Getting Started with `MontePython` Solution to Exercises

Antonio J. Cuesta, Universidad de Córdoba
Miguel Zumalacárregui, NORDITA & Lawrence Berkeley National Lab

Cosmology School in the Canary Islands
Fuerteventura, 18–22 September 2017

Abstract

This is a simple tutorial in order to introduce students to the use of the cosmological code `MontePython`. The exercises were designed for the Tutorial session "Bayesian estimation of cosmological parameters" held on the first day of the *Cosmology School in the Canary Islands*. This school was held in September 18-22 in Fuerteventura, Spain, and organized by the Instituto de Astrofísica de Canarias. The code `MontePython` (Audren et al., 2013) is a Bayesian parameter inference code for Cosmology. Its website is `http://baudren.github.io/montepython.html`

Step 0: Installation

As a requirement for completing this tutorial, one needs to install `CLASS`[1] and `MontePython`[2]. The best way to avoid compilation problems in `CLASS` is to install a Python distribution such as `Anaconda`[3]. If you already have such distribution, you can safely skip its installation. Otherwise the following script should help. The installer for `Anaconda` will ask you some questions, to which you should answer 'yes'. Besides, if you do not want these codes to be installed just below your home folder, feel free to modify the `ROOT` environment variable.

```
#!/bin/bash
ROOT=$HOME

# Downloading and installing Anaconda+PySide
export VERSION=4.4.0-MacOSX-x86_64 # if you use Linux, export VERSION=4.4.0-Linux-x86_64
cd $ROOT
curl https://repo.continuum.io/archive/Anaconda2-$VERSION.sh -o Anaconda2-$VERSION.sh
bash Anaconda2-$VERSION.sh
export PATH="$ROOT/anaconda2/bin:$PATH"
conda install -c https://conda.anaconda.org/anaconda pyside #necessary for GetDist GUI
```

This script will install `CLASS` (Lesgourgues, 2011), `hi_class` (Zumalacárregui et al., 2017), and `MontePython` (Audren et al., 2013). Again, one can set up the `ROOT` environment variable to modify the installation target folder.

```
#!/bin/bash
ROOT=$HOME

# Install hi_class
cd $ROOT
curl https://codeload.github.com/miguelzuma/hi_class_public/zip/hi_class -o hi_class.zip
unzip hi_class.zip
mv hi_class_public-hi_class hi_class
cd hi_class
make all -j
export STRING=‘ls -1 /Users/ajcuesta/class_public/python/build/ |head -n1‘
export STRING=${STRING:4}
export PYTHONPATH=$ROOT/hi_class/python/build/lib.$STRING/:$PYTHONPATH
echo -e "\nexport PYTHONPATH=$ROOT/hi_class/python/build/lib.$STRING/:\$PYTHONPATH" >> ~/.bashrc

# Install CLASS
export VERSION=2.6.1
cd $ROOT
curl https://codeload.github.com/lesgourg/class_public/zip/v$VERSION -o class_public.zip
unzip class_public.zip
mv class_public-$VERSION class_public
cd class_public
make all -j
export STRING=‘ls -1 /Users/ajcuesta/class_public/python/build/ |head -n1‘
export STRING=${STRING:4}
export PYTHONPATH=$ROOT/class_public/python/build/lib.$STRING/:$PYTHONPATH
echo -e "\nexport PYTHONPATH=$ROOT/class_public/python/build/lib.$STRING/:\$PYTHONPATH" >> ~/.bashrc

# Install MontePython
export VERSION=2.2
cd $ROOT
curl https://codeload.github.com/baudren/montepython_public/zip/$VERSION -o montepython_public.zip
unzip montepython_public.zip
mv montepython_public-$VERSION montepython_public
cd montepython_public
cp default.conf.template default.conf
sed -i -e '/root/d' default.conf
echo -e "root='$ROOT'" >> default.conf
echo -e "path['cosmo'] = root+'/class_public'" >> default.conf
echo -e "path['clik']  = root+'/plc-2.0'" >> default.conf
cp default.conf.template hi-class.conf
sed -i -e '/root/d' hi-class.conf
echo -e "root='$ROOT'" >> hi-class.conf
echo -e "path['cosmo'] = root+'/hi_class'" >> hi-class.conf
echo -e "path['clik']  = root+'/plc-2.0'" >> hi-class.conf
```

[1]http://class-code.net/

[2]http://baudren.github.io/montepython.html

[3]http://www.continuum.io

Step 1: Writing the parameter file

Our goal is to reproduce the cosmological constraints from type-Ia supernovae in the paper by Betoule et al. (2014).

