CLASS tutorial

Chakkrit Kaeonikhom

December 16, 2023

CLASS overview

CLASS (The Cosmic Linear Anisotropy Solving System) is an Einstien-Boltzmann code for simulating the evolution of linear perturbations in the universe. CLASS is very structured, user-friendly, and flexible to modify. CLASS was written by Julien Lesgourgues & Thomas Tram, first released in 2011.[1]

CLASS is written in `C` language for each module. It comes with `C++` and `Python` wrapper.

For more information about CLASS can be found on the website: http://class-code.net.

[1] Lesgourgues, J. (2011) [1104.2932, 1104.2933].

What's CLASS do?

- Solves the coupled Einstein-Boltzmann equations for many types of matter in the Universe to first order in perturbation theory.
- Computes CMB observables such as temperature and polarisation correlations C^{TT}, C^{TE}, C^{EE}, C^{BB}
- Computes LSS observables such as the total matter power spectrum $P(k)$ and individual density and velocity transfer functions.

What can you get from CLASS?

Cosmic distances

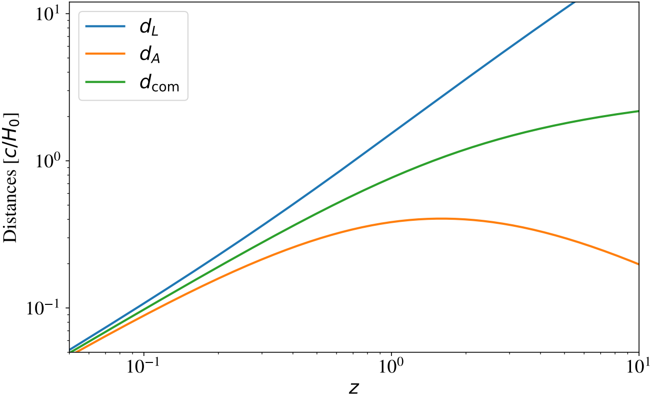


Figure: Cosmic distances

What can you get from CLASS?

Background evolution

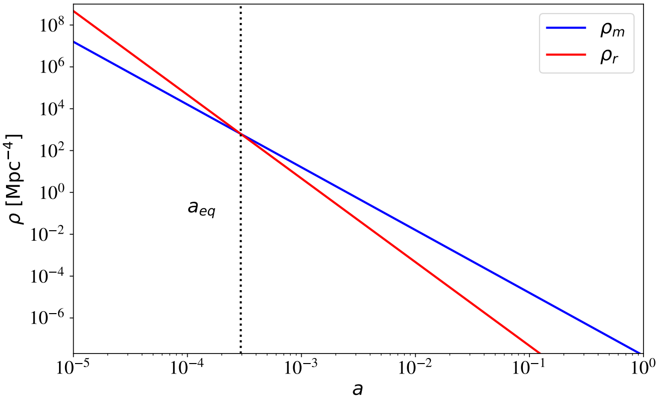


Figure: Evolution of matter and radiation

What can you get from CLASS?

Thermal history

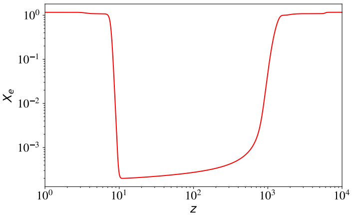


Figure:

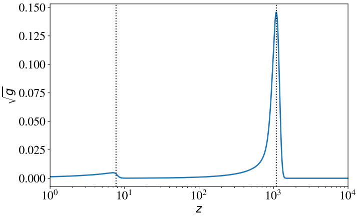


Figure:

What can you get from CLASS?

