YETI Tutorial on CMB anisotropies computation and parameter inference[a]

January 8, 2015

I. CLASS

For this tutorial we will be using the virtual box that you downloaded from the School website. However, once you're ready to use it in the real world, CLASS can be obtained at `http://class-code.net`. The website also includes detailed information on the required packages and installation, as well as links to a week's worth of lectures[1] (from which these notes are partially condensed) and relevant publications about the code.

CLASS, stands for Cosmic Linear Anisotropy Solving System. It is the latest of many "Boltzmann codes", the most well-known being CAMB[2], which has been maintained and updated since 1999. I've chosen CLASS because its logical structure makes it a bit more approachable and (in my opinion) easier to modify.

Given a set of cosmological parameters, the CLASS code computes:

1. The background cosmology evolution
2. The matter power spectrum & transfer functions
3. The CMB anisotropy spectra
4. Lensing potential spectra
5. Thermal history since recombination

The main reasons to use CLASS then would be to **analyze new data**, **look for new observables**, or make predictions based on **new physics**.

A. The CLASS modules

CLASS is written in C. It is composed of 10 "modules" located in the `source/` directory. The main program `class.c` file just runs each of these modules in turn. Each module puts together a tabulated "structure" containing the observables and parameters needed either for subsequent modules or for output. For example, the thermodynamics structure is named `th`, referenced by the pointer `pth`. If you want the free electron fraction x_e, for instance, you'll just write `pth->xe`. Most variables follow this basic structure, so it's normally a question of determining which module computes the parameter you want, and finding its name within that module's header (.h) file.

Here is a quick rundown of the modules, with their associated structure fields

background: (`pba-> ...`) This is where the background cosmology (i.e. Friedmann equations) is computed. Note that **all the units are in Mpc**^{n}: times and distances in Mpc, rates (like H) are in Mpc^{-1}, densities and pressures are in Mpc^{-2}: $\rho = 8\pi G/3\rho_{\rm physical}$. The background module also contains background functions that compute e.g. ρ, p, H without the need to solve any differential equations. Note that the time variable is τ, the proper time (normally denoted by η).

thermodynamics: (`pth-> ...`) This is where CLASS computes the free electron fraction $x_e \equiv n_{\rm free\,electrons}/n_p$, the optical depth κ and the visibility function g, in order to solve the **recombination** and **reionisation** equations. Can choose between different recombination algorithms: accurate (many energy levels), RECFAST (Peebles two-level model with fudge factors), as well as HyRec and CosmoRec.

perturbations: (`ppt-> ...`) This module integrates the Boltzmann equations in order to obtain the source functions $S_i(k,\tau)$. Here the **linear** scalar and tensor perturbation equations are solved, following the Ma & Bertschinger[3] formalism. Both **synchronous** and **Newtonian**[4] gauges are implemented in perturbations.c: the gauge is

---

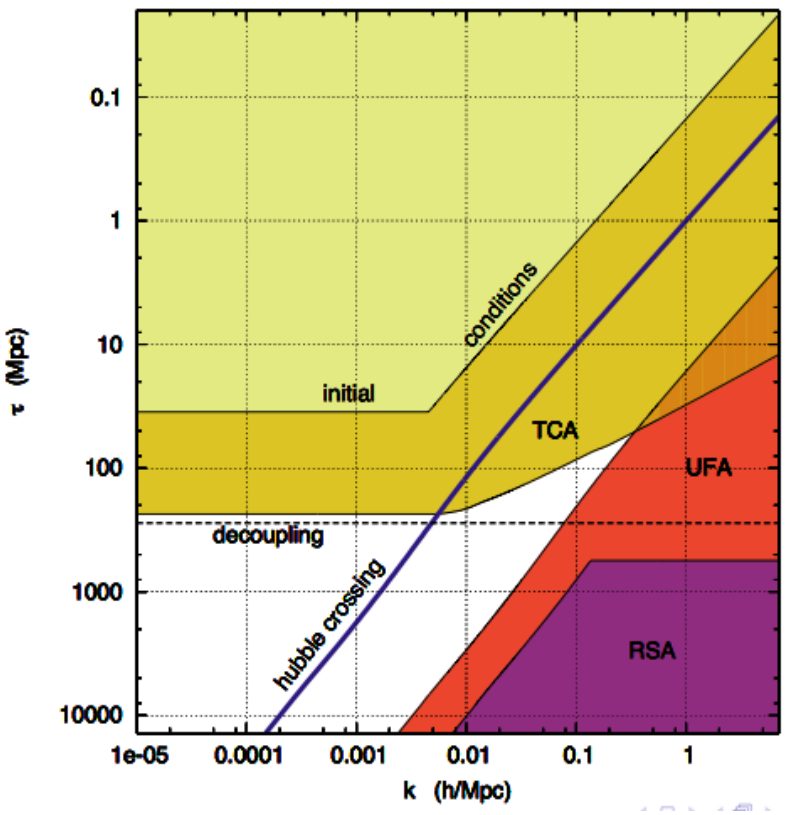
[a] Prepared by Aaron Vincent (aaron.vincent@durham.ac.uk) with help from Ryan Wilkinson (ryan.wilkinson@durham.ac.uk)

