Milestone 1 Code

Daniel Herman

February 2018

1 Code

```
module rec_mod
  use healpix_types
  use params
  use time_mod
  use ode_solver
  use spline_1D_mod
  implicit none

  integer(i4b), private                :: i                    ! make i
     a global variable
  integer(i4b)                         :: n                    ! Number
     of grid points
  real(dp), allocatable, dimension(:)  :: x_rec,a_rec,z_rec  ! Grid
  real(dp), allocatable, dimension(:)  :: tau, tau2, tau22   !
    Splined tau and second derivatives
  real(dp), allocatable, dimension(:)  :: tau4
  real(dp), allocatable, dimension(:)  :: logtau, logtau2, logtau22
  real(dp), allocatable, dimension(:)  :: n_e, n_e2          !
    Electron density, n_e
  real(dp), allocatable, dimension(:)  :: logn_e, logn_e2    !
    Splined log of electron density
  real(dp), allocatable, dimension(:)  :: g, g2, g22, g4     !
    Splined visibility function
  real(dp), allocatable, dimension(:)  :: X_e, H_rec         !
    Fractional electron density, n_e / n_H
  real(dp), allocatable                :: dx                 !

  real(dp)                   :: eps,hmin,yp1,ypn,h1          ! ODE/
    spline stuff we need

contains

  subroutine initialize_rec_mod
    implicit none

    real(dp)      :: saha_limit, y, T_b, n_b, dydx, xmin, xmax, dx,
    f, n_e0, X_e0, xstart, xstop
    real(dp)      :: junk, const, phi2, alpha2, beta, beta2,
    lambda_alpha, lambda2s1s, n1s, C_r
    real(dp)      :: z_start_rec, z_end_rec, z_0
    logical(lgt) :: use_saha
```

```fortran


    saha_limit  = 0.99d0              ! Switch from Saha to Peebles when X_e < 0.99
    xstart      = log(1.d-10)         ! Start grids at a = 10^-10
    xstop       = 0.d0                ! Stop  grids at a = 1
    n           = 1000                ! Number of grid points between xstart and xstop

    ! ODE int variables
    eps         = 1.d-10
    hmin        = 0.d0
    !Spline variables
    yp1         = 1.d30
    ypn         = 1.d30

    z_start_rec = 1630.4d0           ! Redshift at start of recombination
    z_end_rec   = 614.2d0            ! Redshift at the end of recombination
    z_0         = 0.d0               ! Current redshift (duh)


    ! Allocate necessary arrays

    allocate(X_e(n))
    allocate(tau(n),tau2(n),tau22(n),tau4(n))
    allocate(logtau(n),logtau2(n),logtau22(n))
    allocate(n_e(n),n_e2(n))
    allocate(g(n),g2(n),g22(n),g4(n))
    allocate(x_rec(n),a_rec(n),z_rec(n),H_rec(n))
    allocate(logn_e(n),logn_e2(n))



    ! Task: Fill in x,a,z (rec) grid -----------------------------------------------------------


    x_rec(1)    = xstart
    x_rec(n)    = xstop
    dx          = (xstop - xstart)/(n-1)

    do i = 1, n
       x_rec(i) = (i-1)*dx + xstart
    end do

    do i = 1,n
       H_rec(i) = get_H(x_rec(i))
    end do

    a_rec       = exp(x_rec)
    z_rec       = 1.d0/a_rec - 1.d0

    h1          = abs(1.d-2*(x_rec(1) - x_rec(2)))   ! Define the step length for the odeint

