MontePython

Thejs Brinckmann, Deanna C. Hooper, Julien Lesgourgues

MontePython + CLASS Kavli workshop

Code developed by
Audren, Brinckmann, Lesgourgues & many others

Lecture, Cambridge, 12/09/18

Institute for Theoretical Particle Physics and Cosmology
RWTH AACHEN UNIVERSITY

Overview

1. Brief introduction of new features in **MontePython 3**

2. Overview of the structure of the code

3. Basic introduction on how to use the code

4. Advanced usage and details on **MontePython 3** improvements

MontePython 3: new features

New version 3.0 of **MontePython** earlier this year

▶ see release paper Brinckmann & Lesgourgues 1804.07261

Notable new features

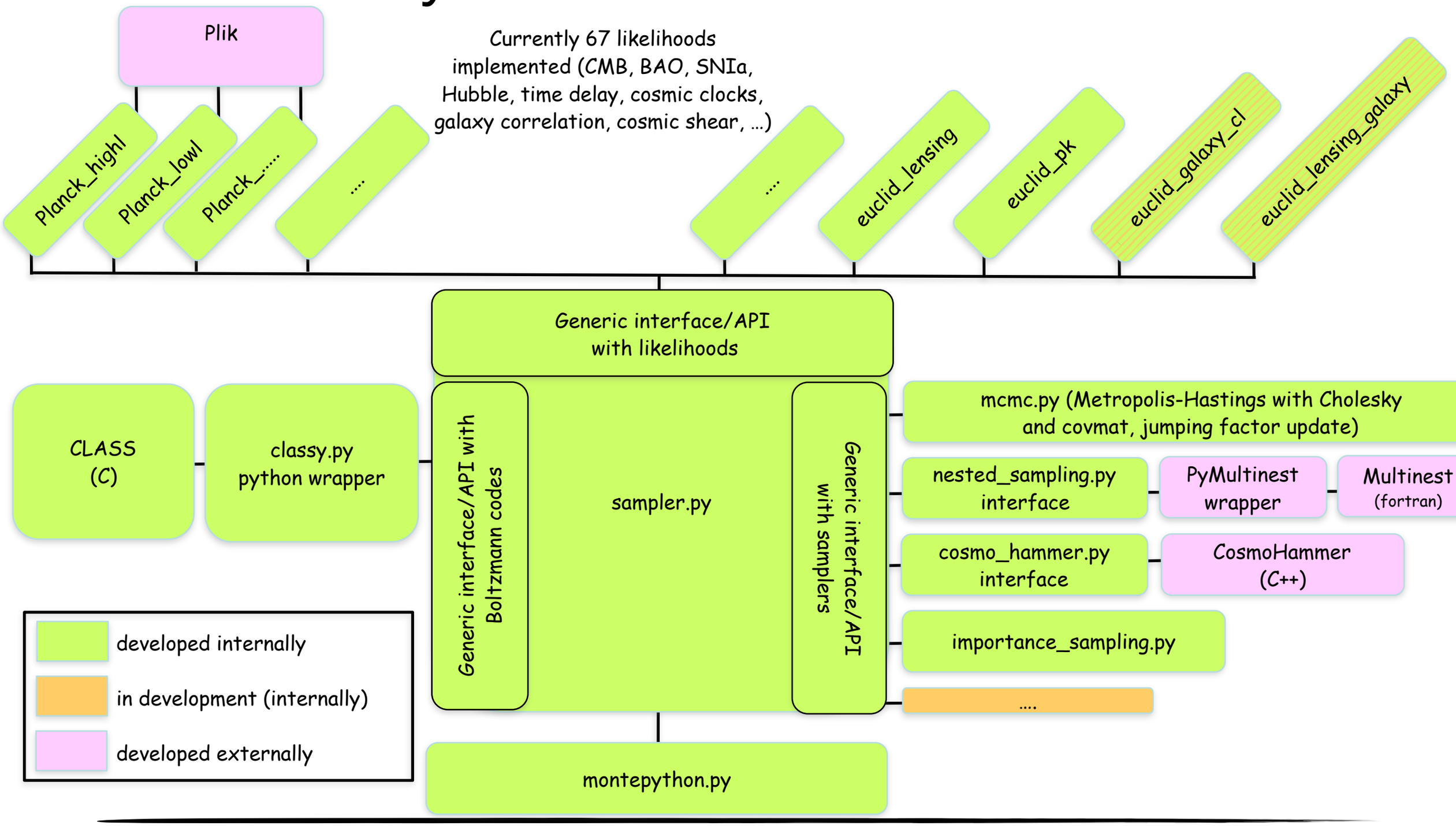
1. Superupdate (speed up & easier to converge)

2. Fisher matrix calculation (speed up & easier to converge)

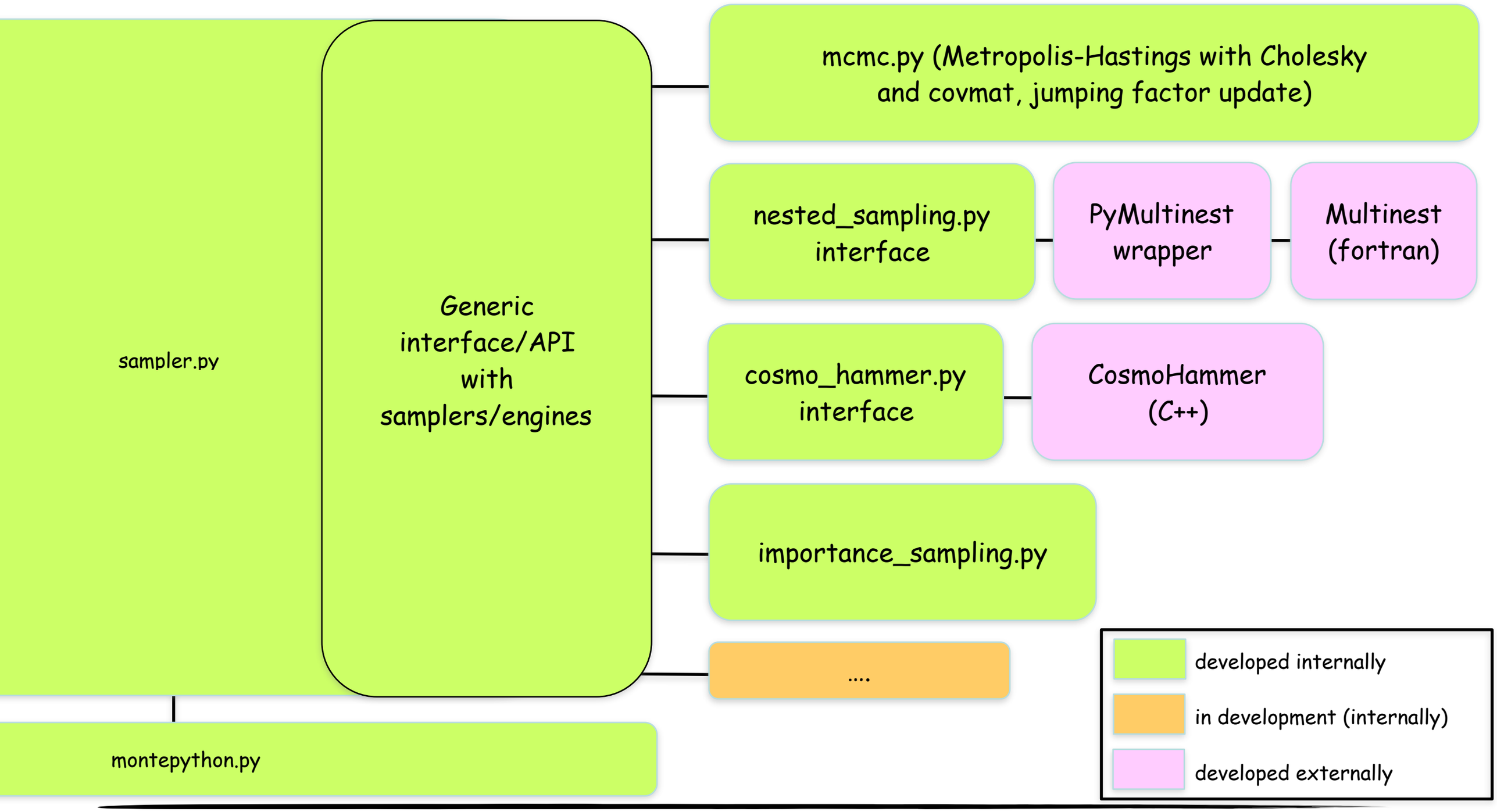
3. New likelihoods (more/new data, mock likelihoods)

