CLASS Tutorial - TOOLS, November 2020

Deanna C. Hooper

deanna.hooper@ulb.be

Exercises 3.2 and 3.3 are based on previous tutorials by Julien Lesgourgues & Thomas Tram

1 CLASS overview

CLASS is designed to simulate the evolution of linear perturbations in the universe and to compute the cosmological observables for a given input model. CLASS was written by Julien Lesgourgues & Thomas Tram, and first released in 2011.

CLASS aims at being

- General: it features numerous models and many different cosmological outputs.
- Modern: CLASS is written in C with a modular structure. It comes with a wrapper for both python and C++.
- User-friendly: it is well documented (both inside the code and with the comprehensive online documentation). The code is easy to follow and equations are not repeated, making it straightforward to modify. CLASS also comes with several example jupyter notebooks.
- Accurate and fast: agrees with CAMB (similar, older code written in Fortran) at the 10^{-4} (0.01%) level for CMB observables, and runs in a couple of seconds for most models.
- Up-to-date: continuously maintained and expanded with more models and features, while maintaining compatibility with older versions.

More information about CLASS, including previous courses and tutorials can be found on the CLASS home page. CLASS can be used freely, provided that in your publications you cite at least the CLASS II: Approximation schemes paper.

2 Getting CLASS

If you are in a rush or are already familiar with this type of code, you can go for the short installation instructions. If you have never done it before, or if you want to know what every step is doing, go for the detailed version.

2.1 Short version (tl:dr)

On most Linux machines (or a Mac with gcc), downloading, compiling, and running CLASS can be done with four simple commands:

```
git clone https://github.com/lesgourg/class_public.git
cd class_public
make -j
./class explanatory.ini
```

2.2 Very detailed version

Requirements

Installation should be straightforward on Linux, and slightly more tricky (but still doable) on Mac. We suggest to not even try on Windows (i.e. you are on your own if you try).

In order to compile the C code, it is recommended to use gcc. Other C compilers like icc can also be used, but clang (native Mac C compiler) can prove problematic. To compile the python wrapper you can use either python 2.7 or 3, and you will need the modules `numpy` and `cython`.

Downloading CLASS

CLASS can be downloaded in two different ways: via git (recommended) or the traditional way from a tar file. In both cases, the installation instructions are the same.

To download via git (always gets you the newest version) type:

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

To download the tar:

- The latest tar file can always be found on the CLASS home page. The current (October 30, 2020) version is class_public-2.9.4.tar.gz.
- Once downloaded, extract the files either manually or via command line with

  ```
  tar -xzvf class_public-2.9.4.tar.gz
  ```

Installing CLASS

Once the code is downloaded, in a terminal navigate to your new CLASS folder. Now you can choose to compile only the main C code, or also the python wrapper (required for doing the CLASS exercises in a python notebook and for using it within MONTEPYTHON).

To compile the C code type:

```
make -j class
```

To compile the C code and the python wrapper type:

```
make -j
```

To test if the C code was installed successfully type:

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

This will run CLASS with the default input file. You should see some output on the screen, and no error messages. To test if the python wrapper installation also worked, you should be able to do the following with no errors:

```
python
>>> from classy import Class
```

If something went wrong with either of the installations, check out the detailed compilation issues page on the main CLASS wiki.

3 CLASS exercises

With the python wrapper CLASS can also be used from within jupyter notebooks, which allows for quicker visualisation of the output, and is ideal if you are used to a cell-like structure as in Mathematica. This can be installed via pip or anaconda, and more detailed instructions can be found on the jupyter site. Note that this is not a requirement for using CLASS, you can also do the exercises by running CLASS from the command line and using your favourite plotting tool to visualise the output.

3.1 Different CLASS outputs

All possible input parameters and details on the syntax for CLASS are explained in the example input file, explanatory.ini. For basic usage of the code, this provides a full documentation of what you need. This file is intended as a reference file: it is recommended that you do not modify this file, but rather you can either copy it and modify your copy, or create a new .ini file with only the parameters you need.

1. Create a copy of explanatory.ini, called myfirsttest.ini. Open your file with your favourite text editor and see if you can locate the main ΛCDM parameters.

2. Run your file once without any modifications with `./class myfirsttest.ini`. Look into the output folder and see what files were created: you should see the C_ℓs, the lensed C_ℓs, a list of used parameters stored in a .ini file that can be used for future runs, and a list of unused parameters (these are declared in your input file, but not used in the run).

3. Now we will change the output files. Open your .ini file and locate the `output` options. By default, CLASS computes the temperature (tCl), polarisation (pCl), and lensing (lCl) C_ℓs. Add the matter power spectrum (mPk) to the list of outputs.

4. Run your file with `./class myfirsttest.ini`, and check in the output folder that you have a new file, containing the matter power spectrum.

