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

We assume the same background cosmology as established in the previous two reports. As before, we will be defining the time parameter by the logarithm of the scale factor of the universe,

$$x = \log a(t). \tag{1}$$

We will be looking at the time frame from $a = 10^{-8}$, up until today, at $a = 1$. The derivative of some parameter f with respect to x, is written as $f' = \frac{\mathrm{d}f}{\mathrm{d}x}$. However, we will now look at the evolution of perturbations in the universe, and so we have the perturbed metric,

$$ds^2 = -(1 + 2\Psi)dt^2 + a^2(1 + 2\Phi)(dx^2 + dy^2 + dz^2). \tag{2}$$

The evolution of perturbations in the universe are governed by the linear Boltzmann-Einstein equations. Without taking neutrinos and polarization into consideration, they are given by:

$$R = \frac{4\Omega_r}{3\Omega_b a}, \tag{3}$$

$$\Theta_0' = -\frac{ck}{\mathcal{H}}\Theta_1 - \Phi', \tag{4}$$

$$\Theta_1' = \frac{ck}{3\mathcal{H}}\Theta_0 - \frac{2ck}{3\mathcal{H}}\Theta_2 + \frac{ck}{3\mathcal{H}}\Psi + \tau'\left[\Theta_1 + \frac{1}{3}v_b\right], \tag{5}$$

$$\Theta_2' = \frac{2ck}{5\mathcal{H}}\Theta_1 - \frac{3ck}{5\mathcal{H}}\Theta_3 + \frac{9\tau'}{10}\Theta_l, \tag{6}$$

$$\text{for } 2 < l < l_{\max}: \tag{7}$$

$$\Theta_l' = \frac{lck}{(2l+1)\mathcal{H}}\Theta_{l-1} - \frac{(l+1)ck}{(2l+1)\mathcal{H}}\Theta_{l+1} + \tau'\Theta_l, \tag{8}$$

$$\Theta_{l_{\max}} = \frac{ck}{\mathcal{H}}\Theta_{l_{\max}-1} - c\frac{l_{\max}+1}{\mathcal{H}\eta(x)}\Theta_{l_{\max}} + \tau'\Theta_{l_{\max}}, \tag{9}$$

$$\Phi' = \Psi - \frac{1}{3}\left(\frac{ck}{\mathcal{H}}\right)^2\Phi + \frac{1}{2}\left(\frac{H_0}{\mathcal{H}}\right)^2 \tag{10}$$

$$\cdot\,[\Omega_m a^{-1}\delta + \Omega_b a^{-1}\delta_b + 4\Omega_r a^{-2}\Theta_0], \tag{11}$$

$$\delta = \frac{ck}{\mathcal{H}}v - 3\Phi', \tag{12}$$

$$\delta_b' = \frac{ck}{\mathcal{H}}v_b - 3\Phi', \tag{13}$$

$$v' = -v - \frac{ck}{\mathcal{H}}\Psi, \tag{14}$$

$$v_b' = -v_b - \frac{ck}{\mathcal{H}}\Psi + \tau' R(3\Theta_1 + v_b). \tag{15}$$

$$\tag{16}$$

Note that all of these quantities are dimensionless.

Here, c is the speed of light, $\mathcal{H} = aH$ is the scaled Hubble parameter, k is the wavenumber of the pertrubations. We are working in Fourier space throughout this report, so k is also the Fourier transformed position $\mathbf{x}$. τ is the optical thickness. Φ is the perturbations to the spatial curvature, δ, is the cold dark matter (CDM) density perturbations, and δ_b is the baryonic density perturbations. The velocities of the perturbations are denoted v and v_b, for CDM and baryons, respectively. The different Θ_l for $l \geq 0$ are an angular scale for the photon temperature perturbations in the CMB, expanded into multipoles. They are the integral over the temperature perturbations over all directions. A low l means we are looking at large scale temperature perturbations, while a low l means we are observing low scale temperature perturbations. The mean temperature Θ_0 is called the monopole, Θ_1 is the dipole, and Θ_2 is the quadrupole.

