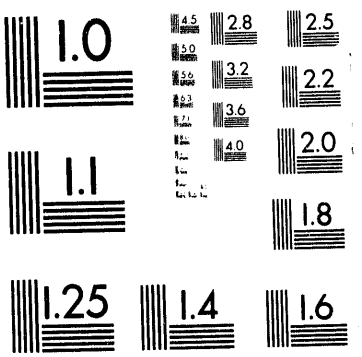
1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1
8.0
9.0

1 of 1

Analytic Formulation for the ac Electrical Conductivity in Two-temperature, Strongly Coupled, Overdense Plasma: FORTRAN Subroutine

R. Cauble

University of California, Lawrence Livermore National Laboratory,
Livermore, CA 94550

and

W. Rozmus

University of Alberta
Edmonton, Alberta T6G 2J1

Abstract

A FORTRAN subroutine for the calculation of the ac electrical conductivity in two-temperature, strongly coupled, overdense plasma is presented. The routine is the result of a model calculation based on classical transport theory with application to plasmas created by the interaction of short pulse lasers and solids. The formulation is analytic and the routine is self-contained.

Laser light absorption in short pulse laser irradiation of solids is difficult to calculate because the plasma is strongly coupled with respect to interparticle interactions. In order to find the absorption, the electrical conductivity or, equivalently, the electron-ion collision frequency must be known. The collision frequency is used in the inverse bremsstrahlung coefficient. [1] Since the driving mechanism is a laser with non-zero frequency, the electrical conductivity must be frequency-dependent, or, ac. In addition, in such plasmas, ions and electrons may have different temperatures, which leads to different absorption values over the usual single temperature case. [2]

We have expanded a strongly coupled transport theory model [3] to include the effects of frequency dependence and electron temperatures distinct from ion temperatures. The details of the calculation will appear elsewhere. [4] In the appendices are FORTRAN routines which contain the results of the calculation. Appendix A contains the subroutine (called 'rho') which performs the calculation given as input: ion density (in cm^{-3}), electron temperature (in eV), ion temperature (in eV), mean ionization state, atomic weight of the material, and the wavelength of the laser light (in nanometers). Two additional arguments finish the argument list: a switch (default on) which modifies the calculation at low temperatures ($\leq \sim 7$ eV for solid density aluminum) when an approximation used in the formulation begins to break down [The approximation involves the use of an effective interparticle potential which is not consistent with the Debye-Hückel particle

correlations also employed; as the plasma becomes more strongly coupled, this approximation becomes worse. Its use, however, allows a completely analytic result.]; and a switch (default on) which returns another analytic result due to Boercker, Rogers, and DeWitt [5] when the coupling is even lower, such that the electrons are degenerate [We have performed a classical calculation which brings quantum effects in only through the use of the effective potential.] Explanations for all variables are commented in the routine.

The routine returns: the electron plasma frequency, ω_{pe}, (in Hz), the electron-ion collision frequency (in Hz), the electron-ion collision frequency divided by ω_{pe}, the ac electrical conductivity (in Hz), the ac electrical conductivity divided by ω_{pe}, the resistivity (in micro-ohm-cm), the electron-ion collision frequency divided by ω_{pe} according to Ref. [5] (including a weak coupling multiplier referred to but not use in [5]), the resistivity according to Ref. [5] (in micro-ohm-cm), and the electron-ion collision frequency (divided by ω_{pe}) if only a 1-Sonine model were used [Most calculations are 1-Sonine equivalents. Our calculation is a 2-Sonine model (See Ref. [3]), which is necessary in strong coupling.] Two additional functions are appended to the 'rho' routine and are required. Function 'e1local' finds the exponential integral of the first kind for the independent variable x, $E_1(x)$, which is used in function 'expe1,' which returns the value of $e^x \times E_1(x)$.

Appendix B is an example FORTRAN driver for the subroutine 'rho.' As part of the driver, a function 'zbar' is included. This is the coded version of More's Thomas-Fermi mean ionization result. [6] This is convenient since the 'rho' routine requires a

value of the average ionization. Appendix C contains sample output of the driver/subroutine for several temperatures of solid density aluminum.

