Development of a Driver Program to Evaluate Diffusion Coefficients

E.P. Marriott

September 2005

UWFDM-1341

FUSION TECHNOLOGY INSTITUTE

UNIVERSITY OF WISCONSIN

MADISON WISCONSIN

Development of a Driver Program to Evaluate Diffusion Coefficients

E.P. Marriott

Fusion Technology Institute
University of Wisconsin
1500 Engineering Drive
Madison, WI 53706

http://fti.neep.wisc.edu

September 2005

UWFDM-1341

Abstract

In the course of developing more complex physical models for nuclear high energy density physics programs, it is often necessary to take sections of codes and simplify them for specific tasks. This report details that goal. The code explained in this report takes the task of determining the diffusion and slowing down coefficients for a multi-group diffusion code and envelops it into one code. The end product is an output file that displays the diffusion and slowing down coefficients for a wide range of densities and temperatures. This task successfully simplified the determining of coefficients into a self-standing code that can be used for many applications. Ultimately, the determining of the electron and ion cross sections, electron and ion stopping powers, and values for τ_g, will prove to be useful for many tasks in the future.

Introduction

The code detailed in this report has several purposes. The first is that it was developed to assist Sandia National Laboratory (SNL) in the development of their Alegra radiation hydrodynamics code. Secondly, the code is used specifically to determine the diffusion and slowing down coefficients to use in the flux limited diffusion model for the alpha particle reaction product transport. This code fulfills the task of developing the thermonuclear burn physics for SNL.

The requirement from SNL is to provide the coefficients for the diffusion model. The FLD routines and matrix solver for the Alegra code will be provided by SNL. This report explains the code that was developed at the University of Wisconsin Fusion Technology Institute.

Multi-Group Diffusion

Methods have been developed to simulate the physics of the slowing down and spatial transport of energetic positive ions in a hot plasma [1] as well as the effects of scattering upon energetic ion energy loss in plasmas [2]. These methods are used in combination in this code.

The main objective of this code is to provide the coefficients for the multi-group form of the diffusion equation. This can be seen here:

$$\frac{\partial N_g}{\partial t} = \nabla \cdot D_g \nabla N_g - \frac{N_g}{\tau_g} + \frac{N_{g+1}}{\tau_{g+1}} \quad . \tag{1}$$

The coefficients that are to be supplied are the group diffusion coefficient D_g, τ_g, and τ_{g+1}. The code utilizes equations that will solve for those coefficients.

The following equation is used to solve for D_g:

$$D_g = \frac{v_g}{\frac{3}{\lambda_g} + \frac{1}{|\bar{\mu}|}\frac{1}{N_g}\left|\vec{\nabla} N_g\right|} \quad . \tag{2}$$

In this equation, the mean free path, λ_g, is equal to $2v_g t_D$. Also, the velocity, v_g, is equal to $\sqrt{\frac{E_g + E_{g+1}}{m}}$, where m is the energetic particle mass. In the code, $|\vec{\mu}|$ is set to a value of 1. When substituting and simplifying, equation (2) is reduced to:

$$D_g = \frac{\frac{E_g + E_{g+1}}{m}}{\frac{3}{2t_D} + \sqrt{\frac{E_g + E_{g+1}}{m}} \frac{|\vec{\nabla} N_g|}{N_g}} \tag{3}$$

where t_D is the time to 90° deflection, E_g and E_{g+1} are the group energies, m is the mass, and N_g is the number of particles in a group.

There is a simplification that can be made to these equations for the group diffusion coefficient, however. Since the code already solves for the electron and ion cross sections as well as the group velocities, the following two equations can be computed:

$$D_g^{ele} = \frac{v_g}{\Sigma_{ele}} \tag{4}$$

$$D_g^{ion} = \frac{v_g}{\Sigma_{ion}} \ . \tag{5}$$

Now the following combination of these can be used to solve for the total group diffusion coefficient, as shown here:

$$\frac{1}{D_g} = \frac{1}{D_g^{ele}} + \frac{1}{D_g^{ion}} \ . \tag{6}$$

To solve for the values of τ_g, the following equation is used:

$$\tau_g = t_E \int_{E_g}^{E_{g+1}} \frac{dE}{F(E)} = \frac{2}{3} t_E \ln\left(\frac{\gamma t_E + E_{g+1}^{3/2}}{\gamma t_E + E_g^{3/2}} \right) \ . \tag{7}$$

The values of τ_g are computed with this equation by using two pre-calculated variables call ele_eloss_rate and ion_ele_ratio. These two variables fill in for t_E and γt_E. This can be seen in the code in the appendix.

This code utilizes the stopping power of the electrons and ions from the work of Li and Petrasso. The following equations illustrate this:

$$\frac{dE^{t/f}}{dx} = -\frac{(Z_t e)^2}{v_t^2} \omega_{pf}^2 G\left(x^{t/f}\right) \ln \Lambda_b \tag{8}$$

$$G\left(x^{t/f}\right)=\mu\left(x^{t/f}\right)-\frac{m_f}{m_t}\left\{\frac{d\mu\left(x^{t/f}\right)}{dx^{t/f}}-\frac{1}{\ln\Lambda_b}\left[\mu\left(x^{t/f}\right)+\frac{d\mu\left(x^{t/f}\right)}{dx^{t/f}}\right]\right\} \tag{9}$$

where $dE^{t/f}/dx$ is the stopping power of a test particle in a field of background charges. The *t* and *f* sub- and superscripts represent test and field particles. Large-angle scattering is predominantly found in the $ln\ \Lambda_b$ terms. Also, Z_te is the test charge, v_t and v_f are the test and field particle velocities respectively, m_t and m_f are the test and field particle masses, ω_{pf} is the field plasma frequency, and $\mu(x^{t/f})$ is the Maxwell integral. The term $x^{t/f}$ is simply equal to v_t^2/v_f^2 .

The physics for the 90° deflection cross section comes from the work of Corman, et al. For the cross section, the code uses the following equation:

$$\sigma_{90}= factor\times gfact\times clog \tag{10}$$

where *factor* is calculated as:

$$factor = mratio\times z_t^2\times\omega/v_t^2/v_t^2/\left(m_t\times m_p\right)\quad . \tag{11}$$

In this equation for *factor*, *mratio* is defined below, z_t is the test particle charge, v_t is the test particle velocity, m_t is the test particle mass, m_p is the proton mass, and ω is given by:

$$\omega = 2\pi\times plasma\_freq \tag{12}$$

where *plasma_freq* is a function in the code. The term *clog* is calculated from the function coulomb_log_ion and *gfact* is calculated from either:

$$gfact=\left(\mu+\frac{d\mu}{dx}-\frac{1}{2}\frac{\mu}{x}\right) \tag{13}$$

or

$$gfact=\left(\mu+\frac{d\mu}{dx}-\frac{1}{2}\frac{\mu}{x}\right)\left(1+\frac{1}{2}\left(mratio-1\right)/\left(mratio+1\right)\right)\quad . \tag{14}$$

In these equations, the term *mratio* is either *mf/mt* or $5.45x10^{-4}/mt$ for ions or electrons respectively. From equations (13) and (14), μ is calculated by the function *gammp_s*. The $\frac{d\mu}{dx}$ term is either set to zero or $\frac{d\mu}{dx}=\frac{2}{\sqrt{\pi}\times e^{-x\sqrt{x}}}$ and *x* is equal to:

$$x=\frac{v_t^2}{v_{th}^2} \tag{15}$$

where v_t is the test particle velocity and v_{th} is the thermal velocity.