[1] specifically, here: `http://lesgourg.web.cern.ch/lesgourg/class-tour/class-tour.html`

[2] `http://camb.info`

[3] `http://arxiv.org/abs/astro-ph/9506072`

[4] Only the sychronous gauge is used in CAMB. This can be a problem, e.g. when looking at models of interacting dark matter, since the DM velocity perturbation $\theta_{DM} = 0$ in synchronous gauge.



specified in the .ini file, so if you are modifying the perturbation routine, it is important to check that you are in the right gauge!

This is also where the approximation schemes are implemented – see figure. These are described in detail in Ma & Bertschinger.

primordial: (`ppm->...`) This is where the primordial perturbations are introduced. It allows various specifications for the primordial power spectra $\mathcal{P}(k)$, including the simple $A_s + n_s$, or more involved parametrisations with running. Alternatively you may specify a model of inflation.

nonlinear: (`pnl->...`) Optional: compute nonlinear correction $R^{NL}(k,\tau) = \delta_m^{NL}/\delta_m^L$ using HALOFIT, and multiply the source functions.

transfer: Where the transfer functions, $\Delta_l^X(q)$ (i.e. projection of the source functions onto Bessels) are computed, where $X = T, E, ...$:

$$\Delta_l^X(q) = \int d\tau S_X(k,\tau)\phi_l^X(q,(\tau_0-\tau) \tag{1}$$

for both CMB and LSS.

spectra: (`psp->...`)Where the spectra are finally computed:

$$P_L(k,z) = \left(\delta_m(k,\tau(z))\right)^2 \mathcal{P}(k) \tag{2}$$

and

$$C_l^{XY} = 4\pi \sum_{i,j} \int \frac{dk}{k} \Delta_l^X(k)\Delta_l^Y(k)\mathcal{P}(k) \tag{3}$$

for all combinations of T (temperature) E (E-mode polarization) B (B-mode) P (CMB lensing) N_i (galaxy number in ith bin) and L_i (galaxy lensing in ith bin).

lensing: This module takes the unlensed spectra $C_l^{TT,TE,\cdots}$ and uses the lensing potential C_l^{PP} to compute the lensed spectra $\tilde{C}_l^{TT,TE,\cdots}$. This is the final piece of physics.

output: Exactly what it says. Depending on the flags in your .ini file, will generate the output based on the structures stored by the previous modules.

B. The .ini file

The .ini file is the basic input for CLASS. The file explanatory.ini contains the full structure, with optional commands commented out (#). It's recommended to leave it as is, and make a copy to play with:

```
$ cp explanatory.ini  yourname.ini
```

You should skim through your copy of explanatory.ini to get an understanding of how CLASS inputs operate. For the most part, this is the only portion that you'll be modifying. However, if you want to do something more fancy, you'll need to play with the code.

C. Running CLASS

To run CLASS, simply type, from the `class_v2.4.1/` directory:

```
$ ./class yourname.ini
```

D. The Outputs

Outputs go into the `output/` directory, and are named `yourname##_observable.dat`. Where `yourname` is whatever you called your .ini file, `##` is a counter to avoid overwriting previous runs.

