elements are on the main
C diagonal of z1.
         if(ib.eq.jb) then
            if(ism.eq.0 .or. it.ne.jt) then
               sign=2.0*sign
            endif
         endif
         if(ib.gt.jb) then
            kb=ib
            ib=jb
            jb=kb
         endif
         indb=(jb-1)*jb/2+ib
C
C check if a sub-block of a partitioned matrix is being filled.
C
         if(part) then
            if(iblock1.le.ib .and. ib.le.iblock2 .and.
1          jblock1.le.jb .and. jb.le.jblock2) then
C matrix element is in the sub-block
               nb=nbi
               ib=ib-iblock1+1
               jb=jb-jblock1+1
               if(iblock1.eq.jblock1 .and. iblock2.eq.jblock2) then
                  indb=(jb-1)*jb/2+ib
               else
                  indb=(jb-1)*nb+ib
               endif
            elseif(jblock1.le.ib.and.ib.le.jblock2.and.
+              iblock1.le.jb.and.jb.le.iblock2)then
C
C matrix element is in transpose of sub-block
C
               nb=nbj
               ib=ib-jblock1+1

```

```
        jb=jb-iblock1+1
        indb=nbi*nbj+(jb-1)*nb+ib
    else
c matrix element is not in either sub-block
        go to 100
    endif
endif
    z1(indb)=sign*zxx+z1(indb)
100 continue
return
end
```

```
function db(x)
c
c express x in decibels
c
  implicit none
  real*4 db,x
  db=-1000.
  if(x.eq.0.)return
  db=10.*log10(x)
  return
end
```

```
      subroutine dot(a,b,x,n)
c
c  x=x+c*(a dot b)
c
      implicit none
      integer i,n
      complex a(*),b(*),x,temp
      temp=(0.,0.)
      do 10 i=1,n
         temp=temp+a(i)*b(i)
10    continue
      x=x+temp
      return
      end
```

```

      subroutine edges(ifacvrt,nface,iedvrt,nedge)
c
c determine patch edges
c
c ifacvrt=indices of vertices forming faces
c nface=number of faces
c iedvrt=indices of vertices forming edges
c nedge=number of edges
c
      implicit none
      integer ifacvrt(3,*),iedvrt(2,*)
      integer nedge,iface,nface,iedge,i1,i2,je,j1,j2
c
c initialize number of edges
c
      nedge=0
c
c loop over faces determining the edges of the face iface
c
      do 200 iface=1,nface
        do 190 iedge=1,3
          if(iedge.eq.1)then
            i1=ifacvrt(2,iface)
            i2=ifacvrt(3,iface)
          elseif(iedge.eq.2)then
            i1=ifacvrt(3,iface)
            i2=ifacvrt(1,iface)
          else
            i1=ifacvrt(1,iface)
            i2=ifacvrt(2,iface)
          endif
c
c does edge exist?
c
c if edge has already been found: common to a previously
c analyzed face goto next edge
c
          do 170 je=1,nedge
            j1=iedvrt(1,je)
            j2=iedvrt(2,je)
c
c new edge?
c
            if((i1.eq.j1.and.i2.eq.j2).or.
              & (i1.eq.j2.and.i2.eq.j1))goto 190
          170 continue
c update edge count
c
          nedge=nedge+1
c
c store newly found edge
c
          iedvrt(1,nedge)=i1
          iedvrt(2,nedge)=i2
        190 continue
      200 continue
      return
      end

```



```

      subroutine eindex(i2,indi)
c
c determine the global unknown numbers corresponding to the edges
c of surface i2
c
      implicit none
      integer indi(*),ni,i2,ie,k,kz
      include 'arraydim.inc'
      include 'edges.inc'
      include 'indx2.inc'
      include 'ecurind.inc'
c
c assign # edges on surface i2 to ni
c
      ni=numedg(i2)
c
c assign the first edge of surface i2 to ie.
c used to calculate offset for global unknown # assignment.
c inde=index of first edge of surface i2
c
      ie=inde(i2)
c
c loop over the edges of surface i2
c
      do 30 k=1,ni
c
c calculate the index of iedind that contains the unknown
c # of edge k
c
      kz=ie+k-1
c
c indi=global unknown # of the kth edge of surface i2
c
      indi(k)=iedind(kz)
30  continue
      return
      end

```

```

      subroutine elegind(newind,noldind,jcur,nedge)
c
c newind <-- noldind (edge indices or unknowns)
c jcur=jcur-1 (decrement edge/unknown count)
c
      implicit none
      include 'arraydim.inc'
      include 'ecurind.inc'
      integer indnew,newind,indold,noldind,iabold,jcur,
+iedge,nedge,indz
      indnew=newind
      indold=noldind
      if(indold.eq.0) return
      if(iabs(indnew).eq.iabs(indold)) return
      iabold=iabs(indold)
      jcur=jcur-1
      do 10 iedge=1,nedge
         indz=iedind(iedge)
         if(indz.eq.indold) then
            indz=indnew
         elseif(indz.eq.-indold) then
            indz=-indnew
         endif
         if(indz.gt.iabold) then
            indz=indz-1
         elseif(indz.lt.-iabold) then
            indz=indz+1
         endif
         iedind(iedge)=indz
10      continue
      return
      end

```

```

      subroutine faces(ivrt,nedge,itype,nvf,nfaces)
C
C determine face information
C
C
C ivrt=indices of vertices forming edges
C nedge=number of edges
C itype=work array that contains edge numbers with vertices common to iedge
C nvf=indices of vertices forming faces
C nfaces = number of faces
C ntype=# of other edges that have common vertices of edge iedge
C
      implicit none
      integer ivrt(2,*),nvf(3,*),itype(*)
      integer nfaces,iedge,nedge,ntype,jedge,i,j,kedge,k
C
C initialize face count
C
      nfaces=0
C
C loop over edges which may form faces
C
      do 500 iedge=1,nedge-2 ! nedge-2 because 3 edges comprise face
C
C initialize count of edges with vertices common to iedge
C
      ntype=0
C
C loop over edges other than iedge that have yet to be checked for common
C vertices
C
      do 200 jedge=iedge+1,nedge ! all edges numbered < iedge+1 have
C either been checked or are the current edge being check (iedge)
C
C loop over vertices of both iedge and jedge testing for common vertices
C
      do 190 i=1,2
        do 180 j=1,2
          if(ivrt(i,iedge).eq.ivrt(j,jedge))then
C
C update count of edges with common vertices to iedge
C
          ntype=ntype+1
C
C store edge numbers with vertices common to iedge
C
          itype(ntype)=jedge
          goto 200
        endif
      180 continue
      190 continue
      200 continue
C
C loop over edges with common vertices to iedge testing for closed perimeter
C
      do 400 jedge=1,ntype-1 ! ntype-1 because it takes 2 edges+iedge to
C form a face
C
      do 390 kedge=jedge+1,ntype ! all itype arguments < jedge+1 have
C been checked to see if they form a perimeter with iedge already already
C or are the current itype argument being checked (jedge)
C
C loop over vertices of both jedge and iedge edges
C
      do 380 j=1,2 ! jedge vertices
        do 370 k=1,2 ! kedge vertices

```

```

c
c face formed when two edges with same common vertiex to either vertex of
c edge iedge and edges are not common to the same vertex of iedge
c
      if((ivrt(j,itype(jedge)).eq.ivrt(k,itype(kedge))).and.
        &(ivrt(j,itype(jedge)).ne.ivrt(1,iedge).and.ivrt(j,itype(jedge))
        &.ne.ivrt(2,iedge)))then
c
c update face count (nfaces)
c
      nfaces=nfaces+1
      nvf(1,nfaces)=ivrt(1,iedge)
      nvf(2,nfaces)=ivrt(2,iedge)
      nvf(3,nfaces)=ivrt(j,itype(jedge))
      goto 400
    endif
370    continue
380    continue
390    continue
400    continue
500  return
    end

```

```

      subroutine fldrcs(ifreq,nfreq,fr,freq)
c
c print out far fields.
c
      implicit none
      logical freq
      include 'arraydim.inc'
      include 'problem.inc'
      include 'wavenumber.inc'
      include 'threat.inc'
      include 'order.inc'
      include 'dbsm.inc'
      include 'scatter.inc'
      include 'const.inc'
      complex*8 a1
      real*4 dtor,fr,p0,s,db,s1,s2,s3,s4
      integer k,numb,k1,ifreq,nfreq
      dtor=pi/180.
      write(6,4) title,fk,fr,2.*pi/fk
4      format(1h1,'problem: ',a80,/,/, ' fk = ',g12.4,/,/, ' fr = ',f9.3,/,
& ' lambda = ',g12.4,/)
      write(6,*)
      if(numzel.ne.0) then
        write(6,*) '      3d-patch'
      endif
      p0=(fk*fk*eta/2./pi)**2/(4.*pi)
      s=2.*db(0.2997925/fr)
      a1=(0.,-1.)
      if(monbi.gt.0) then
        write(6,16) 'mono-static rcs'
        write(6,18) 2,2
      elseif(monbi.eq.0) then
        write(6,16) 'bi-static rcs'
        write(6,*) '      phi = ',phi,'      thet = ',thet
        write(6,18) 2,2
      endif
16      format(//,5x,a,/)
18      format(34x,'sigma/lambda**',i1,' (db)',36x,'sigma/m**',i1,' (db)',
1 /,20x,45(1h-),7x,45(1h-),/,
2 5x,'phis',5x,'thes',2(9x,'00',10x,'00',10x,'00',10x,'00',5x),/,
3 1h+,15x,2(9x,'--',10x,'//',10x,'-/',10x,'/-',5x))
      do 119 k=1,nang
        ttscat(k)=a1*ttscat(k)
        s1=abs(ttscat(k))
        s1=db(p0*s1*s1)
        ppscat(k)=a1*ppscat(k)
        s2=abs(ppscat(k))
        s2=db(p0*s2*s2)
        tpscat(k)=a1*tpscat(k)
        s3=abs(tpscat(k))
        s3=db(p0*s3*s3)
        ptscat(k)=a1*ptscat(k)
        s4=abs(ptscat(k))
        s4=db(p0*s4*s4)
23      write(6,23) phis(k),thes(k),s1,s2,s3,s4,s1+s,s2+s,s3+s,s4+s
        format(1x,2f8.1,4f12.2,4x,4f12.2)
      if(dbsm) then
        s1=s1+s
        s2=s2+s
      endif
c plot file
      if(iangle(k).lt.0) then
        numb=1
        do 110 k1=k+1,nang
          if(iangle(k1).gt.0) then
            numb=numb+1

```

```

        else
            go to 111
        endif
110    continue
111    continue
        if(nang.gt.1) then
            write(31,*) numb,' dbsm = ',dbsm
        elseif(ifreq.eq.1) then
            write(31,*) nfreq,' dbsm = ',dbsm
        endif
    endif
    if(nang.gt.1) then
        if(abs(iangle(k)).eq.1) then
            write(31,1) thes(k),s1,s2
        else
            write(31,1) phis(k),s1,s2
        endif
    else
        if(freq) then
            write(31,1) fr,s1,s2
        else
            write(31,1) fk,s1,s2
        endif
    endif
1    format(1x,f8.2,2f10.2,i5)
119    continue
    return
end

```

subroutine gauss

c determine gaussian abscissas and weights

c
c

```

implicit none
include 'intord.inc'
include 'triint.inc'
real*4 qcoef(28),eta(28),x(28)
real*4 etai,xi,etamxi
integer numpnt(8),np(8),mp(8),mlt(28)
integer i,npnt,k,nformula,imult

```

c 1 point

```

data numpnt(1),np(1),mp(1) /1,1,1/
data mlt(1) /1/
data qcoef(1) /1.0/
data eta(1) /.3333333/
data x(1) /.3333333/

```

c 3 point

```

data numpnt(2),np(2),mp(2) /3,2,1/
data mlt(2) /3/
data qcoef(2) /.3333333/
data eta(2) /.1666666/
data x(2) /.6666666/

```

c 7 point

```

data numpnt(3),np(3),mp(3) /7,3,3/
data (mlt(i),i=3,5) /1,3,3/
data (qcoef(i),i=3,5) /.225,.1259392,.1323941/
data (eta(i),i=3,5) /.3333333,.1012865,.4701421/
data (x(i),i=3,5) /.3333333,.7974270,.0597159/

```

c 12 point

```

data numpnt(4),np(4),mp(4) /12,6,3/
data (mlt(i),i=6,8) /3,3,6/
data (qcoef(i),i=6,8) /.0508449,.1167863,.0828511/
data (eta(i),i=6,8) /.0630890,.2492867,.3103524/
data (x(i),i=6,8) /.873822,.5014265,.6365025/

```

c 15 point

```

data numpnt(5),np(5),mp(5) /15,9,3/
data (mlt(i),i=9,11) /3,6,6/
data (qcoef(i),i=9,11) /.0530778,.0708531,.0692747/
data (eta(i),i=9,11) /.0649305,.5170399,.0438635/
data (x(i),i=9,11) /.870139,.2845756,.3135592/

```

c 16 point

```

data numpnt(6),np(6),mp(6) /16,12,5/
data (mlt(i),i=12,16) /1,3,3,3,6/
data (qcoef(i),i=12,16) /.1443156,.1032174,.0324585,
+.0950916,.0272303/
data (eta(i),i=12,16) /.3333333,.1705693,.0505472,
+.4592926,.2631128/
data (x(i),i=12,16) /.3333333,.6588614,.8989055,
+.0814148,.0083948/

```

c 19 point

```

data numpnt(7),np(7),mp(7) /19,17,6/
data (mlt(i),i=17,22) /1,3,3,3,3,6/
data (qcoef(i),i=17,22) /.0971358,.0313347,.0778275,
+.0796477,.0255777,.0432835/
data (eta(i),i=17,22) /.3333333,.4896825,.4370896,
+.1882035,.0447295,.221963/
data (x(i),i=17,22) /.3333333,.020635,.1258208,
+.6235929,.910541,.0368384/

```

c 25 point

```

data numpnt(8),np(8),mp(8) /25,23,6/
data (mlt(i),i=23,28) /1,3,3,6,6,6/
data (qcoef(i),i=23,28) /.0798945,.0711238,.0082238,
+.0454306,.0373598,.0308867/
data (eta(i),i=23,28) /.3333333,.4250862,.0233089,

```

```

+.223767,.3587401,.1432954/
data (x(i),i=23,28) /.3333333,.1498276,.9533823,
+.1479256,.029946,.0356326/
do 100 npnt=1,25
  nmpont(npnt)=0
100 continue
  k=0
c
c loop over the 8 data sets(1,3,7,12,15,16,19,25)
c
  do 200 nformula=1,8
    npnt=numpnt(nformula)
    nmpont(npnt)=k+1
c
c loop over eta and x indices for each data set
c (i.e., 19 data set: loop over 17-22)
c
    do 150 i=np(nformula),np(nformula)+mp(nformula)-1
      etai=eta(i)
      xi=x(i)
      etamxi=1.-etai-xi
      do 150 imult=1,mlt(i)
        k=k+1
        if(k.gt.108) then
          write(6,*) ' error: subroutine gauss'
          stop
        endif
        weight(k)=0.5*qcoef(i)
        if(imult.eq.1) then
          alfa(k)=etai
          beta(k)=xi
        elseif(imult.eq.2) then
          alfa(k)=etai
          beta(k)=etamxi
        elseif(imult.eq.3) then
          alfa(k)=xi
          beta(k)=etai
        elseif(imult.eq.4) then
          alfa(k)=xi
          beta(k)=etamxi
        elseif(imult.eq.5) then
          alfa(k)=etamxi
          beta(k)=etai
        elseif(imult.eq.6) then
          alfa(k)=etamxi
          beta(k)=xi
        endif
      150 continue
    200 continue
    if(nmpont(nself).eq.0) then
      write(6,*) ' error: nself integration formula not available'
      stop
    endif
    if(nmpont(nnear).eq.0) then
      write(6,*) ' error: nnear integration formula not available'
      stop
    endif
    if(nmpont(nmid).eq.0) then
      write(6,*) ' error: nmid integration formula not available'
      stop
    endif
    if(nmpont(nfar).eq.0) then
      write(6,*) ' error: nfar integration formula not available'
      stop
    endif
  return

```


end

```

subroutine geoin(level,iunord)
implicit none
include 'arraydim.inc'
include 'const.inc'
include 'wavenumber.inc'
include 'numbers.inc'
include 'surcord.inc'
include 'faces.inc'
include 'indx1.inc'
include 'edges.inc'
include 'indx2.inc'
include 'ecurind.inc'
include 'unit.inc'
include 'patchtol.inc'
include 'numj.inc'
include 'order.inc'
integer mompo(nbndmax),indx(2)
integer level(*),iunord(*)
integer nint(nbndmax),next(nbndmax),nje(nbndmax),nren(nbndmax),
& nrpls(nbndmax),nrmin(nbndmax)
integer npts,numface,numedge,inptyp,iunit,nptsp1,numfcp1,nedgpl
+,nfact,nedg,nummove,idx1,idx2,istart,iend,i1,i2,i3,ibnd,nside,
+jstart,jend,jedge,jdx1,jdx2,ijunc,nedint,nedext,nedjun,jbnd,
+j1,j2,ijnctyp,kedge,iside,nunknon,loc,new,iold,idx,ind2,nhy,
+i,j,id2,ii,ioffset,jj,kk,ioffsete,if1,nlev,neuler
real*4 dstance,x21,y21,z21,x31,y31,z31,deltax,deltay,deltaz,
+delta,delinv,del,xx,yy,zz
character*80 geofil
write(6,10)
10  format(//'      geometry input ',//)
c
c initialize number of surfaces , vertices, faces, and edges
c
      nub=0
      npts=0
      numface=0
      numedge=0
c
c loop to read geometry surfaces
c
20  continue
c
c inptyp > 0: mdac carlos file(standard input)
c inptyp = -1: mdac carlos file(external)
c inptyp = -2: acad facet file
c inptyp = -3: agm file
c inptyp = -4: patran-iges file
c
      read(5,*)inptyp
c
c check for last surface read
c
      if(inptyp.eq.0)goto 100      ! terminate geometry files to be read in
c
c count surfaces
c
      nub=nub+1
      write(6,30)nub
30  format(///,'      surface number ',i2)
      if(nub.gt.nbndmax)then
          write(6,*)' insufficient dimension: nbndmax = ',nbndmax
          write(6,*) '                      nub = ',nub
          stop
      endif
c
c mompo=mom or po solution type for surface nub

```

