

Milestone 4 Code

Daniel Herman

April 2018

1 Code

```
1 module cl_mod
2   use healpix_types
3   use evolution_mod
4   use sphbess_mod
5   implicit none
6
7   real(dp), pointer, dimension(:) :: x_hires, k_hires
8   , l_hires, cl_hires
9   real(dp), pointer, dimension(:, :) :: S, S2, j_l, j_l2
10  real(dp), allocatable, dimension(:) :: bessel, z_spline
11  real(dp), allocatable, dimension(:, :) :: Theta_l, integ2,
12  integ
13  integer(i4b), allocatable, dimension(:) :: ls
14
15 contains
16
17 ! Driver routine for (finally!) computing the CMB power spectrum
18 subroutine compute_cls
19   implicit none
20
21   integer(i4b) :: i, j, l, l_num, x_num, n_spline
22   real(dp) :: dx, S_func, j_func, z, eta, eta0, x0, x_min,
23   x_max, d, e
24   real(dp) :: k_min, k_max, dk, l_max
25   real(dp), allocatable, dimension(:) :: integrand,
26   int_test1, int_test2
27   real(dp), allocatable, dimension(:) :: int_test3,
28   int_test4, int_test5, int_test6
29   real(dp), allocatable, dimension(:) :: int2_test1,
30   int2_test2, int2_test3
31   real(dp), allocatable, dimension(:) :: int2_test4,
32   int2_test5, int2_test6
33   real(dp), pointer, dimension(:) :: x_arg, cls,
34   cls2, ls_dp
35   real(dp), pointer, dimension(:) :: k, x, cl_int
36   real(dp), pointer, dimension(:, :, :, :) :: S_coeff
37   real(dp), allocatable, dimension(:) :: j_l_spline,
38   j_l_spline2
39
40   real(dp) :: t1, t2, integral
41   logical(lgt) :: exist
```

```

33 character(len=128) :: filename
34 real(dp), allocatable, dimension(:) :: y, y2
35
36 n_spline = 5400
37 l_num = 44
38
39 ! Allocate all necessary arrays for C_l estimation
40 allocate(S(n_x_hires, n_k_hires))
41 allocate(x_hires(n_x_hires), k_hires(n_k_hires))
42
43 allocate(ls(l_num))
44 allocate(z_spline(n_spline)) ! Note: z is *not* redshift,
dummy argument of j_l(z)
45
46 allocate(j_l(n_spline, l_num))
47 allocate(j_l2(n_spline, l_num))
48
49 allocate(Theta_l(l_num, n_k_hires))
50 allocate(integ(l_num, n_x_hires))
51 allocate(integ2(l_num, n_k_hires))
52
53 allocate(cl_int(l_num))
54 allocate(cls(l_num))
55 allocate(cls2(l_num))
56 allocate(ls_dp(l_num))
57
58 allocate(int_test1(n_x_hires))
59 allocate(int_test2(n_x_hires))
60 allocate(int_test3(n_x_hires))
61 allocate(int_test4(n_x_hires))
62 allocate(int_test5(n_x_hires))
63 allocate(int_test6(n_x_hires))
64 allocate(int2_test1(n_k_hires))
65 allocate(int2_test2(n_k_hires))
66 allocate(int2_test3(n_k_hires))
67 allocate(int2_test4(n_k_hires))
68 allocate(int2_test5(n_k_hires))
69 allocate(int2_test6(n_k_hires))
70
71
72 ! Open C_l files to write to:
73 !-----
74 open(10, file='thetal_squared1.dat')
75 open(21, file='thetal_squared2.dat')
76 open(22, file='thetal_squared3.dat')
77 open(23, file='thetal_squared4.dat')
78 open(24, file='thetal_squared5.dat')
79 open(25, file='thetal_squared6.dat')
80 open(11, file='transfer1.dat')
81 open(12, file='transfer2.dat')
82 open(43, file='transfer3.dat')
83 open(14, file='transfer4.dat')
84 open(15, file='transfer5.dat')
85 open(16, file='transfer6.dat')
86 open(17, file='hi_res_C_ls.dat')
87
88 ! Set up which l's to compute

