Development of a Driver Program to Evaluate Diffusion Coefficients

E.P. Marriott

September 2005

UWFDM-1341
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 $\tau_g$, 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:

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

(1)

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

The following equation is used to solve for $D_g$:

$$D_g = \frac{3}{\lambda_g} + \frac{1}{\mu} \frac{1}{N_g} \left| \frac{\partial N_g}{\partial x} \right| .$$

(2)
In this equation, the mean free path, $\lambda_g$, is equal to $2\nu_g t_D$. 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, $|\bar{\mu}|$ is set to a value of 1. When substituting and simplifying, equation (2) is reduced to:

$$D_g = \frac{E_g + E_{g+1}}{m} \frac{3}{2t_D} \left[ \sqrt{\frac{E_g + E_{g+1}}{m} \left( \frac{\nabla N_g}{N_g} \right)} \right]$$

(3)

where $t_D$ is the time to 90$^\circ$ 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:

$$D^{ele}_{g} = \frac{v_g}{\Sigma_{ele}}$$

(4)

$$D^{ion}_{g} = \frac{v_g}{\Sigma_{ion}}$$

(5)

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

$$\frac{1}{D_g} = \frac{1}{D^{ele}_{g}} + \frac{1}{D^{ion}_{g}}.$$  

(6)

To solve for the values of $\tau_g$, the following equation is used:

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

(7)

The values of $\tau_g$ 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 $\gamma t_E$. 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:

$$\frac{dE^{\text{eff}}}{dx} = - \left( \frac{Z_e e^2}{v_i^2} \omega_{pf} G(x^{\text{eff}}) \right) \ln \Lambda_b$$

(8)
\[
G(x^{t/f}) = \mu(x^{t/f}) \frac{m_f}{m_t} \left\{ \frac{d\mu(x^{t/f})}{dx^{t/f}} - \frac{1}{\ln \Lambda_b} \left[ \mu(x^{t/f}) + \frac{d\mu(x^{t/f})}{dx^{t/f}} \right] \right\}
\]

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, \( \omega_{pf} \) 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:

\[
\sigma_{90} = \text{factor} \times gfact \times clog
\]

where \( \text{factor} \) is calculated as:

\[
\text{factor} = \text{mratio} \times z_t^2 \times \omega / v_t^2 / v_f^2 (m_t \times m_p)
\]

In this equation for \( \text{factor} \), \( \text{mratio} \) 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 \( \omega \) is given by:

\[
\omega = 2\pi \times \text{plasma\_freq}
\]

where \( \text{plasma\_freq} \) is a function in the code. The term \( \text{clog} \) is calculated from the function \( \text{coulomb\_log\_ion} \) and \( gfact \) is calculated from either:

\[
gfact = \left( \mu + \frac{d\mu}{dx} - \frac{1}{2} \frac{\mu}{x} \right)
\]

or

\[
gfact = \left( \mu + \frac{d\mu}{dx} - \frac{1}{2} \frac{\mu}{x} \right) \left[ 1 + \frac{1}{2} \left( \text{mratio} - 1 \right) / \left( \text{mratio} + 1 \right) \right]
\]

In these equations, the term \( \text{mratio} \) is either \( m_f/m_t \) or \( 5.45 \times 10^{-4}/m_t \) for ions or electrons respectively. From equations (13) and (14), \( \mu \) is calculated by the function \( \text{gammp\_s} \). The \( \frac{d\mu}{dx} \) term is either set to zero or

\[
\frac{d\mu}{dx} = \frac{2}{\sqrt{\pi} \times e^{-x^2}}
\]

and \( x \) is equal to:

\[
x = \frac{v_t^2}{v_{th}^2}
\]

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 \( \text{cp\_dc\_type} \) in module \text{module\_types}. As a default, it is set to \text{CP\_DC\_VERDON}. This uses the classical calculation of \( g\text{fact} \).