The choice between equations (13) and (14) is decided by assigning a value to cp_dc_type in module module_types. As a default, it is set to CP_DC_VERDON. This uses the classical calculation of *gfact*.

Other than the few equations shown, this paper does not delve into the detailed derivations of these equations, it merely emphasizes that these are the equations used in the code to produce the diffusion coefficients. For more information, please review references [1] and [2].

Code Operation

There are four sections of code that are used to determine the diffusion coefficients. There is a driver program, two modules and one subroutine. These can be seen in the appendix.

Each of the four sections is written in Fortran 90 code. The program is designed to be run in the Compaq Visual Fortran environment.

The driver program is where any changes can be made depending on which fusion reaction is desired as well as the number of groups in the model and the ranges for the temperatures and densities. In this code, the user can choose one of the three following fusion reactions:

1) $D + D = (50\%)\, T + p$
$\qquad\qquad (50\%)\, {}^{3}He + n$
2) $D + T = {}^{4}He + n$
3) $D + {}^{3}He = {}^{4}He + p$

This allows the user to choose deuterium-deuterium (DD) plasma, deuterium-tritium (DT) plasma, or deuterium-helium3 (DHe3) plasma.

The user can also change the density and temperature ranges by changing the value of the variables min_dens, max_dens, min_temp, and max_temp. The density values are in units of g/cm^3. The temperature values are in units of electron volts [eV]. If the density and/or temperature ranges are changed, then the incremental if statements for the changed variable must be adjusted to compensate for the changed range values. These if statements are located near the end of the code for the driver program.

The other variable that can be changed by the user is the number of groups for the diffusion model. This can be set by changing the variable num_cp_eg to a value other than 10.

Each of these changes allows the user to have a wide range of control over the desired output for the diffusion coefficients. The standard values for the variables are listed here:

```
num_cp_eg  = 10
min_dens   = 1.e+20
max_dens   = 1.e+24
min_temp   = 1.e+3
max_temp   = 1.e+5
```

Also, the program is set to run for DT plasma. To change to either the DD or DHe3 plasma, the user can simply comment out the DT lines and uncomment the lines for whichever plasma they wish to use.

Code Sections

The four sections of the code are each distinctive. The driver program is where the variables are set by the user. The subroutine cp_cross_section is called to compute the cross sections, stopping powers, and τ_g values. Both of these use the module module_types. This module is a list of constants and set values. The final module is module module_plasma. This module includes all of the physics that are used to determine the fusion coefficients. There are twelve functions in module module_plasma.

Code Output and Results

The output for this code is comprised of a single file. The file is called fort.10. This file holds all of the calculated coefficient values for each set combination of temperatures and densities.

The output will display at the top which of the three fusion reactions was used. Also, before each group of diffusion coefficients, there is a heading that shows what the temperature and density are for that set of coefficients. An example of this can be seen in Table 1.

Table 1. Sample Code Output

Coefficients for DT plasma
Densities = 0.10E+21, Temperatures = 0.10E+04

Group	Group Dimensions		Electron CS	Electron SP	Ion CS	Ion SP	Dg	Tau g
1	0.3600E+01	0.3250E+01	0.8557666E-05	0.1693627E+00	0.3548476E-04	0.4072178E-03	0.2953187E+04	0.1605818E-08
2	0.3250E+01	0.2900E+01	0.1014732E-04	0.1649161E+00	0.4387411E-04	0.4521094E-03	0.3344652E+04	0.1740214E-08
3	0.2900E+01	0.2550E+01	0.1227109E-04	0.1595669E+00	0.5565660E-04	0.5083336E-03	0.3841236E+04	0.1910277E-08
4	0.2550E+01	0.2200E+01	0.1521573E-04	0.1531349E+00	0.7295193E-04	0.5808379E-03	0.4490417E+04	0.2131766E-08
5	0.2200E+01	0.1850E+01	0.1950059E-04	0.1453777E+00	0.9984206E-04	0.6779543E-03	0.5372950E+04	0.2431358E-08
6	0.1850E+01	0.1500E+01	0.2615789E-04	0.1359529E+00	0.1450424E-03	0.8149033E-03	0.6637970E+04	0.2858094E-08
7	0.1500E+01	0.1150E+01	0.3752181E-04	0.1243437E+00	0.2300386E-03	0.1022801E-02	0.8594198E+04	0.3513150E-08
8	0.1150E+01	0.8000E+00	0.5999666E-04	0.1096878E+00	0.4205874E-03	0.1376872E-02	0.1199916E+05	0.4644763E-08
9	0.8000E+00	0.4500E+00	0.1180007E-03	0.9028861E-01	0.1008407E-02	0.2118321E-02	0.1932856E+05	0.7074240E-08
10	0.4500E+00	0.1000E+00	0.4081511E-03	0.6146556E-01	0.4924756E-02	0.4569311E-02	0.4574491E+05	0.1637636E-07

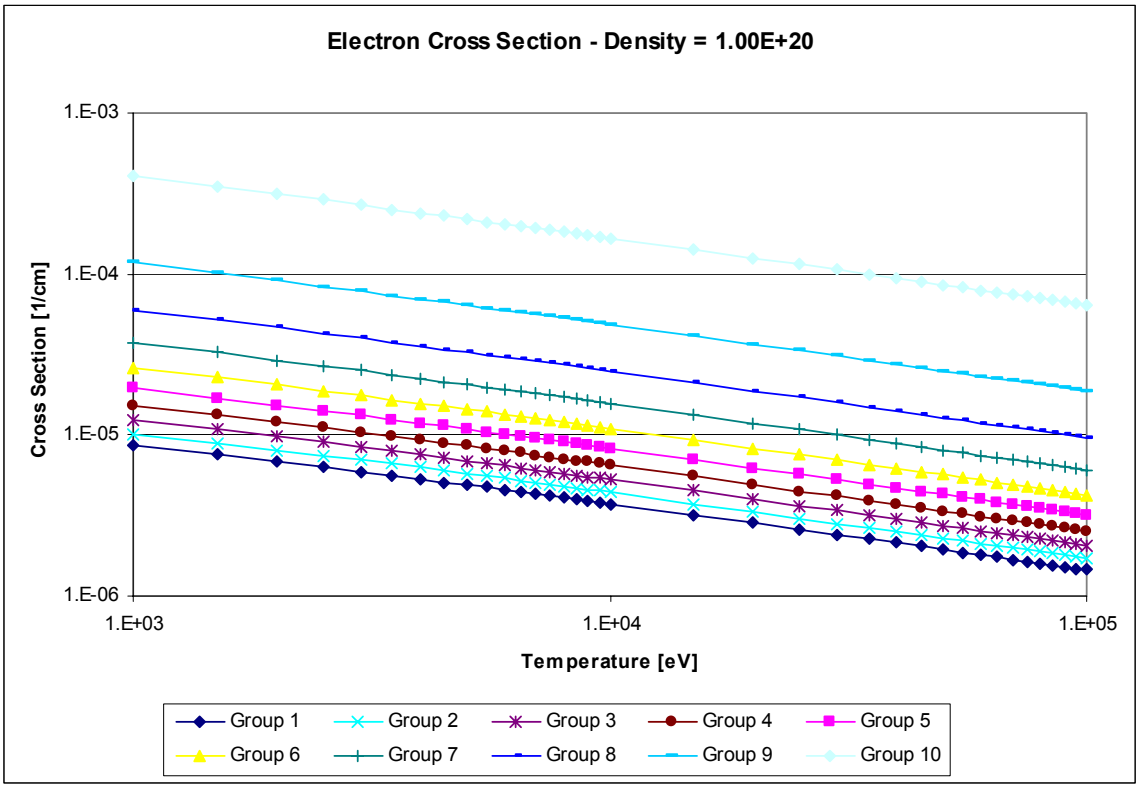
Densities = 0.10E+21, Temperatures = 0.15E+04

