UiO : Institute of Theoretical Astrophysics
The Faculty of Mathematics and Natural Sciences

Report for AST9240: *The CMB power spectrum*

Maksym Brilenkov

June 1, 2019

1 Introduction

This is the last milestone in which, finally, I am going to compute CMB power spectrum. It uses all the knowledge inherited from previous milestones (Ref. [1, 2, 3] together with algorithms described in Eriksen et al (Ref. [4]) and Callin et al (Ref. [5]).

Here, the quantities to be delivered are as follows: Source function, $\tilde{S}$, transfer function, Θ_l, and multipoles C_l.

The report organized as follows. In section 2, I briefly state the problem to be addressed. Section 3 explains the idea behind the code. Section 4 is dedicated for results. Code is listed in the end of this report.

2 METHODS

As already stated previously (Ref. [3]), I am going to use "line-of-sight integration method" nicely described in Callin *et al* (Ref. [5]). Following this tactics, it is possible to get the following expression for multipoles today:

$$\Theta_l(k,\eta_0) = \int_0^{\eta_0} S(k,\eta)\, j_l\left[k\left(\eta_0-\eta\right)\right] d\eta\,, \tag{2.1}$$

or

$$\Theta_l(k,x=0) = \int_{-\infty}^{0} \tilde{S}(k,x)\, j_l\left[k\left(\eta_0-\eta(x)\right)\right] dx\,, \tag{2.2}$$

where j_l is spherical Bessel functions and $\tilde{S}$ is called *source function*

$$\tilde{S}(k,x) = \tilde{g}\left[\Theta_0+\Psi+\frac{\Pi}{4}\right]+\exp(-\tau)\left[\Psi'-\Phi'\right]-\frac{1}{ck}\frac{d}{dx}\left[\mathcal{H}\tilde{g}v_b\right]+\frac{3}{4c^2k^2}\frac{d}{dx}\left[\mathcal{H}\frac{d}{dx}\left(\mathcal{H}\tilde{g}\Pi\right)\right], \tag{2.3}$$

The last term in the source function is simply (Ref. [5])

$$\frac{d}{dx}\left[\mathcal{H}\frac{d}{dx}\left(\mathcal{H}\tilde{g}\Pi\right)\right] = \frac{\left(\mathcal{H}\mathcal{H}'\right)}{dx}\tilde{g}\Pi+3\mathcal{H}\mathcal{H}'\left(\tilde{g}'Pi+\Pi'\tilde{g}\right)+\mathcal{H}^2\left(\tilde{g}''\Pi+2\tilde{g}'\Pi'+\tilde{g}\Pi''\right), \tag{2.4}$$

with

$$\Pi'' = \frac{2ck}{5\mathcal{H}}\left[-\frac{\mathcal{H}'}{\mathcal{H}}\Theta_1+\Theta_1'\right]+\frac{3}{10}\left[\tau''\Pi+\tau'\Pi'\right]-\frac{3ck}{5\mathcal{H}}\left[-\frac{\mathcal{H}'}{\mathcal{H}}\left(\Theta_3+\Theta_1^P+\Theta_3^P\right)+\left(\Theta_3'+\Theta_1^{P\prime}+\Theta_3^{P\prime}\right)\right]. \tag{2.5}$$

In addition, there is also polarization transfer function (Ref. [4])

$$\Theta_l^E(k,\eta_0) = \sqrt{\frac{l+2}{l-2}}\int_0^{\eta_0} \tilde{S}^E(k,\eta)\, j_l\left[k\left(\eta_0-\eta\right)\right] d\eta\,, \tag{2.6}$$

where polarisation source function is

$$\tilde{S}^E(k,\eta) = \frac{3g\Pi}{4k^2\left(\eta_0-\eta\right)}\,. \tag{2.7}$$

In light of this, the final expression for power spectrum can be written as (Ref. [4])

$$C_l = \int_0^{\infty}\left(\frac{ck}{H_0}\right)^{n-1}\Theta_l^2(k)\frac{dk}{k}\,, \tag{2.8}$$

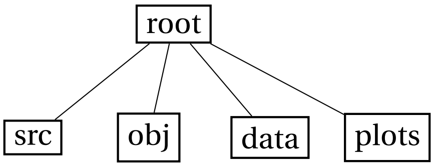
and

$$C_l^{EE} = \int_0^{\infty}\left(\frac{ck}{H_0}\right)^{n-1}\left(\Theta_l^E(k)\right)^2\frac{dk}{k}\,, \tag{2.9}$$

$$C_l^{TE} = \int_0^{\infty}\left(\frac{ck}{H_0}\right)^{n-1}\Theta_l(k)\Theta_l^E(k)\frac{dk}{k}\,, \tag{2.10}$$

3 ALGORITHMS

During the course of this milestone, I significantly developed the program. First of all, the whole folder tree structure has been modified and now has the form



- **root** - contains only Makefile together with compiled binary;
- **obj** - contains all compiled objects, i.e. .mod and .o files;
- **src** - contains all .f90 files;
- **data** - contains all output data files, i.e. .unf and .dat files;
- **plots** - contains all pdf files of plots, produced with python scripts via .ipynb file

The whole algorithm and integration ideas are nicely presented in Erisken at el (Ref. [4]) and Callin et al (Ref. [5]) and I am trying to follow their guidelines. However, my realisation is a little bit different from the initial code written by H.K. Eriksen and below I am briefly summarizing its main features.

The main working module now is supposed to be "cl_mod.f90" and "evolution_mod.f90", but, to make my code easily readable/understandable, I've written two additional modules - "source_func_mod.f90" and "bessel_func_mod.f90". The first one is for source function calculation (according to the idea presented in previous section) and the latter is for computation of bessel function (together with its second derivative). As stated in Eriksen et al (Ref. [4]), "cl_mod.f90" is reserved for power spectrum calculation.

So, the idea here is pretty straightforward. First, I calculate all quantities computed in previous Milestone (Ref. [3]), i.e. Φ, Ψ etc., and store them into the unformatted binary files with corresponding names. Second, I calculate source function, in the "source_func_mod.f90" and also store both $S(k,x)$ and $S^E(k,x)$ in separate binary files. Third, I use "bessel_func_mod.f90" to calculate Bessel functions and its second derivative and also store it in the binary files. With all this in hand I proceed for transfer function and power spectrum calculation, which is done via "cl_mod.f90". In addition, I also store $\Theta_l(k)$, $\Theta_l^E(k)$, $(\Theta_l(k))^2/k$, $(\Theta_l^E)^2/k$ and $\Theta^E(k)\cdot\Theta_l(k)/k$ in separate binary files.

Why binary files? There are two main reasons for it: (1) it reduces the time for running the whole code and, as a consequence, (2) makes it easier debugging. As an example of storing and retrieving data, I present this chunk of code which corresponds to storing

```
folder = "data/"
filename = "integrand.unf"
open(37, file = trim(folder) // trim(filename), form = "
    ↪ unformatted", action = "write", status = "replace")
write(37) integrand
close(37)
! Second, transfer function
filename = "Theta_l.unf"
open(38, file = trim(folder) // trim(filename), form = "
    ↪ unformatted", action = "write", status = "replace")
write(38) Theta_l
close(38)
filename = "ThetaE_l.unf"
open(39, file = trim(folder) // trim(filename), form = "
    ↪ unformatted", action = "write", status = "replace")
write(39) ThetaE_l
close(39)
filename = "integrandT.unf"
open(40, file = trim(folder) // trim(filename), form = "
    ↪ unformatted", action = "write", status = "replace")
write(40) integrandT
close(40)
filename = "integrandE.unf"
open(41, file = trim(folder) // trim(filename), form = "
    ↪ unformatted", action = "write", status = "replace")
write(41) integrandE
close(41)
filename = "integrandTE.unf"
open(42, file = trim(folder) // trim(filename), form = "
    ↪ unformatted", action = "write", status = "replace")
write(42) integrandTE
close(42)
```

and retrieving

```
do i = 6, 10
   inquire(file = trim(folder) // trim(filename(i)), exist =
      ↪ exists(i))
   if ((exists(i) == .True.) .and. (bessel_data_exists == .
      ↪ True.) .and. (source_data_exists == .True.)) then
      transfer_data_exists = .True.
   else
      transfer_data_exists = .False.
      exit
   end if
end do

if (transfer_data_exists == .False.) then
   print *, "Transfer func data doesn't exist, so will be 
```

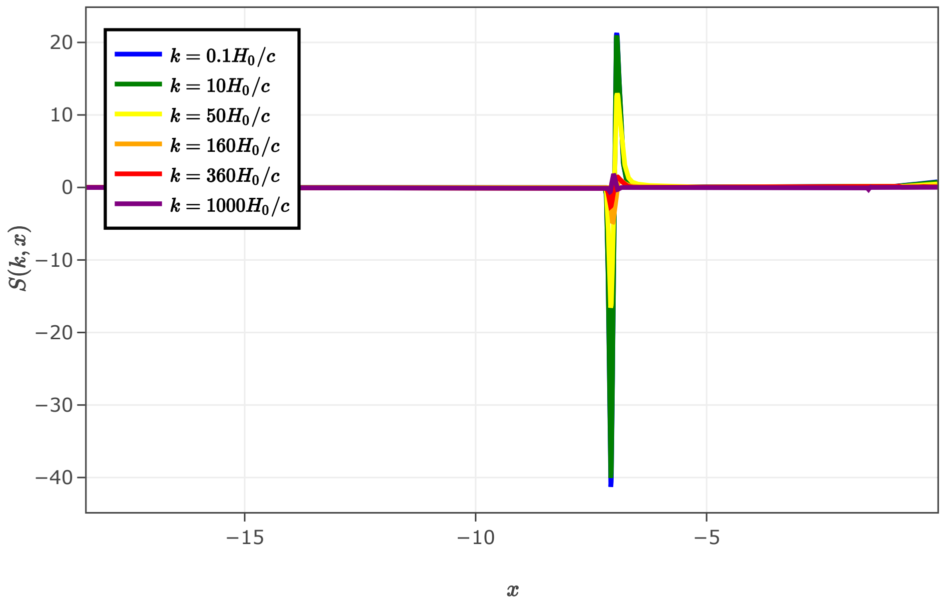
```
          ↪ created"
        call compute_trasfer_func
     else
        print *, "Transfer␣func␣data␣exists,␣so␣will␣be␣retrieved"
        open(38, file = trim(folder) // trim(filename(6)), form = "
           ↪ unformatted", action = "read")
        read(38) Theta_l
        close(38)
        open(39, file = trim(folder) // trim(filename(7)), form = "
           ↪ unformatted", action = "read")
        read(39) ThetaE_l
        close(39)
        open(40, file = trim(folder) // trim(filename(8)), form = "
           ↪ unformatted", action = "read")
        read(40) integrandT
        close(40)
        open(41, file = trim(folder) // trim(filename(9)), form = "
           ↪ unformatted", action = "read")
        read(41) integrandE
        close(41)
        open(42, file = trim(folder) // trim(filename(10)), form =
           ↪ "unformatted", action = "read")
        read(42) integrandTE
        close(42)
     end if
```

data from a binary file. In brief, the last part checks whether one of the binary files, which corresponds to source and bessel functions, exists. If it doesn't, then it will recalculate the missing part and then compute transfer function (and so on).

Such approach presents the significant speed up of running time. For instance, to run the whole algorithm without any binary files present, takes more then two hours, whereas with existing pre-calculated quantities, it takes about 476.677 seconds to calculate all required quantities for chosen l and k values. Again, for accessibility, I put two array, which corresponds to these values, inside the body of the program itself, i.e. "cmbspec.f90". After program computed C_l's, it will run the subroutine "$save\_milestone4\_plot\_data$" which will save all necessary quantities into the ".dat" files.

4 RESULTS

Below I present the main results of this milestone. The C^{EE} and C^{TE} look reasonable, but, unfortunately, due to some bug in the code, I got the incorrect shape for C^{TT} for low values of l.



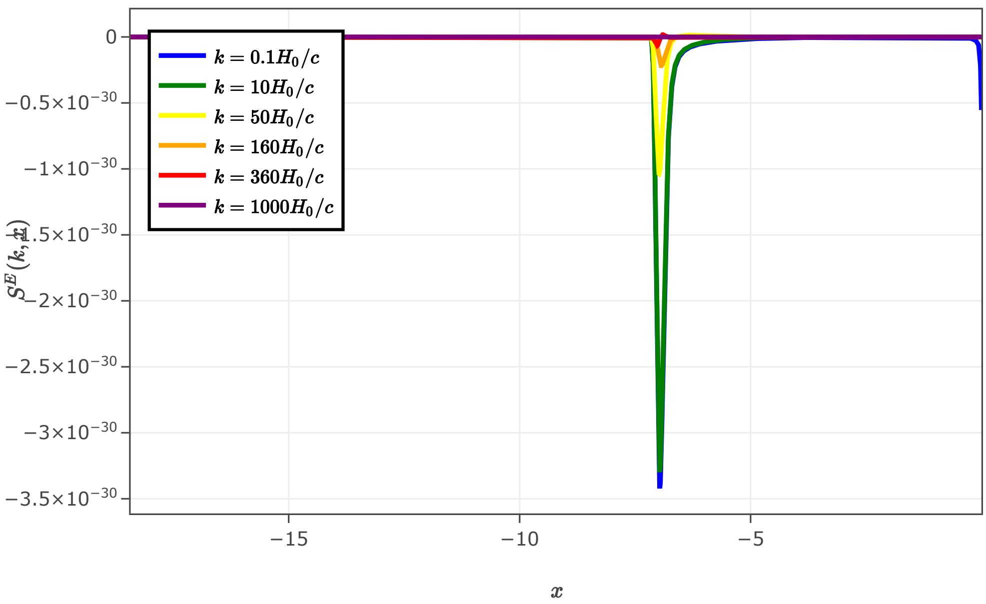


Figure 4.1: A plot of (top) source and polarization (bottom) source functions for different values of k.

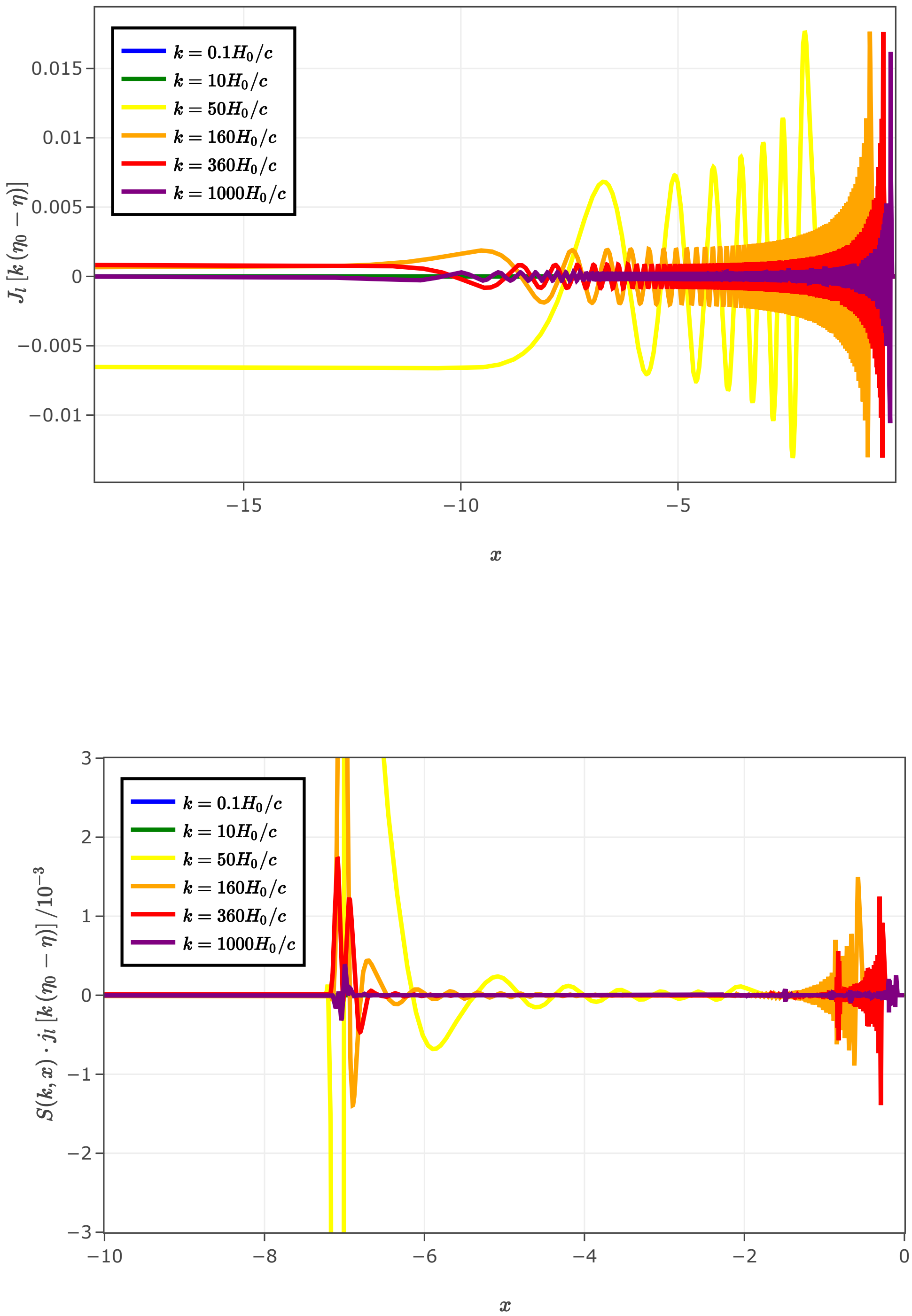


Figure 4.2: A plot of (top) bessel functions and (bottom) integrand in the transfer function for different values of k.

