Computer Program for the Evaluation of the Propagation Characteristics of Lossy Microstrip Lines

T.E. van Deventer, L.P.B. Katehi

Computer Program for the Evaluation of the Propagation Characteristics of Lossy Microstrip Lines

T.E. van Deventer, L.P.B. Katehi

Radiation Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122

Shielded microstrip lines are widely used in microwave integrated circuits where they perform a great variety of functions. It is therefore very important to have an accurate knowledge of their characteristics, i.e. phase velocity and losses as a function of geometry and frequency. Because all interconnects analyzed in this program are within a shielded environment, losses due to surface waves, leakage and radiation are not present, and therefore only dissipative losses, including conductor loss and dielectric loss, are considered.

The FORTRAN 77 program PROPAG.F was written to compute the complex propagation constant (i.e. phase and attenuation constants) and current distribution of the hybrid modes propagating in a two-dimensional shielded multilayered microstrip structure.

1 GEOMETRY

The general version of the program solves for the microstrip modes that propagate in structures as shown in Figure 1. The development considers an infinitely long inhomogeneously-filled waveguide, with several microstrip lines on different levels in a multilayered configuration with lossy dielectric layers, as well as finite conducting strips and ground planes. In this work, both longitudinal and transverse components of the strip current are included in the computational implementation and therefore no restriction is placed on the width of the strips. The theoretical method uses a full-wave approach so the validity of the technique is not limited with respect to the operating frequency.

The conducting strips are assumed to have finite conductivity σ (**input**) and thickness t. The conductor thickness is usually small compared to the strip dimensions, however this need not be the case, especially in monolithic

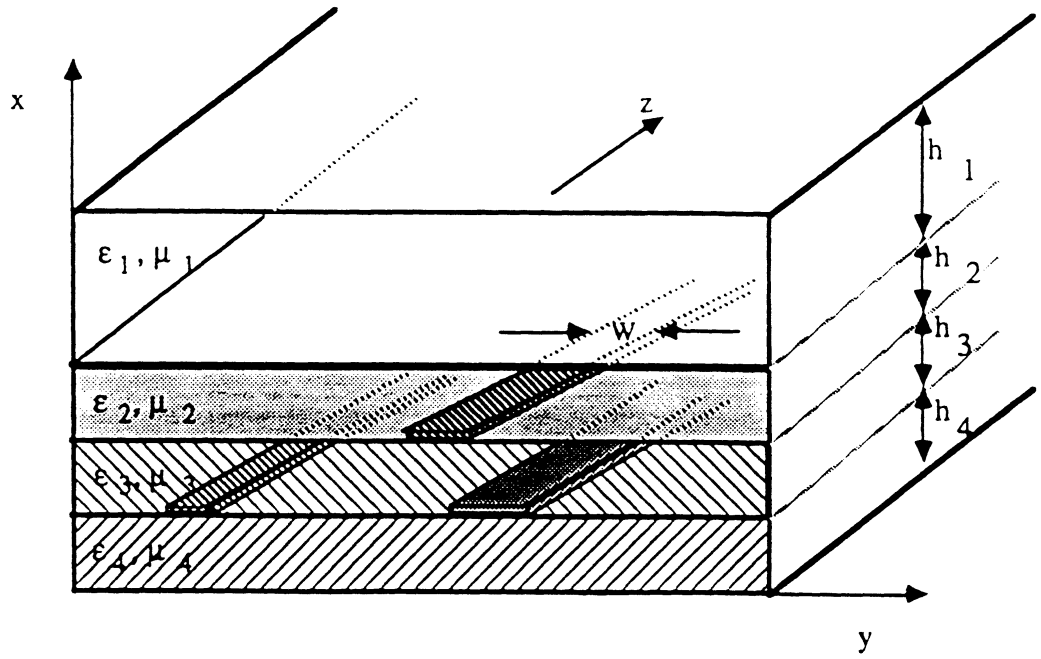


Figure 1: General shielded microstrip configuration

microwave and millimeter wave integrated circuits on GaAs. Also, in practical circuits the strips are usually at least two widths away from the side walls of the waveguide to avoid coupling, therefore losses due to finite conductivity of these walls are neglected in this derivation. However, the effect of a lossy ground plane is analyzed.

The dielectric regions are assumed nonmagnetic ($\mu_r = 1$) and have a permittivity ϵ_r (**input**). Dielectric losses are accounted for by assuming a complex permittivity for each layer i as

$$\epsilon_i = \epsilon_{ri}\,(1 - j\tan\delta_i) \tag{1}$$

which in turn implies that the propagation constant $\gamma_z \equiv jk_z$ is a complex quantity with

$$\gamma_z = \alpha + j\beta \equiv jk_z \tag{2}$$

In equation (2), α is the total attenuation of the propagating wave and β is the phase constant. In the two-dimensional problem, each microstrip mode propagates rectilinearly along the z-direction with a dependence of the form $e^{j(\omega t - k_z z)}$.

2 Formulation

An integral equation method is developed to solve for the complex propagation constant of the microstrip modes

$$\vec{E}(\vec{r}) = \int\int\int_V \bar{\bar{G}}^{\epsilon}(\vec{r}/\vec{r}') \cdot \vec{J}(\vec{r}')\,dv' \tag{3}$$

where the Green's function $\bar{\bar{G}}^e$ included in the integral equation is formulated using a generalized impedance boundary formulation [1].

The microstrip ohmic losses are evaluated through the use of an equivalent frequency-dependent impedance surface $\bar{\bar{Z}}$ which is derived by solving for the fields inside the conductors [2]. This impedance surface (**input**) replaces the conducting strips and takes into account the thickness and skin effect of the strips at high frequencies.

In view of the new boundary condition on the conductors, Pocklington's integral equation becomes

$$\hat{n} \times \bar{\bar{Z}} \cdot \vec{J}(y) = \int_{C_w} \hat{n} \times \bar{\bar{G}}^e(x, y/x', y') \cdot \vec{J}(y')\, dy' \mid_{k_z = k_z'} \tag{4}$$

The method of moments is used to solve the integral equation which results in a homogeneous system of simultaneous algebraic equations that can be solved by setting the determinant of the impedance matrix $[Z]$ equal to zero. The roots of the determinant correspond to the propagation constants of the excited modes.

3 DESCRIPTION

3.1 Menu

The menu is a user-friendly program that allows for on-screen creation of the input data file or for direct reading of the data from an input file. The program prompts for an output file name where results are to be stored. Geometrical dimensions for the inhomogeneously-filled waveguide and the strips are to be entered in meters. Conductor losses in the top and bottom walls may be included, in which case the conductivity of the walls should be given.

3.2 Program

Based on the theory presented in the previous section, the complex propagation constant in high frequency interconnects is evaluated as a function of various parameters by using Muller's algorithm with deflation. The PROPAG.F program calculates the roots of (4) by calling the *IMSL* routine *ZANLYT* and its required subroutines [3], i.e. *UGETIO, USPKD, UERTST*. The routine *ZANLYT* is a FORTRAN code which finds the zeros of a univariate complex function using Muller's method. These subroutines have to be bound to the main program PROPAG.F for the program to run. If this library is not available, another complex search routine may be substituted in the subroutine *mullerin*.

The generalized integral equation (4) is solved numerically using the method of moments. The program will prompt for a choice between subsectional basis functions (pulses) or entire domain basis functions (Chebychev polynomials). In the latter case, the basis functions result in closed-form integrals that simplify to Bessel functions of integer order. These Bessel functions are calculated in the subroutine *BESJRI* based on recursive relations [4]. The testing functions are chosen as Chebychev polynomials whose integrals are expressed in terms of spherical Bessel functions which simplify to complex sines and cosines. For the case of pulse functions, the basis and testing functions result in simple trigonometric integrals. However, use of these subsectional basis functions will result in larger matrices, and is added here for comparison purposes.

The inversion of the matrix and the calculation of the determinant are computed by $LINPACK$ routines for complex matrices.

- *subroutine CGESL(a,lda,n,ipvt,b,job)* : solves the complex system A * X = B
- *subroutine CGEDI(a,lda,n,ipvt,det,work,job)* : computes the determinant and inverse of a matrix using the factors computed by *CGECO or CGEFA*
- *subroutine CGECO(a,lda,n,ipvt,rcond,z)* : factors a complex matrix by Gaussian elimination and estimates the condition of the matrix

Each element of this matrix involves a summation over the modes of the inhomogeneously filled waveguide along the y-direction. The number of modes considered and the number of basis functions (**input**) are chosen large enough to insure convergence.

3.3 List of pertinent variables

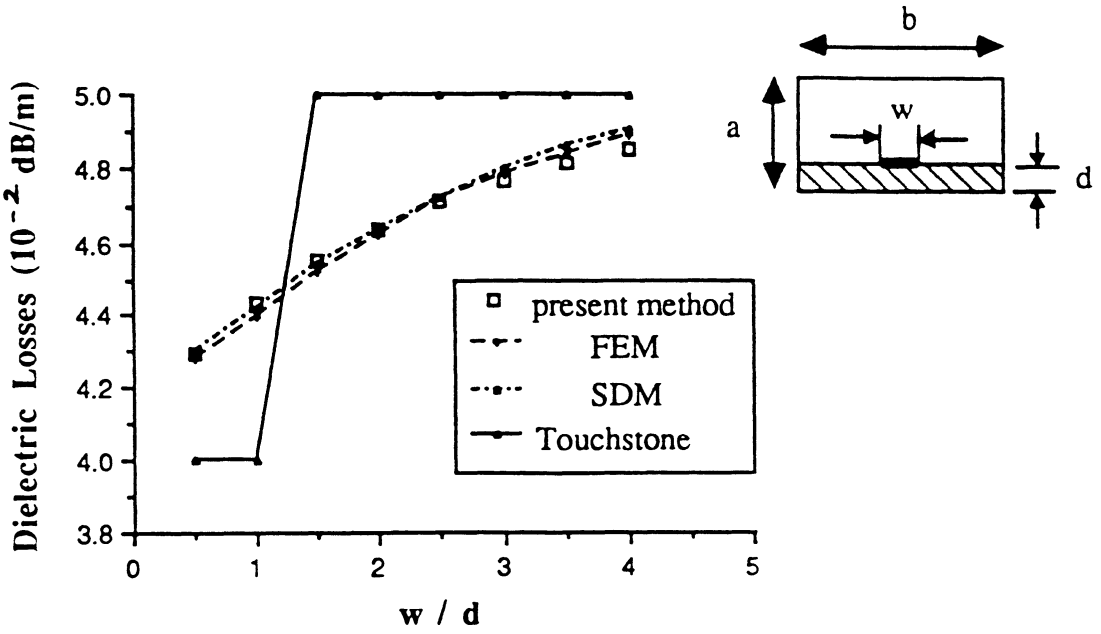
- kz : non-normalized propagation constant (complex)
- $ky(m)$: y-directed phase constant (real)
- $kx(m)$: x-directed phase constant (complex)
- det : determinant of the impedance matrix z (complex)
- jj : Bessel function of integer order (real)
- $icoeff$: Chebychev integrals for the basis and weighting functions (real)
- gyy, gyz, gzy, gzz : impedance submatrix elements (complex)
- t : Green's function components (complex)
- z : final impedance matrix containing all elements (complex)

3.4 Limitations

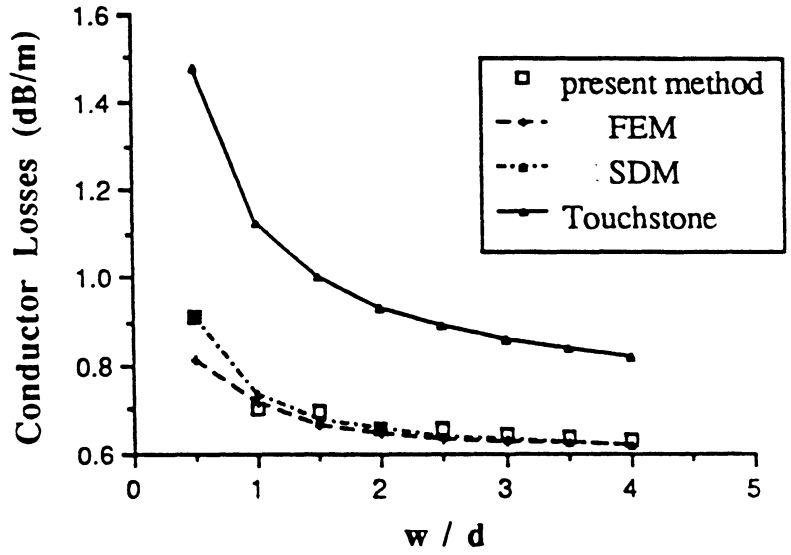
The program PROPAG.F is a generalized code that can handle any number of dielectric layers and metallic strips. However, for programming purposes, the arrays have been dimensioned to a maximum of 20 layers. In this version of the program, the strips should be at least two widths away from the side walls, and should be located either on the same interface or within two adjacent layers.

