MontePython Exercises
TOOLS 2020
Online/Lyon, 2020

Thejs Brinckmann (courtesy of Miguel Zumalacarregui)

November 3, 2020

See the slides from the MontePython talk on indico for more information and help[1]. Slides 19-40 contain additional material that was not covered in the talk. In particular, slides 31-40 may be useful for Exercise 2, slides 31-34 for Exercise 3, and slides 8-13 (explaining the param file) for all exercises.

The exercises have been ranked by level of difficulty (boring = 🌶, interesting = 🌶🌶, challenging = 🌶🌶🌶) and whether they are computationally *fast* or *slow* to do. The *fast* exercises you can likely run directly on your laptop, whereas you may want to leave the *slow* one running on the cluster and return later for results.

MontePython with parallel chains: if your cluster does not have `MPI` pre-installed, do not despair! `MontePython` is fully compatible with parallel runs, with no perceptible loss in efficiency, even with periodic covariance matrix update and running jumping factor adaptation. Instead you should just launch multiple chains with separate executions of `MontePython` pointing to the same chains directory. You can also have each process run on a few cores with e.g. `OMP_NUM_THREADS=4` (in general `CLASS` is parallelized to 8 cores, with some parts up to 32 cores).

Exercise 1: The $\Omega_m - \Omega_\Lambda$ plane (🌶→🌶🌶, *fast*)

This exercise relies on constraints from the cosmic expansion: is fast enough to be done on a laptop ($\sim$ 0.1s/model on mine). You can let it run during the coffee break and make some cool plots when you come back!.

We will obtain Baryon Acoustic Oscillation (BAO) constraints on the $\Omega_m - \Omega_\Lambda$ plane for a two parameter model. We will vary Ω_{cdm} and Ω_k, keeping Ω_b and H_0 fixed by setting the 1σ values to zero (recall Ω_Λ is a derived parameter). Use the following example:

```
data.experiments=['bao_boss_dr12','bao_smallz_2014']
data.parameters['Omega_cdm'] = [0.3, 0, None, 0.05, 1, 'cosmo']
data.parameters['Omega_k'] = [0.0, -0.5,0.5, 0.05, 1, 'cosmo']
data.parameters['Omega_b'] = [0.045, 0, None, 0.0, 1, 'cosmo']
data.parameters['h'] =      [0.68, 0, None,  0, 1, 'cosmo']
data.parameters['Omega_Lambda'] = [1,None,None,0, 1, 'derived']
data.cosmo_arguments['YHe'] = 0.24
data.N=10
data.write_step=5
```

It is important to give meaningful names to the files and folders to keep track of your work. I suggest labeling this combination of parameters and experiments as `lc_bao.param` (this notation means `l` for Λ, `c` = CDM and `bao` = BAO from BOSS galaxies).
Note: the limits on Ω_k are such that `CLASS` does not complain. Fixing the Helium fraction Y_{He} is good

[1] https://indico.cern.ch/event/955391/contributions/4075773/attachments/2136226/3598227/201104_MontePython_TOOLS2020.pdf

to run other cases, like SNe constraints varying Ω_b as well.

The basic part of the exercise (🌶) involves the following steps:

a) Write the above into a `.param` file and do a short Monte Python run:

```
time python montepython/MontePython.py -p lc_bao.param -o chains/lc_bao -N 10
```

The `time` prefix is just to know how long it will take (you can use that information to adjust the parameters to how much time you have).

b) Now you can do the serious run

```
mpirun -np 4 python montepython/MontePython.py -o chains/lc_bao -N 10000
```

where you can adjust `N` to a larger number (`-N 10000` should be feasible, if you take a longer break you can increase accordingly). Because you are running from a folder with a `log.param` you do not need to provide the `.param` file again.

⚠ In order to speed up convergence `--update 50` (default value, to improve the covariance matrix on the fly) and/or provide a covariance matrix with `-c name.covmat` (MP will interpret the parameters for you). You can also benefit from setting `--superupdate 20` (adapt jumping factor to optimize acceptance rate).

