AFIT/GE/ENG/90D-67

TE SCATTERING FROM A DIELECTRIC COATED CONDUCTING STRIP: PROGRAM "PBFSTRIP"

THESIS

William D. Wood, Jr.
Captain, USAF

AFIT/GE/ENG/90D-67

AFIT/GE/ENG/90D-67

TE SCATTERING FROM A DIELECTRIC COATED CONDUCTING STRIP: PROGRAM "PBFSTRIP"

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Electrical Engineering

William D. Wood, Jr., B.S.E.E.

Captain, USAF

December, 1990

Table of Contents

TE SCATTERING FROM A DIELECTRIC COATED CONDUCTING STRIP: PROGRAM "PBFSTRIP"

I. PBFSTRIP Program Description

The physical basis function moment method implementation was coded in standard FORTRAN. The program was written to make maximal use of COMMON blocks to transfer data between program units. Capt W. Irvin contributed the matrix inversion routine, INVERT. Frequency has been scaled out of the program, so all length variables are in units of free-space wavelength λ_0 and $k_0 = 2\pi$.

The program performs the following functions sequentially.

1. Define elementary constants, read geometry and material parameters, calculate the PBF phase constants, partition the dielectric slab into rectangular cells, find the PBF field amplitudes, evaluate the surface integrations and all integrations around the singularity.
2. Read user-specified match points and fill the impedance and voltage matrices.
3. Normalize the over-determined system of equations.
4. Solve the resultant 3×3 system of equations using Gaussian elimination. Write the PBF amplitudes to an output file.
5. Calculate residual errors from the least-squares fit.
6. Calculate and write equivalent currents along the strip and slab.
7. Calculate the bistatic scattering width.

A complete listing of the source code follows.

II. Program PBFSTRIP

```
C*************************************************************************
C                                                                        **
C     >>>>  PROGRAM PBFSTRIP  <<<<                                       **
C                                                                        **
C     THIS PROGRAM CALCULATES THE SCATTERING FROM A DIELECTRIC-          **
C     COATED CONDUCTING STRIP USING A MOMENT METHOD IMPLEMENTATION       **
C     WITH PHYSICAL BASIS FUNCTIONS AND LEAST-SQUARES POINT              **
C     MATCHING.                                                          **
C                                                                        **
C*************************************************************************
C
      PROGRAM PBFSTRIP
C
      IMPLICIT NONE
      INTEGER N
      CHARACTER DATESTRING*9
C
C*************************************************************************
C     SUBROUTINES CALLED:  DEFINE, DATE(INTRINSIC FORTRAN),
C                          IMPEDANCE_MATRIX, ERRORS, FIND_CURRENTS, RADIATE
C     FUNCTIONS CALLED:  NONE
C     COMMON BLOCKS:  ALL, HANKEL, GAUSSIAN_QUADRATURE,
C                     FIELD_AMPLITUDES, SURFACE_INTEGRALS, IMPEDANCE,
C                     MATCH_POINT
C
C     >>>>  INTERNAL VARIABLES  <<<<
C     N   =  IMPLIED DO-LOOP INDEX
C     DATESTRING  =  STRING CONTAINING CURRENT DATE
C
C*************************************************************************
C
C  DEFINE THE VARIABLES IN THE COMMON BLOCKS
C
C  COMMON BLOCK:  ALL -- ELEMENTARY VALUES
C     PI    =  3.141592......
C     CJ    =  IMAGINARY NUMBER OF UNIT AMIPLITUDE
C     EO    =  PERMITTIVITY OF FREE SPACE
C     MUO   =  PERMEABILITY OF FREE SPACE
C     ETA   =  IMPEDANCE OF FREE SPACE
C     KO    =  PHASE CONSTANT OF FREE SPACE, EQUALS 2*PI
C     THETA =  ANGLE OF INCIDENCE, IN RADIANS
C     ER    =  RELATIVE PERMITTIVITY OF DIELECTRIC
C     H     =  THICKNESS OF DIELECTRIC SLAB
C     W     =  WIDTH OF SLAB AND STRIP
C     F     =  3X1 VECTOR CONTAINING Y PHASE CONSTANTS
C     G     =  3X1 VECTOR CONTAINING X PHASE CONSTANTS
C     XNODES=  NUMBER OF NODES IN THE X DIRECTION
```

```
C     YNODES=  NUMBER OF NODES IN THE Y DIRECTION
C     DELX   =  WIDTH OF INTEGRATION CELL
C     DELY   =  HEIGHT OF INTEGRATION CELL
C
C  COMMON BLOCK:  HANKEL -- VALUES FOR HANKEL FUNCTIONS APPROXIMATIONS
C     JO, YO, MAGO, PHASEO, ALPHA, BETA =
C           CONSTANTS USED IN FUNCTION HANKO
C     J1, Y1, MAG1, PHASE1 = CONSTANTS USED IN FUNCTION HANK1
C
C  COMMON BLOCK:  GAUSSIAN_QUADRATURE -- NODES AND WEIGHTS
C     N8       =  8-POINT GAUSSIAN QUADRATURE NODES
C     W8       =  8-POINT GAUSSIAN QUADRATURE WEIGHTS
C     N4       =  4-POINT GAUSSIAN QUADRATURE NODES
C     W4       =  4-POINT GAUSSIAN QUADRATURE WEIGHTS
C
C  COMMON BLOCK:  FIELD_AMPLITUDES -- PBF FIELD AMPLITUDES
C     CX =  3X1 VECTOR OF MULTIPLIERS OF X COMPONENTS OF PBF's
C     CY =  3X1 VECTOR OF MULTIPLIERS OF Y COMPONENTS OF PBF's
C     CZ =  3X1 VECTOR OF MULTIPLIERS OF Z COMPONENTS OF PBF's
C
C  COMMON BLOCK:  SURFACE_INTEGRALS -- PRE-CALCULATED SURFACE INTEGRALS
C                                    AND INTEGRALS ABOUT THE SINGULARITY
C     SURFINT(I,J,N,K) = MATRIX CONTAINING SURFACE INTEGRATION OVER
C           DIELECTRIC CELL WITH LOWER RIGHT CORNER AT (I*DELX,J*DELY),
C           WHERE INTEGRAND INVOLVES Nth PBF AND EITHER COSINE (K=1) OR
C           SINE (K=2).
C     SINGINT(K,N) = MATRIX CONTAINING INTEGRATIONS ABOUT THE
C           SINGULARITY.  SEE SUBROUTINE SINGULAR_INTEGRAL FOR MEANING
C           OF K AND N.
C
C  COMMON BLOCK:  IMPEDANCE -- SYSTEM OF EQUATIONS
C     Z(M,N)  =  MATRIX CONTAINING UNNORMALIZED IMPEDANCE MATRIX
C                ELEMENT FOR Mth MATCH POINT AND Nth PBF.
C     V(M)    =  VECTOR CONTAINING UNNORMALIZED VOLTAGE MATRIX ELEMENT
C                FOR Mth MATCH POINT.
C     ZN(M,N) =  3X3 MATRIX CONTAINING NORMALIZED IMPEDANCE MATRIX
C     VN(N)   =  VECTOR CONTAINING NORMALIZED VOLTAGE MATRIX
C     CUR(N)  =  VECTOR CONTAINING PBF AMPLITUDES
C     CN      =  CONDITION NUMBER OF ZN
C     NUMMPS  =  TOTAL NUMBER OF MATCH POINTS
C
C  COMMON BLOCK:  MATCH_POINT -- VALUES ASSOCIATED WITH CURRENT MATCH
C                                POINT
C     I     =  MATCH POINT X-COORDINATE INDEX
C     J     =  MATCH POINT Y-COORDINATE INDEX
C     X     =  MATCH POINT X-COORDINATE ( = I*DELX)
C     Y     =  MATCH POINT Y-COORDINATE ( = J*DELY)
C     HO2PY =  H OVER 2, PLUS Y
C     HO2PY2=  HO2PY SQUARED
C     HO2MY =  H OVER 2, MINUS Y
C     HO2MY2=  HO2MY SQUARED
```

```
C     W02PX =  W OVER 2, PLUS X
C     W02PX2=  W02PX SQUARED
C     W02MX =  W OVER 2, MINUS X
C     W02MX2=  W02MX SQUARED
C
C --- C O M M O N   B L O C K S ----------------------------------
C
        COMPLEX*16 CJ
        REAL*8 PI, E0, MU0,ETA,K0,THETA,ER,H,W,F(3),G(3),DELX,DELY
        INTEGER XNODES, YNODES
      COMMON / ALL / PI, CJ, E0, MU0, ETA, K0, THETA, ER, H, W, F, G,
     &               XNODES, YNODES, DELX, DELY
C
        REAL*8 J0(0:6), Y0(0:6), BETA(0:6), MAG0(0:6), PHASE0(0:6)
        COMPLEX*16 ALPHA(0:6)
        REAL*8 J1(0:6), Y1(0:6), MAG1(0:6), PHASE1(0:6)
      COMMON / HANKEL / J0, Y0, ALPHA, BETA, MAG0, PHASE0,
     &                  J1, Y1, MAG1, PHASE1
C
        REAL*8 W8(1:8), N8(1:8), W4(1:4), N4(1:4)
      COMMON / GAUSSIAN_QUADRATURE / W8, N8, w4, n4
C
        COMPLEX*16 CX(3), CY(3), CZ(3)
      COMMON / FIELD_AMPLITUDES / CX, CY, CZ
C
        COMPLEX*16 SURFINT(-49:50,-5:5,1:3,1:2), SINGINT(4,3)
      COMMON / SURFACE_INTEGRALS / SURFINT, SINGINT
C
        INTEGER NUMMPS
        REAL*8 CN
        COMPLEX*16 Z(40,3), V(40), ZN(3,3), VN(3), CUR(3)
      COMMON / IMPEDANCE / Z, V , ZN, VN, CUR, CN, NUMMPS
C
        INTEGER I, J
        REAL*8 X, Y, H02PY, H02PY2, H02MY, H02MY2,
     &           W02PX, W02PX2, W02MX, W02MX2
      COMMON / MATCH_POINT / I, J, X, Y, H02PY, H02PY2, H02MY, H02MY2,
     &                       W02PX, W02PX2, W02MX, W02MX2
C
C  OPEN FILES FOR INPUT AND OUTPUT
C
      OPEN (UNIT=19, FILE='PBFSTRIP.IN ', STATUS='OLD')
      OPEN (UNIT=20, FILE='PBFSTRIP.OUT', STATUS='NEW')
      OPEN (UNIT=31, FILE='PBFSTRIP_CUR.PL1', STATUS='NEW',
     &      FORM='UNFORMATTED')
      OPEN (UNIT=22, FILE='PBFSTRIP.FLD', STATUS='NEW')
      OPEN (UNIT=32, FILE='PBFSTRIP_FLD.PL1', STATUS='NEW',
     &      FORM='UNFORMATTED')
      CALL DEFINE
C
C  WRITE HEADER INFORMATION TO THE OUTPUT FILES
```

```
C
      CALL DATE (DATESTRING)
      WRITE (20,98) ER, DATESTRING, W, H, XNODES, YNODES,
     &     THETA*180/PI,(G(N), F(N), N=1,3)
   98 FORMAT (' Relative Permittivity = ', F10.4, 10X, A9/
     &        ' Slab Width     = ', F10.4,
     &        ' Slab Thickness = ', F10.4/
     &        ' Nodes (horizontal) =', I3, ' (vertical)   =', I2/
     &        ' Angle from Normal     = ', F10.4, ' deg'/
     &        ' Phase Constants: ', 3('(',F7.3,',',F7.3,')') )
      WRITE (21,99) ER, DATESTRING, W, H, THETA*180/PI
      WRITE (22,99) ER, DATESTRING, W, H, THETA*180/PI
   99 FORMAT (' Relative Permittivity = ', F10.4, 10X, A9/
     &        ' Slab Width = ', F7.2, ' Slab Thickness = ', F7.3,
     &        ' Angle from Normal = ', F7.3 )
C
C  CALCULATE THE PHYSICAL BASIS FUNCTION AMPLITUDES
C
      CALL IMPEDANCE_MATRIX
C
C  CALCULATE THE MEAN-SQUARE ERRORS, EQUIVALENT CURRENTS, AND FAR-ZONE
C  SCATTERED FIELDS.
C
      CALL ERRORS
      CALL FIND_CURRENTS
      CALL RADIATE
C
C  CLOSE INPUT AND OUTPUT FILES
C
      CLOSE (19)
      CLOSE (21)
      CLOSE (22)
      CLOSE (31)
      CLOSE (32)
C
      STOP
      END
```