4. Improved plotting (nice plots, easier to customize)

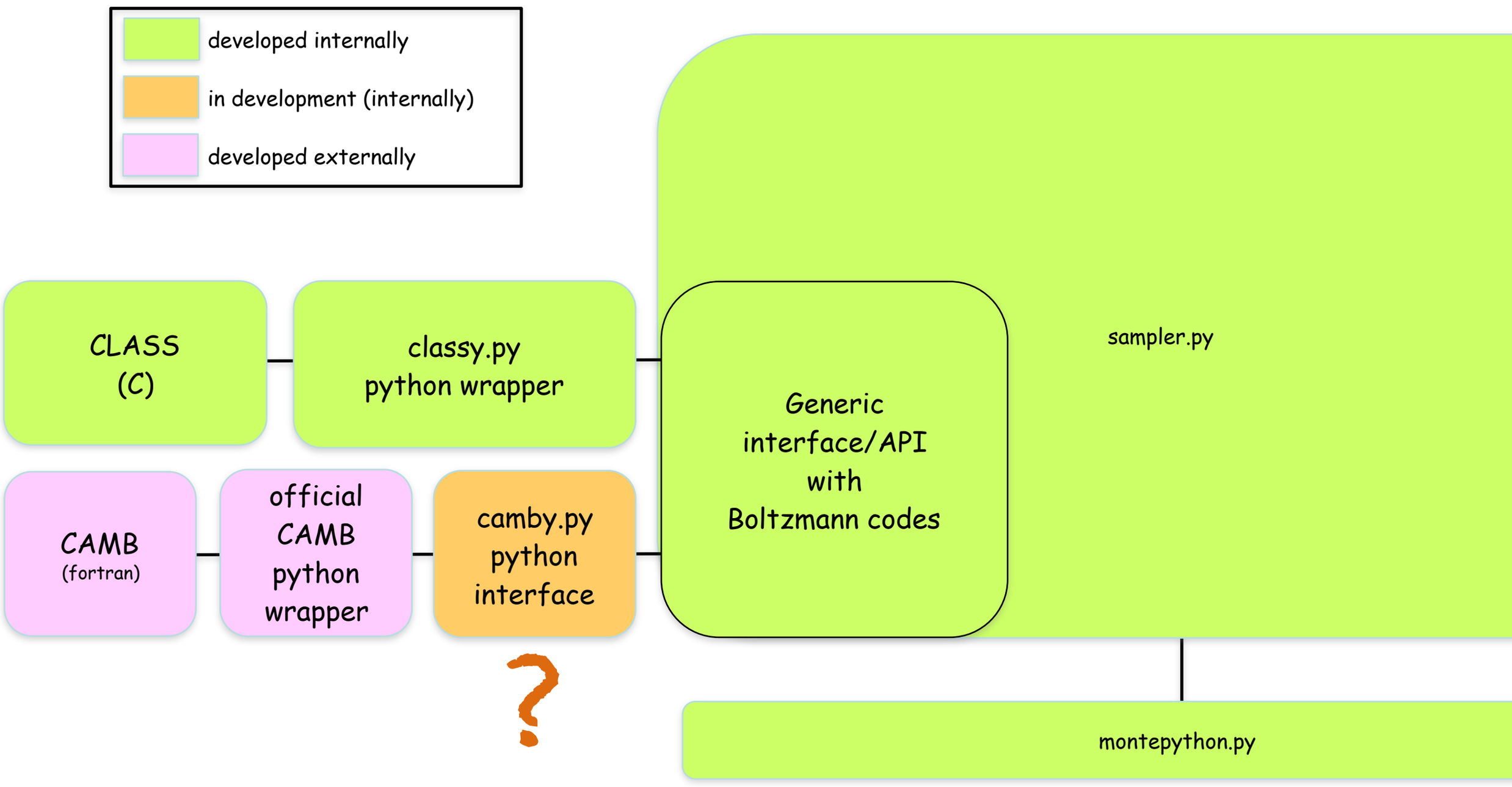
MontePython 3: modular structure



MontePython 3: modular structure

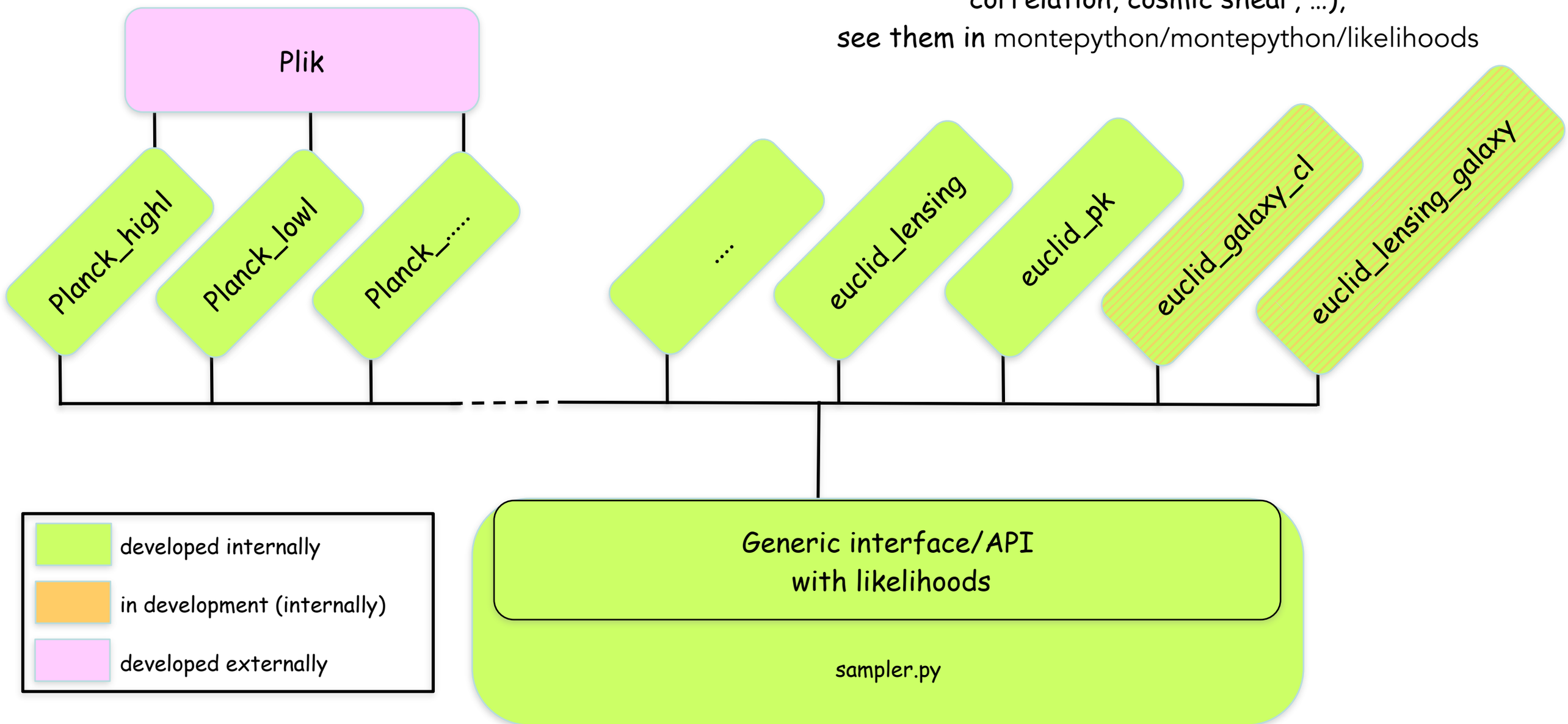


MontePython 3: modular structure



MontePython 3: modular structure

Currently 67 likelihoods implemented (CMB, BAO, SNIa, Hubble, time delay, cosmic clocks, galaxy correlation, cosmic shear, …),
see them in montepython/montepython/likelihoods



MontePython 3: installation

- Download & install **CLASS**
  - https://github.com/lesgourg/class_public
  - compile **CLASS** with wrapper (i.e. use "**make clean;make**" not "**make class**")

- Need **Python 2.7.x** with **numpy**, **scipy**, **matplotlib** and **cython**
  - **Python 3** users will need to install **Python 2.7** and use that version (including libraries!) in order to run **MontePython**. Be careful with pointers to **Python 3**, as they can cause problems.
  - compatibility with **Python 3** is planned
  - for **MPI** parallel runs also need **mpi4py**

- Download or clone **MontePython** from **github**
  - https://github.com/brinckmann/montepython_public

- Set up configuration file
  - **cp default.conf.template default.conf**
  - update paths

That's it!

MontePython 3: running the code

Required input

- `python montepython/MontePython.py run` plus …
  - `-p input/example.param` input parameter file
  - `-o chains/planck_run` output directory
  - `-N 10000` number of proposed steps for each chain

Basic options

- `-c covmat/base2015.covmat` use covariance matrix as proposal distribution
- `-b bestfit/base2015.bestfit` use a bestfit file as starting point for the run

Possible to launch parallel jobs manually (for Metropolis-Hastings) or using `MPI`

- `mpirun -np N montepython/MontePython.py run` …

To use multiple cores per chain first write (`CLASS` is parallelized to 8-32 cores)

- `OMP_NUM_THREADS=M`

This will create N number `MPI` processes each running on M number of cores

MontePython 3: param file

Understanding the `.param` file

```
# Experiments
data.experiments=['Planck_highl_lite','Planck_lowl','Planck_lensing']

# Settings for the over-sampling
data.over_sampling=[1, 4]

# Cosmological parameters list
data.parameters['o                                                    'cosmo']
data.parameters['o                                                    'cosmo']
data.parameters['1                                                    'cosmo']
data.parameters['l                                                    'cosmo']
data.parameters['n                                                    'cosmo']
data.parameters['t                                                    'cosmo']

# Nuisance parameter list, same call, except the name does not have to be a class name
data.parameters['A_planck']          = [100.028,   90,  110,  0.25,  0.01,'nuisance']

# Derived parameters
data.parameters['z_reio']           = [1, None, None, 0,     1,   'derived']
