

Milestone 3 Code

Daniel Herman

April 2018

1 Code

```
1 module evolution_mod
2   use healpix_types
3   use params
4   use time_mod
5   use ode_solver
6   use rec_mod
7   implicit none
8
9   ! Accuracy parameters
10  real(dp),      parameter, private :: a_init    = 1.d-8
11  real(dp),      private :: x_init
12  real(dp),      parameter, private :: k_min     = 0.1d0 * H_0 / c
13  real(dp),      parameter, private :: k_max     = 1.d3  * H_0 / c
14  integer(i4b),  parameter :: n_k              = 100
15  integer(i4b),  parameter, private :: lmax_int = 6
16
17  ! Perturbation quantities
18  real(dp), allocatable, dimension(:, :, :) :: Theta
19  real(dp), allocatable, dimension(:, :) :: delta
20  real(dp), allocatable, dimension(:, :) :: delta_b
21  real(dp), allocatable, dimension(:, :) :: Phi
22  real(dp), allocatable, dimension(:, :) :: Psi
23  real(dp), allocatable, dimension(:, :) :: v
24  real(dp), allocatable, dimension(:, :) :: v_b
25  real(dp), allocatable, dimension(:, :) :: dPhi
26  real(dp), allocatable, dimension(:, :) :: dPsi
27  real(dp), allocatable, dimension(:, :) :: dv_b
28  real(dp), allocatable, dimension(:, :, :) :: dTheta
29
30  ! Fourier mode list
31  real(dp), allocatable, dimension(:) :: ks
32
33  ! Book-keeping variables
34  real(dp),      private :: k_current
35  integer(i4b),  private :: npar = 6+lmax_int
36
37  real(dp), allocatable, dimension(:) :: dydx
38
39 contains
40
```

```

41 ! NB!!! New routine for 4th milestone only; disregard until then
42 !
43 subroutine get_hires_source_function(k, x, S)
44 implicit none
45
46 real(dp), pointer, dimension(:), intent(out) :: k, x
47 real(dp), pointer, dimension(:, :), intent(out) :: S
48
49 integer(i4b) :: i, j
50 real(dp) :: g, dg, ddg, tau, dt, ddt, H_p, dH_p, ddHH_p, Pi
51 , dPi, ddPi
52 real(dp), allocatable, dimension(:, :) :: S_lores
53
54 ! Task: Output a pre-computed 2D array (over k and x) for the
55 ! source function, S(k,x). Remember to set up (and
56 ! allocate) output
57 ! k and x arrays too.
58 !
59 ! Substeps:
60 ! 1) First compute the source function over the existing k
61 ! and x
62 ! grids
63 ! 2) Then spline this function with a 2D spline
64 ! 3) Finally, resample the source function on a high-
65 ! resolution uniform
66 ! 5000 x 5000 grid and return this, together with
67 ! corresponding
68 ! high-resolution k and x arrays
69
70 end subroutine get_hires_source_function
71
72 ! Routine for initializing and solving the Boltzmann and Einstein
73 ! equations
74 subroutine initialize_perturbation_eqns
75 implicit none
76
77 integer(i4b) :: l, i, k
78 x_init = log(a_init)
79
80 ! Task: Initialize k-grid, ks; quadratic between k_min and
81 ! k_max
82 allocate(ks(n_k))
83 do k=1,n_k
84 ks(k) = k_min + (k_max-k_min)*((k-1.d0)/(n_k-1.d0))**2
85 end do
86
87 ! Allocate arrays for perturbation quantities
88 allocate(Theta(1:n_t+1000, 0:lmax_int, n_k))
89 allocate(delta(1:n_t+1000, n_k))
90 allocate(delta_b(1:n_t+1000, n_k))
91 allocate(v(1:n_t+1000, n_k))
92 allocate(v_b(1:n_t+1000, n_k))
93 allocate(Phi(1:n_t+1000, n_k))
94 allocate(Psi(1:n_t+1000, n_k))
95 allocate(dPhi(1:n_t+1000, n_k))
96 allocate(dPsi(1:n_t+1000, n_k))
97 allocate(dv_b(1:n_t+1000, n_k))