2.1 Subroutine DEFINE

```
C*************************************************************************
C                                                                        **
C     >>>>  SUBROUTINE DEFINE  <<<<                                      **
C                                                                        **
C     THIS SUBROUTINE INITIALIZES THE CONSTANTS IN COMMON                **
C     BLOCKS ALL, HANKEL, GAUSSIAN_QUADRATURE,                           **
C     FIELD_AMPLITUDES, AND SURFACE_INTEGRALS.                           **
C                                                                        **
C*************************************************************************
C     CALLED BY:  MAIN
C     SUBROUTINES CALLED:  PHASE_CONSTANTS, MATRIX_FILL,
C                          DEFINE_IMNABC
C     FUNCTIONS CALLED:  NONE
C     COMMON BLOCKS:  ALL, HANKEL, GAUSSIAN_QUADRATURE,
C                     FIELD_AMPLITUDES, SURFACE_INTEGRALS
C
C     >>>>  INTERNAL VARIABLES  <<<<
C     THETA_DEG  =  ANGLE OF INCIDENCE, IN DEGREES
C     SAMPLE  =  NUMBER OF NODES PER FREE-SPACE WAVELENGTH
C     I  =  DO-LOOP INDEX
C
C     >>>>  DATA INPUT FROM CALLING ROUTINE  <<<<
C     NONE
C
C     >>>>  DATA OUTPUT   <<<<
C     ALL DATA IS OUTPUT VIA THE COMMON BLOCKS
C
C*************************************************************************
      SUBROUTINE DEFINE
C
      IMPLICIT NONE
      REAL*8 THETA_DEG, SAMPLE
      INTEGER I
C
C --- C O M M O N   B L O C K S ---------------------------------
C
         COMPLEX*16 CJ
         REAL*8 PI, EO, MUO,ETA,KO,THETA,ER,H,W,F(3),G(3),DELX,DELY
         INTEGER XNODES, YNODES
      COMMON / ALL / PI, CJ, EO, MUO, ETA, KO, THETA, ER, H, W, F, G,
     &               XNODES, YNODES, DELX, DELY
C
         REAL*8 JO(0:6), YO(0:6), BETA(0:6), MAGO(0:6), PHASEO(0:6)
         COMPLEX*16 ALPHA(0:6)
         REAL*8 J1(0:6), Y1(0:6), MAG1(0:6), PHASE1(0:6)
      COMMON / HANKEL / JO, YO, ALPHA, BETA, MAGO, PHASEO,
     &                  J1, Y1, MAG1, PHASE1
C
```

```
         REAL*8 W8(1:8), N8(1:8), W4(1:4), N4(1:4)
      COMMON / GAUSSIAN_QUADRATURE / W8, N8, w4, n4
C
         COMPLEX*16 CX(3), CY(3), CZ(3)
      COMMON / FIELD_AMPLITUDES / CX, CY, CZ
C
         COMPLEX*16 SURFINT(-49:50,-5:5,1:3,1:2), SINGINT(4,3)
      COMMON / SURFACE_INTEGRALS / SURFINT, SINGINT
C
C
C  DEFINE SOME ELEMENTARY CONSTANTS
C
      PI=3.14159 26535 89793 D0
      CJ=(0.D0, 1.D0)
      E0= 8.85418 53368D-12
      K0=2.D0*PI
      MU0=1.25663 70614D-06
      ETA=DSQRT(MU0/E0)
C
C  READ IN CASE-SPECIFIC PARAMETERS
C
         READ (19,56) THETA_DEG, ER, H, W, SAMPLE
   56    FORMAT (T30, F15.8)
C
C  CONVERT THETA_DEG TO RADIANS
C
      THETA=THETA_DEG*PI/180.D0
C
C  DEFINE NODES SUCH THAT THE SEPARATION BETWEEN THEM IS
C  LESS THAN ONE WAVELENGTH IN THE DIELECTRIC.  ENSURE AT LEAST
C  ONE LAYER OF NODES ON THE CONDUCTOR AND ONE IN THE DIELECTRIC.
C
      XNODES=INT(NP_PER_WAVELENGTH*W)-1
      YNODES=INT(NP_PER_WAVELENGTH*H)
      IF (YNODES.LT.2) YNODES=2
C
C  CALCULATE DELX AND DELY BASED ON XNODES AND YNODES.  ENSURE THAT
C  DELY IS LESS THAN DELX.
C
      DELX=W/(XNODES+1)
   10 DELY=H/YNODES
      IF (DELY.GE.DELX) THEN
         YNODES=YNODES+1
         GOTO 10
      ENDIF
C
C  CHECK IF TOO MANY NODES FOR THE DIMENSIONALITY OF SURFINT.
C
      IF (XNODES.GT.50) THEN
         WRITE (6,*) 'Strip too wide in subroutine DEFINE'
         STOP
```

```
      ELSEIF (YMODES.GT.5) THEN
         WRITE (6,*) 'Dielectric slab too thick in subroutine DEFINE'
         STOP
      ENDIF
C
C  DEFINE THE PHASE CONSTANTS F AND G
C
      CALL PHASE_CONSTANTS
C
C  J0, Y0, MAG0, PHASE0,J1, Y1, MAG1, AND PHASE1 ARE DEFINED IN THE
C  HANKEL_COEFFICIENTS BLOCK DATA SUBROUTINE.  CALCULATE ALPHA AND
C  BETA BASED ON THEM.
C
      DO 15 I=0,6
         BETA(I)=2.D0*J0(I)/PI
   15    ALPHA(I)=J0(I)+CJ*(BETA(I)*DLOG(2.D0)-Y0(I))
C
C  GAUSSIAN QUADRATURE WEIGHTS AND NODES DEFINED IN
C  GAUSSIAN_QUADRATURE BLOCK DATA SUBROUTINE.
C  DEFINE THE FIELD AMPLITUDES, CX, CY, AND CZ
C
      CX(1)=CJ*DSQRT(1.D0-DSIN(THETA)**2/ER)
      CY(1)=DSIN(THETA)/DSQRT(ER)
      CZ(1)=DSQRT(ER*E0/MU0)
      CX(2)=F(2)*ETA/(CJ*ER*K0)
      CY(2)=-G(2)*ETA/(K0*ER)
      CZ(2)=-1.D0
      CX(3)=F(3)*ETA/(CJ*ER*K0)
      CY(3)=-G(3)*ETA/(K0*ER)
      CZ(3)=-1.D0
C
C  CALCULATE THE ELEMENTS OF SURFINT.
C
         CALL MATRIX_FILL
C
C  CALL SINGULAR_INTEGRAL TO CALCULATE THE INTEGRATIONS AROUND THE
C  SINGULAR POINT.
C
      CALL SINGULAR_INTEGRAL
C
      RETURN
      END
```

2.1.1 *Block Data Subroutine HANKEL_COEFFICIENTS*

```
C*********************************************************************
C                                                                    **
C     >>>>  BLOCK DATA SUBROUTINE HANKEL_COEFFICIENTS   <<<<         **
C                                                                    **
C     THIS SUBROUTINE INITIALIZES CONSTANTS USED IN FUNCTION         **
C     HANKO.  VALUES COME FROM AMS-55, EQUATIONS 9.4.1 THROUGH       **
C     9.4.6.                                                         **
C                                                                    **
C*********************************************************************
      BLOCK DATA HANKEL_COEFFICIENTS
      REAL*8 J0(0:6), Y0(0:6), BETA(0:6), MAG0(0:6), PHASE0(0:6)
      COMPLEX*16 ALPHA(0:6)
      REAL*8 J1(0:6), Y1(0:6), MAG1(0:6), PHASE1(0:6)
      COMMON / HANKEL / J0, Y0, ALPHA, BETA, MAG0, PHASE0,
     &                  J1, Y1, MAG1, PHASE1
      DATA (J0(I), I=0,6) / +1.0000000D0,-2.2499997D0,+1.2656208D0,
     &         -0.3163866D0,+0.0444479D0,-0.0039444D0,+0.0002100D0/
      DATA (Y0(I), I=0,6) /+0.36746691D0,+0.60559366D0,-0.74350384D0,
     &       +0.25300117D0,-0.04261214D0,+0.00427916D0,-0.00024846D0/
      DATA (MAG0(I), I=0,6) /+0.79788456D0,-0.00000077D0,-0.00552740D0,
     &         -0.00009512D0,+0.00137237D0,-0.00072805D0,+0.00014476D0/
      DATA(PHASE0(I), I=0,6)/-0.78539816D0,-0.04166397D0,-0.00003954D0,
     &         +0.00262573D0,-0.00054125D0,-0.00029333D0,+0.00013558D0/
      DATA (J1(I), I=0,6) / +0.5000000D0,-0.56249985D0,+0.21093573D0,
     &        -0.03954289D0,+0.00443319D0,-0.00031761D0,+0.00001109D0/
      DATA (Y1(I), I=0,6) / -0.6366198D0,+0.2212091D0,+2.1682709D0,
     &         -1.3164827D0,+0.3123951D0,-0.0400976D0,+0.0027873D0/
      DATA (MAG1(I), I=0,6)/ +0.79788456D0,+0.00000156D0,+0.01659667D0,
     &         +0.00017105D0,-0.00249511D0,+0.00113653D0,-0.00020033D0/
      DATA(PHASE1(I), I=0,6)/-2.35619449D0,+0.12499612D0,+0.00005650D0,
     &         -0.00637879D0,+0.00074348D0,+0.00079824D0,-0.00029166D0/
      END
```

2.1.2 *Block Data Subroutine GAUSSIAN_QUADRATURE*

```
C*************************************************************************
C                                                                        **
C     >>>>  BLOCK DATA SUBROUTINE GAUSSIAN_QUADRATURE  <<<<               **
C                                                                        **
C     THIS SUBROUTINE INITIALIZES WEIGHTS AND NODES FOR 8-POINT          **
C     GAUSSIAN QUADRATURE INTEGRATION.  VALUES COME FROM AMS-55,         **
C     TABLE 25.4                                                         **
C                                                                        **
C*************************************************************************
      BLOCK DATA GAUSSIAN_QUADRATURE
      REAL*8 W8(1:8), N8(1:8), W4(1:4), N4(1:4)
      COMMON / GAUSSIAN_QUADRATURE / W8, N8, W4, N4
      DATA (W8(I), I=1,8) /
     &     0.10122 85362 90376D0, 0.22238 10344 53374D0,
     &     0.31370 66458 77887D0, 0.36268 37833 78362D0,
     &     0.36268 37833 78362D0, 0.31370 66458 77887D0,
     &     0.22238 10344 53374D0, 0.10122 85362 90376D0/
      DATA (N8(I), I=1,8) /
     &    -0.96028 98564 97536D0,-0.79666 64774 13627D0,
     &    -0.52553 24099 16329D0,-0.18343 46424 95650D0,
     &     0.18343 46424 95650D0, 0.52553 24099 16329D0,
     &     0.79666 64774 13627D0, 0.96028 98564 97536D0/
      DATA (W4(I), I=1,4) /
     &     0.34785 48451 37454D0, 0.65214 51548 62546D0,
     &     0.65214 51548 62546D0, 0.34785 48451 37454D0/
      DATA (N4(I), I=1,4) /
     &    -0.86113 63115 94053D0,-0.33998 10435 84856D0,
     &     0.33998 10435 84856D0, 0.86113 63115 94053D0/
      END
```

2.1.3 *Subroutine PHASE_CONSTANTS*

```
C*********************************************************************
C                                                                    **
C     >>>>  SUBROUTINE PHASE_CONSTANTS  <<<<                         **
C                                                                    **
C     THIS SUBROUTINE CALCULATES THE PHASE CONSTANTS OF THE          **
C     THREE PHYSICAL BASIS FUNCTIONS.  F IS THE PHASE CONSTANT       **
C     IN THE Y DIRECTION AND G IS THE PHASE CONSTANT IN THE          **
C     X DIRECTION.  M=1 IS THE FORCED WAVE, M=2 IS THE FORWARD       **
C     SURFACE WAVE, AND M=3 IS THE REVERSE SURFACE WAVE.  THE        **
C     NEWTON-RAPHSON ROOT-FINDING ALGORITHM IS USED TO FIND          **
C     F(2), WHICH IS THE ROOT OF THE TRANSCENDENTAL EQUATION         **
C     F(2)*TAN(F(2)*H)-ER*SQRT((ER-1)*K0**2-F(2)**2)=0, WHERE        **
C     F(2) LIES IN THE INTERVAL (0, PI/(2*H)).                       **
C                                                                    **
C*********************************************************************
C     CALLED BY:  DEFINE
C     SUBROUTINES CALLED:  NONE
C     FUNCTIONS CALLED:  NONE
C     COMMON BLOCKS:  ALL
C
C     >>>>  INTERNAL VARIABLES  <<<<
C     TOL   =  THRESHOLD FOR CONVERGENCE OF NEWTON-RAPHSON ALGORITHM
C     SLOPE =  DERIVATIVE OF TRANSCENDENTAL EQUATION
C     OLDX  =  OLD GUESS FOR F(2), USED TO FIND NEW GUESS FOR F(2)
C     OLDY  =  VALUE OF TRANSCENDENTAL EQUATION EVALUATED AT OLDX
C     I     =  ITERATION COUNTER IN NEWTON-RAPHSON ALGORITHM
C
C     >>>>  DATA OUTPUT   <<<<
C     F AND G VECTORS RETURNED VIA COMMON BLOCK "ALL"
C*********************************************************************
      SUBROUTINE PHASE_CONSTANTS
C
      IMPLICIT NONE
      REAL*8 TOL, OLDX, OLDY, SLOPE
      INTEGER I
C
C --- C O M M O N   B L O C K S ---------------------------------
C
         COMPLEX*16 CJ
         REAL*8 PI, E0, MU0,ETA,K0,THETA,ER,H,W,F(3),G(3),DELX,DELY
         INTEGER XNODES, YNODES
      COMMON / ALL / PI, CJ, E0, MU0, ETA, K0, THETA, ER, H, W, F, G,
     &               XNODES, YNODES, DELX, DELY
C
C
C DEFINE THE PHASE CONSTANTS OF THE FORCED WAVE
C
      F(1) = K0*DSQRT(ER-DSIN(THETA)**2)
      G(1) = K0*DSIN(THETA)
```

```
C
C  DEFINE TOL, WHICH CONTROLS NEWTON-RAPHSON CONVERGENCE.  IF ER IS
C  VERY SMALL, WE CAN DEFINE F AND G EXPLICITLY.
C
      TOL = 1.E-08
      IF (ER-1.LT.TOL) THEN
         F(2)=0.0
         F(3)=0.0
         G(2)=KO
         G(3)=-KO
         RETURN
      ENDIF
C
C  MAKE A FIRST GUESS AT F(2) BASED ON ITS MAXIMUM POSSIBLE VALUE
C
      F(2)=MIN(DSQRT(ER-1.)*KO, 0.99*PI/(2.*H) )
C
C  USE THE NEWTON-RAPHSON METHOD TO ITERATIVELY FIND F(2).
C  INITIALIZE COUNTER I.
C
      I=0
   10 I=I+1
C
C  TRANSITION THE CURRENT GUESS TO OLDX, AND EVALUATE THE
C  TRANSCENDENTAL EQUATION AS OLDY.  ALSO, FIND THE DERIVATIVE
C  OF THE TRANSCENDENTAL EQUATION AS SLOPE.
C
      OLDX = F(2)
      OLDY = OLDX*OLDX*TAN(OLDX*H)*TAN(OLDX*H) -
     &          ER*ER*((ER-1.)*KO*KO-OLDX*OLDX)
      SLOPE = 2*OLDX*TAN(OLDX*H)*TAN(OLDX*H)
     &       + 2*OLDX*OLDX*H*TAN(OLDX*H)/COS(OLDX*H)**2
     &       + 2*ER*ER*OLDX
C
C  CALCULATE THE NEW GUESS, F(2), ACCORDING TO NEWTON-RAPHSON
C
      F(2) = MIN(OLDX - OLDY/SLOPE, KO*DSQRT(ER-1.))
C
C  IF THE NEW GUESS LIES OUTSIDE THE POSSIBLE RANGE OF VALUES, THEN
C  SOMETHING IS DREADFULLY WRONG.  RETURN AND FLAG THE ERROR.
C
      IF ((F(2).LE.0.).OR.(F(2).GT.PI/(2.*H))) THEN
         WRITE (6,*) 'ERROR IN SUBROUTINE "COEF"'
         RETURN
      ENDIF
C
C  TEST FOR CONVERGENCE.  IF CONVERGENCE IS NOT REACHED, ITERATE AGAIN.
C  ONLY ITERATE 50 TIMES
C
      IF ((DABS((F(2)-OLDX)/OLDX).GT.TOL).AND.(I.LT.50)) GOTO 10
C
```

```
C  CHECK TO SEE IF MAXIMUM NUMBER OF ITERATIONS HAS BEEN EXCEEDED
C
      IF (I.GE.50) THEN
         WRITE (6,*) 'Non-convergence in SUBROUTINE COEF'
         RETURN
      ENDIF
C
C  GENERATE F(3), G(2), AND G(3) BASED ON F(2)
C
      F(3) = F(2)
      G(2) = DSQRT(ER*KO*KO - F(2)*F(2))
      G(3) = -G(2)
C
      RETURN
      END
```