4 VERIFICATION

Using the approach described in the previous section a computer program was developed to calculate the complex propagation constant. The validity of this program has been verified in the case of thin conducting lines on lossy substrates. Good agreement with other methods such as the finite element method (FEM) and the spectral domain method (SDM) is shown in Figure 2 [1].



a. Dielectric attenuation as a function of w/d



b. Ohmic attenuation as a function of w/d

Figure 2: Conductor and dielectric losses of a single strip versus strip width (a = 10 mm, b = 20 mm, d = 1mm, ϵ_r = 10, σ = 3.33 x 10^7 S/m, tanδ = 2 x 10^{-4}, f = 1 GHz, t = 0.01 mm)

5 I/O FILES AND CODE

5.1 Screen sample

THE UNIVERSITY OF MICHIGAN COLLEGE OF ENGINEERING
RADIATION LABORATORY
ANN ARBOR, MICHIGAN

****

```
MMMM      MMMM  MMMM    MMMM   MMMM MMMM  MMMM    MMMM      MMMM    MMMM    MMMM
 MM M    M MM    MM    MM  MM  MM    MM    MM    MM  MM    MM  MM    MM M    MM
 MM  M M  MM    MM   MM        MMMMMMMM    MM   MM         MM    MM   MM  M  MM
 MM   M   MM    MM   MM        MM    MM    MM   MM  MMMM  MMMMMMMMM   MM    M MM
 MM       MM    MM    MM  MM   MM    MM    MM   MM  MM    MM     MM   MM     MMM
MMMM     MMMM  MMMM    MMMM   MMMM MMMM  MMMM    MMMM    MMMM   MMMM MMMM    MMM
```

****

THIS PROGRAM CALCULATES THE PROPAGATION CONSTANT
OF MULTIPLE LOSSY MICROSTRIP LINES
IN PARTIALLY-FILLED WAVEGUIDES

T. EMILIE VAN DEVENTER -- AND -- LINDA P. B. KATEHI

MARCH, 1993 VERSION

PAUSE: To resume execution, type: go
Any other input will terminate the program.
go
Execution resumed after PAUSE.
do you want input from screen or file (1 / 0) ?
1
Enter name of output data file;
dat.out
Enter name of plot data file;
plot.out

****

Enter the height of the waveguide a
.01
Enter the width of the waveguide b
.02
Enter the number of dielectric layers
2
Which is the source layer ?
1
Enter the height of layer 1.00000
.009
Enter the permittivity of layer 1.00000
1
Enter the loss tangent of layer 1.00000
0
Enter the height of layer 2.00000
.001
Enter the permittivity of layer 2.00000
10
Enter the loss tangent of layer 2.00000
0

****

Enter the width of strip 1
.001
Enter the horizontal position of strip 1
.01
Enter the vertical position of strip 1
0
Enter the p.u.l. resistance of strip 1
0
Enter the p.u.l. inductance of strip 1
0

****

Enter the start frequency
1e9
Enter the stop frequency
1e9
Enter the incremental frequency step
0

****

Enter the conductivity of the strip
3.33e7
Consider conductor losses (0 / 1) ?
0
Consider lower ground plane losses (0 / 1) ?
0
Consider upper ground plane losses (0 / 1) ?
0

****

Enter the start beta
1.
Enter the stop beta
5.
Enter the incremental beta step
.1
Chebychev or pulse functions (1 / 0) ?
1

****

height of the waveguide a 1.00000E-02
width of the waveguide b 2.00000E-02
number of dielectric layers 2
source layer 1
height of layer 1 is 9.00000E-03
permittivity of layer 1 is 1.00000
loss tangent of layer 1 is 0.
height of layer 2 is 1.00000E-03
permittivity of layer 2 is 10.00000
loss tangent of layer 2.00000 is 0.

****

width of strip 1 is 1.00000E-03
horizontal position of strip 1 is 1.00000E-02

```
vertical position of strip     1 is   0.
p.u.l. resistance of strip     1 is   0.
p.u.l. inductance of strip     1 is   0.

*********************************************

start frequency               1.00000E+09
stop frequency                1.00000E+09
incremental frequency step  0.

*********************************************

conductivity of the strip     3.33000E+07
losses                    0
lower ground plane losses   0
upper ground plane losses   0

*********************************************

start beta                1.00000
stop beta                 5.00000
incremental beta step     1.00000E-01
*********************************************
Chebychev basis functions


no conductor losses considered

RESULTS
-------
frequency of operation     1.00000E+09

phase constant for the lossless case     2.59262  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
atten. constant for the lossless case: alpha =   0.

phase  constant with conductor losses:  beta =   0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
atten. constant with conductor losses: alpha =   0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.

phase  constant with diel (+ cond) losses:  beta =   0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
atten. constant with diel (+ cond) losses: alpha =   0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
```