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) \( D + D = (50\%) T + p \)
\( (50\%) \text{^3He} + n \)

2) \( D + T = \text{^4He} + n \)

3) \( D + \text{^3He} = \text{^4He} + p \)

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 \text{min\_dens}, \text{max\_dens}, \text{min\_temp}, and \text{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 \text{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:

\[
\begin{align*}
\text{num\_cp\_eg} &= 10 \\
\text{min\_dens} &= 1.0e+20 \\
\text{max\_dens} &= 1.0e+24 \\
\text{min\_temp} &= 1.0e+3 \\
\text{max\_temp} &= 1.0e+5
\end{align*}
\]

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 $\tau_g$ 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>
<thead>
<tr>
<th>Group</th>
<th>Group Dimensions</th>
<th>Electron CS</th>
<th>Electron SP</th>
<th>Ion CS</th>
<th>Ion SP</th>
<th>Dg</th>
<th>Tau g</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.3600E+00 0.3250E+01</td>
<td>0.8557666E-05</td>
<td>0.162327E+00</td>
<td>0.3548476E-04</td>
<td>0.407217E-03</td>
<td>0.2953187E+04</td>
<td>0.1605818E-08</td>
</tr>
<tr>
<td>2</td>
<td>0.3250E+01 0.2900E+01</td>
<td>0.1649161E+00</td>
<td>0.4378411E-04</td>
<td>0.4521094E-03</td>
<td>0.334652E+04</td>
<td>0.1740214E-08</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.2900E+01 0.2550E+01</td>
<td>0.5565600E-04</td>
<td>0.5083336E-03</td>
<td>0.3641236E-04</td>
<td>0.1910277E+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.2550E+01 0.2200E+01</td>
<td>0.7295193E-04</td>
<td>0.5808378E-03</td>
<td>0.4490417E-04</td>
<td>0.2131766E+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.2200E+01 0.1850E+01</td>
<td>0.9984206E-04</td>
<td>0.7795438E-03</td>
<td>0.5372950E-04</td>
<td>0.2431358E+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.1850E+01 0.1500E+01</td>
<td>0.1450424E-03</td>
<td>0.667797E+04</td>
<td>0.2858049E-04</td>
<td>0.2806426E+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.1500E+01 0.1150E+01</td>
<td>0.2300366E-03</td>
<td>0.8594199E+04</td>
<td>0.3513150E-04</td>
<td>0.2131766E+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.1150E+01 0.8000E+00</td>
<td>0.4205874E+00</td>
<td>0.1376872E+00</td>
<td>0.1932856E+00</td>
<td>0.4644763E+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0.8000E+00 0.4500E+00</td>
<td>0.407217E-03</td>
<td>0.1022801E-02</td>
<td>0.2118321E-02</td>
<td>0.7074240E+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.4500E+00 0.1000E+00</td>
<td>0.4904766E-02</td>
<td>0.1376872E+00</td>
<td>0.1932856E+00</td>
<td>0.4644763E+00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 $\tau_g$. 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 $\tau_g$ 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/cm³

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/cm³ to 1.00E+24/cm³.

Acknowledgment

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

References


Appendix

**Code Section: driver_cp.f90**

```fortran
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
```

7
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)') '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

! 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
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
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
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
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(etype,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

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

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,dr,thetal,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)
rmx = max(r0,debye)
drr = rmx*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*omega2
zt2 = zt*zt*echarge*echarge
vt2 = vt*vt
factor = zt2*omega2/vt2
vth = therm_velocity(tf,mf,choice)
x = vt2/vth/vth
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 = gammmp_s(0.5_rk, x*x)
    if(x .lt. 0.0_rk) erf_s = -erf_s
end function erf_s

real(rk) function gammmp_s(a,x)

    real(rk), intent(in) ::a, x
    if(x .lt. a+1.) then
        gammmp_s = gser_s(a,x)
    else
        gammmp_s = 1.0_rk - gcf_s(a,x)
    end if
end function gammmp_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. 0.0_rk) then
        gcf_s=1.0_rk
        return
    end if
    b = x+1.-a
    c=1.0_rk/fpmin
    d=1.0_rk/b
    h = d
    do i=1,itmax
        an = a+1
        b = b+1
        d = d+1
        h = h+1
        gser_s = gser_s + gser_s
        gcf_s = gcf_s + gcf_s
        if(abs(an) .lt. abs(b)*eps) exit
    end do
end function gcf_s
\[ an = -i*(1-a) \]
\[ b = b+2._{rk} \]
\[ d=an*d+b \]
\[ if(abs(d) .lt. fpmin) d=fpmin \]
\[ c=b+an/c \]
\[ if(abs(c) .lt. fpmin) c=fpmin \]
\[ d=1.0_{rk}/d \]
\[ del=d*c \]
\[ h=h*del \]
\[ if(abs(del-1._rk) .le. eps) exit \]
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/cm³

Figure A2. Ion Cross Section at Density = 1.00E+20/cm³
Figure A3. Electron Stopping Power at Density = 1.00E+20/cm³

Figure A4. Ion Stopping Power at Density = 1.00E+20/cm³
Figure A5. Tau-g at Density = 1.00E+20/cm$^3$

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