AST5220: Milestone 4

Simen Nyhus Bastnes

1. June 2017

1 Introduction

In this fourth and final milestone, we will combine our previous work, in order to compute the CMB power spectrum. Using line-of-sight integration, we will be able to efficiently compute the photon perturbations, finally giving us the power spectrum C_l. Once we have a program that can compute the CMB power spectrum, we will compare it to the Planck data, and adjust the cosmological parameters in order to fit our power spectrum. As with the previous milestones, our model does not include neutrinos or polarization.

2 Theory

2.1 Line-of-sight integration

In order to compute the power spectrum in an efficient way, we will be using Zaldarriaga and Seljak's *line-of-sight integration approach*. Instead of integrating the set of coupled differential equations in milestone 3 [2] up to $l = 1200$, we formally integrate the equations and then expand into multipoles at the end. This way, we only need to integrate Θ up to $l = 6$, significantly reducing the computation time required. The final expression for this is the so-called transfer function Θ_l.

$$\Theta_l(k, x = 0) = \int_{-\infty}^{0} \tilde{S}(k, x) j_l[k(\eta_0 - \eta)] dx \tag{1}$$

where j_l are spherical Bessel functions, taking into account the projection of the 3D field (characterized by k) onto a 2D sphere (characterized by l). $\tilde{S}$ is the source function, defined as

$$\tilde{S}(k, x) = \tilde{g}\left[\Theta_0 + \Psi + \frac{1}{4}\Pi\right] + e^{-\tau}[\Psi' - \Phi'] - \frac{1}{ck}\frac{d}{dx}(\mathcal{H}\tilde{g}v_b) + \frac{3}{4c^2k^2}\frac{d}{dx}\left[\mathcal{H}\frac{d}{dx}(\mathcal{H}\tilde{g}\Pi)\right] \tag{2}$$

where $\Pi = \Theta_2$ when there is no polarization, and we have computed the other quantities earlier. The first term is the Sachs-Wolfe contribution, which is the temperature fluctuation at the last scattering surface (defined by $\tilde{g}$), the Ψ term is a correction to the fluctuation due to the fact that the photons have to climb out of a gravitational field Ψ, losing energy, and Π is a small correction due to polarization.

The second term is the integrated Sachs-Wolfe contribution, where photons gain or lose energy when moving through time-varying potentials. The third term is the Doppler effect, while the fourth term is another small polarization correction.

2.2 Power spectrum

Now that we have computed the transfer function, it is time to compute the power spectrum. The CMB power spectrum is simply given by the integral

$$C_l = \int \frac{d^3k}{(2\pi)^3} P(k)\Theta_l(k)^2 \tag{3}$$

where C_l is the expectation value of the square of the spherical harmonics coefficients, describing the power of fluctuations at scale l, and $P(k)$ is the primordial power spectrum. In the previous milestone, we used that the equations are linear, and set the initial conditions to be independent of the primordial power spectrum $P(k)$ (setting $\Phi = 0$ for all modes). This is now corrected by rescaling everything by $P(k)$, which works since all our equations are linear.

We note that most inflation models predict a so-called Harrison-Zel'dovich spectrum, for which

$$\frac{k^3}{2\pi^2} P(k) = \left(\frac{ck}{H_0}\right)^{n_s - 1} \tag{4}$$

where n_s is the spectral index of scalar perturbations, and expected to be close to unity. We technically need to find a normalization factor A_s to normalize the spectrum, but for simplicity we will simply scale the maximum value of the spectrum to be $5775\mu\mathrm{K}^2$. The final expression for the spectrum is then

$$C_l = A_s \int_0^\infty \left(\frac{ck}{H_0}\right)^{n_s - 1} \Theta_l^2(k) \frac{dk}{k} \tag{5}$$

Finally, as C_l tends to drop as $C_l \propto l^{-2}$, we will plot the power spectrum in units of $l(l+1)/2\pi$ in $\mu\mathrm{K}^2$, as otherwise seeing features can be difficult.

3 Implementation

We continue building upon the code from previous milestones. First of all, we need to compute the source function for a large number of x and k in order to get good enough precision. Since we have computed all the quantities that equation (2) requires, we use the corresponding x and k arrays to compute the source function. We then spline the source function on high resolution x and k grids.