2.1.4 *Subroutine MATRIX_FILL*

```
C*********************************************************************
C                                                                    **
C          >>>>   SUBROUTINE MATRIX_FILL   <<<<                      **
C                                                                    **
C     THIS SUBROUTINE FILLS THE SURFINT MATRIX.  EACH ELEMENT        **
C     CONTAINS THE INTEGRATION OF A FUNCTION OVER A RECTANGULAR      **
C     CELL OF DIMENSION DELX-by-DELY.  THE LOWER-RIGHT CORNER OF     **
C     THE CELL IS LOCATED AT (I*DELX, J*DELY).  THE FUNCTION         **
C     INTEGRATED DEPENDS ON M AND K.                                 **
C                                                                    **
C*********************************************************************
C     >>>>  CALLED BY:  DEFINE
C     >>>>  SUBROUTINES CALLED:  MATRIX_ELEMENT, CORNERS
C     >>>>  FUNCTIONS CALLED:  NONE
C     >>>>  COMMON BLOCKS USED:  ALL, SURFACE_INTEGRALS
C
C     >>>>  DATA FROM MATRIX_ELEMENT  <<<<
C     SURFINT(I,J,1:3,1:2) WHERE (I,J) DEFINE A CELL NOT TOUCHING
C     THE SINGULAR POINT
C
C     >>>>  DATA FROM CORNERS  <<<<
C     SURFINT(I,J,1:3,1:2) WHERE (I,J) DEFINE A CELL TOUCHING
C     THE SINGULAR POINT
C
C     >>>>  INTERNAL VARIABLES   <<<<
C     I, J  =  DEFINE COORDINATES OF LOWER-RIGHT CORNER OF
C              INTEGRATION CELL
C     M     =  PHYSICAL BASIS FUNCTION INDEX
C     K     =  INTEGRATION FUNCTION INDEX, DEFINES TRIG_TERM
C
C     >>>>  DATA OUTPUT TO CALLING ROUTINE  <<<<
C     INT   =  MATRIX CONTAINING INTEGRATIONS OF (I,J) CELL FOR THREE
C              PHYSICAL BASIS FUNCTIONS AND TWO TRIGONOMETRIC FUNCTIONS
C
C*********************************************************************
      SUBROUTINE MATRIX_FILL
C
      IMPLICIT NONE
      INTEGER I, J, M
C
C --- C O M M O N   B L O C K S ---------------------------------
C
         COMPLEX*16 CJ
         REAL*8 PI, EO, MUO,ETA,KO,THETA,ER,H,W,F(3),G(3),DELX,DELY
         INTEGER XNODES, YNODES
      COMMON / ALL / PI, CJ, EO, MUO, ETA, KO, THETA, ER, H, W, F, G,
     &               XNODES, YNODES, DELX, DELY
C
         COMPLEX*16 SURFINT(-49:50,-5:5,1:3,1:2), SINGINT(4,3)
```

```
      COMMON / SURFACE_INTEGRALS / SURFINT, SINGINT
C
C  CALCULATE THE ELEMENTS WITH CELLS TOUCHING THE SINGULAR POINT
C
      CALL CORNERS
C
C  FILL ELEMENTS WITH CELLS NOT TOUCHING THE SINGULAR POINT.
C  START THE I AND J LOOPS
C
      DO 10 I = 1-XNODES, XNODES
         DO 10 J = 0, YNODES-1
            CALL MATRIX_ELEMENT(I, J)
C
C  EXPLOIT SYMMETRY BETWEEN THE (I,J,N,K) AND (I,-1-J,N,K) ELEMENTS
C
            DO 10 N=1,3
               SURFINT(I,-1-J,N,1)=-SURFINT(I,J,N,1)
               SURFINT(I,-1-J,N,2)= SURFINT(I,J,N,2)
   10       CONTINUE
C
      RETURN
      END
```

2.1.4.1 *Subroutine MATRIX_ELEMENT*

```
C*************************************************************************
C                                                                        **
C            >>>>   SUBROUTINE MATRIX_ELEMENT   <<<<                     **
C                                                                        **
C     THIS SUBROUTINE CALCULATES THE ELEMENTS OF                         **
C     SURFINT(I,J,M,K) THAT ARE DEFINED BY THE CELL REFERENCE            **
C     POINT COORDINATES (I,J).  THE CELL CANNOT DIRECTLY TOUCH THE       **
C     SINGULAR POINT.  INTEGRATION IS DONE USING 8-POINT                 **
C     GAUSSIAN QUADRATURE.                                               **
C                                                                        **
C*************************************************************************
C     >>>>  CALLED BY:  MATRIX_FILL
C     >>>>  SUBROUTINES CALLED:  NONE
C     >>>>  FUNCTIONS CALLED:  HANKO
C     >>>>  COMMON BLOCKS:  ALL, GAUSSIAN_QUADRATURE, SURFACE_INTEGRALS
C
C     >>>>  INTERNAL VARIABLES   <<<<
C     U, V  =  COORDINATES OF POINT AT WHICH FUNCTION IS EVALUATED
C              DURING GAUSSIAN QUADRATURE
C     TRIG_TERM   =  PART OF FUNCTION INTEGRATED
C     HANK_TERM   =  PART OF FUNCTION INTEGRATED
C     EXP_TERM    =  PART OF FUNCTION INTEGRATED
C     TERM  =  TEMPORARY STORAGE MATRIX OF INNER INTEGRATIONS
C     M     =  PHYSICAL BASIS FUNCTION INDEX
C     K     =  INTEGRATION FUNCTION INDEX, DEFINES TRIG_TERM
C     II    =  INNER GAUSSIAN QUADRATURE INDEX
C     JJ    =  OUTER GAUSSIAN QUADRATURE INDEX
C
C     >>>>  DATA INPUT FROM CALLING ROUTINE  <<<<
C     I, J  =  COORDINATES OF RECTANGULAR CELL
C
C     >>>>  DATA OUTPUT TO CALLING ROUTINE  <<<<
C     SURFINT(I,J,M,K) WHERE (I,J) DEFINE A CELL NOT TOUCHING
C     THE SINGULAR POINT
C
C*************************************************************************
      SUBROUTINE MATRIX_ELEMENT (I, J)
C
      IMPLICIT NONE
      REAL*8 U, V, TRIG_TERM
      COMPLEX*16 HANKO, TERM(1:3,1:2), EXP_TERM, HANK_TERM
      INTEGER I, J, M, K, II, JJ
C
C --- C O M M O N   B L O C K S ---------------------------------
C
         COMPLEX*16 CJ
         REAL*8 PI, EO, MUO,ETA,KO,THETA,ER,H,W,F(3),G(3),DELX,DELY
         INTEGER XNODES, YNODES
      COMMON / ALL / PI, CJ, EO, MUO, ETA, KO, THETA, ER, H, W, F, G,
```

```
     &                XNODES, YNODES, DELX, DELY
C
         REAL*8 W8(1:8), N8(1:8), W4(1:4), N4(1:4)
       COMMON / GAUSSIAN_QUADRATURE / W8, N8, w4, n4
C
         COMPLEX*16 SURFINT(-49:50,-5:5,1:3,1:2), SINGINT(4,3)
       COMMON / SURFACE_INTEGRALS / SURFINT, SINGINT
C
C  IF (I,J) DEFINE A CELL TOUCHING THE SINGULAR POINT, RETURN
C
         IF ( ((I.EQ.0).AND.(J.EQ. 0)) .OR.
     &        ((I.EQ.0).AND.(J.EQ.-1)) .OR.
     &        ((I.EQ.1).AND.(J.EQ. 0)) .OR.
     &        ((I.EQ.1).AND.(J.EQ.-1)) ) RETURN
C
C  INITALIZE OUTPUTS TO ZERO
C
       DO 5 N=1,3
       DO 5 K=1,2
    5    SURFINT(I,J,N,K)=(0.,0.)
C
C  START THE OUTER INTEGRATION LOOP
C
       DO 50 JJ=1,8
          V=(N8(JJ)+2.D0*J+1.D0)*DELY/2.D0
C
C  INITIALIZE THE INNER LOOP STORAGE MATRIX ELEMENTS TO ZERO
C
         DO 10 N=1,3
         DO 10 K=1,2
  10        TERM(N,K)=(0.,0.)
C
C  START INNER INTEGRATION LOOP
C
         DO 20 II=1,8
C
C  DEFINE U AND HANK_TERM
C
            U=(N8(II)+2.D0*I-1.D0)*DELX/2.D0
            HANK_TERM=HANKO(KO*DSQRT(U*U+V*V))
C
C  START PHYSICAL BASIS FUNCTION LOOP AND DEFINE EXP_TERM
C
            DO 20 N=1,3
               EXP_TERM=EXP(-CJ*G(N)*U)
C
C  START K LOOP AND DEFINE TRIG_TERM
C
            DO 20 K=1,2
               IF (K.EQ.1) THEN
                  TRIG_TERM=DSIN(F(N)*V)
```

```
               ELSE
                  TRIG_TERM=DCOS(F(N)*V)
               ENDIF
C
C  ADD UP THE WEIGHTED FUNCTION EVALUATIONS
C
               TERM(N,K)=TERM(N,K)+W8(II)*
     &                    TRIG_TERM*EXP_TERM*HANK_TERM
C
C  END PHYSICAL BASIS FUNCTION AND K LOOPS
C
   20       CONTINUE
C
C  ADD WEIGHTED INNER INTEGRATION MATRIX TO OUTER INTEGRATION MATRIX
C
         DO 40 N=1,3
         DO 40 K=1,2
   40       SURFINT(I,J,N,K) = SURFINT(I,J,N,K) + TERM(N,K)*W8(JJ)
C
C  END OUTER INTEGRATION LOOP
C
   50 CONTINUE
C
C  PERFORM FINAL GAUSSIAN QUADRATURE MULTIPLICATION
C
      DO 60 N=1,3
      DO 60 K=1,2
   60    SURFINT(I,J,N,K)=SURFINT(I,J,N,K)*DELX*DELY/4.

      RETURN
      END
```

2.1.4.2 Subroutine CORNERS

```
C*********************************************************************
C                                                                     **
C            >>>>   SUBROUTINE CORNERS    <<<<                      **
C                                                                     **
C     THIS SUBROUTINE CALCULATES THE ELEMENTS OF                      **
C     SURFINT(I,J,M,K) THAT DIRECTLY TOUCH THE SINGULAR               **
C     POINT.  THESE ELEMENTS ARE GIVEN BY (I,J)=(0,0), (0,-1),        **
C     (1,0), AND (1,-1).  FOR EACH CELL, THE INTEGRATION OCCURS       **
C     OVER THE NORMAL RECTANGULAR CELL MINUS A QUARTER-CIRCULAR       **
C     AREA CENTERED AT THE CORNER OF THE CELL THAT COINCIDES          **
C     WITH THE SINGULAR ORIGIN.                                       **
C                                                                     **
C*********************************************************************
C     >>>>  CALLED BY:  MATRIX_FILL
C     >>>>  SUBROUTINES CALLED:  NONE
C     >>>>  FUNCTIONS CALLED:  HANKO
C     >>>>  COMMON BLOCKS:  ALL, GAUSSIAN_QUADRATURE, SURFACE_INTEGRALS
C
C     >>>>  INTERNAL VARIABLES   <<<<
C     R     =  RADIUS OF CIRCLE AROUND SINGULAR POINT
C     U, V  =  COORDINATES OF POINT AT WHICH FUNCTION IS EVALUATED
C              DURING GAUSSIAN QUADRATURE
C     X     =  LIMIT OF INTEGRATION DEFINED BY CIRCLE AROUND SINGULAR
C              POINT
C     SPAN  =  LENGTH OF INTEGRATION INTERVAL IN U-DIMENSION
C     TRIG_TERM   =  PART OF FUNCTION INTEGRATED
C     HANK_TERM   =  PART OF FUNCTION INTEGRATED
C     EXP_TERM    =  PART OF FUNCTION INTEGRATED
C     TERM  =  TEMPORARY STORAGE MATRIX OF INNER INTEGRATIONS
C     I, J  =  DEFINE COORDINATE OF LOWER-RIGHT CORNER OF
C              INTEGRATION CELL
C     M     =  PHYSICAL BASIS FUNCTION INDEX
C     K     =  INTEGRATION FUNCTION INDEX, DEFINES TRIG_TERM
C     II    =  INNER GAUSSIAN QUADRATURE INDEX
C     JJ    =  OUTER GAUSSIAN QUADRATURE INDEX
C
C     >>>>  DATA OUTPUT TO CALLING ROUTINE  <<<<
C     SURFINT(I,J,M,K) WHERE (I,J) = (0,0), (0.-1), (1.0), OR (1,-1)
C
C*********************************************************************
      SUBROUTINE CORNERS
C
      IMPLICIT NONE
      REAL*8 R, U, V, X, SPAN, TRIG_TERM
      COMPLEX*16 HANK_TERM, HANKO, EXP_TERM, TERM(0:1,-1:0,1:3,1:2)
      INTEGER I, J, M, K, II, JJ
C
C --- C O M M O N   B L O C K S ----------------------------------
C
```

```
      COMPLEX*16 CJ
      REAL*8 PI, EO, MUO,ETA,KO,THETA,ER,H,W,F(3),G(3),DELX,DELY
      INTEGER XNODES, YNODES
      COMMON / ALL / PI, CJ, EO, MUO, ETA, KO, THETA, ER, H, W, F, G,
     &               XNODES, YNODES, DELX, DELY
C
      REAL*8 W8(1:8), N8(1:8), W4(1:4), N4(1:4)
      COMMON / GAUSSIAN_QUADRATURE / W8, N8, w4, n4
C
      COMPLEX*16 SURFINT(-49:50,-5:5,1:3,1:2), SINGINT(4,3)
      COMMON / SURFACE_INTEGRALS / SURFINT, SINGINT
C
C  DEFINE RADIUS OF CIRCLE ABOUT SINGULAR POINT.  DELX > DELY.
C
      R=MIN(DELY, 3./(DSQRT(ER)*KO))
      IF (R.GT.DELX) THEN
         WRITE (6,*) 'Error in CORNERS.  Slab must be re-partitioned'
         RETURN
      ENDIF
C
C  INITIALIZE OUTPUT TO ZERO
      DO 10 I=0,1
      DO 10 J=-1,0
      DO 10 M=1,3
      DO 10 K=1,2
   10     SURFINT(I,J,M,K)=(0.,0.)
C
C  OUTER GAUSSIAN QUADRATURE INTEGRATION LOOP
      DO 60 JJ=1,8
         V=(N8(JJ)+1)*DELY/2.DO
         X=0.DO
         IF (V.LT.R) X=DSQRT(R*R-V*V)
         SPAN=DELX-X
C
C  INITIALIZE INNER LOOP STORAGE MATRIX TO ZERO
C
         DO 20 I=0,1
         DO 20 J=-1,0
         DO 20 M=1,3
         DO 20 K=1,2
   20       TERM(I,J,M,K)=(0.,0.)
C
C  INNER GAUSSIAN QUADRATURE INTEGRATION LOOP
C
         DO 40 II=1,8
            U=N8(II)*SPAN/2. + (DELX+X)/2.
            HANK_TERM=HANKO(KO*DSQRT(U*U+V*V))
C
C  START (I,J,M,K) LOOP, PERFORMING FUNCTION EVALUATION FOR EACH CELL,
C  PHYSICAL BASIS FUNCTION, AND TRIG_TERM.
C
```

```
            DO 30 I=0,1
            DO 30 J=-1,0
            DO 30 M=1,3
            DO 30 K=1,2
C
C  LOGIC TO DEFINE TRIG_TERM AND EXP_TERM
C
               IF (K.EQ.2) THEN
                  TRIG_TERM=DCOS(F(M)*V)
               ELSEIF (J.EQ.-1) THEN
                  TRIG_TERM=DSIN(F(M)*(-V))
               ELSE
                  TRIG_TERM=DSIN(F(M)*V)
               ENDIF
C
               IF (I.EQ.0) THEN
                  EXP_TERM=EXP( CJ*G(M)*U)
               ELSE
                  EXP_TERM=EXP(-CJ*G(M)*U)
               ENDIF
C
C  ADD UP THE WEIGHTED FUNCTION EVALUATIONS
C
               TERM(I,J,M,K)=TERM(I,J,M,K)+W8(II)*
     &            TRIG_TERM*EXP_TERM*HANK_TERM
C
C  END (I,J,M,K) LOOP
C
   30        CONTINUE
C
C  END INNER GAUSSIAN QUADRATURE INTEGRATION LOOP
C
   40     CONTINUE
C
C  ADD THE WEIGHTED INNER INTEGRATION TO THE OUTER INTEGRATION MATRIX
C
         DO 50 I=0,1
         DO 50 J=-1,0
         DO 50 M=1,3
         DO 50 K=1,2
   50       SURFINT(I,J,M,K)=
     &         SURFINT(I,J,M,K)+TERM(I,J,M,K)*W8(JJ)*SPAN/2.
C
C  END OUTER INTEGRATION LOOP
C
   60 CONTINUE
C
C  PERFORM THE FINAL GAUSSIAN QUADRATURE MULTIPLICATION
C
      DO 70 I=0,1
      DO 70 J=-1,0
```

```
      DO 70 M=1,3
      DO 70 K=1,2
70       SURFINT(I,J,M,K)=SURFINT(I,J,M,K)*DELY/2.

      RETURN
      END
```