We also have a general algebraic expression for the perturbations to the metric, corresponding to the Newtonian gravitational potential. It is denoted Ψ, which we can implement when needed,

$$\Psi = -\Phi - 12\left(\frac{H_0}{cka}\right)^2 \Omega_r \Theta_2, \tag{17}$$

where H_0 is the Hubble parameter today, and Ω_r is the radiation density parameter of today.

When within the tight-coupling regime, there are certain changes, namely for $\Theta_{l>1}$ and Θ_1'. We also need to rewrite the equation for the baryon velocity, since at early times, τ' is very large, while the factor with which it is multiplied, $(3\Theta_1 - v_b)$ is very small. This product is then numerically unstable. We solve this by expanding $(3\Theta_1 - v_b)$ in powers of $1/\tau'$ (Callin 2006). The final changes to the Einstein-Boltzmann equations in the tight coupling regime, are then,

$$\begin{aligned} q = \frac{1}{(1+R)\tau' + \frac{\mathcal{H}'}{\mathcal{H}} - 1}\Big[& -[(1-2R)\tau' + (1+R)\tau''] \\ & \cdot (3\Theta_1 + v_b) - \frac{ck}{\mathcal{H}}\Psi' \\ & + \left(1 - \frac{\mathcal{H}'}{\mathcal{H}}\right)\frac{ck}{\mathcal{H}}(2\Theta_2 - \Theta_0) - \frac{ck}{\mathcal{H}}\Theta_0'\Big], \end{aligned} \tag{18}$$

$$v_b' = \frac{1}{1+R}\left[-v_b - \frac{ck}{\mathcal{H}}\Psi + R\left(q + \frac{ck}{\mathcal{H}}(2\Theta_2 - \Theta_0) - \frac{ck}{\mathcal{H}}\Psi\right)\right], \tag{19}$$

$$\Theta_1' = \frac{1}{3}(q - v_b'), \tag{20}$$

$$\Theta_2 = -\frac{20ck}{45\mathcal{H}\tau'}\Theta_1, \tag{21}$$

$$\Theta_l = -\frac{l}{2l+1}\frac{ck}{\mathcal{H}\tau'}\Theta_{l-1}. \tag{22}$$

The initial conditions when not including polarization and neutrinos, are

$$\Phi = 1 \tag{23}$$

$$\delta = \delta_b = \frac{3}{2}\Phi \tag{24}$$

$$v = v_b = \frac{ck}{2\mathcal{H}}\Phi \tag{25}$$

$$\Theta_0 = \frac{1}{2}\Phi \tag{26}$$

$$\Theta_1 = -\frac{ck}{6\mathcal{H}}\Phi \tag{27}$$

$$\Theta_2 = -\frac{20ck}{45\mathcal{H}\tau'}\Theta_1 \tag{28}$$

$$\Theta_l = -\frac{l}{2l+1}\frac{ck}{\mathcal{H}\tau'}\Theta_{l-1}. \tag{29}$$

$$\tag{30}$$

2.2. Implementation

We implement these equations numerically using the provided code structure. The results can be seen in the next section. The Einstein-Boltzmann equations are computed both as a function of time, x, and of the wavenumbers, k. The k's represent Fourier modes, and have units m^{-1}. However, we want them to be unitless, and so we multiply by H_0 and divide by c.

The k's are distributed quadratically for better results (Callin 2006),

$$k_i = k_{\min} + (k_{\max} - k_{\min})\left(\frac{i}{100}\right)^2. \tag{31}$$

We use 100 different values of k, extending from $k_{\min} = 0.1H_0/c$ to $k_{+}\max = 1000H_0/c$.

We also extend the x_t function from Milestone 1, to include 300 grid points prior to recombination, in addition to the 500 grid points we have previously defined.

We start out in the tight-coupling regime, which ends at recombination. Tight coupling ends when $\tau' < 10$, or $\frac{ck}{\mathcal{H}\tau'} > 0.1$, or $x > x_{\text{start rec}}$. We define a function which, according to these conditions, determines when to switch between the tight-coupling equations, and non-tight-coupling equations.

3. Results

Here are our results and our discussion of them.

