Milestone 1 Code

Daniel Herman

February 2018

1 Code

```
module time_mod
  use healpix_types
  use params
  use spline_1D_mod
  use ode_solver
  implicit none

  integer(i4b)                              :: n_t                !
    Number of x-values
  real(dp),     allocatable, dimension(:) :: x_t                !
    Grid of relevant x-values
  real(dp),     allocatable, dimension(:) :: a_t                !
    Grid of relevant a-values
  real(dp),     allocatable, dimension(:) :: eta_t              !
    Grid of relevenat eta-values

  integer(i4b)                              :: n_eta              !
    Number of eta grid poins
  real(dp),     allocatable, dimension(:) :: x_eta              !
    Grid points for eta
  real(dp),     allocatable, dimension(:) :: z_eta              !
    Grid points for z_eta
  real(dp),     allocatable, dimension(:) :: a_eta              !
    Grid points for a_eta
  real(dp),     allocatable, dimension(:) :: eta, eta2          !
    Eta and eta'' at each grid point
  real(dp),     allocatable, dimension(:) :: dydx

  real(dp)                                  :: rho_crit0          !
    critical density today
  real(dp)                                  :: rho_m0             !
    matter density today
  real(dp)                                  :: rho_b0             !
    baryon density today
  real(dp)                                  :: rho_r0             !
    radiation density today
  real(dp)                                  :: rho_nu0            !
    neutrino density today
  real(dp)                                  :: rho_lambda0        !
    vacuum energy density today
```

```
26   real(dp),     allocatable, dimension(:) :: rho_m              !
       matter density grid
27   real(dp),     allocatable, dimension(:) :: rho_b              !
       baryon density grid
28   real(dp),     allocatable, dimension(:) :: rho_r              !
       radiation density grid
29   real(dp),     allocatable, dimension(:) :: rho_nu             !
       neutrino density grid
30   real(dp),     allocatable, dimension(:) :: rho_lambda         !
       vacuum energy density grid
31   real(dp),     allocatable, dimension(:) :: Omega_mx           !
       Relative densities
32   real(dp),     allocatable, dimension(:) :: Omega_bx           ! ''
33   real(dp),     allocatable, dimension(:) :: Omega_rx           ! ''
34   real(dp),     allocatable, dimension(:) :: Omega_nux          ! ''
35   real(dp),     allocatable, dimension(:) :: Omega_lambdax      ! ''
36   real(dp),     allocatable, dimension(:) :: Hx                 !
       Hubble constant as a func of x
37   real(dp)                                :: m2mpc             !
       Value for changing m to Mpc
38   real(dp)                                :: kmmpc
39
40 contains
41
42   subroutine initialize_time_mod
43     implicit none
44
45     integer(i4b) :: i, n, n1, n2
46     real(dp)     :: z_start_rec, z_end_rec, z_0, x_start_rec,
       x_end_rec, x_0
47     real(dp)     :: dx,x_eta1, x_eta2, a_init,a_end,eta_init
48     real(dp)     :: eps,hmin,yp1,ypn,h1
49
50     ! Define two epochs, 1) during and 2) after recombination.
