Quick introduction to MontePython

code: https://github.com/baudren/montepython_public

doc: http://monte-python.readthedocs.io

Main developers: Benjamin Audren, Julien Lesgourgues, + input from many others

Lisbon EUCLID meeting, 2.06.2016

Sébastien Clesse (RWTH Aachen University)
Modularity in MontePython

Currently ~60 likelihoods implemented (CMB, BAO, SNIa, Hubble, time delay, cosmic clocks, galaxy correlation, cosmic shear, …)

Class (C)
- CAMB (fortran) - official CAMB python wrapper
- CLASS (C) - python wrapper

Generic interface/API with Boltzmann codes
- sampler.py
- nested_sampling.py interface
- cosmo_hammer.py interface
- importance_sampling.py

Generic interface/API with samplers

Generic interface/API with likelihoods

mcmc.py (Metropolis-Hastings with Cholesky and covmat update)

Developed internally

Developed externally

In development (internally)
Modularity in MontePython

sampler.py

Generic interface/API with samplers/engines

montepython.py

mcmc.py (Metropolis-Hastings with Cholesky and covmat update)

nested_sampling.py interface

cosmo_hammer.py interface

importance_sampling.py

PyMultinest wrapper

Multinest (fortran)

CosmoHammer (C++)

....
Modularity in MontePython

CLASS (C)

CAMB (fortran)

classy.py python wrapper

official CAMB python wrapper

camby.py python interface

Generic interface/API with Boltzmann codes

montepython.py

sampler.py

developed internally

in development (internally)

developed externally
Modularity in MontePython

Currently ~60 likelihoods implemented (CMB, BAO, SNIa, Hubble, time delay, cosmic clocks, galaxy correlation, cosmic shear, ...), see them in montepython/montepython/likelihoods

Generic interface/API with likelihoods

sampler.py
Documentation

• CLASS html documentation:
  http://www.class-code.net ➔ click: online html documentation

• MontePython html documentation:
  http://monte-python.readthedocs.io

Installation

• Install CLASS and classy.py as explained in:
  https://github.com/lesgourg/class_public/wiki/Installation

• Install the other python modules necessary for MontePython as explained in:
  [or more details at:] http://monte-python.readthedocs.io ➔ click: Installation Guide

• Running with MPI requires one extra python module mpi4py, but this is is optional.
  - Advantage: run N chains from single command instead of N identical command lines.
  - But even without MPI, the covariance matrix will be updated using all chains (communication through file reading/writing instead of MPI).
Quick start

• **Code can be called in two modes (equivalents of CosmoMC and GetDist)**
  
  python montepython/MontePython.py run <options>
  python montepython/MontePython.py info <options>

• **Get a list of all options available separately for run and info with:**
  
  python montepython/MontePython.py run --help
  python montepython/MontePython.py info --help

• **GetDist fans can still use it instead of info (identical format of the chains; new: MontePython also writes .paramnames files for improved compatibility with GetDist)**

**doc:** http://monte-python.readthedocs.io
Typical run command

- Code can be called with (example for Metropolis-Hastings without MPI):

```bash
python montepython/MontePython.py run -p example.param -o output_directory/
-N 10000 -f 1.7 -c example.covmat -b example.bestfit --update 300
```

- First job in given directory will create a file `output_directory/log.param`
  - `log.param` stores for records everything you may need to remember on this run (details on dataset, Boltzmann code version, all cosmo/nuisance/derived/fixed parameters, etc.)
  - once it exists, `log.param` used as input file, with priority over `-p xxx.param`, to ensure that all chains in given directory correspond precisely to same model and datasets (if Boltzmann code version changed: run stops, forcing you to create a new output directory)
Typical info command

- Chain analysis can be done with:

```python
python montepython/MontePython.py info output_directory1/ [output_directory2/ ... output_directoryN/] --want-covmat
```

... plus lots of customisation options (see them with --help)

... with --extra optional_plot_file, any pyplot command can be passed directly to the code in order to customise the plots
Typical input file

#----- Experiments to test (separated with commas)-----
data.experiments=['planck_lite', 'euclid_pk']

#----- Settings for fast (nuisance) parameters, number of blocks depends on number of likelihoods with nuisance parameters
data.over_sampling=[1, 4]

#----- Parameter list (format: data.parameters[name] = [mean, min, max, 1-sigma, scale, role])----------

# 1. Cosmological parameters list (name must be directly understood by Boltzmann code’s parser)
data.parameters['omega_cdm'] = [0.1120, None, None, 0.0016, 1, 'cosmo']
data.parameters['tau_reio'] = [0, 0, None, 0.03, 1, 'cosmo']
data.parameters['A_s'] = [2.42, -1, -1, 0.038, 1e-9, 'cosmo']
...

# 2. Nuisance parameter list, same call; names must not be understood by Boltzmann code, but by at least one likelihood code

data.parameters['P_shot'] = [0, None, 1, 1, 1, 'nuisance']
...

# 3. Derived parameter list list (name must be understood by Boltzmann code’s wrapper: these are output from the Boltzmann code)
data.parameters['Omega_Lambda'] = [0, None, None, 0, 1, 'derived']
data.parameters['sigma8'] = [0, None, None, 0, 1, 'derived']
...

# fixed parameters for the Boltzmann code (physical parameters, precision parameters, etc.)
data.cosmo_arguments['k_pivot'] = 0.05
...

doc: http://monte-python.readthedocs.io
Adding likelihoods

- One likelihood = one python class defined in
  montepython/likelihoods/some-name/__init__.py__
  which reads all settings / parameters / data files in
  montepython/likelihoods/some-name/some-name.data

- This class inherits from properties of a mother class. May inherit from most general one, or from specific ones:
  Likelihood
  - Likelihood_clik
  - Likelihood_mock_cmb
  - Likelihood_sn
  - euclid_pk
  - etc.
  - Planck_lowl
  - Planck_highl
  - etc.
  - core
  - litebird
  - etc.
  - JLA
  - my-version-of-euclid_pk
  - etc.

- No need to duplicate code when your new likelihood has things in common with its mother... e.g. here is the whole likelihood code in montepython/likelihoods/Planck_highl_TTTEEE/__init__.py__:

```python
from montepython.likelihood_class import Likelihood_clik
class Planck_highl_TTTEEE(Likelihood_clik):
    pass
```

while montepython/likelihoods/Planck_highl_TTTEEE/Planck_highl_TTTEEE.dat just contains the path to clik and the definition of nuisance parameters...
Running with mock data

• your likelihood file `montepython/likelihoods/some-name/some-name.data` contains the path to some fiducial model, e.g. `data/some-name_fiducial.dat`

• First run: the code finds no file there. Then it automatically computes the fiducial model at the starting point of the MCMC, stores it in this file, and stops with a warning.

• Later runs: the code will use this fiducial model as mock data.

• This logic is in-built, you don’t need to think about it anymore when you add a new mock likelihood.