1. Plotting the output

CLASS comes with a python plotting utility called CPU. To see the options, type

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

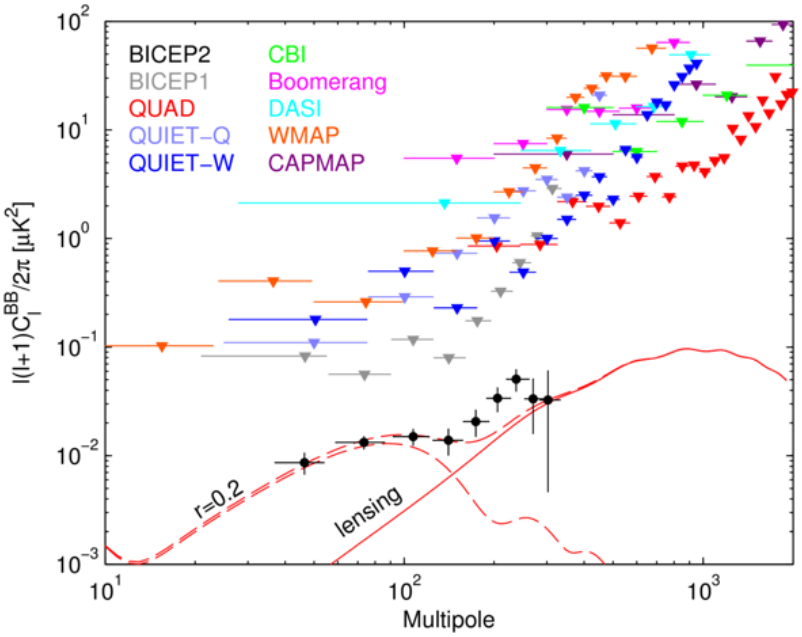
There is also a Matlab script, `plot_CLASS_output.m` which does essentially the same. You can of course use whatever program you prefer.

E. Exercises

1. The ΛCDM model relies on six parameters: $\{\Omega_b, \Omega_{CDM}, H_0, n_s, A_s, z_{reio}\}$. To get familiar with class and its outputs, try varying these. What are the effects on the CMB C_l^{TT}?

2. By default the neutrinos are treated as massless (ultrarelativistic) species, with $N_{UR} = 3.046$. To add a massive neutrino, change this to `N_ur = 2.0328`, and change the number of "non-CDM" species `N_ncdm = 1`. Now give it a mass `m_ncdm = 1.` (eV), and a temperature `T_ncdm = 0.71611`. Plot the (lensed) TT spectrum. Now try a new run with the default massless neutrinos, but this time changing the Hubble parameter to `h = 0.74`. You'll see that the peaks now line up with your massive neutrino case. We've found a degeneracy. Try plotting the lensing potential spectrum. There should be a large difference at high multipoles: this shows how new observables can resolve this type of degeneracy.

3. Reproduce the lines in the following plot:



Note that to get the right units in the output, you must set `format = CAMB`.

4. You can plot the time evolution of perturbations at various scales. Add `k_output_values = 0.01,0.1,1` and plot the evolution of the different components (i.e. δ_b, δ_{cdm}, ...).

5. Adding `write thermodynamics = yes` to the ini file writes the thermodynamics data. Plot the ionisation fraction x_e vs redshift.

   (a) Try a few reionisation models (`camb, none, bins_tanh`) and compare their effects on $x_e(z)$.

   (b) Dark matter annihilation can inject energy into the IGM plasma after recombination, leaving a larger amount of residual ionization. This is parametrized by $p_{ann} \equiv \langle\sigma v\rangle_{ann}/m_\chi$. It is implemented in CLASS with the `annihilation` parameter, in units of kg m^{-3} s^{-1}. Try a few values (current limit is around 10^{-6}) and look at the effect on x_e and C_l^{TT}.

6. Time to modify the code. To get a feeling for the procedure, try adding a new component χ to the cosmological background, with equation of state $p_\chi = w_\chi \rho_\chi$ with constant w_χ.

   (a) You will need to add two new variables: the abundance Ω_χ and the equation of state parameter w_χ. Open `background.h`, and add `double Omega0_chi;` and `double w_chi;` to the `background` structure. These can now be referenced via `pba-> Omega0_chi` and `pba-> w_chi`.