5.2 FORTRAN code

```
c**************************************************************************
c
c          t.e. van deventer
c          radiation laboratory
c          room 3121, eecs bldg
c          (313) 936-2975
c          department of electrical engineering and computer science
c          the university of michigan
c          ann arbor, mi 48105
c
c**************************************************************************
c
c  program propag.f
c
c  this program is divided into 3 parts,
c  (1) main program
c  (2) subroutines
c  (3) functions
c
c  to run the program,
c  (1)  f77 muller.f bessel.f naass.f propag.f -o propag
c  (2)  propag
c
c  topic :
c    computes the propagation constant of a microstrip mode
c    using the method of moments approach and taking into account
c    conductor losses with chebychev polynomials as basis/weighting fns.
c
c**************************************************************************
      include 'my_common.h'
c-------------------------
c     define parameters
c-------------------------
      integer ms,nbnolossroot,nrealroot,ii

      real absdet,absdet1,ermax,differ,old_losstan(10)
      real old_incrmt,error,rratio(20),attdbm(20),fopstart
      real icoeff(qdim,0:mdim,4,nstrip),jj(qdim+3,0:mdim,nstrip)
      real attdbmnol(20)

      complex t(0:mdim,3,nstrip,nstrip),determ,determ_1,det(2)
      complex kz,old_kzstop,kz_1,kznoloss,kzs,xx(20),temp(20)
      complex det_c,expd,root(20),ratio(20),xx11(20)

      character*50 config_file,output_file,plot_file,char_file
      character*50 cfg_file,ofg_file,pfg_file,chr_file

c-------------------------
c     common blocks
c-------------------------
      common/integration/icoeff
      common/modes/t
      common/bess/jj
      common/ynot/expd

      external det_c

c-------------------------
        call logo

c-------------------------
c     compute some constants
c-------------------------
      pi = 4. * atan(1.)
      mu0 = 4.e-7 * pi
      eps0=  1e-9 /(36.*pi)
      c = 1. / sqrt(mu0 * eps0)
      j = cmplx(0.,1.)

c-------------------------
c     open database
c-------------------------
      do 33 i=1,nruns
      write(*,*) 'do you want input from screen or file (1 / 0) ?'
      read(*,*) input

      if (input .eq. 1) then

         write(*,*) 'Enter name of output data file; '
         read(*,'(A50)') output_file
         ofg_file=output_file
         write(*,*) 'Enter name of plot data file; '
         read(*,'(A50)') plot_file
         pfg_file=plot_file
c         write(*,*) 'Enter name of char data file; '
c         read(*,'(A50)') char_file
c         chr_file=char_file
         open(11,file=ofg_file,status='write')
         open(12,file=pfg_file,status='write')
c         open(15,file=chr_file,status='write')
         call infoscreen

      else if (input .eq. 0) then

         write(*,*) 'Enter name of input data file; '
         read(*,'(A50)') config_file
         cfg_file=config_file
         write(*,*) 'Enter name of output data file; '
         read(*,'(A50)') output_file
         ofg_file=output_file
         write(*,*) 'Enter name of plot data file; '
         read(*,'(A50)') plot_file
         pfg_file=plot_file
c         write(*,*) 'Enter name of char data file; '
c         read(*,'(A50)') char_file
c         chr_file=char_file
         open(10,file=cfg_file)
         open(11,file=ofg_file,status='write')
         open(12,file=pfg_file,status='write')
c         open(15,file=chr_file,status='write')
         call infofile

      else
         write(*,*) 'cannot you type 1 or 0 ? '
      endif

      fopstart = fop
c-------------------------
         fop = fopstart
         ermax = 0.0
         do 101 r=1,rdim
            ermax = max(epsr(r),ermax)
  101    continue

c-------------------------
c  routine to compute the chebychev integrals (integ)
c-------------------------
      call cpu(secbeg)
```

```
         if (momint .eq. 1) then
            write(*,*) 'momint',momint
            call integcheb
         else if (momint .eq. 0) then
            write(*,*) 'momint',momint
            call integpulse
         endif

      call cpu(secend)
      sec = secend - secbeg
      write (*,*) 'elapsed time integ', sec

c-------------------------
c frequency loop
c-------------------------
      fop = fop - incrfr
      do 1000 ii=1,10000

         fop = fop + incrfr
         write(*,*) 'fop',fop
         if (fop .gt. fopstop) then
            write(*,*) 'above the frequency range'
c ???            goto 1000
            goto 33
         endif

         omega = 2. * pi * fop
         k0 = cmplx(omega * sqrt(mu0 * eps0),0.)

         if ((groundup.eq.1) .or. (groundlo.eq.1) .or. (losscond.eq.1))
     &   then
            amp = sqrt(pi*fop*mu0/sig)
         else
            amp = 0.0
         endif
         zs = cmplx(amp,amp)
c         write(*,*) 'zs',zs

         kz = kzstart * k0
         kzs = kzstop * k0
c         kzstart = kz
         kznoloss = kz
c--------------------------------
c     store the initial input data
c--------------------------------
         old_incrmt = incrmt
         old_kzstop = kzstop
         do 8 r=1,rdim
            old_losstan(r) = loss_tan(r)
  8      continue

  17     continue

c--------------------------------
c     set the lossless parameters
c      - lcheck refers to losses vs. no losses
c      - losscheck refers to dielectric losses (if they are so large that need to incorp
orate them incrementally)
c      - losscond refers to conductor losses vs. no c.l.
c--------------------------------
      nbnolossroot = 0
      nrealroot = 0
      icondflag = 0

      kz = kznoloss
      incrmt = old_incrmt

      do 11 r=1,rdim
         loss_tan(r) = 0.0
  11  continue

      kz_1 = cmplx(0.0,0.0)
      determ_1 = cmplx(0.0,0.0)

      do 143 jkj = 1,20
         xx(jkj) = cmplx(0.0,0.0)
         xxll(jkj) = cmplx(0.0,0.0)
 143  continue
c-------------------------
c  loop to compute the kz root for lossless case
c-------------------------

  12  do 20 ms=1,10000

         kz = kz + incrmt * k0

         if (real(kz) .gt. real(kzs)) then
            goto 13
         endif

         rkz = real(kz) / k0
         if (rkz .gt. (sqrt(ermax*1.)+10.)) then
             write (*,*) 'beta > quasi-static value'
             goto 990
         endif

         determ = cmplx(0.,0.)

         call impedance(kz,det)

         determ = det(1)
         absdet = sqrt(real(determ)**2 + aimag(determ)**2)
         absdet1 = sqrt(real(determ_1)**2 + aimag(determ_1)**2)
         differ = abs(absdet - absdet1)
         error = 1e-8
         if (differ .lt. error) then
             write (*,*) 'diffs < error',differ,error
             goto 13
         endif

c------------------------- refinement loop-----------------------------

c     the loop will be used if
c     the real or imaginary parts of the determinant
c     changed sign since the previous case

        if ((real(determ_1) .lt. 0.0 .and. real(determ) .ge. 0.0)
     & .or. (real(determ_1) .gt. 0.0 .and. real(determ) .le. 0.0)) then

           write (*,*) 'refinement loop'
           nbnolossroot = nbnolossroot + 1
           root(nbnolossroot) = kz
           write(*,*) 'lossless root ',nbnolossroot,root(nbnolossroot)
        endif
        determ_1 = determ

  20 continue
```

```
   13 continue
      kznoloss = kz
      rkz = real(kz) / k0
      write (*,*) ' '
      write (*,*) 'check for lossless case'
      write (*,*) ' '

c-----------------------
c   Muller's method for lossless case
c-----------------------
      do 133 nb=1,nbnolossroot
         xx(nb) = root(nb)
         expd = det(2) - cmplx(4.,0.)
 133  continue

c      write(*,*) 'initial guesses',xx

      call mullerin(xx,nbnolossroot)

      do 807 nn=1,nbnolossroot
         ratio(nn)  = xx(nn) / k0
         rratio(nn) = real(ratio(nn))
         attdbm(nn) = - aimag(xx(nn)) * 8.686
         write(*,*) 'n,beta,alpha',nn,rratio(nn),attdbm(nn)
         if ((rratio(nn).le.0.0) .or. (abs(attdbm(nn)).ge. 1.0)) then
            write(*,*) 'not a good root'
         else
            nrealroot = nrealroot + 1
            xxll(nrealroot) = xx(nn)
         endif
  807 continue

c-----------------------
c  sort roots
c-----------------------
      write(*,*) 'xxll',xxll
      do 121 jk=2,nrealroot
         aa = xxll(jk)
         do 111 ik=jk-1,1,-1
            if (real(xxll(ik)) .le. real(aa) ) goto 107
            xxll(ik+1) = xxll(ik)
 111     continue
         ik = 0
 107     xxll(ik+1) = aa
 121  continue
      write(*,*) 'xxll',xxll
      do 122 jk=1,nrealroot
         temp(jk) = xxll(nrealroot+1-jk)
 122  continue
      do 123 jk=1,nrealroot
         xxll(jk) = temp(jk)
 123  continue
      write(*,*) 'xxll',xxll

c-----------------------
c  call to calculate the current distribution
c-----------------------
      do 307 nn=1,nrealroot

         bnolnorm(nn) = real(xxll(nn) / k0 )
         attdbmnol(nn) = - aimag(xxll(nn)) * 8.686
         write(*,*) 'n,beta for lossless case',nn,bnolnorm(nn)
         write(12,*) 'n',nn,bnolnorm(nn),xxll(nn)

         call cpu(secbeg)
         call impedance(xx(nn),det)
         call cpu(secend)
         sec = secend - secbeg
         write (*,*) 'elapsed time impedance', sec

         call cpu(secbeg)
         call current1(xx(nn))
         call cpu(secend)
         sec = secend - secbeg
         write (*,*) 'elapsed time current', sec

  307 continue
      write(12,*) 'lossless',fop,bnolnorm,curratio

c-----------------------
c   Muller's method for conductor losses
c-----------------------
      if (losscond .eq. 0) then
         write (*,*) ' '
         write (*,*) 'no conductor losses considered'
         write (*,*) ' '
         goto 57
      else
         write (*,*) ' '
         write (*,*) 'checking for conductor losses'
         write (*,*) ' '
         icondflag = 1
         write(*,*) 'xxll before ',xxll
         call mullerin(xxll,nrealroot)
         write(*,*) 'xxll after ',xxll
         do 312 nn=1,nrealroot
            bcondnorm(nn) = real(xxll(nn) / k0 )
            aconddbm(nn) = - aimag(xxll(nn)) * 8.686
            write(*,*) 'n,beta,alpha for cond',nn,
     &           bcondnorm(nn), aconddbm(nn)
            write(12,*) 'n',nn,bcondnorm(nn),xxll(nn)
            call cpu(secbeg)
            call impedance(xxll(nn),det)
            call cpu(secend)
            sec = secend - secbeg
            write (*,*) 'elapsed time impedance', sec

            call cpu(secbeg)
            call current1(xxll(nn))
            call cpu(secend)
            sec = secend - secbeg
            write (*,*) 'elapsed time current', sec

  312    continue
c         call current
         do 313 nn=1,nrealroot
            write(12,*) fop,zc(nn),bnolnorm(nn),
     &           bdielnorm(nn),adieldbm(nn),curratio
  313    continue
      endif
c-----------------------
c   Muller's method for dielectric losses
c-----------------------
   57    continue
         do 16 r=1,rdim
            if (loss_tan(r) .lt. old_losstan(r)) then
               write (*,*) ' '
               write (*,*) 'checking for dielectric losses'
```

```
            write (*,*) ' '
            write(*,*) 'r,loss_tan,old_losstan',r,loss_tan(r),
     &      old_losstan(r)
            loss_tan(r) = old_losstan(r)

            if (loss_tan(r) .gt. 0.1) then

               do 55 icr=1,100
c                   loss_tan(r) = old_losstan(r) * 0.2 * icr
c                   loss_tan(r) = old_losstan(r) * 10.**(-5+icr)