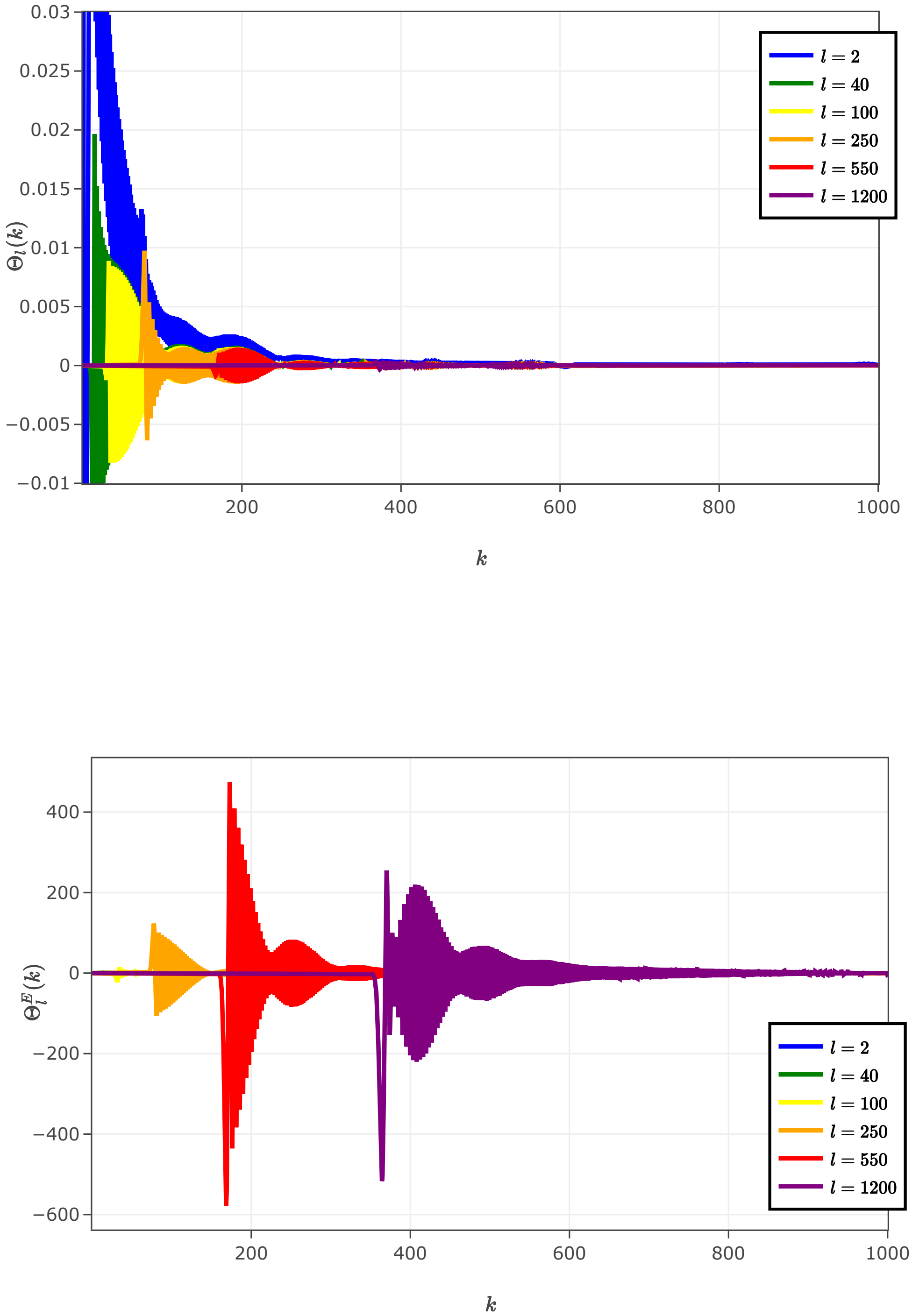


Figure 4.3: A plot of transfer functions for different values of l.

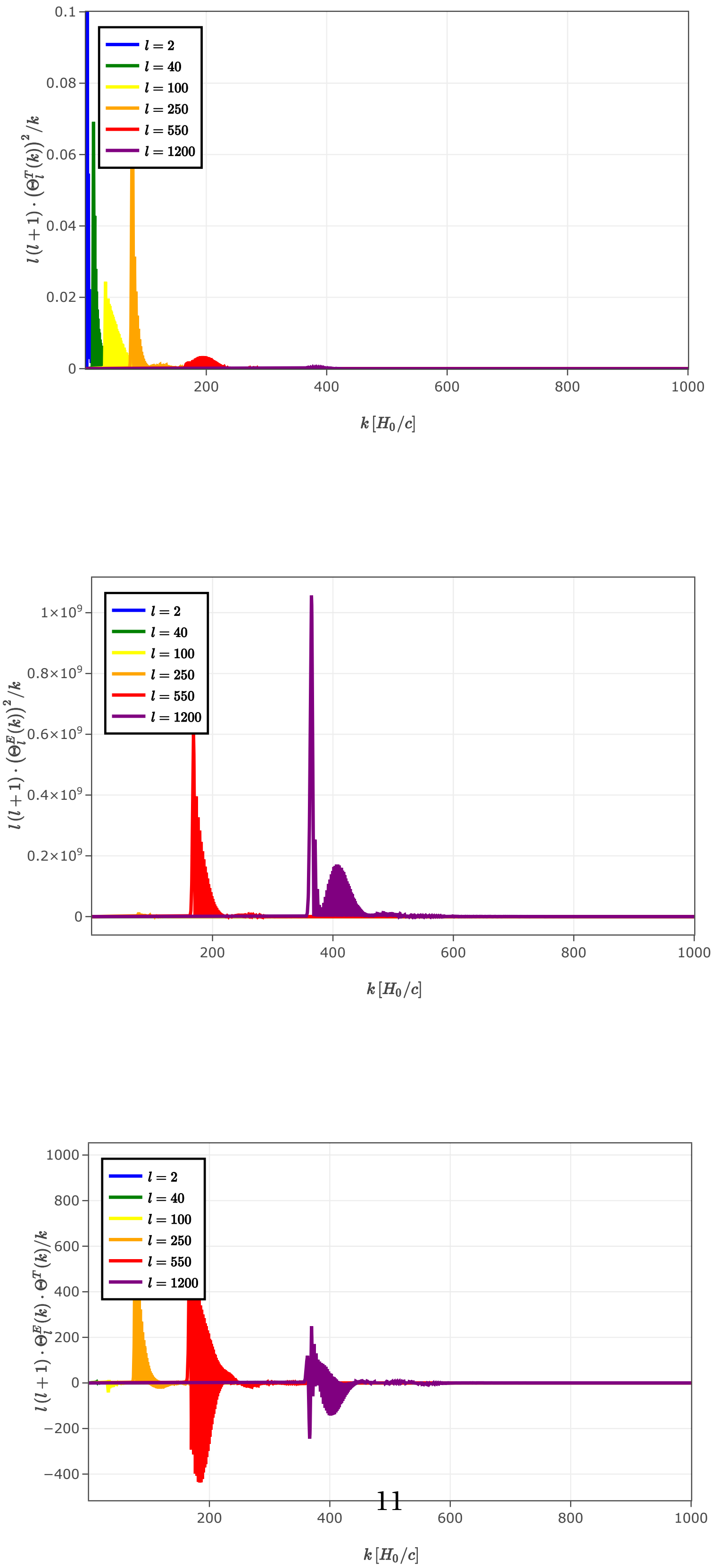


Figure 4.4: A plot of integrands in the C_l^{TT}, C_l^{EE} and C_l^{TE} for different values of l.

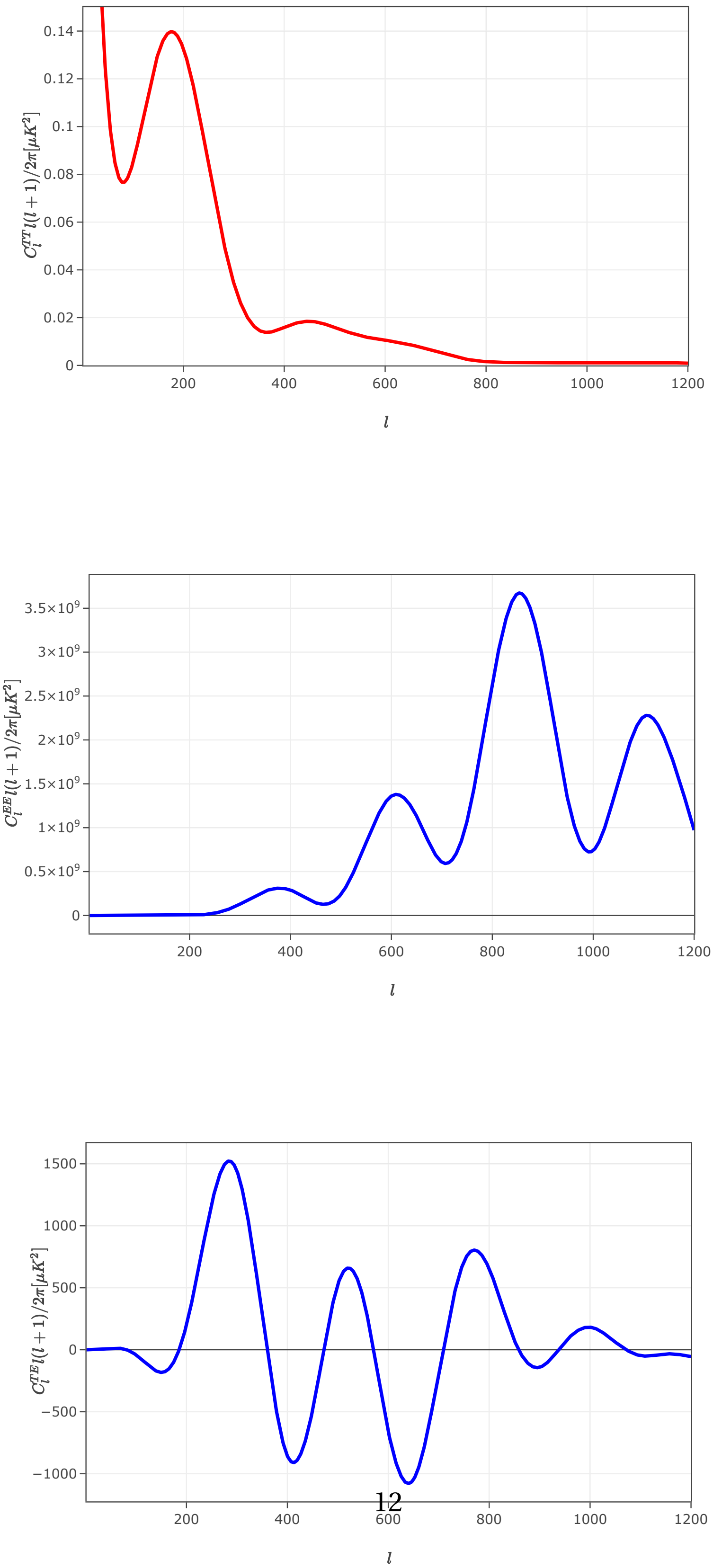


Figure 4.5: A plot of integrands in the C_l^{TT}, C_l^{EE} and C_l^{TE} for different values of l.

REFERENCES

[1] M. Brilenkov, *Report for AST9240: The background evolution of the universe* (2019).

[2] M. Brilenkov, *Report for AST9240: The recombination history of the universe* (2019).

[3] M. Brilenkov, *Report for AST9240: The evolution of structures in the universe* (2019).

[4] H. K. Eriksen, *Milestone 4: The CMB power spectrum* (2019).

[5] P. Callin, *How to calculate the CMB spectrum*, arXiv:astro-ph/0606683.

CODE

Listing 1: cmbspec.f90

```
program cmbspec
  use healpix_types
  use params
  use time_mod
  use rec_mod
  use evolution_mod
  use source_func_mod
  use cl_mod
  implicit none

  ! Defining the parameter for loop
  integer(i4b)                                 :: i, j
  ! For chosing plotting values
  real(dp),      allocatable, dimension(:) :: k_chosen
  integer(i4b), allocatable, dimension(:) :: index_chosen, l_chosen
  ! To find out the number of seconds it takes to calculate all integrals
  real(dp)                                     :: start_time, stop_time
  logical(lgt)                                 :: to_plot

  call cpu_time(start_time)


!  folder = "data/"

  !=================================!
  !   MILESTONE 1: Time Evolution   !
  !=================================!
  call initialize_time_mod

  !=================================!
  !    MILESTONE 2: Recombination   !
  !=================================!
  call initialize_rec_mod

  !=================================!
  !    MILESTONE 3: Perturbation    !
  !=================================!

  ! Milestone 3 part is included into cl computations

  !=================================!
  !   MILESTONE 4: Power Spectrum   !
  !=================================!
!  call get_hires_source_function
  call compute_cls

  ! Choosing k values we want to plot
  allocate(k_chosen(1:6))
  allocate(index_chosen(size(k_chosen)))
  k_chosen     = (/ 0.1d0, 1.d1, 5.d1, 16.d1, 36.d1, 1.d3 /)
  k_chosen     = k_chosen * H_0 / c
  ! Decide whether I want to plot Milestone 3
  to_plot      = .True.
  if (to_plot == .True.) then
     ! Calling special routine for saving chosen values
     ! of milestone 3
     print *, "Milestone 3: Getting ploting data"
```

```
    do i = 1, size(k_chosen)
       ! Finding the index which corresponds between desired stored data
       index_chosen(i) = nint(sqrt((k_chosen(i) - k_min) / (k_max-k_min)) * n_k)
       if (index_chosen(i) == 0) then
          index_chosen(i) = 1
       end if
       ! Passing the chosen index into subroutine in
       ! source_func_mod which will save all computed data
       ! (e.g. Phi, Psi, delta) into .dat files for a
       ! chosen number of values
       call save_milestone3_plot_data(index_chosen(i))
    end do
  end if

  ! Decide wheather I want to plot Milestone 4
!  ls = (/ 2, 3, 4, 6, 8, 10, 12, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100, &
!       & 120, 140, 160, 180, 200, 225, 250, 275, 300, 350, 400, 450, 500, 550, &
!       & 600, 650, 700, 750, 800, 850, 900, 950, 1000, 1050, 1100, 1150, 1200 /)

  l_chosen = (/ 2, 40, 100, 250, 550, 1200 /)
  to_plot     = .True.
  if (to_plot == .True.) then
     ! Calling special routine for saving chosen values
     ! of milestone 3
     print *, "Milestone␣4:␣Getting␣ploting␣data"
     do i = 1, size(k_chosen)
        ! Finding the index which corresponds between desired stored data
        index_chosen(i) = nint(sqrt((k_chosen(i) - k_min) / (k_max-k_min)) *
           ↪ n_k_new)
        if (index_chosen(i) == 0) then
           index_chosen(i) = 1
        end if
        do j = 1, size(l_chosen)
           print *, "k,␣l:␣", index_chosen(i), l_chosen(j)
           call save_milestone4_plot_data(l_chosen(j), index_chosen(i))
        end do
     end do
  end if

!  print *, n_k_new
!  print *, index_chosen

  deallocate(k_chosen)
  deallocate(index_chosen)

  ! Total time for running the program
  call cpu_time(stop_time)
  print *, "␣Total␣Running␣Time:", &
       stop_time - start_time, "seconds"
end program cmbspec
```