```

```

90     allocate(dTheta(1:n_t+1000, 0:lmax_int, n_k))
91
92     Theta(:, :, :) = 0.d0
93     dTheta(:, :, :) = 0.d0
94     dPhi(:, :) = 0.d0
95     dPsi(:, :) = 0.d0
96
97     ! Task: Set up initial conditions for the Boltzmann and
98     Einstein equations
99     Phi(1, :) = 1.d0
100     Psi(1, :) = -Phi(1, :)
101     delta(1, :) = 1.5d0*Phi(1, :)
102     delta_b(1, :) = delta(1, :)
103     Theta(1, 0, :) = 0.5d0*Phi(1, :)
104
105     do i = 1, n_k
106         v(1, i) = c*ks(i)/(2*get_H_p(x_init))*Phi(1, i)
107         v_b(1, i) = v(1, i)
108         Theta(1, 1, i) = -c*ks(i)/(6*get_H_p(x_init))*Phi(1, i)
109         Theta(1, 2, i) = -20.d0*c*ks(i)/(45.d0*get_H_p(x_init))*
110         get_dtau(x_init)*Theta(1, 1, i)
111         do l = 3, lmax_int
112             Theta(1, l, i) = -1/(2.d0*l+1.d0)*c*ks(i)/(get_H_p(x_init))*
113             get_dtau(x_init)*Theta(1, l-1, i)
114         end do
115     end do
116
117     end subroutine initialize_perturbation_eqns
118
119     subroutine integrate_perturbation_eqns
120         implicit none
121
122         integer(i4b) :: i, j, k, l, j_tc
123         real(dp) :: x1, x2, H, ck, ckH, a, dtau, x, bleta
124         real(dp) :: eps, hmin, h1, x_tc, H_p, dt, t1, t2
125
126         real(dp), allocatable, dimension(:) :: y, y_tight_coupling
127         real(dp), allocatable, dimension(:) :: x_temp
128         real(dp), allocatable, dimension(:) :: x_post
129         real(dp), allocatable, dimension(:) :: x_total
130         real(dp), allocatable, dimension(:) :: prints
131
132         eps = 1.d-8
133         hmin = 0.d0
134         h1 = 1.d-5
135
136         allocate(y(npar))
137         allocate(dydx(npar))
138         allocate(y_tight_coupling(7))
139         allocate(x_temp(1000))
140         allocate(x_post(n_t))
141         allocate(x_total(n_t+1000))
142         allocate(prints(6))
143
144         prints(1) = 1

```

```

144     prints(2)          = 10
145     prints(3)          = 30
146     prints(4)          = 50
147     prints(5)          = 80
148     prints(6)          = 100
149
150     y_tight_coupling = 0.d0
151     y                 = 0.d0
152     dydx              = 0.d0
153
154     ! Fill in the x-array with 1000 points before start_rec and
155     500 points after
156     do i = 1,1000
157         x_temp(i) = x_init + (i-1)*(x_start_rec-x_init)/999
158         x_total(i) = x_temp(i)
159     end do
160     do i = 1,n_t
161         x_post(i) = x_start_rec + i*(-x_start_rec)/n_t
162         x_total(i+1000) = x_post(i)
163     end do
164
165     open(27, file='vandb.dat')
166     open(28, file='phi-theta.dat')
167
168     ! Propagate each k-mode independently
169     do k = 1, n_k
170
171         write(*,*) k
172
173         k_current = ks(k) ! Store k_current as a global module
174         variable
175         ck = c*k_current
176
177         ! Initialize equation set for tight coupling
178         y_tight_coupling(1) = delta(1,k)
179         y_tight_coupling(2) = delta_b(1,k)
180         y_tight_coupling(3) = v(1,k)
181         y_tight_coupling(4) = v_b(1,k)
182         y_tight_coupling(5) = Phi(1,k)
183         y_tight_coupling(6) = Theta(1,0,k)
184         y_tight_coupling(7) = Theta(1,1,k)
185
186         ! Find the time to which tight coupling is assumed,
187         ! and integrate equations to that time
188         x_tc = get_tight_coupling_time(k_current)
189
190         !write(*,*) x_tc
191
192         ! Write initial values to file for k=1,10,30,50,80,100
193         do i = 1,6
194             if (k == prints(i)) then
195                 write(27,'(5(E17.8))') x_total(1), delta(1,k), delta_b
196                 (1,k), v(1,k), v_b(1,k)
197                 write(28,'(5(E17.8))') x_total(1), Phi(1,k), Psi(1,k),
198                 Theta(1,0,k), Theta(1,1,k)
199             end if

