# Milestone 4 Code

Daniel Herman

April 2018

## 1 Code

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

  real(dp),     pointer,     dimension(:)       :: x_hires, k_hires
    , l_hires, cl_hires
  real(dp),     pointer,     dimension(:,:)     :: S, S2, j_l, j_l2
  real(dp),     allocatable, dimension(:)       :: bessel,z_spline
  real(dp),     allocatable, dimension(:,:)     :: Theta_l,integ2,
    integ
  integer(i4b), allocatable, dimension(:)       :: ls

contains

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

    integer(i4b) :: i, j, l, l_num, x_num, n_spline
    real(dp)     :: dx, S_func, j_func, z, eta, eta0, x0, x_min,
    x_max, d, e
    real(dp)     :: k_min, k_max, dk, l_max
    real(dp),     allocatable, dimension(:)       :: integrand,
    int_test1,int_test2
    real(dp),     allocatable, dimension(:)       :: int_test3,
    int_test4,int_test5,int_test6
    real(dp),     allocatable, dimension(:)       :: int2_test1,
    int2_test2,int2_test3
    real(dp),     allocatable, dimension(:)       :: int2_test4,
    int2_test5,int2_test6
    real(dp),     pointer,     dimension(:)       :: x_arg, cls,
    cls2, ls_dp
    real(dp),     pointer,     dimension(:)       :: k, x, cl_int
    real(dp),     pointer,     dimension(:,:,:,:) :: S_coeff
    real(dp),     allocatable, dimension(:)       :: j_l_spline,
    j_l_spline2

    real(dp)            :: t1, t2, integral
    logical(lgt)        :: exist
```

```fortran
    character(len=128) :: filename
    real(dp), allocatable, dimension(:) :: y, y2

    n_spline = 5400
    l_num = 44

    ! Allocate all necessary arrays for C_l estimation
    allocate(S(n_x_hires,n_k_hires))
    allocate(x_hires(n_x_hires),k_hires(n_k_hires))

    allocate(ls(l_num))
    allocate(z_spline(n_spline))    ! Note: z is *not* redshift,
    dummy argument of j_l(z)

    allocate(j_l(n_spline, l_num))
    allocate(j_l2(n_spline, l_num))

    allocate(Theta_l(l_num,n_k_hires))
    allocate(integ(l_num,n_x_hires))
    allocate(integ2(l_num,n_k_hires))

    allocate(cl_int(l_num))
    allocate(cls(l_num))
    allocate(cls2(l_num))
    allocate(ls_dp(l_num))

    allocate(int_test1(n_x_hires))
    allocate(int_test2(n_x_hires))
    allocate(int_test3(n_x_hires))
    allocate(int_test4(n_x_hires))
    allocate(int_test5(n_x_hires))
    allocate(int_test6(n_x_hires))
    allocate(int2_test1(n_k_hires))
    allocate(int2_test2(n_k_hires))
    allocate(int2_test3(n_k_hires))
    allocate(int2_test4(n_k_hires))
    allocate(int2_test5(n_k_hires))
    allocate(int2_test6(n_k_hires))


    ! Open C_l files to write to:
    !------------------------------
    open(10, file='thetal_squared1.dat')
    open(21, file='thetal_squared2.dat')
    open(22, file='thetal_squared3.dat')
    open(23, file='thetal_squared4.dat')
    open(24, file='thetal_squared5.dat')
    open(25, file='thetal_squared6.dat')
    open(11, file='transfer1.dat')
    open(12, file='transfer2.dat')
    open(43, file='transfer3.dat')
    open(14, file='transfer4.dat')
    open(15, file='transfer5.dat')
    open(16, file='transfer6.dat')
    open(17, file='hi_res_C_ls.dat')

    ! Set up which l's to compute
```

```
    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 /)

    ! Convert the l's to double precision, and create high-resolution l arrays
    do l=1,l_num
       ls_dp(l) = ls(l)
    end do

    l_max = int(maxval(ls))

    allocate(l_hires(int(l_max)))
    allocate(cl_hires(int(l_max)))

    do l=1,l_max
       l_hires(l) = l
    end do

    ! Task: Get source function from evolution_mod

    call get_hires_source_function(x_hires,k_hires,S)