5. Open up your .ini file again, and locate the `write background` and `write thermodynamics` flags. Set them both to yes, and rerun your file. You should now have two additional files, containing a table of background and thermodynamic quantities as a function of redshift. Check your output folder to find the expected files.

6. Finally, locate the ten verbose flags (at the end of the .ini file). These control how talkative the different modules of CLASS are. Try increasing the verbose flags, and rerun your .ini file. You should see much more output on your screen.

7. *Optional*: use your favourite plotting utility to plot the temperature angular power spectrum. For a quick plot, you can use the in-built CLASS plotting utility (CPU), but note that this is a very bare-bones plotting tool. To plot the temperature angular power spectrum, you can use:

   ```
   python CPU.py output/myfirsttest01_cl.dat -y TT
   ```

   To find the full usage of CPU, type the following:

   ```
   python CPU.py --help
   ```

Parameter	TT+lowE 68% limits	TE+lowE 68% limits	EE+lowE 68% limits	TT,TE,EE+lowE 68% limits	TT,TE,EE+lowE+lensing 68% limits	TT,TE,EE+lowE+lensing+BAO 68% limits
$\Omega_b h^2$	0.02212 ± 0.00022	0.02249 ± 0.00025	0.0240 ± 0.0012	0.02236 ± 0.00015	0.02237 ± 0.00015	0.02242 ± 0.00014
$\Omega_c h^2$	0.1206 ± 0.0021	0.1177 ± 0.0020	0.1158 ± 0.0046	0.1202 ± 0.0014	0.1200 ± 0.0012	0.11933 ± 0.00091
$100\theta_{MC}$	1.04077 ± 0.00047	1.04139 ± 0.00049	1.03999 ± 0.00089	1.04090 ± 0.00031	1.04092 ± 0.00031	1.04101 ± 0.00029
τ	0.0522 ± 0.0080	0.0496 ± 0.0085	0.0527 ± 0.0090	$0.0544^{+0.0070}_{-0.0081}$	0.0544 ± 0.0073	0.0561 ± 0.0071
$\ln(10^{10} A_s)$	3.040 ± 0.016	$3.018^{+0.020}_{-0.018}$	3.052 ± 0.022	3.045 ± 0.016	3.044 ± 0.014	3.047 ± 0.014
n_s	0.9626 ± 0.0057	0.967 ± 0.011	0.980 ± 0.015	0.9649 ± 0.0044	0.9649 ± 0.0042	0.9665 ± 0.0038
H_0 [km s^{-1} Mpc^{-1}]	66.88 ± 0.92	68.44 ± 0.91	69.9 ± 2.7	67.27 ± 0.60	67.36 ± 0.54	67.66 ± 0.42
Ω_Λ	0.679 ± 0.013	0.699 ± 0.012	$0.711^{+0.033}_{-0.026}$	0.6834 ± 0.0084	0.6847 ± 0.0073	0.6889 ± 0.0056
Ω_m	0.321 ± 0.013	0.301 ± 0.012	$0.289^{+0.026}_{-0.033}$	0.3166 ± 0.0084	0.3153 ± 0.0073	0.3111 ± 0.0056
$\Omega_m h^2$	0.1434 ± 0.0020	0.1408 ± 0.0019	$0.1404^{+0.0034}_{-0.0039}$	0.1432 ± 0.0013	0.1430 ± 0.0011	0.14240 ± 0.00087
$\Omega_m h^3$	0.09589 ± 0.00046	0.09635 ± 0.00051	$0.0981^{+0.0016}_{-0.0018}$	0.09633 ± 0.00029	0.09633 ± 0.00030	0.09635 ± 0.00030
σ_8	0.8118 ± 0.0089	0.793 ± 0.011	0.796 ± 0.018	0.8120 ± 0.0073	0.8111 ± 0.0060	0.8102 ± 0.0060
$S_8 \equiv \sigma_8 (\Omega_m/0.3)^{0.5}$	0.840 ± 0.024	0.794 ± 0.024	$0.781^{+0.052}_{-0.060}$	0.834 ± 0.016	0.832 ± 0.013	0.825 ± 0.011
$\sigma_8 \Omega_m^{0.25}$	0.611 ± 0.012	0.587 ± 0.012	0.583 ± 0.027	0.6090 ± 0.0081	0.6078 ± 0.0064	0.6051 ± 0.0058
z_{re}	7.50 ± 0.82	$7.11^{+0.91}_{-0.75}$	$7.10^{+0.87}_{-0.73}$	7.68 ± 0.79	7.67 ± 0.73	7.82 ± 0.71
$10^9 A_s$	2.092 ± 0.034	2.045 ± 0.041	2.116 ± 0.047	$2.101^{+0.031}_{-0.034}$	2.100 ± 0.030	2.105 ± 0.030
$10^9 A_s e^{-2\tau}$	1.884 ± 0.014	1.851 ± 0.018	1.904 ± 0.024	1.884 ± 0.012	1.883 ± 0.011	1.881 ± 0.010
Age [Gyr]	13.830 ± 0.037	13.761 ± 0.038	$13.64^{+0.16}_{-0.14}$	13.800 ± 0.024	13.797 ± 0.023	13.787 ± 0.020
z_*	1090.30 ± 0.41	1089.57 ± 0.42	$1087.8^{+1.6}_{-1.7}$	1089.95 ± 0.27	1089.92 ± 0.25	1089.80 ± 0.21
r_* [Mpc]	144.46 ± 0.48	144.95 ± 0.48	144.29 ± 0.64	144.39 ± 0.30	144.43 ± 0.26	144.57 ± 0.22
$100\theta_*$	1.04097 ± 0.00046	1.04156 ± 0.00049	1.04001 ± 0.00086	1.04109 ± 0.00030	1.04110 ± 0.00031	1.04119 ± 0.00029
z_{drag}	1059.39 ± 0.46	1060.03 ± 0.54	1063.2 ± 2.4	1059.93 ± 0.30	1059.94 ± 0.30	1060.01 ± 0.29
r_{drag} [Mpc]	147.21 ± 0.48	147.59 ± 0.49	146.46 ± 0.70	147.05 ± 0.30	147.09 ± 0.26	147.21 ± 0.23
k_D [Mpc^{-1}]	0.14054 ± 0.00052	0.14043 ± 0.00057	0.1426 ± 0.0012	0.14090 ± 0.00032	0.14087 ± 0.00030	0.14078 ± 0.00028
z_{eq}	3411 ± 48	3349 ± 46	3340^{+81}_{-92}	3407 ± 31	3402 ± 26	3387 ± 21
k_{eq} [Mpc^{-1}]	0.01041 ± 0.00014	0.01022 ± 0.00014	$0.01019^{+0.00025}_{-0.00028}$	0.010398 ± 0.000094	0.010384 ± 0.000081	0.010339 ± 0.000063
$100\theta_{s,eq}$	0.4483 ± 0.0046	0.4547 ± 0.0045	0.4562 ± 0.0092	0.4490 ± 0.0030	0.4494 ± 0.0026	0.4509 ± 0.0020
f_{2000}^{143}	31.2 ± 3.0			29.5 ± 2.7	29.6 ± 2.8	29.4 ± 2.7
$f_{2000}^{143\times217}$	33.6 ± 2.0			32.2 ± 1.9	32.3 ± 1.9	32.1 ± 1.9
f_{2000}^{217}	108.2 ± 1.9			107.0 ± 1.8	107.1 ± 1.8	106.9 ± 1.8