We base our parameter file on the file `jla.param`. Note that the `JLA` likelihood requires four nuisance parameters, so it is necessary to add them to the parameter file. On top of that, we will require (see Section 6 of Betoule et al. 2014) that we vary also the parameter Ω_m, or indirectly, the parameter $\Omega_{\rm cdm}$. In addition, we will ask `MontePython` to compute the parameter Ω_Λ, which (in this model ΛCDM) can be *derived* as a function of the free parameters ($\Omega_\Lambda = 1 - \Omega_m$). In this way we will have Ω_Λ at each MCMC step added to the chains, so that we do not have to compute that information later, which would require a little extra effort (see `MontePython`'s documentation for the `-m Der` option).

```
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, 0.020,  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.parameters['Omega_Lambda'] = [0,    -1, -1, 0,1,  'derived']

data.cosmo_arguments['Omega_b'] = 0.05

data.N=10
data.write_step=5
```

Note that in the ΛCDM model there is no need to vary additional parameters with this dataset, since distances in this model will only depend on Ω_m and H_0, and H_0 is completely degenerate with the absolute magnitude M of the standard supernova. We can save the above input file as `lcdm_jla.param` in MontePython's main folder, and we are ready to run our first chains.

Step 2: Running the chains

Before we start running chains that use the `JLA` likelihood, `MontePython` requires to download the `JLA` data from the official website. We provide another script for that:

```
#!/bin/bash
ROOT=$HOME

# Download JLA data
cd $ROOT
curl http://supernovae.in2p3.fr/sdss_snls_jla/jla_likelihood_v4.tgz -o jla_likelihood_v4.tgz
tar xvzf jla_likelihood_v4.tgz
mkdir $ROOT/montepython_public/data/JLA/
cp jla_likelihood_v4/data/* $ROOT/montepython_public/data/JLA/
```

Once the parameter file `lcdm_jla.param` has been set up, we can type `mkdir lcdm; mkdir lcdm/jla` to create the output folder (following the recommended structure `cosmological_model/dataset_combination`). We want to run several chains to provide different initial conditions to sample the parameter space, and to ensure unimodality of the probability distribution. We also want to sample the distribution with a large number of points because of the high-dimensionality of the parameter space[4]. To run 4 chains for 10000 steps, using `MontePython.py run`, we do the following:

```
for n in {1..4}
do python montepython/MontePython.py run -p lcdm_jla.param -o lcdm/jla/ -N 10000
done
```

On a laptop, this should take about 3 minutes for each 1000 points computed, so about half an hour per chain. If this is run sequentially (e.g. in a single-core processor) it would take about 2 hours to run all four chains. On the other hand, if one has a multi-core processor, appending the & symbol at the end of the `do` command will try to run each instance of `MontePython` in a different core, so in a 4-core processor it would run all chains simultaneously, taking half an hour to complete all of them. Alternatively, if you have Python's `mpi4py` module installed, you can replace the `do` loop with this command:

```
mpirun -np 4 python montepython/MontePython.py run -p lcdm_jla.param -o lcdm/jla/ -N 10000
```

[4]Remember that `MontePython` does not stop automatically when a convergence criterion is met, so one has to perform exploratory runs to study how many points can be enough.

Step 3: Analyzing the results

Now that we have several tens of thousand points sampling our (5-dimensional) parameter space, we can obtain parameter constraints, probability distribution function plots for each parameter, and contour plots showing the marginalized 2-dimensional constraints for each pair of parameters, which are useful to visualize correlations between parameters. Also, we can obtain information about the convergence of our chains given the value of the $R-1$ diagnostic. To obtain all these details, we simply run `MontePython.py info` on our output folder:

```
python montepython/MontePython.py info lcdm/jla/
```

Running the chains to good convergence falls beyond the point of this short session. However, as pointed out in `MontePython`'s documentation `http://monte-python.readthedocs.io/en/latest/` one can obtain acceptable $R-1$ values (i.e. below 0.01 for every parameter) with an iterative strategy: running a series of short ($\sim 10^3$ points) can help estimate a (noisy, but informative) parameter covariance matrix that can be used as an input for a second set of larger chains (with $\sim 10^4$ points) that need to be output to a different directory. The covariance matrix can be generated with the `--want-covmat` option of `MontePython.py info`, and the corresponding `lcdm/jla/jla.bestfit` and `lcdm/jla/jla.covmat` files can be input to `MontePython.py run` with the `--bestfit` and `--covmat` options respectively. Alternatively, one can avoid this iterative strategy by using the recent `--update` option, which updates the parameter covariance matrix on-the-fly. A value of 300-500 steps in this `--update` option is usually recommended.