data.parameters['Omega_Lambda']     = [1, None, None, 0,     1,   'derived']

# Other cosmo parameters (fixed parameters, precision parameters, etc.)
data.cosmo_arguments['sBBN file'] = data.path['cosmo']+'/bbn/sBBN.dat'
data.cosmo_arguments['k_pivot'] = 0.05
```

List of experiments: must match names in likelihood folder

MontePython 3: param file

Understanding the .param file

Over-sampling of fast nuisance parameters

```
# Experiments
data.experiments=['Planck_highl_lite','Planck_lowl','Planck_lensing']

# Settings for the over-sampling
data.over_sampling=[1, 4]

# Cosmological parameters list
data.parameters['omega_b']          = [ 2.2253,    None, None,  0.028, 0.01, 'cosmo']
data.parameters['omega_cdm']        = [0.11919,    None, None, 0.0027,    1, 'cosmo']
data.parameters['100*theta_s']      = [ 1.0418,    None, None,   3e-4,    1, 'cosmo']
data.parameters['ln10^{10}A_s']     = [ 3.0973,    None, None,  0.029,    1, 'cosmo']
data.parameters['n_s']              = [0.96229,    None, None,  0.007,    1, 'cosmo']
data.parameters['tau_reio']         = [0.0463,     0.01, None,  0.013,    1, 'cosmo']

# Nuisance parameter list, same call, except the name does not have to be a class name
data.parameters['A_planck']         = [100.028,   90,  110,  0.25,  0.01,'nuisance']

# Derived parameters
data.parameters['z_reio']           = [1, None, None, 0,     1,   'derived']
data.parameters['Omega_Lambda']     = [1, None, None, 0,     1,   'derived']

# Other cosmo parameters (fixed parameters, precision parameters, etc.)
data.cosmo_arguments['sBBN file'] = data.path['cosmo']+'/bbn/sBBN.dat'
data.cosmo_arguments['k_pivot'] = 0.05
```

Cosmological parameters

Nuisance parameters are sampled 4x as much

MontePython 3: param file

Understanding the .param file

Parameter names must be CLASS compatible, as in explanatory.ini*

Lower and upper bound

Scaling

Mean

1-sigma

Type

```
# Settings for the over-sampling
data.over_sampling=[1, 4]

# Cosmological parameters list
data.parameters['omega_b']          = [ 2.2253,    None, None,  0.028,  0.01,  'cosmo']
data.parameters['omega_cdm']        = [0.11919,    None, None, 0.0027,     1,  'cosmo']
data.parameters['100*theta_s']      = [ 1.0418,    None, None,   3e-4,     1,  'cosmo']
data.parameters['ln10^{10}A_s']     = [ 3.0753,    None, None, 0.0029,     1,  'cosmo']
data.parameters['n_s']              = [0.96229,    None, None, 0.0074,     1,  'cosmo']
data.parameters['tau_reio']         = [0.09463,    0.04, None,  0.013,     1,  'cosmo']

# Nuisance parameter list, same call, except the name does not have to be a class name
data.parameters['A_planck']         = [1.0028,      90,  110,   0.25,  0.01,'nuisance']

# Derived parameters
data.parameters['z_reio']           = [1, None, None, 0,     1,   'derived']
data.parameters['Omega_Lambda']     = [1, None, None, 0,     1,   'derived']

# Other cosmo param
data.cosmo_argument
data.cosmo_arguments['k_pivot'] = 0.05
```