```

```

197         end do
198
199         ! Task: Integrate from x_init until the end of tight
coupling, using
200         !         the tight coupling equations
201         j=2
202         do while (x_total(j) < x_tc)
203             x      = x_total(j)
204             a      = exp(x)
205             bleta  = get_eta(x)
206             H      = get_H_p(x)
207             ckH    = ck/H
208             dtau   = get_dtau(x)
209
210             ! Solve next evolution step
211             call odeint(y_tight_coupling, x_total(j-1), x, eps, h1, hmin
, deriv_tc, bsstep, output)
212
213             ! Save variables
214             delta(j,k) = y_tight_coupling(1)
215             delta_b(j,k) = y_tight_coupling(2)
216             v(j,k) = y_tight_coupling(3)
217             v_b(j,k) = y_tight_coupling(4)
218             Phi(j,k) = y_tight_coupling(5)
219             Theta(j,0,k) = y_tight_coupling(6)
220             Theta(j,1,k) = y_tight_coupling(7)
221             Theta(j,2,k) = -20.d0*ckH/(45.d0*dtau)*Theta(j,1,k)
222
223             ! And for higher l's
224             !do l=3,lmax_int
225             !     Theta(j,l,k) = -1/(2.d0*l+1)*ckH/dtau*Theta(j,l-1,k)
226         )
227         !end do
228
229         Psi(j,k) = -Phi(j,k) - 12.d0*(H_0/(ck*a))**2.d0*
Omega_r*Theta(j,2,k)
230
231         ! Task: Store derivatives that are required for C_l
estimation
232         call deriv_tc(x_total(j), y_tight_coupling, dydx)
233         dv_b(j,k) = dydx(4)
234         dPhi(j,k) = dydx(5)
235         dTheta(j,0,k) = dydx(6)
236         dTheta(j,1,k) = dydx(7)
237         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)
238
239         !do l=3,lmax_int-1
240         !     dTheta(j,l,k) = l*ckH/(2.d0*l+1.d0)*Theta(j,l-1,k)
-&
241         !         (l+1.d0)*ckH/(2.d0*l+1.d0)*Theta(j,
l+1,k) + dtau*Theta(j,l,k)
242         !end do
243
244         !dTheta(j,lmax_int,k) = ckH*Theta(j,lmax_int-1,k) -&
245         !     c*(l+1.d0)/(H*bleta)*Theta(j,

```

```

246     lmax_int,k)&
247     !
248     + dtau*Theta(k,lmax_int,k)
249     dPsi(j,k) = -dPhi(j,k) - 12.d0*H_0**2.d0/(ck
250     *a)**2.d0*Omega_r*&
251     (-2.d0*Theta(j,2,k)+dTheta(j,2,k)
252     ))
253
254     ! Write values to file for k=1,10,30,50,80,100
255     do i = 1,6
256         if (k == prints(i)) then
257             write(27,'(5(E17.8))') x_total(j), delta(j,k),
258             delta_b(j,k), v(j,k), v_b(j,k)
259             write(28,'(5(E17.8))') x_total(j), Phi(j,k), Psi(
260             j,k), Theta(j,0,k), Theta(j,1,k)
261         end if
262     end do
263     j = j+1
264     end do
265     j_tc = j
266
267     ! Task: Set up variables for integration from the end of
268     tight coupling
269     ! until today
270     y(1:7) = y_tight_coupling(1:7)
271     y(8) = -20.d0*ckH/(45.d0*dtau)*Theta(i,1,k)
272
273     do l = 3, lmax_int
274         y(6+l) = -l*ckH/((2.d0*l+1.d0)*dtau)*y(6+l-1)
275     end do
276
277     do i = j_tc,1000+n_t
278         x = x_total(i)
279         a = exp(x)
280         bleta = get_eta(x)
281         H = get_H_p(x)
282         ckH = ck/H
283         dtau = get_dtau(x)
284
285         ! Task: Integrate equations from tight coupling to today
286         call odeint(y,x_total(i-1),x,eps,h1,hmin,deriv,bsstep,
287         output)
288
289         ! Task: Store variables at time step i in global
290         variables
291         delta(i,k) = y(1)
292         delta_b(i,k) = y(2)
293         v(i,k) = y(3)
294         v_b(i,k) = y(4)
295         Phi(i,k) = y(5)
296
297         do l = 0, lmax_int
298             Theta(i,l,k) = y(6+l)
299         end do
300
301         Psi(i,k) = - Phi(i,k) - 12.d0*(H_0/(ck*a))**2.d0*