Group	Group Dimensions		Electron CS	Electron SP	Ion CS	Ion SP	Dg	Tau g
1	0.3600E+01	0.3250E+01	0.7542986E-05	0.1057689E+00	0.3592637E-04	0.4121429E-03	0.2670167E+04	0.2567488E-08
2	0.3250E+01	0.2900E+01	0.8920050E-05	0.1020897E+00	0.4442139E-04	0.4575950E-03	0.3014755E+04	0.2806258E-08
3	0.2900E+01	0.2550E+01	0.1075734E-04	0.9790362E-01	0.5635255E-04	0.5145239E-03	0.3451039E+04	0.3107000E-08
4	0.2550E+01	0.2200E+01	0.1330142E-04	0.9311590E-01	0.7386654E-04	0.5879404E-03	0.4020271E+04	0.3496987E-08
5	0.2200E+01	0.1850E+01	0.1699862E-04	0.8759840E-01	0.1010972E-03	0.6862843E-03	0.4792538E+04	0.4022262E-08
6	0.1850E+01	0.1500E+01	0.2273559E-04	0.8116851E-01	0.1468709E-03	0.8249740E-03	0.5897144E+04	0.4767153E-08
7	0.1500E+01	0.1150E+01	0.3251633E-04	0.7354764E-01	0.2329461E-03	0.1035532E-02	0.7601452E+04	0.5904571E-08
8	0.1150E+01	0.8000E+00	0.5183641E-04	0.6426542E-01	0.4259112E-03	0.1394173E-02	0.1056070E+05	0.7854175E-08
9	0.8000E+00	0.4500E+00	0.1016385E-03	0.5238405E-01	0.1021129E-02	0.2145310E-02	0.1691312E+05	0.1197018E-07
10	0.4500E+00	0.1000E+00	0.3504579E-03	0.3527596E-01	0.4986478E-02	0.4630651E-02	0.3974100E+05	0.2637501E-07

At the beginning of the output, the heading "Coefficients for DT plasma" clearly labels which of the three fusion reactions was used. The next line shows what the density and temperature are for that set of coefficients. The next line has the headings for the columns. Then the next ten lines are the values for the coefficients for each group. The first value in each of these ten rows is the group number. The next two values are the bounds for that group's dimensions. Then comes the electron cross section, the electron stopping power, the ion cross section, the ion stopping power, the group diffusion coefficient, and finally the value for τ_g. The next line once again has density and temperature values and thus begins a new set of coefficients.

After some analysis of the output file for the DT reaction, several graphs were produced. They illustrate the trends of the electron and ion cross sections, electron and ion stopping powers, group diffusion coefficients, and τ_g values for all ten groups in the multi-group model at a density of 1.00E+20/cm^3. The following figure is an example while all six figures can be seen in the appendix.

Figure 1. *Electron Cross Section at Density of 1.00E+20/cm^3*



Though these figures are rather simple in appearance, they do illustrate the physics of calculating these values for each of the ten groups in the model. These figures were produced using Microsoft Excel. Similar graphs could be produced for any of the densities from 1.00E+20/cm^3 to 1.00E+24/cm^3.

Acknowledgment

Support for this work was provided by Sandia National Laboratory under contract #330891.

References

[1] CORMAN, E.G., et al., Multi-group Diffusion of Energetic Charged Particles, Nuclear Fusion 15 (1975)

[2] LI, C.K., PETRASSO, R.D., Charged Particle Stopping Powers in Inertial Confinement Fusion Plasmas, Phys. Rev. 70, 3059 (1993)

Appendix

Code Section: *driver_cp.f90*

```
program driver_cp

 !------------------------------------------------------------------------------
 !  This program is used to calculate the diffusion coefficients for the three major
 !  contributing fusion equations:
 !
 !       1)    D + D   = (50%) T + p
 !                     = (50%) He3 + n
 !       2)    D + T   = He4 + n
 !       3)    D + He3 = He4 + p
 !
 !  This code was developed at the University of Wisconsin Fusion Technology
 !  Institute for a project in conjunction with Sandia National Laboratory.
 !------------------------------------------------------------------------------

    use module_types
    implicit none

    integer  :: num_cp_eg = 10
    integer  :: ngrpp
    integer  :: i, igr
    real(rk) :: ew
    real(rk) :: ion_mass, ion_avgz, ion_density, ele_density, ion_temp, ele_temp
    real(rk) :: ele_cs, ele_sp, ion_cs, ion_sp, dg, taug
    real(rk) :: min_dens = 1.e+20
    real(rk) :: max_dens = 1.e+24
    real(rk) :: min_temp = 1.e+3
    real(rk) :: max_temp = 1.e+5

    allocate( eabin(num_cp_eg+2) )
    ew       = (ealpmax - ealpmin) / real(num_cp_eg, kind=rk)
    ngrpp    = num_cp_eg + 1
    eabin(1) = ealpmax
    do i=2,ngrpp
       eabin(i) = eabin(i-1) - ew
    end do
    eabin(ngrpp+1) = eabin(ngrpp) * mc_ohalf

! *********************************************************
    ! Choose which plasma to use:

  ! FOR DD PLASMA, DD ion mass and DD ion charge
    ! ion_mass = 2.0_rk*prnmass
    ! ion_avgz = 1.0_rk
    ! write(10, '(a)') 'Coefficients for DD plasma'

  ! FOR DT PLASMA, DT ion mass and DT ion charge
     ion_mass = 2.5_rk*prnmass
     ion_avgz = 1.0_rk
     write(10, '(a)') 'Coefficients for DT plasma'

  ! FOR D-He3 PLASMA, D-He3 ion mass and D-He3 ion charge
    ! ion_mass = 2.5_rk*prnmass
    ! ion_avgz = 1.5_rk
    ! write(10, '(a)') 'Coefficients for D-He3 plasma'
! *********************************************************

    ! alpha particles collide with the background DT plasma
    ! ele_cs  :: electron 90 degree deflect cross section
    ! ion_cs  :: ion 90 degree deflect cross section
    ! ele_sp  :: electron stopping power
    ! ion_sp  :: ion stopping power
    ! dg      :: group diffusion coefficient
    ! taug    :: relaxation time defined in the diffusion equation
```

```
    ion_density = min_dens
    ele_density = min_dens
    ion_temp    = min_temp
    ele_temp    = min_temp

    do while ( ion_density .le. max_dens )
       do while ( ion_temp .le. max_temp )
          write(10, '(a, e8.2, a, e8.2)') 'Densities = ', ion_density, &
                    ', Temperatures = ', ion_temp
          write(10, '(a,a,a,a,a,a,a,a,a,a)') 'Group ', 'Group Dimensions         ', &
                    'Electron CS      ', 'Electron SP      ', 'Ion CS           ', &
                    'Ion SP           ', 'Dg               ', 'Tau g'
          do igr=1, num_cp_eg
             call cp_cross_section(igr, ion_mass, ion_avgz, ion_density, &
                    ele_density, ion_temp, ele_temp, ele_cs, ele_sp, ion_cs, &
                    ion_sp, dg, taug)

             write(10,'(i5, 2e11.4,6e17.7)') igr, eabin(igr), eabin(igr+1), &
                    ele_cs, ele_sp, ion_cs, ion_sp, dg, taug
          end do

          ! increment temperature values
          if ( ion_temp .lt. 1.e+4 ) then
             ion_temp = ion_temp + 0.5e+3
             ele_temp = ele_temp + 0.5e+3
          else
             ion_temp = ion_temp + 0.5e+4
             ele_temp = ele_temp + 0.5e+4
          end if

       end do

       ! increment density values
       if ( ion_density .lt. 1.e+21 ) then
          ion_density = ion_density + 0.5e+20
          ele_density = ele_density + 0.5e+20
       else if ( ion_density .lt. 1.e+22 ) then
          ion_density = ion_density + 0.5e+21
          ele_density = ele_density + 0.5e+21
       else if ( ion_density .lt. 1.e+23 ) then
          ion_density = ion_density + 0.5e+22
          ele_density = ele_density + 0.5e+22
       else
          ion_density = ion_density + 0.5e+23
          ele_density = ele_density + 0.5e+23
       end if

       ! reset ion temp and electron temp for while loop
       ion_temp = min_temp
       ele_temp = min_temp

    end do

    deallocate( eabin )

end program driver_cp
```

