Project 1 code

erikalev

February 2018

1 Code

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

  real(dp), allocatable, dimension(:) :: dtau              ! First derivative of tau: tau'
  real(dp), allocatable, dimension(:) :: dg                ! First derivative of g: g'
  real(dp)                            :: yp1, ypn, eps, hmin


  integer(i4b),                        private :: n, n_inter             ! Number of grid points
  real(dp), allocatable, dimension(:), private :: x_rec, x_inter             ! Grid
  real(dp), allocatable, dimension(:), private :: tau, tau2, tau22, log_tau, d2_tau  ! Splined tau and second derivatives
  real(dp), allocatable, dimension(:), private :: n_e, n_e2 ! Splined (log of) electron density, n_e
  real(dp), allocatable, dimension(:), private :: g, g2, g22, dg_dx        ! Splined visibility function

contains

  subroutine initialize_rec_mod
    implicit none

    real(dp), allocatable, dimension(:) :: X_e, a_rec ! Fractional electron density, n_e / n_H
    integer(i4b) :: i, j, k
    real(dp)     :: saha_limit, y, T_b, n_b, dx, hmin, eps, f, n_e0, X_e0, xstart, xstop, step
    real(dp)     :: C_r, constant

    logical(lgt) :: use_saha, test_interpolate

```

```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 xstopo
    n_inter    = 50000
    ! Spline variables
    yp1 = 1.d30
    ypn = 1.d30

    ! Integration variables
    eps = 1.d-10
    hmin = 0.d0

    ! Set as .true. to try the interpolated x-values
    test_interpolate = .false.


    ! Allocating arrays
    allocate(x_rec(n))
    allocate(X_e(n))
    allocate(tau(n))
    allocate(tau2(n))
    allocate(tau22(n))
    allocate(d2_tau(n))
    allocate(log_tau(n))
    allocate(n_e(n))
    allocate(n_e2(n))
    allocate(g(n))
    allocate(g2(n))
    allocate(g22(n))
    allocate(dg_dx(n))
    allocate(x_inter(n_inter))

    !--------------------------------------------------------------
    ! x_(rec) grid
    !--------------------------------------------------------------
    dx = (xstop-xstart)/(n-1)
    x_rec(1) = xstart
    do i=1, (n-1)
       x_rec(i+1) = x_rec(i) + dx
    end do

    !--------------------------------------------------------------
    ! X_e and n_e at all grid times
    !--------------------------------------------------------------
    use_saha = .true.
    do i = 1, n
       n_b = Omega_b0*rho_c0/(m_H*exp(3.d0*x_rec(i)))

       if (use_saha) then
          ! Saha equation
          T_b = T_0/exp(x_rec(i))
          constant = (m_e*k_b*T_b/(2.d0*pi*hbar**2))**1.5d0*exp(-epsilon_0/(k_b*T_b))/n_b     ! Constant infront of x^2 in quadratic formula
```

```fortran

        ! Alternatice quadratic formula to assure numerical presicion
        X_e(i) = 2.d0/(sqrt(1.d0 + 4.d0/constant) + 1.d0)

        if (X_e(i) < saha_limit) then
          use_saha = .false.
        end if
     else
        ! Peeble's equation
        X_e(i) = X_e(i-1)                              ! Using the last equated value of Xe
        call odeint(X_e(i:i), x_rec(i-1), x_rec(i), eps, dx, hmin, dXe_dx, bsstep, output) ! Updating Xe through ODEINT
     end if
     n_e(i) = X_e(i)*n_b                               ! Computing n_e
  end do
  n_e = log(n_e)

  !----------------------------------------------------------------
  ! Splined (log of) electron density function
  !----------------------------------------------------------------
  call spline(x_rec,n_e,yp1,ypn,n_e2)

  !----------------------------------------------------------------
  ! Splined optical depth at all grid points
  !----------------------------------------------------------------
  tau(n) = 0.d0
  log_tau(n) = -18.7d0
  do i = n-1, 1, -1
      tau(i) = tau(i+1)
      call odeint(tau(i:i), x_rec(i+1) ,x_rec(i), eps ,dx, hmin, dtau_dx, bsstep, output)
      log_tau(i) =log(tau(i))
  end do

  !----------------------------------------------------------------
  ! Splined (log of) optical depth
  !----------------------------------------------------------------
  call spline(x_rec, log_tau, yp1, ypn, tau2)


  !----------------------------------------------------------------
  ! Splined second derivative of (log of) optical depth
  !----------------------------------------------------------------
  call spline(x_rec, tau2, yp1, ypn, tau22)


  !----------------------------------------------------------------
  ! Splined vivisility function at all grid points
  !----------------------------------------------------------------
  do i=1,n
    g(i) = -get_dtau(x_rec(i))*exp(-tau(i))
  end do