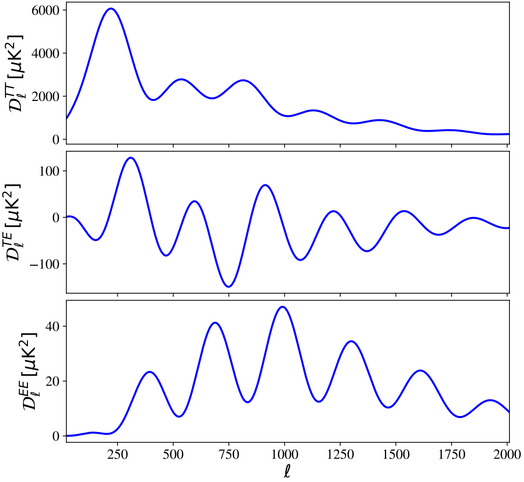


Figure: Plot of CMB spectra. Note that $\mathcal{D}_\ell^{XX} \equiv \ell(\ell+1)C_\ell^{XX}/2\pi$

What can you get from CLASS?

Matter power spectrum

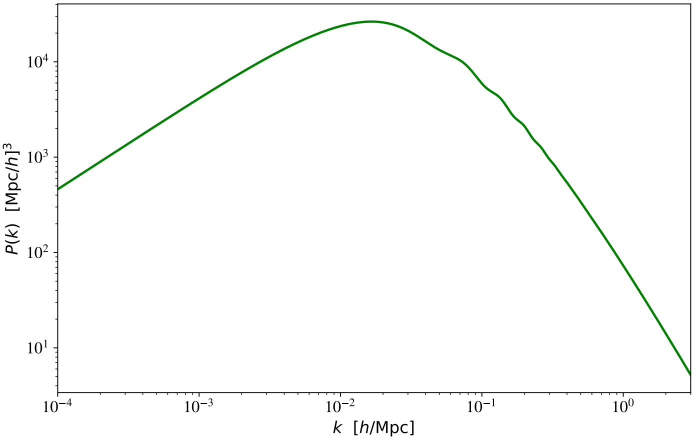


Figure:

What can you get from CLASS?

Evolution of density perturbations

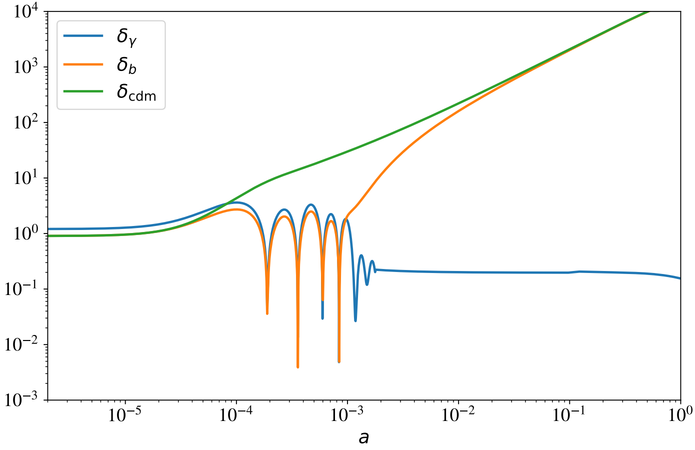


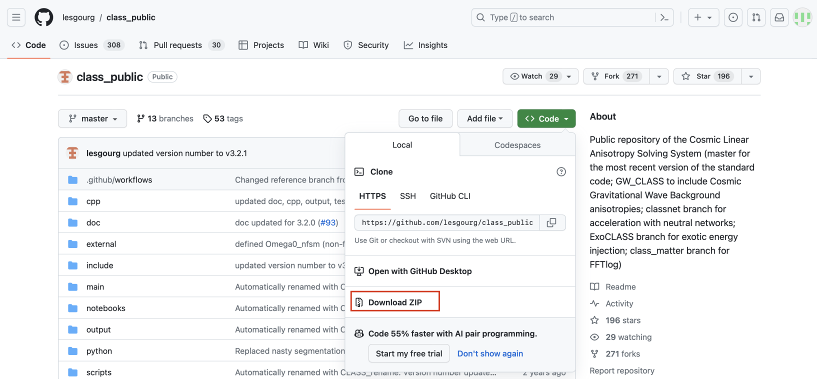
Figure:

CLASS installation

CLASS can be found on `github`, you can easily get the CLASS by opening the terminal and then typing

```
$ git clone https://github.com/lesgourg/class_public.git
```

or download directly from https://github.com/lesgourg/class_public



Go to CLASS directory via terminal. Compile C code and Python wrapper using the command

```
$ make -j
```

Execute the command

```
$ ./class explanatory.ini
```

to test if the C code installed successfully.