After that, we need to compute the spherical Bessel functions (using the `sphbess_mod` module). We calculate $j_l(z)$ for $z = 0$ to $z = 3400$ using 5400 samples, and spline it so that we can get j_l for arbitrary $z = k[\eta_0 - \eta(x)]$.

We combine the previous steps, and integrate equation (1) over x with the trapezoidal integration method in order to get the transfer function $\Theta_l(k)$. In order to speed up the program, we opt to integrate over a smaller x-grid, which as Callin [1] results in approximately the same result, though we will be comparing the two in order to be on the safe side.

Having computed the transfer function, we multiply it with the primordial power spectrum $P(k)$ and for each value of l, we integrate it over the high-resolution k-grid, giving us C_l. Finally, we spline C_l over a larger grid with unit stepsize and scale it with $l(l+1)/2\pi$.

4 Results

4.1 Sanity checks

In order to check different parts of our code, we compare with plots in Callin [1]. First of all, we plot the integrand in equation (1).

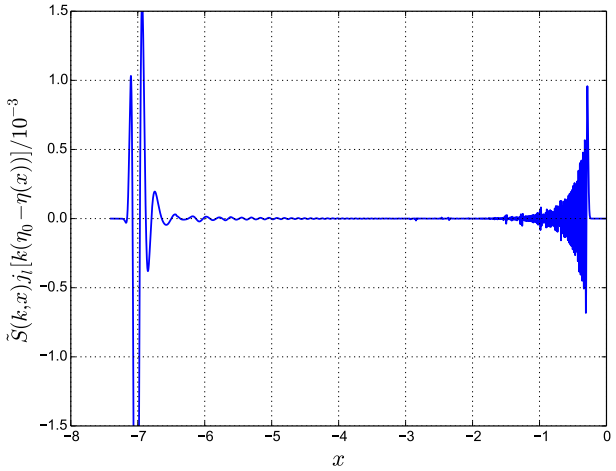


Figure 1: The transfer function integrand plotted against x for $l = 100$ and $k = 340H_0/c$. We note that the plot follows figure 3 in Callin [1] well.

Secondly, we plot the integrand of equation (5) for different values of l

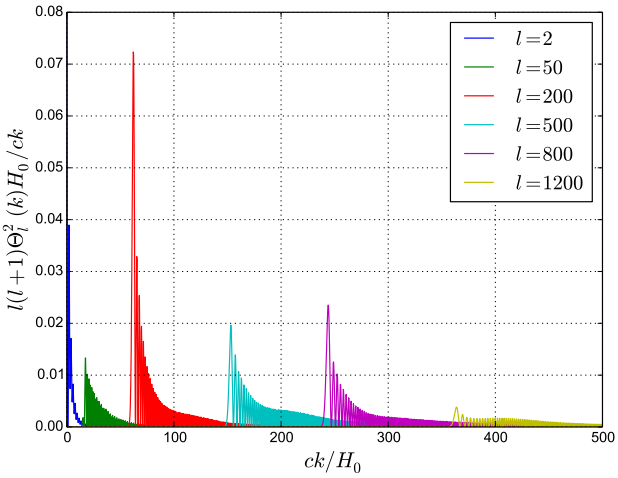


Figure 2: The integrand in equation (5) plotted against ck/H_0 for some different values of l. Comparing to figure 4 in Callin [1], we see that our integrand behaves similarly.

Finally, in order to check if integrating over a smaller x-grid gives an accurate enough result, we compare it with integrating the full grid.

