Milestone 4

Elisabeth Strøm[1]
AST5220 20.05.2018

[1] Institute of Theoretical Astrophysics, University of Oslo

We compute the CMB angular power spectrum, C_l, through integration of the transfer function $\Theta_l(k, x = 0)$, which is a representation of the CMB temperature field. We investigate how varying the cosmological parameters Ω_b, Ω_m, Ω_r, the Hubble parameter h, and the spectral index n, affects the behavior of the spectrum. From this we are able to find a set of values for the parameters that provide a good fit when compared to the CMB power spectrum obtained by the Planck Collaboration et al. (2016). These values are $\Omega_b = 0.063$, $\Omega_m = 0.200$, $\Omega_r = 8.22 \cdot 10^{-5}$, $h = 0.80$, and $n = 0.65$. These parameters are different than those expected by previous observations. We believe this to be due to us not including neutrinos or polarization into our simulation, nor any elements beyond hydrogen.

1. Introduction

In this final project, we will combine everything we have done in Milestone I, II, and III, to finally compute the cosmic microwave background (CMB) power spectrum. As before, we follow the formula laid out by Callin (2006).

2. Method

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

2.1. Theory

We now wish to compute the CMB power spectrum, combining everything from previous projects. As always, we express time with the parameter

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

where a is the scale factor of the universe, and t is the time in seconds. The derivative with respect to x of some parameter f, is denoted $f' = \frac{\mathrm{d}f}{\mathrm{d}x}$.

The first thing we need to compute, is the transfer function, that is, the multipoles Θ_l today,

$$\Theta_l(k, x = 0) = \int_{-\infty}^{0} \tilde{S}(k, x) j_l[k(\eta_0 - \eta)]\mathrm{d}x, \tag{2}$$

where $j_l(x)$ is the spherical Bessel functions, k is the wavenumber of the perturbations as explained in Milestone 3, and η is the conformal time. The source function $\tilde{S}$ is defined as

$$\begin{aligned}\tilde{S}(k, x) = \tilde{g}\left[\Theta_0 + \Psi + \frac{1}{4}\Pi\right] + e^{-\tau}[\Psi' - \Phi'] - \frac{1}{ck}\frac{\mathrm{d}}{\mathrm{d}x}(\mathcal{H}\tilde{g}v_b) \\ + \frac{3}{4c^2k^2}\frac{\mathrm{d}}{\mathrm{d}x}\left[\mathcal{H}\frac{\mathrm{d}}{\mathrm{d}x}(\mathcal{H}\tilde{g}\Pi)\right],\end{aligned} \tag{3}$$

where, $\tilde{g}$ is the visibility function, Θ_0 is the monopole, Ψ and Φ is the gravitational potential as defined in Milestone 3. c is the speed of light, v_b is the baryonic perturbation velocity, and $\mathcal{H} = aH$, where H is the Hubble parameter. Here, $\Pi = \Theta_2 + \Theta_0^P + \Theta_2^P$, which, in the case of no polarization or neutrinos, becomes simply $\Pi = \Theta_0$. The full differentiation of the final derivative in Equation 3, can be seen in section IV A in Callin (2006).

Once this is done, we can compute the CMB power spectrum given by,

$$C_l = \int \frac{\mathrm{d}^3k}{(2\pi)^3} P(k)\Theta_l^2(k), \tag{4}$$

where C_l, the power spectrum, is the variance of the a_{lm}'s, which are the space-time dependent coefficients of the spherically transformed CMB temperature field,

$$T(\hat{n}) = \sum_{lm} a_{lm} Y_{lm}(\hat{n}). \tag{5}$$

Here, $\hat{n}$ is the direction on the sky and Y_{lm} are the spherical harmonics. We have already described the temperature field $T(\hat{n})$ by the multipoles Θ_l, in Milestone 3.

We then have that

$$C_l = \langle |a_{lm}|^2 \rangle = \langle a_{lm} a_{lm}* \rangle, \tag{6}$$

where we have assumed rotational invariance, and simply averaged over m. As most inflation models predicts a Harrison-Zel'dovich spectrum, we can use that

$$\frac{k^3}{2\pi^2}P(k) = \left(\frac{ck}{H_0}\right)^{n-1}, \tag{7}$$

where n is the spectral index of scalar perturbations, and $P(k)$ is the primordial power spectrum. This gives us the final expression for the CMB power spectrum:

$$C_l = \int_0^\infty \left(\frac{ck}{H_0}\right)^{n_s-1} \Theta_l^2(k)\frac{\mathrm{d}k}{k}. \tag{8}$$

We want our spectrum to match with observations done by the Planck satellite (Planck Collaboration et al. 2016), we therefore normalize the spectrum so that its maximum value is $5775\mu\mathrm{K}^2$.