The call `mpirun -np 4` at the begining of the command runs 4 chains in the same console (adjust `-np` to your number of cores). If you don't have `openmpi` you can remove `mpirun` and just run on `-np` different consoles.

c) Relax while your computer does the work. If you come back early and there are enough points (go to output directory and type `wc -l *.txt` to count) you can cancel the work (press `Ctrl+c`).

d) Analyze the chains. You can use MP in `info` mode:

```
python montepython/MontePython.py info chains/lc_bao
```

plus the optional options. Take at the nice plots in `chains/lc_bao/plots` and all the other files generated in the analysis.

e) You are encouraged to play with the different options. Try to plot the marginalized contours using $\Omega_m = \Omega_{cdm} + \Omega_b$ instead of Ω_{cdm}.

f) Once you have several datasets (some ideas are suggested below) you can plot them together. Just feed MP `info` mode with several folders. How do the different constraints compare?

⚠ Make sure that each model/data combination goes to a different `-o` directory! Otherwise you'll keep running the same thing over and over again.

The rest of the exercise is optional and slightly harder (🌶🌶). Bear in mind that the amount of free parameters increases and the runs will require more time.

Realistic BAO: By not varying Ω_b, H_0 we are fixing the comoving BAO scale r_s (i.e. the *coordintate* size of the standard ruler). Although r_s is well constrained by the CMB, there is some variability, which you may take into account by letting Ω_b vary within some range.

Do a run varying the baryon fraction. You can do this in two ways

a) More elegant: add a gaussian prior on $\omega_b \equiv \Omega_b h^2$. This is the goal of exercise 2.

b) Easier: add a hard prior to allow for 2σ deviations (change `Omega_b`$\rightarrow$ `omega_b` in the `.param` file, as well as the central value and limits).

Supernovae: There are other background observations besides BAO, like type 1A Supernovae (SNe). Run the same model with the Union SNe compilation with `data.experiments=['sne']`.
For a more challenging option you can use the JLA SNe sample `data.experiments=['JLA']`. You need to download the data, the `numexpr` package and add several nuisance parameters. Read the instructions in the likelihood folder and `jla.param`.

Exercise 2: Adding a new likelihood (🌶, *fast*)

Adding a new likelihood in Montepython is only as complex as the likelihood itself. You will get to see with this simple example.

Our goal is to add a gaussian prior on the physical baryon density using the Planck result

$$\omega_b = \Omega_b h^2 = 0.02222 \pm 0.00023\,, \tag{1}$$

(`https://arxiv.org/abs/1502.01589`, Table 1, col 6). The steps are:

a) Copy a simple likelihood folder from `montepython/likelihoods` (for instance `hst`) and rename it as `cmb_baryon`. Change the name of the `.data` file to be `cmb_baryon.data`.

b) In `cmb_baryon.data` change `hst`$\rightarrow$`cmb_baryon` and `h`$\rightarrow$`omega_b`. Update the central value and standard deviation according to eq. (1).

c) Update `__init__.py` by changing the name of the class, the data (as given `.data` file, it is read as `self.xxx`) and the theoretical value (`cosmos.omega_b()` as given by `classy`)

You can launch a short run with

```
data.experiments=['cmb_baryon']
data.parameters['omega_b'] = [2, 0, None, 0.02, 1e-2, 'cosmo']
data.N=10
data.write_step=5
```

and check that the chains are roughly gaussianly distributed around the mean (note that the parameter is rescaled by `1e-2`).

Exercise 3: Constraining new input parameters (🌶🌶🌶, *slow*)

During the `CLASS` exercises you considered three new parameters related to neutrinos (more generally non-cold dark matter or extra massive or massless relativistic species) `N_ur`, `N_ncdm` and `m_ncdm`. It is possible to constrain some of these parameters using MontePython, if you select appropriate likelihoods and configure your param file correctly. A simpler option could be to swap out A_s (`A_s` or `ln10^{10}A_s`) for another large-scale structure amplitude parameter σ_8 (`sigma8`), which is often used by observational cosmologists.

We now want to do an MCMC run adding these parameters. Decide which parameter to constrain (or use multiple, if possible) and which datasets to use.