51
52     !--------------------------------------
53     ! Initialize the values for variables that are used during the
       calculations
54
55     n1          = 200                       ! Number of grid points
        during recombination
56     n2          = 300                       ! Number of grid points
        after recombination
57     n_t         = n1 + n2                   ! Total number of grid
       points
58
59     z_start_rec = 1630.4d0                  ! Redshift of start of
       recombination
60     z_end_rec   = 614.2d0                   ! Redshift of end of
       recombination
61     z_0         = 0.d0                      ! Redshift today
62
63     x_start_rec = -log(1.d0 + z_start_rec)  ! x of start of
       recombination
64     x_end_rec   = -log(1.d0 + z_end_rec)    ! x of end of
       recombination
65     x_0         = 0.d0                      ! x today
```

```
66
67    n_eta       = 1000                        ! Number of eta grid
      points (for spline)
68    a_init      = 1.d-10                      ! Start value of a for
      eta evaluation
69    a_end       = 1.d0
70    x_eta1      = log(a_init)                 ! Start value of x for
      eta evaluation
71    x_eta2      = 0.d0                        ! End value of x for
      eta evaluation
72
73    eps = 1.d-10
74    hmin = 0.d0
75
76    m2mpc = 3.2408d-23                        ! 3.086*10^22m = 1Mpc
77    kmmpc = 3.086d19                          ! ''
78
79    !----------------------------------------
80    ! Task: Fill in x and a grids
81
82    allocate(x_t(n_t))
83    do i = 0,n1-1                         ! Fill recombination
      interval
84       x_t(i+1) = x_start_rec + i*(x_end_rec-x_start_rec)/(n1-1)
85    end do
86
87    do i = 1,n2                           ! Fill post recombination
      interval
88       x_t(n1+i) = x_end_rec + i*(x_0-x_end_rec)/(n2)
89    end do
90
91    allocate(a_t(n_t))
92    a_t = exp(x_t)                        ! Fill the a_t-grid using
      the x_t values
93
94    ! Task: 1) Compute the conformal time at each eta time step
95    !       2) Spline the resulting function, using the provided "
      spline" routine in spline_1D_mod.f90
96    allocate(x_eta(n_eta))
97    allocate(a_eta(n_eta))
98    allocate(z_eta(n_eta))
99    allocate(eta2(n_eta))
100
101   ! Compute the x_eta values
102   x_eta(1) = x_eta1
103   do i = 1,n_eta-1
104      x_eta(i+1) = x_eta1 + i*(x_eta2-x_eta1)/(n_eta-1)
105   end do
106
107   ! Compute the a_eta and z_eta values
108   a_eta = exp(x_eta)
109   z_eta = 1.d0/a_eta - 1.d0
110
111   !----------------------------------------
112   ! Density calculations
113
114   rho_crit0  = 3.d0*H_0**2.d0/(8.d0*pi*G_grav)
```

```
115     rho_m0      = Omega_m      * rho_crit0
116     rho_b0      = Omega_b      * rho_crit0
117     rho_r0      = Omega_r      * rho_crit0
118     rho_nu0     = Omega_nu     * rho_crit0
119     rho_lambda0= Omega_lambda * rho_crit0
120
121     allocate(rho_m(n_eta))
122     allocate(rho_b(n_eta))
123     allocate(rho_r(n_eta))
124     allocate(rho_nu(n_eta))
125     allocate(rho_lambda(n_eta))
126     allocate(Omega_mx(n_eta))
127     allocate(Omega_bx(n_eta))
128     allocate(Omega_rx(n_eta))
129     allocate(Omega_nux(n_eta))
130     allocate(Omega_lambdax(n_eta))
131     allocate(Hx(n_eta))
132
133     do i=1,n_eta
134        Hx(i) = get_H(x_eta(i))*kmmpc
135        Omega_mx(i)      = Omega_m      *H_0**2.d0/Hx(i)**2.d0 *
    a_eta(i)**-3.d0
136        Omega_bx(i)      = Omega_b      *H_0**2.d0/Hx(i)**2.d0 *
    a_eta(i)**-3.d0
137        Omega_rx(i)      = Omega_r      *H_0**2.d0/Hx(i)**2.d0 *
    a_eta(i)**-4.d0
138        Omega_nux(i)     = Omega_nu     *H_0**2.d0/Hx(i)**2.d0 *
    a_eta(i)**-4.d0