    ! Initialize values for trapezoidal integration
    x_min = x_hires(1)
    x_max = x_hires(n_x_hires)
    dx    = (x_max-x_min)/n_x_hires
    k_min = k_hires(1)
    k_max = k_hires(n_k_hires)
    dk    = (k_max-k_min)/n_k_hires

    ! 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.

    do i=1,n_spline
       z_spline(i) = 0.d0 + (i-1)*3500.d0/(n_spline-1.d0)
    end do

    inquire(file='bessel.unf', exist=exist)
    if (exist) then
       write(*,*) 'Bessel file found.'
       open(13,file='bessel.unf', form='unformatted')
           read(13) j_l
       close(13)
    else
       write(*,*) 'Calculating Bessel function values.'
       do i=2,n_spline
```

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

       open(13,file='bessel.unf', form='unformatted')
           write(13) j_l
       close(13)
    end if

    do l=1,l_num
       call spline(z_spline,j_l(:,l),yp1,ypn,j_l2(:,l))
    end do

    !write(*,*) splint(z_spline,j_l(:,2), j_l2(:,2), 0.2*H_0/c*get_eta(0.d0))
    !write(*,*) get_eta(0.d0), 0.2*H_0/c*get_eta(0.d0)

    !| Overall task: Compute the C_l's for each given l
    !|---------------------------------------------------
    write(*,*) 'Computing C_ls!'

    do l = 1,l_num
       write(*,*) 'l = ',ls(l)

       ! Task: Compute the transfer function, Theta_l(k)
       ! We will compute the integral by using the trapezoidal method,
       ! summing over all x-values of the Bessel weighted Source Function
       do j=1,n_k_hires
          do i=1,n_x_hires
             integ(l,i) = S(i,j)*j_lfunc(l,k_hires(j),x_hires(i))
          end do
          do i=1,n_x_hires
             Theta_l(l,j) = Theta_l(l,j) + integ(l,i)
          end do
          Theta_l(l,j) = dx*Theta_l(l,j)
       ! Task: Integrate P(k) * (Theta_l^2 / k) over k to find un-normalized C_l's
       !---------------------------------------------------------------------------
          integ2(l,j) = (c*k_hires(j)/H_0)**(n_s-1.d0)*Theta_l(l,j)**(2.d0)/k_hires(j)
          cl_int(l) = cl_int(l) + integ2(l,j)
       end do

       ! Task: Store C_l in an array. Optionally output to file
       !---------------------------------------------------------
       cls(l) = dk*cl_int(l)*ls(l)*(ls(l)+1.d0)/(2.d0*pi)
       write(*,*) 'C_l = ', cls(l)
       if (cls(l) > 1.d0) then
          cls(l) = 4.d-2
       endif
       write(*,*) '-----------------------------------------'
```

```fortran
    end do

    ! Task: Spline C_l's found above, and output smooth C_l curve for each integer l

    ! Now spline the C_l's
    write(*,*) 'Splining C_ls'
    call spline(ls_dp,cls,yp1,ypn,cls2)
    write(*,*) 'C_l spline complete!'

    ! Create high-resolution C_l array
    do l=2,l_max
       cl_hires(l) = splint(ls_dp,cls,cls2,l_hires(l))
       ! write(*,*) 'l = ', l_hires(l), 'C_l = ', cl_hires(l)
       write(17,'(2(E17.8))') l_hires(l), cl_hires(l)
    end do

    close(12)

    do i=1,n_k_hires
       int2_test1(i) = (ls(5)*(ls(5)+1))*(Theta_l(5,i)**2.d0)*H_0/(c*k_hires(i))
       int2_test2(i) = (ls(10)*(ls(10)+1))*(Theta_l(10,i)**2.d0)*H_0/(c*k_hires(i))
       int2_test3(i) = (ls(17)*(ls(17)+1))*(Theta_l(17,i)**2.d0)*H_0/(c*k_hires(i))
       int2_test4(i) = (ls(25)*(ls(25)+1))*(Theta_l(25,i)**2.d0)*H_0/(c*k_hires(i))
       int2_test5(i) = (ls(35)*(ls(35)+1))*(Theta_l(35,i)**2.d0)*H_0/(c*k_hires(i))
       int2_test6(i) = (ls(44)*(ls(44)+1))*(Theta_l(44,i)**2.d0)*H_0/(c*k_hires(i))
    end do
    !|-----------------------------------------
    !| Integrand test for l=100, k = 159.988
    !|-----------------------------------------
    do i=1,n_k_hires
       int_test1(i) = (ls(5)*(ls(5)+1))*Theta_l(5,i)*H_0/(c*k_hires(i))
       int_test2(i) = (ls(10)*(ls(10)+1))*Theta_l(10,i)*H_0/(c*k_hires(i))
       int_test3(i) = (ls(17)*(ls(17)+1))*Theta_l(17,i)*H_0/(c*k_hires(i))
       int_test4(i) = (ls(25)*(ls(25)+1))*Theta_l(25,i)*H_0/(c*k_hires(i))
       int_test5(i) = (ls(35)*(ls(35)+1))*Theta_l(35,i)*H_0/(c*k_hires(i))
       int_test6(i) = (ls(44)*(ls(44)+1))*Theta_l(44,i)*H_0/(c*k_hires(i))
    end do

    !write(*,*) ls(5)
    !write(*,*) ls(10)
    !write(*,*) ls(17)
    !write(*,*) ls(25)
    !write(*,*) ls(35)
    !write(*,*) ls(44)
```

```fortran