```

```

89  ls = (/ 2, 3, 4, 6, 8, 10, 12, 15, 20, 30, 40, 50, 60, 70, 80,
90        & 100, &
91        & 120, 140, 160, 180, 200, 225, 250, 275, 300, 350, 400,
92        & 450, 500, 550, &
93        & 600, 650, 700, 750, 800, 850, 900, 950, 1000, 1050,
94        & 1100, 1150, 1200 /)
95
96  ! Convert the l's to double precision, and create high-
97  ! resolution l arrays
98  do l=1,l_num
99      ls_dp(l) = ls(l)
100  end do
101
102  l_max = int(maxval(ls))
103
104  allocate(l_hires(int(l_max)))
105  allocate(cl_hires(int(l_max)))
106
107  do l=1,l_max
108      l_hires(l) = 1
109  end do
110
111  ! Task: Get source function from evolution_mod
112
113  call get_hires_source_function(x_hires, k_hires, S)
114
115  ! Initialize values for trapezoidal integration
116  x_min = x_hires(1)
117  x_max = x_hires(n_x_hires)
118  dx     = (x_max-x_min)/n_x_hires
119  k_min = k_hires(1)
120  k_max = k_hires(n_k_hires)
121  dk     = (k_max-k_min)/n_k_hires
122
123  ! Task: Initialize spherical Bessel functions for each l; use
124  !       5400 sampled points between
125  !       z = 0 and 3500. Each function must be properly splined
126  ! Hint: It may be useful for speed to store the splined objects
127  !       on disk in an unformatted
128  !       Fortran (= binary) file, so that these only has to be
129  !       computed once. Then, if your
130  !       cache file exists, read from that; if not, generate the
131  !       j_l's on the fly.
132
133  do i=1,n_spline
134      z_spline(i) = 0.d0 + (i-1)*3500.d0/(n_spline-1.d0)
135  end do
136
137  inquire(file='bessel.unf', exist=exist)
138  if (exist) then
139      write(*,*) 'Bessel file found.'
140      open(13,file='bessel.unf', form='unformatted')
141      read(13) j_l
142      close(13)
143  else
144      write(*,*) 'Calculating Bessel function values.'
145      do i=2,n_spline

```

```

138         do l=1,l_num
139             j_l(1,l) = 0.d0
140             call sphbes(ls(l),z_spline(i),j_l(i,l))
141         end do
142     end do
143
144     open(13,file='bessel.unf', form='unformatted')
145     write(13) j_l
146     close(13)
147 end if
148
149 do l=1,l_num
150     call spline(z_spline,j_l(:,l),yp1,ypn,j_l2(:,l))
151 end do
152
153 !write(*,*) splint(z_spline,j_l(:,2), j_l2(:,2), 0.2*H_0/c*
154 get_eta(0.d0))
155 !write(*,*) get_eta(0.d0), 0.2*H_0/c*get_eta(0.d0)
156
157 !| Overall task: Compute the C_l's for each given l
158 !|-----
159 write(*,*) 'Computing C_l's'
160
161 do l = 1,l_num
162     write(*,*) 'l = ',ls(l)
163
164     ! Task: Compute the transfer function, Theta_l(k)
165     ! We will compute the integral by using the trapezoidal
166     ! summing over all x-values of the Bessel weighted Source
167     ! Function
168     do j=1,n_k_hires
169         do i=1,n_x_hires
170             integ(l,i) = S(i,j)*j_lfunc(l,k_hires(j),x_hires(i))
171         end do
172         do i=1,n_x_hires
173             Theta_l(l,j) = Theta_l(l,j) + integ(l,i)
174         end do
175         Theta_l(l,j) = dx*Theta_l(l,j)
176         ! Task: Integrate P(k) * (Theta_l^2 / k) over k to find un-
177         ! normalized C_l's
178         !
179
180         integ2(l,j) = (c*k_hires(j)/H_0)**(n_s-1.d0)*Theta_l(l,j)
181         ** (2.d0)/k_hires(j)
182         cl_int(l) = cl_int(l) + integ2(l,j)
183     end do
184
185     ! Task: Store C_l in an array. Optionally output to file
186     !-----
187     cls(l) = dk*cl_int(l)*ls(l)*(ls(l)+1.d0)/(2.d0*pi)
188     write(*,*) 'C_l = ', cls(l)
189     if (cls(l) > 1.d0) then
190         cls(l) = 4.d-2
191     endif
192     write(*,*) '-----',