   (b) You will also have to add an index in `background.h`: `int index_bg_rho_chi;` around line 166. This will tell us where the density values are stored for our species in the `background` array.

   (c) `input.c` now needs to be modified. Look at how `Omega0_cdm`, for example, is added (around line 800). Try to replicate this for `Omega0_chi` and `w_chi`.

   (d) If the user does not specify a default value, these parameters need to be defined. Have a look around line 2600 of `input.c` and add defaults for your two new parameters.

   (e) Open `background.c` and have a look around line 300. This is where the pressure and densities are computed. Once you've figured out what's going on with baryons, CDM and so forth, try adding the contribution of your new species. Build the code using `$ make`, and see if you can plot the background evolution for a few cases of w_χ and Ω_χ.

II. BAYESIAN INFERENCE WITH MONTEPYTHON

Parameter inference[5] is the process of determining the best model parameters θ_i given data x, where $i = 1, ..., n$. Your model gives you $L(x|\theta_i)$ (the likelihood of observing these data given these parameters), but you really want $L(\theta_i|x)$ (the likelihood of each parameter set being true, given the observed data). Bayes theorem allows us to find this:

$$L(\theta_i|x)\frac{L(x|\theta_i)L(\theta_i)}{\int d^n\theta_i L(x|\theta_i)L(\theta_i)} \tag{4}$$

The left-hand side is called the *posterior distribution*. $L(x|\theta_i)$ is the *sampling distribution*, and $L(\theta_i)$ is the *prior likelihood*, which reflects our degree of belief in the model before looking at the data x. This is sometimes used to indicate an extra constraint on a given parameter: it is information known independent of the data x. The denominator is called the *Bayesian evidence*, and represents the likelihood summed over all realizations of the model: it is crucial in model comparison, but is not useful for parameter inference within a given model. The *marginal* posterior represents the posterior likelihood of a given parameter θ_j, and it is what we are really after:

$$L(\theta_j) \propto \int d^{n-1}\theta_{i\neq j} L(\theta_i|x). \tag{5}$$

Evaluating this integral analytically is impossible in most cases, and for n larger than a few, numerical evaluation of (5) is prohibitively expensive. To get around this we use a Markov Chain Monte Carlo (MCMC) search[6]. An MCMC can be seen as a point jumping stochastically through the parameter space with the probability of each jump proportional to the likelihood.[7] After enough iterations, the *chain* of samples $\{\tilde{\theta}_i\}$ taken by the point will be distributed according to the posterior likelihood. Eq. (5) is then just a histogram of $\{\tilde{\theta}_j\}$.

A. MontePython

MontePython is an MCMC suite that is specially tailored to CLASS (although it can be used beyond CLASS). It can be used for creating and analyzing MCMC chains with likelihoods given by CLASS, and also allows for different sampling algorithms including MultiNest and CosmoHammer.

MontePython is available for download at `http://montepython.net`. The site has extensive documentation including installation instructions (non-trivial) and instructions to install and use the Planck likelihoods (highly non-trivial).

From the base `montepython/` directory, there are several important files and directories:

1. *the .param file*

The basic structure of the .param file can be seen in `base.param`. The line

```
data.experiments=['Planck_highl','Planck_lowl','lowlike']
```

tells MP which likelihoods to use (these are described in the next section). The model parameters θ_i are given with:

```
# data.parameters[class name] = [mean, min, max, 1-sigma, scale, role]
# - if min max irrelevant, put to -1
# - if fixed, put 1-sigma to 0
# - if scale irrelevant, put to 1, otherwise to the appropriate factor
# - role is either 'cosmo' or 'nuisance'
```

"cosmo" means that the parameter is passed to CLASS for each evaluation; nuisance parameters go directly to the likelihood computation, and are normally associated with a given experiment. Other cosmological parameters that you wish to specify can be written like:

```
data.cosmo_arguments['N_eff'] = 2.03351
```

---

[5] This is a ridiculously brief introduction. There are many references on Bayesian inference in cosmology, as well as the different tools of the trade. Here is a good start: `http://arxiv.org/pdf/0803.4089v1.pdf` for the theory side; to better understand how it works in practice, the CosmoMC paper `http://arxiv.org/pdf/astro-ph/0205436v3.pdf` is a good reference.