    !Integrand testing apparatus:
!------------------------------------------------------------------------
                                                                 !|
    do j=1,n_k_hires                                             !|
       if (abs(int2_test1(j)) < 1.d-99) then                     !|
          int2_test1(j) = 0.d0                                    !|
       endif                                                     !|
       if (abs(int2_test2(j)) < 1.d-99) then                     !|
          int2_test2(j) = 0.d0                                    !|
       endif                                                     !|
       if (abs(int2_test3(j)) < 1.d-99) then                     !|
          int2_test3(j) = 0.d0                                    !|
       endif                                                     !|
       if (abs(int2_test4(j)) < 1.d-99) then                     !|
          int2_test4(j) = 0.d0                                    !|
       endif                                                     !|
       if (abs(int2_test5(j)) < 1.d-99) then                     !|
          int2_test5(j) = 0.d0                                    !|
       endif                                                     !|
       if (abs(int2_test6(j)) < 1.d-99) then                     !|
          int2_test6(j) = 0.d0                                    !|
       endif                                                     !|
       if (abs(int_test1(j)) < 1.d-99) then                      !|
          int_test1(j) = 0.d0                                    !|
       endif                                                     !|
       if (abs(int_test2(j)) < 1.d-99) then                      !|
          int_test2(j) = 0.d0                                    !|
       endif                                                     !|
       if (abs(int_test3(j)) < 1.d-99) then                      !|
          int_test3(j) = 0.d0                                    !|
       endif                                                     !|
       if (abs(int_test4(j)) < 1.d-99) then                      !|
          int_test4(j) = 0.d0                                    !|
       endif                                                     !|
       if (abs(int_test5(j)) < 1.d-99) then                      !|
          int_test5(j) = 0.d0                                    !|
       endif                                                     !|
       if (abs(int_test6(j)) < 1.d-99) then                      !|
          int_test6(j) = 0.d0                                    !|
       endif                                                     !|
       if (abs(S(j,2000)) < 1.d-99) then                         !|
          S(j,2000) = 0.d0                                       !|
       endif                                                     !|
       write(10,'(2(E18.7))') c*k_hires(j)/H_0, int2_test1(j)!|
       write(21,'(2(E18.7))') c*k_hires(j)/H_0, int2_test2(j)!|
       write(22,'(2(E18.7))') c*k_hires(j)/H_0, int2_test3(j)!|
       write(23,'(2(E18.7))') c*k_hires(j)/H_0, int2_test4(j)!|
       write(24,'(2(E18.7))') c*k_hires(j)/H_0, int2_test5(j)!|
       write(25,'(2(E18.7))') c*k_hires(j)/H_0, int2_test6(j)!|
       write(11,'(2(E18.7))') c*k_hires(j)/H_0, int_test1(j) !|
       write(12,'(2(E18.7))') c*k_hires(j)/H_0, int_test2(j) !|
       write(43,'(2(E18.7))') c*k_hires(j)/H_0, int_test3(j) !|
       write(14,'(2(E18.7))') c*k_hires(j)/H_0, int_test4(j) !|
       write(15,'(2(E18.7))') c*k_hires(j)/H_0, int_test5(j) !|
       write(16,'(2(E18.7))') c*k_hires(j)/H_0, int_test6(j) !|
    end do                                                       !|
```

```
                                                                             !|
    close(10)                                                                !|
    close(11)                                                                !|
    close(12)                                                                !|
    close(43)                                                                !|
    close(14)                                                                !|