```

```

188 end do
189
190 ! Task: Spline C_l's found above, and output smooth C_l curve
    for each integer l
191
192 ! Now spline the C_l's
193 write(*,*) 'Splining C_ls'
194 call spline(ls_dp, cls, yp1, ypn, cls2)
195 write(*,*) 'C_l spline complete!'
196
197 ! Create high-resolution C_l array
198 do l=2,l_max
199     cl_hires(l) = splint(ls_dp, cls, cls2, l_hires(l))
200     ! write(*,*) 'l = ', l_hires(l), 'C_l = ', cl_hires(l)
201     write(17, '(2(E17.8))') l_hires(l), cl_hires(l)
202 end do
203
204 close(12)
205
206 do i=1,n_k_hires
207     int2_test1(i) = (ls(5)*(ls(5)+1))*(Theta_l(5,i)**2.d0)*H_0/(
        c*k_hires(i))
208     int2_test2(i) = (ls(10)*(ls(10)+1))*(Theta_l(10,i)**2.d0)*
        H_0/(c*k_hires(i))
209     int2_test3(i) = (ls(17)*(ls(17)+1))*(Theta_l(17,i)**2.d0)*
        H_0/(c*k_hires(i))
210     int2_test4(i) = (ls(25)*(ls(25)+1))*(Theta_l(25,i)**2.d0)*
        H_0/(c*k_hires(i))
211     int2_test5(i) = (ls(35)*(ls(35)+1))*(Theta_l(35,i)**2.d0)*
        H_0/(c*k_hires(i))
212     int2_test6(i) = (ls(44)*(ls(44)+1))*(Theta_l(44,i)**2.d0)*
        H_0/(c*k_hires(i))
213 end do
214 !|-----
215 !| Integrand test for l=100, k = 159.988
216 !|-----
217 do i=1,n_k_hires
218     int_test1(i) = (ls(5)*(ls(5)+1))*Theta_l(5,i)*H_0/(c*k_hires
        (i))
219     int_test2(i) = (ls(10)*(ls(10)+1))*Theta_l(10,i)*H_0/(c*
        k_hires(i))
220     int_test3(i) = (ls(17)*(ls(17)+1))*Theta_l(17,i)*H_0/(c*
        k_hires(i))
221     int_test4(i) = (ls(25)*(ls(25)+1))*Theta_l(25,i)*H_0/(c*
        k_hires(i))
222     int_test5(i) = (ls(35)*(ls(35)+1))*Theta_l(35,i)*H_0/(c*
        k_hires(i))
223     int_test6(i) = (ls(44)*(ls(44)+1))*Theta_l(44,i)*H_0/(c*
        k_hires(i))
224 end do
225
226 ! write(*,*) ls(5)
227 ! write(*,*) ls(10)
228 ! write(*,*) ls(17)
229 ! write(*,*) ls(25)
230 ! write(*,*) ls(35)
231 ! write(*,*) ls(44)

```

```

232
233 !Integrand testing apparatus:
234 !
235
236 do j=1,n_k_hires
237   if (abs(int2_test1(j)) < 1.d-99) then
238     int2_test1(j) = 0.d0
239   endif
240   if (abs(int2_test2(j)) < 1.d-99) then
241     int2_test2(j) = 0.d0
242   endif
243   if (abs(int2_test3(j)) < 1.d-99) then
244     int2_test3(j) = 0.d0
245   endif
246   if (abs(int2_test4(j)) < 1.d-99) then
247     int2_test4(j) = 0.d0
248   endif
249   if (abs(int2_test5(j)) < 1.d-99) then
250     int2_test5(j) = 0.d0
251   endif
252   if (abs(int2_test6(j)) < 1.d-99) then
253     int2_test6(j) = 0.d0
254   endif
255   if (abs(int_test1(j)) < 1.d-99) then
256     int_test1(j) = 0.d0
257   endif
258   if (abs(int_test2(j)) < 1.d-99) then
259     int_test2(j) = 0.d0
260   endif
261   if (abs(int_test3(j)) < 1.d-99) then
262     int_test3(j) = 0.d0
263   endif
264   if (abs(int_test4(j)) < 1.d-99) then
265     int_test4(j) = 0.d0
266   endif
267   if (abs(int_test5(j)) < 1.d-99) then
268     int_test5(j) = 0.d0
269   endif
270   if (abs(int_test6(j)) < 1.d-99) then
271     int_test6(j) = 0.d0
272   endif
273   if (abs(S(j,2000)) < 1.d-99) then
274     S(j,2000) = 0.d0
275   endif
276   write(10,'(2(E18.7))') c*k_hires(j)/H_0, int2_test1(j)!
277   write(21,'(2(E18.7))') c*k_hires(j)/H_0, int2_test2(j)!
278   write(22,'(2(E18.7))') c*k_hires(j)/H_0, int2_test3(j)!
279   write(23,'(2(E18.7))') c*k_hires(j)/H_0, int2_test4(j)!
280   write(24,'(2(E18.7))') c*k_hires(j)/H_0, int2_test5(j)!
281   write(25,'(2(E18.7))') c*k_hires(j)/H_0, int2_test6(j)!
282   write(11,'(2(E18.7))') c*k_hires(j)/H_0, int_test1(j)!
283   write(12,'(2(E18.7))') c*k_hires(j)/H_0, int_test2(j)!
284   write(43,'(2(E18.7))') c*k_hires(j)/H_0, int_test3(j)!
285   write(14,'(2(E18.7))') c*k_hires(j)/H_0, int_test4(j)!
286   write(15,'(2(E18.7))') c*k_hires(j)/H_0, int_test5(j)!
287   write(16,'(2(E18.7))') c*k_hires(j)/H_0, int_test6(j)!
288 end do

```

```

289      close(10)
290      close(11)
291      close(12)
292      close(43)
293      close(14)
294      close(15)
295      close(16)
296      close(21)
297      close(22)
298      close(23)
299      close(24)
300      close(25)
301
302  !
303
304  end subroutine compute_cls
305
306
307  function j_lfunc(l,k,x)
308      implicit none
309      integer(i4b), intent(in) :: l
310      real(dp), intent(in) :: x,k
311      real(dp) :: j_lfunc
312
313      j_lfunc = splint(z_spline , j_l(:,l), j_l2(:,l), k*(get_eta(0.d0)-
314      get_eta(x)))
315
316  end function j_lfunc
317
318  end module cl_mod