```

c mompo=2 (po solution)
c mompo=1 (mm solution)
c
  read(5,*)momp0(nub)
  write(6,*)
  if(momp0(nub).eq.2)then
    write(6,*)' PO surface'
  elseif(momp0(nub).eq.1)then
    write(6,*)' MM surface'
  endif
c
c read geometry files
c
  iunit=5
c
c prepare to read external files/devices
c
  if(inptyp.lt.0)then
    read(5,40)geofil
40    format(a80)
    write(6,50)geofil
50    format(1x,'geometry file name: ',a80)
    iunit=1
    open(unit=1,file=geofil,status='old')
  endif
c
c increment total vertices, faces, and edges by 1. Principally
c used as pointers.
c
  nptspl=npts+1
  numfcpl=numface+1
  nedgpl=numedge+1
c
c call to different file types
c
c
c iunit = device number to be read
c inptyp=number of vertices (for mdac format)
c nfact=number of faces
c nedg=number of edges
c xv, yv, and zv are vertices
c idxvf are indices of vertices comprising faces
c idxve are indices of vertices comprising edges
c
  if(inptyp.ge.-1)then
    if(inptyp.lt.0)write(6,*)' mdac carlos file'
    call mdacin(iunit,inptyp,nfact,nedg,xv(nptspl),yv(nptspl),
&   zv(nptspl),idxvf(1,numfcpl),idxve(1,nedgpl))
    elseif(inptyp.eq.-2)then
      write(6,*)' acad file'
      call acad(iunit,inptyp,nfact,xv(nptspl),yv(nptspl),
&   zv(nptspl),idxvf(1,numfcpl))
      nedg=0
    elseif(inptyp.eq.-3)then
      write(6,*)' agm file'
      call agm(iunit,inptyp,nfact,nedg,xv(nptspl),yv(nptspl),
&   zv(nptspl),idxvf(1,numfcpl))
    elseif(inptyp.eq.-4)then
      write(6,*)' patran file'
      call patran(iunit,inptyp,nfact,nedg,xv(nptspl),yv(nptspl),
&   zv(nptspl),idxvf(1,numfcpl))
    endif
c
c close external file
c
  if(iunit.ne.5)close(unit=iunit)

```

```

c
c number of vertices of surface nub (nup)
c note: inptyp is modified in routine that reads geometry file
c (e.g. mdacin,acad,etc.)
c
      nup(nub)=inptyp
c
c start index of global vertices of surface nub
c
      indp(nub)=nptsp1
c
c rolling count of total vertices
c previous total (npts) plus number of vertices of surface nub= new total (npts)
c
      npts=npts+inptyp
      if(npts.gt.nptsmax)then
        write(6,*)'insufficient dimension : nptsmax = ',nptsmax
        write(6,*)'                                npts = ',npts
        stop
      endif
      write(6,*)
      write(6,*)'number of vertex points input= ',inptyp
      write(6,*)'number of faces input = ',nfact
      write(6,*)'number of edges input = ',nedg
c
c edge and face information determined
c if we have edges calculate faces, if we have faces calculate edges
c
      if(nfact.ne.0)then
        call edges(idxvf(1,numfcp1),nfact,idxve(1,nedgpl),nedg)
      else
c
c edtype used only to conserve memory
c
        call faces(idxve(1,nedgpl),nedg,edtype(nedgpl),idxvf(1,numfcp1),
          & nfact)
        endif
c
c write face and edge information
c
      write(6,*)
      write(6,*)'number of faces analyzed = ',nfact
      write(6,*)'number of edges analyzed = ',nedg
      write(6,*)
c
c orient other faces relative to face 1: all faces oriented for
c outwardly directed surface normal
c
c
c idxef is used as work array only here (saves memory)
c
c
c nfact=number of faces, idxef=indices of edges comprising faces
c
      call reorint(nfact,idxvf(1,numfcp1),idxef(1,numfcp1))
      write(6,*)'face 1 vertex indices: ',idxvf(1,numfcp1),
        & idxvf(2,numfcp1),idxvf(3,numfcp1)
      write(6,*)'edge 1 vertex indices: ',idxve(1,nedgpl),
        & idxve(2,nedgpl)
c
c numfac(nub)=number of faces comprising surface nub
c indf=index of first face of surface nub (numface is the number
c of faces through surface nub-1)
c
      numfac(nub)=nfact
      indf(nub)=numface+1

```

```

c
c update face count through surface nub
c
c numface=numface+nfact
c if(numface.gt.nfacmax)then
c   write(6,*)', insufficient dimension: nfacmax = ',nfacmax
c   write(6,*)', numface = ',numface
c   stop
c   endif
c
c numedg(nub)=number of edges comprising surface nub
c inde=index of first edge of surface nub (numedge is the number
c of edges through surface nub-1)
c
c   numedg(nub)=nedg
c   inde(nub)=numedge+1
c
c update edge count through surface nub
c
c numedge=numedge+nedg
c if(numedge.gt.nedgmax)then
c   write(6,*)', insufficient dimension: nedgmax = ',nedgmax
c   write(6,*)', numedge = ',numedge
c   stop
c   endif
c
c end of surface read in and processed
c
c   goto 20
c
c go to here (100) if all surfaces read in
c
c 100 continue
c
c npoynt=total # vertices
c nfases = total # faces
c nedges = total # edges
c
c   npoynt=npts
c   nfases=numface
c   nedges=numedge
c   write(6,*)
c
c collapse vertices
c
c if 2 vertices are separated by a distance < pattol
c they are collapsed to be a single vertex
c
c   if(pattol.ne.0.)then
c
c initialize count of collapsed vertices
c
c   nummove=0
c
c loop over vertices
c
c   do 120 idx1=1,npts
c loop over vertices yet to be checked
c
c   do 110 idx2=idx1+1,npts
c
c calculate separation distance between vertices idx1 and idx2
c
c   distance=sqrt((xv(idx1)-xv(idx2))**2+(yv(idx1)-yv(idx2))**2+

```

```

      &          (zv(idx1)-zv(idx2))**2)
c
c if not the same coordinates and separated by < pattol then collapse vertices
c
c          if(ddistance.ne.0..and.ddistance.lt.pattol)then
c
c update collapsed vertex count
c
c          nummove=nummove+1
c
c collapse vertices: make higher indexed vertex equal to lower:
c this allows the vertex to be checked against distance between it
c and other vertices
c
c          xv(idx2)=xv(idx1)
c          yv(idx2)=yv(idx1)
c          zv(idx2)=zv(idx1)
c          goto 120
c      endif
110      continue
120      continue
      write(6,*)' Number of vertex points collapsed (pattol):'
      write(6,*)' Vertex points changed - ',nummove
    endif
c
c calculate face center coordinates
c
c
c loop over surfaces
c
c      do 140 idx1=1,nub
c          istart=indf(idx1)
c          iend=indf(idx1)+numfac(idx1)-1
c
c loop over faces of surface idx1
c istart and iend represent range of indices of faces of surface idx1
c
c          do 130 numfcpl=istart,iend
c
c calculate global indices of vertices forming faces:
c add offset indp(idx1) to index numbers provided by individual surfaces
c i1,i2,i3=global vertex indices
c
c          i1=indp(idx1)+idxvf(1,numfcpl)-1
c          i2=indp(idx1)+idxvf(2,numfcpl)-1
c          i3=indp(idx1)+idxvf(3,numfcpl)-1
c
c calculate center coordinates
c
c          xc(numfcpl)=(xv(i1)+xv(i2)+xv(i3))*0.33333333
c          yc(numfcpl)=(yv(i1)+yv(i2)+yv(i3))*0.33333333
c          zc(numfcpl)=(zv(i1)+zv(i2)+zv(i3))*0.33333333
130      continue
140      continue
c
c calculate edge lengths
c
c
c loop over surfaces
c
c      do 160 idx1=1,nub
c
c loop over edges of surface idx1
c istart and iend represent range of indices of edges of surface idx1
c
c          istart=inde(idx1)

```

```

        iend=inde(idxl)+numedg(idxl)-1
        do 150 nedgpl=istart,iend
C
C initialize edtype (will be used later to determine the type (interior
C /exterior,etc.) of edge
C
        edtype(nedgpl)=-2
C
C calculate global indices of vertices forming edges:
C add offset indp(idxl) to index numbers provided by individual surfaces
C i1,i2=global vertex indices
C
        i1=indp(idxl)+idxve(1,nedgpl)-1
        i2=indp(idxl)+idxve(2,nedgpl)-1
C
C calculate edge length
C
        edlen(nedgpl)=sqrt((xv(i1)-xv(i2))**2+(yv(i1)-yv(i2))**2+
&      (zv(i1)-zv(i2))**2)
        if ((idxl .eq. 1) .and. (nedgpl .eq. 1)) then
            write(6,*)"edge 1 v1 cords =", xv(i1), yv(i1), zv(i1)
            write(6,*)"edge 1 v2 cords =", xv(i2), yv(i2), zv(i2)
            write(6,*)"edge length =", edlen(nedgpl)
        endif
C
C edge length cannot = 0
C
        if(edlen(nedgpl).eq.0.)then
            write(6,*)" error: zero edge length"
            write(6,*)" surface number= ',idxl
            write(6,*)" edge = ',nedgpl-istart+1
            write(6,*)" point (1) = ',idxve(1,nedgpl)
            write(6,*)" (2) = ',idxve(2,nedgpl)
            write(6,*)" x,y,z (1) = ',xv(i1),yv(i1),zv(i1)
            write(6,*)" (2) = ',xv(i2),yv(i2),zv(i2)
            stop
        endif
150    continue
160    continue
C
C determine edges which form faces (create idxef)
C this also reorients edge array idxve to be
C consistent with idxvf
C
C loop over surfaces
C
        do 190 ibnd=1,nub
            istart=indf(ibnd)
            iend=indf(ibnd)+numfac(ibnd)-1
C
C loop over beginning and ending indices of
C faces of surface ibnd
C
            do 180 numfcpl=istart,iend
C
C loop over the sides of faces
C
                do 170 nside=1,3
C
C calculate global indices of vertices forming faces:
C add offset indp(ibnd) to index numbers provided by
C individual surfaces
C i1 and i2=vertices forming edge opposite vertex nside
C
                    if(nside.eq.1)then

```



```

        i1=indp(ibnd)+idxvf(2,numfcp1)-1
        i2=indp(ibnd)+idxvf(3,numfcp1)-1
    elseif(inside.eq.2) then
        i1=indp(ibnd)+idxvf(3,numfcp1)-1
        i2=indp(ibnd)+idxvf(1,numfcp1)-1
    else
        i1=indp(ibnd)+idxvf(1,numfcp1)-1
        i2=indp(ibnd)+idxvf(2,numfcp1)-1
    endif

c
c calculate starting and ending indices of
c edges of surface ibnd
c
        jstart=inde(ibnd)
        jend=inde(ibnd)+numedg(ibnd)-1

c
c loop over edges
c
        do 165 jedge=jstart,jend

c
c calculate global indices of vertices forming edges:
c add offset indp(ibnd) to index numbers provided by individual surfaces
c jdx1,jdx2=global vertex indices
c
        jdx1=indp(ibnd)+idxve(1,jedge)-1
        jdx2=indp(ibnd)+idxve(2,jedge)-1

c
c ijunc=0:not common edges
c ijunc=1:common edge oriented correctly
c ijunc=-1:common edge oriented incorrectly
c
        ijunc=0
        if(i1.eq.jdx1.and.i2.eq.jdx2) then
            ijunc=1
        elseif(i1.eq.jdx2.and.i2.eq.jdx1) then
            ijunc=-1
        endif
        if(ijunc.ne.0) then
            if(edtype(jedge).eq.-2) then

c
c assign indices creating edge/face array
c
                idxef(inside,numfcp1)=jedge-jstart+1
                if(ijunc.eq.-1) then

c
c reorient edge jedge
c
                    jdx1=idxve(1,jedge)
                    idxve(1,jedge)=idxve(2,jedge)
                    idxve(2,jedge)=jdx1
                endif
            else

c
c assign indices creating edge/face array
c
                idxef(inside,numfcp1)=- (jedge-jstart+1)
            endif

c
c increment count of edges common to jedge with
c count starting at -2
c
                edtype(jedge)=edtype(jedge)+1
                goto 170
            endif
165        continue
c

```



```

c dangling edge
c
      write(6,*) 'error finding edges: ibnd = ',ibnd
      write(6,*) '      numfcpl = ',numfcpl
      write(6,*) '      nside = ',nside
      stop
170    continue
180    continue
190    continue
c
c edtype = 0 - interior
c          -1 - exterior
c          -3 - interior common to more than two faces
c          >0 - exterior connected to edge on another surface
c
c initialize count of edge types
c nedint=# interior edges
c nedext=# exterior edges
c nedjun=# junction edges
c
      nedint=0
      nedext=0
      nedjun=0
      do 230 ibnd=1,nub
        istart=inde(ibnd)
        iend=inde(ibnd)+numedg(ibnd)-1
        do 220 nedgpl=istart,iend
          i1=indp(ibnd)+idxve(1,nedgpl)-1
          i2=indp(ibnd)+idxve(2,nedgpl)-1
c
c edge common to more than 2 faces? edge cannot be common to more
c than two faces
c
          if(edtype(nedgpl).gt.0)then
            edtype(nedgpl)=-3
            write(6,*) '      error: Interior edge is common to more',
&      '      than two faces'
            write(6,*) '      Surface = ',ibnd
            write(6,*) '      Edge = ',nedgpl-istart+1
            write(6,*) '      Point (1) = ',idxve(1,nedgpl)
            write(6,*) '      (2) = ',idxve(2,nedgpl)
            write(6,*) '      X,Y,Z (1) = ',xv(i1),yv(i1),zv(i1)
            write(6,*) '      (2) = ',xv(i2),yv(i2),zv(i2)
            stop
          endif
c
c interior edge?
c
          if(edtype(nedgpl).eq.0)then
c increment interior edge count
c
            nedint=nedint+1
            goto 220
          endif
c
c if edge is an exterior edge it is necessary to determine what type
c of exterior edge
c
          do 215 jband=1,nub
            jstart=inde(jband)
            jend=inde(jband)+numedg(jband)-1
            do 210 jedge=jstart,jend
c
c if edge is interior or edge is same edge as nedgpl no need to check:
c only looking for external edge to external edge relationships

```

```

c          if(edtype(jedge).eq.0)goto 210
c          if(nedgpl.eq.jedge)goto 210
c          j1=indp(jbnd)+idxve(1,jedge)-1
c          j2=indp(jbnd)+idxve(2,jedge)-1
c
c  ijnctyp=0 : not a junction edge
c          =1 : junction edge formed by faces with same orientation
c          =-1 : junction edge formed by faces with opposite orientations
c
c  same orientation means they both have outwardly
c  directed normal vector
c
c          ijnctyp=0
c
c  vertices numbered same direction (error for PO surfaces)
c
c          if(xv(i1).eq.xv(j1).and.yv(i1).eq.yv(j1).and.
&          zv(i1).eq.zv(j1).and.xv(i2).eq.xv(j2).and.
&          yv(i2).eq.yv(j2).and.zv(i2).eq.zv(j2))then
c              ijnctyp=-1
c
c  vertices ordered properly
c
c          elseif(xv(i1).eq.xv(j2).and.yv(i1).eq.yv(j2).and.
&          zv(i1).eq.zv(j2).and.xv(i2).eq.xv(j1).and.
&          yv(i2).eq.yv(j1).and.zv(i2).eq.zv(j1))then
c              ijnctyp=1
c          endif
c          if(ijnctyp.eq.-1)then
&          write(6,*) '      error: Inconsistent triangular face ',
&          'orientation between intersecting surfaces'
&          write(6,*) '      Surfaces ibnd/jbnd = ',ibnd,jbnd
&          stop
c          endif
c          if(ijnctyp.ne.0)then
c              edtype(nedgpl)=jedge
c
c  increment junction edge increment
c
c          nedjun=nedjun+1
c          if(nedgpl.gt.jedge .and. ibnd.eq.jbnd .and.
&          nedgpl.ne.edtype(jedge))then
&          kedge=edtype(jedge)
&          write(6,*) '      error: Surface contains a re-',
&          'entrant junction edge which is multiply defined'
&          write(6,*) '      ibnd = jbnd = ',ibnd
&          write(6,*) '      nedgpl = ',nedgpl-istart+1
&          write(6,*) '      jedge = ',jedge-jstart+1
&          write(6,*) '      kedge = ',kedge-jstart+1
&          stop
c          endif
c          goto 220
c          endif
210      continue
215      continue
c
c  increment exterior edge count
c
c          nedext=nedext+1
220      continue
230      continue
c
c  print geometry information
c
c          write(6,240)nub,npts,numface,numedge

```