2.1.5 *Subroutine SINGULAR_INTEGRAL*

```
C***********************************************************************
C                                                                      **
C     >>>>  SUBROUTINE SINGULAR_INTEGRAL <<<<                          **
C                                                                      **
C     THIS SUBROUTINE CALCULATES THE VALUE OF THE SURFACE              **
C     AND LINE INTEGRALS IN THE NEIGHBORHOOD OF THE SINGULAR           **
C     MATCH POINT.  SINGINT IS ORGANIZED AS SINGINT(K,N) WHERE         **
C                                                                      **
C     K  =  1, SURFACE INTEGRAL OVER FULL CIRCLE                       **
C           2, SURFACE INTEGRAL OVER SEMICIRCLE, EXPONENTIAL           **
C                CONTAINS +F(N)                                        **
C           3, SURFACE INTEGRAL OVER SEMICIRCLE, EXPONENTIAL           **
C                CONTAINS -F(N)                                        **
C           4, LINE INTEGRAL ALONG CONDUCTOR                           **
C     N  =  PHYSICAL BASIS FUNCTION INDEX                              **
C                                                                      **
C     ALGORITHM REPLACES INTEGRANDS BY SMALL-ARGUMENT POLYNOMIAL       **
C     EXPANSIONS AND INTEGRATES TERM-BY-TERM.                          **
C                                                                      **
C***********************************************************************
C     CALLED BY:  DEFINE
C     SUBROUTINES CALLED:  NONE
C     FUNCTIONS CALLED:  INTEGRAL_COSINE, FACT
C     COMMON BLOCKS:  ALL, HANKEL, SURFACE_INTEGRALS
C
C     >>>>  INTERNAL VARIABLES  <<<<
C     I     =  INDEX
C     N     =  PHYSICAL BASIS FUNCTION INDEX
C     K, KK =  INDICES OF OUTER AND INNER DO-LOOPS
C     R     =  RADIUS OF INTEGRATION REGION
C     KOR   =  KO*R
C     KOR32 =  KO2 DIVIDED BY 3, QUANTITY SQUARED
C     LOGTERM= NATURAL LOGARITHM OF KOR
C     ANGLE =  VECTOR CONTAINING ANGLES FOR EACH PBF
C     GR2   =  VECTOR CONTAINING G(N)*R, QUANTITY SQUARED
C     RATIO =  CONVERGENCE TEST FOR CURRENT ITERATION
C     OLD_RATIO =  CONVERGENCE TEST FOR PREVIOU ITERATION
C     TOL   =  CONVERGENCE THRESHOLD
C     JERKOR = CJ*SQRT(ER)*KOR
C     INNER_SUM = RUNNING SUM FOR INNER DO-LOOP
C     TERM  =  RUNNING SUM OF OUTER LOOP
C
C     >>>>  DATA OUTPUT TO CALLING ROUTINE  <<<<
C     SINGINT =  VALUE OF SURFACE AND LINE INTEGRALS, PASSED VIA
C                COMMON BLOCK "SURFACE_INTEGRALS"
C
C***********************************************************************
      SUBROUTINE SINGULAR_INTEGRAL
C
```

```
      IMPLICIT NONE
      INTEGER I, N, K, KK
      REAL*8 R, KOR, KOR32, LOGTERM, ANGLE(3), GR2(3),
     &   RATIO, OLD_RATIO, TOL, INTEGRAL_COSINE, FACT
      COMPLEX*16 TERM, INNER_SUM, JERKOR
C
C --- C O M M O N   B L O C K S ----------------------------------
C
         COMPLEX*16 CJ
         REAL*8 PI, EO, MUO,ETA,KO,THETA,ER,H,W,F(3),G(3),DELX,DELY
         INTEGER XNODES, YNODES
      COMMON / ALL / PI, CJ, EO, MUO, ETA, KO, THETA, ER, H, W, F, G,
     &               XNODES, YNODES, DELX, DELY
C
         REAL*8 JO(0:6), YO(0:6), BETA(0:6), MAGO(0:6), PHASEO(0:6)
         COMPLEX*16 ALPHA(0:6)
         REAL*8 J1(0:6), Y1(0:6), MAG1(0:6), PHASE1(0:6)
      COMMON / HANKEL / JO, YO, ALPHA, BETA, MAGO, PHASEO,
     &                  J1, Y1, MAG1, PHASE1
C
         COMPLEX*16 SURFINT(-49:50,-5:5,1:3,1:2), SINGINT(4,3)
      COMMON / SURFACE_INTEGRALS / SURFINT, SINGINT
C
C
C
C  INITIALIZE SINGINT TO ZERO AND DEFINE SOME INTERNAL VARIABLES.
C
      TOL=1.0D-8
      R=MIN(DELY,3./(KO*DSQRT(ER)))
      DO 10 N=1,3
         ANGLE(N)=DATAN2(F(N),G(N))
         DO 10 K=1,4
   10       SINGINT(K,N)=(0.,0.)
      KOR=KO*R
      LOGTERM=DLOG(KOR)
      JERKOR=CJ*DSQRT(ER)*KOR
      KOR32=KOR*KOR/9.
C
C  CALCULATE SINGINT(1,*) AS THE DOUBLE SUMMATION GENERATED BY
C  INTEGRATING THE PRODUCT OF THE SMALL-ARGUMENT POLYNOMIAL
C  APPROXIMATIONS FOR BESSEL AND HANKEL FUNCTIONS OF ORDER ZERO.
C  LUCKILY, SINGINT(1,1)=SINGINT(1,2)=SINGINT(1,3).
C
      DO 30 K=0,6
         TERM=(0.,0.)
         DO 20 KK=0,6
            INNER_SUM=INNER_SUM+(KOR32**KK/(K+KK+1))*
     &         (ALPHA(KK)-CJ*BETA(KK)*(LOGTERM-1./(2*(K+KK+1))))
   20    CONTINUE
         SINGINT(1,1)=SINGINT(1,1) + INNER_SUM*JO(K)*(KOR32*ER)**K
   30 CONTINUE
```

```
      SINGINT(1,1)=SINGINT(1,1)*PI*R*R
      SINGINT(1,2)=SINGINT(1,1)
      SINGINT(1,3)=SINGINT(1,1)
C
C  SINGINT(2:3,*) ARE SURFACE INTEGRALS OVER SEMICIRCLES, AND ARE
C  CALCULATED AS DOUBLE SUMMATIONS.  WE HAVE TO TEST THE OUTER
C  SUMMATION FOR CONVERGENCE.  START THE OUTER LOOP.
C
      I=-1
   40 I=I+1

C  THE INNER LOOP DOESN'T DEPEND ON N, AND ITS RESULT APPLIES
C  TO BOTH SINGINT(2,*) AND SINGINT(3,*)
C
      INNER_SUM=(0.,0.)
      DO 50 K=0,6
         INNER_SUM=INNER_SUM+
     &      ( KOR32**K / (I+2*K+2) ) *
     &      ( ALPHA(K)-CJ*BETA(K)*(LOGTERM - 1./(I+2*K+2)) )
   50 CONTINUE
C
C  APPLY THE INNER LOOP RESULT TO SINGINT(2:3,*)
C
      DO 60 N=1,3
         TERM = INNER_SUM * JERKOR**I *
     &      INTEGRAL_COSINE(I,ANGLE(N)-PI,ANGLE(N))/FACT(I)
         SINGINT(2,N) = SINGINT(2,N) + TERM
         TERM = INNER_SUM * JERKOR**I *
     &      INTEGRAL_COSINE(I,-ANGLE(N)-PI,-ANGLE(N))/FACT(I)
         SINGINT(3,N) = SINGINT(3,N) + TERM
   60 CONTINUE
C
C  DUMP OLD RATIO INTO OLD_RATIO AND CALCULATE NEW RATIO; GET READY
C  FOR CONVERGANCE TEST.  ASSUME THESE TERMS CONVERGE AT THE SAME
C  RATE, WHICH IS DOMINATED BY THE FACTORIAL.
C
      OLD_RATIO=RATIO
      RATIO=ABS(TERM/SINGINT(2,1))
C
C  IF CONVERGENCE NOT REACHED IN OUTER SERIES, OR LESS THAN FIVE
C  TERMS TAKEN, GO BACK AND FIND ANOTHER TERM.
C
      IF ((I.LE.5).OR.(RATIO+OLD_RATIO.GT.TOL)) GOTO 40
C
C  DONE WITH OUTER SERIES; MULTIPLY BY R**2
C
      DO 70 N=1,3
         SINGINT(2,N) = SINGINT(2,N) * R * R
         SINGINT(3,N) = SINGINT(3,N) * R * R
   70 CONTINUE
C
```

```
C  DONE CALCULATING SINGINT(2:3,*).  ON TO SINGINT(4,*), THE LINE
C  INTEGRAL.  R CAN NOW BE AS LARGE AS 3/KO, INFLUENCING SOME OTHER
C  INTERNAL VARIABLES.
C
      R=MIN(3./KO,DELX)
      KOR=KO*R
      LOGTERM=DLOG(KOR)
      KOR32=KOR*KOR/9.
      DO 80 N=1,3
   80    GR2(N) = G(N) * G(N) * R * R
      IF (GR2(1).EQ.0) GR2(1)=1.D-10
C
C  START THE OUTER LOOP; AGAIN WE MUST TEST FOR CONVERGENCE.
C
      I=-1
   90 I=I+1
C
C  START THE INNER LOOP
C
      INNER_SUM=(0.,0.)
      DO 100 K=0,6
         INNER_SUM=INNER_SUM+
     &      ( KOR32**K / (2*I+2*K+1) ) *
     &      ( ALPHA(K)-CJ*BETA(K)*(LOGTERM - 1./(2*I+2*K+1)) )
  100 CONTINUE
C
C  APPLY THE INNER LOOP RESULT TO THE CURRENT OUTER LOOP TERM.
C
      DO 110 N=1,3
         TERM=INNER_SUM*(-GR2(N))**I/FACT(2*I)
         SINGINT(4,N)=SINGINT(4,N)+TERM
  110 CONTINUE
C
C  TEST FOR CONVERGENCE, ASSUMING ALL SINGINT(4,*) CONVERGE AT
C  THE SAME RATE.
C
      RATIO=ABS(TERM/SINGINT(4,1))
      IF ((I.LE.5).OR.(RATIO.GT.TOL)) GOTO 90
C
C  DONE WITH OUTER SERIES; MULTIPLY BY 2R
C
      DO 120 N=1,3
         SINGINT(4,N)=SINGINT(4,N)*2*R
  120 CONTINUE
C
      RETURN
      END
```

2.2 *Subroutine IMPEDANCE_MATRIX*

```
C*************************************************************************
C                                                                        **
C     >>>>  SUBROUTINE IMPEDANCE_MATRIX  <<<<                            **
C                                                                        **
C     THIS SUBROUTINE GENERATES THE IMPEDANCE AND VOLTAGE                **
C     MATRICES, BOTH UNNORMALIZED AND NORMALIZED.  UNNORMALIZED          **
C     MEANS NON-SQUARE, WHILE NORMALIZATION INVOLVES MATRIX              **
C     MULTIPLICATION BY THE CONJUGATE TRANSPOSE OF THE                   **
C     UNNORMALIZED IMPEDANCE MATRIX.                                     **
C                                                                        **
C*************************************************************************
C     CALLED BY:  MAIN
C     SUBROUTINES CALLED:  NORMALIZE, INVERT
C     FUNCTIONS CALLED:  ZMN, ARGD
C     COMMON BLOCKS:  ALL, IMPEDANCE, MATCH_POINT
C
C     >>>>  INTERNAL VARIABLES  <<<<
C     N  =  PHYSICAL BASIS FUNCTION INDEX
C     K  =  IMPLIED DO-LOOP INDEX
C     POINT =  VECTOR CONTAINING THE MATCH POINTS
C     SINE  =  SINE OF THETA, THE ANGLE OF INCIDENCE
C     COSINE=  COSINE OF THETA
C     EXP_TERM =  PHASE OF INCIDENT WAVE AT MATCH POINT
C     ZMNX  =  3X1 VECTOR CONTAINING UNNORMALIZED IMPEDANCE MATRIX
C              ELEMENTS FOR THE CURRENT MATCH POINT
C
C     >>>>  DATA OBTAINED FROM SUBROUTINE NORMALIZE  <<<<
C     ZN =  NORMALIZED, 3-by-3 IMPEDANCE MATRIX
C     VN =  NORMALIZED, 3-by-1 VOLTAGE MATRIX
C
C     >>>>  DATA OBTAINED FROM SUBROUTINE INVERT  <<<<
C     CUR=  3X1 CURRENT VECTOR (PHYSICAL BASIS FUNCTION
C                AMPLITUDES, THE FINAL FRUIT)
C
C*************************************************************************
      SUBROUTINE IMPEDANCE_MATRIX
C
      IMPLICIT NONE
      INTEGER N, K
      REAL*8 SINE, COSINE, POINT(40), ARGD
      COMPLEX*16 EXP_TERM, ZMNX(3)
C
C --- C O M M O N   B L O C K S ----------------------------------
C
         COMPLEX*16 CJ
         REAL*8 PI, EO, MUO,ETA,KO,THETA,ER,H,W,F(3),G(3),DELX,DELY
         INTEGER XNODES, YNODES
      COMMON / ALL / PI, CJ, EO, MUO, ETA, KO, THETA, ER, H, W, F, G,
```

```
     &               XNODES, YNODES, DELX, DELY
C
         INTEGER NUMMPS
         REAL*8 CN
         COMPLEX*16 Z(40,3), V(40), ZN(3,3), VN(3), CUR(3)
      COMMON / IMPEDANCE / Z, V , ZN, VN, CUR, CN, NUMMPS
C
         INTEGER I, J
         REAL*8 X, Y, HO2PY, HO2PY2, HO2MY, HO2MY2,
     &                WO2PX, WO2PX2, WO2MX, WO2MX2
      COMMON / MATCH_POINT / I, J, X, Y, HO2PY, HO2PY2, HO2MY, HO2MY2,
     &                       WO2PX, WO2PX2, WO2MX, WO2MX2
C
C  INITIALIZE EVERYTHING TO ZERO
C
      DATA (V(II),II=1,40) / 40*(0.,0.) /
      DATA ((Z(II,JJ),II=1,40),JJ=1,3) / 120*(0.,0.) /
C
C  DEFINE SINE AND COSINE OF THETA.
C
         SINE=DSIN(THETA)
         COSINE=DCOS(THETA)
C
C  START LOOP TO STEP THROUGH EACH MATCH POINT
C
         NUMMPS=0
   10    READ (19,*,END=30) POINT(NUMMPS+1)
C
C  CALCULATE MATCH POINT X- AND Y-COORDINATES AND INDICES.  FOR THIS
C  IMPLEMENTATION, J=0 BUT WE COULD MODIFY SO THAT MATCH POINTS WITHIN
C  THE DIELECTRIC COULD BE TAKEN.
C
         I=INT(POINT(NUMMPS+1)*(XNODES+1)+0.5)
         IF ((I.LT.1).OR.(I.GT.XNODES)) GOTO 10
         J=0
         NUMMPS=NUMMPS+1
         X= -W/2.D0 + I*DELX
         Y= -H/2.D0 + J*DELY
         WO2PX=W/2.+X
         WO2PX2=WO2PX*WO2PX
         WO2MX=W/2.-X
         WO2MX2=WO2MX*WO2MX
         HO2PY=(H/2.+Y)
         HO2PY2=HO2PY*HO2PY
         HO2MY=(H/2.-Y)
         HO2MY2=HO2MY*HO2MY
C
C  CALCULATE THE INCIDENT FIELD Ex AT THE CURRENT MATCH POINT.
C
         EXP_TERM=EXP(-CJ*K0*(X*SINE+HO2MY*COSINE))
         V(NUMMPS)=  COSINE*EXP_TERM
```

```
C
C  CALL ZMN TO GET IMPEDANCE MATRIX ELEMENTS FOR CURRENT MATCH POINT.
C  DUMP RESULT INTO THE UNNORMALIZED IMPEDANCE MATRIX.
C
         CALL ZMN (ZMNX)
         DO 20 N=1,3
            Z(NUMMPS,N)=ZMNX(N)
   20    CONTINUE
C
      GOTO 10
C
   30 WRITE (20,40) NUMMPS
   40 FORMAT (1X, I2, ' Match Points:')
      WRITE (20,'(15(F4.2,1X))') (POINT(K), K=1,NUMMPS)
C
C  NORMALIZE IMPEDANCE AND VOLTAGE MATRICES, AND SOLVE THE RESULTANT
C  3X3 SYSTEM OF EQUATIONS.
C
      CALL NORMALIZE
      CALL INVERT (3)
C
      WRITE (20,50) CN, (ABS(CUR(N)), ARGD(CUR(N)), N=1,3)
   50 FORMAT (1X, ' CN=', F7.1, 2x, 3('(', F8.6, ',', F6.1, ')'))
C
C  DONE, RETURN TO CALLING ROUTINE
C
      RETURN
      END
```