Cosmological parameters

MontePython 3: param file

Understanding the **.param** file

```
# Experiments
data.experiments=['Planck_highl_lite','Planck_lowl',                    'Planck_lensing']

# Settings for the over sampling
data.over_sampling=[1, 4]

# Cosmological parameters list
data.parameters['omega_cdm']          = [0.11919,    None, None,  0.028,  0.01, 'cosmo']
data.parameters['100*theta_s']        = [ 1.0418,    None, None,  0.0027,    1, 'cosmo']
data.parameters['ln10^{10}A_s']       = [ 3.0753,    None, None,  0.0029,    1, 'cosmo']
data.parameters['n_s']                = [0.96229,    None, None, 0.0074,     1, 'cosmo']
data.parameters['tau_reio']           = [0.09463,    0.04, None,  0.013,     1, 'cosmo']

# Nuisance parameter list, same call, except the name does not have to be a class name
data.parameters['A_planck']           = [100.028,   90,  110,  0.25,  0.01,'nuisance']

# Derived parameters
data.parameters['z_reio']             = [1, None,    None, 0,     1,   'derived']

# Other cosmo parameters (fixed parameters, precision parameters, etc.)
data.cosmo_arguments['sBBN file']     = data.path['cosmo']+'/bbn/sBBN.dat'
data.cosmo_arguments['k_pivot']       = 0.05
```

Nuisance parameters are only used by MontePython, i.e. are not passed to CLASS

Nuisance type

Nuisance parameters are often faster to vary

Over-sampling speeds up convergence

MontePython 3: param file

Understanding the `.param` file

```
# Experiments
data.experiments=['Planck_highl_lite','Planck_lowl','Planck_lensing']

# Settings for the over-sampling
data.over_sampling=[1, 4]

                                                    one, None,  0.028, 0.01, 'cosmo']
                                                    one, None, 0.0027,    1, 'cosmo']
                                                    one, None,   3e-4,    1, 'cosmo']
                                                    one, Non
                                                    one, Non
                                                    .04, Non

# Nuisance parameter li , same call, except the name does not have to be a class name
data.parameters['A_pl                = [100.028,   90,  110,  0.25,  0.0 ,'nuisance']

# Derived parameters
data.parameters['z_reio']           = [1, None, None, 0,     1,    'derived']
data.parameters['Omega_Lambda']     = [1, None, None, 0,     1,    'derived']
```

Derived parameters do not affect the run and are computed from the other parameters

Derived type

Derived parameters can be added in post-processing

MontePython 3: param file

Understanding the .param file

Arguments control e.g. numerical precision and fixed cosmological parameters

Cosmological arguments are fixed parameters passed to CLASS

```
# Experiments
data.experiments=['Planck_highl_lite','Planck_lowl','Planck_lensing']

# Settings for the over-sampling
data.over_sampling=[1, 4]

# Cosmological parameters list
data.parameters['omega_b']          = [ 2.2253,   None, None,  0.028, 0.01, 'cosmo']
data.parameters['omega_cdm']        = [0.11919,   None, None, 0.0027,    1, 'cosmo']
data.parameters['100*theta_s']      = [ 1.0418,   None, None,   3e-4,    1, 'cosmo']
data.parameters['ln10^10A_s']       = [ 3.075,    None, None, 0.0029,    1, 'cosmo']
data.parameters['n_s']              = [0.9629,    None, None, 0.0074,    1, 'cosmo']
data.parameters['tau_reio']         = [0.0943,    0.04, None,  0.013,    1, 'cosmo']

# Nuisance parameter list, same call, except the name does not have to be a class name
data.parameters['amp_100']          = [100.028,     90,  110,  0.25,  0.01,'nuisance']

# Derived parameters
data.parameters['z_reio']           = [1, None, None, 0,     1,   'derived']
data.parameters['Omega_Lambda']     = [1, None, None, 0,     1,   'derived']

# Other cosmo parameters (fixed parameters, precision parameters, etc.)
data.cosmo_arguments['sBBN file'] = data.path['cosmo']+'/bbn/sBBN.dat'
data.cosmo_arguments['k_pivot'] = 0.05
```

MontePython 3: analyzing output

Plotting received an overhaul in 3.0!

Analyze chains with **info**

- `python montepython/MontePython.py info chains/planck_run` plus …

Additional notable options

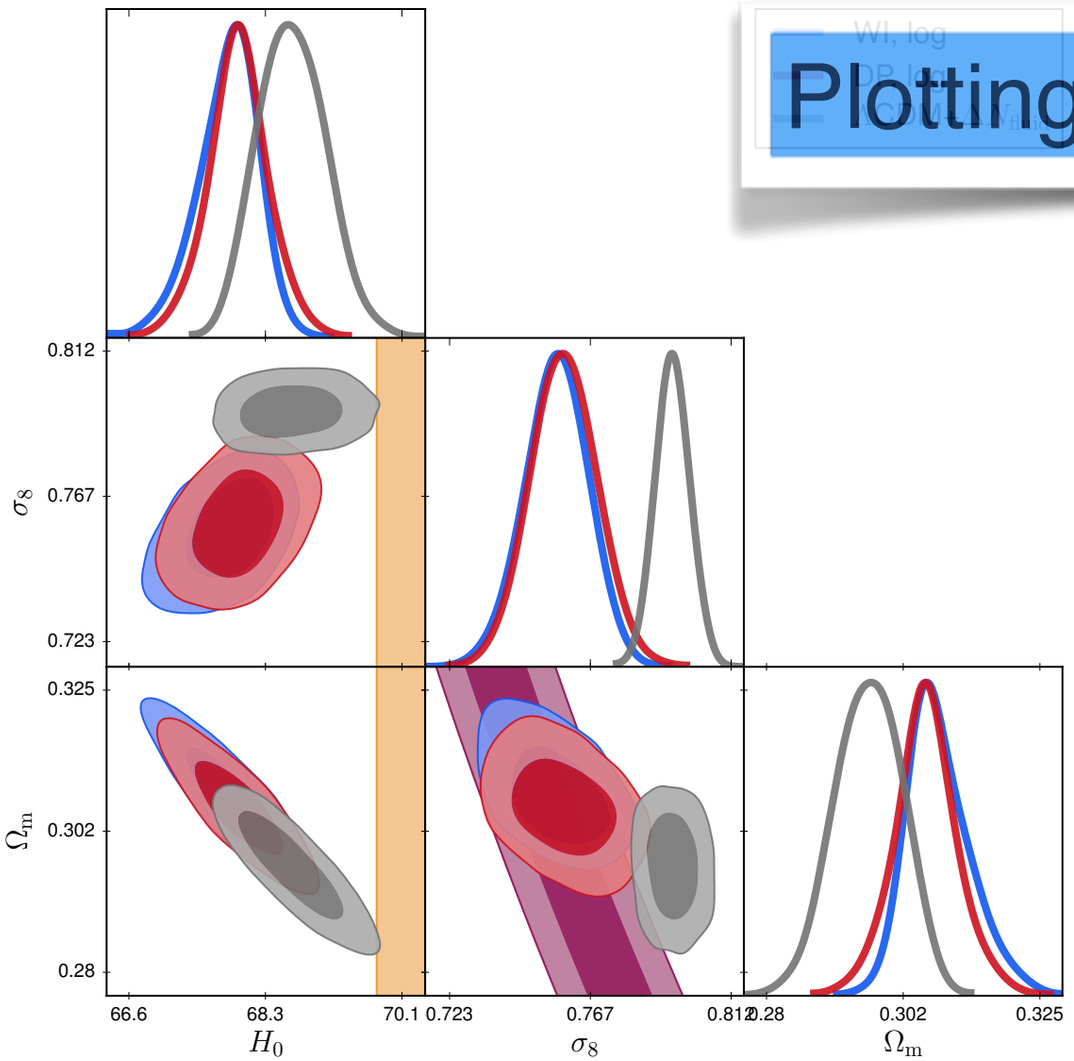
- `--keep-non-markovian` : keep the non-Markovian part of the chains [default: False].
- `--keep-fraction` : pass a decimal fraction, e.g. 0.8 to keep the last 80 % of the part of the chains that remain after the burn-in removal (note: redundant if non-Markovian points are discarded) [default: 1.0]
- `--want-covmat` : compute a covariance matrix based on the chains (note: this will overwrite the one produced by `--update`) [default: False]
- `--minimal` : use this flag to avoid computing posteriors, confidence limits and plots. The code just analyses the chains and outputs the files containing the convergence statistics, the best-fit parameters, and possibly the covariance matrix if `--want-covmat` is on [default: False]

