# Monte Python: basic runs

Benjamin Audren

Institute of Theoretical Physics
École Polytechnique Fédérale de Lausanne

28/10/2014

# Outline

# Outline

# Goals of Monte Python

## Wish list

- **Interfacing** with CLASS (C), other likelihoods

# Goals of Monte Python

## Wish list

- **Interfacing** with CLASS (C), other likelihoods
- **Modular design** (use other Boltzmann codes, algorithms)

# Goals of Monte Python

## Wish list

- **Interfacing** with CLASS (C), other likelihoods
- **Modular design** (use other Boltzmann codes, algorithms)
- **Readability** (a code is read more than it is written)

# Goals of Monte Python

## Wish list

- **Interfacing** with CLASS (C), other likelihoods
- **Modular design** (use other Boltzmann codes, algorithms)
- **Readability** (a code is read more than it is written)
- **Manipulating files** - read, write, renaming

# Goals of Monte Python

## Wish list

- **Interfacing** with CLASS (C), other likelihoods
- **Modular design** (use other Boltzmann codes, algorithms)
- **Readability** (a code is read more than it is written)
- **Manipulating files** - read, write, renaming
- Using **Existing Library** (for complex computations)

# Goals of Monte Python

## Wish list

- **Interfacing** with CLASS (C), other likelihoods
- **Modular design** (use other Boltzmann codes, algorithms)
- **Readability** (a code is read more than it is written)
- **Manipulating files** - read, write, renaming
- Using **Existing Library** (for complex computations)
- **Easy to learn, easy to use**

# Goals

# Goals

## ~~Wish~~ Tick list

- ✓ **Interfacing** with CLASS (C), other likelihoods
- ✓ **Modular design** (use other Boltzmann codes, algorithms)
- ✓ **Readability** (a code is read more than it is written)
- ✓ **Manipulating files** - read, write, renaming
- ✓ Using **Existing Library** (for complex computations)
- ✓ **Easy to learn, easy to use**

# Design policy of Monte Python

## Guidelines

- **Modular structure**: separate I/O, parser, data structures, sampler, interface with the cosmological module, plotting: **Thursday**

# Design policy of Monte Python

## Guidelines

- **Modular structure**: separate I/O, parser, data structures, sampler, interface with the cosmological module, plotting: **Thursday**
- **Remembering a run**: you must be able to reproduce a run done some time ago (version control, folder based)

# Design policy of Monte Python

## Guidelines

- **Modular structure**: separate I/O, parser, data structures, sampler, interface with the cosmological module, plotting: **Thursday**
- **Remembering a run**: you must be able to reproduce a run done some time ago (version control, folder based)
- **Intelligent communication with** CLASS: if CLASS knows how to do it, you can ask for it.

# Design policy of Monte Python

## Guidelines

- **Modular structure**: separate I/O, parser, data structures, sampler, interface with the cosmological module, plotting: **Thursday**
- **Remembering a run**: you must be able to reproduce a run done some time ago (version control, folder based)
- **Intelligent communication with** CLASS: if CLASS knows how to do it, you can ask for it.
- **Convenient Plotting**: since a folder will be self contained, with all the information, producing a plot out of this folder should be easy.

# Design policy of Monte Python

## Guidelines

- **Modular structure**: separate I/O, parser, data structures, sampler, interface with the cosmological module, plotting: **Thursday**
- **Remembering a run**: you must be able to reproduce a run done some time ago (version control, folder based)
- **Intelligent communication with** CLASS: if CLASS knows how to do it, you can ask for it.
- **Convenient Plotting**: since a folder will be self contained, with all the information, producing a plot out of this folder should be easy.
- **Using mock data**: there must be a way to handle mock data easily.

# Conclusion on design

**You tell me !**

We'll see in the coming days if it works !