[6] There are other algorithms, such nested sampling, differential evolution and genetic algorithms which use different methods to reach this goal.

[7] The jumps are found by picking a point inside the proposal distribution $Q(\theta_{iN}, \theta_{iN+1})$. If nothing is known about the distribution, an n-dimensional Gaussian is a rational choice for Q. If degeneracies between parameters are known via a covariance matrix, a judicious choice of proposal distribution can significantly speed up your exploration by favouring long jumps in degenerate directions and shorter jumps in steeper directions.

2. Likelihoods

Likelihood functions are located in the `montepython/likelihoods/` directory. They can contain the data itself, point to data in the `data` directory, or to more complex likelihood functions like the WMAP and Planck clik libraries, which must be installed separately.

3. Running MontePython

Before running MontePython, you need to point it to your CLASS installation. Copy the `default.conf.template` file to `default.conf`. Modify the file so that `path['cosmo']` is your CLASS directory (i.e. `~/cosmoTutorial/class_v2.4.1`).

To see the available run options, type:

```
montepython/$ python montepython/MontePython.py --help
```

At the very minimum, you must specify a .param file and an output directory:

```
montepython/$ python montepython/MontePython.py -p base.param -o chains/base
```

If you are using likelihoods with many nuisance parameters (such as Planck), a covariance matrix can (and should!) be included for the proposal distribution, with the flag `-c covmat/mycovmat.covmat`. This will greatly speed up your runs. You can also use the covariance matrix output from a previous run. The `-N` flag specifies the number of likelihood evaluations that you want. This supersedes the `data.N` line in the .params file.

4. Output

Outputs are stored in the directory specified after the `-o` flag in your run. Output files are numbered sequentially, so you don't have to worry about overwriting. This means that multiple chains can be launched at once, from the same directory! The optimal running strategy, for a run using the Planck likelihood, would be to launch 10 or so chains with `data.N = 10000`.

To analyze the output, use:

```
montepython/$ python montepython/MontePython.py -info chains/yourchains
```

This creates figures in in the `chains/yourchains/plots/` directory. Note that our VirtualBox is a bit capricious. If you are getting Latex errors, try commenting out the following lines in `analyze.py`, by adding a # character in front:

```
matplotlib.rc('text', usetex=True)
```

This will mess up the labels a bit, but don't worry, on a normal machine this isn't necessary.

B. Exercises

1. We do not have time do a full run with any serious likelihood or large number of dimensions, but we can go through the steps. Run a chain of 1000 points using `test.param` and plot the results. It probably looks terrible. Run another few thousand points to see how your contours improve.

2. Now try adding an experiment. The documentation on montepython.net gives us an example: we'll modify the hst likelihood to create a new one, called spitzer. Just remember that the likelihood name needs to be shared by the directory, the class and the data:

   ```
   $ mkdir likelihoods/spitzer
   $ cp likelihoods/hst/hst.data likelihoods/spitzer/spitzer.data
   $ cp likelihoods/hst/__init__.py likelihoods/spitzer/__init__.py
   ```

   Next, change the class name in `montepython/likelihoods/spitzer/__init__.py`, and change the entries in `spitzer.data` to:

   ```
   spitzer.h = 0.743
   spitzer.sigma = 0.021
   ```

   That's it! Try running a chain with the `'spitzer'` data.

3. MontePython can vary a function of one (or more) of the cosmology parameters, rather than the parameter itself. These "mapped" parameters are defined in the `data.py` module in the `update_cosmo_arguments()` function. Have a look at how parameters here are redefined, and try to copy the syntax to create a new variable, `logann`, which varies the log of the `annihilation` parameter in CLASS. Try to run a small chain with your new variable with p_{ann} in the range $[10^{-7}, 10^{-4}]$ kg m^{-3} s^{-1}.

4. We have included some previous chains from a ΛCDM run with the Planck likelihoods in in the `lcdm_planck_chains.zip` file. Unzip these to `montepython/chains/planck` and have a look at the `log.param` file to see how this run was set up. You can also look through the different chains and other output files. Analyze the data using the `-info` option:

   ```
   $ python montepython/MontePython.py info chains/base --extra plot_files/options_base.txt
   ```

   The last part tells MontePython to use our customized `options_base.txt` file in order to plot only the relevant cosmological parameters.