- All possible input parameters and details on the syntax are explained in explanatory.ini
- This is only a reference file, try not to modify it, but rather to copy it and reduce it to a shorter and more friendly file.
- explanatory.ini ≡ full documentation of the code.
- Output comes from 10 verbose parameters fixed to 1 in explanatory.ini (see them with > tail explanatory.ini)

Background module

Units

CLASS uses the unit $\hbar = c = k_B = 1$, This makes all dimensional quantities having unit in the form Mpc^n

- `t` stands for (cosmological or proper time)*c in Mpc
- `tau` stands for (conformal time)*c in Mpc
- `H` stands for (Hubble parameter)/c in Mpc^{-1}
- etc.

New since v3.0: all quantities that should normally scale with some power of a_0^n are renormalised by a_0^{-n}, in order to be independent of a_0, e.g.

- a in the code stands for a/a_0 in reality
- `tau` in the code stands for $a_0 \tau c$ in Mpc
- any prime in the code stands for $(1/a_0) d/d\tau$
- `r` stands for any comoving radius times a_0
- etc.

Background module

Friedmann equation

For example, by using natural units, the Friedmann equation can be written in the form

$$a' = a^2 H = a^2 \sqrt{\sum_i \rho_i - \frac{K}{a^2}}, \tag{1}$$

where we have defined $\rho_i \equiv \frac{8\pi G}{3}\rho_i^{\mathrm{physical}}$, where "$'$" is derivative respected to the conformal time τ,

$$\tau = \int \frac{dt}{a(t)}. \tag{2}$$

Tram (CANTATA Summer School, 2017)

Background functions

background_functions()

Most quantities can be instantly calculated from a given value of a:

$$\text{Energy density:} \quad \rho_i = \Omega_{i,0} H_0^2 a^{-3(1+w_i)} \tag{3}$$

$$\text{Pressure:} \quad p_i = w_i \rho_i \tag{4}$$

$$\text{Hubble parameter:} \quad H = \sqrt{\sum_i \rho_i - \frac{K}{a^2}} \tag{5}$$

$$\text{Derivative of } H\text{:} \quad H' = \left(-\frac{3}{2} \sum_i (\rho_i + p_i) + \frac{K}{a^2} \right) a \tag{6}$$

$$\text{Critical density:} \quad \rho_{\text{crit}} = H^2 \tag{7}$$

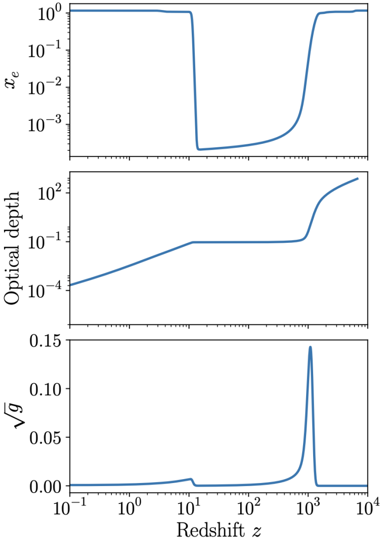
$$\text{Density parameter:} \quad \Omega_i = \frac{\rho_i}{\rho_{\text{crit}}} \tag{8}$$

Thermodynamics module

Thermal history

- electron fraction $x_e \equiv n_e/n_p$.
- optical depth $\kappa(\tau)$,
  $\kappa' = \sigma_T a n_p x_e$.
- visibility function
  $g(\tau) = \kappa' e^{-\kappa}$.
- etc.

Tram (CANTATA Summer School, 2017)



Perturbation module

Einstein and Boltzmann equations

We must solve 2 of the 4 first order Einstein equations:

$$k^2\eta - \frac{1}{2}\frac{a'}{a}h' = -4\pi G a^2 \delta\rho,$$

$$k^2\eta' = 4\pi G a^2 (\rho + p)\theta,$$

$$h'' + 2\frac{a'}{a}h' - 2k^2\eta = -24\pi G a^2 \delta p,$$

$$h'' + 6\eta'' + 2\frac{a'}{a}(h' + 6\eta') - 2k^2\eta = -24\pi G a^2 (\rho + p)\sigma$$

together with the Boltzmann equation for each species present in the Universe.