    close(15)                                                                !|
    close(16)                                                                !|
    close(21)                                                                !|
    close(22)                                                                !|
    close(23)                                                                !|
    close(24)                                                                !|
    close(25)                                                                !|
!------------------------------------------------------------------------------

  end subroutine compute_cls


  function j_lfunc(l,k,x)
    implicit none
    integer(i4b),  intent(in) :: l
    real(dp),      intent(in) :: x,k
    real(dp)                  :: j_lfunc

    j_lfunc = splint(z_spline,j_l(:,l),j_l2(:,l),k*(get_eta(0.d0)-
    get_eta(x)))

  end function j_lfunc


end module cl_mod
```

```
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),                private :: x_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), allocatable, dimension(:)     :: prints
  real(dp), allocatable, dimension(:)     :: dydx

  ! Milestone 4 variables
  real(dp), allocatable, dimension(:,:,:,:) :: S_coeff
  real(dp), allocatable, dimension(:,:)     :: S_lores
  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(x_hires, k_hires, S)
    implicit none

    real(dp), pointer, dimension(:),   intent(out) :: x_hires, k_hires
    real(dp), pointer, dimension(:,:), intent(out) :: S
    real(dp), allocatable, dimension(:)            :: x_use
    integer(i4b) :: i, k, i2
    real(dp)     :: g, dg, ddg, tau, dt, ddt, H_p, dH_p, ddHH_p, Pi_c, dPi, ddPi
    real(dp)     :: xi, kk, ck

    ! 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.

    write(*,*) 'Initializing the source function.'

    allocate(x_hires(n_x_hires),k_hires(n_k_hires))
    allocate(S_lores(501,1:n_k))
    allocate(S_coeff(4,4,501,n_k))
    allocate(S(n_x_hires,n_k_hires))
    allocate(x_use(501))


    do i=1,501
       x_use(i) = x_t(i+999)
    end do

    do i=1,n_x_hires
       do k=1,n_k_hires
          x_hires(i) = x_use(1) + (0.d0-x_use(1))*(i-1.d0)/(n_x_hires-1.d0)
          k_hires(k) = k_min  + (k_max - k_min)*((k-1.d0)/(
```

```
n_k_hires -1.d0))
          end do
       end do

       ! Substeps:
       !   1) First compute the source function over the existing k and x
       !       grids
       do k=1,n_k
          kk = ks(k)
          ck = c*kk

          ! Only computing source function from the beginning of recombination
          do i=1,501
             i2    = i+999
             xi    = x_use(i)
             g     = get_g(xi)
             dg    = get_dg(xi)
             ddg   = get_ddg(xi)
             tau   = get_tau(xi)
             dt    = get_dtau(xi)
             ddt   = get_ddtau(xi)
             H_p   = get_H_p(xi)
             dH_p  = get_dH_p(xi)
             Pi_c  = Theta(i2,2,k)
             dPi   = dTheta(i2,2,k)
             ddPi  = 2.d0*ck/(5.d0*H_p)*(-dH_p/H_p*Theta(i2,1,k) + dTheta(i2,1,k)) &
                     + 0.3d0*(ddt*Pi_c+dt*dPi) &
                     - 3.d0*ck/(5.d0*H_p)*(-dH_p/H_p*Theta(i2,3,k) + dTheta(i2,3,k))

             !if (i == 100 .and. k==10) then
             !   write(*,*) 'need to compare!'
             !   write(*,*) g*(Theta(i2,0,k)+Psi(i2,k)+0.25d0*Pi_c)
             !endif

             S_lores(i,k) = g*(Theta(i2,0,k)+Psi(i2,k)+0.25d0*Pi_c)+exp(-tau)*(dPsi(i2,k)-dPhi(i2,k)) &
                            - 1.d0/ck*(H_p*(g*dv_b(i2,k)+v_b(i2,k)*dg)+g*v_b(i2,k)*dH_p) &
                            + 0.75d0/(ck**2)*((H_0**2/2.d0*((Omega_m+Omega_b)/exp(xi) &
                            + 4.d0*Omega_r/exp(2.d0*xi)+4.d0*Omega_lambda*exp(2.d0*xi)))&
                            * g*Pi_c+3.d0*H_p*dH_p*(dg*Pi_c+g*dPi) &
                            + H_p**2*(ddg*Pi_c+2.d0*dg*dPi+g*ddPi))
           end do
       end do

       write(*,*) '------------------------------'
       !write(*,*) S_lores(100,1), S_lores(100,5), S_lores(100,10)
       !write(*,*) get_g(x_use(100))*(Theta(1099,0,10)+Psi(1099,10)+0.25d0*Theta(1099,2,10))
       !write(*,*) get_g(x_use(100)), x_use(100)