Listing 2: source_func_mod.f90

```
module source_func_mod
  use healpix_types
  use bs_mod
  use ode_solver
  use spline_2D_mod

  use params
  use time_mod
  use rec_mod
  use evolution_mod
  implicit none

  !==================================================!
  !                  Global parameters               !
  !==================================================!
  ! Most of them is taken from "evolution_mod"

  ! Fourier mode list
  real(dp), allocatable, dimension(:) :: k_old, x_old, k_new, x_new
  integer(i4b), parameter             :: n_k_new = 5000, n_tot_new = 5000

contains

  ! Access pre-computed values
  subroutine get_evolution_data()!k)
    implicit none

!    integer(i4b), intent(in)    :: k ! index of k_current
    character(len=1024)                               :: folder
    character(len=1024), dimension(:), allocatable :: filename
    logical(lgt)                                      :: data_exists
    logical(lgt),        dimension(:), allocatable :: exists
    integer(i4b)                                      :: i

    folder = "data/"

    ! Checking for data existence.
    ! If it doesn't, It will calculate the whole loop.
    ! If it exists, then I will simply take it from file
    allocate(filename(20))
    allocate(exists(20))
    filename = (/ "k_old.unf", "x_old.unf", "delta_x.unf", "deltab_x.unf", "v_x.
        ↪ unf", "vb_x.unf", &
                & "Phi_x.unf", "Psi_x.unf", "Theta_x.unf", "ThetaP_x.unf", "Nu_x.
                    ↪ unf", "ddelta_x.unf", &
                & "ddeltab_x.unf", "dv_x.unf", "dvb_x.unf", "dPhi_x.unf", "dPsi_x.
                    ↪ unf", "dTheta_x.unf", &
                & "dThetaP_x.unf", "dNu_x.unf"/)
    do i = 1, 20
       inquire(file = trim(folder) // trim(filename(i)), exist = exists(i))
       if (exists(i)) then
          data_exists = .True.
       else
          data_exists = .False.
          exit
       end if
    end do

    allocate(x_old(0:n_tot))
    allocate(k_old(n_k))
```

```
!     allocate(k_new(n_k_new))
!     allocate(x_new(n_tot_new))
    ! If data doesn't xist will calculate it from evolution_mod
    if (data_exists == .False.) then
       print *, "Evolution␣data␣doesn't␣exist,␣so␣will␣be␣created"
       call initialize_perturbation_eqns
       call integrate_perturbation_eqns
       x_old = x_evol
       k_old = ks
    else
       print *, "Evolution␣data␣exists,␣so␣will␣be␣retrieved"
       ! Allocating arrays for perturbation quantities
       ! deltas
       allocate(delta(0:n_tot, n_k))
       allocate(delta_b(0:n_tot, n_k))
       allocate(ddelta(0:n_tot, n_k))
       allocate(ddelta_b(0:n_tot, n_k))
       ! velocity
       allocate(v(0:n_tot, n_k))
       allocate(v_b(0:n_tot, n_k))
       allocate(dv(0:n_tot, n_k))
       allocate(dv_b(0:n_tot, n_k))
       ! Potentials
       allocate(Phi(0:n_tot, n_k))
       allocate(Psi(0:n_tot, n_k))
       allocate(dPhi(0:n_tot, n_k))
       allocate(dPsi(0:n_tot, n_k))
       ! Theta
       allocate(Theta(0:n_tot, 0:lmax_int, n_k))
       allocate(dTheta(0:n_tot, 0:lmax_int, n_k))
       ! Polarisation
       allocate(ThetaP(0:n_tot, 0:lmax_int, n_k))
       allocate(dThetaP(0:n_tot, 0:lmax_int, n_k))
       ! Neutrinos
       allocate(Nu(0:n_tot, 0:lmax_nu, n_k))
       allocate(dNu(0:n_tot, 0:lmax_nu, n_k))
       ! k grid
       open(13, file = trim(folder) // trim(filename(1)), form = "unformatted",
           ↪ action = "read")
       read(13) k_old
       close(13)
       ! x grid
       open(14, file = trim(folder) // trim(filename(2)), form = "unformatted",
           ↪ action = "read")
       read(14) x_old
       close(14)
       ! delta
       open(15, file = trim(folder) // trim(filename(3)), form = "unformatted",
           ↪ action = "read")
       read(15) delta
       close(15)
       ! delta_b
       open(16, file = trim(folder) // trim(filename(4)), form = "unformatted",
           ↪ action = "read")
       read(16) delta_b
       close(16)
       ! v
       open(17, file = trim(folder) // trim(filename(5)), form = "unformatted",
           ↪ action = "read")
       read(17) v
       close(17)
       ! v_b
```

```
open(18, file = trim(folder) // trim(filename(6)), form = "unformatted",
    ↪ action = "read")
read(18) v_b
close(18)
! Phi
open(19, file = trim(folder) // trim(filename(7)), form = "unformatted",
    ↪ action = "read")
read(19) Phi
close(19)
! Psi
open(20, file = trim(folder) // trim(filename(8)), form = "unformatted",
    ↪ action = "read")
read(20) Psi
close(20)
! Theta
open(21, file = trim(folder) // trim(filename(9)), form = "unformatted",
    ↪ action = "read")
read(21) Theta
close(21)
! ThetaP
open(22, file = trim(folder) // trim(filename(10)), form = "unformatted",
    ↪ action = "read")
read(22) ThetaP
close(22)
! Nu
open(23, file = trim(folder) // trim(filename(11)), form = "unformatted",
    ↪ action = "read")
read(23) Nu
close(23)
!========================
! Derivatives
! ddelta
open(24, file = trim(folder) // trim(filename(12)), form = "unformatted",
    ↪ action = "read")
read(24) ddelta
close(24)
! ddelta_b
open(25, file = trim(folder) // trim(filename(13)), form = "unformatted",
    ↪ action = "read")
read(25) ddelta_b
close(25)
! dv
open(26, file = trim(folder) // trim(filename(14)), form = "unformatted",
    ↪ action = "read")
read(26) dv
close(26)
! dv_b
open(27, file = trim(folder) // trim(filename(15)), form = "unformatted",
    ↪ action = "read")
read(27) dv_b
close(27)
! dPhi
open(27, file = trim(folder) // trim(filename(16)), form = "unformatted",
    ↪ action = "read")
read(27) dPhi
close(27)
! dPsi
open(28, file = trim(folder) // trim(filename(17)), form = "unformatted",
    ↪ action = "read")
read(28) dPsi
close(28)
! dTheta
```

```
      open(29, file = trim(folder) // trim(filename(18)), form = "unformatted",
         ↪ action = "read")
      read(29) dTheta
      close(29)
      ! dThetaP
      open(30, file = trim(folder) // trim(filename(19)), form = "unformatted",
         ↪ action = "read")
      read(30) dThetaP
      close(30)
      ! dNu
      open(31, file = trim(folder) // trim(filename(20)), form = "unformatted",
         ↪ action = "read")
      read(31) dNu
      close(31)
    end if
  end subroutine get_evolution_data

  !===================================================!
  !      MILESTONE 3: Saving chosen evolution data    !
  !===================================================!
  subroutine save_milestone3_plot_data(k)
    implicit none

    integer(i4b), intent(in) :: k
    integer(i4b)             :: i, l, fileindex
    character(len=1024)      :: folder, filename
    ! format descriptor
    character(len=1024)      :: format_string, k1, l1

    folder = "data/"
    ! an integer of width 3 with zeros on the left if the value is not enough
    format_string = "(I3.3)"
    ! converting integer to string using an 'internal file'
    write (k1, format_string) k

    ! Getting data from "unf" binary files
    call get_evolution_data()


    filename = "delta_"//trim(k1)//"_x.dat"
    open(1, file = trim(folder) // trim(filename), action = "write", status = "
       ↪ replace")
    do i = 0, n_tot
      write(1,*) x_old(i), "␣", delta(i, k)
    end do
    close(1)
!     print *, "It works"
    filename = "deltab_"//trim(k1)//"_x.dat"
    open(2, file = trim(folder) // trim(filename), action = "write", status = "
       ↪ replace")
    do i = 0, n_tot
      write(2,*) x_old(i), "␣", delta_b(i, k)
    end do
    close(2)
    filename = "v_"//trim(k1)//"_x.dat"
    open(3, file = trim(folder) // trim(filename), action = "write", status = "
       ↪ replace")
    do i = 0, n_tot
      write(3,*) x_old(i), "␣", v(i, k)
    end do
    close(3)
    filename = "vb_"//trim(k1)//"_x.dat"
```

```
    open(4, file = trim(folder) // trim(filename), action = "write", status = "
        ↪ replace")
    do i = 0, n_tot
       write(4,*) x_old(i), "␣", v_b(i, k)
    end do
    close(4)
    ! Phi
    filename = "Phi_"//trim(k1)//"_x.dat"
    open(5, file = trim(folder) // trim(filename), action = "write", status = "
        ↪ replace")
    do i = 0, n_tot
       write(5,*) x_old(i), "␣", Phi(i, k)
    end do
    close(5)
    ! Psi
    filename = "Psi_"//trim(k1)//"_x.dat"
    open(6, file = trim(folder) // trim(filename), action = "write", status = "
        ↪ replace")
    do i = 0, n_tot
       write(6,*) x_old(i), "␣", Psi(i, k)
    end do
    close(6)
!    print *, "it works"
    do l = 0, 3
       ! changing format of the variable from integer to string
       format_string = "(I1)"
       write (l1, format_string) l
       ! updating the index of a file
       fileindex = 7 + l
       ! File structure is:
       ! Name_l_k_x.dat
       filename = "Theta_"//trim(l1)//"_"//trim(k1)//"_x.dat"
       open(fileindex, file = trim(folder) // trim(filename), action = "write",
           ↪ status = "replace")
       do i = 0, n_tot
          write(fileindex,*) x_old(i), "␣", Theta(i, l, k)
       end do
       close(fileindex)
       ! updating the index of a file
       fileindex = 11 + l
       ! File structure is:
       ! Name_l_k_x.dat
       filename = "ThetaP_"//trim(l1)//"_"//trim(k1)//"_x.dat"
       open(fileindex, file = trim(folder) // trim(filename), action = "write",
           ↪ status = "replace")
       do i = 0, n_tot
          write(fileindex,*) x_old(i), "␣", ThetaP(i, l, k)
       end do
       close(fileindex)
       ! updating the index of a file
       fileindex = 15 + l
       ! File structure is:
       ! Name_l_k_x.dat
       filename = "Nu_"//trim(l1)//"_"//trim(k1)//"_x.dat"
       open(fileindex, file = trim(folder) // trim(filename), action = "write",
           ↪ status = "replace")
       do i = 0, n_tot
          write(fileindex,*) x_old(i), "␣", Nu(i, l, k)
       end do
       close(fileindex)
    end do
```

```
!=======================
! Derivatives
! ddelta, ddelta_b
filename = "ddelta_"//trim(k1)//"_x.dat"
open(19, file = trim(folder) // trim(filename), action = "write", status = "
    ↪ replace")
do i = 0, n_tot
   write(19,*) x_old(i), "␣", ddelta(i, k)
end do
close(19)
filename = "ddeltab_"//trim(k1)//"_x.dat"
open(20, file = trim(folder) // trim(filename), action = "write", status = "
    ↪ replace")
do i = 0, n_tot
   write(20,*) x_old(i), "␣", ddelta_b(i, k)
end do
close(20)

! dv, dv_b
filename = "dv_"//trim(k1)//"_x.dat"
open(21, file = trim(folder) // trim(filename), action = "write", status = "
    ↪ replace")
do i = 0, n_tot
   write(21,*) x_old(i), "␣", dv(i, k)
end do
close(21)
filename = "dvb_"//trim(k1)//"_x.dat"
open(22, file = trim(folder) // trim(filename), action = "write", status = "
    ↪ replace")
do i = 0, n_tot
   write(22,*) x_old(i), "␣", dv_b(i, k)
end do
close(22)
! dPhi
filename = "dPhi_"//trim(k1)//"_x.dat"
open(23, file = trim(folder) // trim(filename), action = "write", status = "
    ↪ replace")
do i = 0, n_tot
   write(23,*) x_old(i), "␣", dPhi(i, k)
end do
close(23)
! Psi
filename = "dPsi_"//trim(k1)//"_x.dat"
open(24, file = trim(folder) // trim(filename), action = "write", status = "
    ↪ replace")
do i = 0, n_tot
   write(24,*) x_old(i), "␣", dPsi(i, k)
end do
close(24)

do l = 0, 3
   ! changing format of the variable from integer to string
   format_string = "(I1)"
   write (l1, format_string) l
   ! updating the index of a file
   fileindex = 25 + l
   ! File structure is:
   ! Name_l_k_x.dat
   ! dTheta
   filename = "dTheta_"//trim(l1)//"_"//trim(k1)//"_x.dat"
   open(fileindex, file = trim(folder) // trim(filename), action = "write",
       ↪ status = "replace")
```

```
      do i = 0, n_tot
         write(fileindex,*) x_old(i), "␣", dTheta(i, l, k)
      end do
      close(fileindex)
      ! updating the index of a file
      fileindex = 29 + 1
      ! dThetaP
      filename = "dThetaP_"//trim(l1)//"_"//trim(k1)//"_x.dat"
      open(fileindex, file = trim(folder) // trim(filename), action = "write",
          ↪ status = "replace")
      do i = 0, n_tot
         write(fileindex,*) x_old(i), "␣", dThetaP(i, l, k)
      end do
      close(fileindex)
      ! updating the index of a file
      fileindex = 29 + 1
      ! dThetaP
      filename = "dThetaP_"//trim(l1)//"_"//trim(k1)//"_x.dat"
      open(fileindex, file = trim(folder) // trim(filename), action = "write",
          ↪ status = "replace")
      do i = 0, n_tot
         write(fileindex,*) x_old(i), "␣", dThetaP(i, l, k)
      end do
      close(fileindex)
      ! updating the index of a file
      fileindex = 33 + 1
      ! dNu
      filename = "dNu_"//trim(l1)//"_"//trim(k1)//"_x.dat"
      open(fileindex, file = trim(folder) // trim(filename), action = "write",
          ↪ status = "replace")
      do i = 0, n_tot
         write(fileindex,*) x_old(i), "␣", dNu(i, l, k)
      end do
      close(fileindex)
   end do

   ! To resolve all the conflicts with the code I need to deallocate these arrays
       ↪ here
   deallocate(x_old)
   deallocate(k_old)
   deallocate(delta)
   deallocate(delta_b)
   deallocate(ddelta)
   deallocate(ddelta_b)
   ! velocity
   deallocate(v)
   deallocate(v_b)
   deallocate(dv)
   deallocate(dv_b)
   ! Potentials
   deallocate(Phi)
   deallocate(Psi)
   deallocate(dPhi)
   deallocate(dPsi)
   ! Theta
   deallocate(Theta)
   deallocate(dTheta)
   ! Polarisation
   deallocate(ThetaP)
   deallocate(dThetaP)
   ! Neutrinos
   deallocate(Nu)
```

```
    deallocate(dNu)

  end subroutine save_milestone3_plot_data


  !====================================================!
  !      MILESTONE 4: Source function calculation     !
  !====================================================!
  subroutine get_hires_source_function(S_high_res, SE_high_res)
    implicit none

    real(dp), allocatable, dimension(:,:), intent(out) :: S_high_res, SE_high_res
!    real(dp), allocatable, dimension(:,:)               :: S_high_res, SE_high_res
    real(dp), allocatable, dimension(:,:,:,:)          :: coeff, coeffE
    integer(i4b)                                       :: i, j
    real(dp)                                           :: deriv1, deriv2, deriv3
    ! expr1 -
    ! expr2 - Sachs-Wolf term
    ! expr3 - Doppler term
    real(dp)                                           :: expr1, expr2, expr3,
        ↪ expr4
    real(dp)                                           :: g, dg, ddg, tau, dtau,
        ↪ ddtau
    real(dp)                                           :: eta, eta_0, H_p, dH_p,
        ↪ ddH_p
    real(dp)                                           :: PI, dPI, ddPI, g_eta,
        ↪ Pi_c, ck_current
    real(dp)                                           :: x_init, x_today, x_step
    ! Low resolution source function
    real(dp),              allocatable, dimension(:,:) :: S_low_res, SE_low_res
    real(dp),              allocatable, dimension(:)   :: eta_
    character(len=1024)                                :: folder, filename
    !real(dp),  dimension(1:,1:,1:,1:)                 :: coeff, coeffE

    ! Step 1 - computing the source function with existing k and x values
    ! Getting data for necessary variables (see Milestone 3)

    call get_evolution_data()

    !================================!
    ! Low Resolution source function !
    !================================!
    ! (i.e. computed on an old grid)
    allocate(eta_(0:n_tot))
    allocate(S_low_res(n_k, 0:n_tot))
    allocate(SE_low_res(n_k, 0:n_tot))

    do j = 1, n_k

       do i = 0, n_tot
          H_p = get_H_p(x_old(i))
          dH_p = get_dH_p(x_old(i))
          ddH_p = get_ddH_p(x_old(i))
          !print *, 'ddH_p is ', ddH_p
          ! eta and eta0
          eta_(i) = get_eta(x_old(i))
!          print *, eta
          eta_0 = get_eta(x_old(n_tot))
          ! tau, tau', tau''
          tau = get_tau(x_old(i))
          dtau = get_dtau(x_old(i))
          ddtau = get_ddtau(x_old(i))
```

```
          ! g, g', g''
          g   = get_g(x_old(i))
!          print *, "g is", g
          dg  = get_dg(x_old(i))
          ddg = get_ddg(x_old(i))
          !print *, "ddg is", ddg
          ! g(eta):
!          g_eta = -H_p * get_dtau(x_old(i)) * exp(-get_tau(x_old(i)))

!          Pi_c     = Theta(i,2,j )
!          ck_current= c * k_old(j)
          ! Without polarization
!          ddPI  = 2.d0 * ck_current / (5.d0 * H_p) * (-dH_p / H_p * Theta(i,1,j)
    ↪ + dTheta(i,1,j)) &
!               + 0.3d0 * (ddtau * Pi_c + dtau * dPi) &
!               -3.d0 * ck_current/(5.d0 * H_p) * (-dH_p / H_p * Theta(i,3,j) +
    ↪ dTheta(i,3,j))

!          S_low_res(j, i) = g*(Theta(i,0,j) + Psi(i,j) + .25d0*Pi_c) &
!                            +exp(-tau)*(dPsi(i,j)-dPhi(i,j)) &
!                            -1.d0/ck_current*(H_p*(g*dv_b(i, j) + v_b(i, j)*dg) +
    ↪ g*v_b(i,j)*dH_p) &
!                            +.75d0/ck_current**2*((H_0**2/2.d0*((Omega_m+Omega_b)/
    ↪ exp(x_t(i)) &
!                            +4.d0*Omega_r/exp(2.d0*x_t(i)) +4.d0*Omega_lambda*exp
    ↪ (2.d0*x_t(i))))*&
!                            g*Pi_c +3.d0*H_p*dH_p*(dg*Pi_c+g*dPI)+H_p**2* &
!                            (ddg*Pi_c +2.d0*dg*dPI+g*ddPI))
          ! Including polarization
          PI = Theta(i, 2, j) + ThetaP(i, 0, j) + ThetaP(i, 2, j)     ! correct
          dPI = dTheta(i, 2, j) + dThetaP(i, 0, j) + dThetaP(i, 2, j) ! correct
          ddPI = (2.d0 * c * k_old(j) / (5.d0 * H_p)) * (-dH_p * Theta(i, 1, j) /
              ↪ H_p + dTheta(i, 1, j)) &
               & + (3.d0 / 10.d0) * (ddtau * PI + dtau * dPI) &
               & - (3.d0 * c * k_old(j) / (5.d0 * H_p)) * ((-dH_p / H_p) * (Theta(
                   ↪ i, 3, j) &
               & + ThetaP(i, 1, j) + ThetaP(i, 3, j)) &
               & + (dTheta(i, 3, j) + dThetaP(i, 1, j) + dThetaP(i, 3, j)))

          deriv1 = dH_p * g * v_b(i, j) + H_p * dg * v_b(i, j) + H_p * g * dv_b(i,
              ↪ j) ! correct
          deriv2 = dH_p**2 + H_p * ddH_p
          deriv3 = deriv2 * g * PI + 3.d0 * H_p * dH_p * (dg * PI + g * dPI) &
               & + H_p**2 * (ddg * PI + 2.d0 * dg * PI + g * ddPI) ! correct
          !print *, "deriv 3 ", deriv3
          ! Calculation of low resolution source function
          expr1 = g * (Theta(i, 0, j) + Psi(i, j) + PI / 4.d0) ! correct
          !print *, "expr1 is", expr1
          expr2 = exp(-tau) * (dPsi(i, j) - dPhi(i, j))         ! correct
          expr3 = (1.d0 / (k_old(j) * c)) * deriv1              ! correct
          expr4 = 3.d0 / (4.d0 * (k_old(j) * c)**2.d0) * deriv3    ! correct
          S_low_res(j, i) = expr1 + expr2 - expr3 + expr4       ! correct
!          print *, x_old(i), S_low_res(j, i)
          ! Calculation of low resolution (polarization) source function

          SE_low_res(j, i) = 3.d0 * g * H_p  * PI / (4.d0 * c * (k_old(j))**2.d0 *
              ↪ (eta_0 - eta_(i-1))**2.d0)

       end do
    end do

!    print *, "S low res is ", SE_low_res
```

```
    ! Step 2 - splining the source function & computing the coefficients
    allocate(coeff(4, 4, n_k, n_tot))
    allocate(coeffE(4, 4, n_k, n_tot))
    call splie2_full_precomp(k_old, x_old, S_low_res, coeff)
    call splie2_full_precomp(k_old, x_old, SE_low_res, coeffE)

!    print *, coeffE
    ! Step 3 - recomputing the grids and obtain high res source function
    allocate(k_new(n_k_new))
    allocate(x_new(n_tot_new))
    k_new(1) = k_min
    do j = 2, n_k_new
       k_new(j) = k_new(1) + (k_max - k_min) * ((j - 1.d0) / (n_k_new - 1.d0))**2
    end do
    x_init = log(a_init)
    x_today = log(a_today)
    x_new(1) = x_init
    x_step = (x_today - x_init) / n_tot_new
    do i = 2, n_tot_new
       x_new(i) = x_new(1) + (i-1) * x_step
    end do

   !==========================================!
   !      High Resolution Source Function     !
   !==========================================!
    allocate(S_high_res(n_k_new, n_tot_new))
    allocate(SE_high_res(n_k_new, n_tot_new))
    print *, "Start␣to␣calculate␣S_high_res"
    do i = 1, n_tot_new
       do j = 1, n_k_new
          S_high_res(j, i) = splin2_full_precomp(k_old, x_old, coeff, k_new(j),
              ↪ x_new(i))
          SE_high_res(j, i) = splin2_full_precomp(k_old, x_old, coeffE, k_new(j),
              ↪ x_new(i))
          !print *, x_new(i), S_high_res(j, i)
       end do
    end do
!    print *, x_old(1), x_new(1)
!    print *, x_old(n_tot), x_new(n_tot_new)
 !   print *, "S is calculated", S_high_res

    ! Saving source function to a file
    folder = "data/"
    filename = "S_high_res.unf"
    open(32, file = trim(folder) // trim(filename), form = "unformatted", action =
        ↪ "write", status = "replace")
    write(32) S_high_res
    close(32)
    filename = "SE_high_res.unf"
    open(33, file = trim(folder) // trim(filename), form = "unformatted", action =
        ↪ "write", status = "replace")
    write(33) SE_high_res
    close(33)

    filename = "x_new.unf"
    open(34, file = trim(folder) // trim(filename), form = "unformatted", action =
        ↪ "write", status = "replace")
    write(34) x_new
    close(34)

    filename = "k_new.unf"
```

```
    open(35, file = trim(folder) // trim(filename), form = "unformatted", action =
        ↪  "write", status = "replace")
    write(35) k_new
    close(35)

    ! Freeing-up the memory
    deallocate(x_old)
    deallocate(k_old)
    ! deltas
    deallocate(delta)
    deallocate(delta_b)
    deallocate(ddelta)
    deallocate(ddelta_b)
    ! velocity
    deallocate(v)
    deallocate(v_b)
    deallocate(dv)
    deallocate(dv_b)
    ! Potentials
    deallocate(Phi)
    deallocate(Psi)
    deallocate(dPhi)
    deallocate(dPsi)
    ! Theta
    deallocate(Theta)
    deallocate(dTheta)
    ! Polarisation
    deallocate(ThetaP)
    deallocate(dThetaP)
    ! Neutrinos
    deallocate(Nu)
    deallocate(dNu)
    ! Source function
!     deallocate(S_low_res)
!     deallocate(SE_low_res)

  end subroutine get_hires_source_function

end module source_func_mod
```