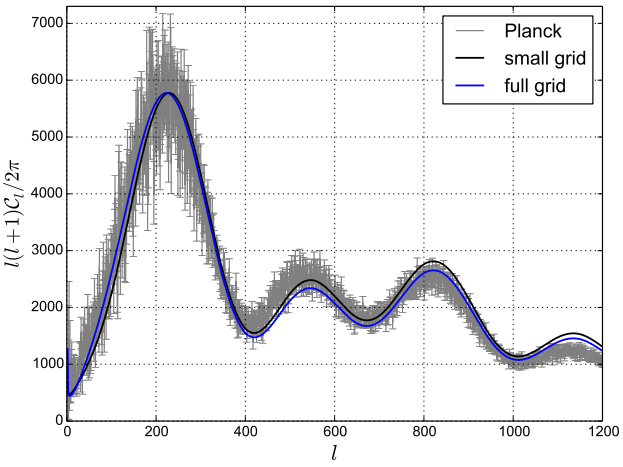


Figure 3: Comparing different x-grid sizes for $\Omega_b = 0.1$. Using the full x-grid, the whole program takes around 9 minutes to complete, compared to a bit under 2 for the smaller grid. We see that the accuracy is decent, and given the speed up, we will continue using the smaller x-grid.

4.2 Default parameters

We plot the power spectrum with the initial parameters given in `params.f90` (with $n_s = 0.96$), with power spectrum measurements from Planck added in order to see how our simulation compares.

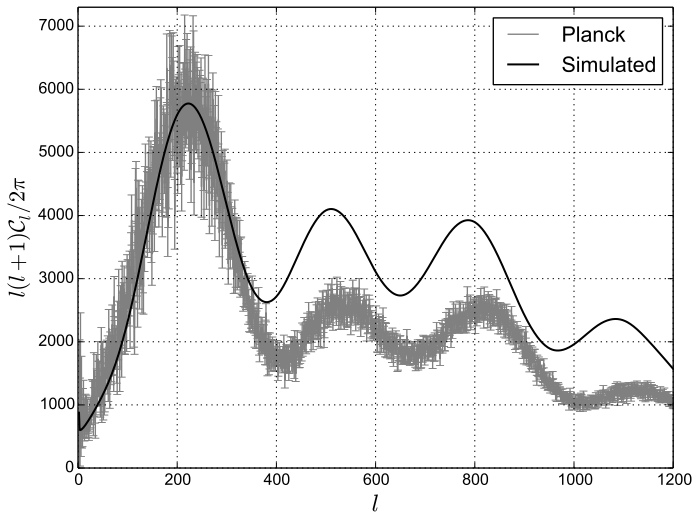
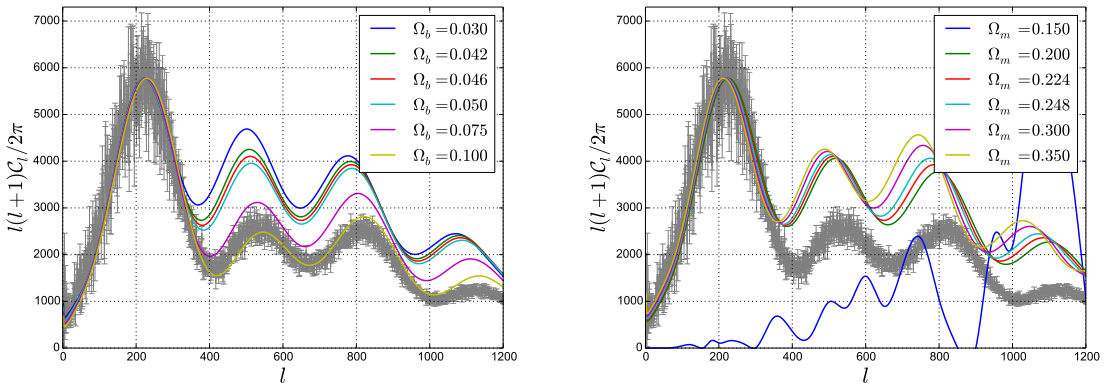


Figure 4: The CMB power spectrum plotted for the default parameters.

We see that the general shape of our power spectrum is the same as the Planck data. For small l, up until the end of the first peak, our power spectrum matches the data very well, but for larger l, all the values are too high. However, as we simply have simply normalized the power spectrum so that the max value is 5775 μK^2, the first peak would necessarily be the correct height. It is worth noting is that the first peak occurs at correct l, and is not shifted. As our model does not include neutrinos or polarization (and our Universe does), this level of discrepancy is probably what we should expect.