```

        write(6,250)nedint,nedext,nedjun
240   format(//,' number of surfaces = ',i3,/,
      & ' number of points = ',i5,/,
      & ' number of faces = ',i5,/,
      & ' number of edges = ',i5)
250   format(' number of interior edges = ',i5,/,
      & ' number of exterior edges = ',i5,/,
      & ' number of junction edges = ',i5,/)
c
c loop over surfaces
c
      do 254 ibnd=1,nub
        istart=indf(ibnd)
        iend=indf(ibnd)+numfac(ibnd)-1
c
c loop over faces
c
        do 253 numfcpl=istart,iend
c
c loop over face sides
c
          do 252 iside=1,3
            il=idxef(inside,numfcpl)
            nedgpl=inde(ibnd)+iabs(il)-1
c
c if interior edge do not evaluate
c
            if(edtype(nedgpl).le.0)goto 252
            jedge=edtype(nedgpl)
            kedge=edtype(jedge)
            if(jedge.lt.nedgpl. and. jedge.lt.kedge)then
              idxef(inside,numfcpl)=-idxef(inside,numfcpl)
            endif
252      continue
253      continue
254      continue
c
c determine number of interior/exterior/junction/reentrant edges by surfaces
c
      do 270 ibnd=1,nub
        nint(ibnd)=0
        next(ibnd)=0
        nje(ibnd)=0
        nren(ibnd)=0
        istart=inde(ibnd)
        iend=inde(ibnd)+numedg(ibnd)-1
        do 260 nedgpl=istart,iend
          if(edtype(nedgpl).eq.0)then
            nint(ibnd)=nint(ibnd)+1
          elseif(edtype(nedgpl).eq.-1)then
            next(ibnd)=next(ibnd)+1
          elseif(edtype(nedgpl).gt.0)then
            nje(ibnd)=nje(ibnd)+1
            if(edtype(nedgpl).ge.istart.and.edtype(nedgpl).le.iend)then
              nren(ibnd)=nren(ibnd)+1
            endif
          endif
260      continue
270      continue
c
c determine number of unknowns and locations
c   nunknon - Total number of unknowns.
c
      nunknon=0
c
c loop over surfaces

```

```

c
c      do 290 ibnd=1,nub
c
c step over boundary edges
c
c      1start=inde(ibnd)
c      iend=inde(ibnd)+numedg(ibnd)-1
c      do 280 nedgpl=1start,iend
c          iedind(nedgpl)=0
c
c increment unknown count
c
c      nunknon=nunknon+1
c      loc=nunknon
c
c assign unknown location
c iedind stores the global unknown number
c of the edge (nedgpl)
c
c      iedind(nedgpl)=loc
c      280 continue
c      290 continue
c
c eliminate duplicate current indices (edges
c common to surfaces)
c
c loop over surfaces
c
c      do 330 ibnd=1,nub
c
c loop over edges
c
c      1start=inde(ibnd)
c      iend=inde(ibnd)+numedg(ibnd)-1
c      do 320 nedgpl=1start,iend
c
c is edge a junction edge?
c if not skip it
c
c      if(edtype(nedgpl).le.0)goto 320
c      jedge=edtype(nedgpl)
c      kedge=edtype(jedge)
c      new=iedind(nedgpl)
c      iold=iedind(jedge)
c
c get rid of duplicate unknown and
c decrement unknown count
c
c      call eleqind(new,iold,nunknon,numedge)
c      320 continue
c      330 continue
c
c assign new unknown count numzel
c
c      numzel=nunknon
c
c count number of edges by surface (nbji=#edges)
c
c      do 350 ibnd=1,nub
c          nbji(ibnd)=0
c          1start=inde(ibnd)
c          iend=inde(ibnd)+numedg(ibnd)-1
c          do 340 nedgpl=1start,iend
c              if(iedind(nedgpl).ne.0)nbji(ibnd)=nbji(ibnd)+1
c          340 continue
c      350 continue

```

```

do 360 idx=1,numzel
  level(idx)=0
360  continue
do 380 ibnd=1,nub
  istart=inde(ibnd)
  iend=inde(ibnd)+numedg(ibnd)-1
do 370 nedgpl=istart,iend
  ind2=iabs(iedind(nedgpl))
  if(ind2.ne.0)then
    if(mompo(ibnd).gt.level(ind2))level(ind2)=mompo(ibnd)
  endif
370  continue
380  continue
c
c count number of unknowns (mm and po)
c
do 390 idx=1,2
  nelcur(idx)=0
390  continue
do 400 idx=1,numzel
  nhy=level(idx)
  if(nhy.gt.0)then
    nelcur(nhy)=nelcur(nhy)+1
  endif
400  continue
c
c reorder unknowns
c
do i=1,2
  do j=1,numzel
    idxuned(j,i)=0
  enddo
enddo
do 410 idx=1,numzel
  iunord(idx)=0
410  continue
indx(1)=0
indx(2)=indx(1)+nelcur(1)
do 430 ibnd=1,nub
  istart=inde(ibnd)
  iend=inde(ibnd)+numedg(ibnd)-1
do 420 nedgpl=istart,iend
  id2=iedind(nedgpl)
  if(id2.ne.0)then
    if(iunord(id2).eq.0)then
      nhy=level(id2)
      indx(nhy)=indx(nhy)+1
      iunord(id2)=indx(nhy)
    endif
  endif
  iedind(nedgpl)=iunord(id2)
c
c idxuned contains edge number corresponding to unknown #
c
  if(idxuned(iedind(nedgpl),1).eq.0)then
    idxuned(iedind(nedgpl),1)=nedgpl
  else
    idxuned(iedind(nedgpl),1)=-idxuned(iedind(nedgpl),1)
    idxuned(iedind(nedgpl),2)=nedgpl
  endif
420  continue
430  continue
c
c idxfe contains faces making edge
c
do ii=1,nub

```

```

    istart=indf(ii)
    iend=indf(ii)+numfac(ii)-1
    ioffset=inde(ii)-1
    do jj=istart,iend
        do kk=1,3
            if(idxef(kk,jj).gt.0)then
                idxfep(idxef(kk,jj)+ioffset)=jj
            elseif(idxef(kk,jj).lt.0)then
                idxfem(-idxef(kk,jj)+ioffset)=jj
            endif
        enddo
    enddo
enddo

```

```

c
c calculate face area and normal unit vector
c

```

```

do ii=1,nub
    istart=indf(ii)
    iend=indf(ii)+numfac(ii)-1
    ioffset=indp(ii)-1
    do numfcpl=istart,iend
        i1=idxvf(1,numfcpl)+ioffset
        i2=idxvf(2,numfcpl)+ioffset
        i3=idxvf(3,numfcpl)+ioffset
        x21=xv(i2)-xv(i1)
        y21=yv(i2)-yv(i1)
        z21=zv(i2)-zv(i1)
        x31=xv(i3)-xv(i1)
        y31=yv(i3)-yv(i1)
        z31=zv(i3)-zv(i1)
        deltax=y21*z31-z21*y31
        deltay=-x21*z31+z21*x31
        deltaz=x21*y31-y21*x31
        delta=sqrt(deltax**2+deltay**2+deltaz**2)
        delinv=1./delta
        area(numfcpl)=.5*delta
        normx(numfcpl)=deltax*delinv
        normy(numfcpl)=deltay*delinv
        normz(numfcpl)=deltaz*delinv
    enddo
enddo

```

```

c
c calculate the unit vectors normal and parallel to edges
c

```

```

do ii=1,nub
    istart=indf(ii)
    iend=indf(ii)+numfac(ii)-1
    ioffset=indp(ii)-1
    iofsete=inde(ii)-1
    do numfcpl=istart,iend
        if1=iabs(idxef(1,numfcpl))+iofsete
        del=edlen(if1)
        delinv=1./del
        i1=idxvf(2,numfcpl)+ioffset
        i2=idxvf(3,numfcpl)+ioffset
        xx=xv(i2)-xv(i1)
        yy=yv(i2)-yv(i1)
        zz=zv(i2)-zv(i1)
        upedx(1,numfcpl)=xx*delinv
        upedy(1,numfcpl)=yy*delinv
        upedz(1,numfcpl)=zz*delinv
        unedx(1,numfcpl)=upedy(1,numfcpl)*normz(numfcpl) -
+         upedz(1,numfcpl)*normy(numfcpl)
        unedy(1,numfcpl)=-upedx(1,numfcpl)*normz(numfcpl) +
+         upedz(1,numfcpl)*normx(numfcpl)
        unedz(1,numfcpl)=upedx(1,numfcpl)*normy(numfcpl) -

```



```

+         upedy(1,numfcp1)*normx(numfcp1)
    if1=iabs(idxef(2,numfcp1))+iofsete
    del=edlen(if1)
    delinv=1./del
    i1=idxvf(3,numfcp1)+ioffset
    i2=idxvf(1,numfcp1)+ioffset
    xx=xv(i2)-xv(i1)
    yy=yv(i2)-yv(i1)
    zz=zv(i2)-zv(i1)
    upedx(2,numfcp1)=xx*delinv
    upedy(2,numfcp1)=yy*delinv
    upedz(2,numfcp1)=zz*delinv
    unedx(2,numfcp1)=upedy(2,numfcp1)*normz(numfcp1) -
+         upedx(2,numfcp1)*normy(numfcp1)
    unedy(2,numfcp1)=-upedx(2,numfcp1)*normz(numfcp1) +
+         upedy(2,numfcp1)*normx(numfcp1)
    unedz(2,numfcp1)=upedx(2,numfcp1)*normy(numfcp1) -
+         upedy(2,numfcp1)*normx(numfcp1)
    if1=iabs(idxef(3,numfcp1))+iofsete
    del=edlen(if1)
    delinv=1./del
    i1=idxvf(1,numfcp1)+ioffset
    i2=idxvf(2,numfcp1)+ioffset
    xx=xv(i2)-xv(i1)
    yy=yv(i2)-yv(i1)
    zz=zv(i2)-zv(i1)
    upedx(3,numfcp1)=xx*delinv
    upedy(3,numfcp1)=yy*delinv
    upedz(3,numfcp1)=zz*delinv
    unedx(3,numfcp1)=upedy(3,numfcp1)*normz(numfcp1) -
+         upedx(3,numfcp1)*normy(numfcp1)
    unedy(3,numfcp1)=-upedx(3,numfcp1)*normz(numfcp1) +
+         upedy(3,numfcp1)*normx(numfcp1)
    unedz(3,numfcp1)=upedx(3,numfcp1)*normy(numfcp1) -
+         upedy(3,numfcp1)*normx(numfcp1)
    enddo
enddo

c
c print out system information
c
write(6,*)
write(6,*) ' Total number of unknowns entered = ',numzel
write(6,*) ' J (total) = ',nunknon
write(6,*)
nlev=0
do 440 idx=1,2
    nlev=nlev+nelcur(idx)
    write(6,*) ' Solution level = ',idx
    write(6,*) ' J (total) = ',nelcur(idx)
440 continue
if(nlev.ne.nunknon)then
    write(6,*) ' error: Incorrect unknown count by level'
    stop
endif

c
c print out surface information
c
do 450 ibnd=1,nub
    write(6,*)
    write(6,*) ' Surface number ',ibnd
    write(6,*) ' Number of vertices (nodes) = ',nup(ibnd)
    write(6,*) ' Number of triangular facets = ',numfac(ibnd)
    write(6,*) ' Number of edges = ',numedg(ibnd)
    write(6,*) ' Number of interior edges = ',nint(ibnd)
    write(6,*) ' Number of exterior edges = ',next(ibnd)
    write(6,*) ' Number of junction edges = ',nje(ibnd)

```

```

write(6,*) '    Number of re-entrant junction edges = ',nren(ibnd)
write(6,*) '    Number of R(+) junction edges = ',nrpls(ibnd)
write(6,*) '    Number of R(-) junction edges = ',nrmin(ibnd)
write(6,*) '    Solution level for exterior = ',momp0(ibnd)

```

c
c
c

check euler formula

```

neuler=nup(ibnd)+numfac(ibnd)-numedg(ibnd)
if(neuler.lt.0.or.neuler.gt.2)then
  write(6,*)
  write(6,*) ' warning: Euler's formula does not hold'
  write(6,*) '    Vertices + Faces - Edges should equal (1,2,0)'
  write(6,*) '                                1 (disk)'
  write(6,*) '                                2 (sphere)'
  write(6,*) '                                0 (cylinder)'
endif
if(numedg(ibnd).ne.nint(ibnd)+next(ibnd)+nje(ibnd))then
  write(6,*)
  write(6,*) ' error: incorrect edge total'
  stop
endif
write(6,*)
write(6,*) '    Geometry information (problem input units):'
numfcpl=indf(ibnd)
write(6,*) '    Centroid of face 1 - x = ',xc(numfcpl)
write(6,*) '                                y = ',yc(numfcpl)
write(6,*) '                                z = ',zc(numfcpl)
write(6,*) '    Unit normal vector to face 1 - normx = ',
&normx(numfcpl)
write(6,*) '                                normy = ',
&normy(numfcpl)
write(6,*) '                                normz = ',
&normz(numfcpl)
write(6,*)
450 continue
return
end

```



```

subroutine grnint(xfkr,yfkr,xcii,ycii,zcii,xvii,yvii,zvii,deii,
1 xcjj,ycjj,zcjj,xvjj,yvjj,zvjj,dej,
2 nxj,nyj,nzj,area,upx,upy,upz,unx,uny,unz,
3 gl1,gl2,gl3,gl4,gal,gaa,gab,gac,gb1,gba,gb2,gb3,gcc)
c
c compute 3d-patch green's function integrals using triangular surface
c quadrature formulas.
c
implicit none
real*4 xvii(3),yvii(3),zvii(3),deii(3)
real*4 xvjj(3),yvjj(3),zvjj(3),dej(3)
real*4 asinh,x,s,a,b,xx,xcii,xcjj,yy,ycii,ycjj,zz,
+zcii,zcjj,apq,bpq,delta,x13,y13,z13,alp,x23,y23,z23,
+bet,gam,r,sing,wi,ai,bi,rbp,yfkr,y,xfkr,ww,b2,b3,b11,
+ba1,baa,bab,bb1,bbb,del1,del2,a11,a21,a12,a22,del,ci
+,xi,yi,zi,d,rhox,rhoy,rhoz,d2,pm2,rm,pp2,rp,p0,term1,
+r02,term2,term3,area,b1,apj,bpj,cpj,xpj,ypj,zpj,
+wj,aj,bj,cj,xj,yj,zj
integer ie,ising,npoint,igaus1,igaus2,jgaus1,jgaus2,
+i,je,j2,j1,j
complex*8 g,gsing,zsing
complex*8 gl1,gl2,gl3,gl4,gal,gaa,gab,gac
complex*8 gb1,gba,gb2,gb3,gcc,gcl,gca,gcb,gcc
real*4 nxj,nyj,nzj,lp,lm
real*4 upx(3),upy(3),upz(3),unx(3),uny(3),unz(3)
include 'intord.inc'
include 'triint.inc'
include 'wavenumber.inc'
include 'const.inc'
complex*8 sumg,sumag,sumbg
asinh(x)=log(x+sqrt(x**2+1.))
s(a,b)=asinh(a+b)+(asinh((a*(a+b)+1.)/b)-asinh(a))*abs(b)/
& sqrt(a**2+1.)
gl1=0.0
gl2=0.0
gl3=0.0
gl4=0.0
gal=0.0
gaa=0.0
gab=0.0
gb1=0.0
gba=0.0
gb2=0.0
gb3=0.0
xx=xcii-xcjj
yy=ycii-ycjj
zz=zcii-zcjj
apq=sqrt(xx**2+yy**2+zz**2)
bpq=apq*fk/(2.*pi)
delta=0.0
do 50 ie=1,3
if(abs(deii(ie)).gt.delta) delta=abs(deii(ie))
if(abs(dej(ie)).gt.delta) delta=abs(dej(ie))
50 continue
ising=0
if(apq.eq.0.0) then
npoint=nself
ising=1
elseif(bpq.le.rnear .or. apq.le.delta) then
npoint=nnear
ising=1
elseif(bpq.lt.rfar) then
npoint=nmid
else
npoint=nfar
endif
igaus1=nmptont(npoint)
igaus2=igaus1+npoint-1

```

```

      jgaus1=igaus1
      jgaus2=igaus2
      if(apq.eq.0.0) then
c
c self term
c
c singular part (analytic)
      x13=xvii(1)-xvii(3)
      y13=yvii(1)-yvii(3)
      z13=zvii(1)-zvii(3)
      alp=sqrt(x13**2+y13**2+z13**2)
      x23=xvii(2)-xvii(3)
      y23=yvii(2)-yvii(3)
      z23=zvii(2)-zvii(3)
      bet=sqrt(x23**2+y23**2+z23**2)
      gam=x13*x23+y13*y23+z13*z23
      r=sqrt((alp*bet)**2-gam**2)
      a=gam/r
      b=alp**2/r
      sing=(s(a-b,b)-s(a,-b))/alp
      g11=sing
      ga1=sing
      gaa=sing
      gab=sing
      gb1=sing
      gbb=sing
c non-singular part (numerical integration)
      do 100 i=igaus1,igaus2
        wi=weight(i)
c region 1
        ai=alfa(i)-1.
        bi=beta(i)
        xx=ai*x13+bi*x23
        yy=ai*y13+bi*y23
        zz=ai*z13+bi*z23
        apq=xx**2+yy**2+zz**2
        rpq=sqrt(apq)
        x=yfkr*rpq
        y=-xfkr*rpq
        ww=wi/rpq
        g=(ww*exp(x))*cmlpx(cos(y),sin(y))
        sing=ww
        b2=(1.+ai)**2
        b3=b2*(1.+ai)
        b11=b2
        ba1=b2*(1.-0.5*ai)
        baa=b3*(1.-ai)
        bab=b2*(1.+2.*bi-4.*ai*bi-ai**2)
        bb1=b2*(1.+ai+1.5*bi)
        bbb=b3*(1.+ai+2.*bi)
        g11=(g*b11-sing)+g11
        ga1=(g*ba1-sing)+ga1
        gaa=(g*baa-sing)+gaa
        gab=(g*bab-sing)+gab
        gb1=(g*bb1-sing)+gb1
        gbb=(g*bbb-sing)+gbb
c region 2
        ai=-alfa(i)
        bi=1.-beta(i)
        xx=ai*x13+bi*x23
        yy=ai*y13+bi*y23
        zz=ai*z13+bi*z23
        apq=xx**2+yy**2+zz**2
        rpq=sqrt(apq)
        x=yfkr*rpq
        y=-xfkr*rpq

```