```

```

1  module evolution_mod
2      use healpix_types
3      use params
4      use time_mod
5      use ode_solver
6      use rec_mod
7      use spline_2D_mod
8      implicit none
9
10     ! Accuracy parameters
11     real(dp), parameter, private :: a_init = 1.d-8
12     real(dp), parameter, private :: x_init
13     real(dp), parameter, private :: k_min = 0.1d0 * H_0 / c
14     real(dp), parameter, private :: k_max = 1.d3 * H_0 / c
15     integer(i4b), parameter :: n_k = 100
16     integer(i4b), parameter, private :: lmax_int = 6
17
18     ! Perturbation quantities
19     real(dp), allocatable, dimension(:, :, :) :: Theta
20     real(dp), allocatable, dimension(:, :, ) :: delta
21     real(dp), allocatable, dimension(:, :, ) :: delta_b
22     real(dp), allocatable, dimension(:, :, ) :: Phi
23     real(dp), allocatable, dimension(:, :, ) :: Psi
24     real(dp), allocatable, dimension(:, :, ) :: v
25     real(dp), allocatable, dimension(:, :, ) :: v_b
26     real(dp), allocatable, dimension(:, :, ) :: dPhi

```

```

27 real(dp), allocatable, dimension(:, :) :: dPsi
28 real(dp), allocatable, dimension(:, :) :: dv_b
29 real(dp), allocatable, dimension(:, :, :) :: dTheta
30
31 ! Fourier mode list
32 real(dp), allocatable, dimension(:) :: ks
33
34 ! Book-keeping variables
35 real(dp), private :: k_current
36 integer(i4b), private :: npar = 6+lmax_int
37 real(dp), allocatable, dimension(:) :: prints
38 real(dp), allocatable, dimension(:) :: dydx
39
40 ! Milestone 4 variables
41 real(dp), allocatable, dimension(:, :, :, :) :: S_coeff
42 real(dp), allocatable, dimension(:, :) :: S_lores
43 integer(i4b), parameter :: n_k_hires = 5000
44 integer(i4b), parameter :: n_x_hires = 5000
45
46 contains
47
48 ! NB!!! New routine for 4th milestone only; disregard until then
49 !
50 subroutine get_hires_source_function(x_hires, k_hires, S)
51 implicit none
52
53 real(dp), pointer, dimension(:), intent(out) :: x_hires,
54 k_hires
55 real(dp), pointer, dimension(:, :), intent(out) :: S
56 real(dp), allocatable, dimension(:) :: x_use
57 integer(i4b) :: i, k, i2
58 real(dp) :: g, dg, ddg, tau, dt, ddt, H_p, dH_p, ddHH_p,
59 Pi_c, dPi, ddPi
60 real(dp) :: xi, kk, ck
61
62 ! Task: Output a pre-computed 2D array (over k and x) for the
63 ! source function, S(k,x). Remember to set up (and
64 ! allocate) output k and x arrays too.
65
66 write(*,*) 'Initializing the source function.'
67
68 allocate(x_hires(n_x_hires), k_hires(n_k_hires))
69 allocate(S_lores(501, 1:n_k))
70 allocate(S_coeff(4, 4, 501, n_k))
71 allocate(S(n_x_hires, n_k_hires))
72 allocate(x_use(501))
73
74 do i=1, 501
75 x_use(i) = x_t(i+999)
76 end do
77
78 do i=1, n_x_hires
79 do k=1, n_k_hires
80 x_hires(i) = x_use(1) + (0.d0-x_use(1))*(i-1.d0)/(
81 n_x_hires-1.d0)
82 k_hires(k) = k_min + (k_max - k_min)*((k-1.d0)/(