Code Section: *subroutine cp_cross_section.F90*

```
subroutine cp_cross_section(igr, ion_mass, ion_avgz, ion_density, ele_density, &
                      ion_temp, ele_temp, ele_cs, ele_sp, ion_cs, ion_sp, dg, taug)

    use module_types
    use module_plasma

    implicit none
    integer, intent(in)       :: igr
    real(rk), intent(in)      :: ion_mass, ion_avgz,ion_density, ele_density, &
                                    ion_temp, ele_temp
    real(rk), intent(out)     :: ele_cs, ele_sp, ion_cs, ion_sp, dg, taug

 !  Local Variables:
    real(real_kind) ::  eup, elow, term1, term2, vagroup, &
                        ewidthp, m_t, z_t, z_f, v_t, m_f, n_f, T_f, &
                        ebinavg, ele_eloss_rate, ion_ele_ratio, ewidth, &
                        dg_ele, dg_ion

    eup     = eabin(igr)
    elow    = eabin(igr+1)
    ewidth  = eup - elow
    ebinavg = (eup + elow) * mc_ohalf
    term1   = ebinavg
    term2   = (elow + eabin(igr+2)) * mc_ohalf
    ewidthp = term1 - term2
    term1   = 2._rk * ebinavg * ergs_per_mev
    term2   = term1 / alpmass
    vagroup = sqrt(term2)
    z_t = mc_two
    m_t = 4.0026_rk
    v_t = vagroup

    m_f = elemass/prnmass
    z_f = mc_one
    n_f = ele_density
    T_f = ele_temp

 !  Calculate ele_cs in units [1/cm]
    ele_cs = deflect90_cross_section(m_t,z_t,v_t,m_f,z_f,n_f,T_f,CONST_ELE)
    ele_sp = stopping_power(m_t,z_t,v_t,m_f,z_f,n_f,T_f,CONST_ELE)
    ele_eloss_rate = vagroup * ele_sp/ebinavg
    dg_ele = vagroup/(3/ele_cs)

    m_f = ion_mass/prnmass
    z_f = ion_avgz
    n_f = ion_density
    T_f = ion_temp

 !  Calculate ion_cs in units [1/cm]
    ion_cs = deflect90_cross_section(m_t,z_t,v_t,m_f,z_f,n_f,T_f,CONST_ION)
    ion_sp = stopping_power(m_t,z_t,v_t,m_f,z_f,n_f,T_f,CONST_ION)
    ion_ele_ratio = sqrt(ebinavg) * ion_sp * vagroup
    dg_ion = vagroup/(3/ion_cs)

    dg = 1/((1/dg_ele) + (1/dg_ion))

    taug = mc_two/mc_three/ele_eloss_rate * &
          log( (ele_eloss_rate*eup**1.5_rk+ion_ele_ratio) / &
               (ele_eloss_rate*elow**1.5_rk+ion_ele_ratio) )

end subroutine cp_cross_section
```

Code Section: *module module_types.f90*

```
module module_types
    integer, parameter :: int_kind = selected_int_kind(8)
    integer, parameter :: ik = selected_int_kind(8)
    integer, parameter :: real_kind = selected_real_kind(13,307)
    integer, parameter :: rk = selected_real_kind(13,307)

    real(real_kind), parameter :: mc_zero      = 0.0_real_kind
    real(real_kind), parameter :: mc_ohalf     = 0.5_real_kind
    real(real_kind), parameter :: mc_one       = 1.0_real_kind
    real(real_kind), parameter :: mc_pi = 3.1415926535897932_real_kind
    real(real_kind), parameter :: mc_three     = 3.0_real_kind
    real(real_kind), parameter :: mc_two       = 2.0_real_kind
    real(real_kind), parameter :: mc_othird    = mc_one / mc_three
    real(real_kind), parameter :: mc_twothirds = mc_two / mc_three

    real(real_kind), target :: prnmass = 1.6726E-24_real_kind
    real(real_kind), target :: echarge = 4.8032E-10_real_kind

    integer(ik), parameter :: CP_DC_VERDON = 0
    integer(ik), parameter :: CP_DC_YUAN   = 1
    integer(ik), parameter :: CP_DC_CORMAN = 2
    integer(ik), target    :: cp_dc_type   = CP_DC_VERDON

    real(real_kind), parameter :: ealpmax = 3.6_real_kind
    real(real_kind), parameter :: ealpmin = 0.1_real_kind

    real(rk), dimension(:), allocatable :: eabin
    real(rk), parameter :: ergs_per_mev = mc_one/6.242E+05_rk
    real(rk), parameter :: avognum      = 6.0221E+23_real_kind
    real(rk), parameter :: alpmass      = 4.0026_real_kind/avognum
    real(rk), parameter :: elemass      = 9.1094E-28_real_kind

end module module_types
```

