Milestone 3 Code

Daniel Herman

April 2018

1 Code

```
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),                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(:)     :: dydx

contains

```

```fortran
  ! 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

  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)

    ! 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+1000, 0:lmax_int, n_k))
    allocate(delta(1:n_t+1000, n_k))
    allocate(delta_b(1:n_t+1000, n_k))
    allocate(v(1:n_t+1000, n_k))
    allocate(v_b(1:n_t+1000, n_k))
    allocate(Phi(1:n_t+1000, n_k))
    allocate(Psi(1:n_t+1000, n_k))
    allocate(dPhi(1:n_t+1000, n_k))
    allocate(dPsi(1:n_t+1000, n_k))
    allocate(dv_b(1:n_t+1000, n_k))
```

```
    allocate(dTheta(1:n_t+1000, 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
    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

  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

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

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

    allocate(y(npar))
    allocate(dydx(npar))
    allocate(y_tight_coupling(7))
    allocate(x_temp(1000))
    allocate(x_post(n_t))
    allocate(x_total(n_t+1000))
    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

    ! Fill in the  x-array with 1000 points before start_rec and 500 points after
    do i = 1,1000
       x_temp(i) = x_init + (i-1)*(x_start_rec-x_init)/999
       x_total(i) = x_temp(i)
    end do
    do i = 1,n_t
       x_post(i) = x_start_rec + i*(-x_start_rec)/n_t
       x_total(i+1000) = x_post(i)
    end do


    open(27, file='vandb.dat')
    open(28, file='phi_theta.dat')

    ! Propagate each k-mode independently
    do k = 1, n_k

       write(*,*) 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(*,*) x_tc

       ! 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_total(1), delta(1,k), delta_b(1,k), v(1,k), v_b(1,k)
             write(28,'(5(E17.8))') x_total(1), Phi(1,k), Psi(1,k), Theta(1,0,k), Theta(1,1,k)
          end if
```

```
      end do

      ! Task: Integrate from x_init until the end of tight coupling, using
      !       the tight coupling equations
      j=2
      do while (x_total(j) < x_tc)
           x     = x_total(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_total(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)

           ! And for higher l's
           !do l=3,lmax_int
           !   Theta(j,l,k) = -l/(2.d0*l+1)*ckH/dtau*Theta(j,l-1,k)
           !end do

           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_total(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)

           !do l=3,lmax_int-1
           !   dTheta(j,l,k) = l*ckH/(2.d0*l+1.d0)*Theta(j,l-1,k) -&
           !                   (l+1.d0)*ckH/(2.d0*l+1.d0)*Theta(j,l+1,k) + dtau*Theta(j,l,k)
           !end do

           !dTheta(j,lmax_int,k) = ckH*Theta(j,lmax_int-1,k) -&
           !                       c*(l+1.d0)/(H*bleta)*Theta(j,
```

```
lmax_int,k)&
        !                          + dtau*Theta(k,lmax_int,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_total(j), delta(j,k), delta_b(j,k), v(j,k), v_b(j,k)
              write(28,'(5(E17.8))') x_total(j), Phi(j,k), Psi(j,k), Theta(j,0,k), Theta(j,1,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
   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,1000+n_t
         x = x_total(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_total(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_total(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))

          do j = 1,6
             if (k == prints(j)) then
                write(27,'(5(E17.8))') x_total(i), delta(i,k), delta_b(i,k), v(i,k), v_b(i,k)
                write(28,'(5(E17.8))') x_total(i), Phi(i,k), Psi(i,k), Theta(i,0,k), Theta(i,1,k)
             end if
          end do

       end do
    end do

    close(27)
    close(28)

    deallocate(y_tight_coupling)
    deallocate(y)
    deallocate(dydx)
    deallocate(x_temp)
    deallocate(x_post)

  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
```

```fortran
    real(dp) :: Psi,dPhi,dTheta0,dv_b,dTheta1
    real(dp) :: dtau, ddtau,a,H_p,dH_p,ckH_p

    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
```

```fortran


  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

    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
```