Milestone 3

Elisabeth Strøm[1]
AST5220 16.04.2018

[1] Institute of Theoretical Astrophysics, University of Oslo

These results are in accordance with those obtained by Callin (2006).

1. Introduction

2. Method

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

2.1. Theory

The initial conditions when not including polarization, are

$$\Phi = 1 \tag{1}$$
$$\delta = \delta_b = \frac{3}{2}\Phi \tag{2}$$
$$v = v_b = \frac{ck}{2\mathcal{H}}\Phi \tag{3}$$
$$\Theta_0 = \frac{1}{2}\Phi \tag{4}$$
$$\Theta_1 = -\frac{ck}{6\mathcal{H}}\Phi \tag{5}$$
$$\Theta_2 = -\frac{20ck}{45\mathcal{H}\tau'}\Theta_1 \tag{6}$$
$$\Theta_l = -\frac{l}{2l+1}\frac{ck}{\mathcal{H}\tau'}\Theta_{l-1} \tag{7}$$
$$\tag{8}$$

We also have a general algebraic expression for Ψ, which we can implement when needed,

$$\Psi = -\Phi - 12\left(\frac{H_0}{cka}\right)^2\Omega_r\Theta_2 \tag{9}$$

The full equation set outside the tight coupling regime, is as follows;

$$R = \frac{4\Omega_r}{3\Omega_b a} \tag{10}$$
$$\Theta_0' = -\frac{ck}{\mathcal{H}}\Theta_1 - \Phi', \tag{11}$$
$$\Theta_1' = \frac{ck}{3\mathcal{H}}\Theta_0 - \frac{2ck}{3\mathcal{H}}\Theta_2 + \frac{ck}{3\mathcal{H}}\Psi + \tau'\left[\Theta_1 + \frac{1}{3}v_b\right], \tag{12}$$
$$\Theta_2' = \frac{2ck}{5\mathcal{H}}\Theta_1 - \frac{3ck}{5\mathcal{H}}\Theta_3 + \frac{9\tau'}{10}\Theta_l, \tag{13}$$
$$\text{for } 2 < l < l_{\max}: \tag{14}$$
$$\Theta_l' = \frac{lck}{(2l+1)\mathcal{H}}\Theta_{l-1} - \frac{(l+1)ck}{(2l+1)\mathcal{H}}\Theta_{l+1} + \tau'\Theta_l, \tag{15}$$
$$\Theta_{l_{\max}} = \frac{ck}{\mathcal{H}}\Theta_{l_{\max}-1} - c\frac{l_{\max}+1}{\mathcal{H}\eta(x)}\Theta_{l_{\max}} + \tau'\Theta_{l_{\max}}, \tag{16}$$
$$\Phi' = \Psi - \frac{1}{3}\left(\frac{ck}{\mathcal{H}}\right)^2\Phi + \frac{1}{2}\left(\frac{H_0}{\mathcal{H}}\right)^2 \tag{17}$$
$$\cdot\,[\Omega_m a^{-1}\delta + \Omega_b a^{-1}\delta_b + 4\Omega_r a^{-2}\Theta_0], \tag{18}$$
$$\delta = \frac{ck}{\mathcal{H}}v - 3\Phi', \tag{19}$$
$$\delta_b' = \frac{ck}{\mathcal{H}}v_b - 3\Phi', \tag{20}$$
$$v' = -v - \frac{ck}{\mathcal{H}}\Psi, \tag{21}$$
$$v_b' = -v_b - \frac{ck}{\mathcal{H}}\Psi + \tau' R(3\Theta_1 + v_b). \tag{22}$$
$$\tag{23}$$

When within the tight-coupling regime, there are certain changes, namly for $\Theta_{l>1}$ and Θ_1'. We also need to rewrite the equation for the baryon velocity, as for small values of .....