```

ww=wi/rpq
g=(ww*exp(x))*cplx(cos(y),sin(y))
sing=ww
b2=(1.-bi)**2
b3=b2*(1.-bi)
b11=b2
ba1=b2*(1.-bi-1.5*ai)
baa=b3*(1.-2.*ai-bi)
bab=b2*(1.-2.*ai-4.*ai*bi-bi**2)
bb1=b2*(1.+0.5*bi)
bbb=b3*(1.+bi)
g11=(g*b11-sing)+g11
ga1=(g*ba1-sing)+ga1
gaa=(g*baa-sing)+gaa
gab=(g*bab-sing)+gab
gb1=(g*bb1-sing)+gb1
gbb=(g*bbb-sing)+gbb
c    region 3
ai=alfa(i)
bi=beta(i)
xx=ai*x13+bi*x23
yy=ai*y13+bi*y23
zz=ai*z13+bi*z23
apq=xx**2+yy**2+zz**2
rpq=sqrt(apq)
x=yfkr*rpq
y=-xfkr*rpq
ww=wi/rpq
g=(ww*exp(x))*cplx(cos(y),sin(y))
sing=ww
b2=(1.-ai-bi)**2
b3=b2*(1.-ai-bi)
b11=b2
ba1=b2*(1.+0.5*ai-bi)
baa=b3*(1.+ai-bi)
bab=b3*(1.+ai+bi)
bb1=b2*(1.-ai+0.5*bi)
bbb=b3*(1.-ai+bi)
g11=(g*b11-sing)+g11
ga1=(g*ba1-sing)+ga1
gaa=(g*baa-sing)+gaa
gab=(g*bab-sing)+gab
gb1=(g*bb1-sing)+gb1
gbb=(g*bbb-sing)+gbb
100    continue
ga1=ga1/3.
gaa=gaa/6.
gab=gab/12.
gb1=gb1/3.
gbb=gbb/6.
g1a=ga1
g1b=gb1
gba=gab
elseif(ising.ne.0) then
c
c    near-self term
c
c    the following variables are used in the singularity extraction.
del1=abs(dejj(1))
del2=abs(dejj(2))
x13=xvjj(1)-xvjj(3)
y13=yvjj(1)-yvjj(3)
z13=zvjj(1)-zvjj(3)
x23=xvjj(2)-xvjj(3)
y23=yvjj(2)-yvjj(3)
z23=zvjj(2)-zvjj(3)

```

```

c      cramer's rule coefficients
      a11=del2**2
      a21=x23*x13+y23*y13+z23*z13
      a12=a21
      a22=del1**2
      del=a11*a22-a21*a12
      a11=a11/del
      a21=a21/del
      a12=a21
      a22=a22/del
c      test integration loop
      do 200 i=igaus1,igaus2
        wi=weight(i)
        ai=alfa(i)
        bi=beta(i)
        ci=1.-ai-bi
        xi=ai*xvii(1)+bi*xvii(2)+ci*xvii(3)
        yi=ai*yvii(1)+bi*yvii(2)+ci*yvii(3)
        zi=ai*zvii(1)+bi*zvii(2)+ci*zvii(3)
c      distance (+ or -) of test point from source plane
        d=nxj*(xi-xvj(1))+nyj*(yi-yvj(1))+nzj*(zi-zvj(1))
c      projection of test point onto source plane
        rhox=xi-d*nxj
        rhoy=yi-d*nyj
        rhoz=zi-d*nzj
c      singular part
        sing=0.0
        d=abs(d)
        d2=d**2
c      compute l+, r+, l-, r-, and p0 for each side of the source triangle
      do 120 je=1,3
        if(je.eq.1) then
          j2=3
          j1=2
        elseif(je.eq.2) then
          j2=1
          j1=3
        else
          j2=2
          j1=1
        endif
        xx=xvj(j1)-rhox
        yy=yvj(j1)-rhoy
        zz=zvj(j1)-rhoz
        lm=xx*upx(je)+yy*upy(je)+zz*upz(je)
        pm2=xx**2+yy**2+zz**2
        rm=sqrt(pm2+d2)
        xx=xvj(j2)-rhox
        yy=yvj(j2)-rhoy
        zz=zvj(j2)-rhoz
        lp=xx*upx(je)+yy*upy(je)+zz*upz(je)
        pp2=xx**2+yy**2+zz**2
        rp=sqrt(pp2+d2)
        p0=xx*unx(je)+yy*uny(je)+zz*unz(je)
        if(p0.ne.0.0) then
          if(lm.ge.0.0) then
            term1=log((rp+lp)/(rm+lm))
          else
            term1=log((rm-lm)/(rp-lp))
          endif
          sing=p0*term1+sing
        endif
      if(d.ne.0.0) then
        r02=p0**2+d2
        term2=atan(p0*lp/(r02+d*rp))
        term3=atan(p0*lm/(r02+d*rm))
      endif
    
```

```

        sing=-d*(term2-term3)+sing
    endif
120    continue
        sing=sing/(2.*areaaj)
c    compute alpha and beta (apj,bpj) for point rho in the source plane
        xx=rhox-xvj(3)
        yy=rhoy-yvj(3)
        zz=rhoz-zvj(3)
c    cramer's rule
        b1=x13*xx+y13*yy+z13*zz
        b2=x23*xx+y23*yy+z23*zz
        apj=b1*a22-b2*a12
        bpj=a11*b2-a21*b1
c    find nearest point on the source triangle (xpj,ypj,zpj)
        cpj=1.-apj-bpj
        if(apj.lt.0.0) then
            apj=0.0
            if(bpj.lt.0.0) then
                bpj=0.0
            elseif(bpj.gt.1.0) then
                bpj=1.0
            endif
        elseif(bpj.lt.0.0) then
            bpj=0.0
            if(apj.lt.0.0) then
                apj=0.0
            elseif(apj.gt.1.0) then
                apj=1.0
            endif
        elseif(cpj.lt.0.0) then
            if(bpj.ge.apj+1.0) then
                apj=0.0
                bpj=1.0
            elseif(bpj.le.apj-1.0) then
                apj=1.0
                bpj=0.0
            else
                apj=0.5*(1.0+apj-bpj)
                bpj=1.0-apj
            endif
        endif
        cpj=1.-apj-bpj
        xpj=apj*xvj(1)+bpj*xvj(2)+cpj*xvj(3)
        ypj=apj*yvj(1)+bpj*yvj(2)+cpj*yvj(3)
        zpj=apj*zvj(1)+bpj*zvj(2)+cpj*zvj(3)
        xx=xi-xpj
        yy=yi-ypj
        zz=zi-zpj
        rpq=sqrt(xx**2+yy**2+zz**2)
        x=yfkr*rpq
        y=-xfkr*rpq
        zsing=exp(x)*cmplx(cos(y),sin(y))
        g=(wi*sing)*zsing
        g11=g+g11
        gla=apj*g+g1a
        glb=bpj*g+g1b
        gal=ai*g+gal
        gaa=(ai*apj)*g+gaa
        gab=(ai*bpj)*g+gab
        gbl=bi*g+gbl
        gba=(bi*apj)*g+gba
        gbb=(bi*bpj)*g+gbb
c    non-singular part
        sumg=0.0
        sumag=0.0
        sumbg=0.0

```

```

c   source integration loop
      do 190 j=jgaus1,jgaus2
        wj=weight(j)
        aj=alfa(j)
        bj=beta(j)
        cj=1.-aj-bj
        xj=aj*xvjj(1)+bj*xvjj(2)+cj*xvjj(3)
        yj=aj*yvjj(1)+bj*yvjj(2)+cj*yvjj(3)
        zj=aj*zvjj(1)+bj*zvjj(2)+cj*zvjj(3)
        xx=xi-xj
        yy=yi-yj
        zz=zi-zj
        apq=xx**2+yy**2+zz**2
        rpq=sqrt(apq)
c real and imaginary part of -jkr
        x=yfkr*rpq
        y=-xfkr*rpq
        ww=wj/rpq
        g=(ww*exp(x))*cplx(cos(y),sin(y))
        gsing=ww*zsing
c the last 3 statements assume that rpq is not zero. the following 3
c statements are required if rpq=0 for self-term integrations.
c...      ww=wj
c...      g=ww*cplx(yfkr,-xfkr)
c...      gsing=0.0
        sumg=(g-gsing)+sumg
        sumag=(aj*g-apj*gsing)+sumag
        sumbg=(bj*g-bpj*gsing)+sumbg
190      continue
        sumg=wi*sumg
        sumag=wi*sumag
        sumbg=wi*sumbg
        g11=sumg+g11
        ga1=ai*sumg+ga1
        gb1=bi*sumg+gb1
        gla=sumag+gla
        gaa=ai*sumag+gaa
        gba=bi*sumag+gba
        glb=sumbg+glb
        gab=ai*sumbg+gab
        gbb=bi*sumbg+gbb
200      continue
      else
c
c non-self term
c
c   test integration loop
      do 300 i=igaus1,igaus2
        wi=weight(i)
        ai=alfa(i)
        bi=beta(i)
        ci=1.-ai-bi
        xi=ai*xvii(1)+bi*xvii(2)+ci*xvii(3)
        yi=ai*yvii(1)+bi*yvii(2)+ci*yvii(3)
        zi=ai*zvii(1)+bi*zvii(2)+ci*zvii(3)
        sumg=0.0
        sumag=0.0
        sumbg=0.0
c   source integration loop
      do 290 j=jgaus1,jgaus2
        wj=weight(j)
        aj=alfa(j)
        bj=beta(j)
        cj=1.-aj-bj
        xj=aj*xvjj(1)+bj*xvjj(2)+cj*xvjj(3)
        yj=aj*yvjj(1)+bj*yvjj(2)+cj*yvjj(3)

```

```

        zj=aj*zvjj(1)+bj*zvjj(2)+cj*zvjj(3)
        xx=xi-xj
        yy=yi-yj
        zz=zi-zj
        apq=xx**2+yy**2+zz**2
        rpq=sqrt(apq)
c real and imaginary part of -jkr
        x=yfkr*rpq
        y=-xfkr*rpq
        g=(wj*exp(x)/rpq)*cmplx(cos(y),sin(y))
        sumg=g+sumg
        sumag=aj*g+sumag
        sumbg=bj*g+sumbg
290      continue
        sumg=wi*sumg
        sumag=wi*sumag
        sumbg=wi*sumbg
        g11=sumg+g11
        ga1=ai*sumg+ga1
        gb1=bi*sumg+gb1
        gla=sumag+gla
        gaa=ai*sumag+gaa
        gba=bi*sumag+gba
        glb=sumbg+glb
        gab=ai*sumbg+gab
        gbb=bi*sumbg+gbb
300      continue
endif
        glc=g11-gla-glb
        gac=ga1-gaa-gab
        gbc=gb1-gba-gbb
        gc1=g11-ga1-gb1
        gca=gla-gaa-gba
        gcb=glb-gab-gbb
        gcc=g11+gaa+gbb+gab+gba-gla-ga1-glb-gb1
        return
end

```

```

      program hybrid_simple
c
c  efie hybrid code (mm/po)
c
      implicit none
      include 'arraydim.inc'
      include 'work.inc'
      include 'const.inc'
      include 'wavenumber.inc'
      include 'order.inc'
      include 'threat.inc'
      include 'numbers.inc'
      include 'surcord.inc'
      include 'scatter.inc'
      include 'unit.inc'
      include 'tilde.inc'
      include 'intord.inc'
      include 'dbsm.inc'
      include 'patchtol.inc'
      include 'numj.inc'
      include 'polar.inc'
      include 'problem.inc'
      character*6 geom
      logical freq
      real*4 lamdl, lamdu, speedl, cvrn, freql, frequ, fk1, fk2,
+temp1, temp2, ang, angs, ange, anginc, winc, finc, fr
      integer nfreq, ncutfar, nangs, idxc, icut, i, nmm, npo, nwt,
+ntotw, idxfr
c
c  set option array: namelist used to set options in
c  input file
c
      namelist /option/monbi, phi,
+thet, units, freq, nfreq, ncutfar,
+dbsm, nself, nnear, rnear, nmid,
+rfar, nfar, pattol, polt, polp, rtilde
c
c  calculate constants pi, speed of light, and free space impedance
c
      pi=acos(-1.)
      speedl=2.997925E8
      eta=4.e-7*pi*speedl
c
c  read the problem's title
c
      read(5,10)title
10    format(a)
      write(6,15)title
15    format(//,15x,' SIMPLE HYBRID CODE',//,
+10x,'problem: ',a80,//)
c
c  set default options (see namelist)
c
c
c  monbi=1 (monostatic scattering)
c  monbi=0 (bistatic scattering)
c
      monbi=1
c
c  thet=bistatic incident theta angle (degrees):monbi=0
c  phi=bistatic incident phi angle (degrees):monbi=0
c
      thet=0.
      phi=0.
c
c  dbsm=false (write rcs in sigma/lambda**2)

```



```

c dbsm=true (write rcs in sigma/meter**2)
c
c     dbsm=.false.
c
c pattol=vertex point tolerance (collapse vertices separated by < pattol)
c
c     pattol=0.
c
c number of integration points for self, near, mid, and far terms
c
c     nself=7
c     nnear=7
c     nmid=3
c     nfar=3
c
c define near and far regions in wavelengths for integrations
c
c     rnear=.25
c     rfar=1.
c
c rtilde=true (rtilde is calculated analytically)
c rtilde=false (rtilde is computed numerically)
c
c     rtilde=.true.
c
c polt and polp are used to select polarizations analyzed
c =true calculate the polarization, =false do not calculate
c
c     polt=.true.
c     polp=.true.
c
c units=units used for the problem
c m=meters, cm=centimeters, in=inches, and ft=feet
c
c     units='m'
c
c freq=false (wavenumber input)
c freq=true (frequency input in GHz)
c
c     freq=.false.
c
c nfreq=number of frequencies to sweep over
c
c     nfreq=1
c
c ncutfar=number of farfield pattern cuts
c
c     ncutfar=1
c
c set options with namelist
c
c     read(5,nml=option)
c     write(6,*)
c     write(6,*) 'options selected:'
c     write(6,*)
c
c print out and check validity of options selected
c
c     if(monbi.eq.1)then
c         write(6,*) ' monostatic scattering'
c     elseif(monbi.eq.0)then
c         write(6,*) ' bistatic scattering'
c         write(6,*) ' phi = ',phi,' thet = ',thet
c     else
c         write(6,*) 'error: no scattering type selected'
c         write(6,*) 'monbi must equal 0 or 1'

```

```

    stop
endif
if(monbi.ge.0.and..not.polt)then
    write(6,*) ' phi polarization selected'
endif
if(monbi.ge.0.and..not.polp)then
    write(6,*) ' theta polarization selected'
endif
if(polt.and.polp)then
    write(6,*) ' theta and phi polarizations selected'
endif
if(.not.polt.and..not.polp)then
    write(6,*) ' error: no polarization selected'
    stop
endif

```

```

c
c calculate conversion factor for geometry files
c
c
c cvrn is conversion factor used to convert various units to meters
c

```

```

    if(units.eq.'m')then
        units='meters'
        cvrn=1.
    elseif(units.eq.'cm')then
        units='cm'
        cvrn=.01
    elseif(units.eq.'in')then
        units='inches'
        cvrn=2.54*.01
    elseif(units.eq.'ft')then
        units='feet'
        cvrn=12.*2.54*.01
    else
        write(6,*) ' error: units = ',units
        stop
    endif
    write(6,*) ' units are in ',units
    if(freq)then
        write(6,*) ' input is in frequency '
    else
        write(6,*) ' input is in wavenumber'
    endif
    if(.not.dbm)then
        write(6,*) ' rcs in sigma/lambda**2'
    else
        write(6,*) ' rcs in sigma/meter**2'
    endif
    if(nfreq.gt.1)then
        write(6,*) ' number frequencies: nfreq = ',nfreq
    endif
    write(6,*) ' number of pattern cuts = ',ncutfar
    write(6,*) ' collapsing vertice tolerance = ',pattol
    write(6,*)
    write(6,*) ' integration points and region definitions:'
    write(6,*) ' nself = ',nself
    write(6,*) ' nnear = ',nnear
    write(6,*) ' nmid = ',nmid
    write(6,*) ' nfar = ',nfar
    write(6,*) ' rnear = ',rnear,' lambda'
    write(6,*) ' rfar = ',rfar,' lambda'
    write(6,*)
    if(rtilde)then
        write(6,*) ' r tilde is calculated analytically'
    endif

```

```

c read in frequency or wavenumber input
c
c freq1 is starting frequency
c fk1 is starting wavenumber
c frequ is the ending frequency
c fk2 is the ending wavenumber
c
    if(nfreq.eq.1)then
        if(freq)then
            read(5,*)freq1
            frequ=freq1
        else
            read(5,*) fk1
            fk2=fk1
        endif
    else
        if(freq)then
            read(5,*) freq1,frequ
        else
            read(5,*) fk1,fk2
        endif
    endif
c
c convert frequency data to wavenumbers or vice versa
c
c frequency calculation gives frequency in GHz
c
    temp1=speed1*1.e-9
    temp2=2.*pi
    if(freq)then
        lamdl=temp1/(freq1*cvrn)
        fk1=temp2/lamdl
        lamdu=temp1/(frequ*cvrn)
        fk2=temp2/lamdu
    else
        lamdl=temp2/fk1
        freq1=temp1/(lamdl*cvrn)
        lamdu=temp2/fk2
        frequ=temp1/(lamdu*cvrn)
    endif
c
c print out frequency information
c
    write(6,*)
    write(6,*)'      frequency selections:'
    write(6,25)fk1,fk2,units,freq1,frequ,lamdl,lamdu,units,
& nfreq
25  format(//,' fk = ',g12.4,' - ',g12.4,1x,a5,'**-1'/,
& ' freq1 = ',f9.3,' - ',f9.3,' GHz',/,
& ' lamdl = ',g12.4,' - ',g12.4,1x,a6,/,
& ' nfreq = ',i3,/)
    fk=max(fk1,fk2)
c
c read geometry files
c
30  continue
c
c geom selects geometry type (patch, wire, bor, bot, etc)
c here code has been limited to patch
c geom='patch' (3d patches are read in)
c geom='end' (end of geometry file inputs)
c
    read(5,10) geom
    if(geom.eq.'end')then
        goto 40
    elseif(geom.eq.'patch')then

```