```

```fortran
    !   2) Then spline this function with a 2D spline
    call splie2_full_precomp(x_use, 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

    do k=1,n_k_hires
       do i=1,n_x_hires
          S(i,k) = splin2_full_precomp(x_use,ks,S_coeff,x_hires(i), k_hires(k))
       end do
    end do

    write(*,*) 'Source funciton initialized.'
    write(*,*) '----------------------------------------'
  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, k
    x_init = log(a_init)

    write(*,*) '----------------------------------------'

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

    ! Allocate arrays for perturbation quantities
    allocate(Theta(1:n_t, 0:lmax_int, n_k))
    allocate(delta(1:n_t, n_k))
    allocate(delta_b(1:n_t, n_k))
    allocate(v(1:n_t, n_k))
    allocate(v_b(1:n_t, n_k))
    allocate(Phi(1:n_t, n_k))
    allocate(Psi(1:n_t, n_k))
    allocate(dPhi(1:n_t, n_k))
    allocate(dPsi(1:n_t, n_k))
    allocate(dv_b(1:n_t, n_k))
    allocate(dTheta(1:n_t, 0:lmax_int, n_k))

    Theta(:,:,:)   = 0.d0
    dTheta(:,:,:)  = 0.d0
    dPhi(:,:)      = 0.d0
    dPsi(:,:)      = 0.d0

    ! Task: Set up initial conditions for the Boltzmann and Einstein equations
```

```fortran
    Phi(1,:)      = 1.d0
    Psi(1,:)      = -Phi(1,:)
    delta(1,:)    = 1.5d0*Phi(1,:)
    delta_b(1,:) = delta(1,:)
    Theta(1,0,:) = 0.5d0*Phi(1,:)

    do i = 1, n_k
       v(1,i)        = c*ks(i)/(2*get_H_p(x_init))*Phi(1,i)
       v_b(1,i)      = v(1,i)
       Theta(1,1,i) = -c*ks(i)/(6*get_H_p(x_init))*Phi(1,i)
       Theta(1,2,i) = -20.d0*c*ks(i)/(45.d0*get_H_p(x_init)*get_dtau(x_init))*Theta(1,1,i)
       do l = 3, lmax_int
          Theta(1,l,i) = -l/(2.d0*l+1.d0)*c*ks(i)/(get_H_p(x_init)*get_dtau(x_init))*Theta(1,l-1,i)
       end do
    end do
    write(*,*) 'Perturbation Equations Initialized.'
    write(*,*) '-----------------------------------'
  end subroutine initialize_perturbation_eqns

  subroutine integrate_perturbation_eqns
    implicit none

    integer(i4b) :: i, j, k, l,j_tc
    real(dp)     :: x1, x2,H,ck,ckH,a,dtau,x,bleta
    real(dp)     :: eps, hmin, h1, x_tc, H_p, dt, t1, t2
    logical(lgt) :: exist

    real(dp), allocatable, dimension(:) :: y, y_tight_coupling
    real(dp), allocatable, dimension(:) :: x_temp,x_post,x_total

    eps    = 1.d-8
    hmin   = 0.d0
    h1     = 1.d-5

    allocate(y(npar))
    allocate(dydx(npar))
    allocate(y_tight_coupling(7))
    allocate(prints(6))

    prints(1)        = 1
    prints(2)        = 10
    prints(3)        = 30
    prints(4)        = 50
    prints(5)        = 80
    prints(6)        = 100

    y_tight_coupling = 0.d0
    y                = 0.d0
    dydx             = 0.d0

    ! Check to see if perturbation equation values are already stored
    ! If not, compute
    inquire(file='evo_dat.unf',exist=exist)
    if (exist) then
```

```

      write(*,*) 'Importing perturbation values...'
      open(29,file='evo_dat.unf',form='unformatted')
      read(29) Theta
      read(29) delta
      read(29) delta_b
      read(29) Phi
      read(29) Psi
      read(29) v
      read(29) v_b
      read(29) dPhi
      read(29) dPsi