Note that one can check if the sampling the parameter space is efficient by running `MontePython.py info` to check the acceptance rate in the Metropolis-Hastings algorithm. The target should be a value between 20% and 25% of the points being accepted. If it falls towards a very different value, one can rerun the chains with a different step size parameter by adjusting the `-f` option of `MontePython.py run` to a different value to the default value of 2.4. One can obtain a short explanation for this value by running `MontePython.py run --help`.

The computed constraints will be written to the files `lcdm/jla/lcdm_jla.v_info` and `lcdm/jla/lcdm_jla.h_info`, and the generated plots will be saved to the folder `lcdm/jla/plots`. However, sometimes we might not be interested in obtaining the constraints from all parameters (in particular from the nuisance ones), so we can leave them out from our plots. In order to do this, we have to use the `--extra` option of `MontePython.py info`. This option takes as an argument the name of a file containing customization options. For example, we can write the following file and save it to `lcdm_jla.extra`:

```
info.to_plot = {'Omega_m', 'alpha', 'beta', 'Delta_M'}
```

Once we do this, we run `MontePython.py info` again, but this time with the customizations contained in the `lcdm_jla.extra` file:

```
python montepython/MontePython.py info lcdm/jla/ --no-mean --extra lcdm_jla.extra
```

This will generate the file `lcdm/jla/plots/jla_triangle.pdf` which is similar to Figure 9 in Betoule et al. (2014) (except that the plots are displayed in a lower-triangular arrangement, and in a different order):

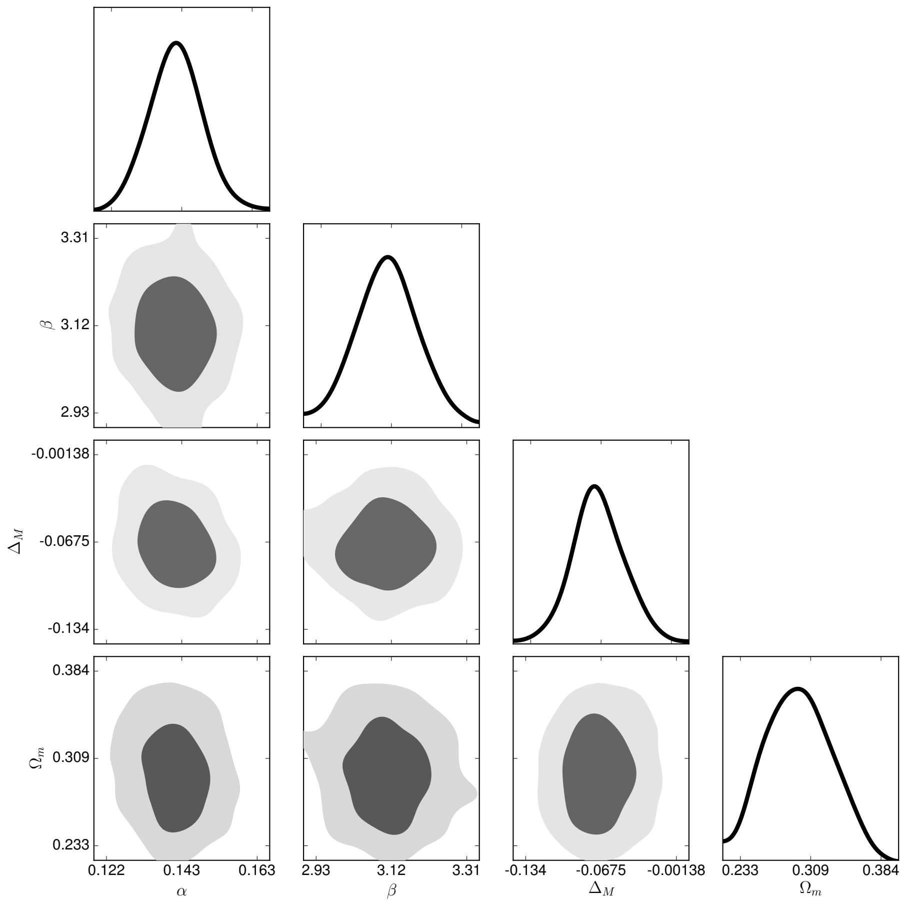

Figure 1: Constraints from the JLA dataset on the ΛCDM cosmological model, including nuisance parameters from such dataset.