```

```

295      Omega_r*Theta(i,2,k)
296
297      ! Task: Store derivatives that are required for C_l
298      estimation
299      call deriv(x_total(i),y,dydx)
300
301      dPhi(i,k)      = dydx(5)
302      dv_b(i,k)      = dydx(4)
303
304      do l=0,lmax_int
305          dTheta(i,:,k) = dydx(6+l)
306      end do
307
308      dPsi(i,k)      = -dPhi(i,k) - (12.d0*H_0**2.d0)/(ck*a)**2.
309      d0*&
310      Omega_r*(-2.d0*Theta(i,2,k)+dTheta(i,2,k)
311      ))
312
313      do j = 1,6
314          if (k == prints(j)) then
315              write(27,'(5(E17.8))') x_total(i), delta(i,k),
316              delta_b(i,k), v(i,k), v_b(i,k)
317              write(28,'(5(E17.8))') x_total(i), Phi(i,k), Psi(i,
318              k), Theta(i,0,k), Theta(i,1,k)
319          end if
320      end do
321
322      end do
323      end do
324
325      close(27)
326      close(28)
327
328      deallocate(y_tight_coupling)
329      deallocate(y)
330      deallocate(dydx)
331      deallocate(x_temp)
332      deallocate(x_post)
333
334      end subroutine integrate_perturbation_eqns
335
336      ! ----- derivative subroutines
337
338      subroutine deriv_tc(x,y_tc,dydx)
339      use healpix_types
340      implicit none
341      real(dp),          intent(in)    :: x
342      real(dp), dimension(:), intent(in) :: y_tc
343      real(dp), dimension(:), intent(out) :: dydx
344
345      real(dp) :: d_delta
346      real(dp) :: d_delta_b
347      real(dp) :: d_v
348      real(dp) :: q,R
349
350      real(dp) :: delta,delta_b,v,v_b,Phi,Theta0,Theta1,Theta2

```

```

345  real(dp) :: Psi,dPhi,dTheta0,dv_b,dTheta1
346  real(dp) :: dtau, ddtau,a,H_p,dH_p,ckH_p
347
348  delta      = y_tc(1)
349  delta_b    = y_tc(2)
350  v          = y_tc(3)
351  v_b        = y_tc(4)
352  Phi        = y_tc(5)
353  Theta0     = y_tc(6)
354  Theta1     = y_tc(7)
355
356  dtau       = get_dtau(x)
357  ddtau      = get_ddtau(x)
358  a          = exp(x)
359  H_p        = get_H_p(x)
360  dH_p       = get_dH_p(x)
361  ckH_p      = c*k_current/H_p
362
363  Theta2     = -20.d0*ckH_p/(45.d0*dtau)*Theta1
364
365  R          = (4.d0*Omega_r)/(3.d0*Omega_b*a)
366
367  Psi        = -Phi - 12.d0*(H_0/(c*k_current*a))*2.d0*Omega_r*
    Theta2
368
369  dPhi       = Psi - (ckH_p**2.d0)/3.d0*Phi + (H_0/H_p)**2.d0/2.d0
    *(Omega_m/a*delta + &
370      Omega_b/a*delta_b + 4.d0*Omega_r*Theta0/a**2.d0)
371
372  dTheta0    = -ckH_p*Theta1 - dPhi
373
374  d_delta    = ckH_p*v - 3.d0*dPhi
375
376  d_delta_b  = ckH_p*v_b - 3.d0*dPhi
377
378  d_v        = -v - ckH_p*Psi
379
380  q          = (-((1.d0-2.d0*R)*dtau + (1.d0+R)*ddtau)*(3.d0*
    Theta1 + v_b) - ckH_p*Psi + &
381      (1.d0-dH_p/H_p)*ckH_p*(-Theta0-2.d0*Theta2)&
382      - ckH_p*dTheta0)/((1.d0+R)*dtau+dH_p/H_p - 1.d0)
383
384  dv_b       = (1.d0/(1.d0+R))*(-v_b-ckH_p*Psi + R*(q+ckH_p*(-
    Theta0+2.d0*Theta2)-ckH_p*Psi))
385
386  dTheta1    = (1.d0/3.d0)*(q-dv_b)
387
388  ! Output
389  dydx(1)    = d_delta
390  dydx(2)    = d_delta_b
391  dydx(3)    = d_v
392  dydx(4)    = dv_b
393  dydx(5)    = dPhi
394  dydx(6)    = dTheta0
395  dydx(7)    = dTheta1
396
397  end subroutine deriv_tc