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

```fortran
    ! Splined visibility function and second derivative
    !------------------------------------------------------------
    call spline(x_rec, g, yp1, ypn, g2)
    call spline(x_rec,g2,yp1,ypn,g22)

    !------------------------------------------------------------
    ! Printing x_rec and X_e to file
    !------------------------------------------------------------
    open(50, file = "X_e.dat")
    do i=1,n
       write(50,'(2(E17.8E3))') x_rec(i), X_e(i)
    end do
    close(50)
    if (test_interpolate) then
        ! x-range set manuallt
        xstart = -7.5
        xstop = 6.0

        x_inter(1) = xstart
        dx = (xstop-xstart)/(n_inter-1)
        do i=1, n_inter-1
            x_inter(i+1) = x_inter(i) + dx
        end do

        ! write to file
        open(51, file = "data.dat")
        do i=1,n
           write(51,'(7(E17.8E3))') x_inter(i), get_tau(x_inter(i)), get_dtau(x_inter(i)), get_ddtau(x_inter(i)), &
           get_g(x_inter(i)), get_dg(x_inter(i)), get_ddg(x_inter(i))
        end do
        close(51)

    else
        ! write to file
        open(51, file = "data.dat")
        do i=1,n
           write(51,'(7(E17.8E3))') x_rec(i), get_tau(x_rec(i)), get_dtau(x_rec(i)), get_ddtau(x_rec(i)), get_g(x_rec(i)), &
           get_dg(x_rec(i)), get_ddg(x_rec(i))
        end do
        close(51)

     end if
  end subroutine initialize_rec_mod

  ! ---------------------- Saha equation for integration ----------------------
  subroutine dXe_dx(x, X_e, deriv)
    use healpix_types
    implicit none
    real(dp),                 intent(in)  :: x
    real(dp), dimension(:), intent(in)  :: X_e
    real(dp), dimension(:), intent(out) :: deriv
    real(dp) :: lambda_21, lambda_a, Cr, beta, alpha2, n1s, beta2, phi2, H, n_b, T_b
```

```fortran
    T_b = T_0/exp(x)
    n_b = Omega_b0*rho_c0/(m_H*exp(3.d0*x))
    H = get_H(x)
    n1s = n_b*(1.d0 - X_e(1))
    phi2 = 0.448d0*log(epsilon_0/(k_b*T_b))
    lambda_a = H*(3.d0*epsilon_0)**3.d0/((8.d0*pi)*(8.d0*pi)*n1s*(c*hbar*c*hbar*c*hbar))   ! [s-1]

    alpha2 = 64.d0*pi / sqrt(27.d0*pi) * (alpha/m_e)**2.d0 * sqrt(epsilon_0/(k_b*T_b)) * phi2 * hbar*hbar/c
    beta = alpha2 * (m_e*k_b*T_b / (2.d0*pi*hbar*hbar))**1.5d0 * exp(-epsilon_0/(k_b*T_b))

    ! combined the betas to avoid infinities
    beta2 = alpha2 * (m_e*k_b*T_b / (2.d0*pi*hbar*hbar))**1.5d0 * exp((-epsilon_0)/(4.d0*k_b*T_b))

    Cr = (lambda_21 + lambda_a)/(lambda_21 + lambda_a + beta2)
    deriv = Cr/H*(beta*(1.d0 - X_e) - n_b*alpha2*X_e*X_e)
  end subroutine dXe_dx


  ! Routine for computing n_e at arbitrary x, using precomputed information
  function get_n_e(x)
    implicit none

    real(dp), intent(in) :: x
    real(dp)             :: get_n_e
    get_n_e = exp(splint(x_rec, n_e, n_e2, x))
  end function get_n_e

  ! Routine for computing tau at arbitrary x, using precomputed information
  function get_tau(x)
      implicit none
      real(dp), intent(in) :: x
      real(dp)             :: get_tau
      get_tau  = exp(splint(x_rec, log_tau, tau2, x))
  end function get_tau

  ! Routine for the derivative of tau used in ODEINT
  subroutine dtau_dx(x,tau, deriv)
    use healpix_types
    implicit none
    real(dp),                intent(in)  :: x
    real(dp), dimension(:), intent(in)  :: tau
    real(dp), dimension(:), intent(out) :: deriv

    deriv = -get_n_e(x)*sigma_T*exp(x)*c/get_H_p(x)
  end subroutine dtau_dx

  ! Routine for computing d_tau at arbitrary x, using precomputed information
  function get_dtau(x)
    implicit none

```

```
    real(dp), intent(in) :: x
    real(dp)             :: get_dtau
    get_dtau = splint_deriv(x_rec, log_tau, tau2, x)
    get_dtau = get_dtau*get_tau(x)
 end function get_dtau

  ! Routine for computing dd_tau at arbitrary x, using precomputed
    information
  function get_ddtau(x)
    implicit none

    real(dp), intent(in) :: x
    real(dp)             :: get_ddtau
    get_ddtau = splint(x_rec, tau2, tau22, x)
    get_ddtau = get_ddtau*get_tau(x) + get_dtau(x)**2/get_tau(x)
  end function get_ddtau

  ! Routine for computing g at arbitrary x, using precomputed
    information
  function get_g(x)
    implicit none

    real(dp), intent(in) :: x
    real(dp)             :: get_g
    get_g = splint(x_rec, g, g2, x)
  end function get_g

  ! Routine for computing the derivative of the visibility function
    , g, at arbitray x
  function get_dg(x)
    implicit none

    real(dp), intent(in) :: x
    real(dp)             :: get_dg
    get_dg = splint_deriv(x_rec, g, g2, x)
  end function get_dg

  ! Task: Complete routine for computing the second derivative of
    the visibility function, g, at arbitray x
  function get_ddg(x)
    implicit none

    real(dp), intent(in) :: x
    real(dp)             :: get_ddg
    get_ddg = splint(x_rec, g2, g22, x)
  end function get_ddg

end module rec_mod
```