2.2. Implementation

The first thing we do is compute the source function $\tilde{S}(k, x)$ on the k and x grid already made in Milestone 3. We then spline the source function over a high resolution k and x grid,

$$x_i = x_{\text{init}} + (i-1)\frac{x_0 - x_{\text{init}}}{n_x - 1} \tag{9}$$

$$k_j = k_{\min} + (j-1)\frac{k_{\max} - k_{\min}}{n_k - 1}, \tag{10}$$

where both x and k have $n_x = n_k = 5000$ grid-points. From Milestone III, we have that $x_{\text{init}} = \log\left(10^{-8}\right)$, $x_0 = 0$, $k_{\max} = 1000H_0/c$, and $k_{\min} = 0.1H_0/c$.

To compute the transfer function Θ_l, we need to compute the spherical Bessel functions j_l for a combination of k and $\eta(x)$ values. These values lie in the range [0,3400](Callin 2006), and so we take a linear sample of 5400 points, and spline the resulting array. The Bessel functions are independent of cosmology, and so we save them to a file, so that we do not have to compute them again.

Next, we integrate over x to find the transfer function for each k and l. After we have found Θ_l, we integrate over k to find C_l for each l, as described in Equation 8. We use the simple method of evaluating the integrand at each x (Θ_l) or k (C_l) value, and adding them together. For the l values, we choose 44 values from $l = [2, 1200]$, and then we later spline the result to get a value of C_l for every l in this range. We make a high resolution l grid for this purpose, which has 1200 grid-points and starts at $l = 1$.

3. Results

The following results are obtained by having the following values for the cosmological parameters:

$$\Omega_m = 0.224, \qquad \Omega_b = 0.046, \qquad \Omega_r = 8.3 \cdot 10^{-5}, \qquad h = 0.7, \qquad n = 0.8. \tag{11}$$

These are referred to as the default parameters.

3.1. Intermediate results

We plot some intermediate results of our code, to see that it runs properly. In Figure 1, we can see the integrand of the $\Theta_l(k, x = 0)$ functions. That is, the source function $S(k, x)$ times the spherical Bessel functions $j_l(k)$ for $k = 340H_0/c$. We see that it has a large spike around recombination, and the Bessel functions makes the integrand oscillate at large k values. If we compare this with Figure 3 in Callin (2006), there are certain differences, i.e, our plot has a much larger dip around $x \approx -7$, and there also seem to be some smaller peaks at large k in our plot. These peaks appear to be due to some numerical imprecision, as rewriting the expressions for Ψ' and Φ' used in Milestone 3, made them smaller. Nevertheless, the overall shape of the function looks correct, and so we proceed in our calculations.

Next we look at the $\Theta_l(k, 0)$ functions for a selected number of l values. This can be seen in Figure 2. Here we can quite clearly see the oscillations for all l values.

In Figure 3, we have plotted the integrand of the C_l expression, ignoring the n-exponential term, multiplied with $l(l+1)$ to make the peaks more apparent. As this integrand contain the term Θ_l^2, it is, unsurprisingly similar in shape as those shown in 2. We see that this is also in accordance with the results of Callin (2006).

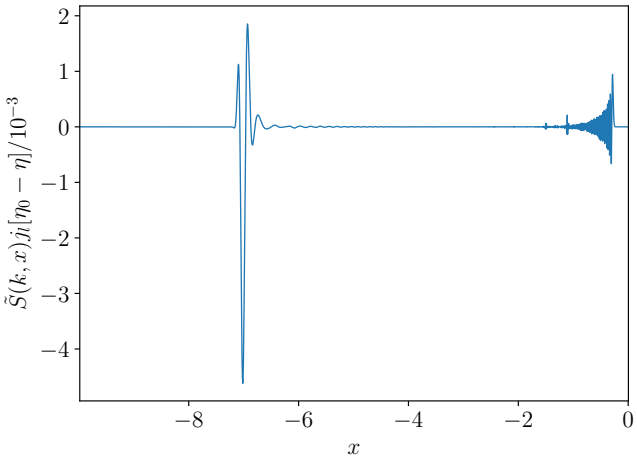

Fig. 1: The $\Theta_l(k, x = 0)$ integrand is shown against kc/H_0. The shape is similar to that obtained by Callin (2006), showing a peak near recombination at $x = -7$, and oscillations at high kc/H_0 values.

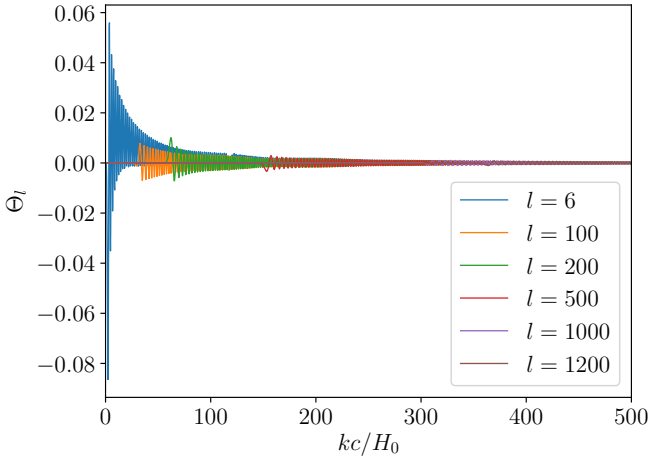

Fig. 2: The multipole moments plotted against kc/H_0 for various angular scales l.