```



```

398
399
400 subroutine deriv(x,y,dydx)
401   use healpix_types
402   implicit none
403
404   real(dp),          intent(in)   :: x
405   real(dp), dimension(:), intent(in) :: y
406   real(dp), dimension(:), intent(out) :: dydx
407
408   real(dp) :: d_delta
409   real(dp) :: d_delta_b
410   real(dp) :: d_v
411   real(dp) :: R
412
413   real(dp) :: delta,delta_b,v,v_b,Phi,Theta0,Theta1,Theta2,Theta3,
414   Theta4,Theta5,Theta6
415   real(dp) :: Psi,dPhi,dTheta0,dv_b,dTheta1,dTheta2
416   real(dp) :: a,H_p,ckH_p,dtau,bleta
417   integer(i4b) :: l
418
419   delta      = y(1)
420   delta_b    = y(2)
421   v          = y(3)
422   v_b        = y(4)
423   Phi        = y(5)
424   Theta0     = y(6)
425   Theta1     = y(7)
426   Theta2     = y(8)
427   Theta3     = y(9)
428   Theta4     = y(10)
429   Theta5     = y(11)
430   Theta6     = y(12)
431
432   a          = exp(x)
433   H_p        = get_H_p(x)
434   ckH_p      = c*k_current/H_p
435   dtau       = get_dtau(x)
436   bleta      = get_eta(x)
437
438   R          = (4.d0*Omega_r)/(3.d0*Omega_b*a)
439   Psi        = -Phi - 12.d0*H_0*H_0/((c*k_current*a)*(c*k_current*
440   a))*Omega_r*Theta2
441
442   dPhi       = Psi - (ckH_p**2.d0)/3.d0*Phi + H_0**2.d0/(2.d0*H_p
443   **2.d0)*(Omega_m/a*delta + &
444   Omega_b/a*delta_b + 4.d0*Omega_r*Theta0/a**2.d0)
445
446   dTheta0    = -ckH_p*Theta1 - dPhi
447   d_delta    = ckH_p*v - 3.d0*dPhi
448   d_delta_b  = ckH_p*v_b - 3.d0*dPhi
449   d_v        = -v - ckH_p*Psi
450
451   dv_b       = -v_b - ckH_p*Psi + dtau*R*(3.d0*Theta1 + v_b)
452
453   dTheta1    = ckH_p*Theta0/3.d0 - 2*ckH_p*Theta2/3.d0 + ckH_p*Psi
454   /3.d0 + dtau*(Theta1 + v_b/3.d0)

```

```

451 dTheta2 = 2.d0/5.d0*ckH_p*Theta1 - 3.d0/5.d0*ckH_p*Theta3+
      dtau*0.9d0*Theta2
452
453 do l=3,lmax_int-1
454     dydx(6+l) = 1/(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)
455 end do
456
457 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)
458
459
460 ! Output
461 dydx(1) = d_delta
462 dydx(2) = d_delta_b
463 dydx(3) = d_v
464 dydx(4) = dv_b
465 dydx(5) = dPhi
466 dydx(6) = dTheta0
467 dydx(7) = dTheta1
468 dydx(8) = dTheta2
469
470 end subroutine deriv
471
472 ! Task: Complete the following routine, such that it returns the
      time at which
473 !     tight coupling ends. In this project, we define this as
      either when
474 !     dtau < 10 or c*k/(H_p*dt) > 0.1 or x < x(start of
      recombination)
475
476 function get_tight_coupling_time(k)
477     implicit none
478
479     real(dp), intent(in) :: k
480     real(dp) :: get_tight_coupling_time
481     integer(i4b) :: i,n
482     real(dp) :: x
483     n = 1d4
484     do i=0,n
485         x = x_init + i*(0.d0-x_init)/n
486         if (x < x_start_rec .and. abs(c*k/(get_H_p(x)*get_dtau(x)))
      < 0.1d0 .and. &
487             abs(get_dtau(x)) > 10.d0) then
488             get_tight_coupling_time = x
489         end if
490     end do
491 end function get_tight_coupling_time
492
493 end module evolution_mod

```