# Outline

# Installation

## Compile the wrapper

```
cd class; make;
cd python; python setup.py install --user
```

## Download

`http://montepython.net` or
`https://github.com/baudren/montepython_public/releases`

## Unzip

```
bunzip2 montepython_v2.1.0.tar.bz2
tar -xvf montepython_v2.1.0.tar
```

## Configure

```
cp default.conf.template default.conf
```

edit it to match your path

# Two modes

**Run**

```
python montepython/MontePython.py run -h/--help
```

**Info**

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

# Doing a run is choosing:

**a set of experiments, . . .**

Planck, WiggleZ, BAO,. . . which might require **nuisance parameters**

**sampled parameters, . . .**

$\Omega_m$, $H_0$, $A_s$, $r$,. . .

**an input covariance matrix**

From a previous run

**and some derived parameters.**

$\sigma_8$, $log(10^{10}A_s)$, . . .

And an output folder

# Parser

## Common arguments

`python montepython/MontePython.py run` plus...

- *A minima*: `-o chains/planck -p example.param`

# Parser

## Common arguments

`python montepython/MontePython.py run` plus...

- *A minima*: `-o chains/planck -p example.param`
- `-N 1000` **Number of proposed steps**

# Parser

## Common arguments

`python montepython/MontePython.py run` plus...

- *A minima*: `-o chains/planck -p example.param`
- `-N 1000`
- `-c covmat/old.covmat` **better proposal density**

# Parser

## Common arguments

`python montepython/MontePython.py run` plus...

- *A minima*: `-o chains/planck -p example.param`
- `-N 1000`
- `-c covmat/old.covmat`
- `-b bestfit/old.bestfit` **best-fit to start the chain**

# Input Parameter File

```
data.experiments=['Planck_highl','Planck_lowl','lowlike']
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['H0']        = [67.802,  None,None, 1.2,       1, 'cosmo']
data.parameters['A_s']       = [2.2177,  None,None, 0.055, 1.e-9, 'cosmo']
data.parameters['n_s']       = [0.96229, None,None, 0.0074,    1, 'cosmo']
data.parameters['tau_reio']  = [0.09463, None,None, 0.013,     1, 'cosmo']

# Nuisance parameter list, same call, except the name does not have to be a class name
data.parameters['A_ps_100']  = [145.83,    0,None,     61, 1, 'nuisance']
data.parameters['A_ps_143']  = [49.578,    0,None,     14, 1, 'nuisance']
```

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

```
data.cosmo_arguments['N_eff'] = 2.03351
data.cosmo_arguments['N_ncdm'] = 1
```

# Input Parameter File

```
data.experiments=['Planck_highl','Planck_lowl','lowlike']
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['H0']         = [67.802,  None,None, 1.2,       1, 'cosmo']
data.parameters['A_s']        = [2.2177,  None,None, 0.055, 1.e-9, 'cosmo']
data.parameters['n_s']        = [0.96229, None,None, 0.0074,    1, 'cosmo']
data.parameters['tau_reio']   = [0.09463, None,None, 0.013,     1, 'cosmo']

# Nuisance parameter list, same call, does not have to be a class name
data.parameters['A_ps_100']   = [145.83,     0,None,      61, 1, 'nuisance']
data.parameters['A_ps_143']   = [49.578,     0,None,      14, 1, 'nuisance']

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

data.cosmo_arguments['N_eff'] = 2.03351
data.cosmo_arguments['N_ncdm'] = 1
```

Mean values

# Input Parameter File

```
data.experiments=['Planck_highl','Planck_lowl','lowlike']
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['H0']         = [67.802,  None,None,  1.2,        1, 'cosmo']
data.parameters['A_s']        = [2.2177,  None,None,  0.055, 1.e-9, 'cosmo']
data.parameters['n_s']        = [0.96229  None,None,  0.0074,     1, 'cosmo']
data.parameters['tau_reio']   = [0.09463  None,None,  0.013,      1, 'cosmo']

# Nuisance parameter list, same call, excep                    o be a class name
data.parameters['A_ps_100']   = [145.83,                                ce']
data.parameters['A_ps_143']   = [49.578,     0,None,      14, 1, 'nuisance']
```

