Milestone 2

Elisabeth Strøm[1]
AST5220 18.02.2018

[1] Institute of Theoretical Astrophysics, University of Oslo

hei

1. Introduction

2. Method

Here we present the methods we have used to obtain the results in Section 3.

2.1. Theory

The optical depth τ quantifies the probability of scattering a photon between some earlier time, and present day. It can be defined as

$$\tau(\eta) = \int_{\eta}^{\eta_0} n_e \sigma_T a \mathrm{d}\eta', \tag{1}$$

where η is the conrformal time and a subscript 0 means its value today. The n_e is the number density of free electrons, σ_T is the Thomson cross-section, and a is the scale factor. As an initial condition, we use that at present day, $\tau = 0$.

As in the previous project we will in our calculations use

$$x = \ln a \tag{2}$$

instead of a. We will also have need of the redshift,

$$z = \frac{1}{a} - 1 = e^{-x} - 1. \tag{3}$$

On a differential form, τ can be written as

$$\tau'(x) = -\frac{n_e \sigma_T a}{\mathcal{H}} = -\frac{n_e \sigma_T}{H}, \tag{4}$$

where H is the Hubble parameter defined in the previous project.

The visibility function is defined as

$$g(\eta) = -\dot{\tau} e^{-\tau(\eta)} = \mathcal{H}\tau' e^{-\tau(x)} = g(x). \tag{5}$$

However, what we will compute is

$$\tilde{g}(x) = -\tau' e^{-\tau} = \frac{g(x)}{\mathcal{H}}, \tag{6}$$

which will be referred to as the visibility function from here on out. It is a probability distribution and describes the probability density for a given photon to scatter at time x, meaning

$$\int_0^{\eta_0} g(\eta)\mathrm{d}\eta = \int_{-\infty}^{0} \tilde{g}(x)\mathrm{d}x = 1. \tag{7}$$

What we need to do next, is to calculate the electron number density n_e. We need two equations to be able to do this, namely the Saha equation, and the Peebles equation. From them we find the free electron fraction X_e, and thereby n_e through the relation

$$n_e = X_e n_b, \tag{8}$$

where n_b is the number density of baryons.

We are assuming that the baryons in the universe are hydrogen atoms, making their number density

$$n_H = n_b = \frac{\rho_b}{m_H} = \frac{\Omega_b \rho_c}{m_H a^3}, \tag{9}$$

where ρ_b is the baryon matter density and Ω_b is the density parameter of baryons today. $\rho_c = \frac{3H^2}{8\pi G} = 9.206 \cdot 10^{-27}$ is the critical density of the Universe and m_H is the hydrogen mass.

The Saha equation is a good approximation around $X_e \approx 1$, and we will use it for $X_e > 0.99$. It is defined as

$$\frac{X_e^2}{1 - X_e} = \frac{1}{n_b}\left(\frac{m_e k_b T_b}{2\pi\hbar^2}\right)^{3/2} e^{-\epsilon_0/(k_b T_b)} \tag{10}$$

where m_e is the electron mass and T_b is the baryon temperature of the Universe. It is a good approximation ot put this equal to the photon temperature, T_r, so that $T_b = T_r = T_0/a$, where $T_0 = 2.725$ K. k_b is the Boltzmann constant and $\epsilon_0 = 13.6e$V is the ionization energy of hydrogen. This is a standard second order polynomial with the solution

$$X_e(x) = \frac{-X_{e0} + \sqrt{X_{e0}^2 + 4X_{e0}}}{2}, \tag{11}$$

$$X_{e0} = \frac{1}{n_b}\left(\frac{m_e k_b T_b}{2\pi\hbar^2}\right)^{2/3} e^{-\epsilon_0/(k_b T_b)}. \tag{12}$$

For smaller values of X_e, we require a more accurate approximation such a the Peebles equation. We use it when $X_e < 0.99$, and it is defined as

$$\frac{\mathrm{d}X_e}{\mathrm{d}x} = \frac{C_r(T_b)}{H}\left[\beta(T_b)(1 - X_e) - n_H \alpha^{(2)}(T_b) X_e^2\right], \tag{13}$$