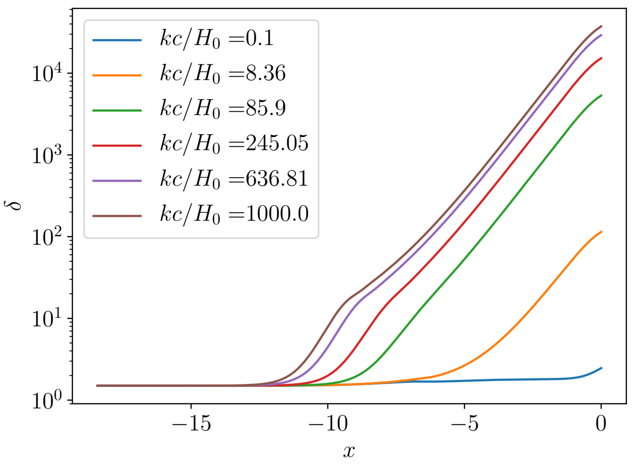
First, we look at the evolution of the dark matter and baryon density perturbations. They can be seen in Figure 1.

We start with the dark matter perturbations. We see that they evolve differently for different scales of the modes, represented by k. While the overdensities are constant for all scales at early times, they eventually start increasing at different times, depending on k. This is due to the fact that, early on, all modes are outside the particle horizon, and so they are constant. The smaller scale perturbations, meaning larger k, enter the horizon first on behalf of their smaller size, and so they begin to grow earlier than larger scale perturbations (small k). As cold dark matter is assumed to be a pressure-less fluid only affected by gravity, there is nothing stopping the DM particles in clumping together more and more, and continue to grow. Hence, the perturbations continuous increase through time.

In the case of $kc/H_0 = 1000$, 636.81 and 245.05, we see that they also seem to have two different phases of growth. Just as they enter the horizon they grow rapidly, but after some time they start increasing at a slower rate. This is because they entered the horizon while the universe was radiation dominated. After matter starts dominating, the overdensities begin to grow proportional to the scale factor, $\delta \propto a$. Perturbations that enter the horizon after matter-radiation equality, such as the green, orange, and blue line, will have a growth proportional to the scale factor straight away.

It is the same thing that happens for the baryon density perturbations. The small scale overdensities enter the horizon first, while the large scale overdensities enters later. Unlike the dark matter, the baryons does not clump together and continue to grow straight after horizon entry, however they do eventually grow at the same rate. This can be seen in Figure 2, where δ and δ_b is plotted together. It seems that up until just after horizon entry, and after recombination, the different matter densities grow together. This is simply due to baryonic matter accumulating in the already overdense regions made by the dark matter.

However, between these two times, the baryonic densities oscillates. This is because, unlike dark matter, baryons are affected by other forces, such as pressure. As the baryons clump together in dark matter halos, the thermal motion of the particles increases, which increases the pressure. The photons are tightly coupled to the baryons due to Thomson scattering. This fluid



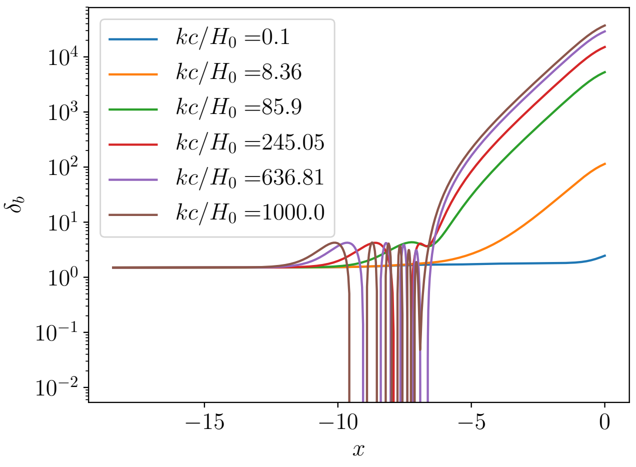


Fig. 1: The topmost plot shows the evolution of the dark matter density perturbations δ as a function of time, x. The lower frame shows the same for the baryon density perturbations, δ_b. Note that the y-axis is log scaled. These graphs correspond to the k values shown in the legend. We see that while the DM perturbations start growing continuously, the low-scale baryon perturbations oscillate around $x = -10$ and does not start growing until after recombination, around $x = -7$. Then they grow at the same rate as δ.

is relativistic with a tremendous Jeans mass. Hence, the radiation pressure is hindering any baryonic structure growth prior to recombination around $x = -7$. The result is that particles are pushed away, gravity from the overdense regions pull them back inn, pressure builds and push them out again, and so on. This results in the oscillations we see in the δ_b plot. After recombination, the baryon overdensities can collapse into the DM halos, and so, grow on a scale equal to that of δ. We see that perturbations that enter the horizon after recombination (orange and blue) evolve without oscillations.