Figure 1: Planck 2018 results.

3.2 Playing with the Planck best-fit model

This exercise does not require any plotting or manipulation of input/output data, so it does not require a python script or notebook. You can simply run this in terminal, the same as you did in the previous exercise.

Figure 1 shows a table with the constraints on the standard cosmological parameters from the Planck 2018 Cosmological Parameter paper. The Planck baseline constraints (including BAO data) are in the last column, so we will use that as our reference, but you could do this exercise with any of the columns.

1. Write a short input file with the appropriate values of ω_b, ω_{cdm} ($\Omega_c h^2$ in the table), H_0 or h, τ_{reio} (τ in the table). Run only the background and thermodynamics module for this

model (to do so, just leave the `output` field blank)[1]. To get some standard output, set `input_verbose`, `background_verbose`, and `thermodynamics_verbose` to one. To be sure that you do not have syntax errors in your input file, setting `write warnings = yes` is advisable. Look at the different lines of the standard output. Do you reproduce accurately the age of the Universe in Gyr given in the table (here, accurately means e.g. up to better than 0.1σ) ?

2. Part of the difference comes from different settings for other parameters. To work with *exactly* the same parameters as in the Planck paper, you need to add one massive neutrino species with $m = 0.06$ eV. Thus you need to add one non-cold-dark-matter (ncdm) species, and to reduce the number of ultra-relativistic (ur) relics to two (plus small corrections coming from the details of neutrino decoupling models, which would be 0.046 for 3 massless neutrinos and 0.03351 for two massless neutrinos). Add the following parameters to your .ini file:

   ```
   N_ur = 2.03351
   N_ncdm = 1
   m_ncdm = 0.06
   ```

   Run again with these additional parameters. Do you get closer to the age indicated in the table?

   Check also that you get the same or nearly the same value for the redshift and comoving sound horizon at recombination (z_*, r_* in the table), redshift and comoving sound horizon at baryon drag ($z_{\rm drag}$, $r_{\rm drag}$ in the table), and reionization redshift ($z_{\rm re}$ in the table).