Code Section: *module module_plasma.F90*

```
module module_plasma

    use module_types

    implicit none

        ! Setup local variables
    integer, parameter :: CONST_ELE=0
    integer, parameter :: CONST_ION=1

    contains
!   ******************************************************
!   t : temperature (eV)
!   a : mass in units of the proton mass
!   therm_velocity (cm/s)  * sqrt(2)
!
    real(rk) function therm_velocity(t,a,choice)

        real(real_kind), intent(in) :: t,a
        integer, intent(in) :: choice

        if(choice .eq. CONST_ELE) then
            therm_velocity = sqrt(2.0_rk)*4.19e+7_rk*sqrt(t)
        else if(choice .eq. CONST_ION) then
            therm_velocity = sqrt(2.0_rk)*9.79e+5_rk*sqrt(t/a)
        else
            stop "error type therm_velocity"
        end if

    end function therm_velocity

!   ******************************************************
!   z: charge, a : mass in units of the proton mass
!   d: density /cc
!   plasma_freq : 1/s ( not including 2*pi)
!
    real(rk) function plasma_freq(z,a,d,choice)

        real(real_kind), intent(in) :: z,a,d
        integer, intent(in) :: choice

        if(choice .eq. CONST_ELE) then
            plasma_freq = 8.98e+3_rk*sqrt(d)
        else if(choice .eq. CONST_ION) then
            plasma_freq = 210.0_rk*z*sqrt(d/a)
        else
            stop "error type plasma_freq"
        end if

    end function plasma_freq

!   ******************************************************
!   etemper, iontemper : electron ion temperature
!   edens : electron density
!   avgz : average charge state
!
    real(rk) function debye_length(etemper,edens,iontemper,avgz)

        real(real_kind), intent(in) :: etemper,edens
        real(real_kind), intent(in), optional :: iontemper,avgz
        real(real_kind) :: it,az,x

        it = etemper
        az = 0._rk
        if(present(iontemper)) it = iontemper
        if(present(avgz)) az = avgz

        x = it * etemper / (it + az * etemper)
```

```
        debye_length = 7.43e+02_rk * sqrt(x/edens)

    end function debye_length

!   ******************************************************
!   ele_density : cgs unit (1/cm^3)
!   ele_temper  : unit (eV)
!
    real(rk) function coulomb_log_ele(ele_density,ele_temper)

        implicit none

        real(real_kind), intent(in) :: ele_temper
        real(real_kind), intent(in) :: ele_density
        real(real_kind) :: xf12(15), xeta(15)

        data xf12/.115_rk,.184_rk,.291_rk,.45_rk,.678_rk,.99_rk,1.4_rk,1.9_rk, &
                2.5_rk,3.2_rk,3.977_rk,4.84_rk,5.77_rk,6.77_rk,7.838_rk/
        data xeta/-2._rk,-1.5_rk,-1._rk,-0.5_rk,0._rk,0.5_rk,1._rk,1.5_rk, &
                2._rk,2.5_rk,3.0_rk,3.5_rk,4.0_rk,4.5_rk,5._rk/

        real(real_kind) :: debye,factor,f12,lamdas2,tmadg,eta

        integer :: index

        tmadg = ele_temper * 11605.0_rk
        debye = 7.43e+02_rk * sqrt(ele_temper / ele_density)
        f12 = ele_density/(6.8e+21_rk * ele_temper * sqrt(ele_temper))

        if(f12 .lt. 0.115_rk) then
            eta = mc_zero
        else if(f12 .lt. 7.838_rk) then
            index = 2
            do while (f12 .gt. xf12(index))
               index = index + 1
            end do
            eta = mc_ohalf*(f12-xf12(index-1)) / &
               (xf12(index)-xf12(index-1)) + xeta(index-1)
        else
            eta = (1.5_rk*f12)**mc_twothirds
        end if

        if(eta .lt. mc_zero) eta=mc_zero

        lamdas2 = 1.766e+16_rk * tmadg * (debye/100.0_rk)**2
        factor  = 0.37_rk + 0.44_rk * eta *eta

        coulomb_log_ele = mc_ohalf * log(mc_one+lamdas2*factor)-mc_ohalf
        coulomb_log_ele = max(coulomb_log_ele, mc_one)

    end function coulomb_log_ele

!   ******************************************************
!   input variables : self-described
!   unit: particle_mass (g/cm^3)
!   velocity: cm/s
!   density : /cm^3
!   temper : eV
!
    real(rk) function coulomb_log_ion(particle_charge,particle_mass, &
        particle_velocity,bgele_density,bgele_temper,bgion_mass,bgion_temper, &
        bgion_avgzn)

        implicit none

        real(real_kind), intent(in) :: particle_charge,particle_mass,particle_velocity, &
                      bgele_density,bgele_temper,bgion_mass,bgion_temper,bgion_avgzn
        real(real_kind) :: hbar,debye,redmass,relvel,r0,rmax,drr,thetam,thetaq,&
                      particle_energy,bgion_density

        data hbar/1.054e-27_rk/
```

```
        debye = debye_length(bgele_temper,bgele_density,bgion_temper,bgion_avgzn)
        redmass = particle_mass*bgion_mass/(particle_mass+bgion_mass)
        particle_energy = 0.5_rk*particle_mass*particle_velocity**2
        relvel = sqrt(2.0_rk*particle_energy/redmass)
        bgion_density = bgele_density/bgion_avgzn
        r0      = (4.19_rk*bgele_density/bgion_avgzn)**(-mc_othird)
        rmax    = max(r0,debye)
        drr     = rmax*redmass*relvel
        thetaq  = hbar/drr
        thetam  = 46.14e-20_rk * particle_charge*bgion_avgzn/(drr*relvel)

        if(thetam .lt. thetaq) thetam=thetaq

        coulomb_log_ion = mc_ohalf*log(mc_one+4./thetam**2) - mc_ohalf

    end function coulomb_log_ion

!   ******************************************************
!   mt : test particle mass (unit: proton mass)
!   zt : test particle charge
!   vt : test particle velocity ( cm/s)
!   zf : field particle charge
!   mf : field particle mass (unit: proton mass)
!   nf : field particle number density ( #/cm^3)
!   tf : field plasma temperature (eV)
!
!   omega : 2*pi*frequency
!   x : vt**2/ vf**2
!   vth : thermal velocity
!
!   the calculation of incomplete gamma should use tabulated value
!   in order to make the computing fast
!
    real(rk) function stopping_power(mt,zt,vt,mf,zf,nf,tf,choice)

        implicit none

        real(real_kind), intent(in) :: mt,zt,vt,mf,zf,nf,tf
        integer, intent(in) :: choice

        real(real_kind) :: omega,omega2,zt2,vt2,factor,clog,x,vth,&
            mu,dmudx,gfact,mratio