This report specifies that these routines can be used by anyone who requests them. The authors request that if problems arise, users of the code report these problems to us. The approximations contained in the calculation, and thus in 'rho,' are known to break down when the plasma becomes too strongly coupled. This is primarily due to the fact that Debye-Hückel correlations were used. We anticipate that the use of correlations generated by, for example, hypernetted chain (HNC) theory [7] would produce more satisfactory results (at the cost of loss of analyticity.) HNC correlations have been shown to produce quite good results for strong coupling. [7] However, a tow-temperature electron-ion HNC code does not seem to exist (yet). Clearly use of more appropriate correlations in the model would lead to better results at very strong coupling.

Appendix A

```
      subroutine rho (ni,tev,tiv,z,aw,xwavel,wpe,
     >  cfw0,cfw0wpe,econd,econdwpe,rhow0,
     >  ppxdhbr,rhoxdhbr,cfppw0wp,iweesw,ibrdsw)
c
      implicit real*8(a-h,o-z)
      common tcond,xmult,omult,wowpe,gamei

c
      real*8 me,ni,ne,mi,kdtkdb,lgeeb,lgeib,lgkdb
      real*8 kde2,kdi2,kd2,kee2,kei2,lgdel
      real*8 kde,kdi,kd2bar,kee2bar,kei2bar

      data pi,c/3.141592653589793238,2.99792458d10/
      data amu,one/1.6605655d-24,1.0d0/
      data sq2,forpiesq /1.4142135623731d0,2.898887d-18/
c     data esqr /2.3068617d-19/

c     For now, at least, restricted to T_e >= T_i
c     Returns equal temperature result at T_i otherwise

      if (tev . lt. tiv) tev = tiv

      me=0.9109534d-27
      hbar=1.05443d-27
c
c     This version uses 2-temperature DH statics everywhere
c     Matrix elements are static - no propagators
c
c     ni      - ion density in cm^-3
c     tev     - electron temperature (= ion temperature) in eV
c     tiv     - ion temperature (= ion temperature) in eV
c     z       - net ion charge (simple number)
c     aw      - atomic weight (simple number)
c     cfdh    - collision frequency / wpe
c     rhodh   - resistivity
c     xmult   - exact zero-coupling multiplier (divide 1 Sonine
result - M_pp - to
c               get approximate 2 Sonine result
c     omult   - actual multiplier
c     wpe     - electron plasma frequency
c     wowpe   - laser frequency / wpe
c     tcond   - thermal conductivity divided by k_boltzmann
c     cfw0    - ei collision frequency (Hz)
c     cfw0wpe - ei coll. freq. in units of wpe
c     cfppw0wp- M_pp ei coll freq in units of wpe
c     econd   - electrical conductivity
c     econdwpe- elect. cond in units of wpe
c     rhow0   - resistivity in micro-ohm-cm
c     rhow0wpe- restistivity in units of wpe
c     ppxdhdbr- Boercker-Rogers-DeWitt coll. freq. ( units of wpe)
c     rhoxdhbr- BRD resistivity in units of wpe
c     iweesw  - switch on (=1) or off (=0) conversion to less
pathological
c               behavior at low temperature (default on)
c     ibrdsw  - switch on (=1) or off (=0) return of Boercker-
Rogers-DeWitt
```

```
c                result at very low temperature (default on)
c
c     Given these params, the subroutine returns cf (electron-ion
collison
c     frequency) * wpe and rho (resistivity) in micro-ohm-cm.
See
c     JPP paper.
c
c
      ne = ni*z
      aion=(3.0/4.0/pi/ni)**(1.0/3.0)
      mi= amu*aw
      beta=one/(tev*1.602021d-12)
      betai=one/(tiv*1.602021d-12)
      betaei=(me+mi)/(me/betai + mi/beta)
      sbeta=dsqrt(beta)
      sbetaei=dsqrt(betaei)
      xmu=one/(one/me+one/mi)
      gamei=z*forpiesq/aion*betaei/4./pi

      constah=aion/hbar
      kei2=constah*pi*2.*xmu*constah/betaei
      kee2=constah*pi*me*constah/beta

      const=aion*aion
      kde2=forpiesq*ne*const*beta
      kdi2=forpiesq*ni*const*z*z*betai
      kd2=kde2+kdi2
      kdi=dsqrt(kdi2)
      kde=dsqrt(kde2)
      wpe2=forpiesq*ne/me
      wpe=dsqrt(wpe2)

      wavel=xwavel*1.0d-7
      wo=2.0*pi*c/wavel
      wowpe=wo/wpe

      rbeta=betaei**2/beta/betai
      delta1r=one-rbeta
      delabs=dabs(delta1r)