Lower-Upper bounds

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

data.cosmo_arguments['N_eff'] = 2.03351
data.cosmo_arguments['N_ncdm'] = 1
```

# Input Parameter File

```
data.experiments=['Planck_highl','Planck_lowl','lowlike']
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['H0']         = [67.802,  None,None, 1.2,       1, 'cosmo']
data.parameters['A_s']        = [2.2177,  None,None, 0.055, 1.e-9, 'cosmo']
data.parameters['n_s']        = [0.96229, None,None, 0.0074,    1, 'cosmo']
data.parameters['tau_reio']   = [0.09463, None,None, 0.013,     1, 'cosmo']

# Nuisance parameter list, same call, except the name does n
data.parameters['A_ps_100']   = [145.83,    0,None,     61, 1
data.parameters['A_ps_143']   = [49.578,    0,None,     14, 1, 'nuisance']
```

$1\sigma$ proposal

class name

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

data.cosmo_arguments['N_eff'] = 2.03351
data.cosmo_arguments['N_ncdm'] = 1
```

# Input Parameter File

```
data.experiments=['Planck_highl','Planck_lowl','lowlike']
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['H0']        = [67.802,  None,None, 1.2,       1, 'cosmo']
data.parameters['A_s']       = [2.2177,  None,None, 0.055, 1.e-9, 'cosmo']
data.parameters['n_s']       = [0.96229, None,None, 0.0074,    1, 'cosmo']
data.parameters['tau_reio']  = [0.09463, None,None, 0.013,     1, 'cosmo']

# Nuisance parameter list, same call, except the name does not have a class name
data.parameters['A_ps_100']  = [145.83,    0,None,     61, 1, 'nuisance']
data.parameters['A_ps_143']  = [49.578,    0,None,     14, 1, 'nuisance']
```

scale

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

data.cosmo_arguments['N_eff'] = 2.03351
data.cosmo_arguments['N_ncdm'] = 1
```

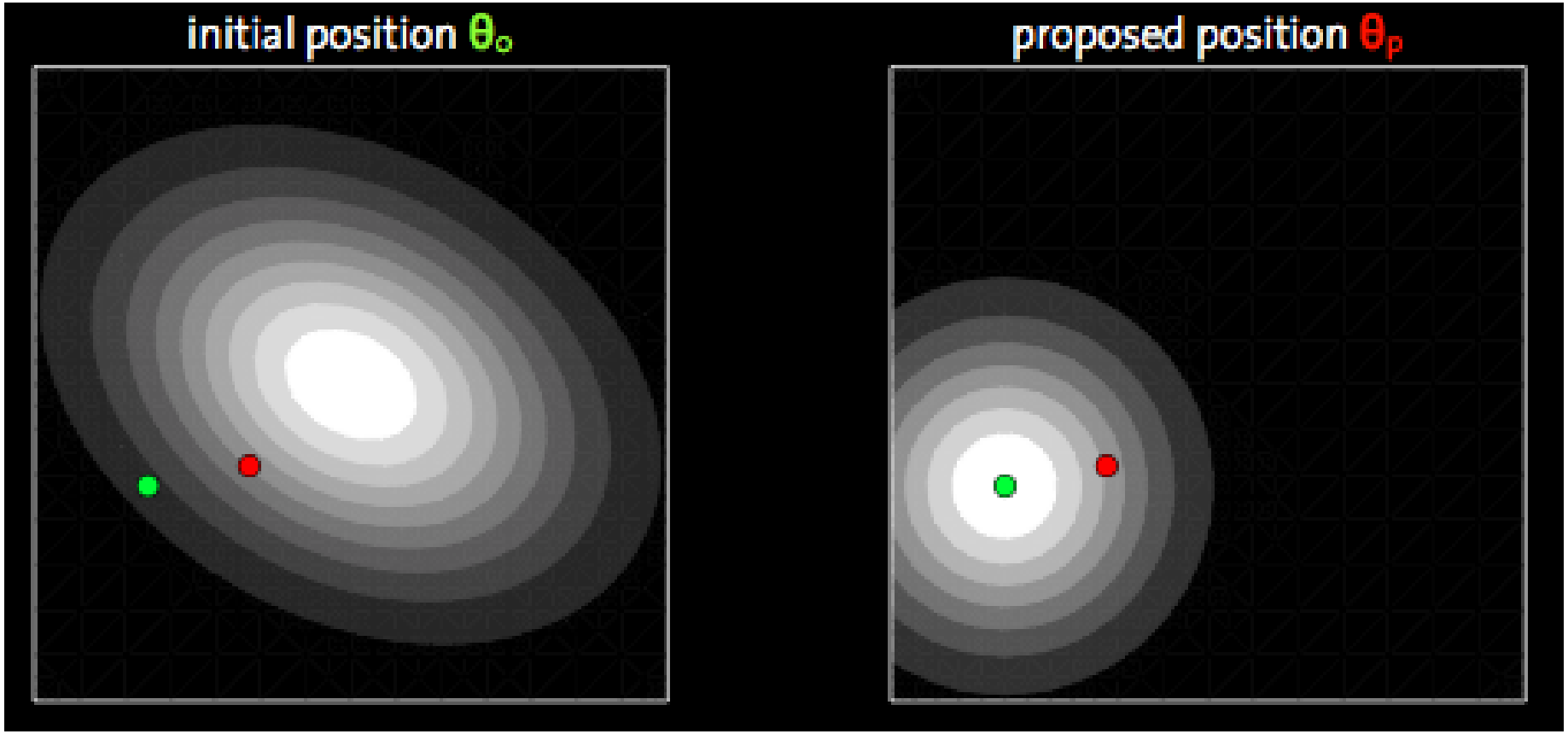
# Input Parameter File