Listing 3: bessel_func_mod.f90

```
module bessel_func_mod
  use healpix_types
  use sphbess_mod
  use spline_1D_mod

  use rec_mod
  implicit none


contains

  subroutine compute_bessel_func(z_spline, j_l, j_l2)
    implicit none
    integer(i4b)        :: i, j, l
    character(len=1024) :: folder, filename
    ! Define global variables
    real(dp) :: x
    integer(i4b)                    :: l_num, n_spline
    real(dp),   pointer, dimension(:,:), intent(out) :: j_l, j_l2
    real(dp), allocatable, dimension(:), intent(out) :: z_spline!, j_l_spline,
        ↪ j_l_spline2
    integer(i4b), allocatable, dimension(:) :: ls

    ! Set up which l's to compute
    l_num = 44
    allocate(ls(l_num))
    ls = (/ 2, 3, 4, 6, 8, 10, 12, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100, &
         & 120, 140, 160, 180, 200, 225, 250, 275, 300, 350, 400, 450, 500, 550, &
         & 600, 650, 700, 750, 800, 850, 900, 950, 1000, 1050, 1100, 1150, 1200 /)
    n_spline = 5400
    ! Note: z is *not* redshift, but the dummy argument of j_l(z)
    allocate(z_spline(n_spline))
    allocate(j_l(n_spline, l_num))
    allocate(j_l2(n_spline, l_num))

    do l = 1, l_num
       do i = 1, n_spline
          z_spline(i) = 0.d0 + (i - 1) * 3500.d0 / (n_spline - 1.d0)
          if (i == 1) then
             j_l(i, l) = 0.d0
             if (l == 1) then
                j_l(i, l) = 1.d0
             end if
          else
             call sphbes(ls(l), z_spline(i), j_l(i,l))
          end if
       end do
       call spline(z_spline, j_l(:,l), yp1, ypn, j_l2(:,l))
    end do

!    print *, "bessel func is: ", bessel_jn(5, x)

!    do i = 1, n_spline
       !z_spline(i) = (i - 1) * 3500.d0 / (n_spline - 1.d0)
!       z_spline(i) = 0.d0 + (i - 1) * 3500.d0 / (n_spline - 1.d0)
!       do l = 1, l_num
!          if (i == 1)
!          j_l(i, l) =

!          if (z_spline(i) > 2.d0) then
```

```
!               call sphbes(ls(l), z_spline(i), j_l(i,l))
!            endif
!         end do
!      end do
     ! Splining Bessel function
!      do l = 1, l_num
!         call spline(z_spline, j_l(:,l), yp1, ypn, j_l2(:,l))
!      end do

     ! Saving splined Bessel functions
     folder = "data/"
     filename = "Bessel_func.unf"
     open(36, file = trim(folder) // trim(filename), form = "unformatted", action =
         ↪  "write", status = "replace")
     write(36) z_spline
     write(36) j_l
     write(36) j_l2
     close(36)

   end subroutine compute_bessel_func

end module bessel_func_mod
```

Listing 4: cl_mod.f90

```
module cl_mod
  use healpix_types
  use sphbess_mod
  use spline_1D_mod

  use rec_mod
  use evolution_mod
  use source_func_mod
  use bessel_func_mod
  implicit none

    integer(i4b)                                         :: l_num, x_num, n_spline
    integer(i4b),        allocatable, dimension(:)       :: ls
    real(dp),            allocatable, dimension(:)       :: ls_dp, l_hires
    ! Bessel function and its second derivative
    real(dp),            pointer,     dimension(:,:)     :: j_l, j_l2
    real(dp),            allocatable, dimension(:)       :: z_spline
    ! Variables I've defined
    ! Source function
    real(dp),            allocatable, dimension(:,:)     :: S_high_res, SE_high_res
    ! Transfer function
    real(dp),            allocatable, dimension(:,:)     :: Theta_l, ThetaE_l!, ls_dp,
        ↪  l_hires
    ! C_l
    real(dp),            allocatable, dimension(:)       :: C_l, CEE_l, CTE_l, C_l2,
        ↪ CEE_l2, CTE_l2
    real(dp),            allocatable, dimension(:)       :: C_l_splined, CEE_l_splined
        ↪ , CTE_l_splined
    real(dp),            allocatable, dimension(:,:,:) :: j_l_splined, integrand,
        ↪ integrand2
    real(dp),            allocatable, dimension(:,:)     :: integrandT, integrandE,
        ↪ integrandTE
!    real(dp),            allocatable, dimension(:,:)     :: integrand
    real(dp),            allocatable, dimension(:)       :: eta_
    real(dp)                                             :: eta_0, int, intE, intEE,
        ↪ intTE, expr1, expr2, factorial
    real(dp)                                             :: hx, heta, hk
    character(len=1024)                                  :: folder, filename_cmb_tt,
        ↪ filename_cmb_te, filename_cmb_ee
    character(len=1024), allocatable, dimension(:)       :: filename
    logical(lgt)                                         :: source_data_exists,
        ↪ bessel_data_exists, transfer_data_exists
    logical(lgt),        allocatable, dimension(:)       :: exists
    ! spectral index (of scalar perturbations)
    real(dp),                          parameter         :: n_spec = 0.96d0
contains

  !==================================================!
  !            MILESTONE 4: C_l's calculation         !
  !==================================================!

  ! Driver routine for (finally!) computing the CMB power spectrum
  subroutine compute_cls
    implicit none

    integer(i4b)                                      :: i, j, l, l_trick
!    real(dp)                                          :: dx, S_func, j_func, z, x0,
    ↪ x_min, x_max, d, e
!    integer(i4b), allocatable, dimension(:)          :: ls
!    real(dp),     allocatable, dimension(:)          :: integrand
```

```
!    real(dp),      pointer,      dimension(:,:)       :: j_l, j_l2
!    real(dp),      pointer,      dimension(:)         :: x_arg, int_arg, cls, cls2,
    ↪ ls_dp
    real(dp),      pointer,      dimension(:)         :: k, x
    real(dp),      pointer,      dimension(:,:,:,:) :: S_coeff
!    real(dp),      pointer,      dimension(:,:)       :: S, S2
!    real(dp),      allocatable, dimension(:,:)       :: Theta
!    real(dp),      allocatable, dimension(:)         :: z_spline !, j_l_spline,
    ↪ j_l_spline2
!    real(dp),      allocatable, dimension(:)         :: x_hires, k_hires


    ! Set up which l's to compute
    l_num = 44
    allocate(ls(l_num))
    ls = (/ 2, 3, 4, 6, 8, 10, 12, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100, &
         & 120, 140, 160, 180, 200, 225, 250, 275, 300, 350, 400, 450, 500, 550, &
         & 600, 650, 700, 750, 800, 850, 900, 950, 1000, 1050, 1100, 1150, 1200 /)

    ! Task: Get source function from evolution_mod
    ! I've created a module to compute the source function.
    ! Checking whether source function was alreeady computed
    ! (and stored on disc). If it wasn't, will be computed now.
    allocate(S_high_res(n_k_new, n_tot_new))
    allocate(SE_high_res(n_k_new, n_tot_new))
    allocate(filename(20))
    allocate(exists(20))
    folder = "data/"
    filename = (/ "S_high_res.unf", "SE_high_res.unf", "x_new.unf", "k_new.unf", "
        ↪ Bessel_func.unf", &
         & "Theta_l.unf", "ThetaE_l.unf", "integrandT.unf", "integrandE.unf", "
             ↪ integrandTE.unf", &
         & "j_l_splined.unf" /)

!    print *, SE_high_res
!    call get_hires_source_function(S_high_res, SE_high_res)

    ! Task: Initialize spherical Bessel functions for each l; use 5400 sampled
        ↪ points between
    !       z = 0 and 3500. Each function must be properly splined
    ! Hint: It may be useful for speed to store the splined objects on disk in an
        ↪ unformatted
    !       Fortran (= binary) file, so that these only has to be computed once.
        ↪ Then, if your
    !       cache file exists, read from that; if not, generate the j_l's on the
        ↪ fly.
    n_spline = 5400
    allocate(z_spline(n_spline))    ! Note: z is *not* redshift, but simply the
        ↪ dummy argument of j_l(z)
    allocate(j_l(n_spline, l_num))
    allocate(j_l2(n_spline, l_num))
    ! Overall task: Compute the C_l's for each given l
    allocate(eta_(n_tot_new))
    allocate(j_l_splined(l_num, n_k_new, n_tot_new))
    allocate(integrand(l_num, n_k_new, n_tot_new))
    allocate(integrand2(l_num, n_k_new, n_tot_new))
    ! Theta^2_l/k etc.
    allocate(integrandT(l_num, n_k_new))
    allocate(integrandE(l_num, n_k_new))
    allocate(integrandTE(l_num, n_k_new))
    ! Transfer function
    allocate(Theta_l(n_k_new, l_num))
```

```
allocate(ThetaE_l(n_k_new, l_num))
! C_ls
allocate(C_l(l_num), C_l2(l_num))
allocate(CEE_l(l_num), CEE_l2(l_num))
allocate(CTE_l(l_num), CTE_l2(l_num))
! Splined C_ls
allocate(C_l_splined(1200))
allocate(CEE_l_splined(1200))
allocate(CTE_l_splined(1200))

! Looping through all the files and checking whether they exist
do i = 1, 4
   inquire(file = trim(folder) // trim(filename(i)), exist = exists(i))
   if (exists(i)) then
      source_data_exists = .True.
   else
      source_data_exists = .False.
      exit
   end if
end do

! If any of the files doesn't exist, then it will create one
if (source_data_exists == .False.) then
   print *, "Source␣func␣data␣doesn't␣exist,␣so␣will␣be␣created"
   call get_hires_source_function(S_high_res, SE_high_res)
else
   print *, "Source␣func␣data␣exists,␣so␣will␣be␣retrieved"
   open(32, file = trim(folder) // trim(filename(1)), form = "unformatted",
       ↪ action = "read")
   read(32) S_high_res
   close(32)
   open(33, file = trim(folder) // trim(filename(2)), form = "unformatted",
       ↪ action = "read")
   read(33) SE_high_res
   close(33)
   ! Retrieving high resolution grid
   allocate(k_new(n_k_new))
   allocate(x_new(n_tot_new))
   open(34, file = trim(folder) // trim(filename(3)), form = "unformatted",
       ↪ action = "read")
   read(34) x_new
   close(34)
   open(35, file = trim(folder) // trim(filename(4)), form = "unformatted",
       ↪ action = "read")
   read(35) k_new
   close(35)
end if

inquire(file = trim(folder) // trim(filename(5)), exist = exists(5))
if ((exists(5) == .True.)) then
   bessel_data_exists = .True.
else
   bessel_data_exists = .False.
end if
if (bessel_data_exists == .False.) then
   print *, "Bessel␣func␣data␣doesn't␣exist,␣so␣will␣be␣created"
   call compute_bessel_func(z_spline, j_l, j_l2)
else
   print *, "Bessel␣func␣data␣exists,␣so␣will␣be␣retrieved"
   open(36, file = trim(folder) // trim(filename(5)), form = "unformatted",
       ↪ action = "read")
   read(36) z_spline
```

```
      read(36) j_l
      read(36) j_l2
      close(36)
   end if

   ! Splining bessel function
   inquire(file = trim(folder) // trim(filename(11)), exist = exists(11))
   if (exists(11) == .True.) then
      bessel_data_exists = .True.
   else
      bessel_data_exists = .False.
   end if
   if (bessel_data_exists == .False.) then
      print *, "Bessel␣func␣needs␣to␣be␣splined"
      call compute_splined_j_l()
   else
      print *, "Retrieving␣splined␣Bessel␣func"
      open(37, file = trim(folder) // trim(filename(11)), form = "unformatted",
          ↪ action = "read")
      read(37) j_l_splined
      close(37)
   end if

   ! Overall task: Compute the C_l's for each given l

   !==========================!
   ! Compute Transfer Function !
   !==========================!
   ! Inquire Transfer function whereabouts:
   ! If one of the files (i.e. Source function and/or Bessel functions)
   ! doesn't exist, it will recompute Transfer functions. On the contrary,
   ! if one of the variables is false it means that data have been modified
   ! and so needs to be recomputed
   do i = 6, 7
      inquire(file = trim(folder) // trim(filename(i)), exist = exists(i))
      if ((exists(i) == .True.) .and. (bessel_data_exists == .True.) .and. (
          ↪ source_data_exists == .True.)) then
         transfer_data_exists = .True.
      else
         transfer_data_exists = .False.
         exit
      end if
   end do

   if (transfer_data_exists == .False.) then
      print *, "Transfer␣func␣data␣doesn't␣exist,␣so␣will␣be␣created"
      call compute_trasfer_func
   else
      print *, "Transfer␣func␣data␣exists,␣so␣will␣be␣retrieved"
      open(38, file = trim(folder) // trim(filename(6)), form = "unformatted",
          ↪ action = "read")
      read(38) Theta_l
      close(38)
      open(39, file = trim(folder) // trim(filename(7)), form = "unformatted",
          ↪ action = "read")
      read(39) ThetaE_l
      close(39)
   end if

   ! Step for trapezoidal method
   hk   = (k_new(n_k_new) - k_new(1)) / n_k_new
   do l = 1, l_num
```

```
        int    = 0.d0
        intEE = 0.d0
        intTE = 0.d0
        do j = 1, n_k_new
           integrandT(j, l)  = (c * k_new(j) / H_0)**(n_spec - 1.d0) * (Theta_l(j,
               ↪ l))**2.d0 / k_new(j)