                    loss_tan(r) = old_losstan(r) * 0.01 * icr
                    write (*,*) ' '
                    write (*,*) 'loop,new and old tan',icr,loss_tan(r)
     &                         ,old_losstan(r)
                    write (*,*) ' '
                    call mullerin(xx11,nrealroot)
                    write (*,*) 'loss_tan,xx',loss_tan(r),xx11

                    do 308 nn=1,nrealroot
                       bdielnorm(nn) = real(xx11(nn) / k0 )
                       adieldbm(nn) = - aimag(xx11(nn)) * 8.686
                       write(*,*) 'n,tan,beta,alpha for diel',nn,
     &                      loss_tan(r),bdielnorm(nn), adieldbm(nn)
  308               continue

c????
                    call current1(xx(nn))
                    do 309 nn=1,nrealroot
                       write(12,*) fop,loss_tan(r),bnolnorm(nn),
     &                      bdielnorm(nn),adieldbm(nn),curratio
  309               continue
   55            continue

            else

               call mullerin(xx11,nrealroot)
               do 310 nn=1,nrealroot
                  bdielnorm(nn) = real(xx11(nn) / k0 )
                  adieldbm(nn) = - aimag(xx11(nn)) * 8.686
                  write(*,*) 'n,tan,beta,alpha for diel',nn,
     &                loss_tan(r),bdielnorm(nn), adieldbm(nn)
  310          continue

c????
               call current1(xx(nn))
               do 311 nn=1,nrealroot
                  write(12,*) fop,loss_tan(r),bnolnorm(nn),
     &                bdielnorm(nn),adieldbm(nn),curratio
  311          continue
            endif

         else
                  write(12,*) fop,loss_tan(r),bnolnorm(nn),
     &                bdielnorm(nn),adieldbm(nn),curratio
         endif
   16 continue
                  write(12,*) fop,loss_tan(r),bnolnorm(nn),
     &                bdielnorm(nn),adieldbm(nn),curratio

  990 continue
      call writeout
 1000 continue
   33 continue
      close(12)
```

```
      stop
      end
ccsssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
c                        subroutines
ccsssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
      subroutine impedance(kz,det)
c
c   computes the four impedance matrices gyy, gyz, gzy, gzz
c      _                                                  _
c     |                                                    |
c     |   gyy                  |    gyz                    |
c     |   qdim x qdim             qdim x qdim             |
c     |                        |                           |
c     |   -   -   -   -   -    -   -   -   -   -   -     |
c     |   gzy                  |    gzz                    |
c     |   qdim x qdim             qdim x qdim             |
c     |_                       |                          _|
c
c
c       where qdim is the highest order of chebyshev polynomials used
c
c-------------------------
      include 'my_common.h'
c-------------------------
      integer q,p,m,nop,nsp,ipvt(2*nstrip*qdim),job

      real icoeff(qdim,0:mdim,4,nstrip)
      real rcond,deltay,deltaz,sjz,sjy

      complex gyy(qdim,qdim,nstrip,nstrip),gyz(qdim,qdim,nstrip,nstrip)
      complex gzy(qdim,qdim,nstrip,nstrip),gzz(qdim,qdim,nstrip,nstrip)
      complex za(2*nstrip*qdim,2*nstrip*qdim)
      complex z(2*nstrip*qdim,2*nstrip*qdim),kz
      complex t(0:mdim,3,nstrip,nstrip)
      complex syye,syze,szye,szze
      complex zz(2*nstrip*qdim),work(2*nstrip*qdim),det(2)
c-------------------------
      common/integration/icoeff
      common/modes/t
      common/submatrices/gyy,gyz,gzy,gzz,z
      common/sub/za
c-------------------------
c set the variables to zero
c-------------------------

      do 200 nsp=1,nstrip
         do 200 nop=1,nstrip
            do 200 p=1,qdim
               do 200 q=1,qdim
                  do 205 m=0,mdim
                     t(m,1,nop,nsp) = cmplx(0.,0.)
                     t(m,2,nop,nsp) = cmplx(0.,0.)
                     t(m,3,nop,nsp) = cmplx(0.,0.)
  205             continue
                  gyy(q,p,nop,nsp) = cmplx(0.,0.)
                  gyz(q,p,nop,nsp) = cmplx(0.,0.)
                  gzy(q,p,nop,nsp) = cmplx(0.,0.)
                  gzz(q,p,nop,nsp) = cmplx(0.,0.)
  200 continue

c-------------------------
c  call the routine that calculates the lse and lsm contributions (hybrid)
c-------------------------
      call hybrid(kz)
```

```
      rkz = real(kz) / k0
c--------------------------- loops -----------------------------
      do 215 nsp=1,nstrip
      do 215 nop=1,nstrip
      do 215 p=1,qdim
      do 215 q=1,qdim

            syye = cmplx(0.,0.)
            syze = cmplx(0.,0.)
            szye = cmplx(0.,0.)
            szze = cmplx(0.,0.)
c---------------------------- m - loop --------------------------------
            do 250 m=0,mdim

c               write(*,*) 'm,nop,nsp,t,i',m,nop,nsp,t(m,1,nop,nsp),
c     &                     icoeff(p,m,2,nsp)
               syye = syye + t(m,1,nop,nsp) * icoeff(p,m,2,nsp) *
     &                icoeff(q,m,4,nop)
               syze = syze + t(m,2,nop,nsp) * icoeff(p,m,1,nsp) *
     &                icoeff(q,m,4,nop)
               szye = szye - t(m,2,nop,nsp) * icoeff(p,m,2,nsp) *
     &                icoeff(q,m,3,nop)
               szze = szze + t(m,3,nop,nsp) * icoeff(p,m,1,nsp) *
     &                icoeff(q,m,3,nop)

c        if (m .eq. mdim) then
c           write(*,*) 'm,nsp,nop,p,q,s',m,nsp,nop,p,q,aimag(syye),
c     &                 real(syze),real(szye),aimag(szze)
c        endif

  250       continue

c
c-----------orthogonality properties of chebychev functions------------
c
            if (p .eq. q .and. q .eq. 1 .and. nop .eq. nsp) then
                deltay = 1./2.
                deltaz = 1.
            else if (p .eq. q .and. q .ne. 1 .and. nop .eq. nsp) then
                deltay = 1./2.
                deltaz = 1./2.
            else
                deltay = 0.
                deltaz = 0.
            endif
            sjy = w(nop) * pi /2. * deltay
            sjz = w(nop) * pi /2. * deltaz
            gyy(q,p,nop,nsp) =  syye - losscond *zs *sjy *icondflag
            gyz(q,p,nop,nsp) =  syze
            gzy(q,p,nop,nsp) =  szye
            gzz(q,p,nop,nsp) =  szze - losscond*zc(nop)*sjz *icondflag

c            write(*,*) 'losscond,zs,zc,sjy,sjz,icondflag',losscond,zs,
c     &                  zc(nop),sjy,sjz,icondflag
c            write(*,*) 'q,p,nop,nsp,gyy,gyz,gzy,gzz'
c            write(*,*) q,p,nop,nsp,gyy(q,p,nop,nsp),
c     &           gyz(q,p,nop,nsp),gzy(q,p,nop,nsp),gzz(q,p,nop,nsp)
c       write(12,*) rkz,aimag(gyy(q,p,nop,nsp)),real(gyz(q,p,nop,nsp)),
c     &        real(gzy(q,p,nop,nsp)),aimag(gzz(q,p,nop,nsp))

  215 continue

c-------------------------
c   fill up the complete matrix
c-------------------------
      call fill

      do 298 q=1,2*nstrip*qdim
         do 298 p=1,2*nstrip*qdim
            za(q,p) = z(q,p)
  298 continue

c-------------------------
c   calculate the determinant of the impedance matrix
c-------------------------
      call cgeco(z,2*nstrip*qdim,2*nstrip*qdim,ipvt,rcond,zz)
      job = 10
      call cgedi(z,2*nstrip*qdim,2*nstrip*qdim,ipvt,det,work,job)

      curratio = -gyy(1,1,1,1)/gyz(1,1,1,1)

      rkz = real(kz) / k0
      write(*,*) 'rekznor,kz,det,cur', rkz,kz,det(1),det(2),curratio
c      write(12,*) rkz,kz,real(det(1)),real(det(2))
c      write (*,*) ' '
      return
      end

cssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
      subroutine current(root)
c-------------------------
c   computes the two-dimensional current distribution
c-------------------------
      include 'my_common.h'
c-------------------------
      integer q,p,py,pz,p1,p2,iv(2*qdim*nstrip-1)
      integer job,ier,n,ntotdim,l

      real del,normy(nstrip),normz(nstrip),x,theta
      real u(qdim),t(qdim),sumy,sumz
c      cdy(nstrip,199),cdz(nstrip,199)
c      real rdynorm(nstrip,199),rdznorm(nstrip,199)
c      real idynorm(nstrip,199),idznorm(nstrip,199)
c      real cdynorm(nstrip,199),cdznorm(nstrip,199)
c      real phasey(nstrip,199),phasez(nstrip,199)

      complex za(2*qdim*nstrip,2*qdim*nstrip)
c     qra(2*qdim*nstrip-1)
      complex zcur(2*qdim*nstrip,2*qdim*nstrip-1)
      complex bb(2*qdim*nstrip),sv1(2*qdim*nstrip-1)
      complex iy(nstrip,qdim),iz(nstrip,qdim)
      complex xf(2*qdim*nstrip),xx(2*qdim*nstrip-1)
c      complex t1(2*qdim*nstrip),t2(2*qdim*nstrip)
c      complex t3(2*qdim*nstrip),t4(2*qdim*nstrip)
      complex root

      common/sub/za
c-------------------------
      ntotdim = 2*nstrip*qdim

      do 300 q=1,ntotdim
      do 300 p=1,ntotdim
         write(*,*) 'q,p,za',q,p,za(q,p)
 300  continue

c-------------------------
c   all current coefficients are normalized w.r.t. iy (q=1,p=1,nstrip=1)
c-------------------------
```

```
c - - first column on the other side of the set of eqs. - - - -

c-      write(*,*) ' '
c-      do 301 q=1,ntotdim
c-         do 302 p=1,ntotdim-1
c-            zcur(q,p) = za(q,p+1)
c-            write(*,*) 'q,p,zcur',q,p,zcur(q,p)
c-  302    continue
c-         bb(q) = - za(q,1)
c-         write(*,*) 'q,bb',q,p,bb(q)
c-  301 continue

c - - last column on the other side of the set of eqs. - - - -

c      write(*,*) ' '
       do 301 q=1,ntotdim
          do 302 p=1,ntotdim-1
c-             zcur(q,p) = za(q,p+1)
              zcur(q,p) = za(q,p)
c             write(*,*) 'q,p,zcur',q,p,zcur(q,p)
  302     continue
c-          bb(q) = - za(q,1)
          bb(q) = - za(q,ntotdim)
c         write(*,*) 'q,bb',q,p,bb(q)
  301 continue


c!!!!!!
        curratio = bb(1)
        write(*,*) 'curratio',curratio
c!!!!!!

      if (ntotdim-1 .eq. 1) then
         goto 399
      endif

      do 303 q=1,ntotdim-1
         xx(q) = cmplx(0.0,0.0)
         iv(q) = 0
  303 continue

c-------------------------
c   calculate the current vector (cqrdc and cqrsl are naas routines)
c-------------------------
      job = 1
      call cqrdc(zcur,ntotdim,ntotdim,ntotdim-1,qra,iv,sv1,job)
      job = 101
      call cqrsl(zcur,ntotdim,ntotdim,ntotdim-1,qra,bb,t1,t2,xx,t3,t4,
     &           job,ier)
      write(*,*) 'ier',ier

c      write(*,*) ' '
c      do 304 q=1,ntotdim
c         write(*,*) 'q,bb',q,bb(q),t4(q)
c  304  enddo

c      write(*,*) ' '
c      do 305 q=1,ntotdim-1
c         write(*,*) 'q,xx',q,xx(q),iv(q)
c  305 enddo

c-------------------------
c   unscramble from pivoting (xf = sol. minimizing the least square error)
c-------------------------
      do 306 q=1,ntotdim-1
         xf(iv(q)) = xx(q)
  306 continue
      do 307 q=1,ntotdim-1
         xx(q) = xf(q)
  307 continue
c-      xf(1) = cmplx(1.,0.)
      xf(ntotdim) = cmplx(1.,0.)
      do 308 q=1,ntotdim-1
c-       xf(q+1) = xx(q)
         xf(q) = xx(q)
  308 continue
      write(*,*) 'xf',xf

c-------------------------
c   store the current components in respective vectors
c-------------------------
      do 310 ns=1,nstrip
         py = 0
         pz = 0

c - - - - y component - - - - -

         p1 = 2*(ns-1)*qdim+1
         do 314 p=p1,p1+qdim-1
            py = py+1
            iy(ns,py) = xf(p)
  314    continue

c - - - - z component - - - - -

         p2 = p1 + qdim
         do 316 p=p2,p2+qdim-1
            pz = pz+1
            iz(ns,pz) = xf(p)
  316    continue

  310 continue

     do 320 ns=1,nstrip
        do 322 q=1,qdim
          write (*,*) 'ns,q,iy,iz',ns,q,iy(ns,q),iz(ns,q)
          write (12,*) 'ns,q,iy,iz',ns,q,iy(ns,q),iz(ns,q)
  322   continue
  320 continue

c-------------------------------------------------------------