139        Omega_lambdax(i) = Omega_lambda*H_0**2.d0/Hx(i)**2.d0
140     end do
141
142     ! Print the density values to file
143
144     open(50, file='densities.dat')
145       do i=1,n_eta
146          write(50,'(5(E17.8))') Hx(i),Omega_mx(i),Omega_bx(i),
    Omega_rx(i),Omega_lambdax(i)
147       end do
148     close(50)
149
150     !---------------------------------------
151     ! Calculate the eta values, initializing eta(1) using the
    assumption that the universe
152     ! is radiation dominated before recomination
153
154     allocate(eta(n_eta))
155     eta(1) = 1.d-10/(H_0*sqrt(Omega_r+Omega_nu)) ! early universe
    is rad/neutrino dominated
156     h1 = abs(1.d-2*(a_eta(1)-a_eta(2)))
157     allocate(dydx(1))
158
159     do i =2,n_eta
160        eta(i) = eta(i-1)
161        call odeint(eta(i:i),a_eta(i-1),a_eta(i),eps,h1,hmin,
    eta_derivs,bsstep,output)
162     end do
163
```

```
164     eta = c*eta                     ! eta is defined by c/a
165     eta = m2mpc*eta                 ! Put eta in units of cMpc
166
167     ! Print x,a, and eta values to file for plotting
168     open(54, file='data.dat')
169        do i=1,n_eta
170           write(54,'(4(E17.8))') x_eta(i),a_eta(i),eta(i),z_eta(i)
171        end do
172     close(54)
173
174   end subroutine initialize_time_mod
175
176   !-------------------------------------
177
178   subroutine eta_derivs(a,eta,dydx) ! Defining the eta deriv
179
180   use healpix_types
181      implicit none
182
183      real(dp),                intent(in)  :: a
184      real(dp), dimension(:), intent(in)  :: eta
185      real(dp), dimension(:), intent(out) :: dydx
186      real(dp)                            :: H_p
187      real(dp)                            :: x
188
189      x = log(a)
190      H_p = get_H_p(x)
191      dydx = c/(a*H_p)
192
193   end subroutine eta_derivs
194
195   subroutine output(x,y)
196      use healpix_types
197      implicit none
198
199      real(dp),                intent(in)  :: x
200      real(dp), dimension(:), intent(in)  :: y
201
202   end subroutine output
203
204 !--------------------Done with subroutines--------
205 !----------------------Define Functions-----------
206
207   ! Task: Write a function that computes H at given x
208   function get_H(x)
209     implicit none
210
211     real(dp), intent(in) :: x
212     real(dp)             :: get_H
213     real(dp)             :: a
214     a = exp(x)
215
216     get_H = H_0*sqrt((Omega_b+Omega_m)*a**-3.d0 + (Omega_r +
        Omega_nu)*a**-4.d0 + Omega_lambda)
217
218   end function get_H
219
```

```
220   ! Task: Write a function that computes H' = a*H  at given x
221   function get_H_p(x)
222     implicit none
223
224     real(dp), intent(in) :: x
225     real(dp)             :: get_H_p
226     real(dp)             :: a
227     a = exp(x)
228
229     get_H_p = a*get_H(x)
230
231   end function get_H_p
232
233   ! Task: Write a function that computes dH'/dx at given x
234   function get_dH_p(x)
235     implicit none
236
237     real(dp), intent(in) :: x
238     real(dp)             :: get_dH_p
239     get_dH_p = H_0/2.d0*1/sqrt((Omega_m+Omega_b)*exp(-x)+Omega_r*
      exp(-2.d0*x) &
240     + Omega_lambda*exp(2.d0*x)) * (-(Omega_m+Omega_b)*exp(-x)-2.d0*
      Omega_r*exp(-2.d0*x) &
241     + 2.d0*Omega_lambda*exp(2.d0*x))
242
243   end function get_dH_p
244
245   ! Task: Write a function that computes eta(x), using the
        previously precomputed splined function
246   function get_eta(x_in)
247     implicit none
248
249     real(dp), intent(in) :: x_in
250     real(dp)             :: get_eta
251     real(dp)             :: a_in
252     a_in = exp(x_in)
253     get_eta = splint(a_eta,eta,eta2,a_in)
254
255   end function get_eta
256
257 end module time_mod
```