$$q = \frac{1}{(1+R)\tau' + \frac{\mathcal{H}'}{\mathcal{H}} - 1}\Big[-[(1-2R)\tau' + (1+R)\tau''] \cdot (3\Theta_1 + v_b) - \frac{ck}{\mathcal{H}}\Psi' + \left(1 - \frac{\mathcal{H}'}{\mathcal{H}}\right)\frac{ck}{\mathcal{H}}(2\Theta_2 - \Theta_0) - \frac{ck}{\mathcal{H}}\Theta_0'\Big] \tag{24}$$

$$v_b' = \frac{1}{1+R}\left[-v_b - \frac{ck}{\mathcal{H}}\Psi + R\left(q + \frac{ck}{\mathcal{H}}(2\Theta_2 - \Theta_0) - \frac{ck}{\mathcal{H}}\Psi\right)\right], \tag{25}$$

$$\Theta_1' = \frac{1}{3}(q - v_b'), \tag{26}$$

$$\Theta_2 = -\frac{20ck}{45\mathcal{H}\tau'}\Theta_1, \tag{27}$$

$$\Theta_l = -\frac{l}{2l+1}\frac{ck}{\mathcal{H}\tau'}\Theta_{l-1}. \tag{28}$$

2.2. Implementation

3. Results

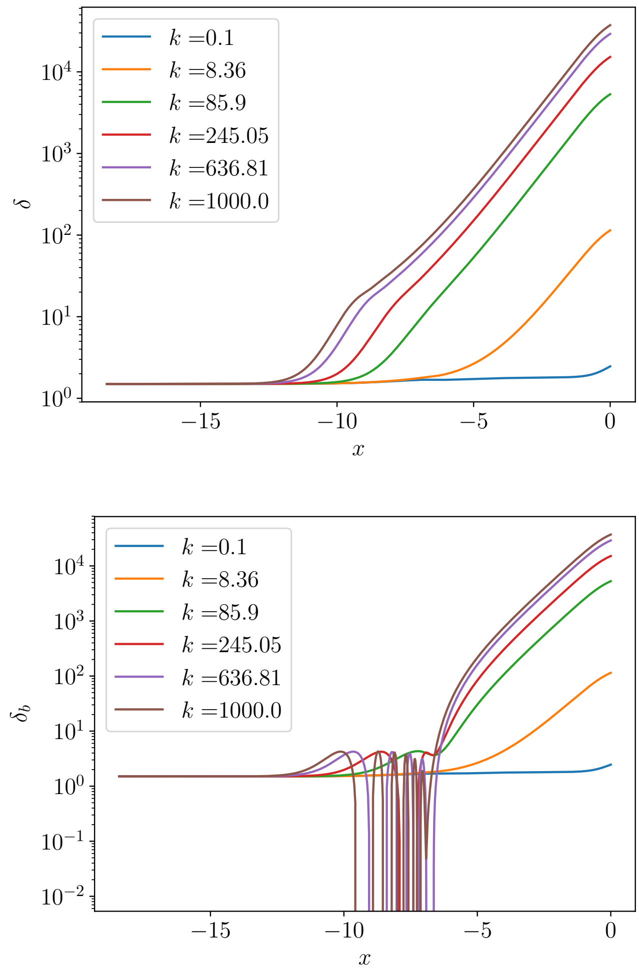

Fig. 1

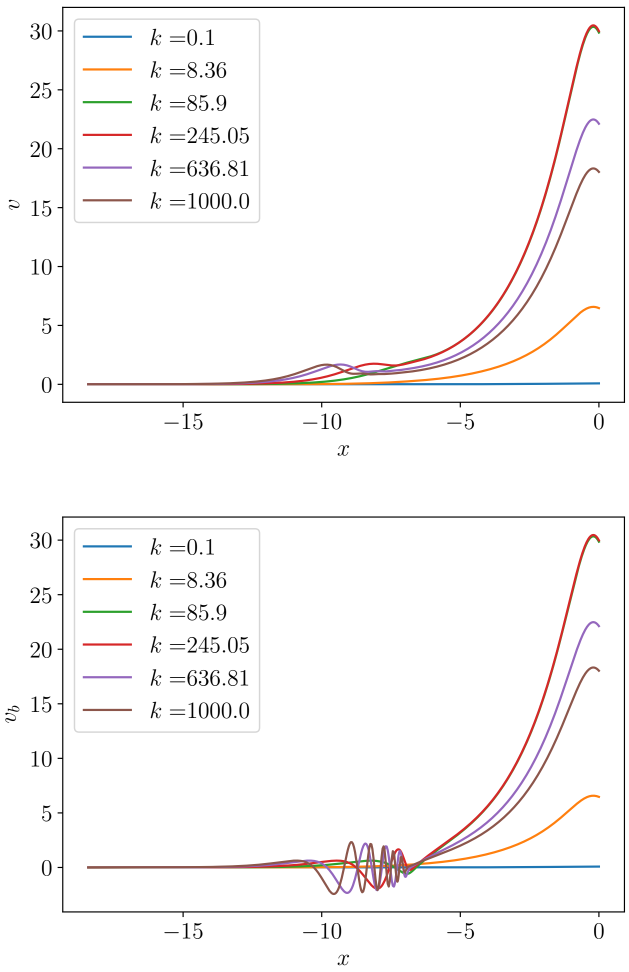

Fig. 2

4. Conclusions

References

Callin, P. 2006, ArXiv Astrophysics e-prints

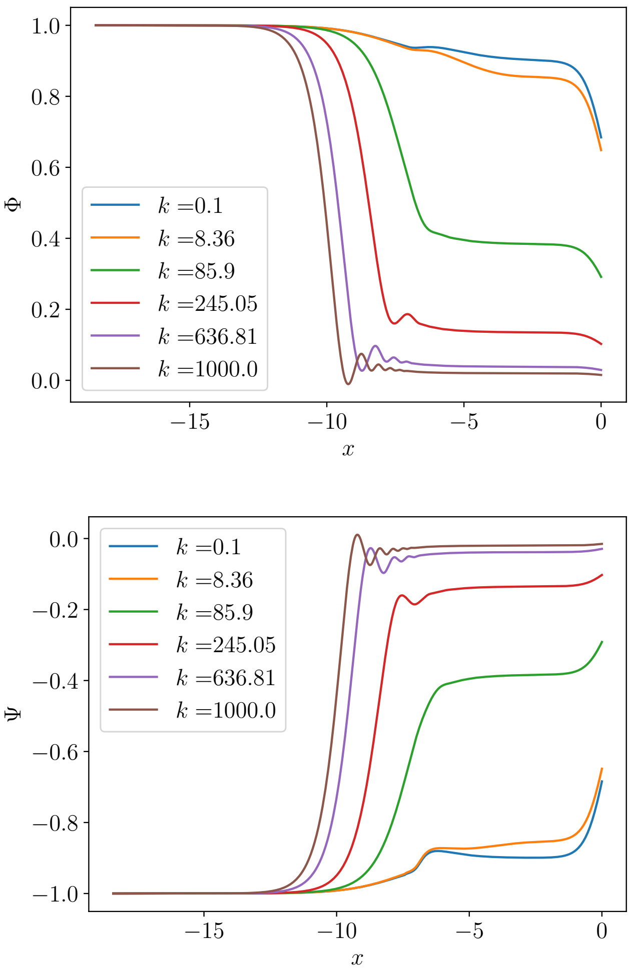
Φ
Ψ
x
k =0.1
k =8.36
k =85.9
k =245.05
k =636.81
k =1000.0

Fig. 3

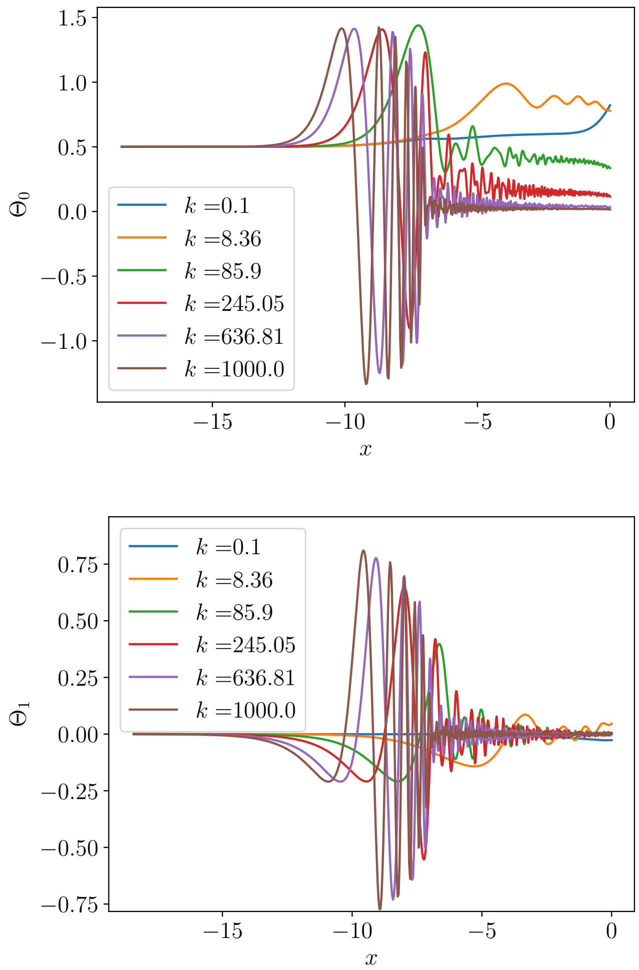
Θ0
Θ1
x
k =0.1
k =8.36
k =85.9
k =245.05
k =636.81
k =1000.0