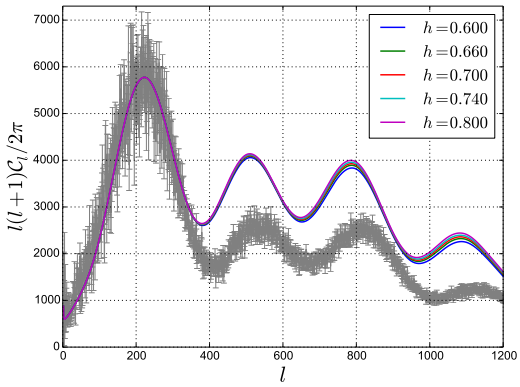
4.3 Parameter estimation

Since our power spectrum is outside of the Planck error bars for a large number of l's, it could be interesting to try to change the parameters for a better fit, which also gives us a way to see how some of the different cosmological parameters affect the CMB power spectrum. In this milestone, we will be testing for different values of the baryonic density Ω_b, cold dark matter density Ω_m, Hubble constant h, and spectral index n_s.

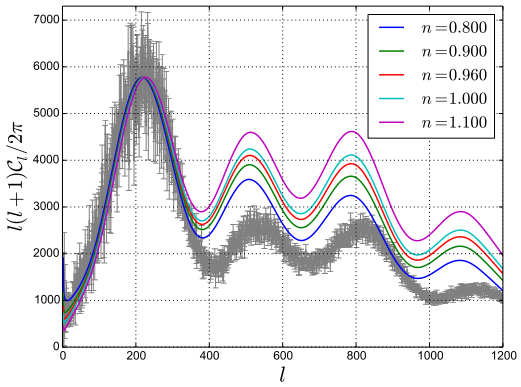


(a) Varying Ω_b

(b) Varying Ω_m



(c) Varying h



(d) Varying n_s

Figure 5: Changing different cosmological parameters in order to see how they affect the CMB power spectrum. All other parameters except the one that is changed have been kept constant at the default values.

From figure 5 we see that changing the baryonic density Ω_b of the universe affects the power spectrum significantly. When increasing Ω_b, we see that the first and third peak gets larger, while the second and fourth peak flattens out. This is what we would expect, as the first and third peak corresponds to compressions, which get stronger for larger baryon density, causing the decompression to be weaker.

Changing the dark matter density Ω_m causes the peaks to be shifted slightly, as well as the height difference between the first and the other peaks. Increasing Ω_b causes the oscillations to be shifted towards smaller l's (more pronounced for large l), as well as increasing the height of the peaks compared to the first peak. The power spectrum with $\Omega_m = 0.150$ should probably be excluded, as something goes wrong, causing the values to be extremely large (if we plotted it non-normalized, it would be many orders of magnitude larger than the others).

Changing the Hubble factor h doesn't do much beyond a small decrease in C_l for larger l when h is decreased. Finally, we look at the spectral index n_s. Increasing n_s causes the peaks to grow, as well as shift very slightly towards larger l.

4.4 Final model

We find a best-fit model by using the information in figure 5 to adjust the parameters by hand until we are satisfied. Ideally, using the Metropolis algorithm to estimate the parameters should have been done, but time constraints makes implementing that difficult. The parameters in our final model can be found in the table below.

Table 1: Table of the cosmological parameters for both default model and best-fit model. The parameters not included here are unchanged from the default `params.f90` file.

	Default	Best-fit
Ω_b	0.046	0.082
Ω_m	0.224	0.200
h	70	60
n_s	0.96	0.82

Finally, we have the CMB power spectrum of our best-fit model.

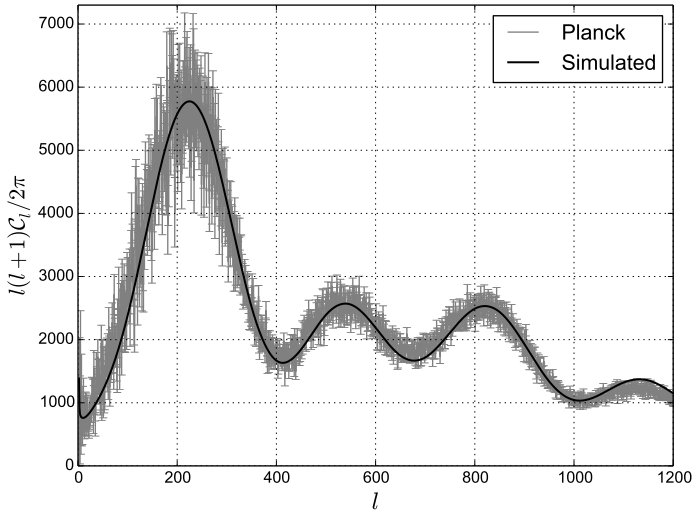


Figure 6: The best-fit power spectrum using the parameters in table 1.