```

```
!--------------------------------------------------------------------------------

! Task: Compute X_e and n_e at all grid times
use_saha = .true.
do i = 1, n

   T_b       = T_0/a_rec(i)
   n_b       = Omega_b*rho_c/(m_H*a_rec(i)**3)
   const     = ((m_e*k_b*T_b/(2.d0*PI*hbar**2))**(1.5))*exp((-epsilon_0)/(T_b*k_b))
   junk      = (1/n_b)*const

   if (use_saha) then
      ! Use the Saha equation
      X_e(i) = (-junk+sqrt(junk*junk+4.d0*junk))/2.d0
      if (X_e(i) < saha_limit) use_saha = .false.
   else
      ! Use the Peebles equation
      X_e(i) = X_e(i-1)
      call odeint(X_e(i:i),x_rec(i-1),x_rec(i),eps,h1,hmin,dX_edx,bsstep, output)
   end if

   ! Calculate the electron density
   n_e(i) = X_e(i)*n_b

end do

! open(26, file='X_e.dat')
!    do i=1,n
!       write(26,'(2(E17.8))') x_rec(i),X_e(i)
!    end do
! close(26)

! Task: Compute splined (log of) electron density function
logn_e = log(n_e)
call spline(x_rec, logn_e, yp1, ypn, logn_e2)

! Task: Compute optical depth at all grid points
tau(n) = 0.d0
do i=n-1,1,-1
   tau(i) = tau(i+1)
   call odeint(tau(i:i),x_rec(i+1),x_rec(i),eps,h1,hmin,dtaudx,bsstep,output)
end do

! Task: Compute splined (log of) optical depth
! Task: Compute splined second derivative of (log of) optical depth

do i=1,n
   tau2(i) = -n_e(i)*sigma_T*c/H_rec(i)
end do

call spline(x_rec,tau,yp1,ypn,tau22)
```

```
    call spline(x_rec,tau22,yp1,ypn,tau4)

    ! open(25, file='tau.dat')
    !    do i=1,n
    !       write(25,'(4(E17.8))') x_rec(i),tau(i),tau2(i),tau22(i)
    !    end do
    ! close(25)

    do i=1,n
       g(i) = -tau2(i)*exp(-tau(i))
    end do

    ! Task: Compute splined visibility function
    call spline(x_rec,g,yp1,ypn,g22)
    ! Task: Compute splined second derivative of visibility function
    call spline(x_rec,g22,yp1,ypn,g4)

    do i=1,n
       g2(i) = -tau22(i)*exp(-tau(i))+tau2(i)*tau2(i)*exp(-tau(i))
    end do

    open(20, file='vis2.dat')
      do i=1,n
         write(20,'(4(E17.8))') x_rec(i),g(i),g2(i),g22(i)
      end do
    close(20)


  end subroutine initialize_rec_mod

!-------------------------------------------------------------------------------------------------------------
!-------------------------------------------------------------------------------------------------------------
!------------------------- Subroutines for odeint -------------------------------------------------------

  subroutine dX_edx(x, X_e, dydx)
    use healpix_types
    implicit none
    real(dp),                 intent(in)  :: x
    real(dp), dimension(:), intent(in)  :: X_e
    real(dp), dimension(:), intent(out) :: dydx
    real(dp)                            :: T_b, n_b, phi2,alpha2, beta,beta2,n1s,lambda_alpha,C_r
    real(dp)                            :: Xe,lambda2s1s
    real(dp)                            :: a
    real(dp)                            :: H

    lambda2s1s    = 8.227d0                          ! [s-1]
    Xe            = X_e(1)
    a             = exp(x)
    H             = get_H(x)
    T_b           = T_0/a
```

```fortran
    n_b          = Omega_b*rho_c/(m_H*a**3)
    phi2         = 0.448*LOG(epsilon_0/(k_b*T_b))
    alpha2       = 64.d0*PI/sqrt(27*PI)*(alpha/m_e)**2*sqrt(epsilon_0/(k_b*T_b))*phi2*hbar*hbar/c
    beta         = alpha2*((m_e*k_b*T_b/(2.d0*PI*hbar*hbar))**(1.5))*exp((-epsilon_0)/(T_b*k_b)
    n1s          = (1.d0 - Xe)*n_b
    lambda_alpha = H*(3.d0*epsilon_0)**3/(n1s*(8.d0*PI)**2)/(c*hbar)**3

    if (T_b <= 169.d0) then
       beta2     = 0.d0
    else
       beta2     = beta*exp(3.d0*epsilon_0/(4.d0*k_b*T_b))
    end if

    C_r          = (lambda2s1s + lambda_alpha)/(lambda2s1s + lambda_alpha + beta2)
    dydx         = C_r/H*(beta*(1.d0-Xe) - n_b*alpha2*Xe**2)
  end subroutine dX_edx

  subroutine dtaudx(x,tau,dydx)
    use healpix_types
    implicit none
    real(dp),                intent(in)  :: x
    real(dp), dimension(:), intent(in)  :: tau
    real(dp), dimension(:), intent(out) :: dydx
    real(dp)                             :: n_e
    real(dp)                             :: H
    n_e  = get_n_e(x)
   H    = get_H(x)
    dydx = -n_e*sigma_T/H*c
  end subroutine dtaudx

!--------------------------------------------------------------------------------------------------------------------------------
!--------------------------------------------------------------------------------------------------------------------------------
!------------ Functions for future work ----------------------------------------------------------

  function get_n_e(x_in)
    implicit none
    real(dp), intent(in) :: x_in
    real(dp)             :: get_n_e
    get_n_e    = splint(x_rec, logn_e, logn_e2, x_in)
    get_n_e    = exp(get_n_e)
  end function get_n_e


  function get_tau(x_in)
    implicit none
    real(dp), intent(in) :: x_in
    real(dp)             :: get_tau
    get_tau    = splint(x_rec, tau, tau22, x_in)
```

```
    ! get_tau  = exp(get_tau)
  end function get_tau


  function get_dtau(x_in)
    implicit none
    real(dp), intent(in) :: x_in
    real(dp)             :: get_dtau
    real(dp)             :: n_e,a,H_p
    H_p        = get_H_p(x_in)
    a          = exp(x_in)
    n_e        = get_n_e(x_in)
    get_dtau   = -n_e*sigma_T*a*c/H_p
  end function get_dtau


  function get_ddtau(x_in)
    implicit none
    real(dp), intent(in) :: x_in
    real(dp)             :: get_ddtau
    get_ddtau  = splint(x_rec,tau22,tau4,x_in)
  end function get_ddtau


  function get_g(x_in)
    implicit none
    real(dp), intent(in) :: x_in
    real(dp)             :: get_g
    real(dp)             :: dtau, tau
    dtau       = get_dtau(x_in)
    tau        = get_tau(x_in)
    get_g      = -dtau*exp(-tau)
  end function get_g


  function get_dg(x_in)
    implicit none
    real(dp), intent(in) :: x_in
    real(dp)             :: get_dg
    get_dg     = splint_deriv(x_rec,g,g22,x_in)
  end function get_dg


  function get_ddg(x_in)
    implicit none
    real(dp), intent(in) :: x_in
    real(dp)             :: get_ddg
    get_ddg    = splint(x_rec,g22,g4,x_in)
  end function get_ddg


end module rec_mod
```