Perturbation module

The Boltzmann equation

- At an abstract level we can write:

$$\mathcal{L}\left[f_\alpha(\tau, \mathbf{x}, \mathbf{p})\right] = \mathcal{C}\left[f_i, f_j\right] (= 0). \qquad (1)$$

  The last equal sign is true for a **collisionless** species.
- We expand f_α to first order:

$$f_\alpha(\tau, \mathbf{x}, \mathbf{p}) \simeq f_0(q)(1 + \Psi(\tau, \mathbf{x}, q, \hat{n})). \qquad (2)$$

- Plugging equation (2) into equation (1) gives a Boltzmann equation for Ψ in Fourier space:

$$\frac{\partial \Psi}{\partial \tau} + i\frac{qk}{\epsilon}(\mathbf{k}\cdot\hat{n})\Psi + \frac{d\ln f_0}{d\ln q}\left[\dot{\eta} - \frac{\dot{h} + 6\dot{\eta}}{2}(\hat{k}\cdot\hat{n})^2\right] = \mathcal{C}$$

Perturbation module

The Boltzmann equation has a formal solution in terms of an integral along the line-of-sight:

$$\Theta_l(\tau_0, k) = \int_{\tau_{\rm ini}}^{\tau_0} d\tau \; S_T(\tau, k) \; j_l(k(\tau_0 - \tau)) \tag{9}$$

where

$$S_T(\tau, k) \equiv \underbrace{g\,(\Theta_0 + \psi)}_{\rm SW} + \underbrace{\left(g\, k^{-2}\theta_{\rm b}\right)'}_{\rm Doppler} + \underbrace{e^{-\kappa}\,(\phi' + \psi')}_{\rm ISW} + \text{pol.} \tag{10}$$

Tram (CANTATA Summer School, 2017)

Python wrapper with Jupyter Notebook

We can use Python wrapper to call cosmological quantities from CLASS modules. Execute a command on the terminal to launch the Jupyter Notebook:

```
$ jupyter notebook
```

Initialise the code

```
%matplotlib inline
import numpy as np
import scipy.constants as const
phi_gold =const.golden_ratio # Optional
import matplotlib.pyplot as plt
import matplotlib

from classy import Class
```

Python wrapper with Jupyter Notebook

You can setup font family and axis-label font size if you prefer (optional)

```
# Optional
font = {'size'    : 18, 'family':'STIXGeneral'}
axislabelfontsize='large'
matplotlib.rc('font', **font)
```

Basic running

```
# LCDM
lcdm = Class()

# This is where to input parameters
lcdm.set({'Omega_b':0.05, 'Omega_cdm':0.26})
lcdm.compute()
```

After compiling `lcdm.compute()`, we can now call CLASS functions. For example, getting distances form CLASS:

```
# Optional
font = {'size'   : 18, 'family':'STIXGeneral'}
axislabelfontsize='large'
matplotlib.rc('font', **font)
```

Example: cosmological distances

```
# LCDM
lcdm = Class()

# This is where to input parameters
lcdm.set({'Omega_b':0.05, 'Omega_cdm':0.26})
lcdm.compute()
```

Since quantities in CLASS using unit in the form of Mpc^n. We may have to convert to the unit we want. So we define some helpful quantities:

```
c_in_kmps = const.c/1000 # speed of light in km/s
H0_1_over_Mpc = lcdm.Hubble(0) # call Hubble constant
↪  (in unit of Mpc^{-1}) at redshift 0
# ----------------------------------------
# Note: To recover the Hubble constant in km/s/Mpc,
# multiplying H_1_over_Mpc with the speed of light (km/s)
H0 = H0_1_over_Mpc*c_in_kmps

# ----------------------------------------
c_over_H0 = 1./H0_1_over_Mpc # c/H0 in Mpc
```

After execute `lcdm.compute()` (or any name you prefer), we can simply type `lcdm.` and press TAB command. You can see a drop-down list that you can call from CLASS.