intelligent analysis: *non-Markovian* points removed by default, no unnecessary removal of data

+ many more (see documentation and 3.0 release paper)!

Fans of GetDist can also use that, as the chains format is the same and a .paramnames file is provided

MontePython 3: plotting



Plotting received an overhaul in 3.0!

Improved contours, added control of e.g. colors, axes, legends,

Custom scripts of python code can be added

Created with **MontePython info** using

▶ **--extra example.plot** with two *customization scripts* passed in example.plot

```
python montepython/MontePython info dir1 dir2 dir3 --extra plot_files/ex.plot
```

MontePython 3: advanced usage

1. New features and advanced sampling options

2. Sampling new parameters in CLASS

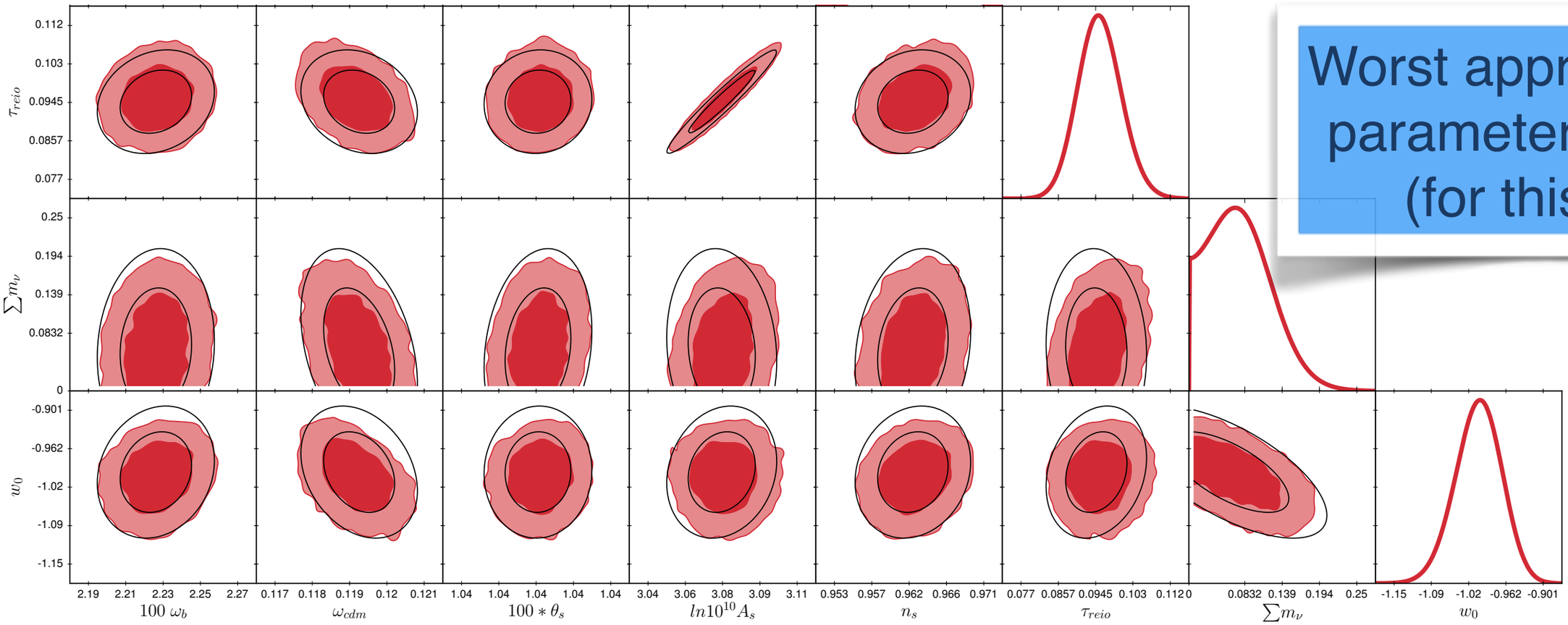
3. Communication with CLASS and adding new parameterizations

4. Adding likelihoods

MontePython 3: Fisher matrix

Compute Fisher matrices by adding

- `--method Fisher`
- Can use inverse as input covariance matrix for faster convergence



Worst approximated parameters shown (for this run)

Much better than a poor or no input covariance matrix!

MontePython 3: Fisher matrix

Important

Fisher matrix options

- `--method Fisher` : compute a Fisher matrix [default: MH]
- `--fisher-asymmetric` : allow for asymmetric steps (note: slows down computation) [default: False]
- `--fisher-step-it` : number of step iterations attempted [default: 10]
- `--fisher-delta` : target $\Delta \ln \mathcal{L}$ value for step iteration [default: 0.1]
- `--fisher-tol` : tolerance for $\Delta \ln \mathcal{L}$ (note: decreasing slows down computation) [default: 0.05]
- `--fisher-sym-lkl` : cut-off for switching to symmetric likelihood assumption in units of σ. Relevant when parameter space boundaries are close to the central value [default: 0.1]

Speed up convergence by first computing an inverse Fisher matrix to use as proposal distribution!

MontePython 3: superupdate

Boosting Metropolis-Hastings sampling

Since v2.2 (October 2015)

- `--update` U: update of covariance matrix, $\mathbf{C}$, every U cycles [default: $U = 50$]
- `--superupdate` SU: additionally, update of jumping factor, j, starting SU cycles after each covariance matrix update [default: $SU = 0$, meaning "deactivated"; recommended: 20]

New in v3.0

--update : Periodically updates the covariance matrix

- changes the shape and size of the proposal distribution
- particularly important with parameter degeneracies

--superupdate : adjusts the jumping factor

- i.e. re-scales the size of the proposal distribution
- mitigates ill effects of poor initial knowledge
- optimizes jumping factor to optimal acceptance rate (~25%)

MontePython 3: superupdate

- `--update` U: update of covariance matrix, $\mathbf{C}$, every U cycles [default: $U = 50$]
- `--superupdate` SU: additionally, update of jumping factor, j, starting SU cycles after each covariance matrix update [default: $SU = 0$, meaning "deactivated"; recommended: 20]

--superupdate : adjusts the jumping factor

$$c_k = c_{k-1} + \frac{1}{(k - k_{\text{update}})}(\overline{a.r.} - 0.26) \qquad |\overline{a.r.} - 0.26| < 0.01$$