2.2.1 *Subroutine ZMN*

```
C*************************************************************************
C                                                                        **
C     >>>>  SUBROUTINE ZMN(ZMNX)  <<<<                                   **
C                                                                        **
C     THIS FUNCTION RETURNS THE IMPEDANCE MATRIX ELEMENTS                **
C     CORRESPONDING TO THE CURRENT MATCH POINT.  ZMNX CORRESPONDS        **
C     TO THE Ex INTEGRAL EQUATION                                        **
C                                                                        **
C*************************************************************************
C     CALLED BY:  IMPEDANCE_MATRIX
C     SUBROUTINES CALLED:  SURF_INTEGRALS, LINE_INTEGRALS
C     FUNCTIONS CALLED:  CONJUGATE, HANKO, HANK1
C     COMMON BLOCKS:  ALL, FIELD_AMPLITUDES, MATCH_POINT,
C                     SURFACE_INTEGRALS
C
C     >>>>  INTERNAL VARIABLES  <<<<
C     M  =  PHYSICAL BASIS FUNCTION INDEX
C     FOURJ =  RECIPROCAL OF 4*CJ
C     EXP_TERM =  COMPLEX EXPONENTIAL INVOLVING X
C     SINE     =  SINE INVOLVING Y
C     COSINE   =  COSINE INVOLVING Y
C     SINE_H   =  SINE INVOLVING H
C     COSINE_H =  COSINE INVOLVING H
C     EXP1     =  COMPLEX EXPONENTIAL INVOLVING W
C     EXP2     =  COMPLEX CONJUGATE OF EXP1
C     PP       =  SQUARE ROOT OF WO2PX2 PLUS HO2PY2
C     PM       =  SQUARE ROOT OF WO2PX2 PLUS HO2MY2
C     MP       =  SQUARE ROOT OF WO2MX2 PLUS HO2PY2
C     MM       =  SQUARE ROOT OF WO2MX2 PLUS HO2MY2
C     HANKO_PP =  HANKEL FUNCTION (ORDER 0) INVOLVING PP
C     HANKO_PM =  HANKEL FUNCTION (ORDER 0) INVOLVING PM
C     HANKO_MP =  HANKEL FUNCTION (ORDER 0) INVOLVING MP
C     HANKO_MM =  HANKEL FUNCTION (ORDER 0) INVOLVING MM
C     HANKO_PP =  HANKEL FUNCTION (ORDER 1) INVOLVING PP
C     HANKO_MP =  HANKEL FUNCTION (ORDER 1) INVOLVING MP
C
C     >>>>  DATA OBTAINED FROM SUBROUTINE SURF_INTEGRALS  <<<<
C     SURFACE INTEGRALS IMNA AND IMNB
C
C     >>>>  DATA OBTAINED FROM SUBROUTINE LINE_INTEGRALS  <<<<
C     LINE INTEGRALS IMNG THROUGH IMNL
C
C*************************************************************************
      SUBROUTINE ZMN (ZMNX)
C
      IMPLICIT NONE
      INTEGER M
      REAL*8 SINE, COSINE, SINE_H, COSINE_H, PP, MP, PM, MM, ER1
      COMPLEX*16 FOURJ, EXP_TERM, EXP1, EXP2, HANKO_PP, HANKO_PM,
```

```
     &  HANKO_MP, HANKO_MM, HANK1_PP, HANK1_MP, HANKO, HANK1,
     &  CONJUGATE, SING_TERM
      COMPLEX*16 ZMNX(3), ZMNY(3), ZMNZ(3), IMNA(3), IMNB(3),
     &     IMNG(3), IMNH(3), IMNI(3), IMNJ(3), IMNK(3), IMNL(3),
C
C --- C O M M O N   B L O C K S ---------------------------------------
C
         COMPLEX*16 CJ
         REAL*8 PI, EO, MUO,ETA,KO,THETA,ER,H,W,F(3),G(3),DELX,DELY
         INTEGER XNODES, YNODES
      COMMON / ALL / PI, CJ, EO, MUO, ETA, KO, THETA, ER, H, W, F, G,
     &               XNODES, YNODES, DELX, DELY
C
         COMPLEX*16 CX(3), CY(3), CZ(3)
      COMMON / FIELD_AMPLITUDES / CX, CY, CZ
C
         INTEGER I, J
         REAL*8 X, Y, HO2PY, HO2PY2, HO2MY, HO2MY2,
     &               WO2PX, WO2PX2, WO2MX, WO2MX2
      COMMON / MATCH_POINT / I, J, X, Y, HO2PY, HO2PY2, HO2MY, HO2MY2,
     &                       WO2PX, WO2PX2, WO2MX, WO2MX2
C
         COMPLEX*16 SURFINT(-49:50,-5:5,1:3,1:2), SINGINT(4,3)
      COMMON / SURFACE_INTEGRALS / SURFINT, SINGINT
C
C
C  DEFINE INTERNAL VARIABLES THAT DO NOT DEPEND ON N
C
      FOURJ=1./(4.*CJ)
      ER1=ER-1.
      PP=DSQRT(WO2PX2+HO2PY2)
      PM=DSQRT(WO2PX2+HO2MY2)
      MP=DSQRT(WO2MX2+HO2PY2)
      MM=DSQRT(WO2MX2+HO2MY2)
      HANKO_PP=HANKO(KO*PP)
      HANKO_PM=HANKO(KO*PM)
      HANKO_MP=HANKO(KO*MP)
      HANKO_MM=HANKO(KO*MM)
      HANK1_PP=HANK1(KO*PP)
      HANK1_MP=HANK1(KO*MP)
C
C  CALL SURF_INTEGRALS AND LINE_INTEGRALS TO GET TERMS IMNA THROUGH
C  IMNN.  SURF_INTEGRALS GIVES US THE SURFACE INTEGRALS OVER THE CROSS-
C  SECTIONAL SURFACE OF THE SLAB.  LINE_INTEGRALS GIVES US LINE
C  INTEGRALS ACROSS THE FOUR EDGES OF THE SLAB.
C
      CALL SURF_INTEGRALS (IMNA, IMNB)
      CALL LINE_INTEGRALS (IMNG, IMNH, IMNI, IMNJ, IMNK, IMNL)
C
C  START LOOP IN N AND DEFINE SOME INTERNAL VARIABLES THAT DEPEND ON N
C
```

```
      DO 10 M=1,3
         EXP_TERM=EXP(-CJ*G(M)*X)
           SINE=DSIN(F(M)*(Y+H/2.))
         COSINE=DCOS(F(M)*(Y+H/2.))
           SINE_H=DSIN(F(M)*H)
         COSINE_H=DCOS(F(M)*H)
         EXP1=EXP( CJ*G(M)*W/2.)
         EXP2=CONJUGATE(EXP1)
C
C  CALCULATE ZMNX(M), USING THE CORRECT SINGULAR TERM
C
         SING_TERM=SINE*SINGINT(1,M)
         IF (J.EQ.0) SING_TERM=(SINGINT(2,M)-SINGINT(3,M))/(2*CJ)
         ZMNX(M)=CX(M)*SINE*EXP_TERM
         ZMNX(M)=ZMNX(M)-FOURJ*ER1*EXP_TERM*CX(M)*KO*KO*
     &         (COSINE*IMNA(M)+SINE*IMNB(M)+SING_TERM)
         ZMNX(M)=ZMNX(M)+0.25*ER1*EXP_TERM*CY(M)*G(M)*
     &         (IMNK(M)-COSINE_H*IMNL(M))
         ZMNX(M)=ZMNX(M)-FOURJ*ER1*CY(M)*((EXP1*HANKO_PP
     &         -EXP2*HANKO_MP)-COSINE_H*(EXP1*HANKO_PM-EXP2*HANKO_MM))
         ZMNX(M)=ZMNX(M)+FOURJ*ER1*CX(M)*KO*
     &         ( SINE*(EXP1*WO2PX*IMNG(M)+EXP2*WO2MX*IMNH(M))
     &        +COSINE*(EXP1*WO2PX*IMNI(M)+EXP2*WO2MX*IMNJ(M)) )
         ZMNX(M)=ZMNX(M)+0.25*CZ(M)*(ETA/KO)*EXP_TERM*
     &         (KO*KO-G(M)*G(M))*IMNK(M)
         ZMNX(M)=ZMNX(M)+FOURJ*CZ(M)*G(M)*(ETA/KO)*
     &         (EXP1*HANKO_PP-EXP2*HANKO_MP)
         ZMNX(M)=ZMNX(M)-0.25*CZ(M)*ETA*( EXP1*WO2PX*HANK1_PP/PP
     &                                  + EXP2*WO2MX*HANK1_MP/MP )
   10 CONTINUE
C
      RETURN
      END
```

2.2.1.1 *Subroutine SURF_INTEGRALS*

```
C*************************************************************************
C                                                                        **
C     >>>>  SUBROUTINE SURF_INTEGRALS (IMNA, IMNB)  <<<<                 **
C                                                                        **
C     THIS SUBROUTINE RETURNS THE THE SURFACE INTEGRALS OVER THE         **
C     DIELECTRIC CROSS-SECTIONAL SURFACE LESS THE CIRCULAR               **
C     REGION ABOUT THE SINGULAR MATCH POINT.  IMNA AND IMNB              **
C     ARE SIMPLY THE SUMS OF ELEMENTS OF MATRIX                          **
C     SURFINT, WHICH HAS BEEN PREVIOUSLY CALCULATED.                     **
C                                                                        **
C*************************************************************************
C     CALLED BY:  ZMN
C     SUBROUTINES CALLED:  NONE
C     FUNCTIONS CALLED:  NONE
C     COMMON BLOCKS:  ALL, SURFACE_INTEGRALS, MATCH_POINT
C
C     >>>>  INTERNAL VARIABLES <<<<
C     MM =  OUTER DO-LOOP INDEX
C     NN =  INNER DO-LOOP INDEX
C     M  =  PHYSICAL BASIS FUNCTION INDEX
C
C     >>>>  DATA OUTPUT TO CALLING ROUTINE  <<<<
C     IMNA, IMNB  =  3X1 VECTORS CONTAINING VALUES OF SURFACE INTEGRALS
C
C*************************************************************************
      SUBROUTINE SURF_INTEGRALS (IMNA, IMNB)
C
      IMPLICIT NONE
      INTEGER MM, NN, M
      COMPLEX*16 IMNA(3), IMNB(3)
C
C --- C O M M O N    B L O C K S ---------------------------------
C
         COMPLEX*16 CJ
         REAL*8 PI, EO, MUO,ETA,KO,THETA,ER,H,W,F(3),G(3),DELX,DELY
         INTEGER XNODES, YNODES
      COMMON / ALL / PI, CJ, EO, MUO, ETA, KO, THETA, ER, H, W, F, G,
     &               XNODES, YNODES, DELX, DELY
C
         COMPLEX*16 SURFINT(-49:50,-5:5,1:3,1:2), SINGINT(4,3)
      COMMON / SURFACE_INTEGRALS / SURFINT, SINGINT
C
         INTEGER I, J
         REAL*8 X, Y, HO2PY, HO2PY2, HO2MY, HO2MY2,
     &               WO2PX, WO2PX2, WO2MX, WO2MX2
      COMMON / MATCH_POINT / I, J, X, Y, HO2PY, HO2PY2, HO2MY, HO2MY2,
     &                       WO2PX, WO2PX2, WO2MX, WO2MX2
C
C
```

```
C  INITIALIZE OUTPUT AND RUNNING SUMS TO ZERO
C
      DO 10 N=1,3
         IMNA(N)=(0.,0.)
   10    IMNB(N)=(0.,0.)
C
C  ADD UP SURFINT ELEMENTS TO GENERATE IMNA AND IMNB.
C  MIDDLE LOOP STEPS ACROSS SLAB HORIZONTALLY, WHILE INNER LOOP STEPS
C  UP SLAB VERTICALLY.
C
      DO 20 N=1,3
         DO 20 MM=1-I,XNODES+1-I
            DO 20 NN=-J,YNODES-J-1
               IMNA(N)=IMNA(N)+SURFINT(MM,NN,N,1)
               IMNB(N)=IMNB(N)+SURFINT(MM,NN,N,2)
   20 CONTINUE
C
      RETURN
      END
```