Finally, we find the CMB angular power spectrum, which can be seen in Figure 4 against l. It has been normalized to have a maximum value of $5775\mu\text{K}^2$, to better compare it with the power spectrum obtained by the Planck Collaboration et al. (2016), which is plotted alongside it. We see that while the overall shape is the same, our simulated spectrum does not completely fit the Planck data. We will attempt to amend this in the next section.

A strange thing our spectrum does, is how, before the first major peak, it seems to increase with lower l. The reason for this is unknown, but as we will see in the next section, certain parameter choices make this effect worse. As our code turned out to be incredibly slow to run, and the spectrum looks fine otherwise, we elect to not invest anymore time into finding the error, and continue on.

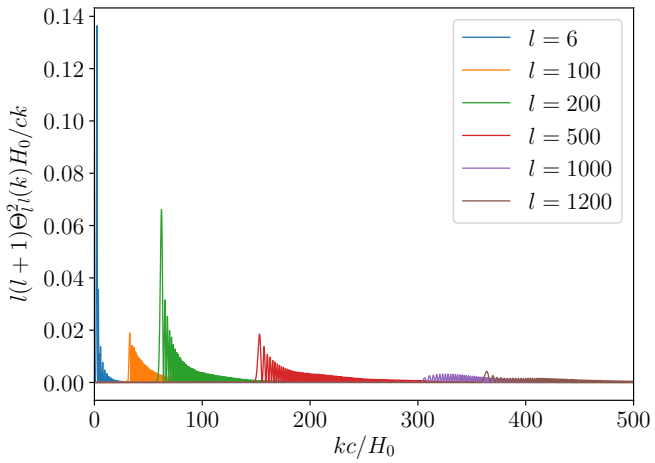

Fig. 3: The integrand of the expression for C_l, from Equation 8, against kc/H_0 for various l. The integrand is multiplied by $l(l+1)$ to make the shapes of the curves more apparent.

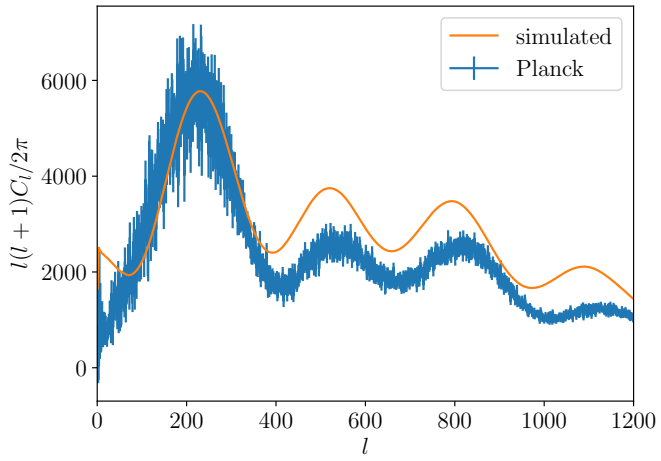

Fig. 4: The computed angular power spectrum of the CMB is seen with an orange line, against the angular scales l. Plotted alongside our simulated spectrum, is that obtained by the Planck Collaboration et al. (2016)

3.2. Parameter fitting

Now we try to make the power spectrum in 4 fit better with the observations. We do this by varying some of the cosmological parameters one at a time. The parameters we will be changing, are the ones mentioned in Equation 11. The result can be seen in Figure 5. As before, all spectra have been normalized to have a maximum value of $5775\mu\text{K}^2$.

We start with the h plot. For the changes we made in h, very little change was obtained in the spectrum. Still, we observe that lowering h, resulted in the peaks being lowered, and shifted slightly to the left. This is barely seen in the first and second peak, but is more pronounced in the third and fourth peak. Increasing h has the opposite effect, increasing the height of the peaks.

Next we look at the changing of n. If we decrease n, the entire spectrum is lowered, the peaks grow smaller in height, and become more narrow. If we increase n, the entire spectrum is heightened, and the peaks also becomes broader and taller. This is not seen so well in the first peak, but can easily be seen in the next three peaks.

Table 1: The parameter values that provide the best fitted model is shown below.

Ω_b	Ω_m	Ω_r	n	h
0.063	0.200	$8.22 \cdot 10^{-5}$	0.80	0.65

In the Ω_m plot, when we increase Ω_m, the second, third, and fourth peak increases in value, and is also shifted to the right. On the other hand, if we decrease Ω_m to much, the spectrum looks very strange, gaining new peaks and a brand new overall shape. The reason for this is unknown, but it seems unwise to lower this particular parameter that much.