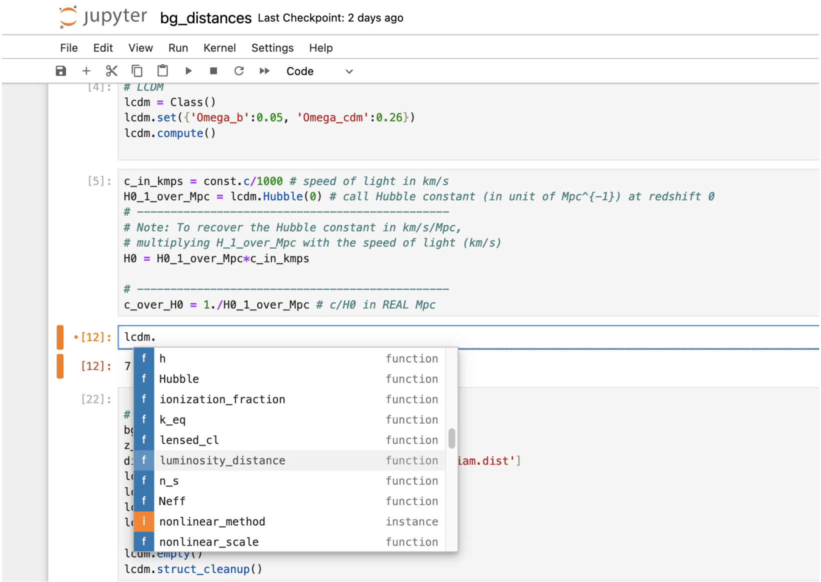


Figure:

To call background quantities, we just simply write

```
bg_lcdm = lcdm.get_background()
```

Now type `bg_lcdm[` and press TAB

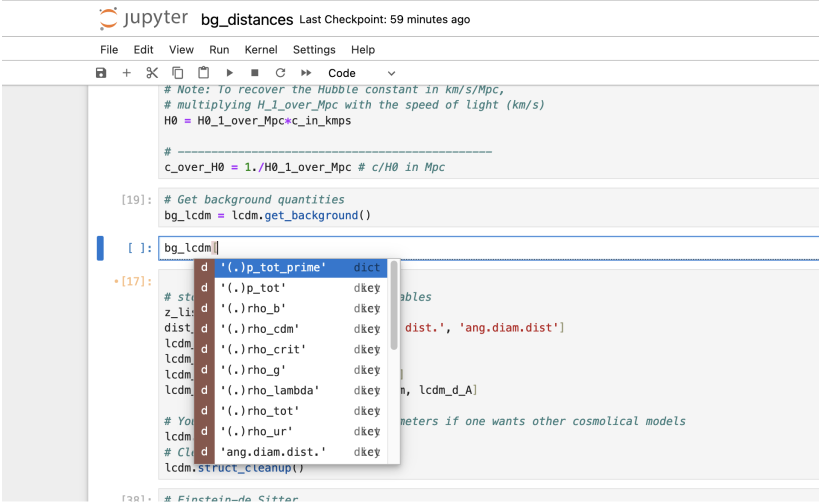


Figure:

Or we can see the keys by compiling

```
bg_lcdm.keys()
```

```
# Output:
# dict_keys(['z', 'proper time [Gyr]', 'conf. time
↪   [Mpc]', 'H [1/Mpc]', 'comov. dist.',
↪   'ang.diam.dist.', 'lum. dist.', 'comov.snd.hrz.',
↪   '(.)rho_g', '(.)rho_b', '(.)rho_cdm',
↪   '(.)rho_lambda', '(.)rho_ur', '(.)rho_crit',
↪   '(.)rho_tot', '(.)p_tot', '(.)p_tot_prime', 'gr.fac.
↪  D', 'gr.fac. f'])
```