      deltsqrt=dsqrt(delta1r)
      kd2bar=kd2/kde/kdi
      kei2bar=kei2/kde/kdi
      kee2bar=kee2/kde/kdi

      eeteeb = kee2bar*kee2bar
      eiteib = kei2bar*kei2bar
      eeteib = kee2bar*kei2bar
      eiteeb = eeteib
      kdtkdb = kd2bar*kd2bar
      eitkdb = kei2bar*kd2bar
      eetkdb = kee2bar*kd2bar
      lgeeb = dlog(kee2bar)
      lgeib = dlog(kei2bar)
      lgkdb = dlog(kd2bar)

c
c      Approximate 2 Sonine / One Sonine ratio
       xmult=(13.0-9.0*sq2/(z+sq2))/4.0
```

```
      c         BRD term with V=Coulomb
      c         If ibrdsw = 1 and coupling is very large, this result is
returned

              balf=betaei/constah/8.0/me/constah*kd2
              dhlaminf=expe1(balf)-expe1(balf/2.)/2.
      c        kmaxinf=2./3.*dhlaminf

              co=-ni/wpe*dsqrt(2.*pi)/(8.*pi**2*dsqrt(me)/sbeta)
              cxdxdh=-(forpiesq*z)**2*beta
              cxppxdh=co*cxdxdh

              ppxdhbr=cxppxdh*2.d0/3.d0*dhlaminf/xmult
              rhoxdhbr = 4.0*pi/wpe*ppxdhbr/xmult*9.0d17


      c23456789112345678921234567893123456789412345678951234567896123456
789712


      c       Calculate static integrals in V_Deutsch, c_DH, and
      c       S_Debye-Huckel approximations for 1 or 2 temps

      c       Recall below to negate all integrals since these are all
contributions
      c        at zero; contribution at infinity is 0

      c  ***************
      c       Delta integral - 1T & 2T are same if cD_ei is constructed
from S_ei

            dconst= -(forpiesq*Z)*betaei*(forpiesq*Z)*kei2/2.

      c       1 / (x + kd^2) / (x + kei^2)

            deltaint = dlog ( kd2 / kei2 ) / (kei2 - kd2)
            delta = - dconst*deltaint

      c  ***************
      c       Omega_ei

            if (delabs .le. 1.e-8) then

      c       1 / (x + kd^2) / (x + kei^2)    [ from delta ]

              wei1 = deltaint

      c       1 / (y + kdbar^2) / (y + keibar^2) / (y + keebar^2)
      c       A_INT

              comei=eiteib-eiteeb-eitkdb+eetkdb
              comee=eiteeb-eeteeb-eitkdb+eetkdb
              comkd=eiteeb-eetkdb-eitkdb+kdtkdb
              aint=lgeib/comei-lgeeb/comee+lgkdb/comkd

              wei2=(one-kee2/kei2)/kdi2*aint

              wei = - dconst * (wei1 + wei2)
```

```
      else

c     *************************
c     We will need Arctan[kd2bar]/sqrt(4 delta1r - kd2bar^2)

       delsqr = 4.*delta1r - kdtkdb

       if (delsqr .lt. 0.) then
        del = dsqrt( -delsqr)
        arg = - kd2bar/del
        arc=dlog( (arg+1.)**2/(arg-1.)**2 ) / 4. / del

       elseif (delsqr .gt. 0.) then
        del = dsqrt( delsqr)
        arc = datan(kd2bar/del) / del
       else

        arc = -one / kd2bar

       endif

c     *************************

      delteeb=delta1r*kee2bar
      delteib=delta1r*kei2bar
      deltkdb=delta1r*kd2bar
      lgdel = dlog(delta1r)


c      y / (y^2 + kdbar^2 + delta1r) / (y + keibar^2)
c      B_INT_IE

       term1 = (2.*delta1r - eitkdb) * arc
       term2 = -kei2bar*dlog( kei2bar/ deltsqrt )
       comcei= delta1r + eiteib - eitkdb
       bintei= (term1 + term2) / comcei

       wei1 = bintei/ kde / kdi

c      y / (y^2 + kdbar^2 + delta1r) / (y + keebar^2) / (y +
keibar^2)
c      C_INT

       ccom1d=(delta1r+eeteeb-eetkdb)
       ccom1d=ccom1d*(delta1r+eiteib-eitkdb)
       ccomn= 2.d0*delteib+2.d0*delteeb
     >       -deltkdb-eetkdb*kei2bar

       term14= ( ccomn*arc+(-delta1r+eeteib)
     >         *lgdel/2. ) / ccom1d

       ccom2d=delteib+kei2bar*eiteib-delteeb
     >      +eiteib*(-kee2bar-kd2bar)
     >      +eitkdb*kee2bar
       term2= kei2bar*lgeib/ccom2d

       ccom3d=-delteib+kee2bar*eeteeb+delteeb
     >      +eeteeb*(-kei2bar-kd2bar)
```

```
     >       +eitkdb*kee2bar
             term3= kee2bar*lgeeb/ccom3d
             cint = term14 + term2 + term3

             wei2 = (one - kee2/kei2) * cint / kdi2

             wei = - dconst * (wei1 + wei2)


            endif

      c  **************
      c       Omega_ee