When varying Ω_b, we get the following effect: Decreasing the Ω_b value will make all but the first peak increase in height, and shift them to the left. Decreasing Ω_b does the opposite, lowering the peaks, and shifting them to the right. These changes seem to affect the second and third peaks the most.

Finally, we look at varying the Ω_r parameter. Decreasing its value results in shifting the spectrum to the left, but it also decreases the size of the second and third peak. Increasing its value to much makes some of the peaks go away, leaving only two.

3.3. Best fit

From these parameter variations, we are able to find some values that provide a good fit with the Planck data (Planck Collaboration et al. 2016). This is done through trial and error, although a Metropolis algorithm should be used for more accurate results. The two power spectra can be seen in Figure 6. We see that our best-fit model fits just fine within the error bars of the Planck data, not considering the weird bit before the first peak. It also fits a little less well for small scales (large l's). The parameter values chosen can be seen in Table 1.

We can tell from Table 1, that the parameter values we have arrived at, are quite different from those suggested by observations and experiments, that is, the default parameters. The reason for this can be that we have not accounted for neutrinos, polarization, or other elements beyond hydrogen, such as helium.

4. Conclusions

We are now done with the final project, having followed in Callin (2006)'s footsteps.

We were able to compute the CMB angular power spectrum, C_l, through integration of the transfer function $\Theta_l(k, x = 0)$. We tested how varying the parameters Ω_b, Ω_m, Ω_r, the Hubble parameter h, and the spectral index n, affected the spectrum. From this we were able to find a set of values for the parameters, that provided a good fit with the CMB power spectrum obtained by the Planck Collaboration et al. (2016). These values were $\Omega_b = 0.063$, $\Omega_m = 0.200$, $\Omega_r = 8.22 \cdot 10^{-5}$, $h = 0.80$, and $n = 0.65$.

These parameters are different than those expected by observations, namely $\Omega_m = 0.224$, $\Omega_b = 0.046$, $\Omega_r = 8.3 \cdot 10^{-5}$,