        real(real_kind) :: mev2erg

        if(choice .eq. CONST_ION) then
            clog = coulomb_log_ion(zt,mt*prnmass,vt,nf,tf,mf*prnmass,tf,zf)
            mratio = mf/mt
        else if(choice .eq. CONST_ELE) then
            clog = coulomb_log_ele(nf*zf,tf)
            mratio = 5.45e-4_rk/mt
        end if

        omega = 2._rk * mc_pi*plasma_freq(zf,mf,nf,choice)
        omega2 = omega*omega

        zt2 = zt*zt*echarge*echarge
        vt2 = vt*vt

        factor = zt2*omega2/vt2

        vth = therm_velocity(tf,mf,choice)
        x = vt2/vth/vth

        ! set mu dmudx in order to avoid the float underflow
        if( x .gt. 300.0_rk) then
            mu = 1._rk
            dmudx = 0._rk
        else
            mu = gammp_s(1.5_rk,x)
            dmudx = 2._rk/sqrt(mc_pi)*exp(-x)*sqrt(x)
```

```
        end if

        gfact = mu-mratio*(dmudx-(mu+dmudx)/clog)

        mev2erg = 1.6022e-6_rk
        stopping_power = factor*gfact*clog / mev2erg

    end function stopping_power

!   ******************************************************
!   mt : test particle mass (unit: proton mass)
!   zt : test particle charge
!   vt : test particle velocity ( cm/s)
!   zf : field particle charge
!   mf : field particle mass (unit: proton mass)
!   nf : field particle number density ( #/cm^3)
!   tf : field plasma temperature (eV)
!
!   omega : 2*pi*frequency
!   x : vt**2/ vf**2
!   vth : thermal velocity
!
    real(rk) function deflect90_cross_section(mt,zt,vt,mf,zf,nf,tf,choice)

        implicit none

        real(real_kind), intent(in) :: mt,zt,vt,mf,zf,nf,tf
        integer, intent(in) :: choice

        real(real_kind) :: omega,omega2,zt2,vt2,factor,clog,x,vth,&
            mu,dmudx,gfact,mratio

        if(choice .eq. CONST_ION) then
            clog = coulomb_log_ion(zt,mt*prnmass,vt,nf,tf,mf*prnmass,tf,zf)
            mratio = mf/mt
        else if(choice .eq. CONST_ELE) then
            clog = coulomb_log_ele(nf*zf,tf)
            mratio = 5.45e-4_rk/mt
        end if

        omega = 2._rk*mc_pi*plasma_freq(zf,mf,nf,choice)
        omega2 = omega*omega

        zt2 = zt*zt*echarge*echarge
        vt2 = vt*vt

        factor = mratio*zt2*omega2/vt2/vt2/(mt*prnmass)

        vth = therm_velocity(tf,mf,choice)
        x = vt2/vth/vth

        mu = gammp_s(1.5_rk,x)

        ! set mu dmudx in order to avoid the float underflow
        if (x .gt. 300.0_rk) then
            dmudx = mc_zero
        else
            dmudx = 2._rk/sqrt(mc_pi)*exp(-x)*sqrt(x)
        end if

        select case (cp_dc_type)
        case (CP_DC_VERDON)
            gfact = (mu+dmudx-0.5_rk*mu/x)
        case (CP_DC_YUAN, CP_DC_CORMAN)
            gfact = (mu+dmudx-0.5_rk*mu/x)*(1._rk+0.5_rk*(mratio-1._rk)/(mratio+1._rk)/clog)
        end select

        deflect90_cross_section = factor*gfact*clog

    end function deflect90_cross_section
```

```
!   ******************************************************
!   From Numerical Recipes
!
    real(rk) function erf_s(x)

        real(rk), intent(in) :: x
        erf_s = gammp_s(0.5_rk, x*x)
        if(x .lt. 0.0_rk) erf_s = -erf_s

    end function erf_s

    real(rk) function gammp_s(a,x)

        real(rk), intent(in) ::a, x

        if(x .lt. a+1.) then
            gammp_s = gser_s(a,x)
        else
            gammp_s = 1.0_rk - gcf_s(a,x)
        end if

    end function gammp_s

    real(rk) function gser_s(a,x)

        integer :: n
        integer, parameter :: itmax = 100
        real(rk), intent(in) :: a,x
        real(rk), parameter :: eps = 3.0e-7_rk
        real(rk) :: ap,summ,del

        if( x .eq. 0.0_rk) then
            gser_s = 0.0_rk
        return
        end if

        ap = a
        summ = 1./a
        del = summ
        do n=1,itmax
            ap = ap + 1.
            del = del*x/ap
            summ = summ+del
            if(abs(del) .lt. abs(summ)*eps) exit
        end do

        if(n .gt. itmax) then
            stop "error in erf"
        end if
        gser_s = summ*exp(-x+a*log(x)-gammln_s(a))

    end function gser_s

    real(rk) function gcf_s(a,x)

        integer :: i
        integer, parameter :: itmax = 100
        real(rk), intent(in) :: a, x
        real(rk), parameter :: eps=1.e-8_rk, fpmin=1.e-100_rk
        real(rk) :: b,c,d,h,del,an

        if(x .eq. 0._rk) then
            gcf_s=1.0_rk
        return
        end if

        b = x+1.-a
        c=1.0_rk/fpmin
        d=1._rk/b
        h = d
        do i=1,itmax
```

```
            an = -i*(i-a)
            b = b+2._rk
            d=an*d+b
            if(abs(d) .lt. fpmin) d=fpmin
            c=b+an/c
            if(abs(c) .lt. fpmin) c=fpmin
            d=1.0_rk/d
            del=d*c
            h=h*del
            if(abs(del-1._rk) .le. eps) exit
        end do

        if(i .gt. itmax) then
            stop "itmax too small in gcf_s"
        end if

        b = -x+a*log(x)-gammln_s(a)
        if (b .lt. -300.0_rk) then
            gcf_s = mc_zero
        else
            gcf_s = exp(b) * h
        end if

    end function gcf_s

    real(rk) function gammln_s(xx)

        integer :: i
        real(rk), intent(in) :: xx
        real(rk) :: x,tmp,ser,y
        real(rk) :: stp = 2.5066282746310005_rk
        real(rk), dimension(6) :: coef = (/ 76.18009172947146_rk, &
                                    -86.50532032941677_rk, &
                                    24.01409824083091_rk, &
                                    -1.231739572450155_rk, &
                                    1.208650973866179e-3_rk, &
                                    -5.395239384953e-6_rk /)

        if(xx .le. 0) then
            stop "fatal error in gammln_s"
        end if
        x = xx
        tmp = x+5.5_rk
        tmp = (x+0.5_rk)*log(tmp)-tmp
        ser = 1.000000000190015_rk
        y=x
        do i=1,size(coef)
            y = y+1.
            ser=ser+coef(i)/y
        end do
        gammln_s = tmp+log(stp*ser/x)

    end function gammln_s

end module module_plasma
```