Constraining a different cosmological model

We have successfully computed constrains on the ΛCDM model from the JLA dataset. If we are interested in constraining a different model (using this very same dataset), we can simply specify the parameters that define such model in the input parameter file, by defining the boundary values and an initial estimate of their expected value and uncertainty. For example, if we are interested in constraining the *o*-ΛCDM model, in which the spatial curvature is a free parameter, the parameter file (which will be saved with a file name that reflects the difference in the model, i.e. `olcdm_jla.param`) will contain an additional entry in the cosmological parameters list:

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

# Cosmological parameters list
data.parameters['Omega_cdm']  = [0.2562, None, None, 0.008,  1, 'cosmo']
data.parameters['Omega_k']    = [0.00,   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, 0.020,  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.parameters['Omega_Lambda'] = [0,    -1, -1, 0,1,  'derived']

data.cosmo_arguments['Omega_b'] = 0.05

data.N=10
data.write_step=5
```

We can take advantage that we have forced `MontePython.py run` to compute the parameter Ω_Λ and to save it to our chain files, so that we can now use it to make a plot of the constraints in the Ω_m–Ω_Λ plane that can be compared to Figure 15 in Betoule et al. (2014). In this case, we can edit our `lcdm_jla.extra` file so that it now contains this:

```
import matplotlib.pyplot as plt
info.to_plot = {'Omega_m', 'Omega_Lambda'}
info.cmaps   = [plt.cm.Blues, plt.cm.Greens, plt.cm.Reds_r, plt.cm.Oranges, plt.cm.Purples, plt.cm.gray_r]
info.cm      = ['b', 'g', 'r', 'darkorange', 'purple', 'k']
```

If we run `MontePython.py info` with this `extra` file and the `--all` option to output every subplot and data in separate files, we will obtain the file `lcdm/jla/plots/lcdm_jla.pdf` which can be readily compared to Figure 15 in Betoule et al. (2014):

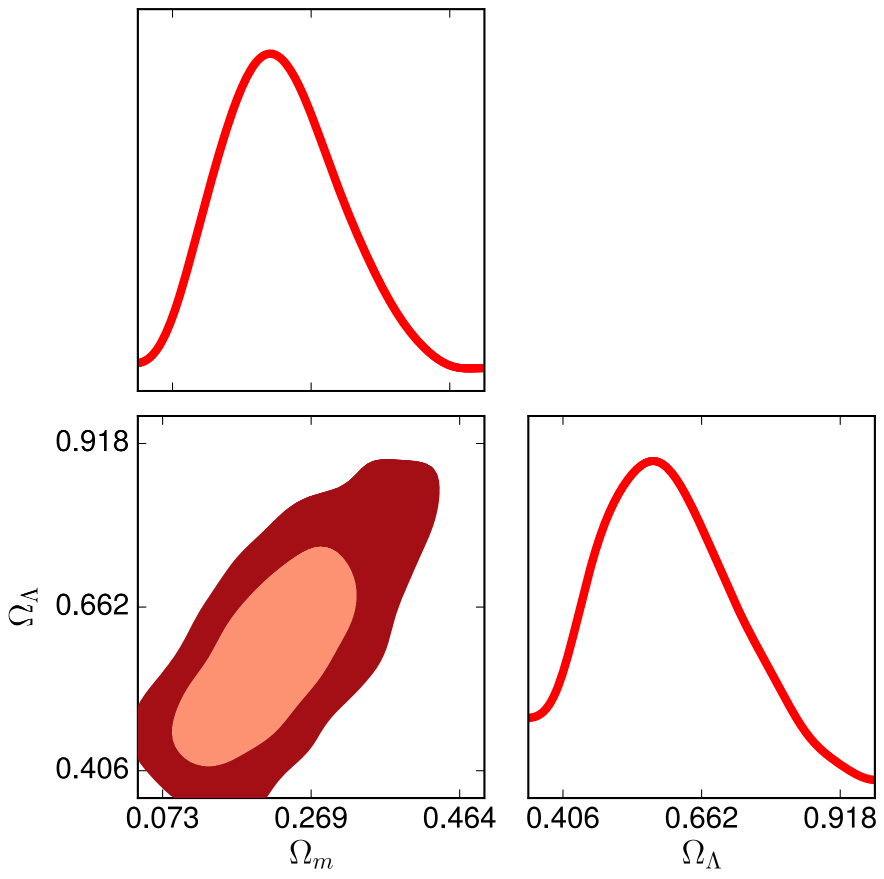

Figure 2: Constraints in the Ω_m–Ω_Λ plane of the o-ΛCDM model from the JLA dataset.

Beyond the basics

The above steps have allowed us to compute cosmological constraints that can be compared to the recent literature and to generate publication-ready plots that can be customized to our best scientific interests. One aspect we have not covered, however, is how to code a (simple) likelihood that allows us to combine the JLA dataset with constraints from Cosmic Microwave Background (CMB) data, at least at the background level[5]. We will call these the CMB *distance priors*. We will use the combination of these two datasets to constrain a slightly more general cosmological model, the wCDM model, in which the dark energy is no longer described by a cosmological constant but by a fluid with an equation of state (the relation between the fluid's pressure and its energy density) equal to w. The ΛCDM model is a particular case of this model in which $w = -1$.

In order to do this, let us create a folder below `montepython/likelihoods` with the name `distance_prior`. This would serve as a replacement for using *Planck* if we are only interested in using the constraints at the background level and not on the perturbations (i.e. on the parameters Ω_b, $\Omega_{\rm cdm}$, and H_0, but not on A_s or n_s, nor on $\tau_{\rm reio}$). This should be much faster than running *Planck*'s full likelihood, and hence this exercise is suitable to be run on a laptop.

As described in `MontePython`'s documentation[6], a likelihood is just a directory which contains a `data` file and a `__init__.py` file. Following equation 18 in Betoule et al. (2014), our likelihood is computed as shown in the following file[7], saved as `montepython/likelihoods/distance_prior/__init__.py`:

```python
import os
import numpy as np
from montepython.likelihood_class import Likelihood_prior


class distance_prior(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):

        omegab, omegac, theta = (
            data.mcmc_parameters[p]['current']*data.mcmc_parameters[p]['scale']
            for p in ['omega_b', 'omega_cdm', '100*theta_s'])
        diffvec = np.array([x-mu for x, mu in zip([omegab, omegac, theta], self.centre)])
        loglkl = -0.5 * np.dot(diffvec.T, np.dot(self.invcov, diffvec))
        return loglkl
```

[5] The installation of the full *Planck* likelihood is out of the scope of this school and its execution for a large number of points is too slow for a laptop.

[6] `http://monte-python.readthedocs.io/en/latest/likelihoods.html`

[7] This is based on the `test_gaussian` likelihood contained in the folder `montepython/likelihoods`.

Then, we just need to define our fiducial (measured) *Planck+WP* values for the parameters $\Omega_b h^2$, $\Omega_{\rm cdm} h^2$, and $100\theta_s$ together with their covariance matrix, also measured by *Planck+WP*. These are specified in equations 19 and 20 of Betoule et al. (2014). In order to avoid doing a matrix inversion every MCMC step, instead of coding the covariance matrix, we will store its inverse. These values are saved into `montepython/likelihoods/distance_prior/distance_prior.py`:

```
# Planck+WP 2013 best fit values for the compressed likelihood {omega_b, omega_cdm, 100*theta_s}.
# (following eq.19 & eq.20 of arXiv:1401.4064,
# see Table 4 of arXiv:1502.01590 for updated values with Planck 2015)
import numpy as np