```

c
c read geometry
c
      call geoin(work1,work2)
      else
        write(6,*) ' error: invalid geometry type, geom = ',geom
        stop
      endif
      goto 30
40  continue
c
c read threat parameters and scattered field options
c
c nang is total numkber of angles analyzed
c
      nang=0
      do 50 idxc=1,ncutfar
c
c nang=# scattering angles for far field cut idxc
c
        read(5,*) nang
        write(6,*)
        write(6,*) 'number of scattering angles = ',nang
        write(6,*)
        write(6,*) ' monbi = ',monbi
c
c write bistatic angle of incidence
c
      if(monbi.eq.0)then
        write(6,*) ' phi and theta incident = ',phi,thet
      endif
c
c ang=fixed angle
c angs=starting angle
c ange=ending angle
c icut=1 (phi cut)
c icut=2 (theta cut)
c
      read(5,*) ang,icut,angs,ange
      write(6,*) ' icut = ',icut
      if(icut.eq.1)then
        write(6,*) ' phi = ',ang
        write(6,*) ' theta = ',angs,' - ',ange
      elseif(icut.eq.2)then
        write(6,*) ' theta = ',ang
        write(6,*) ' phi = ',angs,' - ',ange
      else
        write(6,*) ' error: icut must equal 1 or 2'
        stop
      endif
c
c scattering angles determined
c
      if(nang.gt.1)then
c
c calculate angular increment (anginc)
c
        anginc=(ange-angs)/(nang-1)
      else
        nang=1
        anginc=0.
      endif
c
c calculate scattering angles (phis,thes)
c
      do 45 i=1,nang

```

```

C
C increment scattering angle count
C
      nangs=nangs+1
      if(i.eq.1)then
        iangle(nangs)=-icut
      else
        iangle(nangs)=icut
      endif
      if(icut.eq.1)then
C
C phi cut
C
        phis(nangs)=ang
        thes(nangs)=angs+anginc*(i-1)
      else
C
C theta cut
C
        phis(nangs)=angs+anginc*(i-1)
        thes(nangs)=ang
      endif
45    continue
C
C check angular dimensions
C
      if(nangs.gt.namax)then
        write(6,*) ' insufficient dimension: namax = ',namax
        write(6,*) '                                nangs = ',nangs
        stop
      endif
50    continue
C
C check array sizes
C
      if(numzel.ne.0)then
C
C nmm=#mm unknowns
C npo=#po unknowns
C
        nmm=nelcur(1)
        npo=nelcur(2)
      endif
C
C nwt=# impedance elements required, symmetric mm (upper
C triangle and diagonal stored) + mm/po coupling matrix
C
      nwt=nmm*(nmm+1)/2+nmm*npo
C
C ntotw=total unknowns (mm unknowns + po unknowns)
C
      ntotw=nmm+npo
      write(6,*) ' Total number of Z array elements needed = ',nwt
      write(6,*) ' Total number of work array elements needed = ',ntotw
      if(nwt.gt.nzmax)then
        write(6,*) ' insufficient dimension : nzmax = ',nzmax
        write(6,*) '                                nwt = ',nwt
        stop
      endif
      if(ntotw.gt.nwork)then
        write(6,*) ' insufficient dimension : nwork = ',nwork
        write(6,*) '                                ntotw = ',ntotw
        stop
      endif
      ntotw=numzel
C

```

```

c calculate wave number (winc)
c and frequency (finc) increments
c
    if(ntotw.gt.numax)then
        write(6,*) ' insufficient dimension : numax = ',numax
        write(6,*) '                               ntotw = ',ntotw
        stop
    endif
    if(nfreq.eq.1)then
        winc=0.0
        finc=0.0
    else
        winc=(fk2-fk1)/(nfreq-1)
        finc=(frequ-frequ1)/(nfreq-1)
    endif
    write(6,60)title,fk1,frequ1,temp2/fk
60    format(1h1,'problem: ',a80,/,/, ' fk1 = ',g12.4,/, ' frequ1 = ',f9.3,/,
    & ' lambda = ',g12.4,/)
c
c determine gaussian abscissas and weights
c
    call gauss
c
c loop over # frequencies
c
    do 70 idxfr=1,nfreq
c
c calculate wavenumber and frequency (fk,fr)
c
        fk=fk1+(idxfr-1)*winc
        fr=frequ1+(idxfr-1)*finc
        write(6,*)
        write(6,*) ' Frequency number = ',idxfr
        write(6,*) '    fk = ',fk, '    freq = ',fr
c
c calculate scattered far fields
c
        if(nmm.ne.0.and.npo.ne.0)then
            call solhyb
        elseif(nmm.ne.0.and.npo.eq.0)then
            call solmom
        elseif(nmm.eq.0.and.npo.ne.0)then
            call solpo
        else
            print *, 'unknowns do not define problem that can
            & be handled by code'
            stop
        endif
c
c print out rcs based on far fields
c
        call fldrcs(idxfr,nfreq,fr,freq)
70    continue
    stop
end

```

```

      subroutine ipatch(th,ph,rbt,rbp,
1 nface,nedge,indvf,indf,xc,yc,zc,xv,yv,zv,de)
c
c compute 3d-patch physical optics current coefficients for theta and
c phi polarization.
c
      implicit none
      complex*8 rbt(*),rbp(*)
      integer indvf(3,*),indf(3,*),nedge,iface,nface,ie,i1,
+iv,i2,i1
      real*4 xc(*),yc(*),zc(*),
1 xv(*),yv(*),zv(*),de(*),dtor,th,cth,sth,cph,ph,sph,fkx,
2fky,fkz,xx1,yy1,zz1,xx2,yy2,zz2,del,xx,yy,zz,dot,arg
      complex*8 a1,rt1,rp1
      real*4 xvi(3),yvi(3),zvi(3),dei(3)
      real*4 upx(3),upy(3),upz(3)
      real*4 nx,ny,nz
      integer iedge(3)
      include 'wavenumber.inc'
      include 'const.inc'
      dtor=pi/180.
      call vzero(nedge,rbt)
      call vzero(nedge,rbp)
      if(th.eq.90.0) then
         cth=0.0
         sth=1.0
      else
         cth=cos(th*dtor)
         sth=sin(th*dtor)
      endif
      cph=cos(ph*dtor)
      sph=sin(ph*dtor)
      fkx=fk*sth*cph
      fky=fk*sth*sph
      fkz=fk*cth
c step over each 3d-patch face.
      do 200 iface=1,nface
         do 5 ie=1,3
            i1=indf(ie,iface)
            if(i1.gt.0) then
               dei(ie)=de(i1)
            else
               dei(ie)=-de(-i1)
            endif
            iedge(ie)=iabs(i1)
5          continue
            do 10 iv=1,3
               i1=indvf(iv,iface)
               xvi(iv)=xv(i1)
               yvi(iv)=yv(i1)
               zvi(iv)=zv(i1)
10          continue
c unit normal vector to triangular face
            xx1=xvi(2)-xvi(1)
            yy1=yvi(2)-yvi(1)
            zz1=zvi(2)-zvi(1)
            xx2=xvi(3)-xvi(1)
            yy2=yvi(3)-yvi(1)
            zz2=zvi(3)-zvi(1)
            nx=yy1*zz2-zz1*yy2
            ny=-xx1*zz2+zz1*xx2
            nz=xx1*yy2-yy1*xx2
            del=sqrt(nx**2+ny**2+nz**2)
            nx=nx/del
            ny=ny/del
            nz=nz/del

```



```

c compute unit vectors parallel to each side of the triangle
c note, the sign for the +- facet is also included
  do 20 ie=1,3
    if(ie.eq.1) then
      i2=3
      i1=2
    elseif(ie.eq.2) then
      i2=1
      i1=3
    else
      i2=2
      i1=1
    endif
    xx=xvi(i2)-xvi(i1)
    yy=yvi(i2)-yvi(i1)
    zz=zvi(i2)-zvi(i1)
c +- sign included here
    del=dei(ie)
    upx(ie)=xx/del
    upy(ie)=yy/del
    upz(ie)=zz/del
  20 continue
c determine if the facet is shadowed.
  dot=fkx*nx+fky*ny+fkz*nz
  if(dot.lt.0.0) go to 200
c step over the 3 edges which bound face i.
  do 100 ie=1,3
c skip the minus facets.
    if(dei(ie).lt.0.0) go to 100
    l1=iedge(ie)
    if(ie.eq.1) then
      i2=3
      i1=2
    elseif(ie.eq.2) then
      i2=1
      i1=3
    else
      i2=2
      i1=1
    endif
c po current coefficients computed.
    xx=0.5*(xvi(i1)+xvi(i2))
    yy=0.5*(yvi(i1)+yvi(i2))
    zz=0.5*(zvi(i1)+zvi(i2))
    arg=fkx*xx+fky*yy+fkz*zz
    a1=(2.0/eta)*cmlpx(cos(arg),sin(arg))
    xx=-upx(ie)
    yy=-upy(ie)
    zz=-upz(ie)
c theta polarized h-field.
    rtl=(cth*(xx*cph+yy*sph)-zz*sth)*a1
    rbt(l1)=rtl
c phi polarized h-field.
    rpl=(-xx*sph+yy*cph)*a1
    rbp(l1)=rpl
  100 continue
  200 continue
  return
end

```



```

      subroutine lop(i,j,z,lz)
c
c call l operator calculation for different situations
c only patch-patch is included
c
      implicit none
      include 'arraydim.inc'
      include 'surcord.inc'
      include 'faces.inc'
      include 'indx1.inc'
      include 'edges.inc'
      include 'indx2.inc'
      integer iv,i,if,ie,nfaci,nedgi,indvrt,j,indfac,indedg,
+nfacj,nedgj,ism,lz
      real*4 z
      iv=indp(i)
      if=indf(i)
      ie=inde(i)
      nfaci=numfac(i)
      nedgi=numedg(i)
      indvrt=indp(j)
      indfac=indf(j)
      indedg=inde(j)
      nfacj=numfac(j)
      nedgj=numedg(j)
      ism=0
      if(i.eq.j)ism=1
      call patpatl(z,lz,ism,1.,nfaci,nedgi,idxvf(1,if),
+idxef(1,if),xc(if),yc(if),zc(if),xv(iv),yv(iv),
+zv(iv),edlen(ie),nfacj,nedgj,idxvf(1,indfac),idxef(1,ind
+fac),xc(indfac),yc(indfac),zc(indfac),xv(indvrt),
+yv(indvrt),zv(indvrt),edlen(indedg),normx(indfac)
+,normy(indfac),normz(indfac),area(indfac)
+,upedx(1,indfac),upedy(1,indfac),upedz(1,indfac)
+,unedx(1,indfac),unedy(1,indfac),unedz(1,indfac))
      return
      end

```

```

      subroutine mdacin(nunit,numpat,nface,nedge,xvert
&                      ,yvert,zvert,ifacvrt,iedvrt)
c
c read vertices and faces or edges (mdac format)
c
c
c iunit=device number of geometry file
c numpat=number of vertices
c xvert,yvert,zvert=vertice coordinates
c ifacvrt=indices of vertices forming faces
c iedvrt=indices of vertices forming edges
c nface=number of faces
c nedge=number of edges
c
      implicit none
      integer ifacvrt(3,*),iedvrt(2,*)
      integer nunit,numpat,k,nface,nedge,idxfac,j,idxed
      real xvert(*),yvert(*),zvert(*)
c
c read number of patches from external mdac carlos file.
c
      if(nunit.ne.5)read(nunit,*)numpat
c
c read vertices
c
      read(nunit,*)(xvert(k),k=1,numpat)
      read(nunit,*)(yvert(k),k=1,numpat)
      read(nunit,*)(zvert(k),k=1,numpat)
c
c read face/edge information
c
      read(nunit,*)nface,nedge
      if(nface.ne.0)then
c
c read face vertices
c
        do 100 idxfac=1,nface
          if(nunit.ne.5)then
            read(nunit,*)(ifacvrt(j,idxfac),j=1,3)
          else
            read(nunit,*)(ifacvrt(j,idxfac),j=1,3)
          endif
c
c vertex indices must be between 1 and numpat.
c
          if(ifacvrt(1,idxfac).lt.1.or.ifacvrt(1,idxfac).gt.numpat.or.
&          ifacvrt(2,idxfac).lt.1.or.ifacvrt(2,idxfac).gt.numpat.or.
&          ifacvrt(3,idxfac).lt.1.or.ifacvrt(3,idxfac).gt.numpat)then
            write(6,*)' error: Face vertex index is out of range.'
            stop
          endif
100      continue
        else
c
c edges read
c
          do 110 idxed=1,nedge
            read(nunit,*)(iedvrt(j,idxed),j=1,2)
            if(iedvrt(1,idxed).lt.1.or.iedvrt(1,idxed).gt.numpat.or.
&            iedvrt(2,idxed).lt.1.or.iedvrt(2,idxed).gt.numpat)then
              write(6,*)' error: Edge vertex index is out of range.'
              stop
            endif
110      continue
          endif
          return

```

end

```

      subroutine patpat1(z,lz,ism,c,
1 nfi,nei,indvfi,indefi,xci,yci,zci,xvi,yvi,zvi,dei,
2 nfj,nej,indvfj,indefj,xcj,ycj,zcj,xvj,yvj,zvj,dej
3,nrmx,nrmy,nrmz,areea,upey,upez,unex,uney,unez)
c
c patch-patch 1 operater calculation
c
c isym=0 : entire matrix computed and stored by columns
c isym.ne.0 : upper triangular part calculated and stored
c in packed form by columns
c
      implicit none
      complex*8 z(*)
      dimension xci(*),yci(*),zci(*),indvfi(3,*),xvi(*),yvi(*),zvi(*),
1 indefi(3,*),dei(*)
      dimension xcj(*),ycj(*),zcj(*),indvfj(3,*),xvj(*),yvj(*),zvj(*),
1 indefj(3,*),dej(*)
      real nrmx(*),nrmy(*),nrmz(*),areea(*)
      real upex(3,*),upey(3,*),upez(3,*)
      real unex(3,*),uney(3,*),unez(3,*)
      complex*8 zpp
      complex*8 c1,c2,ctemp
      real*4 xvii(3),yvii(3),zvii(3),deii(3)
      real*4 xvjii(3),yvji(3),zvji(3),deji(3)
      real*4 dot(3,3)
      real*4 nxj,nyj,nzj
      real*4 upx(3),upy(3),upz(3),unx(3),uny(3),unz(3)
      real*4 c,dej,xcjj,xcj,ycjj,ycj,zcjj,zcj,xvj,yvj,zvj,
+areaj,dei,xcii,xcj,ycii,ycj,zcii,zci,xvi,yvi,zvi,
+elj,eli
      complex*8 g11,g1a,g1b,g1c,g1d,g1e,g1f,g1g,g1h,g1i
      complex*8 g21,g2a,g2b,g2c,g2d,g2e,g2f,g2g,g2h,g2i
      complex*8 zface,zedgei(3),zedgej(3),zedge(3,3)
      integer jface,nfj,je,j1,indefj,numb,jv,indvfj,ism
+ ,istop,nfi,iface,ie,il,indefi,iv,indvfi,j,i,jt
+ ,it,lz,nei,nej
      integer iedge(3),jedge(3)
      include 'block.inc'
      include 'const.inc'
      include 'wavenumber.inc'
      c1=c*(0.,1.)*eta*fk/(4.*pi)
      c2=c*(0.,1.)*eta/fk/pi
c
c loop over faces
c
c source loop
c
      do 500 jface=1,nfj
        do 5 je=1,3
          j1=indefj(je,jface)
          if(j1.gt.0) then
            dejj(je)=dej(j1)
          else
            dejj(je)=-dej(-j1)
          endif
          jedge(je)=iabs(j1)
5        continue
        if(part) then
          call blockj(3,jedge,numb)
          if(numb.eq.0) go to 500
        endif
        xcjj=xcj(jface)
        ycjj=ycj(jface)
        zcjj=zcj(jface)
        do 10 jv=1,3
          j1=indvfj(jv,jface)

```

```

        xvjj(jv)=xvj(j1)
        yvjj(jv)=yvj(j1)
        zvjj(jv)=zvj(j1)
10    continue
c    normal vector to source triangle
        nxj=nrnx(jface)
        nyj=nrny(jface)
        nzj=nrnz(jface)
c    area of source triangle
        areaj=areea(jface)
c    unit vectors parallel and normal to each side of source triangle
        do 110 je=1,3
            upx(je)=upex(je,jface)
            upy(je)=upey(je,jface)
            upz(je)=upez(je,jface)
            unx(je)=unex(je,jface)
            uny(je)=uney(je,jface)
            unz(je)=unez(je,jface)
110    continue
        if(isym.eq.0) then
            istop=nfi
        else
            istop=jface
        endif
c    test surface loop
        do 400 iface=1,istop
            do 15 ie=1,3
                il=indefi(ie,iface)
                if(il.gt.0) then
                    deii(ie)=dei(il)
                else
                    deii(ie)=-dei(-il)
                endif
                iedge(ie)=iabs(il)
15    continue
            if(part) then
                call blocki(3,iedge,numb)
                if(numb.eq.0) go to 400
                call block2(3,iedge,3,jedge,numb)
                if(numb.eq.0) go to 400
            endif

            xcii=xc(i,iface)
            ycii=yc(i,iface)
            zcii=zci(i,iface)
            do 20 iv=1,3
                il=indvfi(iv,iface)
                xvii(iv)=xvi(il)
                yvii(iv)=yvi(il)
                zvii(iv)=zvi(il)
20    continue
c    calculate vertex dot products.
            do 30 j=1,3
                do 30 i=1,3
                    dot(i,j)=xvii(i)*xvjj(j)+yvii(i)*yvjj(j)+zvii(i)*zvjj(j)
30    continue
c    calculate green's function integrals.
            call grnint(fk,0.,xcii,ycii,zcii,xvii,yvii,zvii,deii,
1            xcjj,ycjj,zcjj,xvjj,yvjj,zvjj,dej,
2            nxj,nyj,nzj,area,upx,upy,upz,unx,uny,unz,
3            gl1,gl2,gl3,gl4,gal,gaa,gab,gac,gb1,gba,gb2,gb3,
4            gcl,gca,gcb,gcc)
            if(isym.ne.0 .and. iface.eq.jface) then
c    edge-independent term
                zface=(c1*(gaa*dot(1,1)+gba*dot(2,1)+gca*dot(3,1)+
1                    gab*dot(1,2)+gbb*dot(2,2)+gcb*dot(3,2)+

```