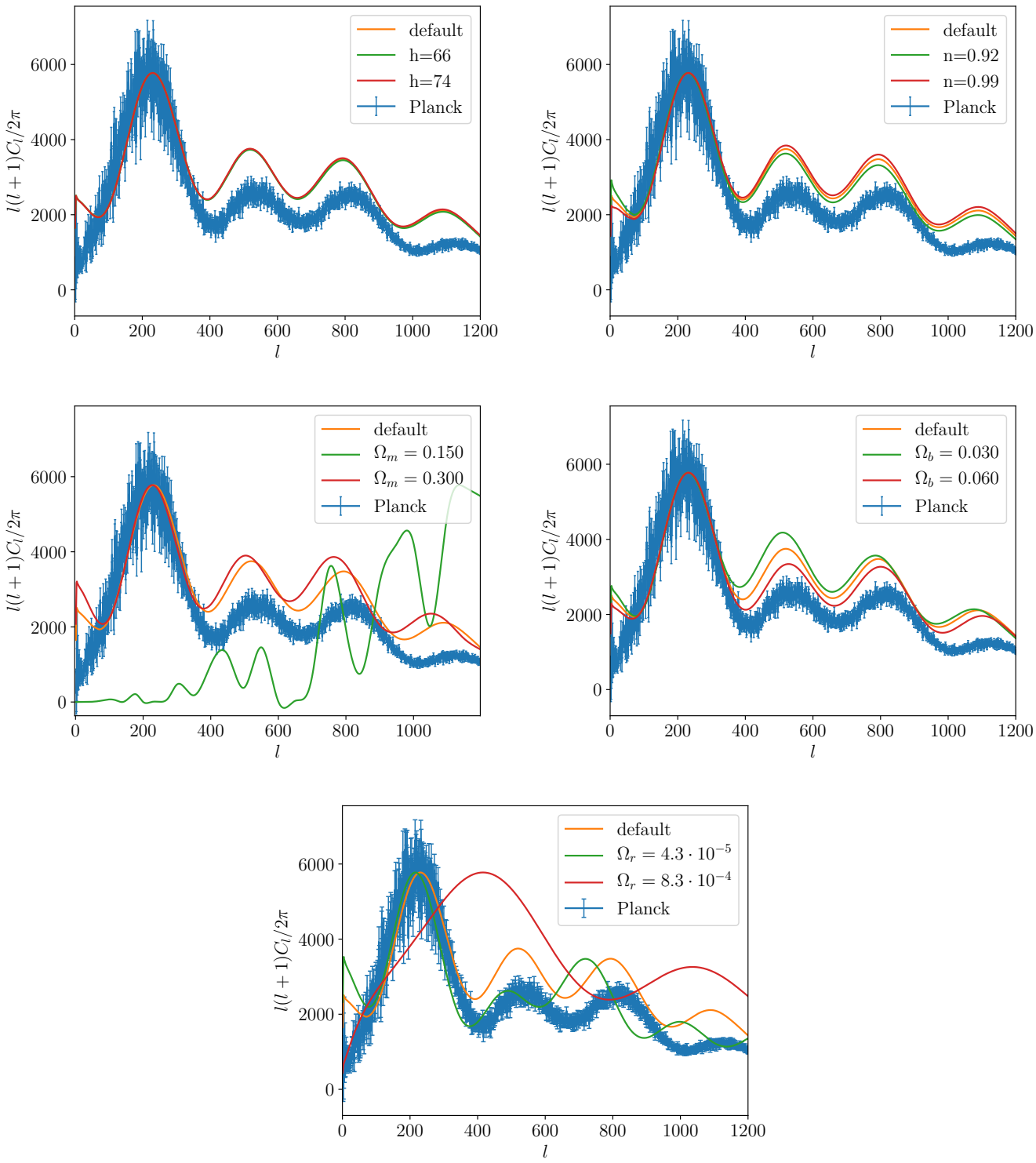

Fig. 5: The Power spectrum when we vary some cosmological parameters. The blue graph with error bars is the Planck data (Planck Collaboration et al. 2016), while the orange line named "default" is the power spectrum made by our program using the default parameters. The other parameters are altered as stated in the legend. In the top-left frame, we change only the dimensionless Hubble parameter h, and in the top-right frame, we change the spectral index n. In the second-right frame, we have changed Ω_m, in the second-left frame we have changed Ω_b, while in the bottom frame, we have altered Ω_r.

$h = 0.7$, $n = 0.8$. We suspect this is due to us not having included neutrinos or polarization into our simulation, nor any elements beyond hydrogen, and so adding this, the spectrum could better fit with previous estimates (Planck Collaboration et al. 2016).

References

Callin, P. 2006, ArXiv Astrophysics e-prints
Planck Collaboration, Ade, P. A. R., Aghanim, N., et al. 2016, A&A, 594, A20

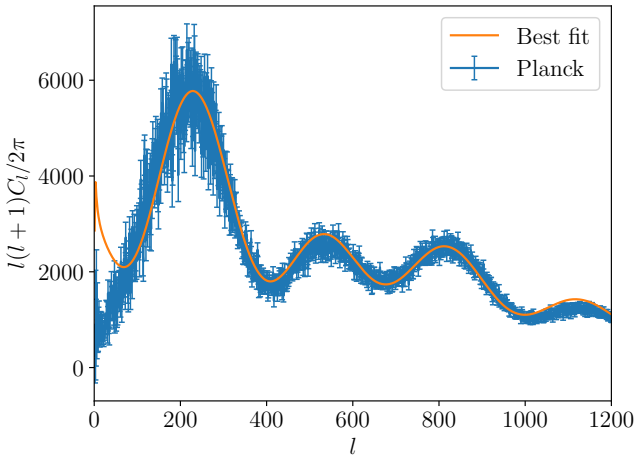

Fig. 6: The normalized CMB angular power spectrum that best fit the Planck data, is shown together with the Planck data, against the angular scales l.