```
data.experiments=['Planck_highl','Planck_lowl','lowlike']
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['H0']         = [67.802,  None,None, 1.2,       1, 'cosmo']
data.parameters['A_s']        = [2.2177,  None,None, 0.055, 1.e-9, 'cosmo']
data.parameters['n_s']        = [0.96229, None,None, 0.0074,    1, 'cosmo']
data.parameters['tau_reio']   = [0.09463, None,None, 0.013,     1, 'cosmo']

# Nuisance parameter list, same call, except the name does not have to be a type name
data.parameters['A_ps_100']   = [145.83,    0,None,     61, 1, 'nuisance']
data.parameters['A_ps_143']   = [49.578,    0,None,     14, 1, 'nuisance']
```

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

```
data.cosmo_arguments['N_eff'] = 2.03351
data.cosmo_arguments['N_ncdm'] = 1
```

# Reminder from lecture 1



# Types of parameter

## cosmological

- **known to Class (can define tricks)**
- **must be compatible (as in `explanatory.ini`)**

# Types of parameter

## cosmological

- known to Class (can define tricks)
- must be compatible (as in `explanatory.ini`)

## derived

- **known to Class**
- **can be redundant**

# Types of parameter

**cosmological**

- known to Class (can define tricks)
- must be compatible (as in `explanatory.ini`)

**derived**

- known to Class
- can be redundant

**nuisance**

- **As the name indicates...**
-

# Types of parameter

## cosmological

- known to Class (can define tricks)
- must be compatible (as in `explanatory.ini`)

## derived

- known to Class
- can be redundant

## nuisance

- As the name indicates. . .
- **Needed pain**

# Warning!

## Input parameter file and log.param

- When a folder is **created**, a file `log.param` is written, copying information from param and likelihoods.
- When a new chain is launched, **the input file is not read any more**, only the `log.param`

# Configuration File