3. Check that if instead of the value of H_0 from the seventh row, you pass `100*theta_s` with the value indicated in the Planck table (in the 3rd row), you get a different value of H_0 from CLASS. The reason is that the table gives $\theta_{\rm MC}$, an analytic approximation to the actual ratio $\theta_s = d_s^{\rm dec}/d_A^{dec}$ (sound horizon at decoupling over angular diameter distance to decoupling) computed internally by CosmoMC. Instead, CLASS computes the true $\theta_s = d_s^{\rm dec}/d_A^{\rm dec}$ with a numerical integration. To know the true θ_s, go back to the previous run in which you specified h or H_0 in input. Use the standard output of the code to get the correct $100\,\theta_s$. Now pass this value and check that you get the right h.

4. *Note:* In addition to the standard .ini file, CLASS can take a .pre file, which is usually where you declare precision parameters if you want to push the code to even higher precision (often at the expense of runtime). These parameters could also be declared in your .ini file, but it is cleaner to separate them. You should see in your CLASS folder an example cl_ref.pre, which you can use as a short guideline. All of the precision parameters, as well as their default setting, can be found in include/precisions.h. To test running with a precision file, use `./class mytest.ini cl_ref.pre`. You should notice the output background and thermodynamics quantities only change at the fifth or sixth digit.

[1] In this exercise, we care only about the background and thermodynamics evolution, so it is not necessary to pass the same values of $\ln(10^{10}A_s)$ (or A_s) and n_s as in the paper. So they can be left unspecified. If we wanted to reproduce the same C_ℓs or $P(k)$ as for the Planck best-fit model, we would need to pass the correct $\ln(10^{10}A_s)$ (or A_s) and n_s, the correct pivot scale `k_pivot = 0.05` , and to specify some fields for `output` = ...

3.3 Impact of baryons on the matter power spectrum (More Advanced)

The objective of this exercise is to run CLASS multiple times, changing the value of one parameter to see how this impacts the matter power spectrum. This exercise is much easier with a python notebook or script calling CLASS in a loop. However, you can also do this exercise by running CLASS in a terminal with different .ini files, and then plotting the corresponding output files with your favourite plotting tool.

Plot the ratio of matter power spectrum for ~ 5 values of ω_b (log-lin) ranging from ~ 0 to 0.1, and check the effect of baryons: it should be a step-like suppression with superimposed oscillations (corresponding to the BAOs).

Hint: Start from `varying_neff.ipynb` (in the notebooks folder) and adapt it to your needs. Ideally you would take a reference model with $\omega_b = 0$, but no Boltzmann code will allow you to do that: a certain amount of baryons is needed in order to have a tightly-coupled photon-baryon fluid at the time of initial conditions. Therefore, it is recommend to take a very small value like $\omega_b = 10^{-3}$ as the reference. Since for the purpose of the plot you will use very unrealistic values of ω_b, the code cannot use its feature of "automatically determining the primordial Helium fraction Y_{He} using an interpolation table encoding the results of BBN codes", so it is better to fix the primordial Helium fraction manually (this can be done with `YHe = 0.25` in your .ini file).

Your final figure should look similar to Figure 2.

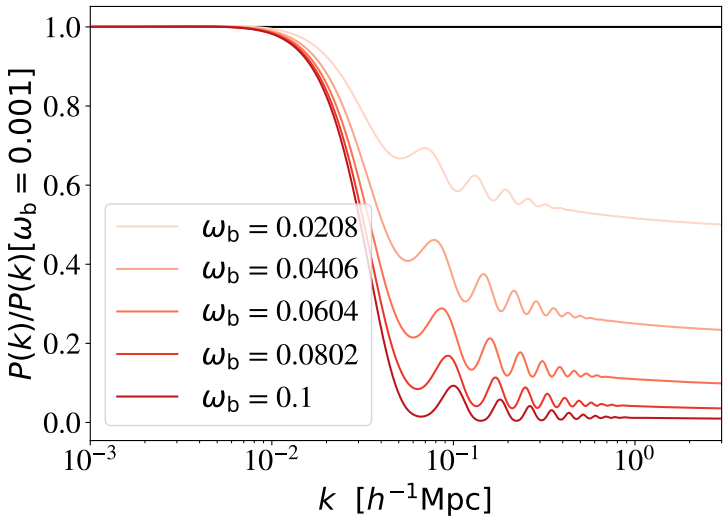

Figure 2: Effect of baryons on the matter power spectrum.

Note: When you modify the ω_b, you increase the overall matter content of the universe (ω_M). In order to maintain the budget equation, CLASS will automatically adjust the amount of Dark Energy (ω_Λ). If instead you want to maintain the amount of Dark Energy fixed, but change the amount of Dark Matter to adjust the budget equation, you can add a line in your script to compute ω_{cdm} such that $\omega_b + \omega_{cdm} =$ constant.