           integrandE(j, l)  = (c * k_new(j) / H_0)**(n_spec - 1.d0) * (ThetaE_l(j,
               ↪  l))**2.d0 / k_new(j)
           integrandTE(j, l) = (c * k_new(j) / H_0)**(n_spec - 1.d0) * Theta_l(j, l
               ↪ ) * ThetaE_l(j, l) / k_new(j)
        end do
        int    = 0.5d0 * (integrandT(1, l) + integrandT(n_k_new, l))
        intE   = 0.5d0 * (integrandE(1, l) + integrandE(n_k_new, l))
        intTE = 0.5d0 * (integrandTE(1, l) + integrandTE(n_k_new, l))
        do j = 2, (n_k_new-1)
          int    = int    + integrandT(j, l)
          intEE = intEE + integrandE(j, l)
          intTE = intTE + integrandTE(j, l)
       end do
       !Store C_l in an array.
       C_l(l)    = hk * int    * ls(l) * (ls(l) + 1.d0) / (2.d0 * pi)
       CEE_l(l) = hk * intEE * ls(l) * (ls(l) + 1.d0) / (2.d0 * pi)
       CTE_l(l) = hk * intTE * ls(l) * (ls(l) + 1.d0) / (2.d0 * pi)
     end do

        ! Task: Compute the transfer function, Theta_l(k)
        ! It has already been computed and/or retrieved
        ! Task: Integrate P(k) * (Theta_l^2 / k) over k to find un-normalized C_l's

     ! Need to convert ls to double precision to be able to use spline
     allocate(ls_dp(l_num))
     do l = 1, l_num
         ls_dp(l) = ls(l)
     end do

     ! Task: Spline C_l's found above, and output smooth C_l curve for each integer
         ↪  l
     call spline(ls_dp, C_l,    yp1, ypn, C_l2)
     call spline(ls_dp, CEE_l, yp1, ypn, CEE_l2)
     call spline(ls_dp, CTE_l, yp1, ypn, CTE_l2)

     allocate(l_hires(1200))
     do l = 1, 1200
         l_hires(l) = l
     end do

     ! Because our array of ls started from 2 (not 1), I do a little trick here as
         ↪ well
     do l = 1, 1200
        C_l_splined(l)    = splint(ls_dp, C_l, C_l2, l_hires(l))
        !print *, C_l_splined(l)
!        print *, l_trick * (l_trick + 1) * C_l_splined(l_trick) * H_0 / (2 * pi *
     ↪ c)
        CEE_l_splined(l) = splint(ls_dp, CEE_l, CEE_l2, l_hires(l))
        CTE_l_splined(l) = splint(ls_dp, CTE_l, CTE_l2, l_hires(l))
     end do

     filename_cmb_tt = "CTT_l.dat"
     open(43, file = trim(folder) // trim(filename_cmb_tt), action = "write",
         ↪ status = "replace")
     do l = 2, 1200
        write(43,*) l, "␣", C_l_splined(l)
```

```
    end do
    close(43)
    filename_cmb_ee = "CEE_l.dat"
    open(44, file = trim(folder) // trim(filename_cmb_ee), action = "write",
        ↪ status = "replace")
    do l = 2, 1200
       write(44,*) l, "␣", CEE_l_splined(l)
    end do
    close(44)
    filename_cmb_te = "CTE_l.dat"
    open(45, file = trim(folder) // trim(filename_cmb_te), action = "write",
        ↪ status = "replace")
    do l = 2, 1200
       write(45,*) l, "␣", CTE_l_splined(l)
    end do
    close(45)
!    print *, C_l_splined

    ! Freeing the memory
    deallocate(S_high_res)
    deallocate(SE_high_res)

    deallocate(z_spline)
    deallocate(j_l)
    deallocate(j_l2)

    deallocate(eta_)
    deallocate(j_l_splined)
    deallocate(integrand)
    deallocate(integrand2)
    ! Transfer function
    deallocate(Theta_l)
    deallocate(ThetaE_l)
    ! Theta^2/k etc.
    deallocate(integrandT)
    deallocate(integrandE)
    deallocate(integrandTE)
    ! C_ls
    deallocate(C_l, C_l2)
    deallocate(CEE_l, CEE_l2)
    deallocate(CTE_l, CTE_l2)
    ! Splined C_ls
    deallocate(C_l_splined)
    deallocate(CEE_l_splined)
    deallocate(CTE_l_splined)

  end subroutine compute_cls

  ! Routine to compute Transfer funciton
  subroutine compute_trasfer_func()
    implicit none

    integer(i4b)         :: i, j, l
    character(len=1024) :: filename

    eta_0 = get_eta(x_new(n_tot_new))
    int = 0.d0
    intE = 0.d0
    ! Factorial for computing ThetaE_l
    factorial = 1.d0

    ! Step for trapezoid method
```

```
  hx   = (x_new(n_tot_new) - x_new(1)) / n_tot_new
  heta = (eta_0  - get_eta(x_new(1)))  / n_tot_new
  ! Precompute intermediate quantities (eta, splined j_l, integrand etc.)
  do l = 1, l_num
     do j =  1, n_k_new
        ! Transfer Function
        ! Computing integrand for trapezoidal method
        do i = 1, n_tot_new
           integrand(l, j, i)  = S_high_res(j, i)  * j_l_splined(l, j, i) !*
               ↪ get_j_l(l, k_new(j), x_new(i))!* H_0
           integrand2(l, j, i) = SE_high_res(j, i) * j_l_splined(l, j, i) !
               ↪ get_j_l(l, k_new(j), x_new(i))
        end do
        ! Computing first part of trapezoidal method for Theta_l:
        Theta_l(j, l) = 0.5d0 * (integrand(l, j, 1) + integrand(l, j, n_tot_new)
            ↪ )
        ! For Theta^E_l
        ! factorial = (l+2)!/(l-2)! = (l-1) * l * (l+1) * (l+2)
        factorial = 1.d0 * (ls(l) - 1.d0) * ls(l) * (ls(l) + 1.d0) * (ls(l) + 2.
            ↪ d0)
        ThetaE_l(j, l) = 0.5d0 * (integrand2(l, j, 1) + integrand2(l, j,
            ↪ n_tot_new))
       do i = 2, n_tot_new - 1
           Theta_l(j, l)  = Theta_l(j, l) + integrand(l, j, i)
          ThetaE_l(j, l) = ThetaE_l(j, l) + integrand2(l, j, i)
        end do
        Theta_l(j, l)  = hx * Theta_l(j, l)
        ThetaE_l(j, l) = factorial * heta * ThetaE_l(j, l)
     end do
  end do

  ! Saving data to binary file
  ! First, integrand, to plot it as intermediate step
  folder = "data/"
  filename = "integrand.unf"
  open(37, file = trim(folder) // trim(filename), form = "unformatted", action =
       ↪ "write", status = "replace")
  write(37) integrand
  close(37)
  filename = "integrand2.unf"
  open(37, file = trim(folder) // trim(filename), form = "unformatted", action =
       ↪ "write", status = "replace")
  write(37) integrand2
  close(37)
  ! Second, transfer function
  filename = "Theta_l.unf"
  open(38, file = trim(folder) // trim(filename), form = "unformatted", action =
       ↪ "write", status = "replace")
  write(38) Theta_l
  close(38)
  filename = "ThetaE_l.unf"
  open(39, file = trim(folder) // trim(filename), form = "unformatted", action =
       ↪ "write", status = "replace")
  write(39) ThetaE_l
  close(39)

end subroutine compute_trasfer_func

subroutine compute_splined_j_l()
  implicit none

  integer(i4b)        :: i, j, l
```

```
   character(len=1024) :: filename

   do l = 1, l_num
      do j =  1, n_k_new
         ! Transfer Function
         ! Computing integrand for trapezoidal method
         do i = 1, n_tot_new
            j_l_splined(l, j, i) = splint(z_spline, j_l(:,l), j_l2(:,l), k_new(j)
                ↪  * &
                 & (get_eta(x_new(n_tot_new)) - get_eta(x_new(i))))
         end do
      end do
   end do

   ! Writing splined function to a file
   filename = "j_l_splined.unf"
   open(42, file = trim(folder) // trim(filename), form = "unformatted", action =
       ↪  "write", status = "replace")
   write(42) j_l_splined
   close(42)
end subroutine compute_splined_j_l

! Subroitine for saving and plotting data
subroutine save_milestone4_plot_data(l, k)
   implicit none
   ! k is the index in the existing high resolution grid
   ! l is the actual value (so I need to find the
   ! corresponding one in ls => look below)
   integer(i4b), intent(in) :: l, k
   integer(i4b)             :: i, j, ls_index
   character(len=1024)      :: folder, filename
   ! format descriptor
   character(len=1024)      :: format_string, k1, l1

   folder = "data/"
   ! an integer of width 4 with zeros on the left if the value is not enough
   format_string = "(I4.4)"
   ! converting integer to string using an 'internal file'
   write (k1, format_string) k

   ! an integer of width 3 with zeros on the left if the value is not enough
   format_string = "(I4.4)"
   ! converting integer to string using an 'internal file'
   write (l1, format_string) l

   ! Goins through ls values and return
   ! an index which corresponds to input value
   do j = 1, size(ls)
      if (l == ls(j)) then
         ls_index = j
      end if
   end do

   ! Source function
   allocate(S_high_res(n_k_new, n_tot_new))
   allocate(SE_high_res(n_k_new, n_tot_new))
   filename = "S_high_res.unf"
   open(32, file = trim(folder) // trim(filename), form = "unformatted", action =
       ↪  "read")
   read(32) S_high_res
   close(32)
   filename = "SE_high_res.unf"
```

```
    open(33, file = trim(folder) // trim(filename), form = "unformatted", action =
        ↪  "read")
    read(33) SE_high_res
    close(33)
    filename = "S_"//trim(k1)//"_x.dat"
    open(32, file = trim(folder) // trim(filename), action = "write", status = "
        ↪ replace")
    do i = 1, n_tot_new
       write(32,*) x_new(i), "␣", S_high_res(k, i)
!        print *, x_new(i), S_high_res(k, i)
    end do
    close(32)
    filename = "SE_"//trim(k1)//"_x.dat"
    open(33, file = trim(folder) // trim(filename), action = "write", status = "
        ↪ replace")
    do i = 1, n_tot_new
       write(33,*) x_new(i), "␣", SE_high_res(k, i)
    end do
    close(33)
    deallocate(S_high_res)
    deallocate(SE_high_res)

    ! Saving integrand, to plot and to check whether I am on the right track
    allocate(integrand(l_num, n_k_new, n_tot_new))
    allocate(integrand2(l_num, n_k_new, n_tot_new))
    filename = "integrand.unf"
    open(38, file = trim(folder) // trim(filename), form = "unformatted", action =
        ↪  "read")
    read(38) integrand
    close(38)
    ! File ctructure: Name_l_k_x.dat
    filename = "int_"//trim(l1)//"_"//trim(k1)//"_x.dat"
    open(38, file = trim(folder) // trim(filename), action = "write", status = "
        ↪ replace")
    do i = 1, n_tot_new
       write(38,*) x_new(i), "␣", integrand(ls_index, k, i) * 10**3.d0
    end do
    close(38)
    filename = "integrand2.unf"
    open(38, file = trim(folder) // trim(filename), form = "unformatted", action =
        ↪  "read")
    read(38) integrand2
    close(38)
    filename = "int2_"//trim(l1)//"_"//trim(k1)//"_x.dat"
    open(38, file = trim(folder) // trim(filename), action = "write", status = "
        ↪ replace")
    do i = 1, n_tot_new
       write(38,*) x_new(i), "␣", integrand2(ls_index, k, i) * 10**3.d0
    end do
    close(38)
    deallocate(integrand2)
    deallocate(integrand)


    ! Transfer function
    allocate(Theta_l(n_k_new, l_num))
    allocate(ThetaE_l(n_k_new, l_num))
    filename = "Theta_l.unf"
    open(38, file = trim(folder) // trim(filename), form = "unformatted", action =
        ↪  "read")
    read(38) Theta_l
    close(38)
```

```
    filename = "ThetaE_l.unf"
    open(39, file = trim(folder) // trim(filename), form = "unformatted", action =
        ↪  "read")
    read(39) ThetaE_l
    close(39)
    ! File structure: Name_l_k.dat
    filename = "ThetaT_"//trim(l1)//"_k.dat"
    open(38, file = trim(folder) // trim(filename), action = "write", status = "
        ↪ replace")
    do i = 1, n_k_new
       write(38,*) (k_new(i) * c / H_0), "␣", Theta_l(i, ls_index)
    end do
    close(38)
    filename = "ThetaE_"//trim(l1)//"_k.dat"
    open(39, file = trim(folder) // trim(filename), action = "write", status = "
        ↪ replace")
    do i = 1, n_k_new
       write(39,*) (k_new(i) * c / H_0), "␣", ThetaE_l(i, ls_index)
    end do
    close(39)
!     deallocate(Theta_l)
!     deallocate(ThetaE_l)

    ! Integrand in angular power spectrum
    ! Theta_l^2/k etc.

    ! File structure: Name_l_k.dat
    filename = "intT_"//trim(l1)//"_k.dat"
    open(40, file = trim(folder) // trim(filename), action = "write", status = "
        ↪ replace")
    do i = 1, n_k_new
       write(40,*) (k_new(i) * c / H_0), "␣", ls(ls_index) * (ls(ls_index) + 1.d0)
           ↪  * &
            & (Theta_l(i, ls_index))**2.d0 * H_0 /  (c * k_new(i))
    end do
    close(40)
    filename = "intE_"//trim(l1)//"_k.dat"
    open(41, file = trim(folder) // trim(filename), action = "write", status = "
        ↪ replace")
    do i = 1, n_k_new
       write(41,*) (k_new(i) * c / H_0), "␣", ls(ls_index) * (ls(ls_index) + 1.d0)
           ↪  * &
            & (ThetaE_l(i, ls_index))**2.d0 * H_0 / (c * k_new(i))
    end do
    close(41)
    filename = "intTE_"//trim(l1)//"_k.dat"
    open(42, file = trim(folder) // trim(filename), action = "write", status = "
        ↪ replace")
    do i = 1, n_k_new
       write(42,*) (k_new(i) * c / H_0), "␣", ls(ls_index) * (ls(ls_index) + 1.d0)
           ↪  * &
            & Theta_l(i, ls_index) * ThetaE_l(i, ls_index) * H_0 / (c * k_new(i))
    end do
    close(42)
    deallocate(Theta_l)
    deallocate(ThetaE_l)

  end subroutine save_milestone4_plot_data

end module cl_mod
```