      read(29) dv_b
      read(29) dTheta
      close(29)
      write(*,*) 'Successful import!'
   else

      open(26, file='derivs.dat')
      open(27, file='vandb.dat')
      open(28, file='phi_theta.dat')
      write(*,*) 'Computing perturbation values!'

      ! Propagate each k-mode independently
      do k = 1, n_k
         write(*,*) 'k =', k
         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
         x_tc = get_tight_coupling_time(k_current)

         ! Write initial values to file for k=1,10,30,50,80,100
         do i = 1,6
            if (k == prints(i)) then
               write(27,'(5(E17.8))') x_t(1), delta(1,k), delta_b(1,k), v(1,k), v_b(1,k)
               write(28,'(5(E17.8))') x_t(1), Phi(1,k), Psi(1,k), Theta(1,0,k), Theta(1,1,k)
               write(26,'(3(E17.8))') x_t(1), dPhi(1,k), dPsi(1,k)
            end if
         end do


         ! Task: Integrate from x_init until the end of tight coupling, using
```

```fortran
       !       the tight coupling equations
       j=2
       do while (x_t(j) < x_tc)
          x     = x_t(j)
          a     = exp(x)
          bleta = get_eta(x)
          H     = get_H_p(x)
          ckH   = ck/H
          dtau  = get_dtau(x)

          ! Solve next evolution step
          call odeint(y_tight_coupling,x_t(j-1),x,eps,h1,hmin,deriv_tc,bsstep,output)

          ! Save variables
          delta(j,k)     = y_tight_coupling(1)
          delta_b(j,k)   = y_tight_coupling(2)
          v(j,k)         = y_tight_coupling(3)
          v_b(j,k)       = y_tight_coupling(4)
          Phi(j,k)       = y_tight_coupling(5)
          Theta(j,0,k)   = y_tight_coupling(6)
          Theta(j,1,k)   = y_tight_coupling(7)
          Theta(j,2,k)   = -20.d0*ckH/(45.d0*dtau)*Theta(j,1,k)
          Psi(j,k)       = -Phi(j,k) - 12.d0*(H_0/(ck*a))**2.d0*Omega_r*Theta(j,2,k)

          ! Task: Store derivatives that are required for C_l estimation
          call deriv_tc(x_t(j),y_tight_coupling,dydx)
          dv_b(j,k)      = dydx(4)
          dPhi(j,k)      = dydx(5)
          dTheta(j,0,k) = dydx(6)
          dTheta(j,1,k) = dydx(7)
          dTheta(j,2,k) = 2.d0/5.d0*ckH*Theta(j,1,k) -&
                          3.d0*ckH/(5.d0)*Theta(j,3,k)+dtau*0.9d0*Theta(j,2,k)
          dPsi(j,k)      = -dPhi(j,k) - 12.d0*H_0**2.d0/(ck*a)**2.d0*Omega_r*&
                          (-2.d0*Theta(j,2,k)+dTheta(j,2,k))

          ! Write values to file for k=1,10,30,50,80,100
          do i = 1,6
             if (k == prints(i)) then
                write(27,'(5(E17.8))') x_t(j), delta(j,k), delta_b(j,k), v(j,k), v_b(j,k)
                write(28,'(5(E17.8))') x_t(j), Phi(j,k), Psi(j,k), Theta(j,0,k), Theta(j,1,k)
                write(26,'(3(E17.8))') x_t(j), dPhi(j,k), dPsi(j,k)
             end if
          end do
          j = j+1
       end do
       j_tc = j

       ! Task: Set up variables for integration from the end of tight coupling until today
```

```fortran
       ! ------------------------------------------------------------------------------------
       y(1:7) = y_tight_coupling(1:7)
       y(8)   = -20.d0*ckH/(45.d0*dtau)*Theta(i,1,k)
       do l = 3, lmax_int