2.2.1.2 *Subroutine LINE_INTEGRALS*

```
C*********************************************************************
C                                                                    **
C     >>>> SUBROUTINE LINE_INTEGRALS (IMNG, IMNH, IMNI,      <<<<    **
C     >>>>                            IMNJ, IMNK, IMNL)      <<<<    **
C                                                                    **
C     THIS SUBROUTINE RETURNS LINE INTEGRALS ALONG THE EDGES         **
C     OF THE SLAB.  THE LINE INTEGRALS ARE FUNCTIONS OF THE          **
C     MATCH POINT AND THE PHYSICAL BASIS FUNCTION.  THE              **
C     SUBROUTINE USES 8-POINT GAUSSIAN QUADRATURE TO NUMERICALLY     **
C     INTEGRATE THE INTEGRAND.                                       **
C                                                                    **
C*********************************************************************
C     CALLED BY:  ZMN
C     SUBROUTINES CALLED:  NONE
C     FUNCTIONS CALLED:  HANKO
C     COMMON BLOCKS:  ALL, GAUSSIAN_QUADRATURE, MATCH_POINT
C
C     >>>>  INTERNAL VARIABLES  <<<<
C     K  =  OUTER DO LOOP INDEX
C     II =  INNER DO LOOP INDEX
C     M  =  PHYSICAL BASIS FUNCTION INDEX
C     U, V  =  GAUSSIAN QUADRATURE NODE
C     SINE     =  SINE TERM IN INTEGRANDS
C     COSINE   =  COSINE TERM IN INTEGRANDS
C     POS_SQRT =  SQUARE ROOT TERM CONTAINING WO2PX2 OR HO2PY2
C     NEG_SQRT =  SQUARE ROOT TERM CONTAINING WO2MX2 OR HO2MY2
C     POS_HANKO=  HANKEL FUNCTION (ORDER 0) CONTAINING POS_SQRT
C     NEG_HANKO=  HANKEL FUNCTION (ORDER 0) CONTAINING NEG_SQRT
C     POS_HANK1=  HANKEL FUNCTION (ORDER 1) CONTAINING POS_SQRT
C     NEG_HANK1=  HANKEL FUNCTION (ORDER 1) CONTAINING NEG_SQRT
C
C     >>>>  DATA FROM CALLING ROUTINE  <<<<
C     M  =  PHYSICAL BASIS FUNCTION INDEX
C
C     >>>>  DATA OUTPUT TO CALLING ROUTINE  <<<<
C     IMNG THRU IMNL = 3X1 VECTORS CONTAINING VALUES OF LINE INTEGRALS
C
C*********************************************************************
      SUBROUTINE LINE_INTEGRALS (IMNG, IMNH, IMNI, IMNJ, IMNK, IMNL)
C
      IMPLICIT NONE
      INTEGER M, II, K
      REAL*8 U, V, SINE, COSINE, POS_SQRT, NEG_SQRT
      COMPLEX*16 POS_HANKO, NEG_HANKO, POS_HANK1, NEG_HANK1, HANKO,
     &           HANK1, EXP_TERM
      COMPLEX*16 IMNG(3), IMNH(3), IMNI(3), IMNJ(3), IMNK(3), IMNL(3)
C
C --- C O M M O N   B L O C K S -----------------------------------
C
```

```
      COMPLEX*16 CJ
      REAL*8 PI, EO, MUO,ETA,KO,THETA,ER,H,W,F(3),G(3),DELX,DELY
      INTEGER XNODES, YNODES
     COMMON / ALL / PI, CJ, EO, MUO, ETA, KO, THETA, ER, H, W, F, G,
     &               XNODES, YNODES. DELX, DELY
C
      REAL*8 W8(1:8), N8(1:8), W4(1:4), N4(1:4)
     COMMON / GAUSSIAN_QUADRATURE / W8, N8, w4, n4
C
      INTEGER I, J
      REAL*8 X, Y, HO2PY, HO2PY2, HO2MY, HO2MY2,
     &               WO2PX, WO2PX2, WO2MX, WO2MX2
     COMMON / MATCH_POINT / I, J, X, Y, HO2PY, HO2PY2, HO2MY, HO2MY2,
     &                      WO2PX, WO2PX2, WO2MX, WO2MX2
C
      COMPLEX*16 SURFINT(-49:50,-5:5,1:3,1:2), SINGINT(4,3)
     COMMON / SURFACE_INTEGRALS / SURFINT, SINGINT
C
C
C  INITIALIZE OUTPUT TO ZERO AND DEFINE SOME INTERNAL VARIABLES
C
     DO 5 N=1,3
        IMNG(N)=(0., 0.)
        IMNH(N)=(0., 0.)
        IMNI(N)=(0., 0.)
        IMNJ(N)=(0., 0.)
        IMNK(N)=(0., 0.)
        IMNL(N)=(0., 0.)
   5 CONTINUE
C
C  INTEGRATE ALONG VERTICAL EDGES.  THE OUTER LOOP STEPS UP THE EDGES
C  OF THE SLAB.
C
     DO 10 K=-J,YNODES-J-1
     DO 10 II=1,8
C
C  DEFINE GAUSSIAN QUADRATURE NODE, AND SOME INTERNAL VARIABLES.
C
      V=(N8(II)+2.*K+1)*DELY/2.DO
      POS_SQRT=DSQRT(WO2PX2+V*V)
      NEG_SQRT=DSQRT(WO2MX2+V*V)
      POS_HANKO=HANKO(KO*POS_SQRT)
      NEG_HANKO=HANKO(KO*NEG_SQRT)
      POS_HANK1=HANK1(KO*POS_SQRT)
      NEG_HANK1=HANK1(KO*NEG_SQRT)
C
C  ADD UP THE WEIGHTED INTEGRAND EVALUATED AT THE CURRENT NODE
C
      DO 10 N=1,3
         SINE=DSIN(F(N)*V)
         COSINE=DCOS(F(N)*V)
```

```
            IMMG(M)=IMMG(M)+W8(II)*COSINE*POS_HANK1 / POS_SQRT
            IMMH(M)=IMMH(M)+W8(II)*COSINE*NEG_HANK1 / NEG_SQRT
            IMMI(M)=IMMI(M)+W8(II)*  SINE*POS_HANK1 / POS_SQRT
            IMMJ(M)=IMMJ(M)+W8(II)*  SINE*NEG_HANK1 / NEG_SQRT
   10 CONTINUE
C
C  MULTIPLY BY DELY/2
C
      DO 20 M=1,3
         IMMG(M)=IMMG(M)*DELY/2.D0
         IMMH(M)=IMMH(M)*DELY/2.D0
         IMMI(M)=IMMI(M)*DELY/2.D0
         IMMJ(M)=IMMJ(M)*DELY/2.D0
   20 CONTINUE
C
C  INTEGRATE ALONG THE HORIZONTAL EDGES.  THE OUTER LOOP STEPS FROM
C  LEFT TO RIGHT ONE CELL AT A TIME.
C
      DO 40 K=-I, XNODES-I
      DO 40 II=1,8
C
C  DEFINE GAUSSIAN QUADRATURE NODE, AND SOME INTERNAL VARIABLES
C
         U=(N8(II)+2.*K+1)*DELX/2.D0
         NEG_HANK0=HANK0(K0*DSQRT(U*U+H02MY2))
         POS_HANK0=HANK0(K0*DSQRT(U*U+H02PY2))

C  ADD UP THE WEIGHTED FUNCTION EVALUATIONS FOR EACH M.  CHECK TO SEE
C  IF IMMK IS NEAR THE SINGULAR POINT.
C
         DO 40 M=1,3
            EXP_TERM=EXP(-CJ*G(M)*U)
            IMML(M)=IMML(M)+W8(II)*EXP_TERM*NEG_HANK0
            IF ((J.EQ.0).AND.((K.EQ.-1).OR.(K.EQ.0))) GOTO 40
            IMMK(M)=IMMK(M)+W8(II)*EXP_TERM*POS_HANK0
   40    CONTINUE
C
C  MULTIPLY BY DELX/2, AND RETURN IF SINGULARITY IS NOT ON THE
C  CONDUCTOR
C
      DO 50 M=1,3
         IMMK(M)=IMMK(M)*DELX/2.
         IMML(M)=IMML(M)*DELX/2.
   50 CONTINUE
      IF (J.NE.0) RETURN
C
C  HANDLE THE INTEGRATIONS NEAR THE SINGULARITY, IF ON THE CONDUCTOR.
C  DO AS MUCH NUMERICAL INTEGRATION OF IMMK AS NECESSARY.
C  IF DELX IS SMALL ENOUGH (DELX <= 3/K0) THEN NONE NEEDED.
C  EXPLOIT THE SYMMETRY OF THE INTEGRAND.
C
```

```
      IF (KO*DELX.GT.3) THEN
         DO 60 II=1,8
            U=0.5*((DELX-3./KO)*W8(II)+DELX+3./KO)
            DO 60 N=1,3
               IMNK(N)=IMNK(N)+(DELX-3./KO)*W8(II)*
     &                 DCOS(G(N)*U)*HANKO(KO*U)
   60    CONTINUE
      ENDIF
C
C  ADD IN THE ANALYTIC INTEGRATION AROUND THE SINGULARITY.
C
      DO 70 N=1,3
         IMNK(N)=IMNK(N)+SINGINT(4,N)
   70 CONTINUE
C
      RETURN
      END
```

2.2.2 *Subroutine NORMALIZE*

```
C*********************************************************************
C                                                                    **
C     >>>>  SUBROUTINE NORMALIZE  <<<<                               **
C                                                                    **
C     THIS SUBROUTINE NORMALIZES THE UNNORMALIZED IMPEDANCE AND      **
C     VOLTAGE MATRICES BY MULTIPLYING BOTH BY THE CONJUGATE          **
C     TRANSPOSE OF THE IMPEDANCE MATRIX.  WHILE THE UNNORMALIZED     **
C     IMPEDANCE AND VOLTAGE MATRICES ARE M-by-3 AND M-by-1,          **
C     RESPECTIVELY, THE NORMALIZED IMPEDANCE AND VOLTAGE MATRICES    **
C     ARE 3-by-3 AND 3-by-1, RESPECTIVELY.                           **
C                                                                    **
C*********************************************************************
C     CALLED BY:  IMPEDANCE_MATRIX
C     SUBROUTINES CALLED: NONE
C     FUNCTIONS CALLED:  CONJUGATE
C     COMMON BLOCKS:  ALL, IMPEDANCE, MATCH_POINT
C
C     >>>>  INTERNAL VARIABLES  <<<<
C     II, JJ, M  =  DO-LOOP INDICES
C
C*********************************************************************
      SUBROUTINE NORMALIZE
C
      IMPLICIT NONE
      INTEGER II, JJ, M
      COMPLEX*16 CONJUGATE
C
C --- C O M M O N   B L O C K S ----------------------------------
C
         COMPLEX*16 CJ
         REAL*8 PI, EO, MUO,ETA,KO,THETA,ER,H,W,F(3),G(3),DELX,DELY
         INTEGER XNODES, YNODES
      COMMON / ALL / PI, CJ, EO, MUO, ETA, KO, THETA, ER, H, W, F, G,
     &               XNODES, YNODES, DELX, DELY
C
         INTEGER NUMMPS
         REAL*8 CN
         COMPLEX*16 Z(40,3), V(40), ZN(3,3), VN(3), CUR(3)
      COMMON / IMPEDANCE / Z, V , ZN, VN, CUR, CN, NUMMPS
C
         INTEGER I, J
         REAL*8 X, Y, HO2PY, HO2PY2, HO2MY, HO2MY2,
     &               WO2PX, WO2PX2, WO2MX, WO2MX2
      COMMON / MATCH_POINT / I, J, X, Y, HO2PY, HO2PY2, HO2MY, HO2MY2,
     &                       WO2PX, WO2PX2, WO2MX, WO2MX2
C
C  GENERATE THE NORMALIZED VOLTAGE MATRIX BY MULTIPLYING THE
C  UNNORMALIZED VOLTAGE MATRIX BY THE CONJUGATE TRANSPOSE OF THE
C  UNNORMALIZED IMPEDANCE MATRIX
```

```
C
      DO 10 II=1,3
         VN(II)=(0., 0.)
         DO 10 M=1,NUMMPS
            VN(II)=VN(II)+CONJUGATE(Z(M,II))*V(M)
   10 CONTINUE
C
C  GENERATE THE NORMALIZED IMPEDANCE MATRIX BY MULTIPLYING THE
C  UNNORMALIZED IMPEDANCE MATRIX BY ITS CONJUGATE TRANSPOSE
C
      DO 20 II=1,3
         DO 20 JJ=1,3
            ZN(II,JJ)=(0., 0.)
            DO 20 M=1,NUMMPS
               ZN(II,JJ)=ZN(II,JJ)+CONJUGATE(Z(M,II))*Z(M,JJ)
   20 CONTINUE
C
      RETURN
      END
```

2.2.3 *Subroutine INVERT*

```
C*************************************************************
C              >>>> SUBROUTINE INVERT <<<<                   **
C                                                            **
C  THIS SUBROUTINE USES GAUSSIAN ELIMINATION TO CALCULATE    **
C  THE INVERSE OF THE SQUARE MATRIX 'BINPUT'.  THE SUB-      **
C  ROUTINE FIRST PUTS THE INPUT VECTOR INTO A SCRATCH VEC-   **
C  TOR CALLED INPUT, THEN USES ROW OPERATIONS TO CHANGE      **
C  THE MATRIX TO UPPER TRIANGULAR FORM AND STORES THE PER-   **
C  MUTATION FACTORS IN THE ZERO ELEMENTS OF THE UPPER TRI-   **
C  ANGULAR MATRIX.  NEXT THE INVERSE IS CALCULATED BY        **
C  SOLVING THE TRIANGULARIZED MATRIX AGAINST EACH PERMUTA-   **
C  TED COLUMN OF THE IDENTITY MATRIX WITH DIMENSION          **
C  'IDIM'.  THE INVERSE OF THE BINPUT MATRIX IS PLACED IN    **
C  THE MATRIX INVERSE.                                       **
C                                                            **
C*************************************************************
C  >>  CALLED BY: SUBROUTINE SOLVNVRS OR SOLVITER
C
C  >>  SUBROUTINES CALLED: ** NONE **
C
C  >>>> INPUT VARIABLES <<<<
C  BINPUT = SQUARE MATRIX TO BE INVERTED
C  IDIM = DIMENSION OF THE INPUT MATRIX
C
C  >>>> INTERNAL CALCULATION VARIABLES <<<<
C  CZERO   = COMPLEX VALUE, 0.0 + j0.0
C  DEL     = COLUMN MATRIX OF CHANGE IN IDENTITY MATRIX COLUMN
C            ELEMENT VALUES RESULTING FROM TRIANGULARIZATION
C            OF THE INPUT MATRIX
C  INPUT   = MATRIX USED AS SCRATCH FOR INVERSION
C  IPIVOT  = COLUMN VECTOR OF ROW PIVOT INFORMATION.  THE
C            VALUE OF EACH ELEMENT REPRESENTS THE ROW NUMBER
C            THAT WAS PIVOTED, AND THE ELEMENT NUMBER REPRE-
C            SENTS THE LOCATION FOR THE PIVOTED VALUES
C  COLMAX  = ABSOLUTE VALUE OF THE MAXIMUM VALUE IN A COLUMN
C            OF THE INPUT VECTOR
C  Q       = VARIABLE USED DURING BACKSOLVING OPERATION, HAS
C            THE VALUE OF SUM OF PRODUCTS OF ALL KNOWN X VALUES
C            TIMES THEIR CORRESPONDING RHS ELEMENT, SPECIFICALLY
C            X(J) = (RHS(J) - Q)/INPUT(I,J)
C  RHS     = COLUMN MATRIX USED TO STORE THE ELEMENTS OF KNOWN
C            VALUES USED TO SOLVE FOR THE IDENTITY MATRIX COLUMN
C  TEMP    = TEMPORARY STORAGE VARIABLE FOR MATRIX ROW PIVOTING
C            OPERATIONS AND TRIANGULARIZATION OPERATIONS
C
C  >>>> OUTPUT VARIABLES <<<<
C  INVERSE = INVERSE OF INPUT MATRIX
C
C*************************************************************
```

```
C
      SUBROUTINE INVERT (IDIM)
C
      IMPLICIT NONE
      INTEGER I, J, K, L, IDIM, IPIVOT(40)
      COMPLEX*16 RS(40), DEL(40), Q, CZERO, TEMP,
     +           ZINV(40,40), RHS, ALPHAS, INPUT(40,40)
      REAL*8 COLMAX, SUM_ZN, SUM_ZINV, NORM_ZN, NORM_ZINV
C
C --- C O M M O N   B L O C K S ----------------------------------
C
         COMPLEX*16 CJ
         REAL*8 PI, EO, MUO,ETA,KO,THETA,ER,H,W,F(3),G(3),DELX,DELY
         INTEGER XNODES, YNODES
      COMMON / ALL / PI, CJ, EO, MUO, ETA, KO, THETA, ER, H, W, F, G,
     &               XNODES, YNODES, DELX, DELY
C
         INTEGER NUMMPS
         REAL*8 CN
         COMPLEX*16 Z(40,3), V(40), ZN(3,3), VN(3), CUR(3)
      COMMON / IMPEDANCE / Z, V , ZN, VN, CUR, CN, NUMMPS
C
C
      CZERO = DCMPLX(0.,0.)
C
C  TRANSFER THE ORIGINAL MATRIX INTO THE SCRATCH MATRIX.
C
      DO 90 I = 1,IDIM
         DO 90 J = 1,IDIM
   90       INPUT(I,J) = ZN(I,J)
C
C  OUTERMOST DO LOOP REPEATS FOR EACH COLUMN OF THE MATRIX
C  AS THE MATRIX IS BEING REDUCED TO LOWER TRIANGULAR FORM.
C
      DO 100, J = 1,(IDIM-1)
         COLMAX = ABS(INPUT(J,J))
         IPIVOT(J) = J
C
C  SEARCH FOR PIVOT ROW.
C
         DO 110, K = J+1, IDIM
            IF(ABS(INPUT(K,J)).GT.COLMAX)THEN
               COLMAX = ABS(INPUT(K,J))
               IPIVOT(J) = K
            END IF
  110    CONTINUE
C
C  IF NEEDED, PIVOT ROW J WITH ROW IPIVOT(J).
C
         IF(IPIVOT(J).NE.J)THEN
            DO 120, K = J,IDIM
```

```
              TEMP = INPUT(J,K)
              INPUT(J,K) = INPUT(IPIVOT(J),K)
              INPUT(IPIVOT(J),K) = TEMP
  120       CONTINUE
         END IF
C
C  WITH THE MAX ABS VALUE OF THE COLUMN ON TOP, REPLACE
C  THE REMAINING ELEMENTS OF THE COLUMN WITH THE PERMUTATION
C  FACTOR OF THE ROW.
C
         DO 130, K = (J+1),IDIM
            TEMP = DCMPLX(-1.,0.) * (INPUT(K,J)/INPUT(J,J))
            INPUT(K,J) = TEMP
            DO 130, L = (J+1),IDIM
  130          INPUT(K,L) = (TEMP * INPUT(J,L)) + INPUT(K,L)
C
C  END DO LOOP THAT PUTS INPUT MATRIX IN UPPER TRIANGULAR FORM.
C
  100 CONTINUE
C
C  NEXT CALCULATE INVERSE MATRIX BY SOLVING THE EQUATION
C  Ax = b FOR b = EACH COLUMN OF THE IDENTITY MATRIX, AND
C  x = EACH COLUMN OF THE INVERSE MATRIX.
C
      DO 150 J = 1,IDIM
C
C  FORM THE J-th COLUMN OF THE IDENTITY MATRIX.
C
         DO 155 K=1,IDIM
  155       RS(K) = CZERO
         RS(J) = DCMPLX(1.,0.)
C
C  PIVOT ELEMENTS OF IDENTITY MATRIX COLUMN IN SAME
C  ORDER AS THE IMPEDANCE MATRIX ROWS WERE PIVOTED
C  DURING REDUCTION TO TRIANGULAR FORM.
C
         DO 156, K=1,(IDIM-1)
            IF(IPIVOT(K).NE.K)THEN
              TEMP = RS(K)
              RS(K) = RS(IPIVOT(K))
              RS(IPIVOT(K)) = TEMP
            ENDIF
  156    CONTINUE
C
C  PERMUTATE THE RHS MATRIX ACCORDING TO THE MULTIPLICATION
C  FACTORS STORED IN THE ELEMENTS OF THE INPUT MATRIX.
C
         DO 160, K = 2, IDIM
            DEL(K) = CZERO
            DO 165, L = 1, (K-1)
               DEL(K) = DEL(K) + (RS(L) * INPUT(K,L))
```

```
  165       CONTINUE
            RS(K) = DEL(K) + RS(K)
  160    CONTINUE
C
C  NOW BACKSOLVE TO FIND THE ELEMENTS OF THE INVERSE MATRIX
C  J-th COLUMN.  ALGORITHM ADAPTED FROM FIGURE 2.1, ON PAGE
C  29 OF THE TEXT "NUMERICAL ANALYSIS" BY JOHNSON AND REISS.
C
         ZINV(IDIM,J) = RS(IDIM)/INPUT(IDIM,IDIM)
         DO 170, K = 1,(IDIM-1)
            Q = CZERO
            DO 175, L = 1,K
               Q = Q + INPUT((IDIM-K),(IDIM-(L-1))) *
     +                 ZINV((IDIM-(L-1)),J)
  175       CONTINUE
            ZINV(IDIM-K,J) = (RS(IDIM-K) - Q) /
     +                            INPUT(IDIM-K,IDIM-K)
  170    CONTINUE
C
  150 CONTINUE
C
C  FIND CURRENT VECTOR AS PRODUCT ZINV*VN
C
      DO 200 I=1,IDIM
         CUR(I)=CZERO
         DO 200 J=1,IDIM
  200       CUR(I)=CUR(I)+ZINV(I,J)*VN(J)
C
C  CALCULATE CONDITION NUMBER
C
      DO 300 I=1,IDIM
         SUM_ZN = 0.0
         SUM_ZINV = 0.0
         DO 290 J=1,IDIM
            SUM_ZN = SUM_ZN+ABS(ZN(I,J))
            SUM_ZINV = SUM_ZINV+ABS(ZINV(I,J))
  290    CONTINUE
         NORM_ZN=MAX(NORM_ZN, SUM_ZN)
         NORM_ZINV=MAX(NORM_ZINV, SUM_ZINV)
  300 CONTINUE
      CN=NORM_ZN*NORM_ZINV
      WRITE (6,*) 'CONDITION NUMBER IS ',CN
C
 1000 RETURN
      END
```