Listing 5: Milestone4_DataPlots.ipynb

```
import os
import sys
import numpy as np
# library for plotting
import plotly
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

plotly.__version__

# Getting the path to current (working) directory
currentDirPath = os.getcwd()
#print(dirpath)

# Checking if data dir exists
inputDir  = 'data'
outputDir = 'plots'
if (os.path.isdir(inputDir) == False):
    print("Data␣directory␣doesn't␣exist!")
    sys.exit(0)
else:
    # Checking if directory for outputting plots exists
    # if it doesn't it will create one
    if (os.path.isdir(outputDir) == False):
        os.mkdir(outputDir)

    plt.init_notebook_mode(connected = True)

    ######
    #Data#
    ######
    # Reading data from a file
    # Milestone 1
    omegaDataFile = np.loadtxt(inputDir + '/' + 'Omega_all.dat')
    etaDataFile   = np.loadtxt(inputDir + '/' + 'eta_x.dat')
    HxDataFile    = np.loadtxt(inputDir + '/' + 'H_x.dat')
    HzDataFile    = np.loadtxt(inputDir + '/' + 'H_z.dat')

    #etaDataFile   = np.loadtxt('eta_x_data.dat')

    # Milestone 1
    # Omega_X(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(z):
Z               = HzDataFile[:,0]
H_z             = HzDataFile[:,1]


##########
#Plotting#
##########

#
    ↪ ########################################################################################
    ↪
# Milestone 1
#
    ↪ ########################################################################################
    ↪
# 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 = '$x$', mirror = True, ticks = 'outside',
                               showline = True),
                  yaxis = dict(title = '$\Omega(x)$', type='log', autorange = True
                      ↪ ,
                               mirror = True, ticks = 'outside',
                               showline = True),
                  )
    etaLayoutPlot = dict(#title = 'Conformal Time',
                  xaxis = dict(title = '$x$',
                               mirror = True, ticks = 'outside',
                               showline = True),
                  yaxis = dict(title = '$\eta(x)␣[pc]$', type = 'log', autorange =
                      ↪ True,
                               mirror = True, ticks = 'outside',
                               showline = True),
                  )
    HxLayoutPlot  = dict(#title = 'Hubble Parameter',
                  xaxis = dict(title = '$x$', mirror = True, ticks = 'outside',
                               showline = True),
                  yaxis = dict(title = '$H(x)␣[s^{-1}]$', type = 'log', autorange
```

```
                        ↪ = True,
                                  mirror = True, ticks = 'outside',
                                  showline = True),
                    )
HzLayoutPlot    = dict(#title = 'Hubble Parameter',
                    xaxis = dict(title = '$z$', mirror = True, ticks = 'outside',
                                  showline = True),
                    yaxis = dict(title = '$H(z)␣[s^{-1}]$', type = 'log', autorange
                        ↪ = True,
                                  mirror = True, ticks = 'outside',
                                  showline = 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, outputDir + '/' + 'Omega_x.pdf')
pio.write_image(etaFigure,   outputDir + '/' + 'eta_x.pdf')
pio.write_image(HxFigure,    outputDir + '/' + 'H_x.pdf')
pio.write_image(HzFigure,    outputDir + '/' + 'H_z.pdf')

#
   ↪ ##################################################################################
   ↪
# Milestone 2
#
   ↪ ##################################################################################
   ↪
XeDataFile        = np.loadtxt(inputDir + '/' + "X_e_z.dat")

tauDataFile       = np.loadtxt(inputDir + '/' + "tau_x.dat")
dtauDataFile      = np.loadtxt(inputDir + '/' + "dtau_x.dat")
ddtauDataFile     = np.loadtxt(inputDir + '/' + "ddtau_x.dat")

gtildeDataFile    = np.loadtxt(inputDir + '/' + "g_x.dat")
dgtildeDataFile   = np.loadtxt(inputDir + '/' + "dg_x.dat")
ddgtildeDataFile  = np.loadtxt(inputDir + '/' + "ddg_x.dat")

# X_e(z):
X4                = XeDataFile[:,0]
Xe                = XeDataFile[:,1]
# tau(x) and the derivatives:
X5                = tauDataFile[:,0]
tau               = tauDataFile[:,1]
dtau              = np.abs(dtauDataFile[:,1])
ddtau             = ddtauDataFile[:,1]
# g(x) and its derivatives:
X6                = gtildeDataFile[:,0]
gtilde            = gtildeDataFile[:,1]
dgtilde           = dgtildeDataFile[:,1]/10
ddgtilde          = ddgtildeDataFile[:,1]/200

# X_e(z):
```

```
XeTrace = go.Scatter(
    x = X4,
    y = Xe,
    name = '$X_e$',
    line = dict(
        color = ('blue'),#'rgb(300, 200, 100)'),
        width = 3))

# tau(x) and its derivatives:
tauTrace = go.Scatter(
    x = X5,
    y = tau,
    name = '$\\tau$',
    line = dict(
        color = ('red'),#'rgb(100, 20, 50)'),
        width = 3))

dtauTrace = go.Scatter(
    x = X5,
    y = dtau,
    name = "$|\\tau'(x)|$",
    line = dict(
        color = ('orange'),#'rgb(205, 12, 24)'),
        width = 3,
        dash  = "dash"))

ddtauTrace = go.Scatter(
    x = X5,
    y = ddtau,
    name = "$\\tau''(x)$",
    line = dict(
        color = ('blue'),#'rgb(300, 200, 100)'),
        width = 3,
        dash  = "dot"))

# g(x) and its derivatives:

gtildeTrace = go.Scatter(
    x = X6,
    y = gtilde,
    name = '$g(x)$',
    line = dict(
        color = ('red'),#'rgb(100, 20, 50)'),
        width = 3))

dgtildeTrace = go.Scatter(
    x = X6,
    y = dgtilde,
    name = "$g'(x)$",
    line = dict(
        color = ('orange'),#'rgb(205, 12, 24)'),
        width = 3,
        dash  = "dash"))

ddgtildeTrace = go.Scatter(
    x = X6,
    y = ddgtilde,
    name = "$g''(x)$",
    line = dict(
        color = ('blue'),#'rgb(300, 200, 100)'),
        width = 3,
        dash  = "dot"))
```

```
XeDataPlot      = [XeTrace]
tauDataPlot     = [tauTrace, dtauTrace, ddtauTrace]
gtildeDataPlot = [gtildeTrace, dgtildeTrace, ddgtildeTrace]

XeLayoutPlot  = dict(#title = 'Xe',
              xaxis = dict(title = '$z$', range = [1800, 0],
                           mirror = True, ticks = 'outside',
                           showline = True),
              yaxis = dict(title = '$X_e$', type = 'log', autorange = True,
                           mirror = True, ticks = 'outside',
                           showline = True),
              )

tauLayoutPlot = dict(#title = 'tau(x) and its derivatives',
              xaxis = dict(title = '$x$', range =[-20, 1],
                           mirror = True, ticks = 'outside',
                           showline = True),
              yaxis = dict(title = "$\\tau(x),␣|\\tau'(x)|,␣\\tau''(x)$",
                           type = 'log', autorange = True,
                           mirror = True, ticks = 'outside',
                           showline = True)
              )

gtildeLayoutPlot = dict(#title = 'tau(x) and its derivatives',
              xaxis = dict(title = '$x$', range = [-9, -5],
                           mirror = True, ticks = 'outside',
                           showline = True,),
              yaxis = dict(title = "$\\tilde{g}(x),␣\\tilde{g}'(x),␣\\tilde{g
                    ↪ }''(x)$",
                           type = 'linear', autorange = True,
                           mirror = True, ticks = 'outside',
                           showline = True)
              )

XeFigure     = dict(data = XeDataPlot, layout = XeLayoutPlot)
tauFigure    = dict(data = tauDataPlot, layout = tauLayoutPlot)
gtildeFigure = dict(data = gtildeDataPlot, layout = gtildeLayoutPlot)

# Plotting everything
plt.iplot(XeFigure,     filename = 'Xe')
plt.iplot(tauFigure,    filename = 'tau')
plt.iplot(gtildeFigure, filename = 'gtilde')

# Saving plots
pio.write_image(XeFigure,     outputDir + '/' + 'X_e_z.pdf')
pio.write_image(tauFigure,    outputDir + '/' + 'tau_x.pdf')
pio.write_image(gtildeFigure, outputDir + '/' + 'g_x.pdf')

#
    ↪ ##################################################################################
    ↪
# Milestone 3
#
    ↪ ##################################################################################
    ↪

# delta(x):
deltaDataFile  = [
                np.loadtxt(inputDir + '/' + "delta_001_x.dat"),
                np.loadtxt(inputDir + '/' + "delta_010_x.dat"),
                np.loadtxt(inputDir + '/' + "delta_022_x.dat"),
```

```
                np.loadtxt(inputDir + '/' + "delta_040_x.dat"),
                np.loadtxt(inputDir + '/' + "delta_060_x.dat"),
                np.loadtxt(inputDir + '/' + "delta_100_x.dat")]

deltabDataFile = [
                np.loadtxt(inputDir + '/' + "deltab_001_x.dat"),
                np.loadtxt(inputDir + '/' + "deltab_010_x.dat"),
                np.loadtxt(inputDir + '/' + "deltab_022_x.dat"),
                np.loadtxt(inputDir + '/' + "deltab_040_x.dat"),
                np.loadtxt(inputDir + '/' + "deltab_060_x.dat"),
                np.loadtxt(inputDir + '/' + "deltab_100_x.dat")]

# v(x):
vDataFile      = [
                np.loadtxt(inputDir + '/' + "v_001_x.dat"),
                np.loadtxt(inputDir + '/' + "v_010_x.dat"),
                np.loadtxt(inputDir + '/' + "v_022_x.dat"),
                np.loadtxt(inputDir + '/' + "v_040_x.dat"),
                np.loadtxt(inputDir + '/' + "v_060_x.dat"),
                np.loadtxt(inputDir + '/' + "v_100_x.dat")]

# v_b(x):
vbDataFile = [
                np.loadtxt(inputDir + '/' + "vb_001_x.dat"),
                np.loadtxt(inputDir + '/' + "vb_010_x.dat"),
                np.loadtxt(inputDir + '/' + "vb_022_x.dat"),
                np.loadtxt(inputDir + '/' + "vb_040_x.dat"),
                np.loadtxt(inputDir + '/' + "vb_060_x.dat"),
                np.loadtxt(inputDir + '/' + "vb_100_x.dat")]

# Phi(x):
PhiDataFile = [
                np.loadtxt(inputDir + '/' + "Phi_001_x.dat"),
                np.loadtxt(inputDir + '/' + "Phi_010_x.dat"),
                np.loadtxt(inputDir + '/' + "Phi_022_x.dat"),
                np.loadtxt(inputDir + '/' + "Phi_040_x.dat"),
                np.loadtxt(inputDir + '/' + "Phi_060_x.dat"),
                np.loadtxt(inputDir + '/' + "Phi_100_x.dat")]

# Psi(x):
PsiDataFile = [
                np.loadtxt(inputDir + '/' + "Psi_001_x.dat"),
                np.loadtxt(inputDir + '/' + "Psi_010_x.dat"),
                np.loadtxt(inputDir + '/' + "Psi_022_x.dat"),
                np.loadtxt(inputDir + '/' + "Psi_040_x.dat"),
                np.loadtxt(inputDir + '/' + "Psi_060_x.dat"),
                np.loadtxt(inputDir + '/' + "Psi_100_x.dat")]

# Theta_0(x):
Theta0DataFile = [
                np.loadtxt(inputDir + '/' + "Theta_0_001_x.dat"),
                np.loadtxt(inputDir + '/' + "Theta_0_010_x.dat"),
                np.loadtxt(inputDir + '/' + "Theta_0_022_x.dat"),
                np.loadtxt(inputDir + '/' + "Theta_0_040_x.dat"),
                np.loadtxt(inputDir + '/' + "Theta_0_060_x.dat"),
                np.loadtxt(inputDir + '/' + "Theta_0_100_x.dat")]
# Theta_1(x):
Theta1DataFile  = [
                np.loadtxt(inputDir + '/' + "Theta_1_001_x.dat"),
                np.loadtxt(inputDir + '/' + "Theta_1_010_x.dat"),
                np.loadtxt(inputDir + '/' + "Theta_1_022_x.dat"),
                np.loadtxt(inputDir + '/' + "Theta_1_040_x.dat"),
```

```
                    np.loadtxt(inputDir + '/' + "Theta_1_060_x.dat"),
                    np.loadtxt(inputDir + '/' + "Theta_1_100_x.dat")]

    # ThetaP_0(x):
    ThetaP0DataFile = [
                    np.loadtxt(inputDir + '/' + "ThetaP_0_001_x.dat"),
                    np.loadtxt(inputDir + '/' + "ThetaP_0_010_x.dat"),
                    np.loadtxt(inputDir + '/' + "ThetaP_0_022_x.dat"),
                    np.loadtxt(inputDir + '/' + "ThetaP_0_040_x.dat"),
                    np.loadtxt(inputDir + '/' + "ThetaP_0_060_x.dat"),
                    np.loadtxt(inputDir + '/' + "ThetaP_0_100_x.dat")]

    # ThetaP_1(x):
    ThetaP1DataFile = [
                    np.loadtxt(inputDir + '/' + "ThetaP_1_001_x.dat"),
                    np.loadtxt(inputDir + '/' + "ThetaP_1_010_x.dat"),
                    np.loadtxt(inputDir + '/' + "ThetaP_1_022_x.dat"),
                    np.loadtxt(inputDir + '/' + "ThetaP_1_040_x.dat"),
                    np.loadtxt(inputDir + '/' + "ThetaP_1_060_x.dat"),
                    np.loadtxt(inputDir + '/' + "ThetaP_1_100_x.dat")]

    # Nu_0(x):
    Nu0DataFile = [
                    np.loadtxt(inputDir + '/' + "Nu_0_001_x.dat"),
                    np.loadtxt(inputDir + '/' + "Nu_0_010_x.dat"),
                    np.loadtxt(inputDir + '/' + "Nu_0_022_x.dat"),
                    np.loadtxt(inputDir + '/' + "Nu_0_040_x.dat"),
                    np.loadtxt(inputDir + '/' + "Nu_0_060_x.dat"),
                    np.loadtxt(inputDir + '/' + "Nu_0_100_x.dat")]

    # Nu_1(x):
    Nu1DataFile = [
                    np.loadtxt(inputDir + '/' + "Nu_1_001_x.dat"),
                    np.loadtxt(inputDir + '/' + "Nu_1_010_x.dat"),
                    np.loadtxt(inputDir + '/' + "Nu_1_022_x.dat"),
                    np.loadtxt(inputDir + '/' + "Nu_1_040_x.dat"),
                    np.loadtxt(inputDir + '/' + "Nu_1_060_x.dat"),
                    np.loadtxt(inputDir + '/' + "Nu_1_100_x.dat")]

    Colors = ['blue', 'green', 'yellow', 'orange', 'red', 'purple', 'black']
    k = ['$k␣=␣0.1␣H_0␣/␣c$', '$k␣=␣10␣H_0␣/␣c$', '$k␣=␣50␣H_0␣/␣c$',
         '$k␣=␣160␣H_0␣/␣c$', '$k␣=␣360␣H_0␣/␣c$','$k␣=␣1000␣H_0␣/␣c$']

    # Grid:
    X7           = [sublist[0] for sublist in deltaDataFile[1]]
    # Values:
    delta        = []

    deltab       = []
    deltab1       = []
    deltab2       = []

    v            = []
    vb           = []
    Phi          = []
    Psi          = []
    Theta0       = []
    Theta1       = []
    ThetaP0      = []
    ThetaP1      = []
    Nu0          = []
    Nu1          = []
```

```
# Traces:
deltaTrace    = []

deltabTrace  = []
deltabTrace1  = []
deltabTrace2  = []

vTrace       = []
vbTrace      = []
PhiTrace     = []
PsiTrace     = []
Theta0Trace  = []
ThetaP0Trace = []
Theta1Trace  = []
ThetaP1Trace = []
Nu0Trace     = []
Nu1Trace     = []
# looping through all files
for i in range(0, 6):
    # Appending the values of delta into an array(i.e. the list of lists)
    delta.append([sublist[1] for sublist in deltaDataFile[i]])

    # Applying trick to show the negative values on y axes
    #deltab.append([np.arcsinh(sublist[1]) for sublist in deltabDataFile[i]])
    deltab.append([np.arcsinh(sublist[1]) for sublist in deltabDataFile[i]])
    deltab1.append([sublist[1] for sublist in deltabDataFile[i]])
    deltab2.append([np.arcsinh(sublist[1]) for sublist in deltabDataFile[i]])
    #for j in range(0, (len(deltab)-1)):
        #print(deltab[i][j])
    #    if (deltab[i][j] < 0):
    #        deltab.append(-np.log(-deltab[j]))
            #print(deltab[i][j])
    #    elif(deltab[i][j] == 0):
    #        deltab.append(0)
    #    elif (deltab[i][j] > 0):
    #        deltab.append(np.log(deltab[j]))


    v.append([sublist[1] for sublist in vDataFile[i]])
    vb.append([sublist[1] for sublist in vbDataFile[i]])
    Phi.append([sublist[1] for sublist in PhiDataFile[i]])
    Psi.append([sublist[1] for sublist in PsiDataFile[i]])
    Theta0.append([sublist[1] for sublist in Theta0DataFile[i]])
    Theta1.append([sublist[1] for sublist in Theta1DataFile[i]])
    ThetaP0.append([sublist[1] for sublist in ThetaP0DataFile[i]])
    ThetaP1.append([sublist[1] for sublist in ThetaP1DataFile[i]])
    Nu0.append([sublist[1] for sublist in Nu0DataFile[i]])
    Nu1.append([sublist[1] for sublist in Nu1DataFile[i]])

    # Feeding the plots with the values
    deltaTrace.append(
        go.Scatter(
            x = X7,
            y = delta[i],
            name = k[i],
            line = dict(
                color = (Colors[i]),#'rgb(300, 200, 100)'),
                width = 3)))

    deltabTrace.append(
        go.Scatter(
            x = X7,
```

```
            y = deltab[i],
            name = k[i],
            line = dict(
                color = (Colors[i]),#'rgb(300, 200, 100)'),
                width = 3)))
    deltabTrace1.append(
        go.Scatter(
            x = X7,
            y = deltab1[i],
            name = k[i],
            line = dict(
                color = (Colors[i]),#'rgb(300, 200, 100)'),
                width = 3)))
    deltabTrace2.append(
        go.Scatter(
            x = X7,
            y = deltab2[i],
            name = k[i],
            line = dict(
                color = (Colors[i]),#'rgb(300, 200, 100)'),
                width = 3)))

    vTrace.append(
        go.Scatter(
            x = X7,
            y = v[i],
            name = k[i],
            line = dict(
                color = (Colors[i]),#'rgb(300, 200, 100)'),
                width = 3)))
    vbTrace.append(
        go.Scatter(
            x = X7,
            y = vb[i],
            name = k[i],
            line = dict(
                color = (Colors[i]),#'rgb(300, 200, 100)'),
                width = 3)))
    PhiTrace.append(
        go.Scatter(
            x = X7,
            y = Phi[i],
            name = k[i],
            line = dict(
                color = (Colors[i]),#'rgb(300, 200, 100)'),
                width = 3)))
    PsiTrace.append(
        go.Scatter(
            x = X7,
            y = Psi[i],
            name = k[i],
            line = dict(
                color = (Colors[i]),#'rgb(300, 200, 100)'),
                width = 3)))
    Theta0Trace.append(
        go.Scatter(
            x = X7,
            y = Theta0[i],
            name = k[i],
            line = dict(
                color = (Colors[i]),#'rgb(300, 200, 100)'),
                width = 3)))
```

```
        Theta1Trace.append(
            go.Scatter(
                x = X7,
                y = Theta1[i],
                name = k[i],
                line = dict(
                    color = (Colors[i]),#'rgb(300, 200, 100)'),
                    width = 3)))
        ThetaP0Trace.append(
            go.Scatter(
                x = X7,
                y = ThetaP0[i],
                name = k[i],
                line = dict(
                    color = (Colors[i]),#'rgb(300, 200, 100)'),
                    width = 3)))
        ThetaP1Trace.append(
            go.Scatter(
                x = X7,
                y = ThetaP1[i],
                name = k[i],
                line = dict(
                    color = (Colors[i]),#'rgb(300, 200, 100)'),
                    width = 3)))
        Nu0Trace.append(
            go.Scatter(
                x = X7,
                y = Nu0[i],
                name = k[i],
                line = dict(
                    color = (Colors[i]),#'rgb(300, 200, 100)'),
                    width = 3)))
        Nu1Trace.append(
            go.Scatter(
                x = X7,
                y = Nu1[i],
                name = k[i],
                line = dict(
                    color = (Colors[i]),#'rgb(300, 200, 100)'),
                    width = 3)))

    # delta(x):
    deltaDataPlot    = deltaTrace
    #flat_list = [item for sublist in l for item in sublist]

    # delta_b(x):
    deltabDataPlot   = deltabTrace
    deltabDataPlot1   = deltabTrace1
    deltabDataPlot2   = deltabTrace2
    # v(x):
    vDataPlot        = vTrace
    # v_b(x):
    vbDataPlot       = vbTrace
    # Phi(x):
    PhiDataPlot      = PhiTrace
    # Psi(x):
    PsiDataPlot      = PsiTrace
    # Theta(x):
    Theta0DataPlot   = Theta0Trace
    Theta1DataPlot   = Theta1Trace
    # Theta^P(x):
    ThetaP0DataPlot = ThetaP0Trace
```

```
ThetaP1DataPlot = ThetaP1Trace
# N(x):
Nu0DataPlot     = Nu0Trace
Nu1DataPlot     = Nu1Trace

# Creating a position for each legend
legendPositionTop = [0.02, 0.96]
legendPositionBot = [0.02, 0.04]

deltaLayoutPlot  = dict(#title = '$\delta(x)$',
              xaxis = dict(title = '$x$', autorange = True,
                           mirror = True, ticks = 'outside',
                           showline = True),
              yaxis = dict(title = '$\delta(x)$', type = 'log', autorange =
                  ↪ True,
                           mirror = True, ticks = 'outside',
                           # To show values as number x 10^*
                           showexponent = 'all',
                           exponentformat = 'power',
                           #tickformat = 'power',
                           showline = True),
              # Tell where to put legends (i.e. labels of k values)
              legend = dict(
                  x = legendPositionTop[0],
                  y = legendPositionTop[1],
                  traceorder = "normal",
                  font = dict(size = 12, color = 'black'),
                  bgcolor = "White",
                  bordercolor = 'Black',
                  borderwidth = 2
              ),
              #height = 600,
              #width = 600,
              )

deltaLayoutPlot3  = dict(#title = '$\delta(x)$',
              xaxis = dict(title = '$x$', autorange = True,
                           mirror = True, ticks = 'outside',
                           showline = True),
              yaxis = dict(title = '$\delta(x)$', type = 'log', autorange =
                  ↪ True,
                           mirror = True, ticks = 'outside',
                           # To show values as number x 10^*
                           showexponent = 'all',
                           exponentformat = 'power',
                           #tickformat = 'power',
                           showline = True),
              # Tell where to put legends (i.e. labels of k values)
              legend = dict(
                  x = legendPositionTop[0],
                  y = legendPositionTop[1],
                  traceorder = "normal",
                  font = dict(size = 12, color = 'black'),
                  bgcolor = "White",
                  bordercolor = 'Black',
                  borderwidth = 2
              ),
              #height = 600,
              #width = 600,
              )

# Recover the values on the y axes for delta_b plot
```

```
#deltabAxesValues = []
#for i in range(0, 6):
    #for j in range(0, (len(deltab[i])-1)):
