A first **MontePython** hands-on

Lisbon EUCLID meeting, 2.06.2016

Julien Lesgourgues & Sébastien Clesse (RWTH Aachen University)

To be done ideally before the session

1) Install **CLASS** (since the MontePython wrapper for **CAMB** is not yet available)
2) Install **MontePython**

- Where?
  - On your laptop if you prefer (OK for this session but you will never do big runs there… so maybe better to install it directly somewhere else?)
  - On some remote desktop or on your institute's cluster.
  - As usual: installation on LINUX is the easiest; on MacOS Mavericks, Yosemite, El Capitan: more tricky but documented in the installation guide; on Windows: just forget it.

- How?
  - Install **CLASS** and its python wrapper classy.py as explained in:

    https://github.com/lesgourg/class_public/wiki/Installation
  - Install the other python modules necessary for **MontePython** as explained in:

    [short version:] https://github.com/baudren/montepython_public/wiki/Installation

    [or more details at:] http://monte-python.readthedocs.io →click: Installation Guide

- In summary, what will you need to do?
  - You probably already have an appropriate C compiler and python 2.6
  - You may need to install 3 public python modules: cython, scipy, numpy (mpi4py is optional). That's all.
  - For these exercises, no need to install external codes like clik (Planck likelihood), Multinest, CosmoHammer, etc.
  - Finally, you will have to write the path to **CLASS** in montepyhton/default.conf as explained in the installation guide.

1. Your first run

1) Create a directory for your chains, e.g.:

```
> mkdir chains/
```

2) Run a mini-chain with 10 tries:

```
> python montepython/MontePython.py run -o chains/test1 -p example.param -N 10
```

Remark: if you know how to create aliases in your linux or MacOS, you can create the alias mprun *for* python montepython/MontePython.py run *and* mpinfo' *for* python montepython/MontePython.py info. *Then the last command would just read:*

```
> mprun -o chains/test1 -p example.param -N 10
```

3) Edit the file chains/test1/log.param to check its content. You can also edit the chain file chains/test1/*.txt to check that it is identical to a chain in CosmoMC format.

Remark: in this example, the likelihood fake_planck_bluebook *is of the type* Likelihood_mock_cmb, *it is an approximation to the Planck likelihood, written before the Planck launch. As you can see in* chains/test1/log.param, *it took its fiducial model in a file* data/fake_planck_bluebook_fiducial.dat *that you downloaded together with the code (you can edit it to see its fiducial parameter values).*

4) Now that the log.param exists, you can run more chains in the same directory without an input file. Check it with

```
> python montepython/MontePython.py run -o chains/test1 -N 10
```

check that this writes a second chain with

```
> ls chains/test1/*.txt
```

2. Your second run: Planck + Euclid_lensing forecast

1) copy example.param to a new input file euclid.param that you will customise

```
> cp example.param euclid.param
```

2) edit it, to add the euclid_lensing likelihood:

```
> data.experiments=['fake_planck_bluebook','euclid_lensing']
```

3) There is not yet a fiducial model for the euclid_lensing likelihood, it will be created automatically in the first run. However there is already a fiducial model for fake_planck_bluebook. We want to be sure that we create fiducial data for the two likelihoods for the same cosmological model. So it is better to create from scratch the fiducial data for both. Hence we will remove the old fiducial file for fake_planck_bluebook:

```
> rm data/fake_planck_bluebook_fiducial.dat
```

4) Now we run one model (-N 1) in order to create the fiducial data for both experiments. *Remark: we may use the option* -f 0 *(meaning: jump amplitude equals zero) in order to force fiducial parameter values equal to the "start" values in the input file (otherwise the values would be those after the first random step of the chain)*

```
> python montepython/MontePython.py run -o chains/test2 -p euclid.param -N 1 -f 0
```

The code should tell you that it created the fiducial data, and stop.

5) Now we can run a chain with e.g. 50 tries:

```
> python montepython/MontePython.py run -o chains/test2 -p euclid.param –N 50
```

Check the chain with

```
> more chains/test2/*txt
```

3. Analyzing the second run

1) of course, with one chain of 50 tries, you cannot make nice plots and tables. So Sébastien will provide you some converged chains to train with plotting.

   But in principle you could get good data for your second run on a cluster or a multi-core machine, after running several chains, preferentially at the same time and with the --update *option for much much faster convergence. For that, you may submit n times the identical command:*

```
> python montepython/MontePython.py run -o chains/test2 -p euclid.param –N 5000
```

   Or if mpi4py *is installed, you may just submit 1 time the command*

```
> mpirun –np 4 python montepython/MontePython.py run -o chains/test2 -p euclid.param –N 5000
```

   (details depend on your machine; you can have hybrid jobs with n chains using j cores each by setting appropriately OMP_NUM_THREADS*, etc. etc.)*

2) Ask a set of converged chains to Sébastien, copy them to some directory (e.g. chains/sebastien/euclid/) and analyse them with

```
> python montepython/MontePython.py info chains/sebastien/euclid/
```

   Then:

   - Check the content of the output files in chains/sebastien/euclid/
   - View the output PDF plots in chains/sebastien/euclid/plots/
   - Have a look at the analyse options with

     ```
     > python montepython/MontePython.py info --help
     ```

   - Have a look at the syntax for doing further cutomisation with an extra file (--extra plot_files/example.plot) with

     ```
     > more plot_files/example.plot
     ```

   - If Sébastien can provide a second run, you can compare the two with

     ```
     > python montepython/MontePython.py info chains/sebastien/euclid/ chains/sebastien/<another run>/
     ```

4. Your third run: Planck + Euclid_pk forecast

1) edit euclid.param, to replace euclid_lensing with euclid_pk :

```
> data.experiments=['fake_planck_bluebook','euclid_pk']
```

2) If you run now with:

```
> python montepython/MontePython.py run -o chains/test3 -p euclid.param -N 1
```

the code will complain: “Check your nuisance parameter list for your set of experiments”

3) Hence euclid_pk needs a nuisance parameter. To check its name, look at the .data file of this likelihood:

```
> more montepython/likelihoods/euclid_pk/euclid_pk.data
```

There is an uncommented line euclid_pk.use_nuisance = ['P_shot'] so this is the nuisance parameter we must add.

So edit your euclid.param file and add:

```
> data.parameters['P_shot'] = [0, -1, 1,1,1,'nuisance']
```

4) We are ready to rerun. Our last attempt to run in chains/test3 was unsuccessful, so by precaution we are first going to remove the log.param created by the unsuccessful run:

```
> rm chains/test3/log.param
```

5) Now we run one point in order to create the fiducial data file for euclid_pk (with –f 0 it will be for the same cosmological parameters as before):

```
> python montepython/MontePython.py run -o chains/test3 -p euclid.param -N 1 -f 0
```

6) Finally we run our chain:

```
> python montepython/MontePython.py run -o chains/test3 -p euclid.param -N 50
```