```

2          gac*dot(1,3)+gbc*dot(2,3)+gcc*dot(3,3))-c2*g11
3      )*.5

c edge-dependant terms
do 50 j=1,3
    zedgei(j)=-c1*(g1a*dot(j,1)+g1b*dot(j,2)+g1c*dot(j,3))*0.5
    zedgej(j)=-c1*(ga1*dot(1,j)+gb1*dot(2,j)+gc1*dot(3,j))*0.5
    ctemp=c1*g11*.5
    zedge(1,j)=ctemp*dot(1,j)
    zedge(2,j)=ctemp*dot(2,j)
    zedge(3,j)=ctemp*dot(3,j)
50    continue
    else
c edge-independant term
    zface=c1*(gaa*dot(1,1)+gba*dot(2,1)+gca*dot(3,1)+
1        gab*dot(1,2)+gbb*dot(2,2)+gcb*dot(3,2)+
2        gac*dot(1,3)+gbc*dot(2,3)+gcc*dot(3,3))-c2*g11
c edge-dependant terms
    do 51 j=1,3
        zedgei(j)=-c1*(g1a*dot(j,1)+g1b*dot(j,2)+g1c*dot(j,3))
        zedgej(j)=-c1*(ga1*dot(1,j)+gb1*dot(2,j)+gc1*dot(3,j))
        ctemp=c1*g11
        zedge(1,j)=ctemp*dot(1,j)
        zedge(2,j)=ctemp*dot(2,j)
        zedge(3,j)=ctemp*dot(3,j)
51    continue
    endif
c step over the 3 edges which bound face j.
    do 100 je=1,3
        jt=jedge(je)
        elj=dej(je)
c step over the 3 edges which bound face i.
        do 100 ie=1,3
            it=iedge(ie)
            eli=dei(ie)
c lpp impedance elements computed.
            zpp=eli*elj*(zface+zedgei(ie)+zedgej(je)+zedge(ie,je))
            if(isym.ne.0) then
                if(it.eq.jt) then
                    zpp=2.0*zpp
                else
                    endif
            endif
            call build(zpp,it,jt,isym,z,lz)
100    continue
400    continue
500    continue
    return
end

```

```

      subroutine patran(nunit,numpat,nface,nedge,xvert,yvert,zvert
+,ifacvrt)
c
c read patran-iges files
c
      implicit none
      integer ifacvrt(3,*)
      integer nunit,nnodes,nel,idx,idx1,idx2,idx3,numpat
+,nface,nedge
      real*4 xvert(*),yvert(*),zvert(*)
      character*30 name
c
c 4 blank lines skipped
c
      read(nunit,*)
      read(nunit,*)
      read(nunit,*)
      read(nunit,*)
c
c count nodes
c
      nnodes=0
100  continue
      read(nunit,*)
      read(nunit,10) name
10  format(56x,a7)
      if(name.eq.'NODE') then
         nnodes=nnodes+1
      else
         backspace(unit=nunit)
         backspace(unit=nunit)
         goto 200
      endif
      goto 100
200  continue
c
c count elements
c
      nel=0
300  continue
      read(nunit,*)
      read(nunit,10) name
      if(name.eq.'ELEMENT') then
         nel=nel+1
      elseif(name.eq.'POINT'.or.name.eq.'LINE'
& .or.name.eq.'SPLSRF'.or. name.eq.'SPLINE') then
      else
         backspace(unit=nunit)
         backspace(unit=nunit)
         goto 400
      endif
      goto 300
400  continue
c
c read vertices
c
      do 500 idx=1,nnodes
         read(nunit,20) xvert(idx),yvert(idx),zvert(idx)
20  format(3x,3(1x,g16.7))
500  continue
c
c read face indices
c
      do 600 idx=1,nel
         read(nunit,30) idx1,idx2,idx3
30  format(7x,3(1x,i8))

```

```

        idx1=(idx1+1)/2
        idx2=(idx2+1)/2
        idx3=(idx3+1)/2
        ifacvrt(1,idx)=idx1
        ifacvrt(2,idx)=idx2
        ifacvrt(3,idx)=idx3
        if(ifacvrt(1,idx).lt.1.or.ifacvrt(1,idx).gt.nnodes.or.
&         ifacvrt(1,idx).lt.1.or.ifacvrt(1,idx).gt.nnodes.or.
&         ifacvrt(1,idx).lt.1.or.ifacvrt(1,idx).gt.nnodes)then
            write(6,*)'    error: face vertex index is out of range'
            stop
        endif
600  continue
    numpat=nnodes
    nface=nel
    nedge=0
    return
end

```



```

      subroutine pogen(thet,phi,rthet,rphi,work1,work2)
C
C compute physical optics current coefficients for each of the 3d-patch
C surfaces.
C
      implicit none
      complex*8 rthet(*),rphi(*),work1(*),work2(*)
      integer ib,iv,if,ie,nfi,nei
      real*4 signe,thet,phi
C
      include 'arraydim.inc'
C
      include 'surcord.inc'
      include 'faces.inc'
      include 'indx1.inc'
      include 'edges.inc'
      include 'indx2.inc'
      include 'ecurind.inc'
      include 'order.inc'
      complex*8 const
      call vzero(numzel,rthet)
      call vzero(numzel,rphi)
      if(numzel.ne.0)then
C
C step over all exterior surfaces
C
        do 500 ib=1,nub
          iv=indp(ib)
          if=indf(ib)
          ie=inde(ib)
          nfi=numfac(ib)
          nei=numedg(ib)
          signe=1.0
C check if this is a boundary of region nreg.
          call ipatch(thet,phi,work1,work2,
            1          nfi,nei,idxvf(1,if),idxef(1,if),xc(if),yc(if),zc(if),
            2          xv(iv),yv(iv),zv(iv),edlen(ie))
C
C j currents.
C
          const=signe
          call vbuild(0,-const,work2,0,nei,0,iedind(ie),rthet)
          call vbuild(0,const,work1,0,nei,0,iedind(ie),rphi)
500      continue
        endif
      return
      end

```

```

      subroutine reorint(nface,ifacvrt,icor)
C
C is face orientation consistent with face 1?
C
C faces must be oriented so that normal vectors to faces are directed outward
C ; it is assumed that face 1 is outwardly directed
C
      implicit none
      integer ifacvrt(3,*),icor(*)
      integer nreor,ncheck,idxf,nface,idxs,isd1f,isd2f,
      +idxf2,idxs1,jsid1f,jsid2f
C
C nreor = number of faces reoriented
C ncheck= number of faces checked for correctness
C icor is a work array used for control only
C ifacvrt= array of vertex numbers comprising faces
C nface is the number of faces
C
      nreor=0
      ncheck=1
      icor(1)=1
      do 100 idxf=2,nface
         icor(idxf)=0
100    continue
C
C loop over faces (if all faces correct stop else continue)
C
200    continue
      do 300 idxf=1,nface
C
C is face orientation correct?
C
         if(icor(idxf).eq.1)goto 400
300    continue
         write(6,*)'consistent patch orientation questionable'
         write(6,*)'                               nface = ',nface
         write(6,*)'                               ncheck = ',ncheck
         goto 800
400    continue
         icor(idxf)=2
C
C loop over edges
C
         do 700 idxs=1,3
            if(idxs.eq.1)then
               isid1f=ifacvrt(2,idxf)
               isid2f=ifacvrt(3,idxf)
            elseif(idxs.eq.2)then
               isid1f=ifacvrt(3,idxf)
               isid2f=ifacvrt(1,idxf)
            else
               isid1f=ifacvrt(1,idxf)
               isid2f=ifacvrt(2,idxf)
            endif
700    continue
C
C are faces adjacent to iface correctly oriented?
C
         do 600 idxf2=1,nface
            if(icor(idxf2).ne.0)goto 600
C
C loop over edges
C
            do 500 idxs1=1,3
               if(idxs1.eq.1)then
                  jsid1f=ifacvrt(2,idxf2)
                  jsid2f=ifacvrt(3,idxf2)

```

```

elseif(idxs1.eq.2)then
  jsid1f=ifacvrt(3,idxf2)
  jsid2f=ifacvrt(1,idxf2)
else
  jsid1f=ifacvrt(1,idxf2)
  jsid2f=ifacvrt(2,idxf2)
endif

c
c are edges common to adjacent faces numbere3d in reverse; yes they are orie
c correctly, no flip vertices of idxf2
c
  if(jsid1f.eq.jsid2f.and.isid2f.eq.jsid1f)then
    ncheck=ncheck+1
    icor(idxf2)=1
    goto 700
  elseif(isid1f.eq.jsid1f.and.isid2f.eq.jsid2f)then
    c change orientation
    c
    ncheck=ncheck+1
    icor(idxf2)=1
    nreor=nreor+1
    if(idxs1.eq.1)then
      ifacvrt(2,idxf2)=jsid2f
      ifacvrt(3,idxf2)=jsid1f
    elseif(idxs1.eq.2)then
      ifacvrt(3,idxf2)=jsid2f
      ifacvrt(1,idxf2)=jsid1f
    else
      ifacvrt(1,idxf2)=jsid2f
      ifacvrt(2,idxf2)=jsid1f
    endif
    goto 700
  endif
endif
continue
continue
continue
if(ncheck.lt.nface)goto 200
continue
write(6,*)
write(6,*)'faces reoriented = ',nreor
return
end
500
600
700
800

```

```

      subroutine rgen(n,th,ph,rbt,rbp,irv,vbt,vbp,wt,wp,xwt,xwp,
1 nbi)
c
c calculate general cfie r and v matrices for theta and phi polarization.
c either geometric symmetry or imaged fields are included.
c the following inner products are calculated:
c       $r = \langle j, e \rangle$ 
c       $\langle m, h \rangle$ 
c       $v = \langle j, \alpha e + (1-\alpha) \eta n \times h \rangle$ 
c       $\langle m, \alpha h + (1-\alpha) / \eta n \times e \rangle$ 
c
      implicit none
      complex*8 rbt(*),rbp(*),vbt(*),vbp(*),wt(*),wp(*),xwt(*),xwp(*)
      integer ib,nbi,n,nei,irv,indi,indzj
      real*4 signe,consta,th,ph
c
      include 'arraydim.inc'
c
      complex*8 conij
      common /index/ conij,indi(nwork),
1 indzj(nwork)
      complex*8 const
c
c step over all exterior surfaces
c
      do 500 ib=1,nbi
         signe=1.0
         consta=1.0
         call rop(ib,n,th,ph,wt,wp,xwt,xwp)
         const=consta
         call vindex(const,ib,nei)
c
c j current testing.
c
c e-field.
         const=signe*conij
         if(irv.eq.1 .or. irv.eq.0) then
            call vbuild(0,const,wt,0,nei,0,indzj,rbt)
            call vbuild(0,const,wp,0,nei,0,indzj,rbp)
         endif
         if(irv.eq.2) then
            call vbuild(0,const,wt,0,nei,0,indzj,vbt)
            call vbuild(0,const,wp,0,nei,0,indzj,vbp)
         endif
500 continue
      return
      end

```

```
subroutine rop(ib,n,th,ph,wt,wp,xwt,xwp)
```

```
c
```

```
c calculate the r matrix operator.
```

```
c
```

```
implicit none
```

```
complex*8 wt(*),wp(*),xwt(*),xwp(*)
```

```
real*4 c,th,ph
```

```
integer iv,ib,if,ie,nfi,nei,n
```

```
include 'arraydim.inc'
```

```
include 'surcord.inc'
```

```
include 'faces.inc'
```

```
include 'indx1.inc'
```

```
include 'edges.inc'
```

```
include 'indx2.inc'
```

```
c=1.0
```

```
iv=indp(ib)
```

```
if=indf(ib)
```

```
ie=inde(ib)
```

```
nfi=numfac(ib)
```

```
nei=numedg(ib)
```

```
call rpatch(th,ph,wt,wp,xwt,xwp,
```

```
1 nfi,nei,idxvf(1,if),idxef(1,if),xc(if),yc(if),zc(if),
```

```
2 xv(iv),yv(iv),zv(iv),edlen(ie))
```

```
return
```

```
end
```

```

      subroutine rpatch(th,ph,rbt,rbp,xrbt,xrbp,
1 nface,nedge,indvf,inde,xc,yc,zc,xv,yv,zv,de)
c
c compute 3d-patch efie r matrices for theta and phi polarization.
c
c analytic expressions.
c
      implicit none
      complex*8 rbt(*),rbp(*),xrbt(*),xrbp(*)
      dimension indvf(3,*),inde(3,*),xc(*),yc(*),zc(*),
1 xv(*),yv(*),zv(*),de(*)
      complex*8 a1,rt1,rp1,xrt1,xrp1
      real*4 j0,j1,nx,ny,nz
      real*4 kp2,kpx,kpy,kpz,knx,kny,knz
      complex*8 cxx,cyy,czz,dskx,dsky,dskz,sk
      real*4 xvi(3),yvi(3),zvi(3),dei(3)
      real*4 dtor,th,cth,sth,cph,ph,sph,fkx,fky,fkz,fk2
      +,de,xc,yc,yci,yc,zci,zc,xv,yv,zv,xx1,yy1,zz1,xx2,
      +yy2,zz2,del,area,dot,rnx,rny,rnz,x1,y1,z1,x2,y2,z2,
      +crossx,crossy,crossz,trip,dot2,xx,yy,zz,arg,el
      integer nedge,iface,nface,ie,i1,inde,iv,indvf
      +,i2,l1
      integer iedge(3)
      include 'wavenumber.inc'
      include 'tilde.inc'
      include 'const.inc'
      dtor=pi/180.
      call vzero(nedge,rbt)
      call vzero(nedge,rbp)
      call vzero(nedge,xrbt)
      call vzero(nedge,xrbp)
      if(th.eq.90.0) then
         cth=0.0
         sth=1.0
      else
         cth=cos(th*dtor)
         sth=sin(th*dtor)
      endif
      cph=cos(ph*dtor)
      sph=sin(ph*dtor)
      fkx=fk*sth*cph
      fky=fk*sth*sph
      fkz=fk*cth
      fk2=fk**2
c step over each 3d-patch face.
      do 200 iface=1,nface
         do 5 ie=1,3
            i1=inde(ie,iface)
            if(i1.gt.0) then
               dei(ie)=de(i1)
            else
               dei(ie)=-de(-i1)
            endif
            iedge(ie)=iabs(i1)
5          continue
            xci=xc(iface)
            yci=yc(iface)
            zci=zc(iface)
            do 10 iv=1,3
               i1=indvf(iv,iface)
               xvi(iv)=xv(i1)
               yvi(iv)=yv(i1)
               zvi(iv)=zv(i1)
10          continue
c unit normal vector to triangular face
            xx1=xvi(2)-xvi(1)

```

```

yy1=yvi(2)-yvi(1)
zz1=zvi(2)-zvi(1)
xx2=xvi(3)-xvi(1)
yy2=yvi(3)-yvi(1)
zz2=zvi(3)-zvi(1)
nx=yy1*zz2-zz1*yy2
ny=-xx1*zz2+zz1*xx2
nz=xx1*yy2-yy1*xx2
del=sqrt(nx**2+ny**2+nz**2)
nx=nx/del
ny=ny/del
nz=nz/del
c area of triangular face
area=0.5*del
if(rtilde) then
c analytic integrals
c compute normal and parallel vectors
dot=xci*nx+yci*ny+zci*nz
rnx=dot*nx
rny=dot*ny
rnz=dot*nz
dot=fkx*nx+fky*ny+fkz*nz
knx=dot*nx
kny=dot*ny
knz=dot*nz
kpx=fkx-knx
kpy=fky-kny
kpz=fkz-knz
c compute sk and grad(sk) integrals
kp2=kpx**2+kpy**2+kpz**2
if(kp2/fk2.ge.0.0001) then
sk=0.0
dskx=0.0
dsky=0.0
dskz=0.0
c loop over the 3 sides of the face.
do 50 i1=1,3
i2=i1+1
if(i2.gt.3) i2=1
x1=xvi(i1)
y1=yvi(i1)
z1=zvi(i1)
x2=xvi(i2)
y2=yvi(i2)
z2=zvi(i2)
xx1=x2-x1
yy1=y2-y1
zz1=z2-z1
crossx=ny*zz1-nz*yy1
crossy=-nx*zz1+nz*xx1
crossz=nx*yy1-ny*xx1
trip=kpx*crossx+kpy*crossy+kpz*crossz
dot=0.5*(kpx*xx1+kpy*yy1+kpz*zz1)
if(abs(dot).ge.1.e-4) then
j0=sin(dot)/dot
j1=0.5*(j0-cos(dot))/dot
elseif(abs(dot).ge.1.e-8) then
dot2=dot**2
j0=1.0-dot2/6.+dot2*dot2/120.
j1=dot/6.-dot*dot2/60.
else
j0=1.0
j1=0.0
endif
xx=0.5*(x2+x1)-rnx
yy=0.5*(y2+y1)-rny

```

```

        zz=0.5*(z2+z1)-rnz
        arg=kpx*xx+kpy*yy+kpz*zz
        a1=cmlpx(cos(arg),sin(arg))
        sk=sk+trip*j0*a1
        dskx=dskx+a1*cmlpx((crossx-2.*kpx/kp2*trip)*j0
&         -xx1*trip*j1,xx*trip*j0)
        dsky=dsky+a1*cmlpx((crossy-2.*kpy/kp2*trip)*j0
&         -yy1*trip*j1,yy*trip*j0)
        dskz=dskz+a1*cmlpx((crossz-2.*kpz/kp2*trip)*j0
&         -zz1*trip*j1,zz*trip*j0)
50    continue
        sk=(0.,1.)*sk/kp2
        dskx=dskx/kp2
        dsky=dsky/kp2
        dskz=dskz/kp2
    else
c parallel component of k nearly zero
        sk=area
        dskx=area*(xci-rnx)
        dsky=area*(yci-rny)
        dskz=area*(zci-rnz)
    endif
c centroid appoximation
        arg=knx*rnx+kny*rny+knz*rnz
        a1=0.5/area*cmlpx(cos(arg),sin(arg))
    else
        arg=fkx*xci+fky*yci+fz*zci
        a1=0.5*cmlpx(cos(arg),sin(arg))
    endif
c step over the 3 edges which bound face i.
    do 100 ie=1,3
        l1=iedge(ie)
        el=dei(ie)
c e-field equation r matrix computed.
        if(rtilde) then
            cxx=dskx+(rnx-xvi(ie))*sk
            cyy=dsky+(rny-yvi(ie))*sk
            czz=dskz+(rnz-zvi(ie))*sk
        else
            cxx=xci-xvi(ie)
            cyy=yci-yvi(ie)
            czz=zci-zvi(ie)
        endif
c theta polarized e-field.
        rtl=(cth*(cxx*cph+cyy*sph)-czz*sth)*a1
        rbt(l1)=el*rtl+rbt(l1)
        xrtl=(-(ny*sth+nz*cth*sph)*cxx+(nx*sth+nz*cth*cph)*cyy+
&         (nx*cth*sph-ny*cth*cph)*czz)*a1
        xrbt(l1)=el*xrtl+xrbt(l1)
c phi polarized e-field.
        rp1=(-cxx*sph+cyy*cph)*a1
        rbp(l1)=el*rp1+rbp(l1)
        xrp1=(-nz*cph*cxx-nz*sph*cyy+(nx*cph+ny*sph)*czz)*a1
        xrbp(l1)=el*xrp1+xrbp(l1)
100    continue
200    continue
    return
end

```



```

      subroutine vgen(n,th,ph,rt,rp,irv,vt,vp,work1,work2,work3,work4)
c
c compute r matrices for 3d-patch and wire surfaces.
c irv = 0 - both r and v arrays are returned
c       1 - r arrays are returned
c       2 - v arrays are returned
      implicit none
      integer irv,n,lz,i
      real*4 th,ph
      complex*8 rt(*),rp(*),vt(*),vp(*)
      complex*8 work1(*),work2(*),work3(*),work4(*)
      include 'arraydim.inc'
      include 'surcord.inc'
      include 'order.inc'
      if(irv.eq.0 .or. irv.eq.1) then
        call vzero(numzel,rt)
        call vzero(numzel,rp)
      endif
      if(irv.eq.0 .or. irv.eq.2) then
        call vzero(numzel,vt)
        call vzero(numzel,vp)
      endif
      if(numzel.ne.0) then
        call rgen(n,th,ph,rt,rp,irv,vt,vp,work1,work2,work3,work4,
1      nub)
      endif
      if(irv.eq.0) then
        lz=numzel
        do 100 i=1,lz
          vt(i)=rt(i)
          vp(i)=rp(i)
100      continue
        endif
      endif
      return
      end

```

```

      subroutine solhyb
C
C calculate mm and po currents and fields
C
      implicit none
      include 'arraydim.inc'
      include 'work.inc'
      include 'order.inc'
      include 'threat.inc'
      include 'numj.inc'
      include 'polar.inc'
      include 'scatter.inc'
      include 'const.inc'
      complex zimp(nzmax)
      complex vth(numax), vph(numax), thcur(numax), phcur(numax)
      complex radt(numax), radp(numax)
      complex ttskat, ppskat, tpskat, ptskat
      integer ipvtz(numax)
      integer ntot, nmom, npo, imom, ipomm, idxmm, idxpo, k, i, i2, idx, i1
      real*4 rcond, dumy
      logical solvth, solvph
C
C assign mm, po, and total number of unknowns
C
      ntot=numzel
      nmom=nelcur(1)
      npo=nelcur(2)
C
C print total number of unknowns, mm unknowns, and po unknowns
C
      write(6,*)
      write(6,*) ' total number of unknowns = ',ntot
      write(6,*) ' number of mm unknowns = ',nmom
      write(6,*) ' number of po unknowns = ',npo
C
C calculate mom and po pointers for impedance matrix
C
      imom=1
      ipomm=imom+nmom*(nmom+1)/2
C
C calculate mm and po pointers for excitation vector
C
      idxmm=1
      idxpo=idxmm+nmom
C
C calculate mm impedance matrix
C
      write(6,*)
      write(6,*) 'calculating mm coefficient matrix'
C
C zimp is the mm impedance matrix
C
      call zimped(0,0,-1,zimp(imom),work1,ntot,idxmm,idxmm+nmom-1,
&idxmm,idxmm+nmom-1)
C
C factor mm impedance matrix using symmetric linpack routine
C
      call cspco(zimp(imom),nmom,ipvtz(idxmm),rcond,work1)
      write(6,*) ' impedance matrix condition = ',1./(rcond+1.e-30)
C
C calculate po/mm impedance matrix
C zimp is the mm/po impedance matrix
C
      write(6,*)
      write(6,*) 'calculating po/mm interaction matrix'
      call zimped(0,0,-1,zimp(ipomm),work1,ntot,idxmm,idxmm+nmom-1,

```

```

      &idxpo,idxpo+npo-1)
c
c determine polarizations for rcs
c
      solvth=(monbi.ge.0 .and. polt)
      solvph=(monbi.ge.0 .and. polp)
      do 100 k=1,nang
        if(monbi.gt.0) then
          thet=thes(k)
          phi=phis(k)
        endif
c
c calculate excitation vector
c
        if(monbi.eq.0) then
c
c bistatic case
c
          call vgen(-1,thes(k),phis(k),radt,radp,1,dumy,dumy,
&            work1,work2,work3,work4)
          if(k.eq.1) then
            call vgen(-1,thet,phi,dumy,dumy,2,vth,vph,
&            work1,work2,work3,work4)
          endif
          else
c
c monostatic case
c
            call vgen(-1,thes(k),phis(k),radt,radp,0,vth,vph,
&            work1,work2,work3,work4)
          endif
          if(monbi.gt.0.or.(monbi.eq.0.and.k.eq.1)) then
            do 139 i=1,ntot
              thcur(i)=0.
              phcur(i)=0.
              if(solvth) thcur(i)=vth(i)
              if(solvph) phcur(i)=vph(i)
139          continue
c
c calculate po currents
c
            call pogen(thet,phi,work1,work2,work3,work4)
            do 50 i2=idxpo,idxpo+npo-1
              thcur(i2)=work1(i2)
              phcur(i2)=work2(i2)
50          continue
c
c calculate mm/po right hand side (v-zpomm*ipo)
c
            idx=ipomm-1
            do 70 i2=idxpo,idxpo+npo-1
              do 60 i1=idxmm,idxmm+nmom-1
                idx=idx+1
c
c thcur,phcur=theta/phi polarized rhss
c
                if(solvth) thcur(i1)=thcur(i1)-zimp(idx)*thcur(i2)
                if(solvph) phcur(i1)=phcur(i1)-zimp(idx)*phcur(i2)
60              continue
70            continue
c
c solve for mm currents using linpack cspsl
c
            if(solvth) call cspsl(zimp(imom),nmom,ipvtz(idxmm),thcur(idxmm))
            if(solvph) call cspsl(zimp(imom),nmom,ipvtz(idxmm),phcur(idxmm))
          endif

```

```

c
c calculate far fields
c
      ttsskat=0.
      tpskat=0.
      ptskat=0.
      ppskat=0.
      if(solvth) then
        call dot(radt,thcur,ttsskat,ntot)
        call dot(radp,thcur,ptsskat,ntot)
      endif
      if(solvph) then
        call dot(radt,phcur,tpsskat,ntot)
        call dot(radp,phcur,ppsskat,ntot)
      endif
      ttsscat(k)=ttsskat
      ppsscat(k)=ppsskat
      tpsscat(k)=tpsskat
      ptsscat(k)=ptsskat
100 continue
      return
      end

```

```

      subroutine solmom
C
C calculate mm currents and fields
C
      implicit none
      include 'arraydim.inc'
      include 'work.inc'
      include 'order.inc'
      include 'threat.inc'
      include 'numj.inc'
      include 'polar.inc'
      include 'scatter.inc'
      include 'const.inc'
      complex zimp(nzmax)
      complex vth(numax), vph(numax), thcur(numax), phcur(numax)
      complex radt(numax), radp(numax)
      complex ttskat, ppskat, tpskat, ptskat
      integer ipvtz(numax)
      integer ntot, nmom, npo, imom, idxmm, k, i
      real*4 rcond, dummy
      logical solvth, solvph
C
C assign mm, po, and total number of unknowns
C
      ntot=numzel
      nmom=nelcur(1)
      npo=nelcur(2)
C
C print total number of unknowns, mm unknowns, and po unknowns
C
      write(6,*)
      write(6,*) ' total number of unknowns = ',ntot
      write(6,*) ' number of mm unknowns = ',nmom
      write(6,*) ' number of po unknowns = ',npo
C
C calculate mom pointers for impedance matrix
C
      imom=1
C
C calculate mm pointers for excitation vector
C
      idxmm=1
C
C calculate mm impedance matrix
C
      write(6,*)
      write(6,*) 'calculating mm coefficient matrix'
C
C zimp is the mm impedance matrix
C
      call zimped(0,0,-1,zimp(imom),work1,ntot,idxmm,idxmm+nmom-1,
&idxmm,idxmm+nmom-1)
C
C factor mm impedance matrix using symmetric linpack routine
C
      call cspco(zimp(imom),nmom,ipvtz(idxmm),rcond,work1)
      write(6,*) ' impedance matrix condition = ',1./(rcond+1.e-30)
C
C determine polarizations for rcs
C
      solvth=(monbi.ge.0 .and. polt)
      solvph=(monbi.ge.0 .and. polp)
      do 100 k=1,nang
        if(monbi.gt.0) then
          thet=thes(k)
          phi=phis(k)

```

```

endif
c
c calculate excitation vector
c
      if(monbi.eq.0)then
c
c bistatic case
c
      call vgen(-1,thes(k),phis(k),radt,radp,1,dumy,dumy,
&      work1,work2,work3,work4)
      if(k.eq.1)then
        call vgen(-1,thet,phi,dumy,dumy,2,vth,vph,
&      work1,work2,work3,work4)
      endif
    else
c
c monostatic case
c
      call vgen(-1,thes(k),phis(k),radt,radp,0,vth,vph,
&      work1,work2,work3,work4)
    endif
    if(monbi.gt.0.or.(monbi.eq.0.and.k.eq.1))then
      do 139 i=1,ntot
        thcur(i)=0.
        phcur(i)=0.
        if(solvth)thcur(i)=vth(i)
        if(solvph)phcur(i)=vph(i)
139      continue
c
c solve for mm currents using linpack cspsl
c
        if(solvth)call cspsl(zimp(imom),nmom,ipvtz(idxmm),thcur(idxmm))
        if(solvph)call cspsl(zimp(imom),nmom,ipvtz(idxmm),phcur(idxmm))
      endif
c
c calculate far fields
c
      ttskat=0.
      tpskat=0.
      ptskat=0.
      ppskat=0.
      if(solvth)then
        call dot(radt,thcur,ttskat,ntot)
        call dot(radp,thcur,ptskat,ntot)
      endif
      if(solvph)then
        call dot(radt,phcur,tpskat,ntot)
        call dot(radp,phcur,ppskat,ntot)
      endif
      ttscat(k)=ttskat
      ppscat(k)=ppskat
      tpscat(k)=tpskat
      ptscat(k)=ptskat
100  continue
      return
    end

```

```

      subroutine solpo
c
c calculate po currents and fields
c
      implicit none
      include 'arraydim.inc'
      include 'work.inc'
      include 'order.inc'
      include 'threat.inc'
      include 'numj.inc'
      include 'polar.inc'
      include 'scatter.inc'
      include 'const.inc'
      complex zimp(nzmax)
      complex vth(numax), vph(numax), thcur(numax), phcur(numax)
      complex radt(numax), radp(numax)
      complex ttskat, ppskat, tpskat, ptskat
      integer ipvtz(numax)
      integer ntot, nmom, npo, ipo, idxpo, k, i, i2
      real*4 dummy
      logical solvth, solvph
c
c assign mm, po, and total number of unknowns
c
      ntot=numzel
      nmom=nelcur(1)
      npo=nelcur(2)
c
c print total number of unknowns, mm unknowns, and po unknowns
c
      write(6,*)
      write(6,*) ' total number of unknowns = ',ntot
      write(6,*) ' number of mm unknowns = ',nmom
      write(6,*) ' number of po unknowns = ',npo
c
c calculate po pointers for impedance matrix
c
      ipo=1
c
c calculate po pointers for excitation vector
c
      idxpo=1
c
c determine polarizations for rcs
c
      solvth=(monbi.ge.0 .and. polt)
      solvph=(monbi.ge.0 .and. polp)
      do 100 k=1,nang
        if(monbi.gt.0)then
          thet=thes(k)
          phi=phis(k)
        endif
c
c calculate excitation vector
c
        if(monbi.eq.0)then
c
c bistatic case
c
          call vgen(-1,thes(k),phis(k),radt,radp,1,dummy,dummy,
& work1,work2,work3,work4)
          if(k.eq.1)then
            call vgen(-1,thet,phi,dummy,dummy,2,vth,vph,
& work1,work2,work3,work4)
          endif
        else

```

```

c
c monostatic case
c
      call vgen(-1,thes(k),phis(k),radt,radp,0,vth,vph,
&      work1,work2,work3,work4)
      endif
      if(monbi.gt.0.or.(monbi.eq.0.and.k.eq.1)) then
        do 139 i=1,ntot
          thcur(i)=0.
          phcur(i)=0.
          if(solvth)thcur(i)=vth(i)
          if(solvph)phcur(i)=vph(i)
139      continue
c
c calculate po currents
c
      call pogen(thet,phi,work1,work2,work3,work4)
      do 50 i2=idxpo,idxpo+npo-1
        thcur(i2)=work1(i2)
        phcur(i2)=work2(i2)
50      continue
      endif
c
c calculate far fields
c
      ttskat=0.
      tpskat=0.
      ptskat=0.
      ppskat=0.
      if(solvth) then
        call dot(radt,thcur,ttskat,ntot)
        call dot(radp,thcur,ptskat,ntot)
      endif
      if(solvph) then
        call dot(radt,phcur,tpskat,ntot)
        call dot(radp,phcur,ppskat,ntot)
      endif
      ttscat(k)=ttskat
      ppscat(k)=ppskat
      tpscat(k)=tpskat
      ptscat(k)=ptskat
100 continue
      return
      end

```



```

      subroutine sysind(ib,jb)
c
c compute system matrix indices for testing and expansion functions
c
      implicit none
      include 'arraydim.inc'
      include 'index.inc'
      integer ib,jb
      nincl=nincl+1
      if(nincl.gt.1)then
        write(6,*)' error subroutine sysind , nincl > 1'
        stop
      endif
c
c given the surface # eindex returns an array containg the global
c unknown indices corresponding to the edges of the surface
c ib and jb are surface numbers
c indxi and indxj are arrays that contain global unknown # of edges
c of surfaces
c
      call eindex(ib,indxi)
      call eindex(jb,indxj)
      return
      end

```

```

      subroutine vbuild(ncomp,const,a,ij1,npi,ij2,indi,b)
c
c add or subtract the subvectors of vector a to vector b.
c
      implicit none
      complex*8 const,sign
      complex*8 a(*),b(*)
      integer indi(*)
      integer li(6)
      integer ncomp,nei,npi,istop,ia,i,ni,ib,inda
      +,indb,ki,ij1,ij2
      if(const.eq.0.0) return
      if(ncomp.eq.0) then
         nei=npi
         istop=nei
      else
         li(1)=1
         li(2)=(npi-3)/2
         li(3)=1
         li(4)=1
         li(5)=(npi-3)/2
         li(6)=1
         istop=ncomp*3
      endif
      ia=1
      do 960 i=1,istop
         if(ncomp.eq.0) then
            ni=1
         else
            ni=li(i)
         endif
         if(ni.eq.0) go to 960
         ib=indi(i)
         if(ib.eq.0) go to 940
c add the (i) subvector of vector a to vector b.
         sign=const
         if(ib.lt.0) sign=-const
         ib=iabs(ib)
         inda=ia-1
         indb=ib-1
         do 110 ki=1,ni
            inda=inda+1
            indb=indb+1
            b(indb)=sign*a(inda)+b(indb)
110      continue
940      continue
         ia=ia+ni
960      continue
      return
      end

```

```

      subroutine vindex(const,ib,nbasis)
c
c compute vector indices for testing functions.
c
      implicit none
      complex*8 const
      integer nbasis,ib,indi,i,indzj
      include 'arraydim.inc'
      complex*8 conij
      common /index/ nloc,conij,indi(nwork),
1 indzj(nwork)
      include 'edges.inc'
      nbasis=numedg(ib)
      conij=const
      call eindex(ib,indi)
      do 100 i=1,nbasis
         indzj(i)=indi(i) ! ak
100 continue
      return
      end

```

```
      subroutine vzero(n,a)
c
c set array a of length n to (0.,0.)
c
      implicit none
      integer i,n
      complex a(*)
      do 10 i=1,n
        a(i)=(0.,0.)
10    continue
      return
      end
```