c      open(15,file='char.dat')
      write(15,*) a,b
      write(15,*) fop,' 1 ',' 1 '
      do 321 r=1,rdim
          write(15,*) h(r),epsr(r),' 1.00 ',loss_tan(r)
 321  continue
      do 323 n=1,nstrip
          write(15,*) w(n),y0(n),xstrip(n)
 323  continue
      write(15,*) root
      do 325 ns=1,nstrip
         do 326 q=1,qdim
           write(15,*) iy(ns,q)
           write(15,*) iz(ns,q)
 326     continue
 325  continue
      do 327 i=1,2
```

```
        write(15,*) '1'
        write(15,*) '2'
        write(15,*) '3'
 327  continue
c-------------------------------------------------------------


c-------------------------
c   print the location and value of the current components
c-------------------------
c      do 320 ns=1,nstrip
c         do 322 q=1,qdim
c           argq = y0(ns) - w(ns)/2.+ (q-1./2.) * w(ns) / qdim
c           write (*,*) 'q,y,ix,iy,iz',q,argq,ix(ns,q),iy(ns,q),iz(ns,q)
c           write (11,*) 'current',q,argq,ix(ns,q),iy(ns,q),iz(ns,q)
c  322    continue
c  320 continue
c-------------------------
c-------------------------
c   calculate the current distribution
c-------------------------
 398  del = 2. / (200)
      do 390 n=1,nstrip
         normy(n) = 0.0
         normz(n) = 0.0

         do 390 l=1,199
            sumy = cmplx(0.0,0.0)
            sumz = cmplx(0.0,0.0)
            x = -1. + del * l
            theta = acos(x)

            do 395 q=1,qdim
               u(q) = sin((q+1)*theta)
               t(q) = cos((q-1)*theta)/sin(theta)
               sumy  = sumy + iy(n,q) * u(q)
               sumz  = sumz + iz(n,q) * t(q)
  395       continue

c            cdy(n,l) = sumy
c            cdz(n,l) = sumz
c            normy(n) = amax1(cabs(cdy(n,l)),normy(n))
c            if (x  .eq. 0.0) then
c               normz(n) = cabs(cdz(n,l))
c            endif

  390 continue

c-------------------------
c   normalize the current distribution
c-------------------------
c      do 360 n=1,nstrip
c      do 360 l=1,199
c         x = -1. + del * l
c         rdynorm(n,l) = real(cdy(n,l) /normy(n))
c         rdznorm(n,l) = real(cdz(n,l) /normz(n))
c         idynorm(n,l) = aimag(cdy(n,l) /normy(n))
c         idznorm(n,l) = aimag(cdz(n,l) /normz(n))
c         cdynorm(n,l) = cabs(cdy(n,l) / normy(n))
c         cdznorm(n,l) = cabs(cdz(n,l) / normz(n))
c         phasey(n,l) = atan2(aimag(cdy(n,l) / normy(n)),
c     &                       real(cdy(n,l) / normy(n)))
c         phasez(n,l) = atan2(aimag(cdz(n,l) / normz(n)),
c     &                       real(cdz(n,l) / normz(n)))
c      write(12,*) x,rdynorm(n,l),rdznorm(n,l),idynorm(n,l),idznorm(n,l)
c
c 360  continue
c-------------------------
 399  continue
      return
      end

cssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
      subroutine current1(root)
c-------------------------
c   computes the two-dimensional current distribution
c-------------------------
      include 'my_common.h'
c-------------------------
      integer q,p,py,pz,p1,p2
      integer job,n,ntotdim,l,ipvt(2*nstrip*qdim-1)

      real del,normy(nstrip),normz(nstrip),x,theta,rcond
      real u(qdim),t(qdim),sumy,sumz
c     cdy(nstrip,199),cdz(nstrip,199)
c      real rdynorm(nstrip,199),rdznorm(nstrip,199)
c      real idynorm(nstrip,199),idznorm(nstrip,199)
c      real cdynorm(nstrip,199),cdznorm(nstrip,199)
c      real phasey(nstrip,199),phasez(nstrip,199)

      complex za(2*qdim*nstrip,2*qdim*nstrip)
c     qra(2*qdim*nstrip-1)
      complex zcur(2*qdim*nstrip-1,2*qdim*nstrip-1)
      complex bb(2*qdim*nstrip-1),sv(2*qdim*nstrip-1)
      complex iy(nstrip,qdim),iz(nstrip,qdim)
c      complex xf(2*qdim*nstrip),xx(2*qdim*nstrip-1)
c      complex t1(2*qdim*nstrip),t2(2*qdim*nstrip)
c      complex t3(2*qdim*nstrip),t4(2*qdim*nstrip)
      complex root

      common/sub/za
c-------------------------
      ntotdim = 2*nstrip*qdim

      do 300 q=1,ntotdim
      do 300 p=1,ntotdim
c         write(*,*) 'q,p,za',q,p,za(q,p)
  300 continue

c-------------------------
c   all current coefficients are normalized w.r.t. iy (q=1,p=1,nstrip=1)
c-------------------------
c - - first column on the other side of the set of eqs. - - - -

c-      write(*,*) ' '
c-      do 301 q=1,ntotdim
c-         do 302 p=1,ntotdim-1
c-            zcur(q,p) = za(q,p+1)
c-            write(*,*) 'q,p,zcur',q,p,zcur(q,p)
c-  302    continue
c-         bb(q) = - za(q,1)
c-         write(*,*) 'q,bb',q,p,bb(q)
c-  301 continue

c - - last column on the other side of the set of eqs. - - - -

c      write(*,*) ' '
```

```
        do 302 p=1,ntotdim-1
           zcur(q,p) = za(q,p)
c           write(*,*) 'q,p,zcur',q,p,zcur(q,p)
  302    continue
         bb(q) = - za(q,ntotdim)
c         write(*,*) 'q,bb',q,p,bb(q)
  301 continue


c!!!!!!
        curratio = bb(1)
c        write(*,*) 'curratio',curratio
c!!!!!!

      if (ntotdim-1 .eq. 1) then
         goto 399
      endif

c-------------------------
c   calculate the current vector (cgeco and cgesl are naas routines)
c-------------------------
      call cgeco(zcur,ntotdim-1,ntotdim-1,ipvt,rcond,sv)
      job = 0
      call cgesl(zcur,ntotdim-1,ntotdim-1,ipvt,bb,job)

c      write(*,*) bb

c-------------------------
c   store the current components in respective vectors
c-------------------------
      do 310 ns=1,nstrip
         py = 0
         pz = 0

c - - - - y component - - - - -

         p1 = 2*(ns-1)*qdim+1
         do 314 p=p1,p1+qdim-1
            py = py+1
            iy(ns,py) = bb(p)
c            write(*,*) ns,py,p,iy(ns,py)
  314    continue

c - - - - z component - - - - -

         p2 = p1 + qdim
         do 316 p=p2,p2+qdim-1
            pz = pz+1
            iz(ns,pz) = bb(p)
c            write(*,*) ns,pz,p,iz(ns,pz)
  316    continue

  310 continue


c???????
      iy(1,1) = bb(1)
      iz(1,1) = bb(2)
      iy(2,1) = bb(3)
      iz(nstrip,qdim) = cmplx(1.,0.)
c???????

      do 320 ns=1,nstrip
         do 322 q=1,qdim
            write (*,*) 'ns,q,iy,iz',ns,q,iy(ns,q),iz(ns,q)
            write (12,*) 'ns,q,iy,iz',ns,q,iy(ns,q),iz(ns,q)
  322    continue
  320 continue

c-------------------------------------------------------------
c      open(15,file='char.dat')
      write(*,*) 'a',a
      write(15,*) a,b
      write(15,*) fop,' 1 ',' 1 '
      do 321 r=1,rdim
          write(15,*) h(r),epsr(r),' 1.00 ',loss_tan(r)
  321 continue
      do 323 n=1,nstrip
          write(15,*) w(n),y0(n),xstrip(n)
  323 continue
      write(15,*) root
      do 325 ns=1,nstrip
         do 326 q=1,qdim
           write(15,*) iy(ns,q)
           write(15,*) iz(ns,q)
  326    continue
  325  continue
      do 327 i=1,2
         write(15,*) '1'
         write(15,*) '2'
         write(15,*) '3'
  327 continue
c-------------------------------------------------------------


c-------------------------
c   print the location and value of the current components
c-------------------------
c      do 320 ns=1,nstrip
c         do 322 q=1,qdim
c           argq = y0(ns) - w(ns)/2.+ (q-1./2.) * w(ns) / qdim
c           write (*,*) 'q,y,ix,iy,iz',q,argq,ix(ns,q),iy(ns,q),iz(ns,q)
c           write (11,*) 'current',q,argq,ix(ns,q),iy(ns,q),iz(ns,q)
c  322    continue
c  320 continue
c-------------------------
c-------------------------
c   calculate the current distribution
c-------------------------
  398 del = 2. / (200)
      do 390 n=1,nstrip
         normy(n) = 0.0
         normz(n) = 0.0

         do 390 l=1,199
            sumy = cmplx(0.0,0.0)
            sumz = cmplx(0.0,0.0)
            x = -1. + del * l
            theta = acos(x)

            do 395 q=1,qdim
               u(q) = sin((q+1)*theta)
               t(q) = cos((q-1)*theta)/sin(theta)
               sumy  = sumy + iy(n,q) * u(q)
               sumz  = sumz + iz(n,q) * t(q)
  395       continue

c            cdy(n,1) = sumy
```

```
c            cdz(n,l) = sumz
c            normy(n) = amax1(cabs(cdy(n,l)),normy(n))
c            if (x  .eq. 0.0) then
c               normz(n) = cabs(cdz(n,l))
c            endif

  390 continue

c-------------------------
c   normalize the current distribution
c-------------------------
c      do 360 n=1,nstrip
c      do 360 l=1,199
c         x = -1. + del * l
c         rdynorm(n,l) = real(cdy(n,l) /normy(n))
c         rdznorm(n,l) = real(cdz(n,l) /normz(n))
c         idynorm(n,l) = aimag(cdy(n,l) /normy(n))
c         idznorm(n,l) = aimag(cdz(n,l) /normz(n))
c         cdynorm(n,l) = cabs(cdy(n,l) / normy(n))
c         cdznorm(n,l) = cabs(cdz(n,l) / normz(n))
c         phasey(n,l) = atan2(aimag(cdy(n,l) / normy(n)),
c     &                          real(cdy(n,l) / normy(n)))
c         phasez(n,l) = atan2(aimag(cdz(n,l) / normz(n)),
c     &                          real(cdz(n,l) / normz(n)))
c       write(12,*) x,rdynorm(n,l),rdznorm(n,l),idynorm(n,l),idznorm(n,l)
c
c 360  continue
c-------------------------
 399  continue
      return
      end

cssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
cssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
      subroutine integcheb
c-------------------------
c   computes the integration of the basis and weighting functions
c       sin(ky*y') *  t_q(x) / sqrt(1 - x**2) corresponds to i(q,m,1)
c       cos(ky*y') *  u_q(x) * sqrt(1 - x**2) corresponds to i(q,m,2)
c       sin(ky*y') *  t_q(x)                  corresponds to i(q,m,3)
c       cos(ky*y') *  u_q(x)                  corresponds to i(q,m,4)
c
c-------------------------
      include 'my_common.h'
c-------------------------
      integer q,ize,m,ncalc,n

      real arg,ky,icoeff(qdim,0:mdim,4,nstrip)
      real*8 mydble,darg,djn(qdim+3)
      real jj(qdim+3,0:mdim,nstrip)
      real sj0,sj1,sj2,sj3,sj4,sj5,sj6,sj7

      external sj0,sj1,sj2,sj3,sj4,sj5,sj6,sj7
c-------------------------
      common/integration/icoeff
      common/bess/jj
c-----------------------------------------------------------------------

c  calculate the bessel functions of integer order for all m's
c
c  as a matter of fact, the routine needs to calculate
c  the bessel functions up to (1 + highest order to be calculated)
c  the order for t_n starts at n = 0                        (even z-excitation)
c  the order for u_n starts at n = 1 to n+1 (** see below)  (odd z-excitation)
c  since the two excitations have to be of different parity
c
c-----------------------------------------------------------------------
      write(*,*) 'in integcheb'
      do 300 n=1,nstrip
         do 300 m=0,mdim
            ize = 0
            arg = m * pi / b  * w(n) / 2.
            darg = mydble(arg)
            call besjri(darg,qdim+3,ize,djn,ncalc)

            do 301 q=1,qdim+3
               jj(q,m,n) = real(djn(q))
c              write (*,*) 'q,m,n,jj(q,m,n)',q,m,n,jj(q,m,n)
  301       continue
  300    continue

c-------------------------
c  calculate the integrals
c-------------------------
      do 315 n=1,nstrip
         do 320 m=0,mdim
            ky = m * pi / b
            arg = ky * w(n) /2.

            do 330 q=1,qdim

c-----------chebychev integrals  (basis functions)

            icoeff(q,m,1,n) = -w(n)/2.*pi*cos(ky*y0(n)+q*pi/2.)*
     &                        jj(q,m,n)

            if (m .eq. 0) then
               icoeff(q,m,2,n) = 0.0
            else
               icoeff(q,m,2,n) = b/m*(q+1)*cos(ky*y0(n)+q*pi/2.)*
     &                           jj(q+2,m,n)

            endif

c-----------chebychev integrals  (weighting functions)

            if (q .eq. 1) then
              icoeff(q,m,3,n) = w(n) * sin(ky*y0(n)) * sj0(arg)