We see that our best model fits the data much better, as the power spectrum is inside of the Planck error bars for all scales l. Looking at our parameters however, considering we had to almost

double the baryon density of the Universe, the end parameters aren't exactly representative of our Universe, but again, as our simulation does not contain any neutrinos nor polarization, we should still be satisfied with what has been achieved.

5 Program listing

The program code used in this project is appended below, as well as available on GitHub [4].

5.1 evolution_mod.f90 - get_hires_source_function

```
1  ! NB!!! New routine for 4th milestone only; disregard until then!!!
2  subroutine get_hires_source_function(k, x, S)
3    implicit none
4
5    real(dp), allocatable, dimension(:),   intent(inout) :: k, x
6    real(dp), allocatable, dimension(:,:), intent(inout) :: S
7
8    integer(i4b) :: i, j, n_hires, ii
9    real(dp)     :: g, dg, ddg, tau, dt, ddt, H_p, dH_p, ddHH_p, Pi, dPi, ddPi, xc, kc
10   real(dp), allocatable, dimension(:,:)     :: S_lores
11   real(dp), allocatable, dimension(:,:,:,:) :: S_coeff
12
13   ! Task: Output a pre-computed 2D array (over k and x) for the
14   !       source function, S(k,x). Remember to set up (and allocate) output
15   !       k and x arrays too.
16   !
17   allocate(S_lores(1000, n_k))
18   allocate(S_coeff(4,4,1000,n_k))  ! okay.....
19
20   n_hires = size(x)
21   do i = 1, n_hires  ! create hires x,k arrays
22      x(i) = x_t2(501) + (i-1.d0)*(0.d0 - x_t2(501))/(n_hires-1.d0) ! change 501 to 1 if i
           revert to x_t
23      k(i) = k_min + (k_max - k_min)*((i-1.d0)/(n_hires-1.d0))!**2
24   end do
25
26   ! Substeps:
27   !   1) First compute the source function over the existing k and x
28   !      grids
29   do i = 1, n_t!500!n_t  !  STICK AROUND
30      ii    = i !500+i ! why did i ever expand the x-grid :/
31      xc    = x_t2(i)
32      g     = get_g(xc)
33      dg    = get_dg(xc)
34      ddg   = get_ddg(xc)
35      tau   = get_tau(xc)
36      dt    = get_dtau(xc)
37      ddt   = get_ddtau(xc)
```

```
38        H_p   = get_H_p(xc)
39        dH_p  = get_dH_p(xc)
40
41        do j = 1, n_k
42           kc    = ks(j)
43           Pi    = Theta(ii,2,j)
44           dPi   = dTheta(ii,2,j)
45
46           ! new attempt as per cmbspec_eqns document (with typos fixed) D:
47           ddPi  = 2.d0*c*kc/(5.d0*H_p)*(-dH_p/H_p*Theta(ii,1,j) + dTheta(ii,1,j)) + 3.d0/10.
                 d0*(ddt*Pi+dt*dPi) - 3.d0*c*kc/(5.d0*H_p)*(-dH_p/H_p*Theta(ii,3,j) + dTheta(ii
                 ,3,j))
48
49           ddHH_p = H_0**2/2.d0*( (Omega_b+Omega_m)*exp(-xc) + 4.d0*Omega_r*exp(-2.d0*xc) + 4.
                 d0*Omega_lambda*exp(2.d0*xc) )
50
51           S_lores(i,j) = g*(Theta(ii,0,j)+Psi(ii,j)+Pi/4.d0) + exp(-tau)*(dPsi(ii,j)-dPhi(ii,
                 j)) - 1.d0/(c*kc)*(dH_p*g*v_b(ii,j)+H_p*dg*v_b(ii,j)+H_p*g*dv_b(ii,j)) + 3.d0
                 /(4.d0*c**2*kc**2)*(ddHH_p*g*Pi + 3.d0*H_p*dH_p*(dg*Pi+g*dPi) + H_p**2*(ddg*Pi
                 +2.d0*dg*dPi+g*ddPi))
52        end do
53     end do  !  CHILL
54
55     !   2) Then spline this function with a 2D spline
56     call splie2_full_precomp(x_t2, ks, S_lores, S_coeff)
57
58     !   3) Finally, resample the source function on a high-resolution uniform
59     !      5000 x 5000 grid and return this, together with corresponding
60     !      high-resolution k and x arrays
61     do i=1, n_hires
62        do j=1, n_hires
63           S(i,j) = splin2_full_precomp(x_t2, ks, S_coeff, x(i), k(j))
64        end do
65     end do
66
67   end subroutine get_hires_source_function
```