Figure A1. *Electron Cross Section at Density = 1.00E+20/cm^3*

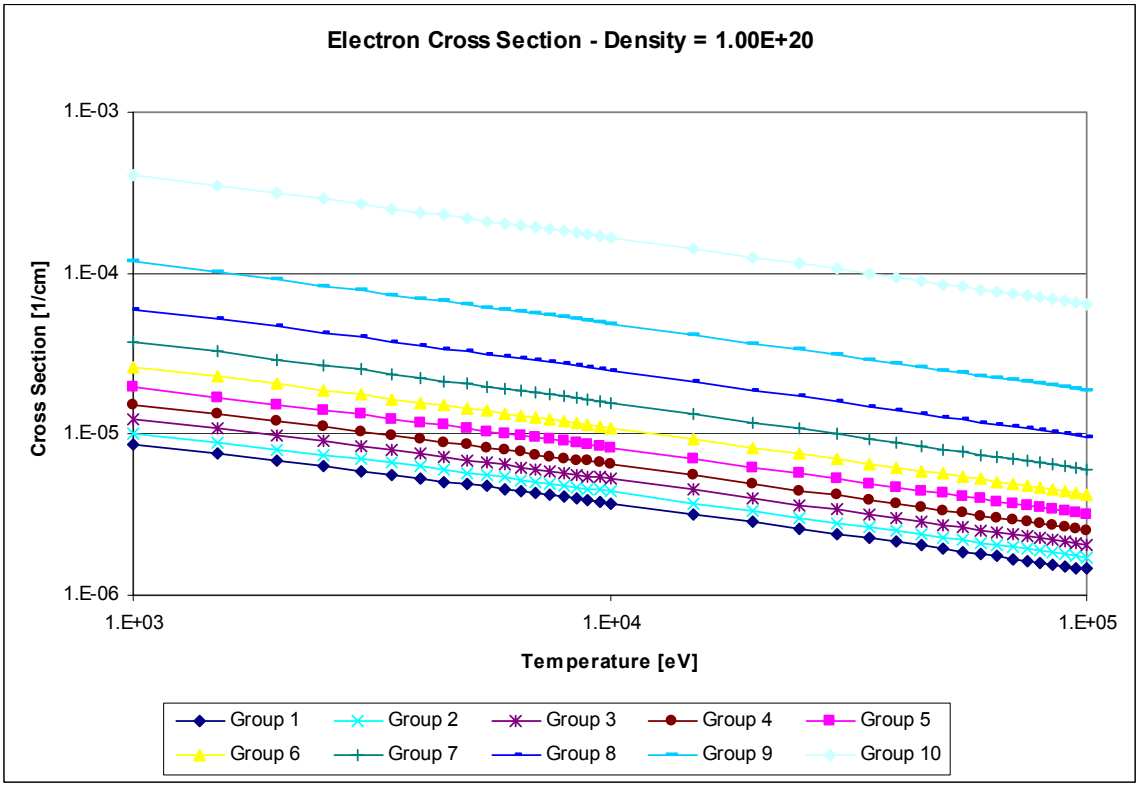


Figure A2. *Ion Cross Section at Density = 1.00E+20/cm^3*

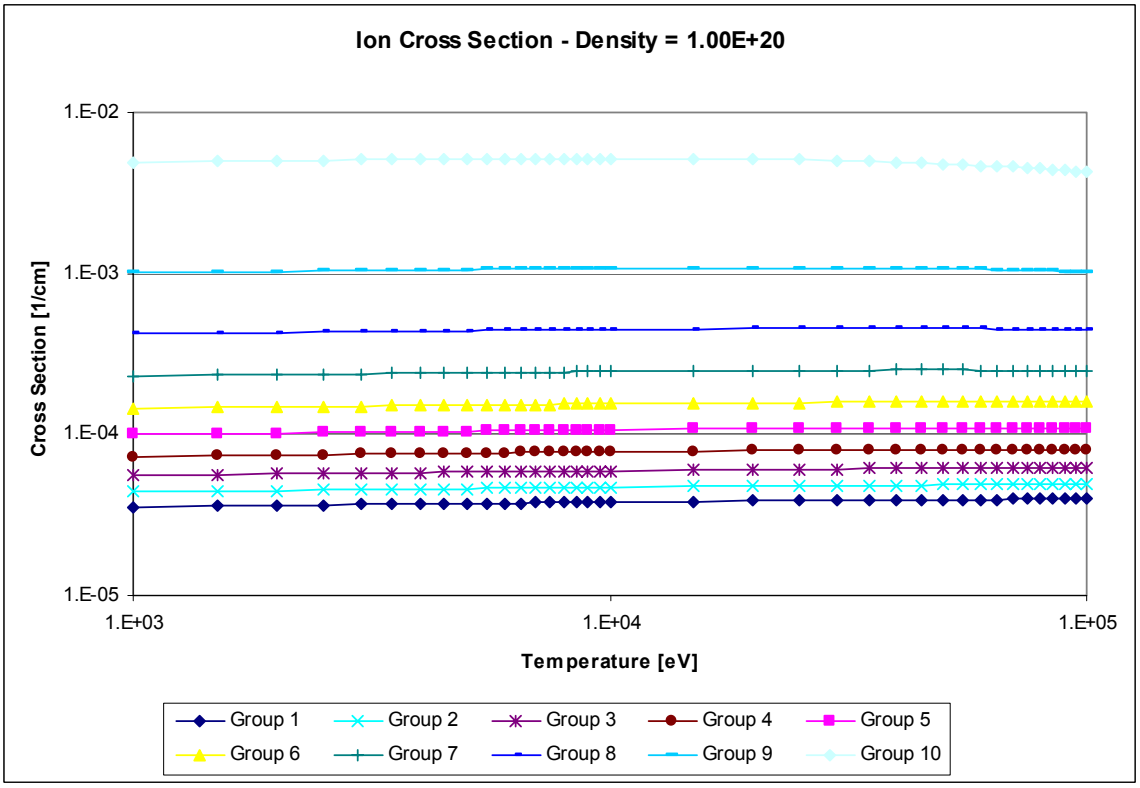


Figure A3. *Electron Stopping Power at Density = 1.00E+20/cm^3*