              icoeff(q,m,4,n) = -2. * w(n) * sin(ky*y0(n)) *
     &                          sj1(arg)
            else if (q .eq. 2) then
              icoeff(q,m,3,n) = w(n) * cos(ky*y0(n)) * sj1(arg)
              icoeff(q,m,4,n) = w(n)/3. * cos(ky*y0(n)) *
     &                         ( sj0(arg) - 8. * sj2(arg))
            else if (q .eq. 3) then
              icoeff(q,m,3,n) = -w(n)/3. * sin(ky*y0(n)) *
     &                          ( sj0(arg) + 4. * sj2(arg))
              icoeff(q,m,4,n) = -w(n)/5. * sin(ky*y0(n)) *
     &                          ( 4. * sj1(arg) - 16. * sj3(arg))
            else if (q .eq. 4) then
              icoeff(q,m,3,n) = -w(n)/5. * cos(ky*y0(n)) *
     &                          (3. * sj1(arg) + 8. * sj3(arg))
              icoeff(q,m,4,n) = w(n) * cos(ky*y0(n)) *
     &                          (0.2*sj0(arg)-8./7*sj2(arg)+
     &                          128. / 35 * sj4(arg))
            else if (q .eq. 5) then
              icoeff(q,m,3,n) = w(n) * sin(ky*y0(n)) *
```

```
     &                          64. / 35 * sj4(arg))
               icoeff(q,m,4,n) = -w(n) * sin(ky*y0(n)) *
     &                          (18./35*sj1(arg)-64./45*sj3(arg)+
     &                          256. / 63 * sj5(arg))
             else if (q .eq. 6) then
               icoeff(q,m,3,n) = w(n) * cos(ky*y0(n)) *
     &                          (-5.*sj1(arg)+8.*sj3(arg)+
     &                          128. * sj5(arg))
               icoeff(q,m,4,n) = w(n) * cos(ky*y0(n)) *
     &                          (1./7*sj0(arg)-16./21*sj2(arg)+
     &                          128./77*sj4(arg)-1024./231*sj6(arg))
             else if (q .eq. 7) then
               icoeff(q,m,3,n) = w(n) * sin(ky*y0(n)) *
     &                          (-1/35.*sj0(arg)+4./21*sj2(arg)
     &                          -384./385*sj4(arg)-512./231*sj6(arg))
               icoeff(q,m,4,n) = -w(n) * sin(ky*y0(n)) *
     &                          (8./21*sj1(arg)-32./33*sj3(arg)+
     &                          512./273*sj5(arg)-2048./429*sj7(arg))
             else
                write (*,*) 'not enough data for spherical bessel'
             endif
c          if (m .le. 501) then
c          write (*,*) 'm,q,n,icoeff(q,m,1,n)',m,q,n,icoeff(q,m,1,n)
c     &     ,icoeff(q,m,2,n),icoeff(q,m,3,n),icoeff(q,m,4,n)
c          endif
c----------------------------
  330    continue
  320 continue
  315 continue
      return
      end
cssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
      subroutine integpulse
c-------------------------
c   computes the integration of the basis and weighting functions
c-------------------------
      include 'my_common.h'
c-------------------------
      integer m,q,ns
      real ky,arg,icoeff(qdim,0:mdim,4,nstrip),sinc,sdel
      external sinc
c-------------------------
      common/integration/icoeff

      write(*,*) 'in integpulse'
c-------------------------
c  calculate the integrals with pulse function (y-direction)
c-------------------------
      do 310 ns=1,nstrip
         do 330 q=1,qdim
            arg = y0(ns) - w(ns) / 2. + (q-1/2.) * w(ns) / qdim

            do 320 m=0,mdim
               ky = m * pi / b
               sdel = sinc( ky * w(ns) / (2.*qdim) )
               icoeff(q,m,1,ns) = cos(ky*arg) * sdel
               icoeff(q,m,2,ns) = sin(ky*arg) * sdel
               icoeff(q,m,3,ns) = icoeff(q,m,1,ns)
               icoeff(q,m,4,ns) = icoeff(q,m,2,ns)
  320       continue

  330    continue
  310 continue
      return
      end
cssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
      subroutine hybrid(kz)
c-------------------------
c  computes the arrays corresponding to lse and lsm common terms
c       ls(1) corresponds to lse modes (electric)
c       ls(2) corresponds to lsm modes (electric)
c
c    -----cannot have eps or mu equal to zero !!!!---------
c
c   t(m,1) corresponds to tyye
c   t(m,2) corresponds to tyze = tzye
c   t(m,3) corresponds to tzze
c-------------------------
      include 'my_common.h'
c-------------------------
      integer m,nop,nsp,i
      real ky,xp,x,delta
      complex kx(10),k(10),kz,t(0:mdim,3,nstrip,nstrip)
c     complex argn,argd,xp1(2),xp2(2),xp3(2),xp4(2),xpd(2),xd(2)
      complex ls(2),zcc(2,10),z0(2,0:11),z0_1(2,0:11),zu(2),zl(2)
      complex aya,rat,ratio
      complex xppxmh,xpmxmh,xpmxph,twg,bbb,xdep,xpdep,cm,deno,zsg
      logical region1

      external twg,rat
c-------------------------
      common/modes/t
c-------------------------
      if (icondflag .eq. 1) then
          zsg = zs
      else
          zsg = cmplx(0.0,0.0)
      endif

      do 400 r=1,rdim
         er_c(r)  = epsr(r)*cmplx(1.00,- loss_tan(r))
         k(r) = k0 * sqrt(er_c(r))
  400 continue

c---------------------------------------------------------------------
      do 405 nop=1,nstrip
         x = xstrip(nop)

         do 410 nsp=1,nstrip
            xp = xstrip(nsp)

            if ((x .le. xp).and.(x .ge. 0.0)) then
               region1 = .false.
c               write(*,*) 'in region2,nop,nsp',nop,nsp
            else if ((x.le.h(rdiel)).and.(x.gt.xp)) then
               region1 = .true.
c               write(*,*) 'in region1,nop,nsp',nop,nsp
            endif

c- - - - - -m-loop - - - - - - -

            do 415 m=0,mdim
               ky = m * pi / b

               do 420 r=1,rdim
                  kx(r) = -j * sqrt(ky**2 + kz**2 - k(r)**2 )
                  aya = ky**2 + kz**2 - k(r)**2
```

```
                if ((abs(real(kx(r)))  .lt. 1e-8) .and.
     &             (abs(aimag(kx(r))) .lt. 1e-8)) then
                   kz = kz + cmplx(0.01,0.0)
                   kx(r) = -j * sqrt(ky**2 + kz**2 - k(r)**2 )
                   write (*,*) 'flag 0 --- kx = 0'
                endif
c                write(*,*) 'm,r,argument,ky,kz,k',m,r,aya,ky,kz,k(r)
                zcc(1,r) = omega * mu0 / kx(r)
                zcc(2,r) = kx(r) / (omega * eps0 * er_c(r))

  420        continue

c- - - - ratio of cos(kx (x-h)) cos (kx x') / cos(kx h) - - - - - - - -

           xppxmh = rat( kx(rdiel) , xp+x-h(rdiel) , h(rdiel) )

           if (region1) then
              xpmxph = rat( kx(rdiel) , xp-x+h(rdiel) , h(rdiel) )
              ratio = ( xppxmh + xpmxph ) / 2.
           else
              xpmxmh = rat( kx(rdiel) , xp-x-h(rdiel) , h(rdiel) )
              ratio = ( xppxmh + xpmxmh ) / 2.
           endif

c-----------------------------------------------------------------------
c  calculates the LSE and LSM terms
c-----------------------------------------------------------------------
         do 430 i=1,2
            z0(i,0) = zsg * groundup
            z0(i,rdim+1) = -zsg * groundlo

c-----------dielectric layers impedances-------------------

           if (rdiel .ne. 1) then
              do 440 r=1,rdiel-1
                 bbb = zcc(i,r)+j*z0(i,r-1)*twg(kx(r)*h(r))
                 if ((abs(real(bbb)).lt. 1e-8).and.
     &               (abs(aimag(bbb)).lt.1e-8)) then
                    write (*,*) 'flag2  den=0'
                    z0(i,r) =  z0_1(i,r)
                 else
                    z0(i,r) = zcc(i,r) *
     &                  (z0(i,r-1)+j*zcc(i,r)*twg(kx(r)*h(r)))/
     &                  (zcc(i,r)+j*z0(i,r-1)*twg(kx(r)*h(r)))
                 endif
                 z0_1(i,r) = z0(i,r)
  440         continue
           endif

  441      if (rdiel .le. (rdim-1)) then
              do 450 r=rdim,rdiel+1,-1
                 z0(i,r) = zcc(i,r) *
     &                   (z0(i,r+1)+j*zcc(i,r)*twg(-kx(r)*h(r)))/
     &                   (zcc(i,r)+j*z0(i,r+1)*twg(-kx(r)*h(r)))
  450         continue
           endif

  442      zu(i) = z0(i,rdiel-1) / zcc(i,rdiel)
           zl(i) = z0(i,rdiel+1) / zcc(i,rdiel)
           deno  = j*(zl(i)-zu(i)) + twg(kx(rdiel) * h(rdiel)) *
     &             (1.-zl(i)*zu(i))

c----------- x and x' dependence ------------------
           if (region1) then
              xpdep =  twg(kx(rdiel)*xp) + j * zl(i)
              xdep  =  twg(kx(rdiel)*(x-h(rdiel))) + j * zu(i)
           else
              xpdep =  twg(kx(rdiel)*(xp-h(rdiel))) + j * zu(i)
              xdep  =  twg(kx(rdiel)*x)  + j * zl(i)
           endif

           ls(i) = xpdep * xdep * ratio / deno

c                  if (m.gt.130) then
c                  write(*,*) 'ls,i',ls(i),i
c                  endif

  430     continue

c         write(*,*) 'm,zc ',m,zcc(1,rdiel),zcc(2,rdiel)
c         write(*,*) 'm,z0 ',m,z0(1,rdiel-1),z0(1,rdiel+1),
c     &                        z0(2,rdiel-1),z0(2,rdiel+1)
c         write(*,*) 'm,zul',m,zu(1),zl(1),zu(2),zl(2)
c-----------------------------------------------------------------------
c  calculates the 4 main impedance matrices
c-----------------------------------------------------------------------
c         write (*,*) 'm,n,nn,ls(1),ls(2)',m,n,nn,ls(1),ls(2)


          if (m .eq. 0) then
             delta = 1.
          else
             delta = 2.
          endif
          cm = delta / b / (kz**2 + ky**2)

          t(m,1,nop,nsp) = j * cm * ( kz**2 * zcc(1,rdiel) * ls(1) +
     &                                ky**2 * zcc(2,rdiel) * ls(2) )
          t(m,2,nop,nsp) = cm * ky * kz *
     &                  ( zcc(1,rdiel) * ls(1) - zcc(2,rdiel) * ls(2) )
          t(m,3,nop,nsp) = j * cm * ( ky**2 * zcc(1,rdiel) * ls(1) +
     &                                kz**2 * zcc(2,rdiel) * ls(2) )

c      if (m.lt.10) then
c      write(*,*) 'm,nop,nsp,t',m,nop,nsp,t(m,1,nop,nsp),t(m,2,nop,nsp),
c     &  t(m,3,nop,nsp)
c      endif

  415 continue
  410 continue
  405 continue

      return
      end
cssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
      subroutine fill
c-------------------------
      include 'my_common.h'
c-------------------------
      integer q,p,pp,qq,p1,p2,p3,p4,q1,q2,q3,q4,nop,nsp,n
      complex z(2*nstrip*qdim,2*nstrip*qdim)
      complex gyy(qdim,qdim,nstrip,nstrip),gyz(qdim,qdim,nstrip,nstrip)
      complex gzy(qdim,qdim,nstrip,nstrip),gzz(qdim,qdim,nstrip,nstrip)
c-------------------------
      common/submatrices/gyy,gyz,gzy,gzz,z
c-------------------------
```

```
      do 580 q=1,2*nstrip*qdim
            z(q,p) = cmplx(0.,0.)
 580 continue

     do 500 nop=1,nstrip
        q1 = qdim*(nop-1)+1
        q2 = q1+qdim-1
        do 500 q=q1,q2
           qq = q - qdim*(nop-1)

           do 500 nsp=1,nstrip
              p1 = 2*(nsp-1)*qdim+1
              p2 = p1+qdim-1
              p3 = p2+1
              p4 = p3+qdim-1

              do 510 p=p1,p2
                 pp = p-p1+1
                  z(q,p) = gyy(qq,pp,nop,nsp)
 510          continue
              do 520 p=p3,p4
                 pp = p-p3+1
                 z(q,p) = gyz(qq,pp,nop,nsp)
 520          continue
 500 continue

     do 530 n=nstrip,2*nstrip-1
        nop = n - nstrip+1
        q3 = qdim*n+1
        q4 = q3+qdim-1
        do 530 q=q3,q4
           qq = q - qdim*n

           do 530 nsp=1,nstrip
              p1 = 2*(nsp-1)*qdim+1
              p2 = qdim*(2*nsp-1)
              p3 = (2*nsp-1)*qdim+1
              p4 = qdim*2*nsp

              do 540 p=p1,p2
                 pp = p - 2*(nsp-1)*qdim
                  z(q,p) = gzy(qq,pp,nop,nsp)
 540          continue
              do 550 p=p3,p4
                 pp = p - (2*nsp-1)*qdim
                 z(q,p) = gzz(qq,pp,nop,nsp)
 550          continue
 530 continue

     return
     end
cssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
      subroutine mullerin(xxl,nb)