Cosmologists usually aim for an a.r. of 25% since Dunkley et al. 2005

▶ achieved by above criteria because in many cases a.r. starts low

The jumping factor is directly related to the covariance matrix

▶ must update jumping factor after changing the covariance matrix

I.e., we want to keep the volume of the proposal density constant

$$j_{\text{after}} = j_{\text{before}} \left[\frac{\det(C_{\text{before}})}{\det(C_{\text{after}})} \right]^{\frac{1}{2N}}$$

MontePython 3: superupdate

Planck 2015 (highl TT, lowℓ, lensing) + BAO (MGS, 6dFGS, LOWZ, CMASS)

Running time: 12 hours

model	# param.	$R-1$: update	$R-1$: superupdate	$R-1$: superupdate + Fisher
ΛCDM	6	0.019	0.0098	0.029

2x

Planck 2015 (highl TTTEEE lite, lowℓ, lensing) + BAO (MGS, 6dFGS, LOWZ, CMASS) + galaxy clustering (SDSS DR7 LRG), weak lensing (CFHTLenS)

Running time: 12 hours

ΛCDM	6	0.042	0.032	0.018

Running time: 48 hours

ΛCDM	6	0.0062	0.0047	0.0038

superupdate: consistent performance boost

NB: Fisher method struggles without a good bestfit and/or with non-Gaussian parameters

Mock data: fake_planck_realistic, fake_desi_vol

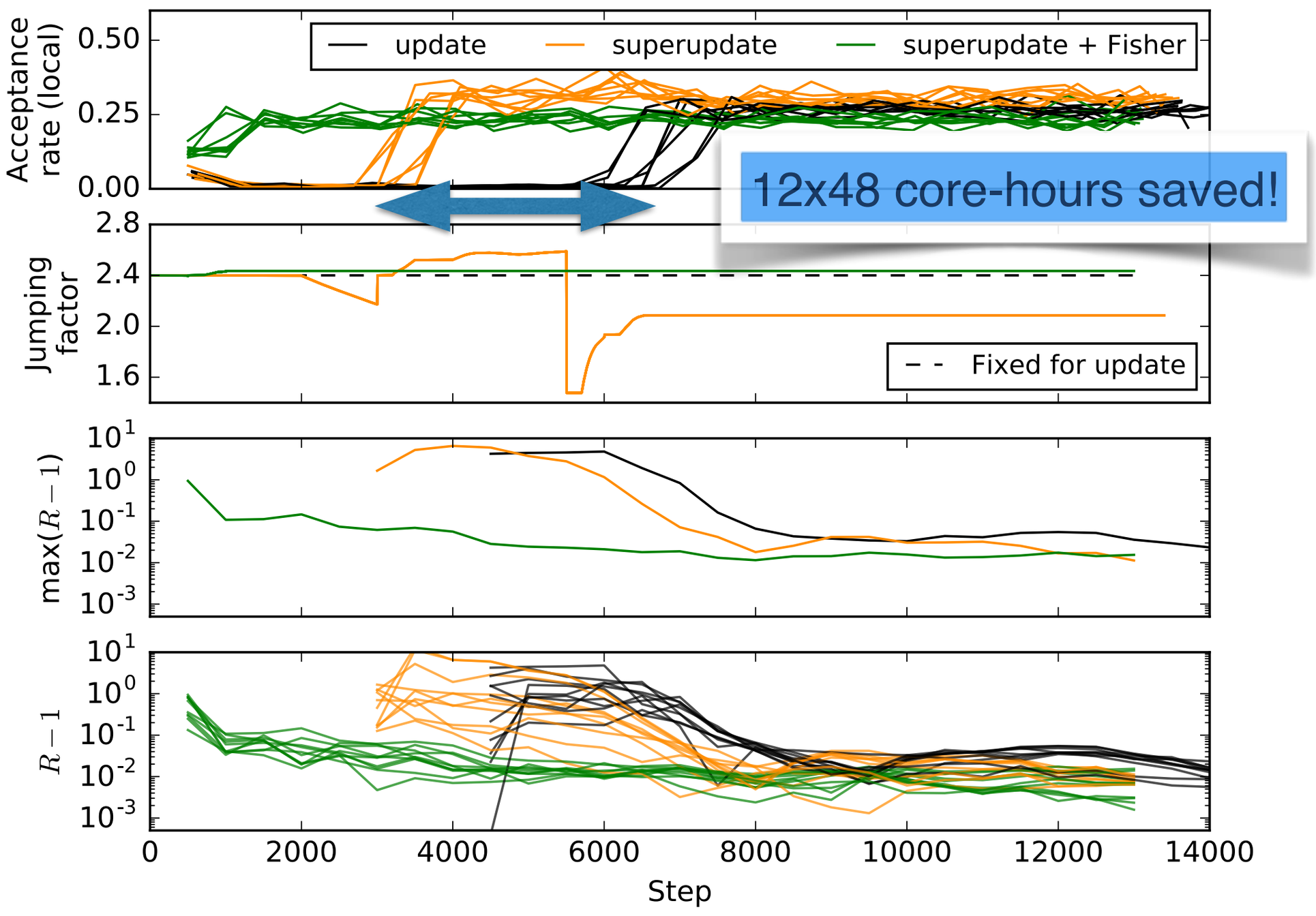
Running time: 12 hours

model	# param.	$R-1$: update	$R-1$: superupdate	$R-1$: superupdate + Fisher
ΛCDM	6	0.030	0.015	0.013 ← 3x
$+\sum m_\nu + w_0$	8	0.036	0.022	0.018
$+ N_{\rm eff}$ + running	10	not converged	not converged	0.040
$+ \Omega_k$	11	not converged	not converged	0.048 ← !
$+ w_a$	12	not converged	not converged	0.088

Running time: 48 hours

ΛCDM	6	0.0035	0.0029	0.0019
νw_0CDM + $N_{\rm eff}$ + running	10	0.014	0.0054	0.0038 ← 3x

MontePython 3: superupdate



MontePython 3: superupdate

Boosting Metropolis-Hastings sampling

Since v2.2 (October 2015)

- `--update` U: update of covariance matrix, $\mathbf{C}$, every U cycles [default: $U = 50$]
- `--superupdate` SU: additionally, update of jumping factor, j, starting SU cycles after each covariance matrix update [default: $SU = 0$, meaning "deactivated"; recommended: 20]

New in v3.0

Note: **superupdate** always enables **update**

Main advantages

- ▶ faster convergence
- ▶ better at dealing with difficult cases

Always neutral or better

MontePython 3: sampling options

Can also use MultiNest

Advanced options

- `--method` : sampling method (MH, NS, CH, IS, Der, Fisher) [default: MH]
  which refer respectively to Metropolis-Hastings, Nested Sampling (= MultiNest), Cosmo Hammer (= emcee), Importance Sampling, Derived (= reprocessing the chains to add columns with extra derived parameters requiring a new `CLASS` run for each model), and Fisher.

Options for Metropolis-Hastings and variants

Important

- `--method MH` : Metropolis-Hastings sampling [default: MH]
- `--update` : proposal distribution update frequency in number of cycles [default: 50]
- `--superupdate` : also adapt jumping factor. Adaptation delay in number of cycles [default: 0] (i.e. deactivated by default. Recommended: 20)
- `--superupdate-ar` : target local acceptance rate [default: 0.26]
- `--superupdate-ar-tol` : tolerance for local acceptance rate [default: 0.01]
- `--adaptive` : running adaptation of covariance matrix and jumping factor (note: only suitable for single chain runs) [default: 0]
- `--adaptive-ts` : starting step for adapting the jumping factor [default: 1000]
- `-f` : jumping factor [default: 2.4]

Speed up convergence of Metropolis-Hastings runs!