We may have to store the background quantities in new variables, then we can compile `lcdm.struct_cleanup()` to recover the machine memory.

```
# store background quantities in new variables
z_list = bg_lcdm['z']
dist_names = ['lum. dist.', 'comov. dist.',
↪   'ang.diam.dist']
lcdm_d_L = bg_lcdm['lum. dist.']
lcdm_com = bg_lcdm['comov. dist.']
lcdm_d_A = bg_lcdm['ang.diam.dist.']
lcdm_distances = [lcdm_d_L, lcdm_com, lcdm_d_A]

# You may have empty the input parameters if one wants
↪   other cosmolical models
lcdm.empty()
# Clean up memory used in running
lcdm.struct_cleanup()
```

Einstein-de Sitter model

```
# Einstein-de Sitter
EdS = Class()
EdS.set({'Omega_b':0.05, 'Omega_cdm':0.95})
EdS.compute()

# store backgound quantities in new variables
bg_EdS = EdS.get_background()
z_list = bg_EdS['z']
dist_names = ['lum. dist.', 'comov. dist.', 'ang.diam.dist']
EdS_d_L = bg_EdS['lum. dist.']
EdS_com = bg_EdS['comov. dist.']
EdS_d_A = bg_EdS['ang.diam.dist.']
EdS_distances = [EdS_d_L, EdS_com, EdS_d_A]

EdS.empty()
EdS.struct_cleanup()
```

Plotting

```
# Plotting
from matplotlib.lines import Line2D # this library is for creating 2D lines
cl = ['r','b','g'] # color: red, blue, green

fig, ax = plt.subplots(figsize=(6*phi_gold, 6))
# loop over three types of distances
for i in range(3):
    # with Lambda
    ax.loglog(z_list, lcdm_distances[i]/c_over_H0, color=cl[i],
    ↪  label=r'$\Lambda\mathrm{CDM}$, $O_\Lambda=0.69$')
    # EdS
    ax.loglog(z_list, EdS_distances[i]/c_over_H0, ls = '--', color=cl[i],
    ↪  label=r'EdS')

plt.xlim(0.05,10)
plt.ylim(0.05,20)
plt.xlabel('$z$')
plt.ylabel(r'Distances [$c/H_0$]')
plt.text(2, 6, r'$d_L$')
plt.text(3, 2, r'$d_\text{com}$')
plt.text(3.5, 0.4, r'$d_A$')

# customise plot legend
custom_lines = [Line2D([0], [0], color='k', lw=2),
                Line2D([0], [0], color='k', lw=2, ls='--')]
plt.legend(custom_lines, [r'$\Omega_\Lambda=0.69$', 'EdS'])
plt.show()
# plt.savefig('bg_distances.pdf') # comment in if you want to export the plot
```

And Vala!

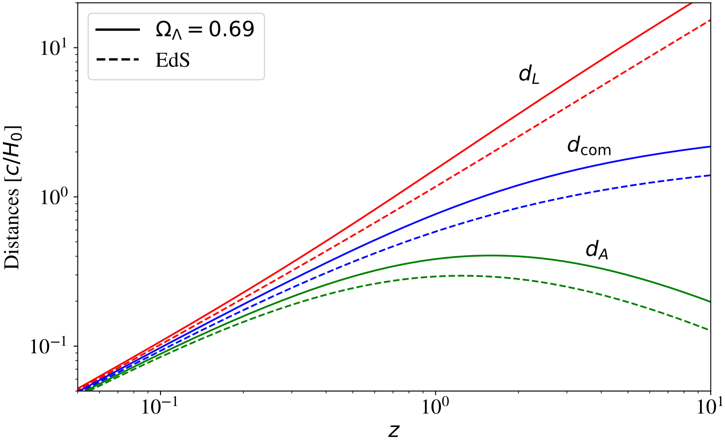


Figure: Cosmological distances

Plotting CMB spectra

To get CMB spectra, we need to add more information in the setting:

```
fixed_settings = {
                'T_cmb':2.7255,
                'omega_b':0.02238280,
                'omega_cdm':0.1201075,
                'h':0.67810,
                'A_s':2.100549e-09, # amplitude of primodial power
                ↪   spectrum
                'n_s':0.9660499, # scalar spectral index
                'output':'tCl,pCl,lCl', # temperature, polarisation
                ↪   and lensing spectrum
                'lensing':'yes' # say yes if you want CMB lensing,
                ↪   needs 'lCl'
                }
```