```



```

n_k_hires -1.d0))
79     end do
80 end do
81
82 ! Substeps:
83 ! 1) First compute the source function over the existing k
and x
84 ! grids
85 do k=1,n_k
86     kk = ks(k)
87     ck = c*kk
88
89     ! Only computing source function from the beginning of
recombination
90     do i=1,501
91         i2 = i+999
92         xi = x_use(i)
93         g = get_g(xi)
94         dg = get_dg(xi)
95         ddg = get_ddg(xi)
96         tau = get_tau(xi)
97         dt = get_dtau(xi)
98         ddt = get_ddtau(xi)
99         H_p = get_H_p(xi)
100        dH_p = get_dH_p(xi)
101        Pi_c = Theta(i2,2,k)
102        dPi = dTheta(i2,2,k)
103        ddPi = 2.d0*ck/(5.d0*H_p)*(-dH_p/H_p*Theta(i2,1,k) +
dTheta(i2,1,k)) &
104            + 0.3d0*(ddt*Pi_c+dt*dPi) &
105            - 3.d0*ck/(5.d0*H_p)*(-dH_p/H_p*Theta(i2,3,k) +
dTheta(i2,3,k))
106
107        ! if (i == 100 .and. k==10) then
108        !     write(*,*) 'need to compare!'
109        !     write(*,*) g*(Theta(i2,0,k)+Psi(i2,k)+0.25d0*Pi_c)
110        ! endif
111
112        S_lores(i,k) = g*(Theta(i2,0,k)+Psi(i2,k)+0.25d0*Pi_c)+
exp(-tau)*(dPsi(i2,k)-dPhi(i2,k)) &
113            - 1.d0/ck*(H_p*(g*dv_b(i2,k)+v_b(i2,k)*dg)
+g*v_b(i2,k)*dH_p) &
114            + 0.75d0/(ck**2)*((H_0**2/2.d0*((Omega_m+
Omega_b)/exp(xi) &
115            + 4.d0*Omega_r/exp(2.d0*xi)+4.d0*
Omega_lambda*exp(2.d0*xi)))&
116            * g*Pi_c+3.d0*H_p*dH_p*(dg*Pi_c+g*dPi) &
117            + H_p**2*(ddg*Pi_c+2.d0*dg*dPi+g*ddPi))
118    end do
119 end do
120
121 write(*,*) '_____ '
122 ! write(*,*) S_lores(100,1), S_lores(100,5), S_lores(100,10)
123 ! write(*,*) get_g(x_use(100))*(Theta(1099,0,10)+Psi(1099,10)
+0.25d0*Theta(1099,2,10))
124 ! write(*,*) get_g(x_use(100)), x_use(100)
125

```

```

126 ! 2) Then spline this function with a 2D spline
127 call splie2-full-precomp(x-use, ks, S_lores, S-coeff)
128
129 ! 3) Finally, resample the source function on a high-
130 resolution uniform
131 ! 5000 x 5000 grid and return this, together with
132 corresponding
133 ! high-resolution k and x arrays
134
135 do k=1,n_k-hires
136   do i=1,n_x-hires
137     S(i,k) = splin2-full-precomp(x-use, ks, S-coeff, x_hires(i)
138   , k_hires(k))
139   end do
140 end do
141
142 write(*,*) 'Source function initialized.'
143 write(*,*) '_____',
144 end subroutine get_hires-source-function
145
146 ! Routine for initializing and solving the Boltzmann and Einstein
147 equations
148 subroutine initialize-perturbation-eqns
149   implicit none
150
151   integer(i4b) :: l, i, k
152   x_init = log(a_init)
153
154   write(*,*) '_____',
155
156   ! Task: Initialize k-grid, ks; quadratic between k_min and
157   k_max
158   allocate(ks(n_k))
159   do k=1,n_k
160     ks(k) = k_min + (k_max-k_min)*((k-1.d0)/(n_k-1.d0))**2
161   end do
162
163   ! Allocate arrays for perturbation quantities
164   allocate(Theta(1:n_t, 0:lmax_int, n_k))
165   allocate(delta(1:n_t, n_k))
166   allocate(delta_b(1:n_t, n_k))
167   allocate(v(1:n_t, n_k))
168   allocate(v_b(1:n_t, n_k))
169   allocate(Phi(1:n_t, n_k))
170   allocate(Psi(1:n_t, n_k))
171   allocate(dPhi(1:n_t, n_k))
172   allocate(dPsi(1:n_t, n_k))
173   allocate(dv_b(1:n_t, n_k))
174   allocate(dTheta(1:n_t, 0:lmax_int, n_k))
175
176   Theta(:, :, :) = 0.d0
177   dTheta(:, :, :) = 0.d0
178   dPhi(:, :, :) = 0.d0
179   dPsi(:, :, :) = 0.d0
180
181   ! Task: Set up initial conditions for the Boltzmann and
182   Einstein equations