            weeconst=-forpiesq*beta*forpiesq*kee2/2.0

            if (delabs .le. 1.e-8) then

      c      1 / (x + kd^2) / (x + kee^2)

             wee1 = dlog ( kd2 / kee2 ) / (kee2 - kd2)

      c      1 / (y + kdbar^2) / (y + keibar^2) / (y + keebar^2) [ from
    wei ]
      c      A_INT

             wee2=(one - kei2/kee2)/kde2*aint

             wee = - weeconst * (wee1 + wee2)
      c ****
      c      The following statement is a compromise. Kei != kee which
    causes
      c       problems with the wee2 integral.  wee1 is negative; wee2
    is positive.
      c       As Tev decreases, wee2 becomes large enough to make wee
    positive.
      c       As Tev further decreases, this sign change + increasing
    absolute
      c       value of wee causes 'bigd' to change sign.  Since bigd is
    a
      c       determinant, this causes a "resonance" to occur in nu_ei.
      c       Not permitting wee to be > 0, say, by absolute-valuing wee
    doesn't work
      c       as well as just zeroing out the entire wee term
      c
      c       iweesw must equal 1 for this to occur

             if ((wee .gt. 0.) .and. (iweesw .eq. 1)) wee = 0.
      c ****

            else

      c      y / (y^2 + kdbar^2 + deltalr) / (y + keebar^2)
      c      B_INT_EE

             term1 = (2.*deltalr - eetkdb) * arc
             term2 = -kee2bar*dlog( kee2bar/ deltsqrt )
             comcee= deltalr + eeteeb - eetkdb
             bintee= (term1 + term2) / comcee
```

```
            wee1 = bintee/ kde / kdi

      c     y / (y^2 + kdbar^2 + delta1r) / (y + keebar^2) / (y +
keibar^2)
      c     C_INT


            wee2 = (one - kei2/kee2*betaei/betai) * cint / kde2

      c     1 / (y^2 + kdbar^2 + delta1r) / (y + keebar^2) / (y +
keibar^2)
      c     D_INT  [ similar to C_INT]

      c**         absorb (1-betaei/betai) = delta1r into result

            dcomng= delta1r+eiteib-eitkdb
            dcomnh= delta1r+eeteeb-eetkdb
            dcomngh= kei2bar-kee2bar

            comn1x=2.d0*(-delta1r+eiteeb)-eitkdb
            comn1=delta1r*(comn1x-eetkdb+kdtkdb)
            comn2=-delteib-delteeb+deltkdb
            term12= (comn1*arc+comn2*lgdel/2.d0)/dcomng/dcomnh

            term3= - delta1r*lgeib/dcomng/dcomngh

            term4= delta1r*lgeeb/dcomnh/dcomngh

            dint = term12 + term3 + term4

            wee3 = kei2bar * dint / kde2


      c      Now get extra terms with delta1r in front due to cD_ee
      c**         absorb (1-betaei/betai) = delta1r into all results


      c     y / (y^2 + kdbar^2 + delta1r) / (y + kdbar^2) / (y +
keebar^2)
      c     F_INT


            termn1=2.d0*delteeb+deltkdb-kee2bar*kdtkdb
            term1=termn1*arc

            term2=(-delta1r+eetkdb)*lgdel/2.d0

            term3=delteeb*lgeeb / (kee2bar-kd2bar)

            term4=kd2bar*lgkdb/(-kee2bar+kd2bar)

            fint = (term1+term2+term3)/dcomnh + term4

            wee4 = -fint / kde2

      c     1 / (y^2 + kdbar^2 + delta1r) / (y + kdbar^2) / (y +
keebar^2)
      c     G_INT_EE
```

```
            termn1= 2.d0*delta1r-eetkdb
            term1= -termn1*arc

            term2= -kee2bar*lgdel / 2.d0

            term3= delta1r*lgeeb/(-kee2bar+kd2bar)
            term4= lgkdb/(kee2bar-kd2bar)

            gintee= (term1+term2+term3)/dcomnh + term4


            wee5 = -kdi/kde/kde2 * gintee

      c     1 / (y^2 + kdbar^2 + delta1r) / (y + keibar^2) / (y +
kdbar^2)
      c     G_INT_IE

            termn1= 2.d0*delta1r-eitkdb
            term1= -termn1*arc

            term2= -kei2bar*lgdel / 2.d0

            term3= delta1r*lgeib/(-kei2bar+kd2bar)
            term4= lgkdb/(kei2bar-kd2bar)

            gintie= (term1+term2+term3)/dcomng + term4

            wee6 = kdi/kde/kde2*betaei/betai*kei2/kee2 * gintie

            wee = - weeconst * (wee1+wee2+wee3+wee4+wee5+wee6)
      c ****
            if ((wee .gt. 0.) .and. (iweesw .eq. 1)) wee = 0.
      c ****

          endif


      c     Now get the matrix elements.