#     deltabAxesValues.append([np.sinh(deltab[i])])
#     print(deltabAxesValues[i])
deltabAxesValues = [-4, -2, 0, 2, 4, 6, 8, 10, 12]
deltabAxesText    = ['$-10^1$', '$-10^0$','$0$', '$10^0$', '$10^1$', '$10^2$',
    ↪ '$10^3$', '$10^4$', '$10^5$']
#k = -2
deltabLayoutPlot   = dict(#title = '$\delta(x)$',
              xaxis = dict(title = '$x$', autorange = True,
                           mirror = True, ticks = 'outside',
                           showline = True),
              yaxis = dict(title = '$\delta_b(x)$', type = 'linear', autorange
                  ↪  = True,
                           mirror = True, ticks = 'outside',
                           #tickvals = [(k+2) for k in range(-2, 13)],
                           #ticktext = ['$-10^1$', '$-10^0$','$0$', '$-10^1$',
                               ↪  '$-10^1$', '$-10^1$'],
                           tickvals = deltabAxesValues,
                           ticktext = deltabAxesText,
                           tickfont = dict(
                                family  = 'Courier␣New,␣monospace',
                                size    = 12,
                                color   = 'black'
                            ),
                           # To show values as number x 10^*
                           #showexponent = 'all',
                           #exponentformat = 'power',
                           showline = True),
              legend = dict(
                  x = legendPositionTop[0],
                  y = legendPositionTop[1],
                  traceorder = "normal",
                  font = dict(size = 12, color = 'black'),
                  bgcolor = "White",
                  bordercolor = 'Black',
                  borderwidth = 2
              ),
              )
deltabLayoutPlot1   = dict(#title = '$\delta(x)$',
              xaxis = dict(title = '$x$', autorange = True,
                           mirror = True, ticks = 'outside',
                           showline = True),
              yaxis = dict(title = '$\delta_b(x)$', type = 'log', autorange =
                  ↪ True,
                           mirror = True, ticks = 'outside',
                           # To show values as number x 10^*
                           showexponent = 'all',
                           exponentformat = 'power',
                           showline = True),
              legend = dict(
                  x = legendPositionTop[0],
                  y = legendPositionTop[1],
                  traceorder = "normal",
                  font = dict(size = 12, color = 'black'),
                  bgcolor = "White",
                  bordercolor = 'Black',
                  borderwidth = 2
              ),
              )
deltabLayoutPlot2   = dict(#title = '$\delta(x)$',
```

```
                    xaxis = dict(title = '$x$', autorange = True,
                                 mirror = True, ticks = 'outside',
                                 showline = True),
                    yaxis = dict(title = '$\delta_b(x)$', type = 'linear', autorange
                        ↪  = True,
                                 mirror = True, ticks = 'outside',
                                 # To show values as number x 10^*
                                 showexponent = 'all',
                                 exponentformat = 'power',
                                 showline = True),
                    legend = dict(
                        x = legendPositionTop[0],
                        y = legendPositionTop[1],
                        traceorder = "normal",
                        font = dict(size = 12, color = 'black'),
                        bgcolor = "White",
                        bordercolor = 'Black',
                        borderwidth = 2
                    ),
                    )
    vLayoutPlot  = dict(#title = '$\delta(x)$',
                    xaxis = dict(title = '$x$', autorange = True,
                                 mirror = True, ticks = 'outside',
                                 showline = True),
                    yaxis = dict(title = '$v(x)$', type = 'linear', autorange = True
                        ↪ ,
                                 mirror = True, ticks = 'outside',
                                 # To show values as number x 10^*
                                 showexponent = 'all',
                                 exponentformat = 'power',
                                 showline = True),
                    legend = dict(
                        x = legendPositionTop[0],
                        y = legendPositionTop[1],
                        traceorder = "normal",
                        font = dict(size = 12, color = 'black'),
                        bgcolor = "White",
                        bordercolor = 'Black',
                        borderwidth = 2
                    ),
                    )
    vbLayoutPlot  = dict(#title = '$\delta(x)$',
                    xaxis = dict(title = '$x$', autorange = True,
                                 mirror = True, ticks = 'outside',
                                 showline = True),
                    yaxis = dict(title = '$v_b(x)$', type = 'linear', autorange =
                        ↪ True,
                                 mirror = True, ticks = 'outside',
                                 # To show values as number x 10^*
                                 showexponent = 'all',
                                 exponentformat = 'power',
                                 showline = True),
                    legend = dict(
                        x = legendPositionTop[0],
                        y = legendPositionTop[1],
                        traceorder = "normal",
                        font = dict(size = 12, color = 'black'),
                        bgcolor = "White",
                        bordercolor = 'Black',
                        borderwidth = 2
                    ),
                    )
```

```
PhiLayoutPlot  = dict(#title = '$\delta(x)$',
                xaxis = dict(title = '$x$', autorange = True,
                             mirror = True, ticks = 'outside',
                             showline = True),
                yaxis = dict(title = '$\Phi(x)$', type = 'linear', autorange =
                    ↪ True,
                             mirror = True, ticks = 'outside',
                             showline = True),
                legend = dict(
                    x = legendPositionBot[0],
                    y = legendPositionBot[1],
                    traceorder = "normal",
                    font = dict(size = 12, color = 'black'),
                    bgcolor = "White",
                    bordercolor = 'Black',
                    borderwidth = 2
                ),
                )
PsiLayoutPlot  = dict(#title = '$\delta(x)$',
                xaxis = dict(title = '$x$', autorange = True,
                             mirror = True, ticks = 'outside',
                             showline = True),
                yaxis = dict(title = '$\Psi(x)$', type = 'linear', autorange =
                    ↪ True,
                             mirror = True, ticks = 'outside',
                             # To show values as number x 10^*
                             showexponent = 'all',
                             exponentformat = 'power',
                             showline = True),
                legend = dict(
                    x = legendPositionTop[0],
                    y = legendPositionTop[1],
                    traceorder = "normal",
                    font = dict(size = 12, color = 'black'),
                    bgcolor = "White",
                    bordercolor = 'Black',
                    borderwidth = 2
                ),
                )


Theta0LayoutPlot  = dict(#title = '$\delta(x)$',
                xaxis = dict(title = '$x$', autorange = True,
                             mirror = True, ticks = 'outside',
                             showline = True),
                yaxis = dict(title = '$\Theta_0(x)$', type = 'linear', autorange
                    ↪  = True,
                             mirror = True, ticks = 'outside',
                             # To show values as number x 10^*
                             showexponent = 'all',
                             exponentformat = 'power',
                             showline = True),
                legend = dict(
                    x = legendPositionBot[0],
                    y = legendPositionBot[1],
                    traceorder = "normal",
                    font = dict(size = 12, color = 'black'),
                    bgcolor = "White",
                    bordercolor = 'Black',
                    borderwidth = 2
                ),
```

```
                    )
    Theta1LayoutPlot   = dict(#title = '$\delta(x)$',
                    xaxis = dict(title = '$x$', autorange = True,
                                 mirror = True, ticks = 'outside',
                                 showline = True),
                    yaxis = dict(title = '$\Theta_1(x)$', type = 'linear', autorange
                        ↪  = True,
                                 mirror = True, ticks = 'outside',
                                 # To show values as number x 10^*
                                 showexponent = 'all',
                                 exponentformat = 'power',
                                 showline = True),
                    legend = dict(
                        x = legendPositionTop[0],
                        y = legendPositionTop[1],
                        traceorder = "normal",
                        font = dict(size = 12, color = 'black'),
                        bgcolor = "White",
                        bordercolor = 'Black',
                        borderwidth = 2
                    ),
                    )
    ThetaP0LayoutPlot   = dict(#title = '$\delta(x)$',
                    xaxis = dict(title = '$x$', autorange = True,
                                 mirror = True, ticks = 'outside',
                                 showline = True),
                    yaxis = dict(title = '$\Theta^P_0(x)$', type = 'linear',
                        ↪ autorange = True,
                                 mirror = True, ticks = 'outside',
                                 # To show values as number x 10^*
                                 showexponent = 'all',
                                 exponentformat = 'power',
                                 showline = True),
                    legend = dict(
                        x = legendPositionTop[0],
                        y = legendPositionTop[1],
                        traceorder = "normal",
                        font = dict(size = 12, color = 'black'),
                        bgcolor = "White",
                        bordercolor = 'Black',
                        borderwidth = 2
                    ),
                    )
    ThetaP1LayoutPlot   = dict(#title = '$\delta(x)$',
                    xaxis = dict(title = '$x$', autorange = True,
                                 mirror = True, ticks = 'outside',
                                 showline = True),
                    yaxis = dict(title = '$\Theta^P_1(x)$', type = 'linear',
                        ↪ autorange = True,
                                 mirror = True, ticks = 'outside',
                                 # To show values as number x 10^*
                                 showexponent = 'all',
                                 exponentformat = 'power',
                                 showline = True),
                    legend = dict(
                        x = legendPositionTop[0],
                        y = legendPositionTop[1],
                        traceorder = "normal",
                        font = dict(size = 12, color = 'black'),
                        bgcolor = "White",
                        bordercolor = 'Black',
                        borderwidth = 2
```

```
                 ),
                 )
    Nu0LayoutPlot  = dict(#title = '$\delta(x)$',
                 xaxis = dict(title = '$x$', autorange = True,
                              mirror = True, ticks = 'outside',
                              showline = True),
                 yaxis = dict(title = '$\mathcal{N}_0(x)$', type = 'linear',
                     ↪ autorange = True,
                              mirror = True, ticks = 'outside',
                              # To show values as number x 10^*
                              showexponent = 'all',
                              exponentformat = 'power',
                              showline = True),
                 legend = dict(
                     x = legendPositionTop[0],
                     y = legendPositionTop[1],
                     traceorder = "normal",
                     font = dict(size = 12, color = 'black'),
                     bgcolor = "White",
                     bordercolor = 'Black',
                     borderwidth = 2
                 ),
                 )
    Nu1LayoutPlot  = dict(#title = '$\delta(x)$',
                 xaxis = dict(title = '$x$', autorange = True,
                              mirror = True, ticks = 'outside',
                              showline = True),
                 yaxis = dict(title = '$\mathcal{N}_1(x)$', type = 'linear',
                     ↪ autorange = True,
                              mirror = True, ticks = 'outside',
                              # To show values as number x 10^*
                              showexponent = 'all',
                              exponentformat = 'power',
                              showline = True),
                 legend = dict(
                     x = legendPositionTop[0],
                     y = legendPositionTop[1],
                     traceorder = "normal",
                     font = dict(size = 12, color = 'black'),
                     bgcolor = "White",
                     bordercolor = 'Black',
                     borderwidth = 2
                 ),
                 )

    deltaFigure    = dict(data = deltaDataPlot, layout = deltaLayoutPlot)

    deltabFigure   = dict(data = deltabDataPlot, layout = deltabLayoutPlot)
    deltabFigure1  = dict(data = deltabDataPlot1, layout = deltabLayoutPlot1)
    deltabFigure2  = dict(data = deltabDataPlot2, layout = deltabLayoutPlot2)

    vFigure        = dict(data = vDataPlot, layout = vLayoutPlot)
    vbFigure       = dict(data = vbDataPlot, layout = vbLayoutPlot)

    PhiFigure      = dict(data = PhiDataPlot, layout = PhiLayoutPlot)
    PsiFigure      = dict(data = PsiDataPlot, layout = PsiLayoutPlot)

    Theta0Figure   = dict(data = Theta0DataPlot, layout = Theta0LayoutPlot)
    Theta1Figure   = dict(data = Theta1DataPlot, layout = Theta1LayoutPlot)
    ThetaP0Figure  = dict(data = ThetaP0DataPlot, layout = ThetaP0LayoutPlot)
    ThetaP1Figure      = dict(data = ThetaP1DataPlot, layout = ThetaP1LayoutPlot)
    #PhiFigure      = dict(data = PhiDataPlot, layout = PhiLayoutPlot)
```

```
Nu0Figure      = dict(data = Nu0DataPlot, layout = Nu0LayoutPlot)
Nu1Figure      = dict(data = Nu1DataPlot, layout = Nu1LayoutPlot)

# Plotting everything
plt.iplot(deltaFigure,   filename = 'delta_x')

plt.iplot(deltabFigure,  filename = 'deltab_x')
#plt.iplot(deltabFigure1,  filename = 'deltab1_x')
#plt.iplot(deltabFigure2,  filename = 'deltab2_x')

plt.iplot(vFigure,       filename = 'v_x')
plt.iplot(vbFigure,      filename = 'vb_x')

plt.iplot(PhiFigure,     filename = 'Phi_x')
plt.iplot(PsiFigure,     filename = 'Psi_x')

plt.iplot(Theta0Figure,  filename = 'Theta0_x')
plt.iplot(Theta1Figure,  filename = 'Theta1_x')
plt.iplot(ThetaP0Figure, filename = 'ThetaP0_x')
plt.iplot(ThetaP1Figure, filename = 'ThetaP1_x')
plt.iplot(Nu0Figure,     filename = 'Nu0_x')
plt.iplot(Nu1Figure,     filename = 'Nu1_x')

# Saving plots (to the plots directory)
pio.write_image(deltaFigure,   outputDir + '/' + 'delta_x.pdf')

pio.write_image(deltabFigure,  outputDir + '/' + 'deltab_x.pdf')
#pio.write_image(deltabFigure1,  outputDir + '/' + 'deltab1_x.pdf')
#pio.write_image(deltabFigure2,  outputDir + '/' + 'deltab2_x.pdf')

pio.write_image(vFigure,       outputDir + '/' + 'v_x.pdf')
pio.write_image(vbFigure,      outputDir + '/' + 'vb_x.pdf')
pio.write_image(PhiFigure,     outputDir + '/' + 'Phi_x.pdf')
pio.write_image(PsiFigure,     outputDir + '/' + 'Psi_x.pdf')
pio.write_image(Theta0Figure,  outputDir + '/' + 'Theta0_x.pdf')
pio.write_image(Theta1Figure,  outputDir + '/' + 'Theta1_x.pdf')
pio.write_image(ThetaP0Figure, outputDir + '/' + 'ThetaP0_x.pdf')
pio.write_image(ThetaP1Figure, outputDir + '/' + 'ThetaP1_x.pdf')
pio.write_image(Nu0Figure,     outputDir + '/' + 'Nu0_x.pdf')
pio.write_image(Nu1Figure,     outputDir + '/' + 'Nu1_x.pdf')

#
    ↪ #########################################################################################
    ↪
# Milestone 4
#
    ↪ #########################################################################################
    ↪

# Source funciton
sourceDataFile  = [
            np.loadtxt(inputDir + '/' + "S_0001_x.dat"),
            np.loadtxt(inputDir + '/' + "S_0498_x.dat"),
            np.loadtxt(inputDir + '/' + "S_1117_x.dat"),
            np.loadtxt(inputDir + '/' + "S_1999_x.dat"),
            np.loadtxt(inputDir + '/' + "S_3000_x.dat"),
            np.loadtxt(inputDir + '/' + "S_5000_x.dat")]

sourceEDataFile  = [
            np.loadtxt(inputDir + '/' + "SE_0001_x.dat"),
            np.loadtxt(inputDir + '/' + "SE_0498_x.dat"),
            np.loadtxt(inputDir + '/' + "SE_1117_x.dat"),
```

```
            np.loadtxt(inputDir + '/' + "SE_1999_x.dat"),
            np.loadtxt(inputDir + '/' + "SE_3000_x.dat"),
            np.loadtxt(inputDir + '/' + "SE_5000_x.dat")]

# Transfer function
transferDataFile  = [
            np.loadtxt(inputDir + '/' + "ThetaT_0002_k.dat"),
            np.loadtxt(inputDir + '/' + "ThetaT_0040_k.dat"),
            np.loadtxt(inputDir + '/' + "ThetaT_0100_k.dat"),
            np.loadtxt(inputDir + '/' + "ThetaT_0250_k.dat"),
            np.loadtxt(inputDir + '/' + "ThetaT_0550_k.dat"),
            np.loadtxt(inputDir + '/' + "ThetaT_1200_k.dat")]
transferEDataFile  = [
            np.loadtxt(inputDir + '/' + "ThetaE_0002_k.dat"),
            np.loadtxt(inputDir + '/' + "ThetaE_0040_k.dat"),
            np.loadtxt(inputDir + '/' + "ThetaE_0100_k.dat"),
            np.loadtxt(inputDir + '/' + "ThetaE_0250_k.dat"),
            np.loadtxt(inputDir + '/' + "ThetaE_0550_k.dat"),
            np.loadtxt(inputDir + '/' + "ThetaE_1200_k.dat")]
# Intermediate quantities
JlDataFile  = [
            np.loadtxt(inputDir + '/' + "j_0100_0001_x.dat"),
            np.loadtxt(inputDir + '/' + "j_0100_0498_x.dat"),
            np.loadtxt(inputDir + '/' + "j_0100_1117_x.dat"),
            np.loadtxt(inputDir + '/' + "j_0100_1999_x.dat"),
            np.loadtxt(inputDir + '/' + "j_0100_3000_x.dat"),
            np.loadtxt(inputDir + '/' + "j_0100_5000_x.dat")]
integrand1DataFile  = [
            np.loadtxt(inputDir + '/' + "int_0100_0001_x.dat"),
            np.loadtxt(inputDir + '/' + "int_0100_0498_x.dat"),
            np.loadtxt(inputDir + '/' + "int_0100_1117_x.dat"),
            np.loadtxt(inputDir + '/' + "int_0100_1999_x.dat"),
            np.loadtxt(inputDir + '/' + "int_0100_3000_x.dat"),
            np.loadtxt(inputDir + '/' + "int_0100_5000_x.dat")]
integrand11DataFile = np.loadtxt(inputDir + '/' + "int_0100_3000_x.dat")
# Theta^2/k etc.
integrandTDataFile  = [
            np.loadtxt(inputDir + '/' + "intT_0002_k.dat"),
            np.loadtxt(inputDir + '/' + "intT_0040_k.dat"),
            np.loadtxt(inputDir + '/' + "intT_0100_k.dat"),
            np.loadtxt(inputDir + '/' + "intT_0250_k.dat"),
            np.loadtxt(inputDir + '/' + "intT_0550_k.dat"),
            np.loadtxt(inputDir + '/' + "intT_1200_k.dat")]
integrandEDataFile  = [
            np.loadtxt(inputDir + '/' + "intE_0002_k.dat"),
            np.loadtxt(inputDir + '/' + "intE_0040_k.dat"),
            np.loadtxt(inputDir + '/' + "intE_0100_k.dat"),
            np.loadtxt(inputDir + '/' + "intE_0250_k.dat"),
            np.loadtxt(inputDir + '/' + "intE_0550_k.dat"),
            np.loadtxt(inputDir + '/' + "intE_1200_k.dat")]
integrandTEDataFile  = [
            np.loadtxt(inputDir + '/' + "intTE_0002_k.dat"),
            np.loadtxt(inputDir + '/' + "intTE_0040_k.dat"),
            np.loadtxt(inputDir + '/' + "intTE_0100_k.dat"),
            np.loadtxt(inputDir + '/' + "intTE_0250_k.dat"),
            np.loadtxt(inputDir + '/' + "intTE_0550_k.dat"),
            np.loadtxt(inputDir + '/' + "intTE_1200_k.dat")]
# C_l
ClTTDataFile  = np.loadtxt(inputDir + '/' + "CTT_l.dat")
ClTEDataFile  = np.loadtxt(inputDir + '/' + "CTE_l.dat")
ClEEDataFile  = np.loadtxt(inputDir + '/' + "CEE_l.dat")
```

```
l = ['$l␣=␣2$', '$l␣=␣40$', '$l␣=␣100$', '$l␣=␣250$', '$l␣=␣550$', '$l␣=␣1200$
    ↪ ']
# Grid:
X8 = [sublist[0] for sublist in sourceDataFile[1]]
K8 = [sublist[0] for sublist in transferDataFile[1]]
L8 = ClTTDataFile[:,0]

# Values:
source        = []
sourceE       = []
transfer      = []
transferE     = []
# Intermediate quantities
Jl = []
integrand1 = []
integrand11 = integrand11DataFile[:,1]
integrand2 = []

integrandT = []
integrandE = []
integrandTE = []
CTT = ClTTDataFile[:,1]
print(CTT)
CEE = ClEEDataFile[:,1]
CTE = ClTEDataFile[:,1]
# Traces:
sourceTrace   = []
sourceETrace  = []
transferTrace = []
transferETrace   = []
# Intermediate quantities
JlTrace = []
integrand1Trace   = []
integrand2Trace   = []

integrandTTrace   = []
integrandETrace   = []
integrandTETrace   = []
CTTTrace = []
CEETrace = []
CTETrace = []

# looping through all files
for i in range(0, 6):
    # Appending the values of delta into an array(i.e. the list of lists)
    # Source function:
    source.append([sublist[1] for sublist in sourceDataFile[i]])
    sourceE.append([sublist[1] for sublist in sourceEDataFile[i]])
    # Transfer function:
    transfer.append([sublist[1] for sublist in transferDataFile[i]])
    transferE.append([sublist[1] for sublist in transferEDataFile[i]])
    # Intermediate quantities
    Jl.append([sublist[1] for sublist in JlDataFile[i]])