          y(6+l) = -l*ckH/((2.d0*l+1.d0)*dtau)*y(6+l-1)
       end do

       do i = j_tc,n_t
          x     = x_t(i)
          a     = exp(x)
          bleta = get_eta(x)
          H     = get_H_p(x)
          ckH   = ck/H
          dtau  = get_dtau(x)

          ! Task: Integrate equations from tight coupling to today
          call odeint(y,x_t(i-1),x,eps,h1,hmin,deriv,bsstep,output)

          ! Task: Store variables at time step i in global variables
          ! ----------------------------------------------------------
          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))**2.d0*Omega_r*Theta(i,2,k)

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


          ! Write to file
          do j = 1,6
             if (k == prints(j)) then
                write(27,'(5(E17.8))') x_t(i), delta(i,k),
```

```
  delta_b(i,k), v(i,k), v_b(i,k)
                write(28,'(5(E17.8))') x_t(i), Phi(i,k),   Psi(i
  ,k), Theta(i,0,k), Theta(i,1,k)
                write(26,'(3(E17.8))') x_t(i), dPhi(i,k),  dPsi(
  i,k)
             end if
          end do

        end do
     end do

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

     open(29,file='evo_dat.unf',form='unformatted')
     write(29) Theta
     write(29) delta
     write(29) delta_b
     write(29) Phi
     write(29) Psi
     write(29) v
     write(29) v_b
     write(29) dPhi
     write(29) dPsi
     write(29) dv_b
     write(29) dTheta

     close(26)
     close(27)
     close(28)
     close(29)
     write(*,*) 'Perturbation Equations Integrated.'

  endif

  write(*,*) '————————————————————————————————————'
end subroutine integrate_perturbation_eqns

! ———————————————————— derivative subroutines
  ————————————————————————

subroutine deriv_tc(x,y_tc,dydx)
  use healpix_types
  implicit none
  real(dp),                intent(in)  :: x
  real(dp), dimension(:), intent(in)  :: y_tc
  real(dp), dimension(:), intent(out) :: dydx

  real(dp) :: d_delta
  real(dp) :: d_delta_b
  real(dp) :: d_v
  real(dp) :: q,R

  real(dp) :: delta,delta_b,v,v_b,Phi,Theta0,Theta1,Theta2
  real(dp) :: Psi,dPhi,dTheta0,dv_b,dTheta1
  real(dp) :: dtau, ddtau,a,H_p,dH_p,ckH_p
```

```fortran

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

    dtau       = get_dtau(x)
    ddtau      = get_ddtau(x)
    a          = exp(x)
    H_p        = get_H_p(x)
    dH_p       = get_dH_p(x)
    ckH_p      = c*k_current/H_p

    Theta2     = -20.d0*ckH_p/(45.d0*dtau)*Theta1

    R          = (4.d0*Omega_r)/(3.d0*Omega_b*a)

    Psi        = -Phi - 12.d0*(H_0/(c*k_current*a))**2.d0*Omega_r*Theta2

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

    dTheta0    = -ckH_p*Theta1 - dPhi

    d_delta    = ckH_p*v - 3.d0*dPhi

    d_delta_b  = ckH_p*v_b - 3.d0*dPhi

    d_v        = -v - ckH_p*Psi

    q          = (-((1.d0-2.d0*R)*dtau + (1.d0+R)*ddtau)*(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)*dtau+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)

    ! Output
    dydx(1)    = d_delta
    dydx(2)    = d_delta_b
    dydx(3)    = d_v
    dydx(4)    = dv_b
    dydx(5)    = dPhi
    dydx(6)    = dTheta0
    dydx(7)    = dTheta1

  end subroutine deriv_tc