      c     Set up constants:
      c     pi^(-3/2)*one/me^(-1/2) & dsqrts of beta's
      c      this excludes a factor of ni; put in later

            constm=one/(pi**1.5)/(dsqrt(me))
            sbeta=dsqrt(beta)
            sbetaei=dsqrt(betaei)


      c                 M_pp

            xnum1 = one/6.0/dsqrt(2.0d0)
            ppw0 = -constm*sbetaei*xnum1*delta

      c                 M_rr

            xnum1 = 3.0/28.0
            xnum2 = 433./280./6./dsqrt(2.0d0)
            rrw0 = -constm*z*(sbeta*xnum1*wee+sbetaei*xnum2/z*wei)
```

```
c                M_qq

      xnum1 = one/15.
      xnum2 = 13./60./dsqrt(2.0d0)
      qqw0 = -constm*z*(sbeta*xnum1*wee+sbetaei*xnum2/z*wei)

c                M_qr


      xnum1 = one/5./dsqrt(28.0d0)
      xnum2 = 23./20./dsqrt(2.0d0)/dsqrt(28.0d0)
      qrw0 = constm*z*(sbeta*xnum1*wee+sbetaei*xnum2/z*wei)

c                M_pq

      xnum1 = one/8./dsqrt(5.0d0)
      pqw0 = constm*sbetaei*xnum1*(delta+wei)

c                M_pr

      xnum1 = 5./16./dsqrt(35.0d0)
      prw0 = -constm*sbetaei*xnum1*(delta+wei)


c     Put together the combinations

c     Each Mx element is missing a factor of ni in front
c     Do same for frequency here (all to prevent overflows)

      woni = wo / ni
      wonisq = woni * woni

      bigd=  rrw0*qqw0 - qrw0*qrw0

      bigrr= rrw0*pqw0 - qrw0*prw0
      bigqq= qqw0*prw0 - qrw0*pqw0

c     thermal conductivity const excludes k_boltzmann
c     reported as purely static

      tcond=2.5*ne/(bigd*ni)/beta/me*rrw0

      cfppw0 = ppw0*ni
      cfppw0wp = cfppw0 / wpe / xmult

      Dsqr=(bigd-wonisq)**2+wonisq*(qqw0+rrw0)**2
      psi1=(bigd-wonisq)/Dsqr*(pqw0*bigrr+prw0*bigqq)
      psi2=wonisq/Dsqr*(qqw0+rrw0)*(pqw0**2+prw0**2)

      psi = (psi1 + psi2) * ni
      cfw0 = cfppw0 - psi
      omult = cfppw0 / cfw0
      cfw0wpe = cfw0 / wpe


c     Return BRD result if CR result is bad (large coupling) and
c      ibrdsw = 1
```

```
            if ((cfw0wpe .lt. ppxdhbr).and.(ibrdsw .eq. 1)) cfw0wpe =
    ppxdhbr
            cfw0 = cfw0wpe * wpe


            psii1=woni/Dsqr*(qqw0+rrw0)*(pqw0*bigrr+prw0*bigqq)
            psii2=-woni/Dsqr*(bigd-wonisq)*(qqw0+rrw0)**2
            psii = (psii1 + psii2) * ni
            psiiw = woni + psii

            econd = wpe2/4./pi*cfw0/(cfw0*cfw0+psiiw*psiiw)

            psiiwpe = psiiw/wpe
            econdwpe = cfw0wpe/4./pi/(cfw0wpe*cfw0wpe+psiiwpe*psiiwpe)

            rhow0 = 4.*pi/wpe2*cfw0*9.0e17

            return
            end




      c ******************************************************

            real*8 function expe1(x)
              implicit real*8 (a-h,o-z)

      c      returns the value of exp(x) * E1(x)
      c      (have to watch out for large x)

            if(x .gt. 50.0) then
             x2=x*x
             x4=x2*x2
             poly=1.0-1.0/x+2.0/x2-6.0/x2/x+24.0/x4
             expe1=poly/x
             return
            endif

            expe1=dexp(x)*e1local(x)
            return
            end
```

```
c ***************************************************
      real*8 function ellocal(x)
        implicit real*8 (a-h,o-z)

c
c     Returns exponential integral of the first kind.  Abramowitz
& Stegan p. 231.
c
      dimension a(4),b(4),c(6)

      data c /-.57721566, .99999193,-.24991055,
     >         .05519968,-.00976004, .00107857/
      data a / 8.5733287401,18.0590169730,
     >         8.6347608925,  .2677737343/
      data b / 9.5733223454,25.6329561486,
     >        21.0996530827, 3.9584969228/

      if(x.le. 0.0) then
      write(*,100)
  100 format(' argument of E1 < or = zero)')
      stop
      endif

      if(x.le. 1.0)then
      e=c(6)*x+c(5)
      do 10 i=4,1,-1
      e=e*x+c(i)
   10 continue
      ellocal=e - dlog(x)
      return
      endif

      enum=x+a(1)
      eden=x+b(1)
      do 20 i=2,4
      enum=enum*x+a(i)
      eden=eden*x+b(i)
   20 continue
      ellocal=enum/eden*dexp(-x)/x

      return

      end
```