where

$$
\begin{aligned}
C_r(T_b) &= \frac{\Lambda_{2s\to1s} + \Lambda_\alpha}{\Lambda_{2s\to1s} + \Lambda_\alpha + \beta^{(2)}(T_b)}, && (14)\\
\Lambda_{2s\to1s} &= 8.227\ \mathrm{s}^{-1}, && (15)\\
\Lambda_\alpha &= \frac{H}{(c\hbar)^3}\frac{(3\epsilon_0)^3}{(8\pi)^2 n_{1s}} && (16)\\
n_{1s} &= (1 - X_e)n_H, && (17)\\
\beta^{(2)}(T_b) &= \beta(T_b)e^{3\epsilon_0/(4k_bT_b)}, && (18)\\
\beta(T_b) &= \alpha^{(2)}(T_b)\left(\frac{m_ek_bT_b}{2\pi\hbar^2}\right)^{3/2} e^{-\epsilon_0/(k_bT_b)}, && (19)\\
\alpha^{(2)}(T_b) &= \frac{64\pi}{\sqrt{27\pi}}\frac{\alpha^2\hbar^2}{m_e^2c}\sqrt{\frac{\epsilon_0}{k_bT_b}}\phi_2(T_b), && (20)\\
\phi_2(T_b) &= 0.448\ln\left(\frac{\epsilon_0}{k_bT_b}\right). && (21)
\end{aligned}
$$

Here, α is the fine structure constant.

2.2. Implementation

As in Milestone 1, we use the skeleton structure of a program code that is provided us. The first thing we do is define a x-grid, called `x_rec` in out code. The x-grid consists of $n = 1000$ points, and the difference between two neighboring points, is given as

$$dx = \frac{x_{\mathrm{stop}} - x_{\mathrm{start}}}{n-1} \tag{22}$$

where $x_{\mathrm{stop}} = 0$ og $x_{\mathrm{start}} = \ln\left(10^{-10}\right)$ are the values of x at the end and the beginning of our time frame, respectively. That is, from the very early days of the universe when the scale factor was small, up until today. The x-grid can now be defined as

$$x_{i+1} = x_i + dx = x_{\mathrm{start}} + i\cdot dx. \tag{23}$$

We find X_e by calculating the Saha equation (Equation 11) and solving the Peebles equation (Equation 13). We do this in a loop, using an `if` test to make sure that we are using the proper equation for the correct value of X_e. We solve the Peebles Equation by using an ODE solver, where we also use that the interval between two values is $dx/100$. We can now calculate n_e using Equation 8. We then spline the natural logarithm of these values,

Next we solve Equation 4 to find the optical depth τ, also here by using the same ODE solver. We spline both the τ and its second derivative by using the `spline` command.

The visibility function is found by calculating Equation 6, which we again spline, along with its second derivative.

The last thing we do is create subroutines, that, using the splined values of n_e, τ, τ'', $\tilde{g}$, and $\tilde{g}''$, can find these parameter values for any other x, using the `splint` command. We also find the splined first derivatives of both $\tau(x)$ and $\tilde{g}(x)$ by using the command `splint_deriv`, which takes τ or $\tilde{g}$ and x as arguments, and returns $\tau'(x)$ or $\tilde{g}'(x)$.

3. Results

The free electron fraction X_e can be seen in Figure 1. We see that it looks similar to that produced by Callin (2006). The Saha equation is used up until a redshift of $z = 1559$, at which point the Peebles equation takes over. We see that when this occurs, $X_e(z)$ starts decreasing, and does not stop decreasing right up until present day, although the slope becomes less steep around z 700. Today, the electron fraction have the value $X_e = 1.95\cdot10^{-4}$.

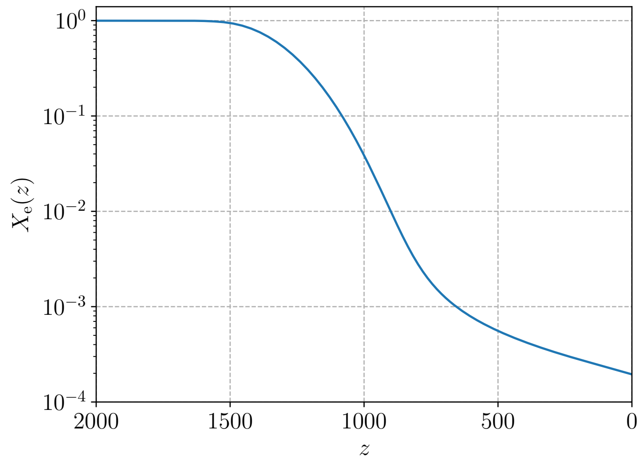

Fig. 1: The figure shows the free electron fraction X_e as a function of redshift z. The y-axis is log scaled. The Saha approximation (Equation 11) is used up until $X_e < 0.99$, corresponding to a redshift of z=1559. After this the Peebles equation (Equation 13) is used.