2.3 *Subroutine FIND_CURRENTS*

```
C*********************************************************************
C                                                                    **
C     >>>>  SUBROUTINE FIND_CURRENTS  <<<<                           **
C                                                                    **
C     THIS SUBROUTINE CALCULATES THE EQUIVALENT CURRENTS ALONG       **
C     THE STRIP AND ALONG THE MIDDLE OF THE SLAB.                    **
C                                                                    **
C*********************************************************************
C     CALLED BY:  PBFSTRIP
C     SUBROUTINES CALLED:  NONE
C     FUNCTIONS CALLED:  NONE
C     COMMON BLOCKS:  ALL, FIELD_AMPLITUDES, IMPEDANCE
C
C     >>>>  INTERNAL VARIABLES  <<<<
C     M  =  DO-LOOP INDEX
C     N  =  PHYSICAL BASIS FUNCTION INDEX
C     X  =  X-COORDINATE OF CURRENT-SAMPLING POINT
C     Y  =  Y-COORDINATE OF CURRENT-SAMPLING POINT
C     JEQX  =  X-DIRECTED POLARIZATION CURRENT
C     JEQY  =  Y-DIRECTED POLARIZATION CURRENT
C     JCX  =  X-DIRECTED CONDUCTION CURRENT
C
C*********************************************************************
      SUBROUTINE FIND_CURRENTS
C
      IMPLICIT NONE
      REAL*8 X, Y
      INTEGER N, M
      COMPLEX*16 JEQX, JEQY, JCX
C
C --- C O M M O N   B L O C K S ---------------------------------
C
         COMPLEX*16 CJ
         REAL*8 PI, EO, MUO,ETA,KO,THETA,ER,H,W,F(3),G(3),DELX,DELY
         INTEGER XNODES, YNODES
      COMMON / ALL / PI, CJ, EO, MUO, ETA, KO, THETA, ER, H, W, F, G,
     &               XNODES, YNODES, DELX, DELY
C
         COMPLEX*16 CX(3), CY(3), CZ(3)
      COMMON / FIELD_AMPLITUDES / CX, CY, CZ
C
         INTEGER NUMMPS
         REAL*8 CN
         COMPLEX*16 Z(40,3), V(40), ZN(3,3), VN(3), CUR(3)
      COMMON / IMPEDANCE / Z, V , ZN, VN, CUR, CN, NUMMPS
C
C  STEP ACROSS STRIP AND SLAB, 0.05 WAVELENGTH AT A TIME
C
```

```
      DO 30 M=0,W*20
         X=-W/2. + 0.05*M
         Y=0
         JEQX=(0.,0.)
         JEQY=(0.,0.)
         JCX=(0.,0.)
C
C  ADD UP THE CURRENTS FOR EACH PBF
C
         DO 10 N=1,3
            JEQX=JEQX+CUR(N)*CX(N)*DSIN(F(N)*H/2.)*EXP(-CJ*G(N)*X)
            JEQY=JEQY+CUR(N)*CY(N)*DCOS(F(N)*H/2.)*EXP(-CJ*G(N)*X)
            JCX=JCX+CUR(N)*CZ(N)*EXP(-CJ*G(N)*X)
   10    CONTINUE
         JEQX=JEQX*CJ*(KO/ETA)*(ER-1.)
         JEQY=JEQY*CJ*(KO/ETA)*(ER-1.)
C
C  WRITE RESULTS TO ASCII AND UNFORMATTED FILES
C
         WRITE (31) SNGL(M), SNGL(ABS(JEQX)),
     &              SNGL(ABS(JEQY)), SNGL(ABS(JCX))
         WRITE (21,20) X, ABS(JEQX), ABS(JEQY), ABS(JCX)
   20    FORMAT (1X, F5.2, 3(3X, E20.10))
   30 CONTINUE
C
       RETURN
       END
```

2.4 *Subroutine ERRORS*

```
C*************************************************************************
C                                                                        **
C     >>>>  SUBROUTINE ERRORS  <<<<                                      **
C                                                                        **
C     THIS SUBROUTINE CALCULATES THE RMS AND AVERAGE RELATIVE            **
C     ERRORS ASSOCIATED WITH THE RESIDUAL OF THE LEAST-SQUARES           **
C     NORMALIZATION.                                                     **
C                                                                        **
C*************************************************************************
C     CALLED BY:  PBFSTRIP
C     SUBROUTINES CALLED:  NONE
C     FUNCTIONS CALLED:  NONE
C     COMMON BLOCKS:  IMPEDANCE
C
C     >>>>  INTERNAL VARIABLES  <<<<
C     RMS_ERROR  =  RMS ERROR
C     AVG_REL_ERROR  =  AVERAGE RELATIVE ERROR
C     ERROR  =  INTERMEDIATE TERM
C     II  =  DO-LOOP INDEX
C
C*************************************************************************
      SUBROUTINE ERRORS
C
      IMPLICIT NONE
      REAL*8 ERROR, RMS_ERROR, AVG_REL_ERROR
      INTEGER II
C
C --- C O M M O N   B L O C K S ----------------------------------
C
         INTEGER NUMMPS
         REAL*8 CN
         COMPLEX*16 Z(40,3), V(40), ZN(3,3), VN(3), CUR(3)
      COMMON / IMPEDANCE / Z, V , ZN, VN, CUR, CN, NUMMPS
C
C  CALCULATE THE ROOT-MEAN-SQUARE (RMS) ERROR
C
      RMS_ERROR=0.0
      DO 10 II=1,NUMMPS
         ERROR=ABS(Z(II,1)*CUR(1)+Z(II,2)*CUR(2)+
     &           Z(II,3)*CUR(3)-V(II))**2
         RMS_ERROR=RMS_ERROR+ERROR
   10 CONTINUE
      RMS_ERROR=DSQRT(RMS_ERROR/NUMMPS)
C
C  CALCULATE THE AVERAGE RELATIVE ERROR
C
      AVG_REL_ERROR=0.0
      DO 20 II=1,NUMMPS
```

```
      ERROR=ABS((Z(II,1)*CUR(1)+Z(II,2)*CUR(2)+
     &           Z(II,3)*CUR(3)-V(II))/V(II))
      AVG_REL_ERROR=AVG_REL_ERROR+ERROR
   20 CONTINUE
      AVG_REL_ERROR=AVG_REL_ERROR/NUMMPS
      WRITE (20,30) RMS_ERROR, AVG_REL_ERROR
   30 FORMAT (' RMS Error = ', F8.6, ',  Avg Rel Error = ', F8.6)
C
      RETURN
      END
```

2.5 *Subroutine RADIATE*

```
C*************************************************************************
C                                                                        **
C     >>>>  SUBROUTINE RADIATE <<<<                                      **
C                                                                        **
C     THIS SUBROUTINE CALCULATES THE RMS AND AVERAGE RELATIVE            **
C     ERRORS ASSOCIATED WITH THE RESIDUAL OF THE LEAST-SQUARES           **
C     NORMALIZATION.                                                     **
C                                                                        **
C*************************************************************************
C     CALLED BY:  PBFSTRIP
C     SUBROUTINES CALLED:  NONE
C     FUNCTIONS CALLED:  SINC
C     COMMON BLOCKS:  ALL, FIELD_AMPLITUDES, IMPEDANCE
C
C     >>>>  INTERNAL VARIABLES  <<<<
C     N  =  PHYSICAL BASIS FUNCTION INDEX
C     ANGLE  =  BISTATIC ANGLE DO-LOOP INDEX
C     TERM1, TERM2  =  INTERMEDIATE TERMS
C     TERM  =  3X1 VECTOR CONTAINING INTERMEDIATE TERMS
C     EXPTERM  =  INTERMEDIATE TERM INVOLVING AN EXPONENTIAL
C     INTEGRAL =  PROPORTIONAL TO FIELD INTENSITY
C     PHI  =  BISTATIC ANGLE IN RADIANS
C     SINE  =  SINE OF PHI
C     COSINE  =  COSINE OF PHI
C     SINEH  =  3X1 VECTOR CONTAINING INTERMEDIATE TERMS
C     COSINEH  =  3X1 VECTOR CONTAINING INTERMEDIATE TERMS
C     SIGMA  =  TOTAL SCATTERING WIDTH IN dB
C     PBF_SIGMA  =  3X1 VECTOR CONTAINING SCATTERING WIDTH
C                   CONTRIBUTED BY THE INDIVIDUAL PBFs
C
C*************************************************************************
      SUBROUTINE RADIATE
C
      IMPLICIT NONE
      INTEGER J, N, ANGLE
      REAL*8 PHI, SINE, COSINE, SINEH(3), COSINEH(3),
     &       SIGMA, PBF_SIGMA(3), SINC
      COMPLEX*16 TERM1, TERM2, TERM(3), EXPTERM, INTEGRAL
C
C --- C O M M O N   B L O C K S ----------------------------------
C
         COMPLEX*16 CJ
         REAL*8 PI, EO, MUO,ETA,KO,THETA,ER,H,W,F(3),G(3),DELX,DELY
         INTEGER XNODES, YNODES
      COMMON / ALL / PI, CJ, EO, MUO, ETA, KO, THETA, ER, H, W, F, G,
     &               XNODES, YNODES, DELX, DELY
C
         COMPLEX*16 CX(3), CY(3), CZ(3)
```

```
      COMMON / FIELD_AMPLITUDES / CX, CY, CZ
C
         INTEGER NUMMPS
         REAL*8 CN
         COMPLEX*16 Z(40,3), V(40), ZN(3,3), VN(3), CUR(3)
      COMMON / IMPEDANCE / Z, V , ZN, VN, CUR, CN, NUMMPS
C
C  CALCULATE SOME INTERMEDIATE TERMS
C
      DO 10 N=1,3
         SINEH(N)=DSIN(F(N)*H)
         COSINEH(N)=DCOS(F(N)*H)
   10 CONTINUE
C
C  STEP THROUGH EACH BISTATIC ANGLE, ONE DEGREE AT A TIME
C
      DO 40 ANGLE=0, 180
         PHI=ANGLE*PI/180.
         SINE=DSIN(PHI)
         COSINE=DCOS(PHI)
         EXPTERM=EXP(CJ*KO*H*SINE)
         INTEGRAL=(0.,0.)
C
C  CALCULATE THE CONTRIBUTION FROM EACH PBF AND SUM THEM
C
         DO 20 N=1,3
            TERM1=EXPTERM*(CJ*KO*SINE*SINEH(N)-F(N)*COSINEH(N))+F(N)
            TERM2=EXPTERM*(CJ*KO*SINE*COSINEH(N)+F(N)*SINEH(N))
     &            -CJ*KO*SINE
            TERM(N)=CX(N)*SINE*TERM1 - CY(N)*COSINE*TERM2
            TERM(N)=CJ*(KO/ETA)*(ER-1.)*TERM(N)/
     &              (F(N)*F(N)-KO*KO*SINE*SINE)+CZ(N)*SINE
            TERM(N)=TERM(N)*SINC((KO*COSINE-G(N))*W/2.)*W
            TERM(N)=TERM(N)*CUR(N)*EXP(-CJ*KO*H*SINE/2.)
            INTEGRAL=INTEGRAL+TERM(N)
            IF (TERM(N).EQ.0*CJ) THEN
               PBF_SIGMA(N)=-100.0
            ELSE
               PBF_SIGMA(N)=10*DLOG10(.25*KO*ETA*ETA*ABS(TERM(N))**2)
            ENDIF
   20    CONTINUE
         SIGMA=10.*DLOG10(0.25*KO*ETA*ETA*ABS(INTEGRAL)**2)
C
C  WRITE THE RESULTS TO ASCII AND UNFORMATTED FILES
C
         WRITE (32) SNGL(ANGLE), (SNGL(PBF_SIGMA(N)), N=1,3),
     &             SNGL(SIGMA)
         WRITE (22,30) ANGLE, (PBF_SIGMA(N), N=1,3), SIGMA
   30    FORMAT (1X, I4, 4(3X, F10.4))
   40 CONTINUE
C
```

```
RETURN
END
```