Fig. 4

5. Appendix

Source code

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

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

  ! Accuracy parameters
  real(dp),     parameter, private :: a_init = 1.d-8
  real(dp),     parameter, private :: x_init = log(a_init)
  real(dp),     parameter, private :: k_min = 0.1d0 * H_0 / c
  real(dp),     parameter, private :: k_max = 1.d3 * H_0 / c
  integer(i4b), parameter        :: n_k     = 100
  integer(i4b), parameter, private :: lmax_int = 6

  ! Perturbation quantities
  real(dp), allocatable, dimension(:,:,:) :: Theta
  real(dp), allocatable, dimension(:,:) :: delta
  real(dp), allocatable, dimension(:,:) :: delta_b
  real(dp), allocatable, dimension(:,:) :: Phi
  real(dp), allocatable, dimension(:,:) :: Psi
  real(dp), allocatable, dimension(:,:) :: v
  real(dp), allocatable, dimension(:,:) :: v_b
  real(dp), allocatable, dimension(:,:) :: dPhi
  real(dp), allocatable, dimension(:,:) :: dPsi
  real(dp), allocatable, dimension(:,:) :: dv_b
  real(dp), allocatable, dimension(:,:,:) :: dTheta

  ! Fourier mode list
  real(dp), allocatable, dimension(:) :: ks

  ! Book-keeping variables
  real(dp),     private :: k_current
  integer(i4b), private :: npar = 6+lmax_int

  real(dp),     private :: ck, H_p, ckH_p, dt

contains


  ! NB!!! New routine for 4th milestone only; disregard until then!!!
  subroutine get_hires_source_function(k, x, S)
    implicit none

    real(dp), pointer, dimension(:), intent(out) :: k, x
    real(dp), pointer, dimension(:,:), intent(out) :: S

    integer(i4b) :: i, j
    real(dp)     :: g, dg, ddg, tau, dt, ddt, H_p, dH_p, ddHH_p, Pi, dPi, ddPi
    real(dp), allocatable, dimension(:,:) :: S_lores

    ! Task: Output a pre-computed 2D array (over k and x) for the
    !       source function, S(k,x). Remember to set up (and allocate) output
    !       k and x arrays too.
    !
    ! Substeps:
    !   1) First compute the source function over the existing k and x
    !      grids
    !   2) Then spline this function with a 2D spline
    !   3) Finally, resample the source function on a high-resolution uniform
    !      5000 x 5000 grid and return this, together with corresponding
    !      high-resolution k and x arrays
```

```fortran
end subroutine get_hires_source_function



! Routine for initializing and solving the Boltzmann and Einstein equations
subroutine initialize_perturbation_eqns
  implicit none

  integer(i4b) :: l, i

  ! Task: Initialize k-grid, ks; quadratic between k_min and k_max
  allocate(ks(n_k))
  do i = 1, n_k
     ks(i) = k_min + (k_max-k_min)*((i-1.d0)/(n_k-1.d0))**2.d0
  end do

  ! Allocate arrays for perturbation quantities
  allocate(Theta(0:n_t, 0:lmax_int, n_k))
  allocate(delta(0:n_t, n_k))
  allocate(delta_b(0:n_t, n_k))
  allocate(v(0:n_t, n_k))
  allocate(v_b(0:n_t, n_k))
  allocate(Phi(0:n_t, n_k))
  allocate(Psi(0:n_t, n_k))
  allocate(dPhi(0:n_t, n_k))
  allocate(dPsi(0:n_t, n_k))
  allocate(dv_b(0:n_t, n_k))
  allocate(dTheta(0:n_t, 0:lmax_int, n_k))
  ! Task: Set up initial conditions for the Boltzmann and Einstein equations
  !Theta(:,:,:) = 0.d0
  !dTheta(:,:,:) = 0.d0
  !dPhi(:,:) = 0.d0
  !dPsi(:,:) = 0.d0