```

      subroutine zelcal(ipoctrl,notused,zimpmat,nordimp,work,
& nbndi,nzji,nbndj,nzjj)
c
c do not use any optimization when compiling this routine on Paragon
c -O3 resulted in wrong answer (compiler error)
c
c calculate <j,e>
c
      implicit none
      complex zimpmat(*),work(*)
      complex const
      integer nzji(*),nzjj(*)
      integer jbnd,nbndj,ibnd,nbndi,njj,nordimp,ipoctrl,notused
      include 'arraydim.inc'
      include 'block.inc'
      include 'index.inc'
      if(part)then
        write(6,*)' generate the following matrix sub-block:'
        write(6,*)'      rows      ',iblck1,' thru ',iblck2
        write(6,*)'      columns ',jblck1,' thru ',jblck2
        write(6,*)
        if(iblck1.gt.jblck1)then
          write(6,*)' error: block must be above the main diagonal'
          stop
        endif
      endif
      do 500 jbnd=1,nbndj
      do 500 ibnd=1,nbndi
        if(ibnd.gt.jbnd)goto 500
        write(6,*)
        write(6,*) ' ibnd = ',ibnd,'      jbnd = ',jbnd
c
c e-field and h-field l operator.
c
        nincl=0
        njj=nzji(ibnd)*nzjj(jbnd)
        if(njj.eq.0)goto 500
c e-field , < j , l(j) >.
        curij=1.
        call sysind(ibnd,jbnd)
c
c always true
c
        if(part)call block(ibnd,jbnd)
        if(nincl.ne.0)then
          call lop(ibnd,jbnd,zimpmat,nordimp)
        endif
500 continue
      return
      end

```

```

      subroutine zimped(izadd,iopctrl,n,zmat,work,lvecmax
& ,ilowbnd,iupbnd,jlowbnd,jupbnd)
C
C calculate impedance matrix
C
      implicit none
      complex zmat(*),work(*)
      integer ilowbnd,iupbnd,jlowbnd,jupbnd, isize, jsize,
+lvecmax,nzsize,iopctrl,izadd,n
      include 'arraydim.inc'
      include 'surcord.inc'
      include 'ecurind.inc'
      include 'order.inc'
      include 'block.inc'
C
C set row/column boundaries for block
C
      iblck1=ilowbnd
      iblck2=iupbnd
      jblck1=jlowbnd
      jblck2=jupbnd
C
C calculate the number of rows and columns in block
C
      isize=iblck2-iblck1+1
      jsize=jblck2-jblck1+1
      if(isize.le.0.or.jsize.le.0)then
        write(6,*)' error: block size <= 0 (zimped)'
        stop
      endif
C
C is this a partitioned solution? for mm/po part=true
C
      if(isize.eq.lvecmax.and.jsize.eq.lvecmax)then
        part=.false.
      else
        part=.true.
      endif
C
C if symmetric block only calculate upper triangle and diagonal
C
      if(iblck1.eq.jblck1.and.iblck2.eq.jblck2)then
        nzsize=isize*(isize+1)/2
      else
        nzsize=isize*jsize
      endif
      call vzero(nzsize,zmat)
      if(numzel.ne.0)then
C
C calculate the elements of the impedance matrix (by block)
C
        call zelcal(iopctrl,0,zmat,isize,work,nub,nbji,
& nub,nbji)
      endif
      return
      end

```

```

c
c arraydim.inc
c
c      nptsmax: max number of vertex points
c      namax: max number of scattering angles
c      nzmax: max number Z matrix elements
c      numax: max number of unknowns
c      nbndmax: max number of boundaries
c      nfacmax: max number of faces
c      nedgmax: max number of edges
c      nwork: max number work matrix elements
c
      integer nptsmax,namax,nzmax,numax,nbndmax,nfacmax,
+nedgmax,nwork
      parameter (nptsmax=3000,namax=722,nzmax=900000
& ,numax=5000,nbndmax=40,nfacmax=4000,
& nedgmax=6000,nwork=6000)

```

```
c
c block.inc
c
```

```
integer iblck1,iblck2,jblck1,jblck2
logical part
common/blok1/part
common/bloki/iblck1,iblck2,jblck1,jblck2
```



```
c
c const.inc
```

```
c      real*4 pi,eta
      common/const/pi,eta
```

```
c
c dbsm.inc
c
    logical dbsm
    common/dbsm1/dbsm
```

```
c
c ecurind.inc
c
      integer nbji (nbndmax) , iedind (nedgmax) , idxuned (numax, 2)
      common /ecuri/ nbji, iedind, idxuned
```

```
c
c edges.inc
c
      integer numedg(nbndmax),edtype(nedgmax)
      real*4 edlen(nedgmax)
      common/edgei/numedg,edtype
      common/edger/edlen
```

C
C
C

faces.inc

```
real*4 normx(nfacmax),normy(nfacmax),normz(nfacmax)
real*4 area(nfacmax)
real*4 upedx(3,nfacmax),upedy(3,nfacmax),upedz(3,nfacmax)
real*4 unedx(3,nfacmax),unedy(3,nfacmax),unedz(3,nfacmax)
real*4 xc(nfacmax),yc(nfacmax),zc(nfacmax)
integer numfac(nbndmax)
common/facesi/numfac
common/facesr/xc,yc,zc
+,normx,normy,normz,area
+,upedx,upedy,upedz,unedx,unedy,unedz
```

```
c
c index.inc
c
  complex*8 curij
  integer nincl,indx1(nwork),indxj(nwork)
  common /indexc/curij
  common /indexi/nincl,indx1,indxj
```

```
c
c indxl.inc
c
      integer idxvf(3,nfacmax),idxef(3,nfacmax),idxve(2,nedgmax)
      +,idxfem(nedgmax),idxfep(nedgmax)
      common/indxl/idxvf,idxef,idxve,idxfem,idxfep
```

```
c
c indx2.inc
c
      integer indp(nbndmax),indf(nbndmax),inde(nbndmax)
      common/indx2/indp,indf,inde
```



```
C  
C intord.inc
```

```
C  
    real*4 rnear,rfar  
    integer nself,nnear,nmid,nfar  
    common /intdist/ rnear,rfar  
    common /intord/  nself,nnear,nmid,nfar
```

```
c
c numbers.inc
c
      integer npoynt,nfaces,nedges
      common/numbers/npoynt,nfaces,nedges
```

```
c
c numj.inc
c
      integer nelcur(2)
      common/numj/nelcur
```

```
c
c order.inc
c
    integer numzel
    common /order/numzel
```

```
c
c patchtol.inc
c
    real*4 pattol
    common/patol/pattol
```

```
c
c polar.inc
c
      logical polt,polp
      common/polar/polt,polp
```

c
c problem.inc
c

character*80 title
common/problem/title

```
c
c scatter.inc
c
      complex*8 ttscat(namax),ppscat(namax),
+tpscat(namax),ptscat(namax)
      integer nang,iangle(namax)
      real*4 thes(namax),phis(namax)
      common/scatc/ttscat,ppscat,tpscat,ptscat
      common/scati/nang,iangle
      common/scatr/thes,phis
```



```
c
c surcord.inc
c
      integer nub,nup(nbndmax)
      real*4 xv(nptsmax),yv(nptsmax),zv(nptsmax)
      common/surcri/nub,nup
      common/surcrr/xv,yv,zv
```

```
c  
c threat.inc  
c
```

```
integer monbi  
real*4 thet,phi  
common /threti/monbi  
common /thretr/thet,phi
```

```
c  
c tilde.inc  
c  
    logical rtilde  
    common /tilde/rtilde
```

```
c
c triint.inc
c
      real*4 weight(108),alfa(108),beta(108)
      integer nmpont(25)
      common /triint/ weight,alfa,beta
      common /tripnt/ nmpont
```

c
c unit.inc

c
character*5 units
common/unit/units

```
c
c wavenumber.inc
c
      real*4 fk
      common /wavnum/ fk
```

```
C  
C work.inc
```

```
C  
    complex work1 (nwork) , work2 (nwork) , work3 (nwork) ,  
+work4 (nwork)  
    common/work/work1,work2,work3,work4
```

CYLINDER (MM/PO SOLUTION)

\$option \$end

6.2831853

patch

17 surface 1 endcap at z=0

1	0.10000	0.80900E-01	0.30900E-01	-0.30900E-01	-0.80900E-01	-0.10000
	-0.80900E-01	-0.30900E-01	0.30900E-01	0.80900E-01	-0.37300E-01	0.79208E-02
	-0.45221E-01	0.00000E+00	0.45221E-01	-0.79208E-02	0.37300E-01	
	0.00000E+00	0.58780E-01	0.95110E-01	0.95110E-01	0.58780E-01	0.00000E+00
	-0.58780E-01	-0.95110E-01	-0.95110E-01	-0.58780E-01	0.37876E-01	0.52567E-01
	-0.14690E-01	0.00000E+00	0.14690E-01	-0.52567E-01	-0.37876E-01	
	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00
	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00
	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00	

22	0					
13	11	14	(.3,.2)	0.0	0.0	(.68,.24) 0.0 0.0 0.0
12	11	14				
12	14	15				
14	13	16				
14	16	17				
15	14	17				
6	11	5				
6	13	11				
7	13	6				
7	16	13				
8	16	7				
9	16	8				
9	17	16				
10	17	9				
1	17	10				
1	15	17				
2	15	1				
2	12	15				
3	12	2				
4	12	3				
4	11	12				
5	11	4				

90 250 surface 2 cylinder body

1	0.1000	0.8090E-01	0.3090E-01	-0.3090E-01	-0.8090E-01	-0.1000
	-0.8090E-01	-0.3090E-01	0.3090E-01	0.8090E-01	0.1000	0.8090E-01
	0.3090E-01	-0.3090E-01	-0.8090E-01	-0.1000	-0.8090E-01	-0.3090E-01
	0.3090E-01	0.8090E-01	0.1000	0.8090E-01	0.3090E-01	-0.3090E-01
	-0.8090E-01	-0.1000	-0.8090E-01	-0.3090E-01	0.3090E-01	0.8090E-01
	0.1000	0.8090E-01	0.3090E-01	-0.3090E-01	-0.8090E-01	-0.1000
	-0.8090E-01	-0.3090E-01	0.3090E-01	0.8090E-01	0.1000	0.8090E-01
	0.3090E-01	-0.3090E-01	-0.8090E-01	-0.1000	-0.8090E-01	-0.3090E-01
	0.3090E-01	0.8090E-01	0.1000	0.8090E-01	0.3090E-01	-0.3090E-01
	-0.8090E-01	-0.1000	-0.8090E-01	-0.3090E-01	0.3090E-01	0.8090E-01
	0.1000	0.8090E-01	0.3090E-01	-0.3090E-01	-0.8090E-01	-0.1000
	-0.8090E-01	-0.3090E-01	0.3090E-01	0.8090E-01	0.1000	0.8090E-01
	0.3090E-01	-0.3090E-01	-0.8090E-01	-0.1000	-0.8090E-01	-0.3090E-01
	0.3090E-01	0.8090E-01	0.1000	0.8090E-01	0.3090E-01	-0.3090E-01
	-0.8090E-01	-0.1000	-0.8090E-01	-0.3090E-01	0.3090E-01	0.8090E-01
	0.0000E+00	-0.5878E-01	-0.9511E-01	-0.9511E-01	-0.5878E-01	0.0000E+00
	0.5878E-01	0.9511E-01	0.9511E-01	0.5878E-01	0.0000E+00	-0.5878E-01
	-0.9511E-01	-0.9511E-01	-0.5878E-01	0.0000E+00	0.5878E-01	0.9511E-01
	0.9511E-01	0.5878E-01	0.0000E+00	-0.5878E-01	-0.9511E-01	-0.9511E-01
	-0.5878E-01	0.0000E+00	0.5878E-01	0.9511E-01	0.9511E-01	0.5878E-01
	0.0000E+00	-0.5878E-01	-0.9511E-01	-0.9511E-01	-0.5878E-01	0.0000E+00
	0.5878E-01	0.9511E-01	0.9511E-01	0.5878E-01	0.0000E+00	-0.5878E-01
	-0.9511E-01	-0.9511E-01	-0.5878E-01	0.0000E+00	0.5878E-01	0.9511E-01
	0.9511E-01	0.5878E-01	0.0000E+00	-0.5878E-01	-0.9511E-01	-0.9511E-01
	-0.5878E-01	0.0000E+00	0.5878E-01	0.9511E-01	0.9511E-01	0.5878E-01
	0.0000E+00	-0.5878E-01	-0.9511E-01	-0.9511E-01	-0.5878E-01	0.0000E+00

0.5878E-01	0.9511E-01	0.9511E-01	0.5878E-01	0.0000E+00	-0.5878E-01
-0.9511E-01	-0.9511E-01	-0.5878E-01	0.0000E+00	0.5878E-01	0.9511E-01
0.9511E-01	0.5878E-01	0.0000E+00	-0.5878E-01	-0.9511E-01	-0.9511E-01
-0.5878E-01	0.0000E+00	0.5878E-01	0.9511E-01	0.9511E-01	0.5878E-01
0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.1250	0.1250
0.1250	0.1250	0.1250	0.1250	0.1250	0.1250
0.1250	0.1250	0.2500	0.2500	0.2500	0.2500
0.2500	0.2500	0.2500	0.2500	0.2500	0.2500
0.3750	0.3750	0.3750	0.3750	0.3750	0.3750
0.3750	0.3750	0.3750	0.3750	0.5000	0.5000
0.5000	0.5000	0.5000	0.5000	0.5000	0.5000
0.5000	0.5000	0.6250	0.6250	0.6250	0.6250
0.6250	0.6250	0.6250	0.6250	0.6250	0.6250
0.7500	0.7500	0.7500	0.7500	0.7500	0.7500
0.7500	0.7500	0.7500	0.7500	0.8750	0.8750
0.8750	0.8750	0.8750	0.8750	0.8750	0.8750
0.8750	0.8750	1.000	1.000	1.000	1.000
1.000	1.000	1.000	1.000	1.000	1.000

0 250

2 1 (.3,.2) 0.0 0.0 (.68,.24) 0.0 0.0 1.0

2 3

3 4

4 5

5 6

6 7

7 8

8 9

9 10

10 1

1 11

1 12

2 12

2 13

3 13

3 14

4 14

4 15

5 15

5 16

6 16

6 17

7 17

7 18

8 18

8 19

9 19

9 20

10 20

10 11

11 12

12 13

13 14

14 15

15 16

16 17

17 18

18 19

19 20

20 11

11 21

11 22

12 22

12 23

13 23

13 24

14	24
14	25
15	25
15	26
16	26
16	27
17	27
17	28
18	28
18	29
19	29
19	30
20	30
20	21
21	22
22	23
23	24
24	25
25	26
26	27
27	28
28	29
29	30
30	21
21	31
21	32
22	32
22	33
23	33
23	34
24	34
24	35
25	35
25	36
26	36
26	37
27	37
27	38
28	38
28	39
29	39
29	40
30	40
30	31
31	32
32	33
33	34
34	35
35	36
36	37
37	38
38	39
39	40
40	31
31	41
31	42
32	42
32	43
33	43
33	44
34	44
34	45
35	45
35	46
36	46
36	47

37	47
37	48
38	48
38	49
39	49
39	50
40	50
40	41
41	42
42	43
43	44
44	45
45	46
46	47
47	48
48	49
49	50
50	41
41	51
41	52
42	52
42	53
43	53
43	54
44	54
44	55
45	55
45	56
46	56
46	57
47	57
47	58
48	58
48	59
49	59
49	60
50	60
50	51
51	52
52	53
53	54
54	55
55	56
56	57
57	58
58	59
59	60
60	51
51	61
51	62
52	62
52	63
53	63
53	64
54	64
54	65
55	65
55	66
56	66
56	67
57	67
57	68
58	68
58	69
59	69
59	70

60	70
60	61
61	62
62	63
63	64
64	65
65	66
66	67
67	68
68	69
69	70
70	61
61	71
61	72
62	72
62	73
63	73
63	74
64	74
64	75
65	75
65	76
66	76
66	77
67	77
67	78
68	78
68	79
69	79
69	80
70	80
70	71
71	72
72	73
73	74
74	75
75	76
76	77
77	78
78	79
79	80
80	71
71	81
71	82
72	82
72	83
73	83
73	84
74	84
74	85
75	85
75	86
76	86
76	87
77	87
77	88
78	88
78	89
79	89
79	90
80	90
80	81
81	82
82	83
83	84
84	85

85 86
86 87
87 88
88 89
89 90
90 81

17 surface 3 endcap at z=1

1

0.10000	0.80900E-01	0.30900E-01	-0.30900E-01	-0.80900E-01	-0.10000
-0.80900E-01	-0.30900E-01	0.30900E-01	0.80900E-01	-0.37300E-01	0.79208E-
-0.45221E-01	0.00000E+00	0.45221E-01	-0.79208E-02	0.37300E-01	
0.00000E+00	0.58780E-01	0.95110E-01	0.95110E-01	0.58780E-01	0.00000E+
-0.58780E-01	-0.95110E-01	-0.95110E-01	-0.58780E-01	0.37876E-01	0.52567E-
-0.14690E-01	0.00000E+00	0.14690E-01	-0.52567E-01	-0.37876E-01	
1.00000E+00	1.00000E+00	1.00000E+00	1.00000E+00	1.00000E+00	1.00000E+
1.00000E+00	1.00000E+00	1.00000E+00	1.00000E+00	1.00000E+00	1.00000E+
1.00000E+00	1.00000E+00	1.00000E+00	1.00000E+00	1.00000E+00	

22	0		
11	13	14	(.3,.2) 0.0 0.0 (.68,.24) 0.0 0.0
12	11	14	
12	14	15	
14	13	16	
14	16	17	
15	14	17	
6	11	5	
6	13	11	
7	13	6	
7	16	13	
8	16	7	
9	16	8	
9	17	16	
10	17	9	
1	17	10	
1	15	17	
2	15	1	
2	12	15	
3	12	2	
4	12	3	
4	11	12	
5	11	4	

0
end
91
0. 1 0. 90.

DISTRIBUTION:

2 Intel Corporation
15201 NW Greenbrier Parkway
Beaverton, OR 97006
Attn: Adrian King

2 MS 1436 4523 C. E. Meyers
4 MS 1109 9224 A. L. Hale
5 MS 0899 ~~0899~~⁴⁹¹⁶ Technical Library (5)
1 MS 9018 8940-2 Central Technical Files
2 MS 0619 12690 Review & Approval Desk (2)
for DOE/OSTI

NOTE