Milestone 1

Elisabeth Strøm[1]
AST5220 15.02.2018

[1] Institute of Theoretical Astrophysics, University of Oslo

hei

1. Introduction

2. Method

Here we present the methods we have used to obtain the results in Section 3.

2.1. Theory

2.1.1. Future projects

3. Results

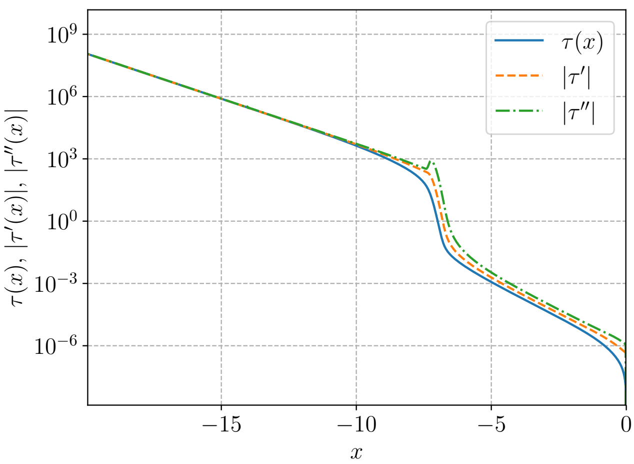

Fig. 1

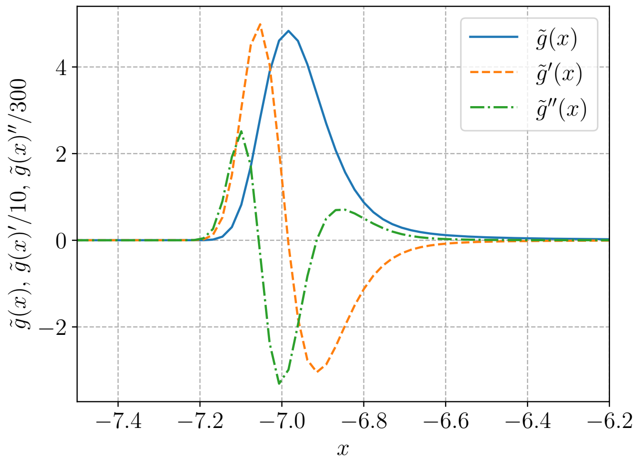

Fig. 2

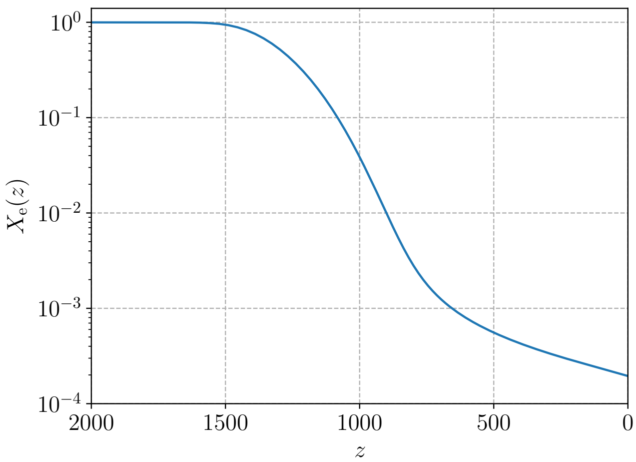

Fig. 3

4. Conclusions

5. Appendix

Source code

Listing 1: C:/Users/elini/Documents/AST5220/Ast5220/src/rec_mod.f90

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

  integer(i4b),                     private :: n              ! Number of grid points
  real(dp), allocatable, dimension(:), private :: x_rec    ! Grid
  real(dp), allocatable, dimension(:), private :: a_rec    ! Grid

  real(dp), allocatable, dimension(:), private :: tau, tau2, tau22 ! 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 ! Splined visibility function

contains

  subroutine initialize_rec_mod
    implicit none

    integer(i4b) :: i, j, k
    real(dp)   :: saha_limit, y, T_b, n_b, dydx, xmin, xmax, dx, f, n_e0, X_e0, xstart, xstop
    logical(lgt) :: use_saha
    real(dp), allocatable, dimension(:) :: X_e ! Fractional electron density, n_e / n_H
    real(dp)   :: step, stepmin, eps, z
    !real(dp), dimension(1) :: y

    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

    eps       = 1.d-10      ! spline error limit

    allocate(x_rec(n))
    allocate(X_e(n))
    allocate(tau(n))
    allocate(tau2(n))
    allocate(tau22(n))
    allocate(n_e(n))
    allocate(n_e2(n))
    allocate(g(n))
    allocate(g2(n))
    allocate(g22(n))

    ! Task: Fill in x (rec) grid
    write(*,*) "making x grid"
    x_rec(1) = xstart
    dx   = (xstop-xstart)/(n-1)

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


    ! Task: Compute X_e and n_e at all grid times
    step      = abs((x_rec(1) - x_rec(2))*1.d-3) ! n-1 maybe integration step length
    stepmin   = 0.d0

    write(*,*) "calculating X_e"
    use_saha = .true.
    do i = 1, n
       !write(*,*) "loop 2"
```