In figure 3, we see the velocities of the DM parturbations, v, and the baryon pertrubations, v_b. These plots mirrors what have already been said of δ and δ_b: As the perturbations enter the horizon and begin to grow, their velocities naturally increase as well. In the case of the dark matter, we see that the velocity curves decrease just after $x = -10$, corresponding to where the δ curves transition to a slower growth rate. The velocity then quickly picks up again, and starts increasing, reaching a peak just before present day. Today, it seems that the velocity of the

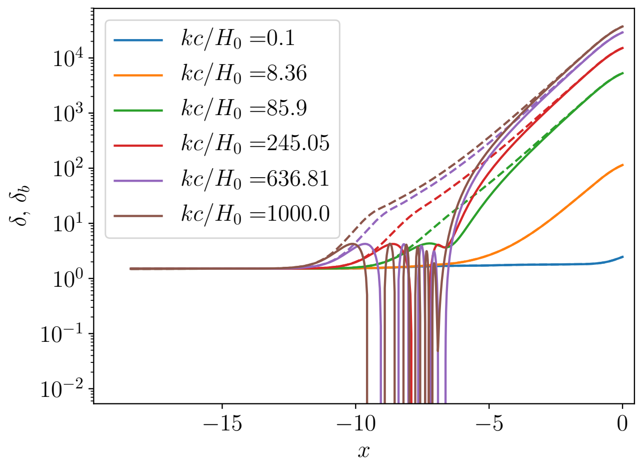


Fig. 2: δ and δ_b overplotted against x. Lines with the same color have the same k value. The y-axis is logarithmic

perturbations for all k modes, have started to slow down. For the baryons, we see that v_b also oscillates in the same area as the density perturbations δ_b oscillates, which is what we would expect. Beyond this, v_b follows the same curves as v. We see that for the smallest mode, $kc/H_0 = 0.1$, the velocity is constantly zero.