In Figure 2, we see the optical depth τ, along with the absolute values of its first and second derivative, $|\tau'|$, and $|tau''|$. This too matches the result of Callin (2006). The optical depth was larger in the past, but steadily decreasing. At $x =$ it experiences a sudden drop in optical depth, before steadily decreasing again. Close to our own time, $x = 0$, it seems to be undergoing a sudden drop again.

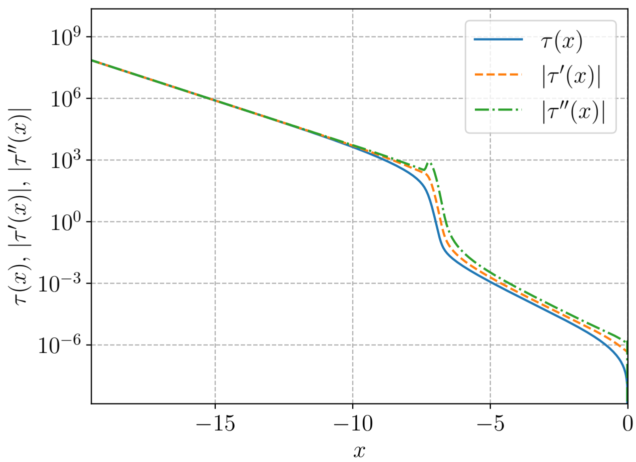

Fig. 2: The optical depth, $\tau(x)$, along with the absolute value of its first and second derivatives, $|\tau'(x)|$, $|\tau''(x)|$ are on the log scaled y-axis. They are a function of the logarithm of the scale factor, $x = \ln a$, on the x-axis.

4. Conclusions

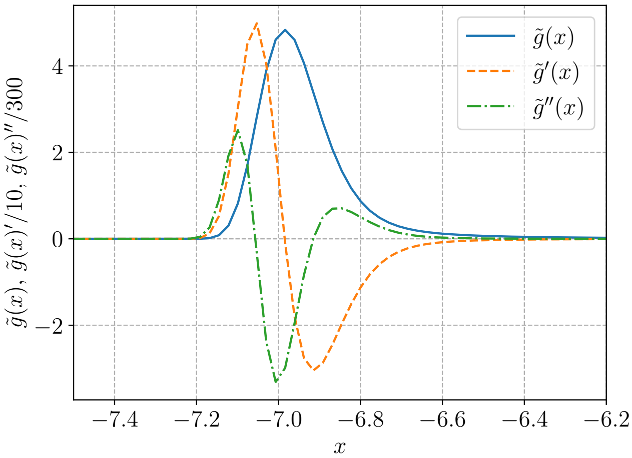

Fig. 3: The visibility function $\tilde{g}(x)$, and its first and second derivatives $\tilde{g}'(x)\,\tilde{g}''(x)$ can be seen along the y-axis as a function of $x = \ln a$ along the x-axis. The derivatives have been scaled, so that what is shown in the figure is $\frac{\tilde{g}'}{<}10$ and $\frac{\tilde{g}''}{300}$. This scaling is chosen in accordance with Callin (2006), to better resemble their Figure 2.

References

Callin, P. 2006, ArXiv Astrophysics e-prints

5. Appendix

Source code

Listing 1: C:/Users/elini/Documents/AST5220/Ast5220/src/rec_mod.f90

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

  integer(i4b),                      private :: n              ! Number of grid points
  real(dp), allocatable, dimension(:), private :: x_rec,x_rec1   ! Grid
  real(dp), allocatable, dimension(:), private :: a_rec    ! Grid

  real(dp), allocatable, dimension(:), private :: tau, tau2, tau22 ! Splined tau and second derivatives
  real(dp), allocatable, dimension(:), private :: n_e, n_e2 ! Splined (log of) electron density, n_e
  real(dp), allocatable, dimension(:), private :: g, g2, g22 ! Splined visibility function

contains

  subroutine initialize_rec_mod
    implicit none

    integer(i4b) :: i, j, k
    real(dp)    :: saha_limit, y, T_b, n_b, dydx, xmin, xmax, dx, f, n_e0, X_e0, xstart, xstop
    logical(lgt) :: use_saha
    real(dp), allocatable, dimension(:) :: X_e ! Fractional electron density, n_e / n_H
    real(dp)    :: step, stepmin, eps, z
    !real(dp), dimension(1) :: y