  Phi(0,:)    = 1.d0
  delta(0,:) = 1.5d0 * Phi(0,:)
  delta_b(0,:) = delta(0,:)
  Theta(0,0,:) = 0.5d0*Phi(0,:)
  H_p         = get_H_p(x_init)
  dt          = get_dtau(x_init)

  do i = 1, n_k
     ckH_p      = c*ks(i)/H_p

     v(0,i)      = ckH_p/2.d0*Phi(0,i)
     v_b(0,i)    = v(0,i)

     Theta(0,1,i) = -ckH_p/6.d0*Phi(0,i)
     Theta(0,2,i) = -20.d0/45.d0*ckH_p/(dt)*Theta(0,1,i)
     do l = 3, lmax_int
        Theta(0,l,i) = - l/(2.d0*l + 1.d0)*ckH_p/dt *Theta(0,l-1,i)
     end do
     Psi(0,i) = - Phi(0,i) - 12.d0*(H_0/(c*ks(i)*a_init))**2.d0*Omega_r*Theta(0,2,i)
  end do

end subroutine initialize_perturbation_eqns

subroutine integrate_perturbation_eqns
  implicit none

  integer(i4b) :: i, j, k, l, i_tc
  real(dp)    :: x1, x2
  real(dp)    :: eps, hmin, h1, x_tc, t1, t2


  real(dp), allocatable, dimension(:) :: y, y_tight_coupling, dydx

  eps    = 1.d-8
  hmin   = 0.d0
```

```fortran
h1    = 1.d-5

allocate(y(npar))
allocate(dydx(npar))
allocate(y_tight_coupling(7))

! Propagate each k-mode independently
do k = 1, n_k
   write(*,*) "starting k loop in integrate"
   k_current = ks(k) ! Store k_current as a global module variable
   ck = c*k_current

   ! Initialize equation set for tight coupling
   y_tight_coupling(1) = delta(0,k)
   y_tight_coupling(2) = delta_b(0,k)
   y_tight_coupling(3) = v(0,k)
   y_tight_coupling(4) = v_b(0,k)
   y_tight_coupling(5) = Phi(0,k)
   y_tight_coupling(6) = Theta(0,0,k)
   y_tight_coupling(7) = Theta(0,1,k)

   ! Find the time to which tight coupling is assumed,
   ! and integrate equations to that time
   write(*,*) " entering get_tight_coupling_time"
   x_tc = get_tight_coupling_time(k_current)
   write(*,*) "x_tc", x_tc
   write(*,*) "k", k
   ! Task: Integrate from x_init until the end of tight coupling, using
   !      the tight coupling equations
   write(*,*) "integrating tight coupling equations"
   i_tc = 1

   do while(x_t(i_tc)< x_tc)
      !write(*,*) "evol i_tc lopp!", i_tc
      ! Integration while tc
      call odeint(y_tight_coupling, x_t(i_tc-1), x_t(i_tc),eps, h1, hmin, dy_tc_dx, bsstep, output)
      ! some parameters
      ckH_p  = ck*get_H_p(x_t(i_tc))
      dt     = get_dtau(x_t(i_tc))

      delta(i_tc,k)    = y_tight_coupling(1)
      delta_b(i_tc,k)  = y_tight_coupling(2)
      v(i_tc,k)        = y_tight_coupling(3)
      v_b(i_tc,k)      = y_tight_coupling(4)
      Phi(i_tc,k)      = y_tight_coupling(5)
      Theta(i_tc,0,k)  = y_tight_coupling(6)
      Theta(i_tc,1,k)  = y_tight_coupling(7)
      Theta(i_tc,2,k)  = -20.d0/45.d0*ckH_p/dt * Theta(i_tc,1,k)
      do l = 3, lmax_int
         Theta(i_tc,l,k) = - l/(2.d0*l + 1.d0)*ckH_p/dt *Theta(i_tc,l-1,k)
      end do
      Psi(i_tc,k)      = - Phi(i_tc,k) - 12.d0*(H_0/(ck*a_t(i_tc)))**2.d0*Omega_r*Theta(i_tc,2,k)