```

```

177 Phi(1,:) = 1.d0
178 Psi(1,:) = -Phi(1,:)
179 delta(1,:) = 1.5d0*Phi(1,:)
180 delta_b(1,:) = delta(1,:)
181 Theta(1,0,:) = 0.5d0*Phi(1,:)
182
183 do i = 1, n_k
184     v(1,i) = c*ks(i)/(2*get_H_p(x_init))*Phi(1,i)
185     v_b(1,i) = v(1,i)
186     Theta(1,1,i) = -c*ks(i)/(6*get_H_p(x_init))*Phi(1,i)
187     Theta(1,2,i) = -20.d0*c*ks(i)/(45.d0*get_H_p(x_init))*
get_dtau(x_init)*Theta(1,1,i)
188     do l = 3, lmax_int
189         Theta(1,l,i) = -1/(2.d0*l+1.d0)*c*ks(i)/(get_H_p(x_init)*
get_dtau(x_init))*Theta(1,l-1,i)
190     end do
191 end do
192 write(*,*) 'Perturbation Equations Initialized.'
193 write(*,*) '_____',
194 end subroutine initialize_perturbation_eqns
195
196 subroutine integrate_perturbation_eqns
197     implicit none
198
199     integer(i4b) :: i, j, k, l, j_tc
200     real(dp) :: x1, x2, H, ck, ckH, a, dtau, x, bleta
201     real(dp) :: eps, hmin, h1, x_tc, H_p, dt, t1, t2
202     logical(lgt) :: exist
203
204     real(dp), allocatable, dimension(:) :: y, y_tight_coupling
205     real(dp), allocatable, dimension(:) :: x_temp, x_post, x_total
206
207     eps = 1.d-8
208     hmin = 0.d0
209     h1 = 1.d-5
210
211     allocate(y(npar))
212     allocate(dydx(npar))
213     allocate(y_tight_coupling(7))
214     allocate(prints(6))
215
216     prints(1) = 1
217     prints(2) = 10
218     prints(3) = 30
219     prints(4) = 50
220     prints(5) = 80
221     prints(6) = 100
222
223     y_tight_coupling = 0.d0
224     y = 0.d0
225     dydx = 0.d0
226
227     ! Check to see if perturbation equation values are already
stored
228     ! If not, compute
229     inquire(file='evo-dat.unf', exist=exist)
230     if (exist) then

```

```

231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282

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

```

```

283         !           the tight coupling equations
284         j=2
285         do while (x_t(j) < x_tc)
286             x      = x_t(j)
287             a      = exp(x)
288             bleta  = get_eta(x)
289             H      = get_H_p(x)
290             ckH    = ck/H
291             dtau   = get_dtau(x)
292
293             ! Solve next evolution step
294             call odeint(y_tight_coupling , x_t(j-1), x, eps, h1, hmin,
deriv_tc , bsstep , output)
295
296             ! Save variables
297             delta(j,k) = y_tight_coupling(1)
298             delta_b(j,k) = y_tight_coupling(2)
299             v(j,k) = y_tight_coupling(3)
300             v_b(j,k) = y_tight_coupling(4)
301             Phi(j,k) = y_tight_coupling(5)
302             Theta(j,0,k) = y_tight_coupling(6)
303             Theta(j,1,k) = y_tight_coupling(7)
304             Theta(j,2,k) = -20.d0*ckH/(45.d0*dtau)*Theta(j,1,k)
305             Psi(j,k) = -Phi(j,k) - 12.d0*(H_0/(ck*a))**2.d0*
Omega_r*Theta(j,2,k)
306
307             ! Task: Store derivatives that are required for C_l
estimation
308             call deriv_tc(x_t(j), y_tight_coupling, dydx)
309             dv_b(j,k) = dydx(4)
310             dPhi(j,k) = dydx(5)
311             dTheta(j,0,k) = dydx(6)
312             dTheta(j,1,k) = dydx(7)
313             dTheta(j,2,k) = 2.d0/5.d0*ckH*Theta(j,1,k) -&
314             3.d0*ckH/(5.d0)*Theta(j,3,k)+dtau*0.9
d0*Theta(j,2,k)
315             dPsi(j,k) = -dPhi(j,k) - 12.d0*H_0**2.d0/(ck*a)
**2.d0*Omega_r*&
316             (-2.d0*Theta(j,2,k)+dTheta(j,2,k))
317
318             ! Write values to file for k=1,10,30,50,80,100
319             do i = 1,6
320                 if (k == prints(i)) then
321                     write(27, '(5(E17.8))') x_t(j), delta(j,k),
delta_b(j,k), v(j,k), v_b(j,k)
322                     write(28, '(5(E17.8))') x_t(j), Phi(j,k), Psi(j,k)
), Theta(j,0,k), Theta(j,1,k)
323                     write(26, '(3(E17.8))') x_t(j), dPhi(j,k), dPsi(j
,k)
324                 end if
325             end do
326             j = j+1
327         end do
328         j_tc = j
329
330         ! Task: Set up variables for integration from the end of
tight coupling until today