For σ_8

a) In order to constrain σ_8 we need datasets that constrain the amplitude of the matter power spectrum, for example `sdss_lrgDR7` and/or `kids450_qe_likelihood_public`. The SDSS galaxy clustering likelihood works out of the box, whereas the recent KIDS-450 weak lensing likelihood by Fabian Köhlinger requires downloading the data externally and changing the path to the data in the `.data` file. For instructions on using KIDS-450 see the readme in the likelihood folder for instructions.

b) Use a minimal param file as in exercise 1, adjusted for the new likelihoods, and try to get a run converged (do not expect it to converge during the exercise session). Tips: you can remove `Omega_k` in order to speed up the run. When constructing the param file, see the relevant example param files for SDSS or KIDS in the `input/` directory.

For `N_ur`:

1. To constrain `N_ur` we instead need a dataset that can constrain the number of extra relativistic species. CMB likelihoods are particularly useful for this. Here you have two options:
2. Install the Planck likelihoods (which can be some work, see the `MontePython` readme for help), and use a Planck temperature or temperature + polarization likelihood,
3. or use mock likelihoods. These are ready out of the box and you need a CMB likelihood, such as `fake_planck_realistic`, which is a CMB mock likelihood intended to mimic the sensitivity of a full Planck mission. Using a mock BAO likelihood would also be advantageous, for example `fake_desi_euclid_bao`, which is a neat likelihood combining redshift bins from each of the two surveys DESI and Euclid in order to construct a combined likelihood where redshift bins are used from the survey with the best expected sensitivity at that redshift.
4. First construct a param file starting from the example param files in `input/`, e.g. example.param for mock data or one of the Planck example param files (starting with base2018) for real data.
5. In order to do forecasting we first need to compute fiducial data spectra, which will serve as mock data in place of real data. This is done by setting the jumping factor to zero, i.e.

   ```
   python montepython/MontePython.py run -p param_file -o chains_dir -f 0
   ```

6. For Planck lite or mock Planck likelihoods (either with or without BAO likelihoods) we can begin by computing an inverse Fisher matrix for use as input covariance matrix via

   ```
   python montepython/MontePython.py run -p param_file -o chains_dir --method Fisher
   ```

7. Launch your run, remembering to pass the inverse Fisher matrix you computed as input covariance matrix, and try to get the run to converge (do not expect the run to converge during the exercise session). Note: you can find the inverse Fisher matrix in `chains_dir` named `inv_fisher.mat`.

For parameters related to ncdm it is a bit more complicated and you have multiple options. In the following we will cover the simplest example of constraining the neutrino mass sum using the ncdm species.

1. You have to make similar considerations as for `N_ur`, so start with the previous part (`N_ur`) and return here when you are at point 4.
2. When adding the new parameters to your parameter file you have to make additional choices. The simplest is to run with degenerate neutrino masses (all species have the same mass), this is faster and quite accurate for estimating the sum of neutrino masses.
3. To do this you have to set `N_ncdm` to 1 in the param file using `cosmo_arguments`, which corresponds to having one type of neutrinos.
4. You then set the degeneracy (number of degenerate species) with `deg_ncdm`. Set this to 3 using `cosmo_arguments`, corresponding to three active neutrinos. You could vary this parameter to constrain the number of extra relativistic species, but it would make the run take longer to converge. (note with this setup in order to get $N_{\text{eff}} = 3.046$ predicted by the Standard Model you should set `N_ur` to 0.00641 with `cosmo_arguments`, because of how CLASS computes N_{eff}, although this is not likely to change your constraints).
5. You can then vary `m_ncdm` as a cosmo parameter. Keep in mind that your sum of neutrino masses would be 3 x `m_ncdm` (or `deg_ncdm` x `m_ncdm`). MontePython has a parameter name that does this automatically for you, `M_tot`, which you can choose to vary instead and get the total neutrino mass directly.
6. Keep in mind that negative neutrino masses are unphysical and make sure to set the lower bound for `m_ncdm` or `M_tot` to 0 in the param file, as CLASS will complain otherwise.
7. Now you can continue with points 5-7 of the previous part (`N_ur`).