```
      n_b = Omega_b*rho_c/(m_H*exp(x_rec(i))**3)
      if (use_saha) then
         ! Use the Saha equation
         T_b = T_0/exp(x_rec(i))
         X_e0 = ((m_e*k_b*T_b)/(2.d0*pi*hbar**2))**(1.5d0) * exp(-epsilon_0/(k_b * T_b))/n_b
         X_e(i)= (-X_e0 + sqrt(X_e0**2 + 4.d0*X_e0))/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, step, stepmin, dXe_dx, bsstep, output)


      end if
      n_e(i) =X_e(i)*n_b
      write(*,*) "loop ",i
   end do


   ! Task: Compute splined (log of) electron density function
   n_e = log(n_e)
   write(*,*) "splining ne"
   call spline(x_rec, n_e, 1.d30, 1.d30, n_e2)

   ! Task: Compute optical depth at all grid points
   write(*,*) "calculating tau"

   tau(n) = 0.d0 ! initial condition, present day value
   do i = n-1, 1,-1
     tau(i) = tau(i+1)
     call odeint(tau(i:i), x_rec(i+1), x_rec(i), eps, step, stepmin, dtau_dx, bsstep, output)
   end do


   ! Task: Compute splined (log of) optical depth
   write(*,*) "splining tau and ddtau"
   call spline(x_rec, tau, 1.d30,1.d30, tau2)

   ! Task: Compute splined second derivative of (log of) optical depth
   call spline(x_rec, tau2,1.d30,1.d30,tau22)

!------------- visibility function g ---

   write(*,*) "calculating g"
   do i=1, n
     g(i) = -get_dtau(x_rec(i)) * exp(-tau(i))
   end do

   ! Task: Compute splined visibility function
   write(*,*) "splining g and ddg"
   call spline(x_rec, g, 1.d30, 1.d30, g2)

   ! Task: Compute splined second derivative of visibility function
   call spline(x_rec, g2, 1.d30, 1.d30, g22)


!---------- write to file ---
   write(*,*) "opening files "
   open (unit=1, file = 'x_tau.dat', status='replace')
   open (unit=2, file = 'x_g.dat', status='replace')
   open (unit=3, file = 'x_z_Xe.dat', status='replace')

   write(*,*) "writing stuff"
   do i=1, n
     z = exp(-x_rec(i))-1
     write (1,*) tau(i), get_dtau(x_rec(i)), get_ddtau(x_rec(i))

     write (2,*) g(i), get_dg(x_rec(i)), get_ddg(x_rec(i))
```

```fortran
      write (3,*) x_rec(i), z, X_e(i)

    end do

    write(*,*) " closing files "
    do i=1,3 ! close files
      close(i)
    end do

  end subroutine initialize_rec_mod


!---------------------- Peebles equation ----

  subroutine dXe_dx(x, X_e, dydx)
    ! we define dy/dx
    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) :: beta, beta2, alpha2, n_b, n1s, lambda_21s, lambda_alpha
    real(dp) :: C_r, T_b, H, phi2


    H    = get_H(x)
    !write(*,*) "H"
    T_b  = T_0/exp(x)
    n_b  = Omega_b*rho_c/(m_H*exp(x)**3)

    phi2 = 0.448d0*log(epsilon_0/(k_b * T_b))
    alpha2 = 64.d0*pi/sqrt(27.d0*pi) *(alpha/m_e)**2 *sqrt(epsilon_0/(k_b * T_b)) *phi2 *hbar**2/c
    beta = alpha2*((m_e*k_b*T_b)/(2.d0*pi*hbar**2))**(1.5d0) * exp(-epsilon_0/(k_b * T_b))

    !beta2 = beta * exp(3.d0*epsilon_0/(4.d0 * k_b*T_b))
    ! To avoid beta2 going to infinity, set it to 0

    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
    n1s = (1.d0 - X_e(1))* n_b ! X_e(1)

    lambda_alpha = H * (3.d0*epsilon_0)**3/((8.d0*pi)**2 * n1s)/(c*hbar)**3
    lambda_21s = 8.227d0

    C_r = (lambda_21s + lambda_alpha)/(lambda_21s + lambda_alpha + beta2)

    dydx = C_r/H * (beta * (1.d0-X_e(1)) - n_b * alpha2 * X_e(1)**2)

  end subroutine dXe_dx


  subroutine dtau_dx(x, tau, dydx)
    ! we define dy/dx
    use healpix_types
    implicit none
    real(dp),              intent(in) :: x
    real(dp), dimension(:), intent(in) :: tau
    real(dp), dimension(:), intent(out) :: dydx

    dydx = -get_n_e(x) * sigma_T * exp(x) * c/get_H_p(x)

  end subroutine dtau_dx



  ! Task: Complete routine for computing n_e at arbitrary x, using precomputed information
  ! Hint: Remember to exponentiate...
```

```fortran
function get_n_e(x)
  implicit none

  real(dp), intent(in) :: x
  real(dp)           :: get_n_e
  ! n_e is actually log(n_e)
  get_n_e = exp(splint(x_rec, n_e, n_e2, x))

end function get_n_e

! Task: Complete 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 = splint(x_rec, tau, tau2, x)

end function get_tau

! Task: Complete routine for computing the derivative of 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,tau,tau2,x)

end function get_dtau

! Task: Complete routine for computing the second derivative of 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)

end function get_ddtau

! Task: Complete routine for computing the visibility function, g, at arbitray x
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

! Task: Complete 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
```