

# Milestone 1 Code

Daniel Herman

February 2018

## 1 Code

```
1 module rec_mod
2   use healpix_types
3   use params
4   use time_mod
5   use ode_solver
6   use spline_1D_mod
7   implicit none
8
9   integer(i4b), private                :: i                ! make i
10   a global variable
11   integer(i4b)                        :: n                ! Number
12   of grid points
13   real(dp), allocatable, dimension(:) :: x_rec, a_rec, z_rec ! Grid
14   real(dp), allocatable, dimension(:) :: tau, tau2, tau22  !
15   Splined tau and second derivatives
16   real(dp), allocatable, dimension(:) :: tau4
17   real(dp), allocatable, dimension(:) :: logtau, logtau2, logtau22
18   real(dp), allocatable, dimension(:) :: n_e, n_e2        !
19   Electron density, n_e
20   real(dp), allocatable, dimension(:) :: logn_e, logn_e2  !
21   Splined log of electron density
22   real(dp), allocatable, dimension(:) :: g, g2, g22, g4    !
23   Splined visibility function
24   real(dp), allocatable, dimension(:) :: X_e, H_rec        !
25   Fractional electron density, n_e / n_H
26   real(dp), allocatable                :: dx                !
27
28   real(dp)                            :: eps, hmin, ypl, ypn, h1 ! ODE/
29   spline stuff we need
30
31 contains
32
33 subroutine initialize_rec_mod
34   implicit none
35
36   real(dp) :: saha_limit, y, T_b, n_b, dydx, xmin, xmax, dx,
37   f, n_e0, X_e0, xstart, xstop
38   real(dp) :: junk, const, phi2, alpha2, beta, beta2,
39   lambda_alpha, lambda2s1s, nls, C_r
40   real(dp) :: z_start_rec, z_end_rec, z_0
41   logical(lgt) :: use_saha
```

```

32
33
34      saha_limit = 0.99d0          ! Switch from Saha to Peebles
      when X_e < 0.99
35      xstart    = log(1.d-10)      ! Start grids at a = 10^-10
36      xstop     = 0.d0             ! Stop grids at a = 1
37      n         = 1000             ! Number of grid points
      between xstart and xstop
38
39      ! ODE int variables
40      eps       = 1.d-10
41      hmin      = 0.d0
42      !Spline variables
43      yp1       = 1.d30
44      ypn       = 1.d30
45
46      z_start_rec = 1630.4d0        ! Redshift at start of
      recombination
47      z_end_rec   = 614.2d0         ! Redshift at the end of
      recombination
48      z_0         = 0.d0            ! Current redshift (duh)
49
50
51      ! Allocate necessary arrays
52
53      allocate(X_e(n))
54      allocate(tau(n),tau2(n),tau22(n),tau4(n))
55      allocate(logtau(n),logtau2(n),logtau22(n))
56      allocate(n_e(n),n_e2(n))
57      allocate(g(n),g2(n),g22(n),g4(n))
58      allocate(x_rec(n),a_rec(n),z_rec(n),H_rec(n))
59      allocate(logn_e(n),logn_e2(n))
60
61
62
63      ! Task: Fill in x,a,z (rec) grid


---


64
65      x_rec(1)    = xstart
66      x_rec(n)    = xstop
67      dx         = (xstop - xstart)/(n-1)
68
69      do i = 1, n
70          x_rec(i) = (i-1)*dx + xstart
71      end do
72
73      do i = 1,n
74          H_rec(i) = get_H(x_rec(i))
75      end do
76
77      a_rec       = exp(x_rec)
78      z_rec       = 1.d0/a_rec - 1.d0
79
80      h1          = abs(1.d-2*(x_rec(1) - x_rec(2))) ! Define the
      step length for the odeint
81

```

```

82      !
83
84      ! Task: Compute X_e and n_e at all grid times
85      use_saha = .true.
86      do i = 1, n
87
88          T_b      = T_0/a_rec(i)
89          n_b      = Omega_b*rho_c/(m_H*a_rec(i)**3)
90          const    = ((m_e*k_b*T_b/(2.d0*PI*hbar**2))**(1.5))*exp((-
epsilon_0)/(T_b*k_b))
91          junk     = (1/n_b)*const
92
93          if (use_saha) then
94              ! Use the Saha equation
95              X_e(i) = (-junk+sqrt(junk*junk+4.d0*junk))/2.d0
96              if (X_e(i) < saha_limit) use_saha = .false.
97          else
98              ! Use the Peebles equation
99              X_e(i) = X_e(i-1)
100             call odeint(X_e(i:i), x_rec(i-1), x_rec(i), eps, h1, hmin,
dX_edx, bsstep, output)
101         end if
102
103         ! Calculate the electron density
104         n_e(i) = X_e(i)*n_b
105
106     end do
107
108     ! open(26, file='X_e.dat')
109     ! do i=1,n
110     !     write(26,'(2(E17.8))') x_rec(i),X_e(i)
111     ! end do
112     ! close(26)
113
114     ! Task: Compute splined (log of) electron density function
115     logn_e = log(n_e)
116     call spline(x_rec, logn_e, yp1, ypn, logn_e2)
117
118     ! Task: Compute optical depth at all grid points
119     tau(n) = 0.d0
120     do i=n-1,1,-1
121         tau(i) = tau(i+1)
122         call odeint(tau(i:i), x_rec(i+1), x_rec(i), eps, h1, hmin, dtaudx,
bsstep, output)
123     end do
124
125     ! Task: Compute splined (log of) optical depth
126     ! Task: Compute splined second derivative of (log of) optical
depth
127
128     do i=1,n
129         tau2(i) = -n_e(i)*sigma_T*c/H_rec(i)
130     end do
131
132     call spline(x_rec, tau, yp1, ypn, tau22)