5. Appendix

Source code: `evolution_mod.f90`

```
module evolution_mod
  use healpix_types
  use params
  use time_mod
  use ode_solver
  use rec_mod
  use spline_2D_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

  ! Hires source function variables
  integer(i4b), parameter       :: n_k_hires = 5000
  integer(i4b), parameter       :: n_x_hires = 5000

contains


  ! NB!!! New routine for 4th milestone only; disregard until then!!!
  subroutine get_hires_source_function(k, x, S)
    implicit none
    real(dp), allocatable, dimension(:), intent(out) :: k, x
    real(dp), allocatable, 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, ck, x_0,ckH_p
    real(dp), allocatable, dimension(:,:) :: S_lores
    real(dp), allocatable, dimension(:,:,:,:) :: S_coeff

     write(*,*) 'entered get_hires_source_function'
    ! 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.
    allocate(S_lores(n_t,n_k))
    allocate(S_coeff(4,4,n_t,n_k))
    allocate(x(n_x_hires))
    allocate(k(n_k_hires))
    allocate(S(n_x_hires,n_k_hires))
```

```
  write(*,*) '  making x and k grids'
  ! Make grids
  x_0 = 0.d0
  do i=1, n_x_hires
    x(i) = x_init + (x_0 - x_init)*(i-1.d0)/(n_x_hires-1.d0)
    k(i) = k_min + (k_max-k_min)*(i-1.d0)/(n_k_hires-1.d0) ! not square
  end do

  ! Substeps:
  !   1) First compute the source function over the existing k and x
  !      grids
  write(*,*) '  computing source function over k an x'
  do i=1,n_t
      g    = get_g(x_t(i))
      dg   = get_dg(x_t(i))
      ddg  = get_ddg(x_t(i))
      tau  = get_tau(x_t(i))
      dt   = get_dtau(x_t(i))
      ddt  = get_ddtau(x_t(i))
      H_p  = get_H_p(x_t(i))
      dH_p = get_dH_p(x_t(i))

      do j=1,n_k
          ck = c*ks(j)
          ckH_p = ck/H_p
          Pi   = Theta(i,2,j)
          dPi  = dTheta(i,2,j)

          ddPi = 2.d0/5.d0*ckH_p*(-dH_p/H_p*Theta(i,1,j) + dTheta(i,1,j)) &
               + 0.3d0*(ddt*Pi + dt*dPi)&
               - 3.d0/5.d0*ckH_p*(-dH_p/H_p*Theta(i,3,j)+dTheta(i,3,j))

          ddHH_p= H_0**2/2.d0*((Omega_b+Omega_m)/a_t(i) &
                       + 4.d0*Omega_r/a_t(i)**2 + 4.d0*Omega_lambda*a_t(i)**2)

          S_lores(i,j) = g*(Theta(i,0,j) + Psi(i,j) + 0.25d0*Pi)&
                       + exp(-tau)*(dPsi(i,j) - dPhi(i,j)) &
                       - 1.d0/ck*(dH_p*g*v_b(i,j) + H_p*(dg*v_b(i,j) + g*dv_b(i,j)))&
                       + 0.75d0/ck**2*(ddHH_p*g*Pi &
                       + 3.d0*H_p*dH_p*(dg*Pi + g*dPi) + H_p**2*(ddg*Pi + 2.d0*dg*dPi + g*ddPi))
      end do

  end do

  !   2) Then spline this function with a 2D spline
  write(*,*) '  splining source function'
  call splie2_full_precomp(x_t, ks, S_lores, S_coeff)
  !   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
  write(*,*) '  resample source function over high res grid'
  do i=1, n_x_hires
    do j=1, n_k_hires
      S(j,i) = splin2_full_precomp(x_t, ks, S_coeff, x(i), k(j))
    end do
  end do


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(n_t, 0:lmax_int, n_k))
  allocate(delta(n_t, n_k))
  allocate(delta_b(n_t, n_k))
  allocate(v(n_t, n_k))
  allocate(v_b(n_t, n_k))
  allocate(Phi(n_t, n_k))
  allocate(Psi(n_t, n_k))
  allocate(dPhi(n_t, n_k))
  allocate(dPsi(n_t, n_k))
  allocate(dv_b(n_t, n_k))
  allocate(dTheta(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(1,:)   = 1.d0
  delta(1,:) = 1.5d0 * Phi(1,:)
  delta_b(1,:) = delta(1,:)
  Theta(1,0,:) = 0.5d0*Phi(1,:)
  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(1,i)     = ckH_p/2.d0*Phi(1,i)
     v_b(1,i)   = v(1,i)

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

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)**2.d0/exp(x_t(i))*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)**2.d0/exp(x_t(i))
       *Omega_r*(-2.d0*Theta(i_tc,2,k)+dTheta(i_tc,2,k))

   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

        !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)**2.d0/exp(x_t(i))*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)**2.d0/exp(x_t(i)) *
           Omega_r*(-2.d0*Theta(i,2,k)+dTheta(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)

     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
  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')
    open (unit=10, file = 'dPhi.dat', status='replace')
    open (unit=11, file = 'dPsi.dat', status='replace')

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

    write(*,*) "writing stuff"
    do i=1, n_t
      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))
      write (10,'(*(2X, ES14.6E3))')
          dPhi(i,k(1)),dPhi(i,k(2)),dPhi(i,k(3)),dPhi(i,k(4)),dPhi(i,k(5)),dPhi(i,k(6))
      write (11,'(*(2X, ES14.6E3))') dPsi(i,k(1)),dPsi(i,k(2)),dPsi(i,k(3)),dPsi(i,k(4)),dPsi(i,k(5)),
          dPsi(i,k(6))
    end do

    write(*,*) "closing files "
    do i=0, 11
      close(i)
    end do

  end subroutine write_to_file_evolution_mod

end module evolution_mod
```