    saha_limit = 0.99d0    ! Switch from Saha to Peebles when X_e < 0.99
    xstart     = log(1.d-10) ! Start grids at a = 10^-10
    xstop      = 0.d0        ! Stop grids at a = 1
    n          = 1000        ! Number of grid points between xstart and xstopo

    eps        = 1.d-10       ! spline error limit

    allocate(x_rec(n))
    allocate(x_rec1(n))
    allocate(X_e(n))
    allocate(tau(n))
    allocate(tau2(n))
    allocate(tau22(n))
    allocate(n_e(n))
    allocate(n_e2(n))
    allocate(g(n))
    allocate(g2(n))
    allocate(g22(n))

    ! Task: Fill in x (rec) grid
    write(*,*) "making x grid"
    x_rec(1) = xstart
    dx   = (xstop-xstart)/(n-1)

    do i = 2, n
       !x_rec(i) = x_rec(i-1) + dx
       !write(*,*) "1", x_rec1(i+1)-x_rec1(i)
       x_rec(i) = xstart + (i-1)*dx
       !write(*,*) "2", x_rec(i+1)-x_rec(i)
    end do


    ! Task: Compute X_e and n_e at all grid times
    step      = abs((x_rec(1) - x_rec(2))*1.d-3) ! n-1 maybe integration step length
    stepmin   = 0.d0
```

```fortran
   write(*,*) "calculating X_e"
   use_saha = .true.
   do i = 1, n
      write(*,*) "loop ", i
      n_b = Omega_b*rho_c/(m_H*exp(x_rec(i))**3.d0)
      if (use_saha) then
         ! Use the Saha equation
         T_b = T_0/exp(x_rec(i))
         X_e0 = 1.d0/n_b*((m_e*k_b*T_b)/(2.d0*pi*hbar**2))**(1.5d0) * exp(-epsilon_0/(k_b * T_b))
         X_e(i)= (-X_e0 + sqrt(X_e0**2.d0 + 4.d0*X_e0))/2.d0
         if (X_e(i) < saha_limit) use_saha = .false.
      else
         ! Use the Peebles equation
         ! write(*,*) X_e(i)
         X_e(i) = X_e(i-1)
         call odeint(X_e(i:i), x_rec(i-1), x_rec(i), eps, step, stepmin, dXe_dx, bsstep, output)

      end if
      n_e(i) =X_e(i)*n_b
   end do


   ! Task: Compute splined (log of) electron density function
   n_e = log(n_e)
   write(*,*) "splining ne"
   call spline(x_rec, n_e, 1.d30, 1.d30, n_e2)

   ! Task: Compute optical depth at all grid points
   write(*,*) "calculating tau"

   tau(n) = 0.d0 ! initial condition, present day value
   do i = n-1, 1,-1
     tau(i) = tau(i+1)
     call odeint(tau(i:i), x_rec(i+1), x_rec(i), eps, step, stepmin, dtau_dx, bsstep, output)
   end do


   ! Task: Compute splined (log of) optical depth
   write(*,*) "splining tau and ddtau"
   call spline(x_rec, tau, 1.d30,1.d30, tau2)

   ! Task: Compute splined second derivative of (log of) optical depth
   call spline(x_rec, tau2,1.d30,1.d30,tau22)

!------------- visibility function g ---

   write(*,*) "calculating g"
   do i=1, n
     g(i) = -get_dtau(x_rec(i)) * exp(-tau(i))
   end do

   ! Task: Compute splined visibility function
   write(*,*) "splining g and ddg"
   call spline(x_rec, g, 1.d30, 1.d30, g2)

   ! Task: Compute splined second derivative of visibility function
   call spline(x_rec, g2, 1.d30, 1.d30, g22)


!---------- write to file ---
   write(*,*) "opening files "
   open (unit=1, file = 'x_tau.dat', status='replace')
   open (unit=2, file = 'x_g.dat', status='replace')
   open (unit=3, file = 'x_z_Xe.dat', status='replace')

   write(*,*) "writing stuff"
   do i=1, n
     z = exp(-x_rec(i))-1
     write (1,*) tau(i), get_dtau(x_rec(i)), get_ddtau(x_rec(i))
```