III. Auxiliary Functions

3.1 Function HANK0

```
C*************************************************************************
C                                                                        **
C     >>>>  FUNCTION HANK0(X)  <<<<                                      **
C                                                                        **
C     THIS FUNCTION RETURNS THE HANKEL FUNCTION OF THE SECOND            **
C     KIND OF ORDER 0 OF POSITIVE, REAL ARGUMENT X.  THE                 **
C     ALGORITHM IS BASED ON THE SERIES EXPANSIONS FOUND                  **
C     IN AMS-55.                                                         **
C                                                                        **
C*************************************************************************
C     SUBROUTINES CALLED:  NONE
C     FUNCTIONS CALLED:  NONE
C     COMMON BLOCKS:  HANKEL
C
C     >>>>  INTERNAL VARIABLES  <<<<
C     XX    =  X*X/9 IF X <= 3, OR 3/X IF X > 3
C     LOGTERM  =  NATURAL LOAGRITHM OF X
C     MAG   =  MAGNITUDE OF HANK1 WHEN X > 3
C     PHASE =  PHASE OF HANK1 WHEN X > 3
C     I     =  DO-LOOP INDEX
C*************************************************************************
      COMPLEX*16 FUNCTION HANK0(X)
C
      IMPLICIT NONE
      REAL*8 X, XX, LOGTERM, MAG, PHASE
      INTEGER I
C
C --- C O M M O N   B L O C K S ----------------------------------
C
         REAL*8 J0(0:6), Y0(0:6), BETA(0:6), MAG0(0:6), PHASE0(0:6)
         COMPLEX*16 ALPHA(0:6)
         REAL*8 J1(0:6), Y1(0:6), MAG1(0:6), PHASE1(0:6)
      COMMON / HANKEL / J0, Y0, ALPHA, BETA, MAG0, PHASE0,
     &                  J1, Y1, MAG1, PHASE1
C
C
C X MUST BE A POSITIVE, REAL NUMBER.  RETURN ZERO IF X=0.
C
      HANK0=(0., 0.)
      IF (X.LT.0.0) X=-X
      IF (X.EQ.0.0) RETURN
C
C  IF X IS LESS THAN OR EQUAL TO 3, USE THE SMALL-ARGUMENT POLYNOMIAL
C  EXPANSION FOUND IN AMS-55 EQUATIONS 9.4.1 AND 9.4.2
C
```

```
      IF (X.LE.3) THEN
         XX=X*X/9.DO
         LOGTERM=DLOG(X)
         HANKO=ALPHA(6)-(0., 1.)*BETA(6)*LOGTERM
         DO 10 I=5,0,-1
            HANKO=HANKO*XX + ALPHA(I)-(0., 1.)*BETA(I)*LOGTERM
   10    CONTINUE
C
C  IF X IS GREATER THAN 3, USE THE LARGE-ARGUMENT ASYMPTOTIC
C  EXPANSION FOUND IN AMS-55 EQUATION 9.4.3
C
      ELSE
         XX=3.DO/X
         MAG=MAGO(6)
         PHASE=PHASEO(6)
         DO 30 I=5,0,-1
            PHASE=PHASE*XX+PHASEO(I)
            MAG=MAG*XX+MAGO(I)
   30    CONTINUE
         HANKO=EXP(-(0., 1.)*(PHASE+X))*MAG/DSQRT(X)
      ENDIF
C
      RETURN
      END
```

3.2 Function HANK1

```
C***********************************************************************
C                                                                      **
C     >>>>  FUNCTION HANK1(X)  <<<<                                    **
C                                                                      **
C     THIS FUNCTION RETURNS THE HANKEL FUNCTION OF THE SECOND          **
C     KIND OF ORDER 1 OF POSITIVE, REAL ARGUMENT, X.  THE              **
C     ALGORITHM IS BASED ON THE SERIES EXPANSIONS FOUND                **
C     IN AMS-55.                                                       **
C                                                                      **
C***********************************************************************
C     SUBROUTINES CALLED:  NONE
C     FUNCTIONS CALLED:  NONE
C     COMMON BLOCKS:  ALL, HANKEL
C
C     >>>>  INTERNAL VARIABLES  <<<<
C     XX    =  X/3 IF X <= 3, OR 3/X IF X > 3
C     REAL  =  REAL PART OF HANK1 WHEN X <= 3
C     IMAG  =  IMAGINARY PART OF HANK1 WHEN X <= 3
C     MAG   =  MAGNITUDE OF HANK1 WHEN X > 3
C     PHASE =  PHASE OF HANK1 WHEN X > 3
C     I     =  DO-LOOP INDEX
C***********************************************************************
      COMPLEX*16 FUNCTION HANK1(X)
C
      IMPLICIT NONE
      REAL*8 X, XX, REAL, IMAG, MAG, PHASE
      INTEGER I
C
C --- C O M M O N   B L O C K S ---------------------------------
C
         COMPLEX*16 CJ
         REAL*8 PI, EO, MUO,ETA,KO,THETA,ER,H,W,F(3),G(3),DELX,DELY
         INTEGER XNODES, YNODES
      COMMON / ALL / PI, CJ, EO, MUO, ETA, KO, THETA, ER, H, W, F, G,
     &               XNODES, YNODES, DELX, DELY
C
         REAL*8 JO(0:6), YO(0:6), BETA(0:6), MAGO(0:6), PHASEO(0:6)
         COMPLEX*16 ALPHA(0:6)
         REAL*8 J1(0:6), Y1(0:6), MAG1(0:6), PHASE1(0:6)
      COMMON / HANKEL / JO, YO, ALPHA, BETA, MAGO, PHASEO,
     &                  J1, Y1, MAG1, PHASE1
C
C
C X MUST BE A POSITIVE, REAL NUMBER.  RETURN ZERO IF X=0.
C
      HANK1=(0., 0.)
      IF (X.LT.0.0) X=-X
      IF (X.EQ.0.0) RETURN
```

```
C
C  IF X IS LESS THAN 3, USE THE SMALL-ARGUMENT SERIES EXPANSION
C  FOUND IN AMS-55 EQUATIONS 9.4.4 AND 9.4.5
C
      IF (X.LE.3) THEN
         XX=(X*X/9.D0)
         REAL=0.D0
         REAL=J1(6)
         IMAG=Y1(6)
         DO 10 I=5,0,-1
            REAL=REAL*XX+J1(I)
   10       IMAG=IMAG*XX+Y1(I)
         REAL=REAL*X
         IMAG=(2.D0/PI)*DLOG(X/2.D0)*REAL+IMAG/X
         HANK1=REAL-CJ*IMAG
C
C  IF X IS GREATER THAN 3, USE THE LARGE-ARGUMENT ASYMPTOTIC
C  FORM FOUND IN AMS-55 EQUATION 9.4.6
C
      ELSE
         XX=3.D0/X
         MAG=MAG1(6)
         PHASE=PHASE1(6)
         DO 30 I=5,0,-1
            PHASE=PHASE*XX+PHASE1(I)
   30       MAG=MAG*XX+MAG1(I)
         PHASE=PHASE+X
         HANK1=(MAG/DSQRT(X))*EXP(-CJ*PHASE)
      ENDIF
C
      RETURN
      END
```

3.3 Function HANK2

```
C***********************************************************************
C                                                                      **
C     >>>>  FUNCTION HANK2(X)  <<<<                                    **
C                                                                      **
C     THIS FUNCTION RETURNS THE HANKEL FUNCTION OF THE SECOND          **
C     KIND OF ORDER TWO OF REAL ARGUMENT X.                            **
C                                                                      **
C***********************************************************************
C     FUNCTIONS CALLED:  HANK0, HANK1
C
      COMPLEX*16 FUNCTION HANK2(X)
C
      REAL*8 X
      COMPLEX*16 HANK0, HANK1
C
C X MUST BE A POSITIVE, REAL NUMBER.  RETURN ZERO IF X=0.
C
      HANK2=(0., 0.)
      IF (X.LT.0.0) X=-X
      IF (X.EQ.0.0) RETURN
C
C  USE A RECURRENCE RELATION TO GENERATE HANK2
C
      HANK2=(2./X)*HANK1(X)-HANK0(X)
C
      RETURN
      END
```

```
C*************************************************************************
C                                                                        **
C     >>>>  FUNCTION BINOMIAL(I,J)  <<<<                                 **
C                                                                        **
C     THIS FUNCTION RETURNS THE BINOMIAL COEFFICIENT OF                  **
C     "I CHOOSE J" WHERE I >= J.  FORMULA FROM AMS-55.                   **
C                                                                        **
C*************************************************************************
      REAL*8 FUNCTION BINOMIAL(I,J)
C
      INTEGER I, J
C
C  IF I<J THEN RETURN ZERO AND FLAG THE ERROR
C
      IF (I.LT.J) THEN
         WRITE (6,*) 'Illegal arguments in FUNCTION BINOMIAL'
         BINOMIAL=0.DO
         RETURN
      ENDIF
C
C  IF J=0 OR J=I, RETURN 1
C
      BINOMIAL=1.DO
      IF ((J.EQ.0).OR.(J.EQ.I)) RETURN
C
C  CALCULATE BINOMIAL, USING AMS-55'S EQUATION 24.1.1.C
C
      DO 10 II=1,J
   10    BINOMIAL=BINOMIAL*DBLE(I-II+1)/DBLE(J-II+1)
C
      RETURN
      END
```

3.5 Function FACT

```
C*************************************************************************
C                                                                        **
C     >>>>  FUNCTION FACT(X)  <<<<                                       **
C                                                                        **
C     THIS FUNCTION RETURNS THE FACTORIAL OF A NON-NEGATIVE              **
C     INTEGER, X.                                                        **
C                                                                        **
C*************************************************************************
      REAL*8 FUNCTION FACT(X)
C
      INTEGER X
C
C  FACT(0)=1
C
      FACT=1.
      IF (X.EQ.0) RETURN
C
C  CHECK FOR UPPER LIMIT ON X
C
      IF (X.GT.30) THEN
         WRITE (6,*) 'Overflow in SUBROUTINE FACT'
         FACT=9.9999E+32
         RETURN
      ENDIF
C
C  CHECK FOR ILLEGAL NEGATIVE X
C
      IF (X.LT.0) THEN
         WRITE (6,*) 'Negative argument in SUBROUTINE FACT'
         FACT=0.D0
         RETURN
      ENDIF
C
C  CALCULATE FACT
C
      DO 10 I=1,X
   10    FACT=FACT*DBLE(I)
C
      RETURN
      END
```

```
C*********************************************************************
C                                                                    **
C     >>>>  FUNCTION INTEGRAL_COSINE(N, A, B)  <<<<                  **
C                                                                    **
C     THIS FUNCTION CALCULATES THE DEFINITE INTEGRAL OF              **
C     COSINE(X) RAISED TO AN INTEGRAL POWER.  FORMULAS               **
C     ARE TAKEN FROM GRADSHTEYN & RYZHIK.                            **
C                                                                    **
C*********************************************************************
C     >>>>  CALLED BY:  IMNABC
C     >>>>  SUBROUTINES CALLED:  NONE
C     >>>>  FUNCTIONS CALLED:  BINOMIAL
C
C     >>>>  INTERNAL VARIABLES  <<<<
C     I  =  DO-LOOP INDEX
C
C     >>>>  DATA INPUT FROM CALLING ROUTINE  <<<<
C     N  =  POWER TO WHICH COSINE IS RAISED
C     A  =  LOWER LIMIT OF INTEGRATION
C     B  =  UPPER LIMIT OF INTEGRATION
C
C*********************************************************************
      REAL*8 FUNCTION INTEGRAL_COSINE(N, A, B)
C
      INTEGER I, N
      REAL*8 A, B, BINOMIAL
C
C  IF N=0, INTEGRAL IS TRIVIALLY EASY TO EVALUATE
C
      IF (N.EQ.0) THEN
         INTEGRAL_COSINE=B-A
C
C  IF N IS EVEN, USE GRADSHTEYN & RYZHIK'S EQUATION 2.513.3
C
      ELSEIF (MOD(N,2).NE.1) THEN
         INTEGRAL_COSINE=(BINOMIAL(N, N/2)/(2.**N))*(B-A)
         DO 10 I=0, (N/2)-1
   10       INTEGRAL_COSINE=INTEGRAL_COSINE+(0.5**(N-1))*
     &            (BINOMIAL(N,I)/(N-2.*I))*
     &            (DSIN((N-2.*I)*B)-DSIN((N-2.*I)*A))
C
C  IF N IS ODD, USE GRADSHTEYN & RYZHIK'S EQUATION 2.513.4
C
      ELSE
         INTEGRAL_COSINE=0.D0
         DO 20 I=0, (N-1)/2
   20       INTEGRAL_COSINE=INTEGRAL_COSINE+(0.5**(N-1))*
     &            (BINOMIAL(N,I)/(N-2.*I))*
```

```
     &            (DSIN((N-2.*I)*B)-DSIN((N-2.*I)*A))
      ENDIF
C
      RETURN
      END
```

3.7 Function SINC

```
C*************************************************************************
C                                                                        **
C     >>>>  FUNCTION SINC(X)  <<<<                                       **
C                                                                        **
C     THIS FUNCTION RETURNS THE SINC FUNCTION, OR SIN(X)/X, OF           **
C     A REAL NUMBER, X.                                                  **
C                                                                        **
C*************************************************************************
      REAL*8 FUNCTION SINC(X)
C
      REAL*8 X
C
C  CALCULATE SINC, USING THE LIMITING FORM IF X IS LESS THAN 0.0001
C
      SINC=1.D0
      IF (ABS(X).GT.0.0001) SINC=DSIN(X)/X
C
C  ELIMINATE ROUND-OFF ERROR IF NEAR A ZERO CROSSING
C
      IF ((ABS(SINC).LT.1.D-12).AND.(X.LT.100)) SINC=0.D0
C
      RETURN
      END
```

3.8 Function CONJUGATE

```
C*************************************************************************
C                                                                        **
C     >>>>  FUNCTION CONJUGATE(X)  <<<<                                  **
C                                                                        **
C     THIS FUNCTION RETURNS THE COMPLEX CONJUGATE OF A COMPLEX           **
C     NUMBER, X.                                                         **
C                                                                        **
C*************************************************************************
      COMPLEX*16 FUNCTION CONJUGATE(X)
C
      COMPLEX*16 X
C
C  CALCULATE CONJUGATE
C
      CONJUGATE=REAL(X)+(0.,-1.)*DIMAG(X)
C
      RETURN
      END
```

3.9 Function ARG

```
C*************************************************************************
C                                                                        **
C     >>>>  FUNCTION ARG(X)  <<<<                                        **
C                                                                        **
C     THIS FUNCTION RETURNS THE PHASE, OR ARGUMENT, OF A COMPLEX         **
C     NUMBER, X.                                                         **
C                                                                        **
C*************************************************************************
      REAL*8 FUNCTION ARG(X)
C
      COMPLEX*16 X
C
C  IF |X|=0 THEN RETURN ZERO AND FLAG THE ERROR
C
      ARG=0.D0
      IF (X.EQ.(0.,0.)) THEN
         WRITE (6,*) 'Undefined phase in subroutine ARG'
         RETURN
      ENDIF
C
C  CALCULATE ARG(X)
C
      ARG=DATAN2(DIMAG(X),DREAL(X))
C
      RETURN
      END
```