cl_mod.f90

```
module cl_mod
  use healpix_types
  use evolution_mod
  use sphbess_mod
  use spline_1D_mod
  implicit none

  real(dp),   allocatable, dimension(:,:) :: S
  integer(i4b), allocatable, dimension(:) :: ls
  real(dp),   allocatable, dimension(:)  :: ls_dp, ls_hires, cls_hires
  real(dp),   allocatable, dimension(:)  :: x_hires, k_hires, integrand1
  real(dp),   allocatable, dimension(:,:) :: Theta_l
  real(dp),   allocatable, dimension(:,:) :: integrand2

contains

  ! Driver routine for (finally!) computing the CMB power spectrum
  subroutine compute_cls
    implicit none

    integer(i4b) :: i, j, l, l_num, n_spline, j_loc

    real(dp),   allocatable,  dimension(:,:)  :: j_l, j_l2
    real(dp),   allocatable,  dimension(:)    :: cls, cls2
```

```
real(dp),   allocatable,  dimension(:)     :: k, x
real(dp),   allocatable,  dimension(:,:,:,:) :: S_coeff

real(dp),   allocatable, dimension(:)  :: z_spline, j_l_spline, j_l_spline2
real(dp),   allocatable, dimension(:)  :: besseltest
real(dp),   allocatable, dimension(:)  :: eta_arr

real(dp)        :: integral1, integral2, h1, h2, eta0

logical(lgt)    :: exist
character(len=128) :: filename1,filename
real(dp), allocatable, dimension(:) :: y, y2
real(dp) :: start_time, end_time

! Set up which l's to compute
l_num = 44
allocate(ls(l_num))
ls = (/ 2, 3, 4, 6, 8, 10, 12, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100, &
    & 120, 140, 160, 180, 200, 225, 250, 275, 300, 350, 400, 450, 500, 550, &
    & 600, 650, 700, 750, 800, 850, 900, 950, 1000, 1050, 1100, 1150, 1200 /)

! Task: Get source function from evolution_mod
allocate(S(n_k_hires,n_x_hires))
allocate(x_hires(n_x_hires))
allocate(k_hires(n_k_hires))

filename1 = 'source.bin'
inquire(file=filename1, exist=exist)
if (exist) then
   write(*,*) 'reading Source function from file'
   open(0, form='unformatted', file=filename1)
   read(0) x_hires, k_hires, S
   close(0)
else
    write(*,*) "initializing evolution module"
    call initialize_perturbation_eqns
    call integrate_perturbation_eqns

    call get_hires_source_function(k_hires, x_hires, S)
    write(*,*) 'made the call to evol_mod'
    open(0, form='unformatted', file=filename1)
    write(0) x_hires, k_hires, S
    close(0)
end if

! Task: Initialize spherical Bessel functions for each l; use 5400 sampled points between
!      z = 0 and 3500. Each function must be properly splined
! Hint: It may be useful for speed to store the splined objects on disk in an unformatted
!      Fortran (= binary) file, so that these only has to be computed once. Then, if your
!      cache file exists, read from that; if not, generate the j_l's on the fly.
n_spline = 5400
allocate(z_spline(n_spline)) ! Note: z is *not* redshift, but simply the dummy argument of j_l(z)
allocate(j_l(n_spline, l_num))
allocate(j_l2(n_spline, l_num))

write(*,*) 'making z_spline'
do i=1,n_spline
  z_spline(i) = 0.d0 + (i-1)*(3400.d0-0.d0)/(n_spline-1.d0)
end do

! checking for binary file
filename = 'j_l.bin'
inquire(file=filename, exist=exist)
if (exist) then
   write(*,*) 'reading Bessel functions from file'
   open(1, form='unformatted', file=filename)
   read(1) j_l, j_l2
   close(1)
else
  write(*,*) "compute spherical Bessel functions"
```

```
  do l=1, l_num
    j_l(1,l) = 0.d0
    do i=2, n_spline
      call sphbes(ls(l),z_spline(i), j_l(i,l))
    end do
  end do

  ! spline bessel functions across z for all l
  write(*,*) "spline bessel functions"
  do l=1,l_num
    call spline(z_spline, j_l(:,l), 1.0d30, 1.0d30, j_l2(:,l))
  end do

  ! Write to file
  open(1, form='unformatted', file=filename)
  write(1) j_l, j_l2
  close(1)
end if

j_loc = locate_dp(k_hires,340.d0*H_0/c)

allocate(besseltest(n_x_hires))
open (unit=3 ,file="besseltest.dat",action="write",status="replace")
do i =1,n_x_hires
  besseltest(i) = splint(z_spline,j_l(:,17),j_l2(:,17),k_hires(j_loc)*(get_eta(0.d0)-get_eta(x_hires(i))))
  write (3 ,*) besseltest(i)
end do
close (3)
!stop

allocate(Theta_l(l_num,n_k_hires))
allocate(integrand1(n_x_hires))
allocate(integrand2(l_num, n_k_hires))
allocate(cls(l_num))
allocate(cls2(l_num))

!allocate(x_lores(n_x_hires/10)) ! creating low-res x grid for fast Theta_l-integration

! Overall task: Compute the C_l's for each given l
! Precompute eta0-eta(i)
allocate(eta_arr(n_x_hires))
eta0 = get_eta(0.d0)
do i=1, n_x_hires
  eta_arr(i) = eta0-get_eta(x_hires(i))
end do