5.2 cl_mod.f90

```
1  module cl_mod
2    use healpix_types
3    use evolution_mod
4    use sphbess_mod
5    implicit none
6
7  contains
```

```
8
9   ! Driver routine for (finally!) computing the CMB power spectrum
10  subroutine compute_cls
11    implicit none
12
13    integer(i4b) :: i, j, l, l_num, x_num, n_spline, n_hires, n_cl, m, i_rec ! HEY CHRISTMAS
          TREE
14    real(dp)     :: dx, S_func, j_func, z, eta, eta0, x0, x_min, x_max, d, e
15    integer(i4b), allocatable, dimension(:)       :: ls
16    real(dp),     allocatable, dimension(:)       :: integrand
17    real(dp),     allocatable, dimension(:,:)     :: j_l, j_l2
18    real(dp),     allocatable, dimension(:)       :: x_arg, int_arg, cls, cls2, ls_dp
19    real(dp),     allocatable, dimension(:)       :: k, x
20    real(dp),     allocatable, dimension(:,:,:,:) :: S_coeff
21    real(dp),     allocatable, dimension(:,:)     :: S, S2
22    real(dp),     allocatable, dimension(:,:)     :: Theta
23    real(dp),     allocatable, dimension(:)       :: z_spline, j_l_spline, j_l_spline2
24    real(dp),     allocatable, dimension(:)       :: x_hires, k_hires, cls_hires, ls_hires,
          x_lores
25
26    real(dp)           :: t1, t2, integral
27    logical(lgt)       :: exist
28    character(len=128) :: filename
29    real(dp), allocatable, dimension(:) :: y, y2
30
31    ! Set up which l's to compute
32    l_num = 44
33    allocate(ls(l_num))
34    ls = (/ 2, 3, 4, 6, 8, 10, 12, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100, &
35         & 120, 140, 160, 180, 200, 225, 250, 275, 300, 350, 400, 450, 500, 550, &
36         & 600, 650, 700, 750, 800, 850, 900, 950, 1000, 1050, 1100, 1150, 1200 /)
37
38    ! Task: Get source function from evolution_mod
39    n_hires = 5000
40    allocate(x_hires(n_hires))
41    allocate(k_hires(n_hires))
42    allocate(S(n_hires,n_hires))
43
44    call get_hires_source_function(k_hires, x_hires, S)
45
46    ! Task: Initialize spherical Bessel functions for each l; use 5400 sampled points between
47    !       z = 0 and 3500. Each function must be properly splined
48    ! Hint: It may be useful for speed to store the splined objects on disk in an unformatted
49    !       Fortran (= binary) file, so that these only has to be computed once. Then, if
          your
50    !       cache file exists, read from that; if not, generate the j_l's on the fly.
51    n_spline = 5400
52    allocate(z_spline(n_spline))    ! Note: z is *not* redshift, but simply the dummy
          argument of j_l(z)
53    allocate(j_l(n_spline, l_num))
```

```
54      allocate(j_l2(n_spline, l_num))
55      allocate(integrand(n_hires))
56
57      ! Initializing z array
58      do i = 1, n_spline
59         z_spline(i) = (i-1.d0)*3500.d0/(n_spline-1.d0)
60      end do
61
62      ! Checking for binary file
63      filename = 'j_l.bin'
64      inquire(file=filename, exist=exist)
65      if (exist) then
66         open(10, form='unformatted', file=filename)
67         read(10) j_l, j_l2
68         close(10)
69      else
70         ! Computing spherical Bessel functions
71         do l = 1, l_num
72            do i = 1, n_spline
73               if (z_spline(i) > 0.d0) then ! BECAUSE I'M GOING TO SAY PLEASE