Appendix B

```
      program rhotest

          implicit real*8(a-h,o-z)
          real*8 ni
          common tcond,xmult,omult,wowpe,gamei

          write(*,2)
        2 format(' Restricted to T_electron => T_ion',/
         >       ' If Zbar < 0, calc Zbar',/)
          z=-9.
          iweesw = 1
          ibrdsw = 1

       99 write(*,100)
      100 format(' Input n_ion (cm-3), kT_e (eV),kT_i, Zbar,',/,
         >       '     Atomic Weight, Wavelength (nm)')
          read(*,*) ni,tev,tiv,z,aw,xwavel
c
          if (ni .le. 0.0) stop

          write(*,101)
      101 format(' Input Low T switch, BRD switch')
          read(*,*) iweesw,ibrdsw

          if (z .le. 0.) then
           write(*,98)
       98  format(' Zbar, input Z_nuc and Temperature (T_e Default)')
           temp=tev
           read(*,*) znuc,temp
           z = zbar(ni,temp,znuc)
           write(*,501) z
      501  format(' Zbar = ',f7.3)
          endif

          call rho (ni,tev,tiv,z,aw,xwavel,wpe,
         >  cfw0,cfw0wpe,econd,econdwpe,rhow0,
         >  ppxdhbr,rhoxdhbr,cfppw0wp,iweesw,ibrdsw)

          write(*,699) wpe,z,xmult,omult,gamei
          write(*,700) tcond
          write(*,701) cfw0,cfw0wpe,econd,econdwpe,rhow0
          write(*,702) cfppw0wp,ppxdhbr,rhoxdhbr
      699 format(' wpe = ',e12.6,' Z = ',f6.3,' t_mult = ',f6.3,
         >       ' o_mult = ',f6.3,' Gamma_ei = ',f7.3)
      700 format(' thermal conductivity/k_B: ',e12.6)
      701 format(' nu_ei: ',e12.6,2x,e12.6,/,
         >       ' econd: ',e12.6,2x,e12.6,/,
         >       '   rho: ',e12.6,2x,e12.6)
      702 format(' M_pp only: ',8x,e12.6,/,
         >       ' BRD nu_ei: ',8x,e12.6,/,' BRD rho: ',10x,e12.6)

          go to 99
          end
```

```
c
**********************************************************************

      real*8 function zbar (deni,tev,znuc)
      implicit real*8 (a-h,o-z)

c       calculates average charge in a plasma given the
c       temp, dens, z, and atomic number of the plasma
c       ions.  taken from r. m. more, ucrl-84991 (5 march,
c       1981), p. 61   dens is the ion density.
c
      data av/6.022e23/
      data al,be/14.3139,0.6624/
      data a1,a2,a3,a4/0.003323,0.971832,9.26148e-5,3.10165/
      data b0,b1,b2/-1.763,1.43175,0.315463/
      data c1,c2/-0.366667,0.983333/
c
      to=tev/znuc**(4.0/3.0)
      tf=to/(1.0+to)
      r=deni/znuc/av
      aa=a1*to**a2+a3*to**a4
      bb=-exp(b0+b1*tf+b2*tf**7)
      cc=c1*tf+c2
      q1=aa*r**bb
      q=(r**cc+q1**cc)**(1.0/cc)
      x=al*q**be
      zbr=znuc*x/(1.0+x+sqrt(1.0+2.0*x))
      zbar=zbr
      return
      end
```