! For integration
h1 = (x_hires(n_x_hires) - x_hires(1))/n_x_hires
h2 = (k_hires(n_k_hires) - k_hires(1))/n_k_hires

do l = 1, l_num
  write(*,*) 'l=', l
  ! Task: Compute the transfer function, Theta_l(k)
  integral2 = 0.d0          ! Reset C_l integration
  write(*,*) 'integration for theta_l'
  !Start timer
  call cpu_time(start_time)

  do j=1, n_k_hires

    integral1= 0.d0          ! Reset Theta integration
    do i=1, n_x_hires
      integrand1(i) = S(j,i)*splint(z_spline,j_l(:,l),j_l2(:,l), k_hires(j)*eta_arr(i))
      integral1 = integral1 + integrand1(i)
    end do
    Theta_l(l,j) = h1*integral1 !-0.5d0*(integrand1(1)+integrand1(n_x_hires)))

    if(l==17 .and. j==j_loc .and. n_s==0.96) then ! Save l=100 and k=340 compare with Callin
      write(*,*)'writing integrand to file for l=17,k=',j_loc
      open (unit=2 ,file="Sj_l.dat",action="write",status="replace")
```

```
            do i=1,n_x_hires
              write (2 ,*) integrand1(i)
            end do
            close (2)
           !stop
         end if

        ! Task: Integrate P(k) * (Theta_l^2 / k) over k to find un-normalized C_l's
         integrand2(l,j) = (c*k_hires(j)/H_0)**(n_s-1.d0)*Theta_l(l,j)**2/k_hires(j)
         integral2 = integral2 + integrand2(l,j)
       end do
       write(*,*) 'integration for cl'
       integral2 = h2*integral2! - 0.5d0*(integrand2(l,1)+integrand2(l,n_k_hires)))

       ! Task: Store C_l in an array. Optionally output to file
       cls(l) = integral2*ls(l)*(ls(l)+1.d0)/(2.d0*pi)
       !Print time used
       call cpu_time(end_time)

       print'("Time used = ",f7.2," seconds.")',end_time-start_time
       write(*,*) 'a11'

     end do


     ! Task: Spline C_l's found above, and output smooth C_l curve for each integer l
     allocate(ls_dp(l_num))
     allocate(ls_hires(int(maxval(ls))))
     allocate(cls_hires(int(maxval(ls))))

     do l=1, l_num ! spline requires double precision
        ls_dp(l) = ls(l)
     end do

     write(*,*) 'splining cls'
     call spline(ls_dp, cls, 1.d30, 1.d30, cls2)

     ! new unit stepsize l-grid
     write(*,*) 'making new highresolution l-grid'
     do l=1, int(maxval(ls))
       ls_hires(l) = l
     end do

     ! find Cls for all ls_hires
     write(*,*) 'saving splined cls'
     do l=1, int(maxval(ls))
       cls_hires(l) = splint(ls_dp, cls, cls2, ls_hires(l))
     end do

   !call write_to_file_cl_mod
   end subroutine compute_cls

   subroutine write_to_file_cl_mod
     use healpix_types
     implicit none

     integer(i4b) :: i,j
     integer(i4b), dimension(6) :: l_val,l
     integer(i4b), dimension(44) :: ll

     ! Finds index of chosen l values
     l_val(1:6)=(/6, 100, 200, 500, 1000, 1200/)
     do j=1, 6
        forall (i=1:44) ll(i) = abs(ls(i)-l_val(j))
        l(j) = minloc(ll,1)
     end do
     write(*,*) "writing to file; cl_mod"

!-------------- write to file ---------------
     write(*,*) "opening files "
```

```
    open (unit=1, file = 'l_val_ns099.dat', status='replace')
    open (unit=2, file = 'x_k_hires_ns099.dat', status='replace')
    open (unit=3, file = 'Theta_l_ns099.dat', status='replace')
    open (unit=4, file = 'C_l_integrand_ns099.dat', status='replace')
    open (unit=5, file = 'C_l.dat', status='replace')

    write(*,*) "writing stuff"
    write(*,*) '  writing chosen k values'
    do i=1, 6 ! write the k values used
      write(1, *) l_val(i)
    end do

    write(*,*) '  writing x, k, source func., Theta_intl100, C_l100_int,Theta_l, C_l_int'
    do i=1, n_x_hires
      write (2,'(*(2X, ES14.6E3))') x_hires(i), k_hires(i)
      write (3,'(*(2X, ES14.6E3))')&
        Theta_l(l(1), i),Theta_l(l(2), i),Theta_l(l(3), i),Theta_l(l(4),i ),Theta_l(l(5), i),Theta_l(l(6), i)
      write (4,'(*(2X, ES14.6E3))')&
        integrand2(l(1), i)/(c*k_hires(i)/H_0)**(n_s-1.d0),&
        integrand2(l(2), i)/(c*k_hires(i)/H_0)**(n_s-1.d0),&
        integrand2(l(3), i)/(c*k_hires(i)/H_0)**(n_s-1.d0),&
        integrand2(l(4), i)/(c*k_hires(i)/H_0)**(n_s-1.d0),&
        integrand2(l(5), i)/(c*k_hires(i)/H_0)**(n_s-1.d0),&
        integrand2(l(6), i)/(c*k_hires(i)/H_0)**(n_s-1.d0)
    end do

    write(*,*) '  writing l_hires and c_l_hires'
    do i=1,1200
      write (5,'(*(2X, ES14.6E3))') ls_hires(i), cls_hires(i)
    end do

    write(*,*) 'closing files'
    do i=1, 5
      close(i)
    end do

  end subroutine write_to_file_cl_mod
end module cl_mod
```