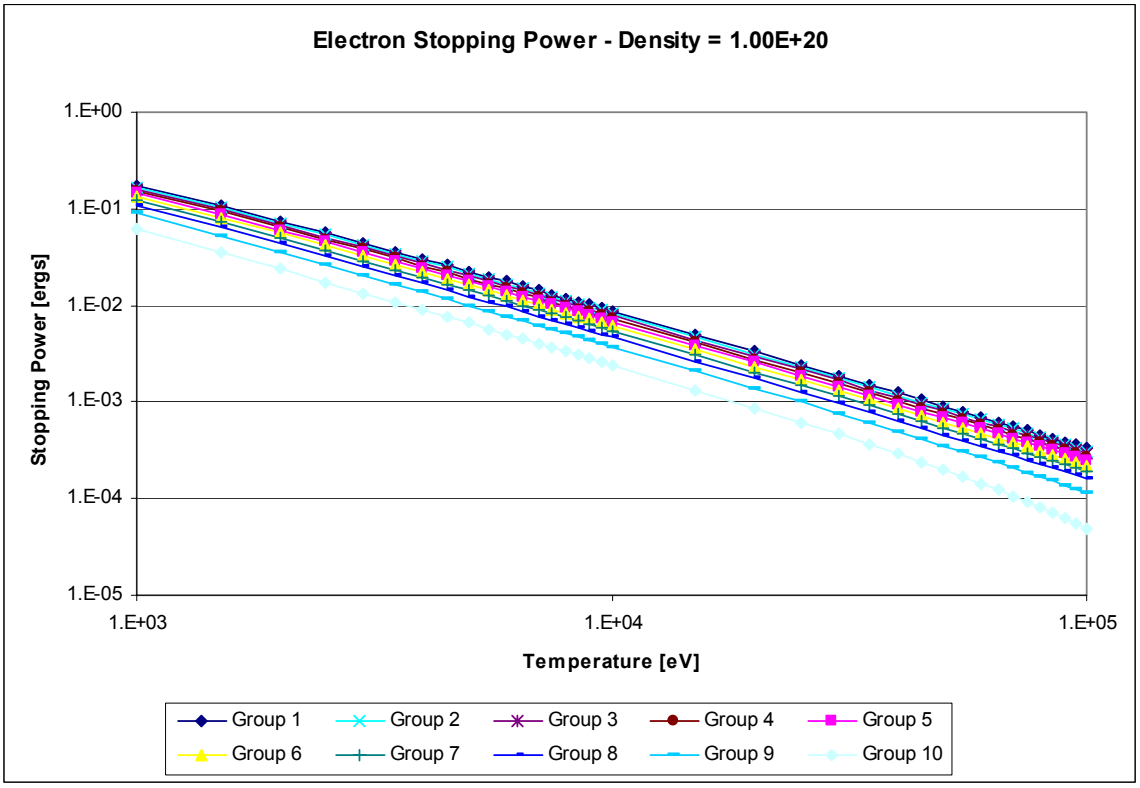


Figure A4. *Ion Stopping Power at Density = 1.00E+20/cm^3*

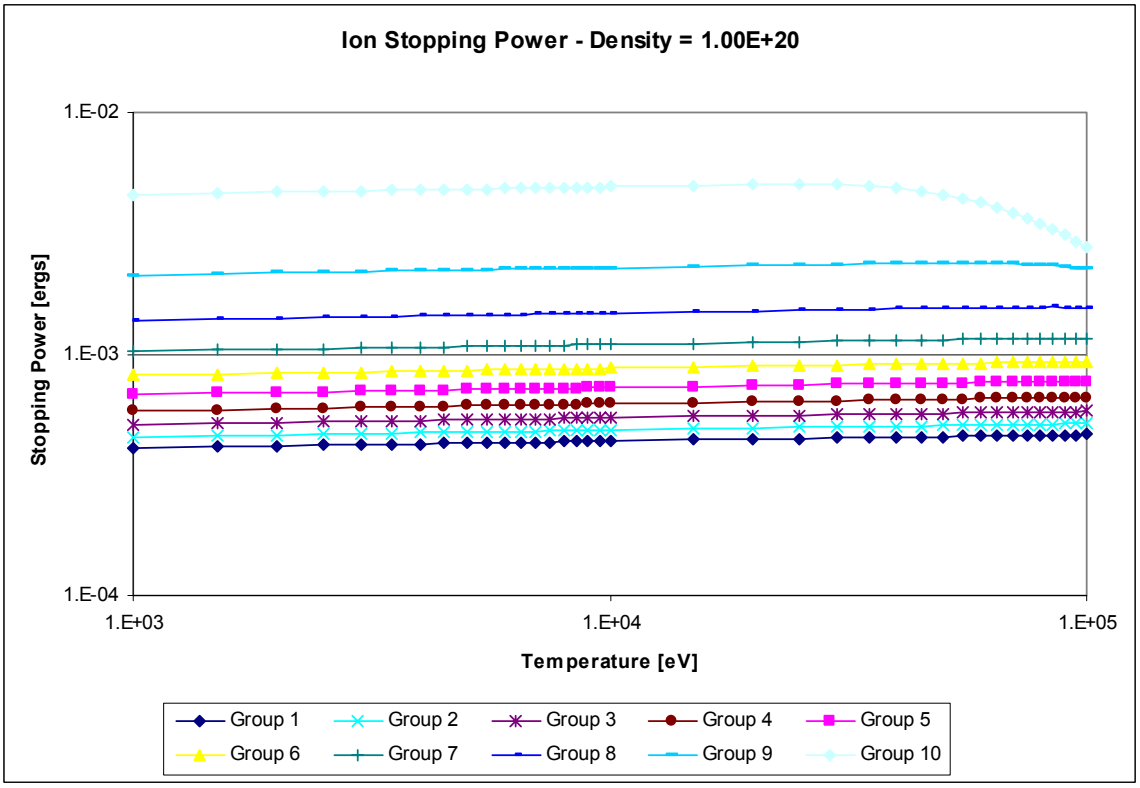


Figure A5. *Tau-g at Density = 1.00E+20/cm^3*

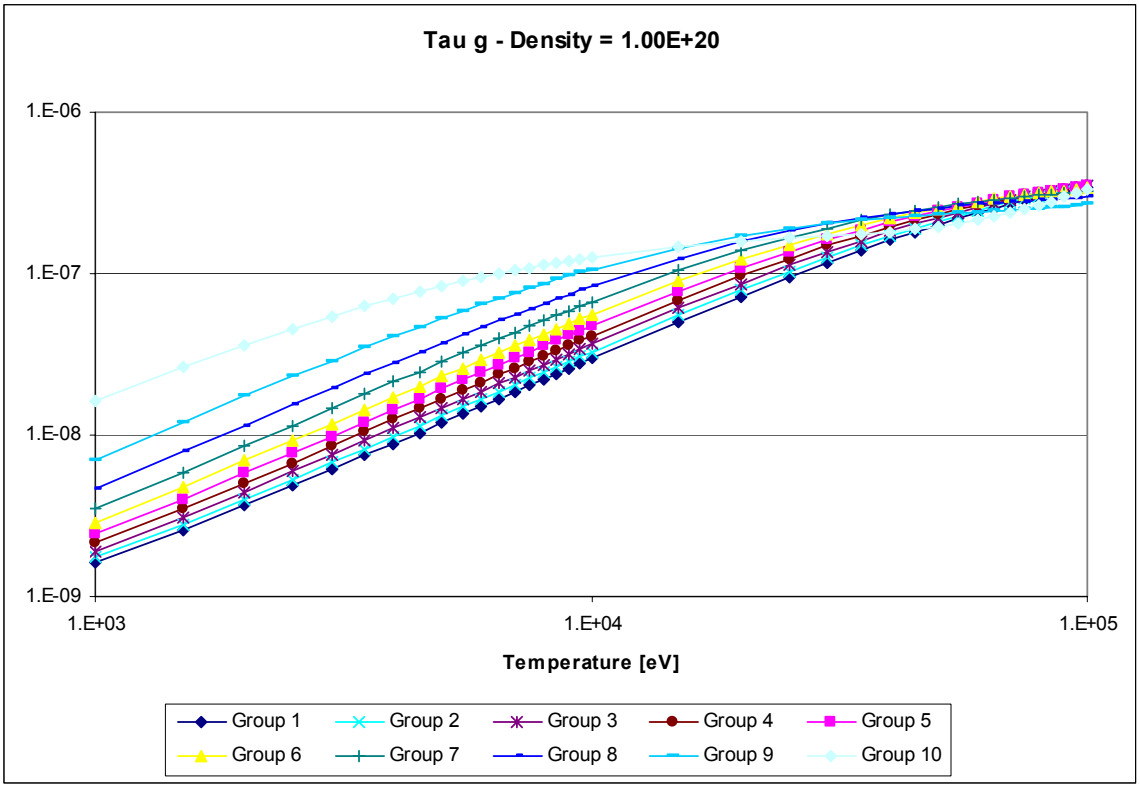


Figure A6. *Group One Diffusion Coefficients over a Range of Densities*