4. Conclusions

References

Callin, P. 2006, ArXiv Astrophysics e-prints

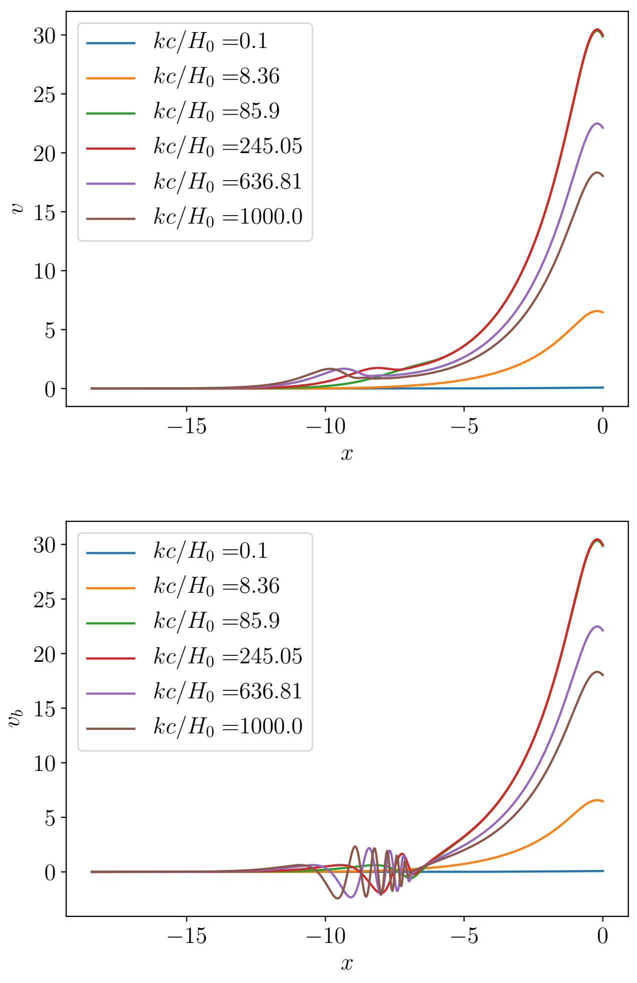


Fig. 3: The evolution of the velocity curves for the density perturbations for dark matter, v (top), and baryonic matter, v_b (lower frame), against time, x. We see that the velocity for all matter starts increasing upon horizon entry. v_b starts oscillating not long after this, consistent with the behavior of δ_b. v have a slight decrease before increasing rapidly again, consistent with the evolution of δ.

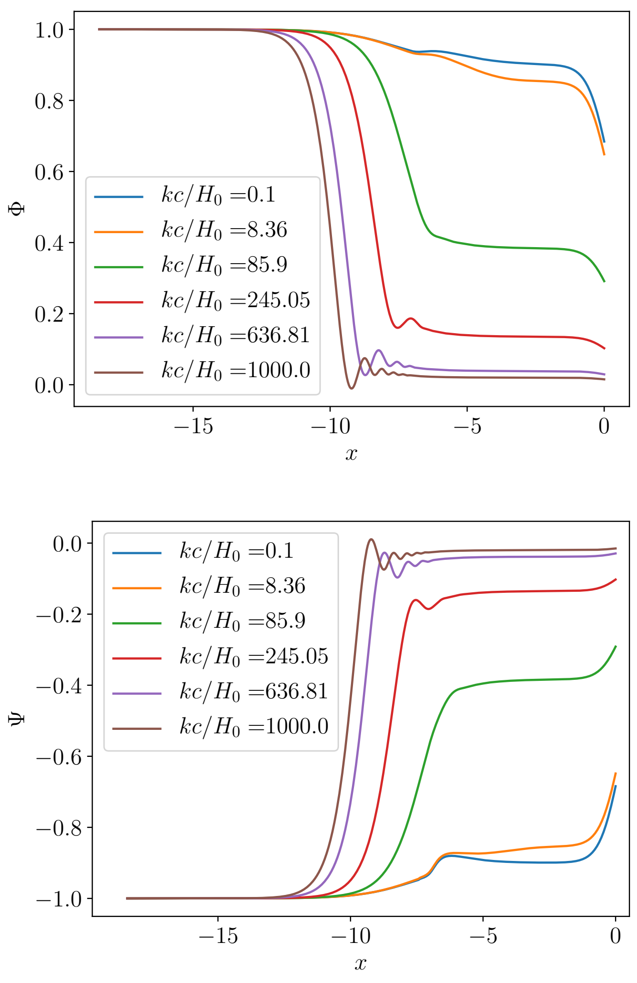


Fig. 4

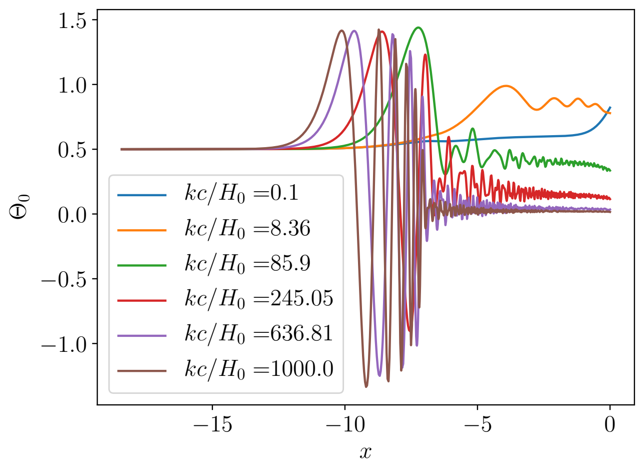


Fig. 5

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+lmax_int) = ckH_p*y(6+lmax_int-1) - c*(lmax_int+1.d0)/(H_p*eta)*y(6+lmax_int) + dt*y(6+lmax_int)

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
```