MontePython 3: sampling options

Can also use MultiNest

Advanced options

- `--method` : sampling method (MH, NS, CH, IS, Der, Fisher) [default: MH]
  which refer respectively to Metropolis-Hastings, Nested Sampling (= MultiNest), Cosmo Hammer (= emcee), Importance Sampling, Derived (= reprocessing the chains to add columns with extra derived parameters requiring a new `CLASS` run for each model), and Fisher.

More options

- `-T` : sample from the probability distribution $P^{1/T}$ instead of P [default: 1.0]
- `--minimize` : attempt to re-evaluate starting point using a χ^2 minimization algorithm [by default uses `SLSQP` via `numpy.optiminize.minimize()`, can be changed in `sampler.py` function `get_minimum()`]
- `--minimize-tol` : tolerance for minimization [default: 10^{-5}]
- **`--display-each-chi2`** : display the χ^2 contribution of each likelihood in a given point
- **`--quiet`** : reduce output
- **`--silent`** : silence all but essential output

E.g. use with bestfit file
--b <file> and no jump -f 0

Restarting chains

- **`-r <file>`** : restart chains from previous run. Must pass the lowest index chains file, e.g. 2018-09-12_10000__1.txt .

  MontePython will then create copies of all chains index 1 through N (number of MPI processes) with new names 2018-09-12_20000__1.txt etc. Once the chains have been copied the old chains can be moved to a backup folder or deleted. Note they will be automatically deleted at the completion of the run.

The old chains should not be included in the analysis

MontePython 3: likelihoods

Likelihoods can be found in the folder:

- <montepython_directory>/montepython/likelihoods/

New (18) or updated (3) likelihoods in MontePython 3.0

```
BK14
BK14priors
CFHTLens
CFHTLens_correlation
ISW
JLA
JLA_simple
Planck_SZ
Planck_actspt
Planck_highl
Planck_highl_TTTEEE
Planck_highl_TTTEEE_lite
Planck_highl_lite
Planck_lensing
Planck_lowl
WiggleZ
WiggleZ_bao
```

```
__init__.py
acbar
bao
bao_angular
bao_boss
bao_boss_aniso
bao_boss_aniso_gauss_approx
bao_boss_dr12
bao_fs_boss_dr12
bao_known_rs
bao_smallz_2014
bicep
bicep2
boomerang
cbi
clik_wmap_full
clik_wmap_lowl
```

```
core_m5
cosmic_clocks_2016
cosmic_clocks_BC03
cosmic_clocks_BC03_all
cosmic_clocks_MaStro
da_rec
euclid_lensing
euclid_pk
fake_desi
fake_desi_euclid_bao
fake_desi_vol
fake_planck_bluebook
fake_planck_realistic
gunn_peterson
hst
igm_temperature
kids450_qe_likelihood_public
```

```
litebird
lowlike
polarbear
quad
sdss_lrgDR4
sdss_lrgDR7
simlow
sn
spt
spt_2500
test_gaussian
test_nuisance1
test_nuisance2
timedelay
wmap
wmap_9yr
```

MontePython 3: likelihoods

Easy to do forecasts with MontePython!

Start by creating fiducial spectra (may need to delete existing one first)

- `python montepython/MontePython.py run -p <file> -o <directory> -f 0`

No jump

Then run as normal!

More coming soon!

E.g. CMB-S4, PICO, SKA and updated euclid likelihoods

```
BK14
BK14priors
CFHTLens
CFHTLens_correlation
ISW
JLA
JLA_simple
Planck_SZ
Planck_actspt
Planck_highl
Planck_highl_TTTEEE
Planck_highl_TTTEEE_lite
Planck_highl_lite
Planck_lensing
Planck_lowl
WiggleZ
WiggleZ_bao
```

```
__init__.py
acbar
bao
bao_angular
bao_boss
bao_boss_aniso
bao_boss_aniso_gauss_approx
bao_boss_dr12
bao_boss_dr12_2014
bao_known_rs
bicep
bicep2
boomerang
cbi
clik_wmap_full
clik_wmap_lowl
```

```
core_m5
cosmic_clocks_2016
cosmic_clocks_BC03
cosmic_clocks_BC03_all
cosmic_clocks_MaStro
da_rec
euclid_lensing
euclid_pk
fake_desi
fake_desi_euclid_bao
fake_desi_vol
fake_planck_bluebook
fake_planck_realistic
gunn_peterson
hst
igm_temperature
kids450_qe_likelihood_public
```

```
litebird
lowlike
polarbear
quad
sdss_lrgDR4
sdss_lrgDR7
simlow
sn
spt
spt_2500
test_gaussian
test_nuisance1
test_nuisance2
timedelay
wmap
wmap_9yr
```

MontePython 3: communication with CLASS

MontePython interacts with **CLASS** through the wrapper and automatically accepts **any** parameter that is implemented in **CLASS**.

Example: you have implemented a model in **CLASS** with parameters **my_parameter1** and **my_parameter2**, as well as precision parameters **my_precision1** and **my_precision2**.

These parameters can be directly added to the **.param** file.

```
# Cosmological parameters list

data.parameters['omega_b']      = [ 2.2253,   None, None,  0.028, 0.01, 'cosmo']
data.parameters['omega_cdm']    = [0.11919,   None, None, 0.0027,    1, 'cosmo']
data.parameters['100*theta_s']  = [ 1.0418,   None, None,   3e-4,    1, 'cosmo']
data.parameters['ln10^{10}A_s'] = [ 3.0753,   None, None, 0.0029,    1, 'cosmo']
data.parameters['n_s']          = [0.96229,   None, None, 0.0074,    1, 'cosmo']
data.parameters['tau_reio']     = [0.09463,   0.04, None,  0.013,    1, 'cosmo']
data.parameters['my_parameter1'] = [     0.,    -1.,   1.,    0.1,    1, 'cosmo']
data.parameters['my_parameter2'] = [     1.,   None,  10.,    0.2,    1, 'cosmo']
```

MontePython 3: communication with CLASS

These parameters can be directly added to the **.param** file.

```
# Cosmological parameters list

data.parameters['omega_b']      = [ 2.2253,   None, None,  0.028, 0.01, 'cosmo']
data.parameters['omega_cdm']    = [0.11919,   None, None, 0.0027,    1, 'cosmo']
data.parameters['100*theta_s']  = [ 1.0418,   None, None,   3e-4,    1, 'cosmo']
data.parameters['ln10^{10}A_s'] = [ 3.0753,   None, None, 0.0029,    1, 'cosmo']
data.parameters['n_s']          = [0.96229,   None, None, 0.0074,    1, 'cosmo']
data.parameters['tau_reio']     = [0.09463,   0.04, None,  0.013,    1, 'cosmo']
data.parameters['my_parameter1'] = [     0.,    -1.,   1.,    0.1,    1, 'cosmo']
data.parameters['my_parameter2'] = [     1.,   None,  10.,    0.2,    1, 'cosmo']

# Other cosmo parameters (fixed parameters, precision parameters, etc.)

data.cosmo_arguments['sBBN file'] = data.path['cosmo']+'/bbn/sBBN.dat'
data.cosmo_arguments['k_pivot'] = 0.05
data.cosmo_arguments['my_precision1'] = 0.005
data.cosmo_arguments['my_precision2'] = 0.02
```

That’s it!