```

```

331      !

332      y(1:7) = y_tight_coupling(1:7)
333      y(8)   = -20.d0*ckH/(45.d0*dtau)*Theta(i,1,k)
334      do l = 3, lmax_int
335          y(6+l) = -l*ckH/((2.d0*l+1.d0)*dtau)*y(6+l-1)
336      end do

337      do i = j_tc, n_t
338          x      = x_t(i)
339          a      = exp(x)
340          bleta  = get_eta(x)
341          H      = get_H_p(x)
342          ckH    = ck/H
343          dtau   = get_dtau(x)
344
345          ! Task: Integrate equations from tight coupling to
346      today
347          call odeint(y, x_t(i-1), x, eps, h1, hmin, deriv, bsstep,
348      output)
349          ! Task: Store variables at time step i in global
350      variables
351          !
352          delta(i,k) = y(1)
353          delta_b(i,k) = y(2)
354          v(i,k) = y(3)
355          v_b(i,k) = y(4)
356          Phi(i,k) = y(5)
357          do l = 0, lmax_int
358              Theta(i,l,k) = y(6+l)
359          end do
360          Psi(i,k) = -Phi(i,k) - 12.d0*(H_0/(ck*a))**2.d0*
361      Omega_r*Theta(i,2,k)
362          ! Task: Store derivatives that are required for C_l
363      estimation
364          !
365          call deriv(x_t(i), y, dydx)
366          dPhi(i,k) = dydx(5)
367          dv_b(i,k) = dydx(4)
368          do l=0,lmax_int
369              dTheta(i,:,k) = dydx(6+l)
370          end do
371          dPsi(i,k) = -dPhi(i,k) - (12.d0*H_0**2.d0)/(ck*a)
372      **2.d0*&
373      Omega_r*(-2.d0*Theta(i,2,k)+dTheta(i
374      ,2,k))
375
376          ! Write to file
377          do j = 1,6
378              if (k == prints(j)) then
379                  write(27,'(5(E17.8))') x_t(i), delta(i,k),

```

```

377     delta_b(i,k), v(i,k), v_b(i,k)
378         write(28,'(5(E17.8))') x_t(i), Phi(i,k), Psi(i
379         ,k), Theta(i,0,k), Theta(i,1,k)
380         write(26,'(3(E17.8))') x_t(i), dPhi(i,k), dPsi(
381         i,k)
382     end if
383 end do
384
385     deallocate(y_tight_coupling)
386     deallocate(y)
387     deallocate(dydx)
388
389     open(29,file='evo_dat.unf',form='unformatted')
390     write(29) Theta
391     write(29) delta
392     write(29) delta_b
393     write(29) Phi
394     write(29) Psi
395     write(29) v
396     write(29) v_b
397     write(29) dPhi
398     write(29) dPsi
399     write(29) dv_b
400     write(29) dTheta
401
402     close(26)
403     close(27)
404     close(28)
405     close(29)
406     write(*,*) 'Perturbation Equations Integrated.'
407
408 endif
409
410     write(*,*) '_____',
411 end subroutine integrate_perturbation_eqns
412
413 ! _____ derivative subroutines
414
415 subroutine deriv_tc(x,y_tc,dydx)
416     use healpix_types
417     implicit none
418     real(dp), intent(in) :: x
419     real(dp), dimension(:), intent(in) :: y_tc
420     real(dp), dimension(:), intent(out) :: dydx
421
422     real(dp) :: d_delta
423     real(dp) :: d_delta_b
424     real(dp) :: d_v
425     real(dp) :: q,R
426
427     real(dp) :: delta,delta_b,v,v_b,Phi,Theta0,Theta1,Theta2
428     real(dp) :: Psi,dPhi,dTheta0,dv_b,dTheta1
429     real(dp) :: dtau,ddtau,a,H_p,dH_p,ckH_p

```

```

430      delta      = y_tc(1)
431      delta_b     = y_tc(2)
432      v           = y_tc(3)
433      v_b         = y_tc(4)
434      Phi         = y_tc(5)
435      Theta0      = y_tc(6)
436      Theta1      = y_tc(7)
437
438
439      dtau         = get_dtau(x)
440      ddtau        = get_ddtau(x)
441      a            = exp(x)
442      H_p          = get_H_p(x)
443      dH_p         = get_dH_p(x)
444      ckH_p        = c*k_current/H_p
445
446      Theta2       = -20.d0*ckH_p/(45.d0*dtau)*Theta1
447
448      R            = (4.d0*Omega_r)/(3.d0*Omega_b*a)
449
450      Psi          = -Phi - 12.d0*(H_0/(c*k_current*a))**2.d0*Omega_r*
      Theta2
451
452      dPhi         = Psi - (ckH_p**2.d0)/3.d0*Phi + (H_0/H_p)**2.d0/2.d0
      *(Omega_m/a*delta + &
453          Omega_b/a*delta_b + 4.d0*Omega_r*Theta0/a**2.d0)
454
455      dTheta0      = -ckH_p*Theta1 - dPhi
456
457      d_delta      = ckH_p*v - 3.d0*dPhi
458
459      d_delta_b    = ckH_p*v_b - 3.d0*dPhi
460
461      d_v          = -v - ckH_p*Psi
462
463      q            = (-((1.d0-2.d0*R)*dtau + (1.d0+R)*ddtau)*(3.d0*
      Theta1 + v_b) - ckH_p*Psi + &
464          (1.d0-dH_p/H_p)*ckH_p*(-Theta0-2.d0*Theta2)&
465          - ckH_p*dTheta0)/((1.d0+R)*dtau+dH_p/H_p - 1.d0)
466
467      dv_b         = (1.d0/(1.d0+R))*(-v_b-ckH_p*Psi + R*(q+ckH_p*(-
      Theta0+2.d0*Theta2)-ckH_p*Psi))
468
469      dTheta1      = (1.d0/3.d0)*(q-dv_b)
470
471      ! Output
472      dydx(1)      = d_delta
473      dydx(2)      = d_delta_b
474      dydx(3)      = d_v
475      dydx(4)      = dv_b
476      dydx(5)      = dPhi
477      dydx(6)      = dTheta0
478      dydx(7)      = dTheta1
479
480      end subroutine deriv_tc
481
482