c-------------------------
      include 'my_common.h'
c-------------------------
      integer nsig,kn,nguess,n,itmax,infer(10),ier,nb
      real eps
      complex xxl(10),det_c,expd

      common/ynot/expd
      external det_c
c-------------------------
      eps = 1e-8
      nsig = 7
      kn = 0
      nguess = nb
      n = nb
      itmax = 100
c      write(*,*) 'nb,init g.',nb,xxl

      call zanlyt(det_c,eps,nsig,kn,nguess,n,xxl,itmax,infer,ier)

      return
      end
csssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
      subroutine cpu (sec)
c-------------------------
c this routine calculate the cpu time in second since the beginning
c of a run
c-------------------------
c      implicit real *8 (s)
c      real *8 sec
c      integer *2 clock(3)
c-------------------------
c%include '//sw_master/sys/ins/base.ins.ftn'
c%include '//sw_master/sys/ins/time.ins.ftn'
c-------------------------
c      call proc1_$get_cput (clock)
c      call cal_$float_clock(clock,sec)

      return
      end
csssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
      subroutine infoscreen
c-------------------------
      include 'my_common.h'
c-------------------------
c     define parameters
c-------------------------

      write(*,*)
      write(*,*) '************************************************'
      write(*,*)
      write (*,*) 'Enter the height of the waveguide a'
          read (*,*) a
      write (*,*) 'Enter the width of the waveguide b'
          read (*,*) b

          write (*,*) 'Enter the number of dielectric layers '
          read (*,*) rdim
      write (*,*) 'Which is the source layer ?'
          read (*,*) rdiel
      htot = 0.0
      do 1 r=1,rdim
          write (*,*) 'Enter the height of layer ',r
          read (*,*) h(r)
          write (*,*) 'Enter the permittivity of layer ',r
              read (*,*) epsr(r)
c         write (*,*) 'Enter the permeability of layer ',r
c             read (*,*) mur(r)
          write (*,*) 'Enter the loss tangent of layer ',r
              read (*,*) loss_tan(r)
          htot = htot + h(r)
    1 continue
      if ( abs(htot-a)/a .gt. 0.01 ) then
         write(*,*) 'trouble with vertical dimensions'
      endif
```

```
      write(*,*)
      write(*,*) '*********************************************'
      write(*,*)
c         write (*,*) 'Enter the number of strips '
c         read (*,*) nstrip
      do 2 n=1,nstrip
          write (*,*) 'Enter the width of strip ',n
          read (*,*) w(n)
          write (*,*) 'Enter the horizontal position of strip ',n
              read (*,*) y0(n)
          write (*,*) 'Enter the vertical position of strip ',n
              read (*,*) xstrip(n)
          write (*,*) 'Enter the p.u.l. resistance of strip ',n
              read (*,*) rsc(n)
          write (*,*) 'Enter the p.u.l. inductance of strip ',n
              read (*,*) xsc(n)
          zc(n) = cmplx(rsc(n),xsc(n))
    2 continue

      write(*,*)
      write(*,*) '*********************************************'
      write(*,*)

      write (*,*) 'Enter the start frequency'
          read (*,*) fop
      write (*,*) 'Enter the stop frequency'
          read (*,*) fopstop
      write (*,*) 'Enter the incremental frequency step'
          read (*,*) incrfr

      write(*,*)
      write(*,*) '*********************************************'
      write(*,*)
      write (*,*) 'Enter the conductivity of the strip'
          read (*,*) sig
      write (*,*) 'Consider conductor losses (0 / 1) ?'
          read (*,*) losscond
c      if (losscond .eq. 0) then
c         groundlo = 0
c         groundup = 0
c      else
        write (*,*) 'Consider lower ground plane losses (0 / 1) ?'
        read (*,*) groundlo
        write (*,*) 'Consider upper ground plane losses (0 / 1) ?'
        read (*,*) groundup
c      endif

      write(*,*)
      write(*,*) '*********************************************'
      write(*,*)
      write (*,*) 'Enter the start beta'
          read (*,*) rkzstart
      write (*,*) 'Enter the stop beta'
          read (*,*) rkzstop
      write (*,*) 'Enter the incremental beta step'
          read (*,*) incrmt
      write (*,*) 'Chebychev or pulse functions (1 / 0) ?'
          read (*,*) momint
c      write (*,*) 'Enter the maximum number of modes'
c         read (*,*) mdim
c      write (*,*) 'Enter the maximum number of basis functions'
c         read (*,*) qdim
c         read (*,*) nruns

      kzstart = cmplx(rkzstart,0.0)
      kzstop = cmplx(rkzstop,0.0)

      call writeinput
      call writeoutput
          return
          end
cssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
      subroutine infofile
c-------------------------
      include 'my_common.h'
c-------------------------
c     define parameters
c-------------------------
c      open(unit=10, file='input.dat')

      read (10,*) a
      read (10,*) b
      read (10,*) rdim
      read (10,*) rdiel
      do 1 r=1,rdim
         read (10,*) h(r)
         read (10,*) epsr(r)
         read (10,*) loss_tan(r)
    1 continue

      do 2 n=1,nstrip
         read (10,*) w(n)
         read (10,*) y0(n)
         read (10,*) xstrip(n)
         read (10,*) rsc(n)
         read (10,*) xsc(n)
         zc(n) = cmplx(rsc(n),xsc(n))
    2 continue

      read (10,*) fop
      read (10,*) fopstop
      read (10,*) incrfr

      read (10,*) sig
      read (10,*) losscond
      read (10,*) groundlo
      read (10,*) groundup

      read (10,*) rkzstart
      read (10,*) rkzstop
      read (10,*) incrmt

      kzstart = cmplx(rkzstart,0.0)
      kzstop = cmplx(rkzstop,0.0)

      read (10,*) momint
      close(10)

      call writeinput
      call writeoutput
      return
      end
cssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
      subroutine writeinput
c-------------------------
```

```
c-------------------------
      write(*,*) ' '
      write(*,*) ' '
      write(*,*)
      write(*,*) '*********************************************'
      write(*,*)
      write (*,*) 'height of the waveguide a',a
      write (*,*) 'width of the waveguide b ',b

          write (*,*) 'number of dielectric layers ',rdim
      write (*,*) 'source layer ',rdiel
      do 1 r=1,rdim
          write (*,*) 'height of layer       ',r,' is ',h(r)
          write (*,*) 'permittivity of layer ',r,' is ',epsr(r)
          write (*,*) 'loss tangent of layer ',r,' is ',loss_tan(r)
  1   continue

      write(*,*)
      write(*,*) '*********************************************'
      write(*,*)
      do 2 n=1,nstrip
          write (*,*) 'width of strip               ',n,' is ',w(n)
          write (*,*) 'horizontal position of strip ',n,' is ',y0(n)
          write (*,*) 'vertical position of strip   ',n,' is ',xstrip(n)
          write (*,*) 'p.u.l. resistance of strip   ',n,' is ',rsc(n)
          write (*,*) 'p.u.l. inductance of strip   ',n,' is ',xsc(n)
  2   continue

      write(*,*)
      write(*,*) '*********************************************'
      write(*,*)

      write (*,*) 'start frequency           ',fop
      write (*,*) 'stop frequency            ',fopstop
      write (*,*) 'incremental frequency step',incrfr

      write(*,*)
      write(*,*) '*********************************************'
      write(*,*)
      write (*,*) 'conductivity of the strip ',sig
      write (*,*) 'losses                    ', losscond
      write (*,*) 'lower ground plane losses ',groundlo
      write (*,*) 'upper ground plane losses ',groundup

      write(*,*)
      write(*,*) '*********************************************'
      write(*,*)
      write (*,*) 'start beta           ',rkzstart
      write (*,*) 'stop beta            ',rkzstop
      write (*,*) 'incremental beta step',incrmt
      write(*,*) '*********************************************'
      if (momint .eq. 1) then
         write (*,*) 'Chebychev basis functions
      else if (momint .eq. 0) then
         write (*,*) 'Pulse basis functions '
      endif
      write(*,*) ' '
      write(*,*) ' '

      return
      end
cssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
      subroutine writeoutput
c-------------------------
      include 'my_common.h'
c-------------------------
c      open(unit=11, file='output.dat')

      write(11,*)
      write(11,*) '*********************************************'
      write(11,*)
      write(11,*) 'height of the waveguide a',a
      write(11,*) 'width of the waveguide b ',b

          write(11,*) 'number of dielectric layers ',rdim
      write(11,*) 'source layer ',rdiel
      do 1 r=1,rdim
          write(11,*) 'height of layer       ',r,' is ',h(r)
          write(11,*) 'permittivity of layer ',r,' is ',epsr(r)
          write(11,*) 'loss tangent of layer ',r,' is ',loss_tan(r)
  1   continue

      write(11,*)
      write(11,*) '*********************************************'
      write(11,*)
      do 2 n=1,nstrip
          write(11,*) 'width of strip               ',n,' is ',w(n)
          write(11,*) 'horizontal position of strip ',n,' is ',y0(n)
          write(11,*) 'vertical position of strip   ',n,' is ',xstrip(n)
          write(11,*) 'p.u.l. resistance of strip   ',n,' is ',rsc(n)
          write(11,*) 'p.u.l. inductance of strip   ',n,' is ',xsc(n)
  2   continue

      write(11,*)
      write(11,*) '*********************************************'
      write(11,*)

      write(11,*) 'start frequency           ',fop
      write(11,*) 'stop frequency            ',fopstop
      write(11,*) 'incremental frequency step',incrfr

      write(11,*)
      write(11,*) '*********************************************'
      write(11,*)
      write(11,*) 'conductivity of the strip ',sig
      write(11,*) 'losses                    ', losscond
      write(11,*) 'lower ground plane losses ',groundlo
      write(11,*) 'upper ground plane losses ',groundup

      write(11,*)
      write(11,*) '*********************************************'
      write(11,*)
      write(11,*) 'start beta           ',rkzstart
      write(11,*) 'stop beta            ',rkzstop
      write(11,*) 'incremental beta step',incrmt

      if (momint .eq. 1) then
         write (11,*) 'Chebychev basis functions
      else if (momint .eq. 0) then
         write (11,*) 'Pulse basis functions '
      endif

      return
      end
cssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
      subroutine writeout
c-------------------------
      include 'my_common.h'
```

```
c-------------------------

      write(11,*) ' '
      write(11,*) 'RESULTS'
      write(11,*) '-------'
      write(11,*) 'frequency of operation ', fop
      write(11,*) ' '
      write(11,*) 'phase constant for the lossless case ', bnolnorm
      write(11,*) ' '
      write(11,*) 'phase  constant with conductor losses:  beta = ',
     &    bcondnorm
      write(11,*) 'atten. constant with conductor losses: alpha = ',
     &    aconddbm
      write(11,*) ' '
      write(11,*) 'phase  constant with diel (+ cond) losses:  beta = ',
     &    bdielnorm
      write(11,*) 'atten. constant with diel (+ cond) losses: alpha = ',
     &    adieldbm

c      close(11)

      write(*,*) 'RESULTS'
      write(*,*) '-------'
      write(*,*) 'frequency of operation ', fop
      write(*,*) ' '
      write(*,*) 'phase constant for the lossless case ', bnolnorm
      write(*,*) 'atten. constant for the lossless case: alpha = ',
     &    attdbmnol
      write(*,*) ' '
      write(*,*) 'phase  constant with conductor losses:  beta = ',