      ! The store derivatives necessary here?
      call dy_tc_dx(x_t(i_tc), y_tight_coupling, dydx)
      dv_b(i_tc,k)   = dydx(4)
      dPhi(i_tc,k)   = dydx(5)
      dTheta(i_tc,0,k) = dydx(6)
      dTheta(i_tc,1,k) = dydx(7)
      dTheta(i_tc,2,k) = 2.d0/5.d0*ckH_p*Theta(i_tc,1,k) -
          3.d0/5.d0*ckH_p*Theta(i_tc,3,k)+dt*0.9d0*Theta(i_tc,2,k)

      do l=3,lmax_int-1
         dTheta(i_tc,l,k) = l/(2.d0*l+1.d0)*ckH_p*dTheta(i_tc,l-1,k) -
             (l+1.d0)/(2.d0*l+1.d0)*ckH_p*dTheta(i_tc,l+1,k) + dt*Theta(i_tc,l,k)
       end do
      dPsi(i_tc,k)   = -dPhi(i_tc,k) - 12.d0*(H_0/(ck*a_t(i_tc)))**2.d0
          *Omega_r*(-2.d0*Theta(i_tc,2,k)+dTheta(i_tc,2,k))
```

```fortran
       i_tc = i_tc+1
    end do ! end while do

    ! Task: Set up variables for integration from the end of tight coupling
    ! until today
    y(1:7) = y_tight_coupling(1:7)
    y(8)  = Theta(i_tc-1,2,k)
    do l = 3, lmax_int
       y(6+l) = Theta(i_tc-1,l,k)
    end do

    write(*,*) "integrating non-tight coupling equations"
    do i = i_tc, n_t-1

        !write(*,*) "after tc loop", i
       ! Task: Integrate equations from tight coupling to today
        call odeint(y, x_t(i-1), x_t(i),eps, h1, hmin, dy_dx, bsstep, output)
       ! Task: Store variables at time step i in global variables
       !write(*,*) "made it through"
       delta(i,k) = y(1)
       delta_b(i,k) = y(2)
       v(i,k)      = y(3)
       v_b(i,k)    = y(4)
       Phi(i,k)    = y(5)

       do l = 0, lmax_int
          Theta(i,l,k) = y(6+l)
       end do

       Psi(i,k)    = - Phi(i,k) - 12.d0*(H_0/(ck*a_t(i)))**2.d0*Omega_r*Theta(i,2,k)

       ! Task: Store derivatives that are required for C_l estimation
       call dy_dx(x_t(i), y, dydx)
       dv_b(i,k)    = dydx(4)
       dPhi(i,k)    = dydx(5)
       do l=0, lmax_int
         dTheta(i,l,k) = dydx(6+l)
       end do
       dPsi(i,k)    = -dPhi(i,k) - 12.d0*(H_0/(ck*a_t(i)))**2 * Omega_r*(dTheta(i,2,k)-2.d0*Theta(i,2,k))
    end do

  end do

  deallocate(y_tight_coupling)
  deallocate(y)
  deallocate(dydx)

end subroutine integrate_perturbation_eqns


! Task: Complete the following routine, such that it returns the time at which
!       tight coupling ends. In this project, we define this as either when
!       dtau < 10 or c*k/(H_p*dt) > 0.1 or x > x(start of recombination)
function get_tight_coupling_time(k)
  implicit none

  real(dp), intent(in) :: k
  real(dp)            :: get_tight_coupling_time
  real(dp)            :: x, x_start_rec, z_start_rec
  integer(i4b)        :: i, n

  z_start_rec = 1630.4d0             ! Redshift of start of recombination
  x_start_rec = -log(1.d0 + z_start_rec) ! x of start of recombination

  n=1d4
  do i=0,n
     x = x_init + i*(0.d0- x_init)/n
     dt     = get_dtau(x)
     H_p    = get_H_p(x)
```