MontePython 3: communication with CLASS

* We can add a custom parameterization for our new parameters. To do this, we have to modify

▶ <montepython_directory>/montepython/data.py

```
def update_cosmo_arguments(self):
    """
    Put in :attr:`cosmo_arguments` the current values of
    :attr:`mcmc_parameters`

    This method is called at every step in the Markov chain, to update the
    dictionary. In the Markov chain, the scale is not remembered, so one
    has to apply it before giving it to the cosmological code.

    .. note::

        When you want to define new parameters in the Markov chain that do
        not have a one to one correspondance to a cosmological name, you
        can redefine its behaviour here. You will find in the source
        several such examples.
...
```

```
elif elem == 'my_parameterization':
    try:
        self.cosmo_arguments['my_parameter1']
    except:
        warnings.warn('my_parameter1 not defined. This quantity must be definde to sample my_parameterization')
    self.cosmo_arguments['my_parameter2'] = self.cosmo_arguments[elem] * self.cosmo_arguments['my_parameter1']**2
```

We have now redefined **my_parameter2** as a function of **my_parameter1** and **my_parameterization**

MontePython 3: communication with CLASS

The new parameterization can be directly added to the **.param** file.

```
# Cosmological parameters list

data.parameters['omega_b']              = [ 2.2253,   None, None,  0.028, 0.01, 'cosmo']
data.parameters['omega_cdm']            = [0.11919,   None, None, 0.0027,    1, 'cosmo']
data.parameters['100*theta_s']          = [ 1.0418,   None, None,   3e-4,    1, 'cosmo']
data.parameters['ln10^{10}A_s']         = [ 3.0753,   None, None, 0.0029,    1, 'cosmo']
data.parameters['n_s']                  = [0.96229,   None, None, 0.0074,    1, 'cosmo']
data.parameters['tau_reio']             = [0.09463,   0.04, None,  0.013,    1, 'cosmo']
data.parameters['my_parameter1']        = [     0.,    -1.,   1.,    0.1,    1, 'cosmo']
data.parameters['my_parameterization'] = [     1.,   None,  10.,    0.2,    1, 'cosmo']

# Other cosmo parameters (fixed parameters, precision parameters, etc.)

data.cosmo_arguments['sBBN file'] = data.path['cosmo']+'/bbn/sBBN.dat'
data.cosmo_arguments['k_pivot'] = 0.05
data.cosmo_arguments['my_precision1'] = 0.005
data.cosmo_arguments['my_precision2'] = 0.02
```

That’s it!

MontePython 3: communication with CLASS

Can call any function in the wrapper:

```
class my_likelihood(Likelihood_prior):

    # initialisation of the class is done within the parent Likelihood_prior. For
    # this case, it does not differ, actually, from the __init__ method in
    # Likelihood class.
    def loglkl(self, cosmo, data):

        tau = cosmo.tau_reio()
        loglkl = -0.5 * (tau - self.tau) ** 2 / (self.sigma ** 2)
        return loglkl

class my_likelihood(Likelihood_prior):

    # initialisation of the class is done within the parent Likelihood_prior. For
    # this case, it does not differ, actually, from the __init__ method in
    # Likelihood class.
    def loglkl(self, cosmo, data):

        rs_drag_theo = cosmo.rs_drag()
        loglkl = -0.5 * (rs_drag_theo - self.rs_drag) ** 2 / (self.sigma ** 2)
        return loglkl
```

MontePython 3: adding likelihoods

1. Create folder, e.g. starting from a similar likelihood

- `cp -r simlow` **`my_likelihood`**

The folder contains two files

- `__init__.py  simlow.data`

2. Rename .data file to match folder name, i.e. **`my_likelihood`**

- `__init__.py  my_likelihood.data`

MontePython 3: adding likelihoods

3. Rename class in **__init__.py** to match folder name

```
import os
from montepython.likelihood_class import Likelihood_prior


class simlow(Likelihood_prior):

    # initialisation of the class is done within the parent Likelihood_prior. For
    # this case, it does not differ, actually, from the __init__ method in
    # Likelihood class.
    def loglkl(self, cosmo, data):
```

```
import os
from montepython.likelihood_class import Likelihood_prior


class my_likelihood(Likelihood_prior):

    # initialisation of the class is done within the parent Likelihood_prior. For
    # this case, it does not differ, actually, from the __init__ method in
    # Likelihood class.
    def loglkl(self, cosmo, data):
```

MontePython 3: adding likelihoods

4. Rename class variables in **my_likelihood.data**

```
# Values for tau (following Astro-ph/1605.02985)
# This likelihood is a prior on tau_reio inspired
# by simlow, the low multipole polarization
# likelihood from the Planck 2016 paper
simlow.tau    = 0.055
simlow.sigma  = 0.009
```

```
# Values for tau (following Astro-ph/1605.02985)
# This likelihood is a prior on tau_reio inspired
# by simlow, the low multipole polarization
# likelihood from the Planck 2016 paper
my_likelihood.tau    = 0.055
my_likelihood.sigma  = 0.009
```

MontePython 3: adding likelihoods

Perhaps we want a prior on the sound horizon at drag epoch

5. Modify **my_likelihood.data** as necessary, e.g. in our example

```
# Values for tau (following Astro-ph/1605.02985)
# This likelihood is a prior on tau_reio inspired
# by simlow, the low multipole polarization
# likelihood from the Planck 2016 paper
my_likelihood.tau    = 0.055
my_likelihood.sigma  = 0.009
```

```
# Planck 2015 prior for the sound horizon at drag epoch
my_likelihood.rs_drag = 147.41
my_likelihood.sigma   = 0.30
```

MontePython 3: adding likelihoods

6. Modify **__init__.py** as necessary, e.g. in our example

```
class my_likelihood(Likelihood_prior):

    # initialisation of the class is done within the parent Likelihood_prior. For
    # this case, it does not differ, actually, from the __init__ method in
    # Likelihood class.
    def loglkl(self, cosmo, data):

        tau = cosmo.tau_reio()
        loglkl = -0.5 * (tau - self.tau) ** 2 / (self.sigma ** 2)
        return loglkl
```

```
class my_likelihood(Likelihood_prior):

    # initialisation of the class is done within the parent Likelihood_prior. For
    # this case, it does not differ, actually, from the __init__ method in
    # Likelihood class.
    def loglkl(self, cosmo, data):

        rs_drag_theo = cosmo.rs_drag()
        loglkl = -0.5 * (rs_drag_theo - self.rs_drag) ** 2 / (self.sigma ** 2)
        return loglkl
```

MontePython 3: more information

MontePython 3 paper
- Brinckmann & Lesgourgues 1804.07261

Official documentation
- http://monte-python.readthedocs.io/en/latest/

Help files
- `python montepython/MontePython.py run -h` (more details: `--help`)
- `python montepython/MontePython.py info -h` (more details: `--help`)

Github readme
https://github.com/brinckmann/montepython_public

Wiki
https://github.com/baudren/montepython_public/wiki

Previous talks and tutorials on Julien Lesgourgues' website
https://lesgourg.github.io/courses.html

Problems? Open a ticket on my github page (after trying to find a solution yourself)
https://github.com/brinckmann/montepython_public