74                  call sphbes(ls(l), z_spline(i), j_l(i,l))
75               end if ! YOU HAVE NO RESPECT FOR LOGIC
76            end do
77         end do
78         ! Splining Bessel functions
79         do l = 1, l_num
80            call spline(z_spline, j_l(:,l), 1.d30, 1.d30, j_l2(:,l))
81         end do
82         ! Write to file
83         open(10, form='unformatted', file=filename)
84         write(10) j_l, j_l2
85         close(10)
86      end if
87
88      ! write integrand to file to check
89      open(1,file='../results/integrand.dat')
90      j = locate_dp(k_hires,340.d0*H_0/c) ! locating 340*H_0/c in k array
91      l = 17
92      do i = 1, n_hires
93         integrand(i) = S(i,j)*splint(z_spline,j_l(:,l),j_l2(:,l), k_hires(j)*(get_eta(0.d0)-
               get_eta(x_hires(i))))
94         write(1,*) integrand(i)
95      end do
96
97      ! Overall task: Compute the C_l's for each given l
98      allocate(Theta(l_num,n_hires))
99      allocate(int_arg(n_hires))
100     allocate(cls(l_num))
101     allocate(cls2(l_num))
102     allocate(x_lores(n_hires/10)) ! creating low-res x grid for fast Theta-integration
```

```
103
104     ! locate end of recombination
105     i_rec = locate_dp(x_hires, -log(1.d0+614.2d0))
106 !    write(*,*) i_rec
107  !   stop
108     do l = 1, l_num
109        ! Task: Compute the transfer function, Theta_l(k)
110        do j = 1, n_hires
111           ! Compute integrand for current k
112           do i = 1, n_hires/10
113              if (i<=300) then
114                 m = 1 + (i-1)*(i_rec-1)/(300-1) ! i just wanna speed up the integration
115              end if
116              if (i>=300) then
117                 m = i_rec + (i-301)*(n_hires-i_rec)/(199)
118              end if
119
120              x_lores(i) = x_hires(m)
121              integrand(i)=S(m,j)*splint(z_spline,j_l(:,l),j_l2(:,l),k_hires(j)*(get_eta(0.d0)
                     -get_eta(x_hires(m))))
122           end do
123
124           ! integrate with trapezoidal, maybe improve later    EDIT: no
125           call trapz(x_lores, integrand(1:500), Theta(l,j))
126
127        end do
128
129        ! Task: Integrate P(k) * (Theta_l^2 / k) over k to find un-normalized C_l's
130        do j = 1, n_hires
131           int_arg(j) = (c*k_hires(j)/H_0)**(n_s-1.d0)*Theta(l,j)**2/k_hires(j)
132        end do
133        call trapz(k_hires, int_arg, integral)
134
135        ! Task: Store C_l in an array. Optionally output to file
136        cls(l) = integral*ls(l)*(ls(l)+1.d0)/(2.d0*pi)
137
138        write(*,*) ls(l), cls(l) ! writing mostly so I know how far the program has come
139
140        ! write C_l integrand to file
141        if (ls(l) == 2) then
142           call write_cl_int(ls(l), c*k_hires/H_0, ls(l)*(ls(l)+1.d0)*Theta(l,:)**2/(c*k_hires
                  /H_0))
143        else if (ls(l) == 50) then
144           call write_cl_int(ls(l), c*k_hires/H_0, ls(l)*(ls(l)+1.d0)*Theta(l,:)**2/(c*k_hires
                  /H_0))
145        else if (ls(l) == 200) then
146           call write_cl_int(ls(l), c*k_hires/H_0, ls(l)*(ls(l)+1.d0)*Theta(l,:)**2/(c*k_hires
                  /H_0))
147        else if (ls(l) == 500) then
```

```
148           call write_cl_int(ls(l), c*k_hires/H_0, ls(l)*(ls(l)+1.d0)*Theta(l,:)**2/(c*k_hires