```



```

483 subroutine deriv(x,y,dydx)
484     use healpix_types
485     implicit none
486
487     real(dp),                intent(in)    :: x
488     real(dp), dimension(:), intent(in)    :: y
489     real(dp), dimension(:), intent(out)   :: dydx
490
491     real(dp) :: d_delta
492     real(dp) :: d_delta_b
493     real(dp) :: d_v
494     real(dp) :: R
495
496     real(dp) :: delta,delta_b,v,v_b,Phi,Theta0,Theta1,Theta2,Theta3
497     ,Theta4,Theta5,Theta6
498     real(dp) :: Psi,dPhi,dTheta0,dv_b,dTheta1,dTheta2
499     real(dp) :: a,H_p,ckH_p,dtau,bleta
500     integer(i4b) :: l
501
502     delta      = y(1)
503     delta_b    = y(2)
504     v          = y(3)
505     v_b       = y(4)
506     Phi        = y(5)
507     Theta0     = y(6)
508     Theta1     = y(7)
509     Theta2     = y(8)
510     Theta3     = y(9)
511     Theta4     = y(10)
512     Theta5     = y(11)
513     Theta6     = y(12)
514
515     a          = exp(x)
516     H_p        = get_H_p(x)
517     ckH_p      = c*k_current/H_p
518     dtau       = get_dtau(x)
519     bleta      = get_eta(x)
520
521     R          = (4.d0*Omega_r)/(3.d0*Omega_b*a)
522     Psi        = -Phi - 12.d0*H_0*H_0/((c*k_current*a)*(c*k_current*
523     a))*Omega_r*Theta2
524
525     dPhi       = Psi - (ckH_p**2.d0)/3.d0*Phi + H_0**2.d0/(2.d0*H_p
526     **2.d0)*(Omega_m/a*delta + &
527     Omega_b/a*delta_b + 4.d0*Omega_r*Theta0/a**2.d0)
528
529     dTheta0    = -ckH_p*Theta1 - dPhi
530     d_delta    = ckH_p*v - 3.d0*dPhi
531     d_delta_b  = ckH_p*v_b - 3.d0*dPhi
532     d_v        = -v - ckH_p*Psi
533
534     dv_b       = -v_b - ckH_p*Psi + dtau*R*(3.d0*Theta1 + v_b)
535
536     dTheta1    = ckH_p*Theta0/3.d0 - 2*ckH_p*Theta2/3.d0 + ckH_p*Psi
537     /3.d0 + dtau*(Theta1 + v_b/3.d0)
538     dTheta2    = 2.d0/5.d0*ckH_p*Theta1 - 3.d0/5.d0*ckH_p*Theta3+
539     dtau*0.9d0*Theta2

```

```

535
536 do l=3,lmax_int-1
537     dydx(6+l) = 1/(2.d0*l+1.d0)*ckH_p*y(5+l) - (1+1.d0)/(2.d0*l
+1.d0)*ckH_p*y(7+l) + dtau*y(6+l)
538 end do
539
540 dydx(6+lmax_int) = ckH_p*y(5+lmax_int) - c*(lmax_int+1.d0)/(H_p
*bleta)*y(6+lmax_int) &
541     + dtau*y(6+lmax_int)
542
543 ! Output
544 dydx(1) = d_delta
545 dydx(2) = d_delta_b
546 dydx(3) = d_v
547 dydx(4) = dv_b
548 dydx(5) = dPhi
549 dydx(6) = dTheta0
550 dydx(7) = dTheta1
551 dydx(8) = dTheta2
552
553 end subroutine deriv
554
555 ! Task: Complete the following routine, such that it returns the
time at which
556 ! tight coupling ends. In this project, we define this as
either when
557 ! dtau < 10 or c*k/(H_p*dt) > 0.1 or x < x(start of
recombination)
558
559 function get_tight_coupling_time(k)
560     implicit none
561
562     real(dp), intent(in) :: k
563     real(dp) :: get_tight_coupling_time
564     integer(i4b) :: i,n
565     real(dp) :: x
566     n = 1d4
567     do i=0,n
568         x = x_init + i*(0.d0-x_init)/n
569         if (x < x_start_rec .and. abs(c*k/(get_H_p(x)*get_dtau(x)))
< 0.1d0 .and. &
570             abs(get_dtau(x)) > 10.d0) then
571             get_tight_coupling_time = x
572         end if
573     end do
574 end function get_tight_coupling_time
575
576 end module evolution_mod

```