Then set the parameters, and do compute

```
cosmo = Class()               # call class
cosmo.set(fixed_settings)     # input parameters
cosmo.compute()               # compute cosmology
T_cmb = cosmo.T_cmb()         # get CMB temperature
raw_cl = cosmo.raw_cl(2500)   # get raw Cl
cosmo.empty()                 # clear input
cosmo.struct_cleanup()        # free the machine memory
```

Full code

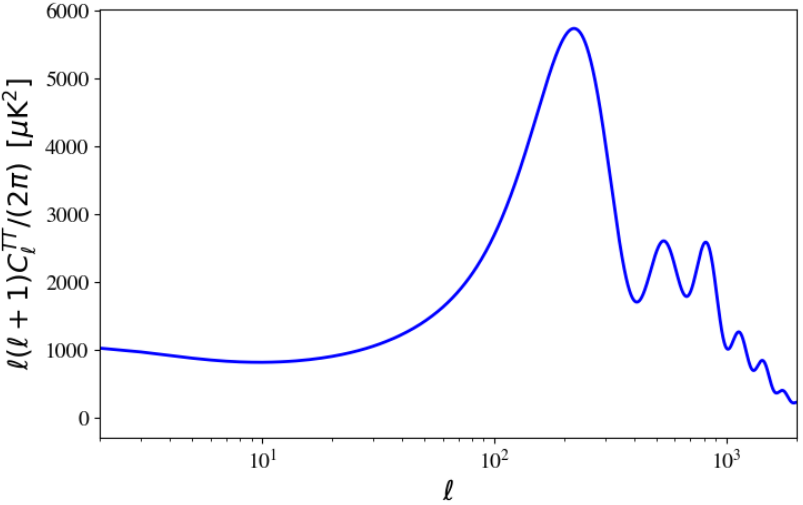
```
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import scipy.constants as const
phi_golden = const.golden_ratio # optional
# ----------- import Class ------------
from classy import Class
# -------- setup the plot (optional) -----
font = {'size'   : 14, 'family':'STIXGeneral'}
axislabelfontsize='large'
matplotlib.rc('font', **font)
plt.rcParams["figure.figsize"] = [5.0*phi_golden,5.0]
```

Full code (continue)

```
fixed_settings = {
                'T_cmb':2.7255,
                'omega_b':0.02238280,
                'omega_cdm':0.1201075,
                'h':0.67810,
                'A_s':2.100549e-09, # amplitude of primodial power
                ↪  spectrum
                'n_s':0.9660499, # scalar spectral index
                'output':'tCl,pCl,lCl', # temperature, polarisation
                ↪  and lensing spectrum
                'lensing':'yes' # say yes if you want CMB lensing,
                ↪  needs 'lCl'
                }
# --- computing ---
cosmo = Class()              # call class
cosmo.set(fixed_settings)    # input parameters
cosmo.compute()              # compute cosmology
T_cmb = cosmo.T_cmb()        # get CMB temperature
raw_cl = cosmo.raw_cl(2500) # get raw Cl
cosmo.empty()                # clear input
cosmo.struct_cleanup()       # free the machine memory
```

Full code (continue)

```
l = raw_cl['ell'][1:]
Cl_TT = raw_cl['tt'][1:]
factor = l*(l+1)/(2*np.pi)*T_cmb**2*1e12
plt.semilogx(l, factor*Cl_TT, color='b',lw=2)
plt.xlim(2, 2500)
plt.ylabel(r'$\mathcal{D}_\ell^{TT}\,\,\,[\mu\mathrm{K}^
↪   2]$',fontsize=18)
plt.xlabel(r"$\ell$",fontsize=18)
plt.show()
# plt.savefig('Cl_TT.pdf') # save figure
```



6000
5000
4000
3000
2000
1000
0
$\ell(\ell+1)C_\ell^{TT}/(2\pi)$ [μK^2]
10^1
10^2
10^3
ℓ

Figure: