UNIVERSITY OF OSLO

Report for AST9240: The background evolution of the universe

Maksym Brilenkov

March 1, 2019

1 INTRODUCTION

As the goal of the current project is to compute the expansion history of the universe, and look at the uniform background densities of the various matter and energy components, I start with describing the background geometry given by the FRW metric for spatially flat expanding universe

$$ds^2 = -cdt^2 + a^2(t)\delta_{ij}dx^i dx^j = a^2(\eta)\left(d\eta^2 + \delta_{ij}dx^i dx^j\right), \tag{1.1}$$

where t, η are the physical and conformal time, a is a scale factor, c is the speed of light in vacuum. The expansion of the universe is governed by the Friedmann equation, which has the form

$$H(a) = \frac{1}{a}\frac{da}{dt} = H_0\sqrt{(\Omega_m + \Omega_b)a^{-3} + (\Omega_r + \Omega_\nu)a^{-4} + \Omega_\Lambda}, \tag{1.2}$$

$$\mathscr{H}(a) = H_0\sqrt{(\Omega_m + \Omega_b)a^{-1} + (\Omega_r + \Omega_\nu)a^{-2} + \Omega_\Lambda a^2}, \tag{1.3}$$

where H, H_0 is the Hubble parameter at present time, $\mathscr{H} = aH$ is the scaled Hubble parameter, and we assume that the universe consists of cold dark matter (CDM, m), baryons (b), neutrinos (ν), radiation (r), and a cosmological constant (Λ), which are represented by its relative densities, i.e. Ω_m, Ω_b, Ω_ν, Ω_r, Ω_Λ. The evolution of all these components are perfectly described by Friedmann equation (1.2)

$$\rho_m = \rho_{m,0}a^{-3}, \tag{1.4}$$

$$\rho_b = \rho_{b,0} a^{-3}, \tag{1.5}$$

$$\rho_r = \rho_{b,0} a^{-4}, \tag{1.6}$$

$$\rho_\nu = \rho_{\nu,0} a^{-4}, \tag{1.7}$$

$$\rho_\Lambda = \rho_{\Lambda,0}, \tag{1.8}$$

Here, quantities with subscripts 0 indicate today's values.

By introducing logarithm of scale factor - $x = \ln a$ - we can write the relation between appropriate quantities as

$$\frac{dt}{d\eta} = a, \quad \frac{dx}{dt} = H, \quad \frac{dx}{d\eta} = \mathscr{H}, \quad \frac{d}{dt} = H\frac{d}{dx}, \quad \frac{d}{d\eta} = \mathscr{H}\frac{d}{dx}, \tag{1.9}$$

and recalling that the *conformal time* is the distance light may have traveled since the Big Bang ($t = 0$), we define it as

$$\frac{d\eta}{dt} = \frac{c}{a}, \quad \rightarrow \quad \eta(t) = \int_0^t \frac{c\,dt'}{a(t')}, \tag{1.10}$$

or, recalling that $\frac{d}{dt} = \frac{d}{da}\frac{da}{dt}$,

$$\frac{d\eta}{da} = \frac{c}{a^2 H} = \frac{c}{a\mathscr{H}}, \quad \rightarrow \quad \eta(a) = \int_0^a \frac{c\,da'}{a'\mathscr{H}(a')}, \tag{1.11}$$

which is perfectly converge as $a\mathscr{H} \rightarrow H_0\sqrt{\Omega_r + \Omega_\nu}$ for $a \rightarrow 0$. Now, using $da/dx = a$ we can express $\eta(x)$ as follows

$$\eta(x) = \int_{-\infty}^{x} \frac{c\,dx'}{\mathscr{H}(x')}. \tag{1.12}$$

The purpose of this work is to numerically compute and plot the Hubble parameter, conformal time and the cosmological parameters.

thus, this report organized as follows. In sections (2) and (3) I present the methods and algorithms used for numerical computation of $H(x)$, $H(z)$ and $\eta(x)$. In section (4) I introduce plots of resulting functions. The code is listed in the end of this report

2 METHODS

We want to find the values for Hubble parameter, the cosmological parameters and the conformal time as a function of x. For the first two, the conversion is pretty straightforward if we remember the definition of x. Moreover, following the definition of cosmological parameters

$$\Omega_x = \frac{\rho_x}{\rho_{\text{crit}}}, \quad \rho_{\text{crit}} = \frac{3H^2}{8\pi G}, \tag{2.1}$$

and remembering (1.4)-(1.8), we can write

$$\Omega_m = \Omega_{m,0}a^{-3}, \quad \Omega_b = \Omega_{b,0}a^{-3}, \quad \Omega_r = \Omega_{b,0}a^{-4}, \quad \Omega_\nu = \Omega_{\nu,0}a^{-4}, \quad \Omega_\Lambda = \Omega_{\Lambda,0}, \tag{2.2}$$

where, once again, 0 indicates present day values.

However, in order to calculate $\eta(x)$ we need to think of a little trick as when $a \to 0$ the $x \to -\infty$ and it is, in principle, hard to compute. Remembering the convergence criteria discussed in the end of previous section, we can split the integral (1.12) into two and calculate the first one $\eta(x) = \eta(a = e^x)$ up to some small value, a_{small}. In this way, the (1.12) is transformed as follows

$$\eta(x) = \frac{ca_{\text{lim}}}{H_0\sqrt{\Omega_r + \Omega_\nu}} + \int_{\ln(a_{\text{small}})}^{x} \frac{cdx'}{\mathscr{H}(x')}. \tag{2.3}$$

To calculate this numerically we need to know the derivative of η with respect to x. Remembering (1.2) and (1.9), we get

$$\frac{d\eta}{dx} = \frac{c}{\mathscr{H}} = \frac{c}{H_0}\left[(\Omega_m + \Omega_b)a^{-1} + (\Omega_r + \Omega_\nu)a^{-2} + \Omega_\Lambda a^2\right]^{1/2}. \tag{2.4}$$

For the later use, we also need to compute the second derivative of $\eta(x)$

$$\frac{d^2\eta}{dx^2} = -\frac{c}{\mathscr{H}^2}\frac{d\mathscr{H}}{dx}, \tag{2.5}$$

where

$$\frac{d\mathscr{H}}{dx} = \frac{H_0^2}{2\mathscr{H}}\left[2\Omega_\Lambda a^2 - (\Omega_m + \Omega_b)a^{-1} - 2(\Omega_r + \Omega_\nu)a^{-2}\right]. \tag{2.6}$$

In addition, to compute $H(z)$ we invoke

$$1 + z = \frac{a_0}{a(t)}, \quad \to \quad z = \frac{a_0}{\exp(x)} - 1, \tag{2.7}$$

where $a_0 = 1$ is the scale factor at present time.

3 Algorithms

To proceed with numerical computations, we first need to set-up a fine grid during recombination ($n_1 = 200$ points between $z = 1630.4$ and $z = 614.2$), and a coarser grid between the end of recombination and today ($n_2 = 300$ points between $z = 614.2$ and $z = 0$). Thus,

$$\Delta x_1 = \frac{x_{\rm end} - x_{\rm start}}{n_1}, \quad \Delta x_2 = \frac{x_0 - x_{\rm end}}{n_2}, \tag{3.1}$$

where

$$x_{\rm start} = -\ln(1 + z_{\rm start}), \quad x_{\rm end} = -\ln(1 + z_{\rm end}), \quad x_0 = 0, \tag{3.2}$$

Now it is simple to compute the above mentioned parameters using Fortran and the template code provided by H.K. Eriksen. To start, I first copied it from:

```
$ ~hke/AST5220/v14/src/cmbspec
```

The main working files are "params.f90", "time_mod.f90" and "cmbspec.f90". In "params.f90" we choose our cosmological parameters to be

$$\begin{aligned}
\text{Hobble constant:} \quad & h = 0.7, \\
\text{Hobble rate:} \quad & H_0 = h \cdot 100\,\text{km}\,\text{s}^{-1}\,\text{Mpc}^{-1}, \\
\text{CMB temperature:} \quad & T_0 = 2.725\,\text{K}, \\
\Rightarrow \quad & \Omega_r = 5.04 \cdot 10^{-5}, \\
& \Omega_\nu = 8.3 \cdot 10^{-5} - \Omega_r \\
& \quad = 3.26 \cdot 10^{-5}, \\
\text{Baryon density:} \quad & \Omega_b = 0.046, \\
\text{CDM density:} \quad & \Omega_m = 0.224, \\
\text{Vacuum density:} \quad & \Omega_\Lambda = 1 - (\Omega_m + \Omega_b + \Omega_r + \Omega_\nu) \\
& \quad = 0.72992.
\end{aligned}$$

I set-up my grids in "time_mod.f90" by simple looping

```
allocate(x_t(n_t))
dx = (x_end_rec - x_start_rec) / (n1 - 1)
x_t(1) = x_start_rec
do i = 2, n1
    x_t(i) = x_t(1) + dx*(i - 1)
end do
dx = (x_0 - x_end_rec) / n2
do i = 1, n2
    x_t(n1 + i) = x_t(n1) + dx * i
end do

allocate(a_t(n_t))
a_t = exp(x_t)
```

To calculate the Hubble parameters I write several functions as

```
  function get_H(x)
    implicit none

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

    ! As defined by Friedmann equations
    get_H = H_0 * sqrt( (Omega_m + Omega_b) * exp(-3*x) + (Omega_r +
       ↪ Omega_nu) * exp(-4*x) + Omega_lambda)
  end function get_H
```

for H,

```
  function get_H_p(x)
    implicit none

    real(dp), intent(in) :: x
    real(dp)             :: get_H_p
    ! H' = H * a
    get_H_p = get_H(x) * exp(x)
  end function get_H_p
```

for $\mathscr{H}$, and

```
  function get_dH_p(x)
    implicit none

    real(dp), intent(in) :: x
    real(dp)             :: get_dH_p
    ! The derivative of H prime with respect to x
    get_dH_p = H_0**2 / (2 * get_H_p(x)) * ( - (Omega_m + Omega_b) *
       ↪ exp(-2*x) - 2 * (Omega_r + Omega_nu) * exp(-3*x)\
 + 2 * Omega_lambda * exp(x))
  end function get_dH_p
```

for $d\mathscr{H}/dx$.

To compute $\eta(x)$, I also set-up the grid for it as

```
allocate(x_eta(n_eta))
x_eta(1) = x_eta1
dx = (x_eta2 - x_eta1) / (n_eta - 1)
do i = 2, n_eta
    x_eta(i) = x_eta(1) + dx * (i - 1)
end do
```

and then use the ODE solver from "ode_solver.f90" to compute an integral by setting $a_{\rm small} = 10^{-10}$ as follows

```
allocate(eta(n_eta))
! Starting point of integration
eta(1) = c * a_init / (H_0 * sqrt(Omega_r + Omega_nu))
eta_i(1) = eta(1)
h1 = 1.d-6
```

```
h_min = 0.d0
eps = 1.d-6
do i = 2, n_eta
    call odeint(eta_i, x_eta(i - 1), x_eta(i), eps, h1, h_min, derivs1,
       ↪  bsstep, output1)
    eta(i) = eta_i(1)
end do
```

where h1 is the step length, hmin is the minimum length of integration steps, eps is the error of integration, derivs1 is the subroutine which calculates $d\eta/dx$ each time

```
    subroutine derivs1(x, y, dydx)
      use healpix_types
      implicit none
      real(dp),                intent(in)  :: x
      real(dp), dimension(:), intent(in)  :: y
      real(dp), dimension(:), intent(out) :: dydx

      dydx = c / get_H_p(x)
    end subroutine derivs1
```

I am saving the results to the .dat files by modifying "cmbspec.f90". Here is the code example for Ω parameters

```
open(6, file='Omega_all_data.dat', action='write', status='replace')
! The structure is: x Omega_b Omega_m Omega_r Omega_nu Omega_lambda
do i = 1, n_t
    write(6,'(6E20.13)') x_t(i), Omega_b * a_t(i)**(-3), Omega_m * a_t(
       ↪ i)**(-3), Omega_r * a_t(i)**(-4), Omega_nu * \
a_t(i)**(-4), Omega_lambda
end do
close(6)
```

4 Results

Below, I present the plots of $H(x)$, $H(z)$, Ω and $\eta(x)$.

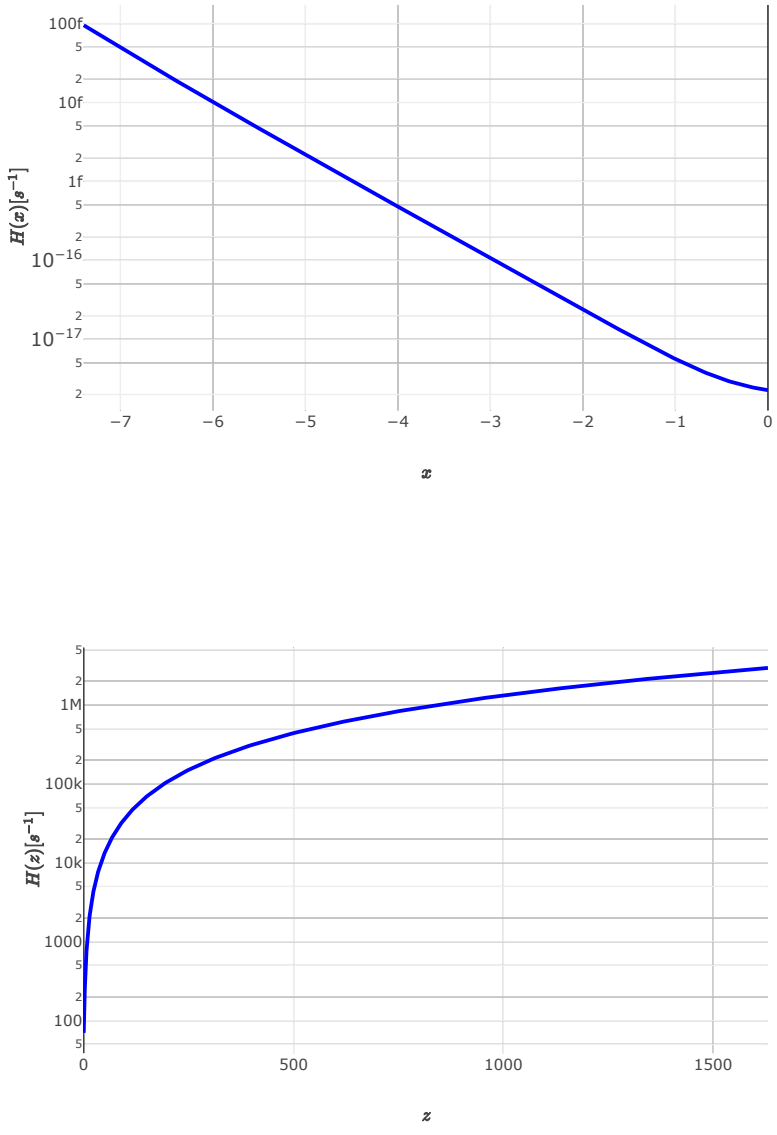

Figure 4.1: Plot of the Hubble rate H as a function of x (*Top*) and z (*Bottom*) from the start of recombination to present time. (Logarithmic scale)

From fig. (4.2) we can see that the universe started being radiation dominated, going over to be matter dominated, and eventually becoming Λ dominated (as it should be).

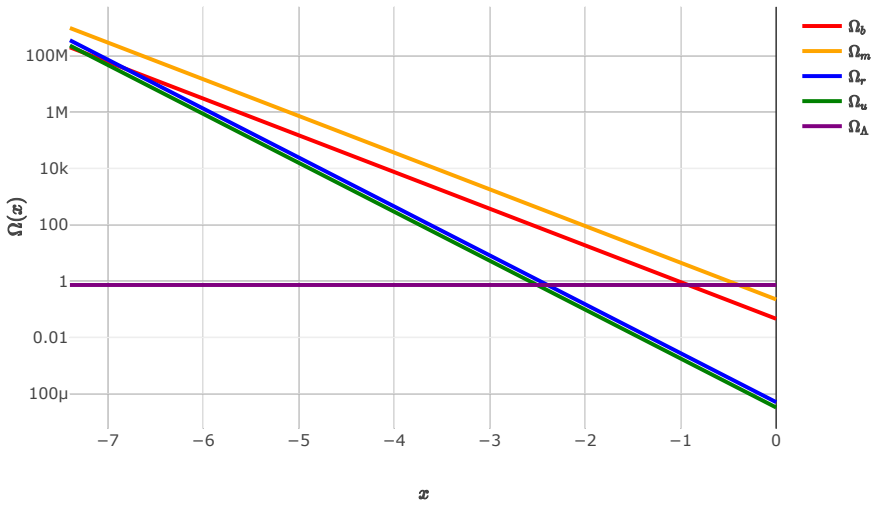

Figure 4.2: Plot of the matter/energy densities of the different components relative to present time's critical density as a function of x from the start of recombination to present time. (Logarithmic scale)

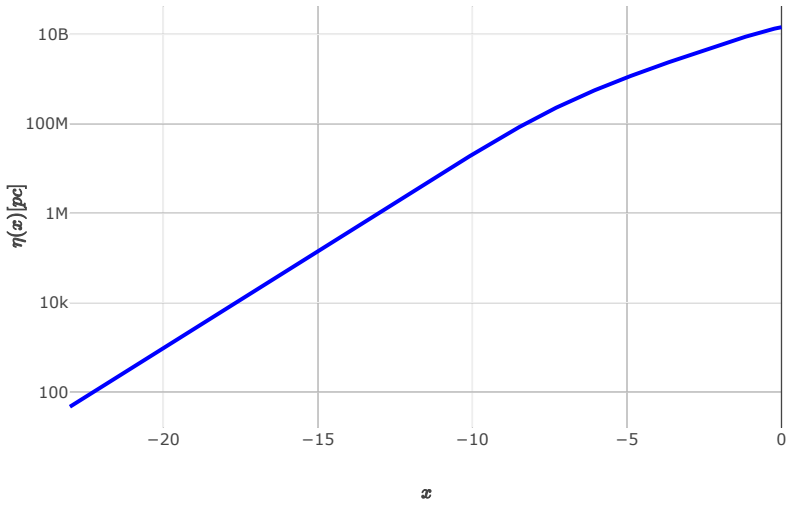

Figure 4.3: Plot of the conformal time η as a function of x from $a = 10^{-10}$ to present time. (Logarithmic scale)

CODE

Listing 1: cmbspec.f90

```
program cmbspec
  use healpix_types
  use params
  use time_mod
  implicit none
  ! Defining the parameter for loop
  integer :: i
  ! Initialize time grids
  call initialize_time_mod

  ! Output to file desired quantities here
  ! Writing data to a file
  ! eta(x):
  open(1, file='eta_x_data.dat', action='write', status='replace')
  do i = 1, n_eta
     write(1,*) x_eta(i), eta(i)
  end do
  close(1)
  print *, 'eta_x_data.dat has been saved'

  ! H(a):
  open(2, file='H_a_data.dat', action='write', status='replace')
  do i = 1, n_t
     write(2,*) a_t(i), get_H(x_t(i))
  end do
  close(2)
  print *, 'H_a_data.dat has been saved'

  ! H(x):
  open(3, file='H_x_data.dat', action='write', status='replace')
  do i = 1, n_t
     write(3,*) x_t(i), get_H(x_t(i))
  end do
  close(3)
  print *, 'H_x_data.dat has been saved'

  ! H(z), units are km / (Mpc s):
  open(5, file='H_z_data.dat', action='write', status='replace')
  do i = 1, n_t
     write(5,*) (1 / a_t(i) - 1), get_H(x_t(i)) * Mpc / 1.0d3
  end do
  close(5)
  print *, 'H_z_data.dat has been saved'

  ! Omega (one file to rule them all):
  open(6, file='Omega_all_data.dat', action='write', status='replace')
  ! The structure is: x Omega_b Omega_m Omega_r Omega_nu Omega_lambda
  do i = 1, n_t
     write(6,'(6E20.13)') x_t(i), Omega_b * a_t(i)**(-3), Omega_m * a_t(i)**(-3),
        ↪ Omega_r * a_t(i)**(-4), Omega_nu * a_t(i)**(-4), Omega_lambda
  end do
  close(6)
  print *, 'Omega_all_data.dat has been saved'
end program cmbspec
```

Listing 2: time_mod.f90

```
module time_mod
  use healpix_types
  use params
```

```
  use ode_solver
  use bs_mod
  use spline_1D_mod
  implicit none

  integer(i4b)                              :: n_t                ! Number of x-values
  real(dp),    allocatable, dimension(:) :: x_t                ! Grid of relevant x-
      ↪ values
  real(dp),    allocatable, dimension(:) :: a_t                ! Grid of relevant a-
      ↪ values

  integer(i4b)                              :: n_eta              ! Number of eta grid
      ↪ poins
  real(dp),    allocatable, dimension(:) :: x_eta              ! Grid points for eta
  real(dp),    allocatable, dimension(:) :: eta, eta2          ! Eta and eta'' at each
      ↪ grid point


contains

  subroutine initialize_time_mod
    implicit none

    integer(i4b) :: i, n, n1, n2
    real(dp)     :: z_start_rec, z_end_rec, z_0, x_start_rec, x_end_rec, x_0, dx,
        ↪ x_eta1, x_eta2, a_init

      ! Variables I've declared
    real(dp) :: eta_i(1)
    real(dp)                                :: h1, h_min, eps

    ! Define two epochs, 1) during and 2) after recombination.
    n1          = 200                       ! Number of grid points during
        ↪ recombination
    n2          = 300                       ! Number of grid points after recombination
    n_t         = n1 + n2                   ! Total number of grid points
    z_start_rec = 1630.4d0                  ! Redshift of start of recombination
    z_end_rec   = 614.2d0                   ! Redshift of end of recombination
    z_0         = 0.d0                      ! Redshift today
    x_start_rec = -log(1.d0 + z_start_rec)  ! x of start of recombination
    x_end_rec   = -log(1.d0 + z_end_rec)    ! x of end of recombination
    x_0         = 0.d0                      ! x today

    n_eta       = 1000                      ! Number of eta grid points (for spline)
    a_init      = 1.d-10                    ! Start value of a for eta evaluation
    x_eta1      = log(a_init)               ! Start value of x for eta evaluation
    x_eta2      = 0.d0                      ! End value of x for eta evaluation

    ! Task: Fill in x and a grids
    print *, "Starting␣the␣1st␣task"
    allocate(x_t(n_t)) != (x_end_rec - x_start_rec) / n1
    dx = (x_end_rec - x_start_rec) / (n1 - 1)
    x_t(1) = x_start_rec
    do i = 2, n1
       x_t(i) = x_t(1) + dx*(i - 1)
    end do
    dx = (x_0 - x_end_rec) / n2
    do i = 1, n2
       x_t(n1 + i) = x_t(n1) + dx * i
    end do
    !print *, "x_t is ", x_t

    allocate(a_t(n_t)) != EXP(x_t(n_t))   !(1/(1 + z_end_rec) - 1 / (1 + z_start_rec))
        ↪ / n1
    a_t = exp(x_t)
    !print *, "a_t is ", a_t
```

```
  print *, "End of the 1st task"



  ! Task: 1) Compute the conformal time at each eta time step
  !       2) Spline the resulting function, using the provided "spline" routine in
      ↪ spline_1D_mod.f90

  ! First, we need to compute the grid for eta, i.e. by doing the same as above:
  print *, "Starting the 2nd task"
  allocate(x_eta(n_eta))
  x_eta(1) = x_eta1
  dx = (x_eta2 - x_eta1) / (n_eta - 1)
  do i = 2, n_eta
     x_eta(i) = x_eta(1) + dx * (i - 1)
  end do
  !print *, "x_eta is ", x_eta

  ! Now, we can integrate using the subroutine from
  allocate(eta(n_eta))
  ! Starting point of integration
  eta(1) = c * a_init / (H_0 * sqrt(Omega_r + Omega_nu))
  !
  eta_i(1) = eta(1)
  h1 = 1.d-6
  h_min = 0.d0
  eps = 1.d-6
  do i = 2, n_eta
     call odeint(eta_i, x_eta(i - 1), x_eta(i), eps, h1, h_min, derivs1, bsstep,
         ↪ output1)
     eta(i) = eta_i(1)
  end do
  ! Writing data to a file
  !open(10, file='eta_x_data.dat', action='write', status='replace')
  !do i = 1, n_eta
  !   write(10,*) x_eta(i), eta(i)
  !end do
  !close(10)
  !print *, "eta is ", eta

  ! Calculating the second derivative of eta with respect to x to be able to use
      ↪ spline to construct the function
  allocate(eta2(n_eta))
  do i = 1, n_eta
     eta2(i) = - (c / (get_H_p(x_eta(i)))**2) * get_dH_p(x_eta(i))
  end do
  !print *, "eta2 is ", eta2

  print *, "End of the 2nd task"



end subroutine initialize_time_mod

  ! Create subroutine to calculate deta/dx
  subroutine derivs1(x, y, dydx)
    use healpix_types
    implicit none
    real(dp),               intent(in)  :: x
    real(dp), dimension(:), intent(in)  :: y
    real(dp), dimension(:), intent(out) :: dydx

    dydx = c / get_H_p(x)
  end subroutine derivs1

  subroutine output1(x, y)
```

```
    use healpix_types
    implicit none
    real(dp),                intent(in)  :: x
    real(dp), dimension(:), intent(in)  :: y
  end subroutine output1


! Task: Write a function that computes H at given x
function get_H(x)
  implicit none

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

  ! As defined by Friedmann equations
  get_H = H_0 * sqrt( (Omega_m + Omega_b) * exp(-3*x) + (Omega_r + Omega_nu) * exp
      ↪ (-4*x) + Omega_lambda)
end function get_H

! Task: Write a function that computes H' = a*H  at given x
function get_H_p(x)
  implicit none

  real(dp), intent(in) :: x
  real(dp)             :: get_H_p
  ! H' = H * a
  get_H_p = get_H(x) * exp(x)
end function get_H_p

! Task: Write a function that computes dH'/dx at given x
function get_dH_p(x)
  implicit none

  real(dp), intent(in) :: x
  real(dp)             :: get_dH_p
  ! The derivative of H prime with respect to x
  get_dH_p = H_0**2 / (2 * get_H_p(x)) * ( - (Omega_m + Omega_b) * exp(-2*x) - 2 * (
      ↪ Omega_r + Omega_nu) * exp(-3*x) + 2 * Omega_lambda * exp(x))
end function get_dH_p

! Task: Write a function that computes eta(x), using the previously precomputed
    ↪ splined function
function get_eta(x_in)
  implicit none

  real(dp), intent(in) :: x_in
  real(dp)             :: get_eta
  ! Calling the splint function to get the conformal time at arbitrary times
  get_eta = splint(x_eta, eta, eta2, x_in)
end function get_eta

end module time_mod
```