Appendix C

```
 Restricted to T_electron => T_ion
 If Zbar < 0, calc Zbar

 Input n_ion (cm-3), kT_e (eV),kT_i, Zbar,
     Atomic Weight, Wavelength (nm)
  6.e+22 ,  10.0 ,  10.0,  3.0  ,  27.0,  400.0
 Input Low T switch, BRD switch
  /
 wpe =  .239334e+17 Z =  3.000 t_mult =  2.529 o_mult =  2.662
Gamma_ei =   2.726
 nu_ei:  .735262e+16   .307212e+00
 econd:  .619514e+15   .258849e-01
   rho:  .145173e+03
 M_pp only:          .323319e+00
 BRD nu_ei:          .886300e-01
 BRD rho:            .165598e+02

 Input n_ion (cm-3), kT_e (eV),kT_i, Zbar,
     Atomic Weight, Wavelength (nm)
   6.e+22 ,  50.0 ,  10.0,  -9.0  ,  27.0,  1.0E20
 Input Low T switch, BRD switch
1,1
 Zbar, input Z_nuc and Temperature (T_e Default)
 13.0,50.0
 Zbar =   5.233
 wpe =  .316084e+17 Z =  5.233 t_mult =  2.771 o_mult =  2.779
Gamma_ei =    .951
 nu_ei:  .279881e+16   .885463e-01
 econd:  .284068e+17   .898710e+00
   rho:  .316826e+02
 M_pp only:          .887862e-01
 BRD nu_ei:          .339936e-01
 BRD rho:            .438901e+01

 Input n_ion (cm-3), kT_e (eV),kT_i, Zbar,
     Atomic Weight, Wavelength (nm)
   6.e+22 ,  50.0 ,  10.0,  3.0  ,  27.0,  1.0E20
 Input Low T switch, BRD switch
/
 wpe =  .239334e+17 Z =  3.000 t_mult =  2.529 o_mult =  2.560
Gamma_ei =    .545
 nu_ei:  .159570e+16   .666725e-01
 econd:  .285659e+17   .119356e+01
   rho:  .315061e+02
 M_pp only:          .674836e-01
 BRD nu_ei:          .358051e-01
 BRD rho:            .668989e+01

 Input n_ion (cm-3), kT_e (eV),kT_i, Zbar,
     Atomic Weight, Wavelength (nm)
  -9. /

  stop
```

REFERENCES

1. R. Cauble and W. Rozmus, Phys. Fluids **28**, 3387 (1985).

2. R. M. More, Z. Zinamon, K. H. Warren, R. Falcone, and M. Murnane, J. de Physique **49**, C7-43 (1988).

3. R. Cauble and W. Rozmus, J. Plas. Phys. **37**, 405 (1987).

4. R. Cauble and W. Rozmus, "Electrical conductivity in solid density plasmas produced by ultra-short laser pulses," to be submitted to Phys. Rev. E (1993).

5. D. B. Boercker, F. J. Rogers, and H. E. DeWitt, Phys. Rev. A **25**, 1623 (1982).

6. R. M. More, UCRL-84991, 61 (1981).

7. J. P Hansen and I. R. McDonald, Phys. Rev. A **23**, 2041 (1981).

DATE
FILMED
4/25/94
END