    integrand1.append([sublist[1] for sublist in integrand1DataFile[i]])
    # Theta^2/k etc.:
    integrandT.append([sublist[1] for sublist in integrandTDataFile[i]])
    integrandE.append([sublist[1] for sublist in integrandEDataFile[i]])
    integrandTE.append([sublist[1] for sublist in integrandTEDataFile[i]])

    # Feeding the plots with the values
    sourceTrace.append(
```

```
        go.Scatter(
            x = X8,
            y = source[i],
            name = k[i],
            line = dict(
                color = (Colors[i]),#'rgb(300, 200, 100)'),
                width = 3)))

    sourceETrace.append(
        go.Scatter(
            x = X8,
            y = sourceE[i],
            name = k[i],
            line = dict(
                color = (Colors[i]),#'rgb(300, 200, 100)'),
                width = 3)))
    transferTrace.append(
        go.Scatter(
            x = K8,
            y = transfer[i],
            name = l[i],
            line = dict(
                color = (Colors[i]),#'rgb(300, 200, 100)'),
                width = 3)))
    transferETrace.append(
        go.Scatter(
            x = K8,
            y = transferE[i],
            name = l[i],
            line = dict(
                color = (Colors[i]),#'rgb(300, 200, 100)'),
                width = 3)))
    # Intermediate quantities
    integrand1Trace.append(
        go.Scatter(
            x = X8,
            y = integrand1[i],
            name = k[i],
            line = dict(
                color = (Colors[i]),#'rgb(300, 200, 100)'),
                width = 3)))
    J1Trace.append(
        go.Scatter(
            x = X8,
            y = J1[i],
            name = k[i],
            line = dict(
                color = (Colors[i]),#'rgb(300, 200, 100)'),
                width = 3)))


    integrandTTrace.append(
        go.Scatter(
            x = K8,
            y = integrandT[i],
            name = l[i],
            line = dict(
                color = (Colors[i]),#'rgb(300, 200, 100)'),
                width = 3)))
    integrandETrace.append(
        go.Scatter(
            x = K8,
```

```
                y = integrandE[i],
                name = l[i],
                line = dict(
                    color = (Colors[i]),#'rgb(300, 200, 100)'),
                    width = 3)))
        integrandTETrace.append(
            go.Scatter(
                x = K8,
                y = integrandTE[i],
                name = l[i],
                line = dict(
                    color = (Colors[i]),#'rgb(300, 200, 100)'),
                    width = 3)))

    integrand11Trace =  go.Scatter(
                x = X8,
                y = integrand1[5],
                name = k[5],
                line = dict(
                    color = (Colors[1]),#'rgb(300, 200, 100)'),
                    width = 3))
    # Power spectrum trace
    CTTTrace = go.Scatter(
        x = L8,
        y = CTT,
        name = '$C_l$',
        line = dict(
            color = ('red'),#'rgb(100, 20, 50)'),
            width = 3))
    CEETrace = go.Scatter(
        x = L8,
        y = CEE,
        name = '$C_l$',
        line = dict(
            color = ('blue'),#'rgb(100, 20, 50)'),
            width = 3))
    CTETrace = go.Scatter(
        x = L8,
        y = CTE,
        name = '$C_l$',
        line = dict(
            color = ('blue'),#'rgb(100, 20, 50)'),
            width = 3))
    # S(k, x):
    sourceDataPlot  = sourceTrace
    sourceEDataPlot = sourceETrace
    transferDataPlot   = transferTrace
    transferEDataPlot = transferETrace

    # Intermediate quantities
    JlDataPlot = JlTrace
    integrand1DataPlot  = integrand1Trace
    integrand11DataPlot = [integrand11Trace]

    integrandTDataPlot  = integrandTTrace
    integrandEDataPlot  = integrandETrace
    integrandTEDataPlot = integrandTETrace

    CTTDataPlot = [CTTTrace]
    CEEDataPlot = [CEETrace]
    CTEDataPlot = [CTETrace]
```

```
legendPositionTopRight = [0.85, 0.96]
legendPositionBotRight = [0.85, 0.04]

sourceLayoutPlot  = dict(#title = '$\delta(x)$',
              xaxis = dict(title = '$x$', autorange = True,
                          mirror = True, ticks = 'outside',
                          showline = True),
              yaxis = dict(title = '$S(k,␣x)$', type = 'linear', autorange =
                  ↪ True,
                          mirror = True, ticks = 'outside',
                          # To show values as number x 10^*
                          showexponent = 'all',
                          exponentformat = 'power',
                          #tickformat = 'power',
                          showline = True),
              # Tell where to put legends (i.e. labels of k values)
              legend = dict(
                  x = legendPositionTop[0],
                  y = legendPositionTop[1],
                  traceorder = "normal",
                  font = dict(size = 12, color = 'black'),
                  bgcolor = "White",
                  bordercolor = 'Black',
                  borderwidth = 2
              ),
              #height = 600,
              #width = 600,
              )
sourceELayoutPlot  = dict(#title = '$\delta(x)$',
              xaxis = dict(title = '$x$', autorange = True,
                          mirror = True, ticks = 'outside',
                          showline = True),
              yaxis = dict(title = '$S^E(k,␣x)$', type = 'linear', autorange =
                  ↪  True,
                          mirror = True, ticks = 'outside',
                          # To show values as number x 10^*
                          showexponent = 'all',
                          exponentformat = 'power',
                          #tickformat = 'power',
                          showline = True),
              # Tell where to put legends (i.e. labels of k values)
              legend = dict(
                  x = legendPositionTop[0],
                  y = legendPositionTop[1],
                  traceorder = "normal",
                  font = dict(size = 12, color = 'black'),
                  bgcolor = "White",
                  bordercolor = 'Black',
                  borderwidth = 2
              ),
              #height = 600,
              #width = 600,
              )
transferLayoutPlot  = dict(#title = '$\delta(x)$',
              xaxis = dict(title = '$k$', autorange = True,#autorange = True,
                          mirror = True, ticks = 'outside',
                          showline = True),
              yaxis = dict(title = '$\Theta_l(k)$', type = 'linear', range =
                  ↪ [-0.01, 0.03],
                          mirror = True, ticks = 'outside',
                          # To show values as number x 10^*
                          showexponent = 'all',
```

```
                            exponentformat = 'power',
                            #tickformat = 'power',
                            showline = True),
                # Tell where to put legends (i.e. labels of k values)
                legend = dict(
                    x = legendPositionTopRight[0],
                    y = legendPositionTopRight[1],
                    traceorder = "normal",
                    font = dict(size = 12, color = 'black'),
                    bgcolor = "White",
                    bordercolor = 'Black',
                    borderwidth = 2
                ),
                #height = 600,
                #width = 600,
                )
transferELayoutPlot  = dict(#title = '$\delta(x)$',
                xaxis = dict(title = '$k$', autorange = True,
                            mirror = True, ticks = 'outside',
                            showline = True),
                yaxis = dict(title = '$\Theta^E_l(k)$', type = 'linear',
                    ↪ autorange = True,
                            mirror = True, ticks = 'outside',
                            # To show values as number x 10^*
                            showexponent = 'all',
                            exponentformat = 'power',
                            #tickformat = 'power',
                            showline = True),
                # Tell where to put legends (i.e. labels of k values)
                legend = dict(
                    x = legendPositionBotRight[0],
                    y = legendPositionBotRight[1],
                    traceorder = "normal",
                    font = dict(size = 12, color = 'black'),
                    bgcolor = "White",
                    bordercolor = 'Black',
                    borderwidth = 2
                ),
                #height = 600,
                #width = 600,
                )
# Intermediate quantities
JlLayoutPlot  = dict(#title = '$\delta(x)$',
                xaxis = dict(title = '$x$', autorange = True,
                            mirror = True, ticks = 'outside',
                            showline = True),
                yaxis = dict(title = '$J_l\\left[k\\left(\eta_0-\eta\\right)\\
                    ↪ right]$',
                            type = 'linear', autorange = True,
                            mirror = True, ticks = 'outside',
                            # To show values as number x 10^*
                            showexponent = 'all',
                            exponentformat = 'power',
                            #tickformat = 'power',
                            showline = True),
                # Tell where to put legends (i.e. labels of k values)
                legend = dict(
                    x = legendPositionTop[0],
                    y = legendPositionTop[1],
                    traceorder = "normal",
                    font = dict(size = 12, color = 'black'),
                    bgcolor = "White",
```

```
                    bordercolor = 'Black',
                    borderwidth = 2
                ),
                #height = 600,
                #width = 600,
                )
integrand1LayoutPlot  = dict(#title = '$\delta(x)$',
                xaxis = dict(title = '$x$', range = [-10, 0],
                             mirror = True, ticks = 'outside',
                             showline = True),
                yaxis = dict(title = '$S(k,␣x)\cdot␣j_l\\left[k\\left(\eta_0-\
                    ↪ eta\\right)\\right]$',
                             type = 'linear', range = [-3 *10**(-3), 3*10**(-3)
                                 ↪ ],
                             mirror = True, ticks = 'outside',
                             # To show values as number x 10^*
                             showexponent = 'all',
                             exponentformat = 'power',
                             #tickformat = 'power',
                             showline = True),
                # Tell where to put legends (i.e. labels of k values)
                legend = dict(
                    x = legendPositionTop[0],
                    y = legendPositionTop[1],
                    traceorder = "normal",
                    font = dict(size = 12, color = 'black'),
                    bgcolor = "White",
                    bordercolor = 'Black',
                    borderwidth = 2
                ),
                #height = 600,
                #width = 600,
                )

integrand11LayoutPlot  = dict(#title = '$\delta(x)$',
                xaxis = dict(title = '$x$', autorange = True,
                             mirror = True, ticks = 'outside',
                             showline = True),
                yaxis = dict(title = '$S(k,␣x)j_l\\left[k\\left(\eta_0-\eta\\
                    ↪ right\\right]$',
                             type = 'linear', autorange = True,
                             mirror = True, ticks = 'outside',
                             # To show values as number x 10^*
                             showexponent = 'all',
                             exponentformat = 'power',
                             #tickformat = 'power',
                             showline = True),
                # Tell where to put legends (i.e. labels of k values)
                legend = dict(
                    x = legendPositionTop[0],
                    y = legendPositionTop[1],
                    traceorder = "normal",
                    font = dict(size = 12, color = 'black'),
                    bgcolor = "White",
                    bordercolor = 'Black',
                    borderwidth = 2
                ),
                #height = 600,
                #width = 600,
                )

integrandTLayoutPlot  = dict(#title = '$\delta(x)$',
```

```
                 xaxis = dict(title = '$k␣\\left[H_0␣/␣c\\right]$', type = '
                     ↪ linear',
                            autorange = True,
                            mirror = True, ticks = 'outside',
                            showline = True),
                 yaxis = dict(title = '$\\left(\Theta_l^T(k)\\right)^2/k$', type
                     ↪ = 'linear',
                            range = [0, 0.1],
                            mirror = True, ticks = 'outside',
                            # To show values as number x 10^*
                            showexponent = 'all',
                            exponentformat = 'power',
                            #tickformat = 'power',
                            showline = True),
                 # Tell where to put legends (i.e. labels of k values)
                 legend = dict(
                     x = legendPositionTop[0],
                     y = legendPositionTop[1],
                     traceorder = "normal",
                     font = dict(size = 12, color = 'black'),
                     bgcolor = "White",
                     bordercolor = 'Black',
                     borderwidth = 2
                 ),
                 #height = 600,
                 #width = 600,
                 )
integrandELayoutPlot  = dict(#title = '$\delta(x)$',
                 xaxis = dict(title = '$k␣\\left[H_0␣/␣c\\right]$', type = '
                     ↪ linear', autorange = True,
                            mirror = True, ticks = 'outside',
                            showline = True),
                 yaxis = dict(title = '$\\left(\Theta_l^E(k)\\right)^2/k$', type
                     ↪ = 'linear',
                            autorange = True,
                            mirror = True, ticks = 'outside',
                            # To show values as number x 10^*
                            showexponent = 'all',
                            exponentformat = 'power',
                            #tickformat = 'power',
                            showline = True),
                 # Tell where to put legends (i.e. labels of k values)
                 legend = dict(
                     x = legendPositionTop[0],
                     y = legendPositionTop[1],
                     traceorder = "normal",
                     font = dict(size = 12, color = 'black'),
                     bgcolor = "White",
                     bordercolor = 'Black',
                     borderwidth = 2
                 ),
                 #height = 600,
                 #width = 600,
                 )
integrandTELayoutPlot  = dict(#title = '$\delta(x)$',
                 xaxis = dict(title = '$k␣\\left[H_0␣/␣c\\right]$', type = '
                     ↪ linear', autorange = True,
                            mirror = True, ticks = 'outside',
                            showline = True),
                 yaxis = dict(title = '$\Theta_l^E(k)\cdot\Theta^T(k)/k$', type =
                     ↪  'linear',
                            autorange = True,
```

```
                          mirror = True, ticks = 'outside',
                          # To show values as number x 10^*
                          showexponent = 'all',
                          exponentformat = 'power',
                          #tickformat = 'power',
                          showline = True),
             # Tell where to put legends (i.e. labels of k values)
             legend = dict(
                 x = legendPositionTop[0],
                 y = legendPositionTop[1],
                 traceorder = "normal",
                 font = dict(size = 12, color = 'black'),
                 bgcolor = "White",
                 bordercolor = 'Black',
                 borderwidth = 2
             ),
             #height = 600,
             #width = 600,
             )

CTTLayoutPlot  = dict(#title = '$\delta(x)$',
             xaxis = dict(title = '$l$', autorange = True,
                          mirror = True, ticks = 'outside',
                          showline = True),
             yaxis = dict(title = '$C_l␣l␣(l+1)$', type = 'linear',
                          autorange = True,
                          mirror = True, ticks = 'outside',
                          # To show values as number x 10^*
                          showexponent = 'all',
                          exponentformat = 'power',
                          #tickformat = 'power',
                          showline = True),
             # Tell where to put legends (i.e. labels of k values)
             legend = dict(
                 x = legendPositionTop[0],
                 y = legendPositionTop[1],
                 traceorder = "normal",
                 font = dict(size = 12, color = 'black'),
                 bgcolor = "White",
                 bordercolor = 'Black',
                 borderwidth = 2
             ),
             #height = 600,
             #width = 600,
             )
CEELayoutPlot  = dict(#title = '$\delta(x)$',
             xaxis = dict(title = '$l$', autorange = True,
                          mirror = True, ticks = 'outside',
                          showline = True),
             yaxis = dict(title = '$C_l␣l␣(l+1)$', type = 'linear',
                          autorange = True,
                          mirror = True, ticks = 'outside',
                          # To show values as number x 10^*
                          showexponent = 'all',
                          exponentformat = 'power',
                          #tickformat = 'power',
                          showline = True),
             # Tell where to put legends (i.e. labels of k values)
             legend = dict(
                 x = legendPositionTop[0],
                 y = legendPositionTop[1],
                 traceorder = "normal",
```

```
                  font = dict(size = 12, color = 'black'),
                  bgcolor = "White",
                  bordercolor = 'Black',
                  borderwidth = 2
              ),
              #height = 600,
              #width = 600,
              )
CTELayoutPlot  = dict(#title = '$\delta(x)$',
              xaxis = dict(title = '$l$', autorange = True,
                           mirror = True, ticks = 'outside',
                           showline = True),
              yaxis = dict(title = '$C_l␣l␣(l+1)$', type = 'linear',
                           autorange = True,
                           mirror = True, ticks = 'outside',
                           # To show values as number x 10^*
                           showexponent = 'all',
                           exponentformat = 'power',
                           #tickformat = 'power',
                           showline = True),
              # Tell where to put legends (i.e. labels of k values)
              legend = dict(
                  x = legendPositionTop[0],
                  y = legendPositionTop[1],
                  traceorder = "normal",
                  font = dict(size = 12, color = 'black'),
                  bgcolor = "White",
                  bordercolor = 'Black',
                  borderwidth = 2
              ),
              #height = 600,
              #width = 600,
              )

sourceFigure   = dict(data = sourceDataPlot, layout = sourceLayoutPlot)
sourceEFigure   = dict(data = sourceEDataPlot, layout = sourceELayoutPlot)
transferFigure   = dict(data = transferDataPlot, layout = transferLayoutPlot)
transferEFigure   = dict(data = transferEDataPlot, layout =
    ↪ transferELayoutPlot)

JlFigure   = dict(data = JlDataPlot, layout = JlLayoutPlot)
integrand1Figure   = dict(data = integrand1DataPlot, layout =
    ↪ integrand1LayoutPlot)
integrand11Figure   = dict(data = integrand11DataPlot, layout =
    ↪ integrand11LayoutPlot)

integrandTFigure   = dict(data = integrandTDataPlot, layout =
    ↪ integrandTLayoutPlot)
integrandEFigure   = dict(data = integrandEDataPlot, layout =
    ↪ integrandELayoutPlot)
integrandTEFigure   = dict(data = integrandTEDataPlot, layout =
    ↪ integrandTELayoutPlot)

CTTFigure   = dict(data = CTTDataPlot, layout = CTTLayoutPlot)
CEEFigure   = dict(data = CEEDataPlot, layout = CEELayoutPlot)
CTEFigure   = dict(data = CTEDataPlot, layout = CTELayoutPlot)

# Plotting everything
plt.iplot(sourceFigure,  filename = 'S_k_x')
plt.iplot(sourceEFigure,  filename = 'SE_k_x')
plt.iplot(transferFigure,  filename = 'Theta_l_x')
plt.iplot(transferEFigure,  filename = 'ThetaE_l_x')
```

```
plt.iplot(JlFigure,   filename = 'Jl_x')
plt.iplot(integrand1Figure,   filename = 'integrand1_x')

plt.iplot(integrand11Figure,   filename = 'integrand11_x')


plt.iplot(integrandTFigure,   filename = 'integrandT_k')
plt.iplot(integrandEFigure,   filename = 'integrandE_k')
plt.iplot(integrandTEFigure,   filename = 'integrandTE_k')

plt.iplot(CTTFigure,   filename = 'CTT_l')
plt.iplot(CEEFigure,   filename = 'CEE_l')
plt.iplot(CTEFigure,   filename = 'CTE_l')

pio.write_image(sourceFigure,  outputDir + '/' + 'S_x.pdf')
pio.write_image(sourceEFigure, outputDir + '/' + 'SE_x.pdf')
pio.write_image(transferFigure,  outputDir + '/' + 'Theta_l_k.pdf')
pio.write_image(transferEFigure,  outputDir + '/' + 'ThetaE_l_k.pdf')

pio.write_image(JlFigure,  outputDir + '/' + 'Jl_x.pdf')
pio.write_image(integrand1Figure,  outputDir + '/' + 'int1_x.pdf')
pio.write_image(integrand11Figure,  outputDir + '/' + 'int11_x.pdf')

pio.write_image(integrandTFigure,  outputDir + '/' + 'intT_k.pdf')
pio.write_image(integrandEFigure,  outputDir + '/' + 'intE_k.pdf')
pio.write_image(integrandTEFigure,  outputDir + '/' + 'intTE_k.pdf')

pio.write_image(CTTFigure,  outputDir + '/' + 'CTT_l.pdf')
pio.write_image(CEEFigure,  outputDir + '/' + 'CEE_l.pdf')
pio.write_image(CTEFigure,  outputDir + '/' + 'CTE_l.pdf')
```