```
     write (2,*) g(i), get_dg(x_rec(i)), get_ddg(x_rec(i))
     write (3,*) x_rec(i), z, X_e(i)

   end do

   write(*,*) " closing files "
   do i=1,3 ! close files
     close(i)
   end do

  end subroutine initialize_rec_mod


!---------------------- Peebles equation ----

  subroutine dXe_dx(x, X_e, dydx)
   ! we define dy/dx
   use healpix_types
   implicit none
   real(dp),             intent(in) :: x
   real(dp), dimension(:), intent(in) :: X_e
   real(dp), dimension(:), intent(out) :: dydx
   real(dp) :: beta, beta2, alpha2, n_b, n1s, lambda_2s1s, lambda_alpha
   real(dp) :: C_r, T_b, H, phi2


   H   = get_H(x)
   !write(*,*) "H"
   T_b  = T_0/exp(x)
   n_b  = Omega_b*rho_c/(m_H*exp(3.d0*x))

   phi2 = 0.448d0*log(epsilon_0/(k_b * T_b))
   alpha2 = 64.d0*pi/sqrt(27.d0*pi) *(alpha/m_e)**2 *sqrt(epsilon_0/(k_b * T_b)) *phi2 *hbar*hbar/c
   beta = alpha2*((m_e*k_b*T_b)/(2.d0*pi*hbar*hbar))**(1.5d0) * exp(-epsilon_0/(k_b * T_b))

   ! To avoid beta2 going to infinity, set it to 0
   if(T_b <= 169.d0) then
      beta2 = 0.d0
   else
      beta2 = beta * exp(3.d0*epsilon_0/(4.d0 * k_b*T_b))
   end if
   n1s = (1.d0 - X_e(1))* n_b ! X_e(1)

   lambda_alpha = H * (3.d0*epsilon_0)**3/((8.d0*pi)**2 * n1s)/(c*hbar)**3
   lambda_2s1s = 8.227d0

   C_r = (lambda_2s1s + lambda_alpha)/(lambda_2s1s + lambda_alpha + beta2)

   dydx = C_r/H * (beta * (1.d0-X_e(1)) - n_b * alpha2 * X_e(1)**2)

  end subroutine dXe_dx


  subroutine dtau_dx(x, tau, dydx)
   ! we define dy/dx
   use healpix_types
   implicit none
   real(dp),             intent(in) :: x
   real(dp), dimension(:), intent(in) :: tau
   real(dp), dimension(:), intent(out) :: dydx

   dydx = -get_n_e(x) * sigma_T * c/get_H(x)

  end subroutine dtau_dx



  ! Task: Complete routine for computing n_e at arbitrary x, using precomputed information
  ! Hint: Remember to exponentiate...
  function get_n_e(x)
```

```fortran
  implicit none

  real(dp), intent(in) :: x
  real(dp)          :: get_n_e
  ! n_e is actually log(n_e)
  get_n_e = splint(x_rec, n_e, n_e2, x)
  get_n_e = exp(get_n_e)

end function get_n_e

! Task: Complete routine for computing tau at arbitrary x, using precomputed information
function get_tau(x)
  implicit none

  real(dp), intent(in) :: x
  real(dp)          :: get_tau

  get_tau = splint(x_rec, tau, tau2, x)

end function get_tau

! Task: Complete routine for computing the derivative of tau at arbitrary x, using precomputed information
function get_dtau(x)
  implicit none

  real(dp), intent(in) :: x
  real(dp)          :: get_dtau

  get_dtau = splint_deriv(x_rec,tau,tau2,x)

end function get_dtau

! Task: Complete routine for computing the second derivative of tau at arbitrary x,
! using precomputed information
function get_ddtau(x)
  implicit none

  real(dp), intent(in) :: x
  real(dp)          :: get_ddtau

  get_ddtau = splint(x_rec, tau2, tau22, x)

end function get_ddtau

! Task: Complete routine for computing the visibility function, g, at arbitray x
function get_g(x)
  implicit none

  real(dp), intent(in) :: x
  real(dp)          :: get_g

  get_g = splint(x_rec, g, g2, x)

end function get_g

! Task: Complete routine for computing the derivative of the visibility function, g, at arbitray x
function get_dg(x)
  implicit none

  real(dp), intent(in) :: x
  real(dp)          :: get_dg

  get_dg = splint_deriv(x_rec, g, g2, x)

end function get_dg

! Task: Complete routine for computing the second derivative of the visibility function, g, at arbitray x
function get_ddg(x)
  implicit none
```

```fortran
    real(dp), intent(in) :: x
    real(dp)           :: get_ddg

    get_ddg = splint(x_rec, g2, g22, x)

  end function get_ddg



end module rec_mod
```