Listing 3: Milestone1_DataPlots.py

```
import numpy as np
# library for plotting
import plotly.plotly as py
# for plotting in offline mode
import plotly.offline as plt
import plotly.graph_objs as go
# for making subplots
from plotly import tools
# for writing plots to a file
import plotly.io as pio
#from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot


plt.init_notebook_mode(connected = True)

######
#Data#
######
# Reading data from a file
omegaDataFile = np.loadtxt('Omega_all_data.dat')
etaDataFile   = np.loadtxt('eta_x_data.dat')
HxDataFile    = np.loadtxt('H_x_data.dat')
HzDataFile    = np.loadtxt('H_z_data.dat')



# Omega(x):
X1            = omegaDataFile[:,0]
Omega_b       = omegaDataFile[:,1]
Omega_m       = omegaDataFile[:,2]
Omega_r       = omegaDataFile[:,3]
Omega_nu      = omegaDataFile[:,4]
Omega_lambda  = omegaDataFile[:,5]
# eta(x):
X2            = etaDataFile[:,0]
eta_x         = etaDataFile[:,1] / (3.08567758 * 10**(16)) # changing meters to pc
# H(x):
X3            = HxDataFile[:,0]
H_x           = HxDataFile[:,1]
# H(x):
Z             = HzDataFile[:,0]
H_z           = HzDataFile[:,1]

##########
#Plotting#
##########
# Create traces:
# Omega(x):
omegaB = go.Scatter(
    x = X1,
    y = Omega_b,
    name = '$\Omega_b$',
    line = dict(
        color = ('red'),#'rgb(100, 20, 50)'),
        width = 3))

omegaM = go.Scatter(
    x = X1,
    y = Omega_m,
    name = '$\Omega_m$',
    line = dict(
        color = ('orange'),#'rgb(205, 12, 24)'),
        width = 3))

omegaR = go.Scatter(
```

```
    x = X1,
    y = Omega_r,
    name = '$\Omega_r$',
    line = dict(
        color = ('blue'),#'rgb(300, 200, 100)'),
        width = 3))

omegaNu = go.Scatter(
    x = X1,
    y = Omega_nu,
    name = '$\Omega_\nu$',
    line = dict(
        color = ('green'),#'rgb(0, 15, 46)'),
        width = 3))

omegaL = go.Scatter(
    x = X1,
    y = Omega_lambda,
    name = '$\Omega_\Lambda$',
    line = dict(
        color = ('purple'),#'rgb(10, 40, 250)'),
        width = 3))

# eta(x):
etaX = go.Scatter(
    x = X2,
    y = eta_x,
    name = '$\eta(x)$',
    line = dict(
        color = ('blue'),#'rgb(100, 20, 50)'),
        width = 3))

# H(x):
Hx = go.Scatter(
    x = X3,
    y = H_x,
    name = '$H(x)$',
    line = dict(
        color = ('blue'),#'rgb(100, 20, 50)'),
        width = 3))

# H(z):
Hz = go.Scatter(
    x = Z,
    y = H_z,
    name = '$H(z)$',
    line = dict(
        color = ('blue'),#'rgb(100, 20, 50)'),
        width = 3))

omegaDataPlot = [omegaB, omegaM, omegaR, omegaNu, omegaL]
etaDataPlot   = [etaX]
HxDataPlot    = [Hx]
HzDataPlot    = [Hz]

#figure = tools.make_subplots(rows = 2, cols = 2, subplot_titles = ('Cosmological
    ↪ Parameters', 'Conformal Time',
#                                                                   'Plot 3', 'Plot 4')
    ↪ )

#figure.add_traces(omegaDataPlot)
#figure.add_traces(etaDataPlot)
#figure.append_trace(trace3, 2, 1)
#figure.append_trace(trace4, 2, 2)

# Edit the layout
```

```
omegaLayoutPlot = dict(#title = 'Cosmological Parameters',
              xaxis = dict(title = '$x$'),
              yaxis = dict(title = '$\Omega(x)$', type='log', autorange = True),
              )
etaLayoutPlot = dict(#title = 'Conformal Time',
              xaxis = dict(title = '$x$'),
              yaxis = dict(title = '$\eta(x)␣[pc]$', type = 'log', autorange = True),
              )
HxLayoutPlot  = dict(#title = 'Hubble Parameter',
              xaxis = dict(title = '$x$'),
              yaxis = dict(title = '$H(x)␣[s^{-1}]$', type = 'log', autorange = True),
              )
HzLayoutPlot  = dict(#title = 'Hubble Parameter',
              xaxis = dict(title = '$z$'),
              yaxis = dict(title = '$H(z)␣[s^{-1}]$', type = 'log', autorange = True),
              )

omegaFigure = dict(data = omegaDataPlot, layout = omegaLayoutPlot)
etaFigure   = dict(data = etaDataPlot,   layout = etaLayoutPlot)
HxFigure    = dict(data = HxDataPlot,    layout = HxLayoutPlot)
HzFigure    = dict(data = HzDataPlot,    layout = HzLayoutPlot)

# Plotting everything
plt.iplot(omegaFigure, filename = 'omega')
plt.iplot(etaFigure,   filename = 'eta')
plt.iplot(HxFigure,    filename = 'Hx')
plt.iplot(HzFigure,    filename = 'Hz')

# Saving plots
pio.write_image(omegaFigure, 'Omega_x.pdf')
pio.write_image(etaFigure,   'eta_x.pdf')
pio.write_image(HxFigure,    'H_x.pdf')
pio.write_image(HzFigure,    'H_z.pdf')
```