```

```

133     call spline(x_rec,tau22,yp1,ypn,tau4)
134
135     ! open(25, file='tau.dat')
136     ! do i=1,n
137     !     write(25,'(4(E17.8))') x_rec(i),tau(i),tau2(i),tau22(i)
138     ! end do
139     ! close(25)
140
141     do i=1,n
142         g(i) = -tau2(i)*exp(-tau(i))
143     end do
144
145     ! Task: Compute splined visibility function
146     call spline(x_rec,g,yp1,ypn,g22)
147     ! Task: Compute splined second derivative of visibility
148     ! function
149     call spline(x_rec,g22,yp1,ypn,g4)
150
151     do i=1,n
152         g2(i) = -tau22(i)*exp(-tau(i))+tau2(i)*tau2(i)*exp(-tau(i))
153     end do
154
155     open(20, file='vis2.dat')
156     do i=1,n
157         write(20,'(4(E17.8))') x_rec(i),g(i),g2(i),g22(i)
158     end do
159     close(20)
160
161 end subroutine initialize_rec_mod
162
163 !
164 !
165 !----- Subroutines for odeint -----
166
167 subroutine dX_edx(x, X_e, dydx)
168     use healpix_types
169     implicit none
170     real(dp), intent(in) :: x
171     real(dp), dimension(:), intent(in) :: X_e
172     real(dp), dimension(:), intent(out) :: dydx
173     real(dp) :: T_b, n_b, phi2, alpha2,
174     beta, beta2, n1s, lambda_alpha, C_r
175     real(dp) :: X_e, lambda2s1s
176     real(dp) :: a
177     real(dp) :: H
178
179     lambda2s1s = 8.227d0 ! [s-1]
180     X_e = X_e(1)
181     a = exp(x)
182     H = get_H(x)
183     T_b = T_0/a

```

```

183     n_b          = Omega_b*rho_c/(m_H*a**3)
184     phi2         = 0.448*LOG(epsilon_0/(k_b*T_b))
185     alpha2       = 64.d0*PI/sqrt(27*PI)*(alpha/m_e)**2*sqrt(
186         epsilon_0/(k_b*T_b))*phi2*hbar*hbar/c
187     beta         = alpha2*((m_e*k_b*T_b/(2.d0*PI*hbar*hbar))
188         *(1.5))*exp((-epsilon_0)/(T_b*k_b))
189     nls          = (1.d0 - Xe)*n_b
190     lambda_alpha = H*(3.d0*epsilon_0)**3/(nls*(8.d0*PI)**2)/(c*
191         hbar)**3
192
193     if (T_b <= 169.d0) then
194         beta2     = 0.d0
195     else
196         beta2     = beta*exp(3.d0*epsilon_0/(4.d0*k_b*T_b))
197     end if
198
199     C_r          = (lambda2s1s + lambda_alpha)/(lambda2s1s +
200         lambda_alpha + beta2)
201     dydx         = C_r/H*(beta*(1.d0-Xe) - n_b*alpha2*Xe**2)
202 end subroutine dX_edx
203
204 subroutine dtaudx(x,tau,dydx)
205     use healpix_types
206     implicit none
207     real(dp), intent(in) :: x
208     real(dp), dimension(:), intent(in) :: tau
209     real(dp), dimension(:), intent(out) :: dydx
210     real(dp) :: n_e
211     real(dp) :: H
212     n_e = get_n_e(x)
213     H = get_H(x)
214     dydx = -n_e*sigma_T/H*c
215 end subroutine dtaudx
216
217 !
218 !
219 !----- Functions for future work
220
221 function get_n_e(x_in)
222     implicit none
223     real(dp), intent(in) :: x_in
224     real(dp) :: get_n_e
225     get_n_e = splint(x_rec, logn_e, logn_e2, x_in)
226     get_n_e = exp(get_n_e)
227 end function get_n_e
228
229 function get_tau(x_in)
230     implicit none
231     real(dp), intent(in) :: x_in
232     real(dp) :: get_tau
233     get_tau = splint(x_rec, tau, tau22, x_in)

```

```

231     ! get_tau = exp(get_tau)
232 end function get_tau
233
234
235 function get_dtau(x_in)
236     implicit none
237     real(dp), intent(in) :: x_in
238     real(dp)              :: get_dtau
239     real(dp)              :: n_e, a, H_p
240     H_p                   = get_H_p(x_in)
241     a                     = exp(x_in)
242     n_e                   = get_n_e(x_in)
243     get_dtau              = -n_e*sigma_T*a*c/H_p
244 end function get_dtau
245
246
247 function get_ddtau(x_in)
248     implicit none
249     real(dp), intent(in) :: x_in
250     real(dp)              :: get_ddtau
251     get_ddtau             = splint(x_rec, tau22, tau4, x_in)
252 end function get_ddtau
253
254
255 function get_g(x_in)
256     implicit none
257     real(dp), intent(in) :: x_in
258     real(dp)              :: get_g
259     real(dp)              :: dtau, tau
260     dtau                   = get_dtau(x_in)
261     tau                    = get_tau(x_in)
262     get_g                  = -dtau*exp(-tau)
263 end function get_g
264
265
266 function get_dg(x_in)
267     implicit none
268     real(dp), intent(in) :: x_in
269     real(dp)              :: get_dg
270     get_dg                 = splint_deriv(x_rec, g, g22, x_in)
271 end function get_dg
272
273
274 function get_ddg(x_in)
275     implicit none
276     real(dp), intent(in) :: x_in
277     real(dp)              :: get_ddg
278     get_ddg                = splint(x_rec, g22, g4, x_in)
279 end function get_ddg
280
281
282 end module rec_mod

```