```
root = '/Users/benjaminaudren/Desktop/professional/codes'

path['cosmo']    = root+'/class/'
```

# Usage: summary

## After installation

- `cp default.conf.template default.conf` and edit
- `python montepython/MontePython.py -o chains/test -p example.param`

# Running in parallel

## Installing mpi

```
apt-get install python-mpi4py
```

## Using mpi

```
mpirun -np N python montepython/Montepython run...
```

# What Monte Python does for you

## Convenience

- No need to choose a **name** for your chain

# What Monte Python does for you

## Convenience

- No need to choose a **name** for your chain
- Can use a covariance matrix with **partial information**

# What Monte Python does for you

## Convenience

- No need to choose a **name** for your chain
- Can use a covariance matrix with **partial information**
- Can analyze a subset of chains

# Running strategies

## Starting

- Choosing **experiments** to combine - **careful with that**
- Varying **parameters**, proposal distribution
- After M ($\simeq 10$) chains of N ($\simeq 10000$) points, **analyze**

## Analyzing (after the break)

```
python montepython/MontePython.py info folder
```

## Main

- Feed the new found **covariance matrix** to new runs.
- Analyze **only these new chains**
- Compare best-fit likelihood, or evidence, read parameter constraints

# Simple Exercise

**Hst measurement (or at least a first approx)**

- **Model**: $H_0$ varying.
- **Data**: `hst`
- Find the posterior distribution.

# Advanced exercise for a laptop

Because there are no better feeling than sipping a coffee while a computer computes

## Homogeneous cosmology constraints

- Let's confirm the contours of the recent **JLA** results on $\Omega_m$.
- Use the `JLA` likelihood (find out the derived parameters requirements).

# Advanced exercise for a laptop

Because there are no better feeling than sipping a coffee while a computer computes

## Homogeneous cosmology constraints

- Let's confirm the contours of the recent **JLA** results on $\Omega_m$.
- Use the `JLA` likelihood (find out the derived parameters requirements).
- There should be one main parameter, $\Omega_{cdm}$, to reproduce Fig.15 from `http://arxiv.org/abs/1401.4064`
- we need derived parameter $\Omega_m$, and fix CLASS parameters.
- Run over the break to have something to analyze while doing the next lecture.

# Exercise for a laptop

## Because there are no better feeling than sipping a coffee while a computer computes

```
data.experiments=['JLA']

# Cosmological parameters list
data.parameters['Omega_cdm'] = [0.2562,  None, None, 0.008,  1, 'cosmo']

# Nuisance
data.parameters['alpha']   = [0.15 ,  None, None, 0.001, 1, 'nuisance']
data.parameters['beta']    = [3.559,  None, None, 0.02,  1, 'nuisance']
data.parameters['M']       = [-19.02, None, None, 0.004,  1, 'nuisance']
data.parameters['Delta_M'] = [-0.10,  None, None, 0.004,  1, 'nuisance']

# Derived parameter list
data.parameters['Omega_m'] = [0,    -1, -1, 0,1,  'derived']


data.cosmo_arguments['Omega_b'] = 0.05
data.cosmo_arguments['h'] = 0.70
data.cosmo_arguments['T_cmb'] = 2.726
data.cosmo_arguments['N_eff'] = 3.046
data.cosmo_arguments['N_ncdm'] = 0
#------ Mcmc parameters ----
# Number of steps taken, by default (overwritten by the -N command)
data.N=10
# Number of accepted steps before writing to file the chain. Larger means less
# access to disc, but this is not so much time consuming.
data.write_step=5
```

# Commands

```
(mpirun -np 4) python montepython/MontePython.py -N 1000 -p jla.param -o chains/jla
```

```
python montepython/MontePython.py info chains/jla
cp chains/jla/jla.covmat chains
cp chains/jla/jla.bestfit chains
```

```
(mpirun -np 4) python montepython/MontePython.py -N 20000 -p jla.param -o chains/jla \
               -c chains/jla.covmat -b chains/jla.bestfit
```