```

```
  subroutine deriv(x,y,dydx)
    use healpix_types
    implicit none

    real(dp),                intent(in)  :: x
    real(dp), dimension(:), intent(in)  :: y
    real(dp), dimension(:), intent(out) :: dydx

    real(dp) :: d_delta
    real(dp) :: d_delta_b
    real(dp) :: d_v
    real(dp) :: R

    real(dp) :: delta,delta_b,v,v_b,Phi,Theta0,Theta1,Theta2,Theta3
    ,Theta4,Theta5,Theta6
    real(dp) :: Psi, dPhi,dTheta0,dv_b,dTheta1,dTheta2
    real(dp) :: a,H_p,ckH_p,dtau,bleta
    integer(i4b) :: l

    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    = y(9)
    Theta4    = y(10)
    Theta5    = y(11)
    Theta6    = y(12)

    a         = exp(x)
    H_p       = get_H_p(x)
    ckH_p     = c*k_current/H_p
    dtau      = get_dtau(x)
    bleta     = get_eta(x)

   R          = (4.d0*Omega_r)/(3.d0*Omega_b*a)
    Psi       = -Phi - 12.d0*H_0*H_0/((c*k_current*a)*(c*k_current*
    a))*Omega_r*Theta2

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

    dTheta0   = -ckH_p*Theta1 - dPhi
    d_delta   = ckH_p*v   - 3.d0*dPhi
    d_delta_b = ckH_p*v_b - 3.d0*dPhi
    d_v       = -v - ckH_p*Psi

    dv_b      = -v_b - ckH_p*Psi + dtau*R*(3.d0*Theta1 + v_b)

    dTheta1   = ckH_p*Theta0/3.d0 - 2*ckH_p*Theta2/3.d0 + ckH_p*Psi
    /3.d0 + dtau*(Theta1 + v_b/3.d0)
    dTheta2   = 2.d0/5.d0*ckH_p*Theta1 - 3.d0/5.d0*ckH_p*Theta3+
    dtau*0.9d0*Theta2
```

```fortran

    do l=3,lmax_int-1
       dydx(6+l) = l/(2.d0*l+1.d0)*ckH_p*y(5+l) - (l+1.d0)/(2.d0*l+1.d0)*ckH_p*y(7+l) + dtau*y(6+l)
    end do

    dydx(6+lmax_int) = ckH_p*y(5+lmax_int) - c*(lmax_int+1.d0)/(H_p*bleta)*y(6+lmax_int) &
                       + dtau*y(6+lmax_int)

    ! Output
    dydx(1)   = d_delta
    dydx(2)   = d_delta_b
    dydx(3)   = d_v
    dydx(4)   = dv_b
    dydx(5)   = dPhi
    dydx(6)   = dTheta0
    dydx(7)   = dTheta1
    dydx(8)   = dTheta2

  end subroutine deriv

  ! 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
    integer(i4b)          :: i,n
    real(dp)              :: x
    n = 1d4
    do i=0,n
       x = x_init + i*(0.d0-x_init)/n
       if (x < x_start_rec .and. abs(c*k/(get_H_p(x)*get_dtau(x))) < 0.1d0 .and. &
            abs(get_dtau(x)) > 10.d0) then
          get_tight_coupling_time = x
       end if
    end do
  end function get_tight_coupling_time

end module evolution_mod
```