                  /H_0))
149        else if (ls(l) == 800) then
150           call write_cl_int(ls(l), c*k_hires/H_0, ls(l)*(ls(l)+1.d0)*Theta(l,:)**2/(c*k_hires
                  /H_0))
151        else if (ls(l) == 1200) then
152           call write_cl_int(ls(l), c*k_hires/H_0, ls(l)*(ls(l)+1.d0)*Theta(l,:)**2/(c*k_hires
                  /H_0))
153        end if
154
155     end do
156
157     ! Task: Spline C_l's found above, and output smooth C_l curve for each integer l
158     n_cl = ls(l_num)-1 ! 1999
159     allocate(ls_dp(l_num))
160     allocate(ls_hires(ls(l_num)-1)) ! 1199 different l's for unit stepsiez
161     allocate(cls_hires(ls(l_num)-1))
162
163     do l = 1, l_num
164        ls_dp(l) = ls(l) ! spline/splint requires dp array as input
165     end do
166
167     call spline(ls_dp, cls, 1.d30, 1.d30, cls2) ! spline
168     do i = 1, ls(l_num)-1
169        ls_hires(i)  = ls_dp(1) + (i-1.d0)*(ls_dp(l_num)-ls_dp(1))/(ls_dp(l_num)-2.d0) ! -2
               cause reasons
170        cls_hires(i) = splint(ls_dp, cls, cls2, ls_hires(i))
171     end do
172
173     ! Write C_l's to file
174     open(20,file='../results/cl.dat')
175     write(20,'(5E20.5)') n_s, h0, Omega_m, Omega_b, Omega_r
176     do i = 1, ls(l_num)-1
177        write(20,'(2E20.8)') ls_hires(i), cls_hires(i)
178     end do
179     close(20)
180
181  end subroutine compute_cls
182
183
184  subroutine trapz(x,y,integral) ! Its a trap!
185    implicit none
186    real(dp), dimension(:), intent(in)  :: x,y
187    real(dp),               intent(out) :: integral
188    integer(i4b)                        :: n, i
189    real(dp)                            :: h
190
191    if (size(x) .ne. size(y)) then
192       write(*,*) 'x and y does not have same shape'
193       return
```

```
194     end if
195
196     integral = 0.d0
197     n = size(x)
198     h = (x(n) - x(1))/n ! apparently k was supposed to be evenly distributed
199     do i=1, n-1
200        integral = integral + h*(y(i+1)+y(i))/2.d0
201     end do
202
203   end subroutine trapz
204
205   ! actually working now
206   subroutine write_cl_int(l, ckH0, cls_int)
207     implicit none
208     integer(i4b),            intent(in) :: l
209     real(dp), dimension(:),  intent(in) :: ckH0, cls_int
210     integer(i4b)                        :: n, i
211     character(len=128)                  :: filename, str1, str2, str3
212     if (size(ckH0) .ne. size(cls_int)) then
213        write(*,*) 'ckH0 and cls_int does not have same shape'
214        return
215     end if
216     n = size(ckH0)
217
218     ! Creating filename (cls_int_<l>.dat)
219     str1 = '../results/cls_int_'
220     write(str2,'(I5.5)') l
221     str3 = '.dat'
222     filename = trim(str1)//trim(str2)//trim(str3)
223
224     open(30,file=filename)
225     do i = 1, n
226        write(30,'(*(2X, ES14.6E3))') ckH0(i), cls_int(i)
227     end do
228     close(30)
229   end subroutine write_cl_int
230
231 end module cl_mod
```

References

[1] P. Callin 2006: *How to calculate the CMB spectrum*

[2] Milestone 3: The evolution of structures in the universe

[3] Milestone 4: The CMB power spectrum

[4] https://github.com/simennb/ast5220/tree/master/mk4/src