     &    bcondnorm
      write(*,*) 'atten. constant with conductor losses: alpha = ',
     &    aconddbm
      write(*,*) ' '
      write(*,*) 'phase  constant with diel (+ cond) losses:  beta = ',
     &    bdielnorm
      write(*,*) 'atten. constant with diel (+ cond) losses: alpha = ',
     &    adieldbm
      return
      end
cssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
      subroutine logo
c           THE UNIVERSITY OF MICHIGAN COLLEGE OF ENGINEERING
c                        RADIATION LABORATORY
c                        ANN ARBOR, MICHIGAN
c
c ***************************************************************************
c ***************************************************************************
c
c MMMM      MMMM  MMMM    MMMM  MMMM MMMM   MMMM    MMMM       MMMM     MMMM    MMMM
c  MM M    M MM    MM    MM   MM   MM    MM    MM   MM  MM     MM  MM     MM M    MM
c  MM  M M  MM    MM    MM         MM    MM    MM   MM         MM  MM     MM  M   MM
c  MM   M   MM    MM    MM         MM    MM    MM   MMMM       MMMMMMMM   MM    M MM
c  MM       MM    MM    MM   MM    MM    MM    MM   MM         MM  MM     MM      MMM
c MMMM     MMMM  MMMM    MMMM  MMMM MMMM   MMMM    MMMM       MM  MM     MMMM    MMM
c
c ***************************************************************************
c ***************************************************************************
c
c
c                                                        MARCH 11, 1993 VERSION


      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) '          THE UNIVERSITY OF MICHIGAN COLLEGE OF ENGI',
     &'NEERING'
      WRITE(*,*) '                      RADIATION LABORATORY'
      WRITE(*,*) '                      ANN ARBOR, MICHIGAN'
      WRITE(*,*) ' '
      WRITE(*,*) '*****************************************************',
     &'************************'
      WRITE(*,*) '*****************************************************',
     &'************************'
      WRITE(*,*) ' '
      WRITE(*,*) 'MMMM      MMMM  MMMM    MMMM  MMMM MMMM   MMMM    MMMM  ',
     &'    MMMM    MMMM    MMMM'
      WRITE(*,*) ' MM M    M MM    MM    MM   MM   MM    MM    MM  MM  MM',
     &'    MM  MM     MM M    MM  MM  MM '
      WRITE(*,*) ' MM  M M  MM    MM    MM         MM    MM    MM  MM  MM',
     &'    MM  MM     MM  M   MM  MM      '
      WRITE(*,*) ' MM   M   MM    MM    MM         MM    MM    MM  MM  MM',
     &'    MMMMMMMM   MM    M MM  MM      '
      WRITE(*,*) ' MM       MM    MM    MM   MM    MM    MM    MM  MM  MM',
     &'    MM  MM     MM      MMM  MM  MM '
      WRITE(*,*) 'MMMM     MMMM  MMMM    MMMM  MMMM MMMM   MMMM    MMMM  ',
     &'    MM  MM     MMMM    MMM  MMMM  '
      WRITE(*,*) ' '
      WRITE(*,*) '*****************************************************',
     &'************************'
      WRITE(*,*) '*****************************************************',
     &'************************'
      WRITE(*,*) ' '
      WRITE(*,*) 'THIS PROGRAM CALCULATES THE PROPAGATION CONSTANT ',
     &'OF MULTIPLE LOSSY MICROSTRIP LINES '
      WRITE(*,*) 'IN PARTIALLY-FILLED WAVEGUIDES'
      WRITE(*,*) ' '
      WRITE(*,*) '     T. EMILIE  VAN DEVENTER  -- AND --  LINDA P. B.',
     &' KATEHI'
      WRITE(*,*) ' '
      WRITE(*,*) '                                                    ',
     &'  MARCH, 1993 VERSION'
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      PAUSE

      RETURN
      END



cfffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
c                          functions
cfffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
      function det_c(kz)
c
c   calls for the impedance matrix calculation
c   specifies the equation (det_c) whose roots have to be found (kz's)
c
c-------------------------
      complex kz,det(2),det_c,expd
      common/vpot/expd
```

```
c-------------------------
      det(1) = cmplx(0.,0.)
      det(2) = cmplx(0.,0.)
      call impedance(kz,det)
      det_c = det(1)* 10.0 ** (det(2) - expd)
c      write(*,*) 'det_c',det_c
      return
      end
cffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
      function sj0(arg)
      real arg,sj0
c-----------------
      if (arg .eq. 0.0) then
         sj0 = 1.0
      else
         sj0 = sin(arg) / arg
      endif
      return
      end
cffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
      function sj1(arg)
      real arg,sj1
c-----------------
      if (arg .eq. 0.0) then
         sj1 = 0.0
      else
         sj1 = sin(arg) / (arg)**2 - cos(arg) / arg
      endif
      return
      end
cffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
      function sj2(arg)
      real arg,sj2
c-----------------
      if (arg .eq. 0.0) then
         sj2 = 0.0
      else
         sj2 = sin(arg) * (3./(arg)**3 - 1./arg) - 3*cos(arg)/(arg)**2
      endif
      return
      end
cffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
      function sj3(arg)
      real arg,sj3
c-----------------
      if (arg .eq. 0.0) then
         sj3 = 0.0
      else
         sj3 = sin(arg) * (15. / arg**4 -6. / arg**2) +
     &         cos(arg) * (-15./(arg)**3 +1./arg)
      endif
      return
      end
cffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
      function sj4(arg)
      real arg,sj4
c-----------------
      if (arg .eq. 0.0) then
         sj4 = 0.0
      else
         sj4 = sin(arg)*(105./(arg)**5 -45./(arg)**3 +1./arg) +
     &         cos(arg)*(-105./(arg)**4 +10./(arg)**2)
      endif
      return
      end
cffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
      function sj5(arg)
      real arg,sj5
c-----------------
      if (arg .eq. 0.0) then
         sj5 = 0.0
      else
         sj5 = sin(arg)*(945./(arg)**6 -420./(arg)**4 +15./(arg)**2) +
     &         cos(arg)*(-945./(arg)**5 +105./(arg)**3 - 1./arg)
      endif
      return
      end
cffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
      function sj6(arg)
      real arg,sj6
c-----------------
      if (arg .eq. 0.0) then
         sj6 = 0.0
      else
         sj6 = sin(arg)*(10395./(arg)**7-4725./(arg)**5+210./(arg)**3
     &                  - 1./arg) -
     &      cos(arg)*(10395./(arg)**6 -1260./(arg)**4 + 21./(arg)**2)
      endif
      return
      end
cffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
      function sj7(arg)
      real arg,sj7
c-----------------
      if (arg .eq. 0.0) then
         sj7 = 0.0
      else
         sj7 = sin(arg)*(135135./(arg)**8-62370./(arg)**6+3150./
     &                 (arg)**4 - 28./(arg)**2 ) +
     &      cos(arg)*(-135135./(arg)**7 +17325./(arg)**5 -
     &                  378./(arg)**3 +1./arg)
      endif
      return
      end
cffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
c  function to make a single-precision real into a double-precision real
c  by adding zeros at the end of it ( using dble adds junk !!)
c  written by leland pierce
c-------------------------
       real*8 function mydble(a)
       real a
       character*30 chara

       write(chara,10)a
10     format(e30.17)

       read(chara,10)mydble

       return

       end
cffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
      function twg(arg)
      real r,ai
      complex arg,j,twg
c-----------------
      j = cmplx(0.,1.)
```

```
      if (aimag(arg) .gt. 45.0) then
         twg = j
      else if (aimag(arg) .lt. -45.0) then
         twg = -j
      else
         r = real(arg)
         ai = aimag(arg)
         twg = ( sin(r)*cosh(ai) + j*cos(r)*sinh(ai) ) /
     &          ( cos(r)*cosh(ai) - j*sin(r)*sinh(ai) )
      endif
      return
      end
cffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
      function rat(arg,pos1,pos2)
      real pos1,pos2,rn,rd,ain,aid
      complex rat,arg,argn,argd,j
c-----------------------------------------------------------------------
c  calculates the ratio  cos(arg*pos1) / cos(arg*pos2)
c-----------------------------------------------------------------------
      j = cmplx(0.,1.)
      argn = arg * pos1
      argd = arg * pos2

c-------asymptotic form for large arguments----------------------

      if (abs(aimag(argd)) .ge. 44.) then
         if (pos1 .ge. 0.) then
            rat = exp(-j * (argn - argd ))
c            write(*,*) '1,argn,argd,rat',argn,argd,rat
         else
            rat = exp(j * (argn + argd ))
c            write(*,*) '2,argn,argd,rat',argn,argd,rat
         endif

c-------asymptotic form for small arguments(l'hopital's rule)----

c      else if (abs(1.+exp(2.*j*argd)) .le. 1e-1 .and.
c     &         abs(1.+exp(2.*j*argn)) .le. 1e-1) then
c         if (pos1 .ge. 0.) then
c            rat = argn / argd * exp(-j*(argn-argd))
c            write (*,*) 'loop C1,argn,argd,rat',argn,argd,rat
c         else
c            rat = argn / argd * exp(j*(argn+argd))
c            write (*,*) 'loop C2,argn,argd,rat',argn,argd,rat
c         endif

c-------normal division ----------------------------------------

      else
         rn = real(argn)
         ain = aimag(argn)
         rd = real(argd)
         aid = aimag(argd)
         rat = ( cos(rn)*cosh(ain) - j*sin(rn)*sinh(ain) ) /
     &          ( cos(rd)*cosh(aid) - j*sin(rd)*sinh(aid) )
c         write(*,*) 'straight,argn,argd,rat',argn,argd,rat
      endif

c      write(*,*) 'pos1,pos2,argn,argd,rat',pos1,pos2,argn,argd,rat
      return
      end
cffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
      function sinc(x)
      real sinc,x
c-----------------
      if (abs(x) .le. 1e-4) then
         sinc = 1.
      else
         sinc = sin(x) / x
      endif
      return
      end
cffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
```

```
c*********************************************************************
      integer nstrip,rdim,rdiel,nruns,mdim,qdim,momint
      integer groundlo,groundup,losscond,icondflag

      real a,b,h(10),w(10),y0(10),xstrip(10),rsc(10),xsc(10)
      real fop,fopstop,incrfr,sig,rkzstart,rkzstop,incrmt,amp
      real mu0,eps0,c,pi,omega,rkz,loss_tan(10),epsr(10)
      real bnolnorm(20),bcondnorm(20),aconddbm(20)
      real bdielnorm(20),adieldbm(20)
c     attdbmnol(20)

      complex kzstart,kzstop,zs,curratio,j,zc(10),k0,er_c(10)

      PARAMETER(NSTRIP=1)
      PARAMETER(QDIM=1)
      PARAMETER(MDIM=1000)
      PARAMETER(NRUNS=1)

      common/constant/mu0,eps0,c,pi,omega,j,zc,k0,a,b,h,w,xstrip,y0
      common/permitt/er_c,loss_tan,epsr,losscond,groundup,groundlo
      common/diel/rdim,rdiel,zs,icondflag,curratio,incrmt,amp
      common/freq/fop,fopstop,incrfr,sig,kzstart,kzstop,rkz
      common/results/bnolnorm,bcondnorm,aconddbm,bdielnorm,adieldbm
      common/reject/rkzstart,rkzstop,rsc,xsc,momint
c      common/file/char_file
```

References

[1] T.E. van Deventer, P.B. Katehi, A.C. Cangellaris, "An Integral Equation Method for the Evaluation of Conductor and Dielectric Losses in High Frequency Interconnects," *IEEE Trans. Microwave Theory Tech.*, vol. MTT-37 No. 12, pp 1964-1972, December 1989.

[2] L.P. Vakanas, A.C. Cangellaris, J.L. Prince, "Frequency-Dependent [L] and [R] Matrices for Lossy Microstrip Lines", University of Arizona, 1991.

[3] IMSL User's Manual, IMSL Library, Problem-Solving Software System For Mathematical and Statistical FORTRAN Programming, Edition 9.2, Revised November 1984, IMSL LIB-0009.

[4] D. J. Sooke, "Bessel Functions I and J of Complex Argument and Integer order," *Journal of Research B of the National Bureau of Standards* , vol 77b, pp. 111 - 114, 1973.