```fortran
       if (abs(dt) > 10.d0 .and. abs((c*k/H_p)/dt) <= 0.1d0 .and. x<=x_start_rec) then
          get_tight_coupling_time = x
       end if

    end do

  end function get_tight_coupling_time

  subroutine dy_tc_dx(x, y, dydx)
    ! Tight coupling, only l=0,1 for dTheta
    use healpix_types
    implicit none
    real(dp),               intent(in) :: x
    real(dp), dimension(:), intent(in) :: y
    real(dp), dimension(:), intent(out) :: dydx

    real(dp) :: delta, delta_b, v, v_b, Phi, Theta0, Theta1, Theta2, Psi
    real(dp) :: ddelta, ddelta_b, dv, dv_b, dPhi, dTheta0, dTheta1
    real(dp) :: q, R, a, dH_p, ddt

    delta     = y(1)
    delta_b   = y(2)
    v         = y(3)
    v_b       = y(4)
    Phi       = y(5)
    Theta0    = y(6)
    Theta1    = y(7)

    a      = exp(x)
    dt     = get_dtau(x)
    ddt    = get_ddtau(x)
    H_p    = get_H_p(x)
    dH_p   = get_dH_p(x)
    ckH_p  = ck/H_p

    ! Derivatives
    Theta2   = - 20.d0*ckH_p/(45.d0*dt) * Theta1
    R        = (4.d0*Omega_r)/(3.d0*Omega_b*a)
    Psi      = - Phi - 12.d0*(H_0/(ck*a))**2.d0 * Omega_r * Theta2

    dPhi     = Psi - ckH_p**2.d0/3.d0*Phi + 0.5d0*(H_0/H_p)**2.d0 * (Omega_m/a*delta + Omega_b/a*delta_b +
        4.d0*Omega_r*Theta0/a**2.d0)
    dv       = - v - ckH_p * Psi

    ddelta   = ckH_p * v - 3.d0*dPhi
    ddelta_b = ckH_p * v_b - 3.d0*dPhi

    dTheta0 = - ckH_p*Theta1 - dPhi
    !------ special for tight coupling ------
    q    = (-((1.d0-2.d0*R)*dt + (1.d0+R)*ddt)*(3.d0*Theta1 + v_b)- ckH_p*Psi +
         (1.d0-(dH_p/H_p))*ckH_p*(-Theta0 + 2.d0*Theta2) - ckH_p*dTheta0)/((1.d0+R)*dt + (dH_p/H_p) -1.d0)

    dv_b = (1.d0/(1.d0 + R)) * (-v_b - ckH_p*Psi + R*(q + ckH_p*(-Theta0 + 2.d0*Theta2) - ckH_p*Psi))
    dTheta1 = (1.d0/3.d0)*(q-dv_b)
    ! ---------------------------


    ! Final array
    dydx(1) = ddelta
    dydx(2) = ddelta_b
    dydx(3) = dv
    dydx(4) = dv_b
    dydx(5) = dPhi
    dydx(6) = dTheta0
    dydx(7) = dTheta1

  end subroutine dy_tc_dx

  subroutine dy_dx(x, y, dydx)
    ! we define dy/dx
```

```fortran
  use healpix_types
  implicit none
  real(dp),             intent(in) :: x
  real(dp), dimension(:), intent(in) :: y
  real(dp), dimension(:), intent(out) :: dydx

  integer(i4b) :: l
  real(dp) :: delta, delta_b, v, v_b, Phi, Theta0, Theta1, Theta2, Psi
  real(dp) :: ddelta, ddelta_b, dv, dv_b, dPhi, dTheta0, dTheta1
  real(dp) :: q, R, a, eta


  ! what we take in, use in derivation
  delta     = y(1)
  delta_b   = y(2)
  v         = y(3)
  v_b       = y(4)
  Phi       = y(5)
  Theta0    = y(6)
  Theta1    = y(7)
  Theta2    = y(8)
  ! Theta3-6: y(9)-y(12)

  a      = exp(x)
  eta    = get_eta(x)
  dt     = get_dtau(x)
  H_p    = get_H_p(x)
  ckH_p  = ck/H_p

  ! Derivatives
  R       = (4.d0*Omega_r)/(3.d0*Omega_b*a)
  Psi     = - Phi - 12.d0*(H_0/(ck*a))**2.d0 * Omega_r * Theta2
  dPhi    = Psi - ckH_p**2.d0/3.d0*Phi + 0.5d0*(H_0/H_p)**2.d0 * (Omega_m/a*delta + Omega_b/a*delta_b +
      4.d0*Omega_r*Theta0/a**2.d0)

  dv      = - v - ckH_p*Psi
  dv_b    = - v_b - ckH_p*Psi + dt*R*(3.d0*Theta1 + v_b)

  ddelta  = ckH_p * v - 3.d0*dPhi
  ddelta_b = ckH_p * v_b - 3.d0*dPhi

  dTheta0 = - ckH_p*Theta1 - dPhi
  dTheta1 = ckH_p/3.d0*Theta0 - 2.d0/3.d0*ckH_p*Theta2 + ckH_p/3.d0*Psi + dt*(Theta1 + v_b/3.d0)

  ! dTheta2 - dTheta5
  do l = 2, lmax_int-1
     dydx(6+l) = l/(2.d0*l+1.d0)*ckH_p*y(6+l-1) - (l+1.d0)/(2.d0*l + 1.d0)*ckH_p*y(6+l+1) + dt*(y(6+l) -
         0.1d0*y(6+l)*abs(l==2))
  end do


  ! Final array
  dydx(1) = ddelta
  dydx(2) = ddelta_b
  dydx(3) = dv
  dydx(4) = dv_b
  dydx(5) = dPhi
  dydx(6) = dTheta0
  dydx(7) = dTheta1
  ! dTheta6
  dydx(6+l) = ckH_p*y(6+l-1) - c*(l+1.d0)/(H_p*eta)*y(6+l) + dt*y(6+l)

end subroutine dy_dx

subroutine write_to_file_evolution_mod
   use healpix_types
  implicit none

  integer(i4b) :: i
  integer(i4b), dimension(6) :: k
```

```
    write(*,*) "writing to file; evolution_mod"

    k(1:6)=(/1, 10, 30, 50, 80, 100 /)
    !k(1:6)=(/1, 2, 3, 4, 5, 10 /)

!---------- write to file ---
    write(*,*) "opening files "
    open (unit=0, file = 'k_ks.dat', status='replace')
    open (unit=1, file = 'x_t.dat', status='replace')
    open (unit=2, file = 'Phi.dat', status='replace')
    open (unit=3, file = 'Psi.dat', status='replace')
    open (unit=4, file = 'delta.dat', status='replace')
    open (unit=5, file = 'delta_b.dat', status='replace')
    open (unit=6, file = 'v.dat', status='replace')
    open (unit=7, file = 'v_b.dat', status='replace')
    open (unit=8, file = 'Theta0.dat', status='replace')
    open (unit=9, file = 'Theta1.dat', status='replace')

    do i=1,6
       write(0,*) k(i),ks(k(i))
    end do

    write(*,*) "writing stuff"
    do i=0, n_t-1
      write (1,*) x_t(i)
      write (2,'(*(2X, ES14.6E3))') Phi(i,k(1)),Phi(i,k(2)),Phi(i,k(3)),Phi(i,k(4)),Phi(i,k(5)),Phi(i,k(6))
      write (3,'(*(2X, ES14.6E3))') Psi(i,k(1)),Psi(i,k(2)),Psi(i,k(3)),Psi(i,k(4)),Psi(i,k(5)),Psi(i,k(6))
      write (4,'(*(2X, ES14.6E3))')
          delta(i,k(1)),delta(i,k(2)),delta(i,k(3)),delta(i,k(4)),delta(i,k(5)),delta(i,k(6))
      write (5,'(*(2X, ES14.6E3))')
          delta_b(i,k(1)),delta_b(i,k(2)),delta_b(i,k(3)),delta_b(i,k(4)),delta_b(i,k(5)),delta_b(i,k(6))
      write (6,'(*(2X, ES14.6E3))') v(i,k(1)),v(i,k(2)),v(i,k(3)),v(i,k(4)),v(i,k(5)),v(i,k(6))
      write (7,'(*(2X, ES14.6E3))') v_b(i,k(1)),v_b(i,k(2)),v_b(i,k(3)),v_b(i,k(4)),v_b(i,k(5)),v_b(i,k(6))
      write (8,'(*(2X, ES14.6E3))')
          Theta(i,0,k(1)),Theta(i,0,k(2)),Theta(i,0,k(3)),Theta(i,0,k(4)),Theta(i,0,k(5)),Theta(i,0,k(6))
      write (9,'(*(2X, ES14.6E3))')
          Theta(i,1,k(1)),Theta(i,1,k(2)),Theta(i,1,k(3)),Theta(i,1,k(4)),Theta(i,1,k(5)),Theta(i,1,k(6))

    end do

    write(*,*) "closing files "
    do i=0, 9
      close(i)
    end do

  end subroutine write_to_file_evolution_mod

end module evolution_mod
```