distance_prior.centre  = [0.022065, 0.1199, 1.041]
distance_prior.invcov  = [[ 1.98527408e7,  0.09401318e7, -0.23904972e7], \
[ 0.09401318e7,  0.02267473e7,  0.02045338e7], [-0.23904972e7,  0.02045338e7,  0.33599883e7]]
```

Once we have written these two files, we have coded a new likelihood that can be readily used by `MontePython`. In particular, since we are interested in computing CMB+SNe constraints on the wCDM model, the parameter file should look like this:

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

#------ Parameter list -------
# data.parameters[class name] = [mean, min, max, 1-sigma, scale, role]
# - if min max irrelevant, put to -1 or None (if you want a boundary of -1, use -1.0)
# - if fixed, put 1-sigma to 0
# - if scale irrelevant, put to 1, otherwise to the appropriate factor
# - role is either 'cosmo', 'nuisance' or 'derived'


# Cosmological parameters list
data.parameters['omega_b']      = [ 2.2253,   None, None,  0.028, 0.01, 'cosmo']
data.parameters['omega_cdm']    = [ 0.1120,   None, None, 0.0016,    1, 'cosmo']
data.parameters['100*theta_s']  = [ 1.0418,   None, None,   3e-4,    1, 'cosmo']
data.parameters['w0_fld']       = [ -1.000,   None, None,   0.05,    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['H0']           = [0, None, None, 0, 1, 'derived']
data.parameters['Omega_m']      = [0, None, None, 0, 1, 'derived']
data.parameters['Omega0_fld']   = [0, None, None, 0, 1, 'derived']

data.cosmo_arguments['Omega_Lambda'] = 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
```

Now we can make a plot that can be compared to Figure 16 in Betoule et al. (2014), at least for the *Planck*+WP+JLA dataset combination. We will make this plot using `CosmoMC`'s graphical interface for analysis, named `GetDist GUI`. Since this GUI has been developed in Python, we do not need to compile `CosmoMC` for this. Instead, we just use the command `python python/GetDistGUI.py` to launch the interface. To download `CosmoMC` we can use this script:

```
#!/bin/bash
ROOT=$HOME

# Download CosmoMC (no compilation, just for GetDist GUI)
export VERSION=Nov2016
cd $ROOT
curl https://codeload.github.com/cmbant/CosmoMC/zip/$VERSION -o CosmoMC.zip
unzip CosmoMC.zip
cd CosmoMC-$VERSION/
export PYTHONPATH="$ROOT/CosmoMC-$VERSION/python:$PYTHONPATH"
echo -e "\nexport PYTHONPATH=$ROOT/CosmoMC-$VERSION/python:\$PYTHONPATH" >> ~/.bashrc
```

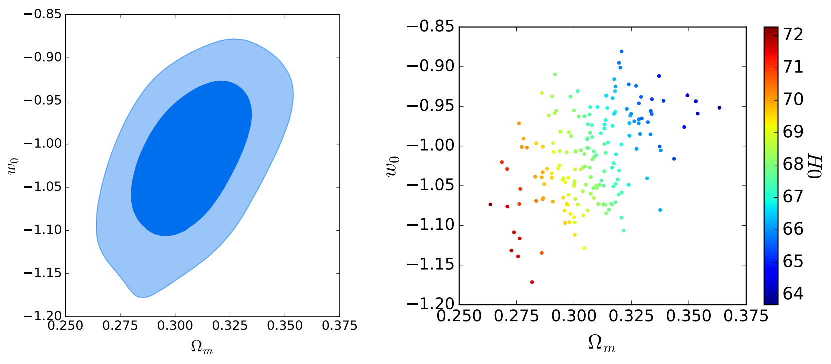

Figure 3: Constraints in the w–Ω_m plane of the w-CDM model from the JLA dataset together with CMB distance priors (left) and color-coded according to their corresponding value of the Hubble constant (right).

Acknowledgments

This tutorial would not have been possible without the support of the organizers of the Cosmology School in the Canary Islands, held in September 18-22 in Fuerteventura, Spain. In particular we would like to thank Francisco-Shu Kitaura (Instituto de Astrofísica de Canarias) for this successful school.

References

Audren, B., Lesgourgues, J., Benabed, K., and Prunet, S. (2013). Conservative constraints on early cosmology with MONTE PYTHON. *Journal of Cosmology and Astroparticle Physics*, 2:001.

Betoule, M., Kessler, R., Guy, J., Mosher, J., Hardin, D., Biswas, R., Astier, P., El-Hage, P., Konig, M., Kuhlmann, S., Marriner, J., Pain, R., Regnault, N., Balland, C., Bassett, B. A., Brown, P. J., Campbell, H., Carlberg, R. G., Cellier-Holzem, F., Cinabro, D., Conley, A., D'Andrea, C. B., DePoy, D. L., Doi, M., Ellis, R. S., Fabbro, S., Filippenko, A. V., Foley, R. J., Frieman, J. A., Fouchez, D., Galbany, L., Goobar, A., Gupta, R. R., Hill, G. J., Hlozek, R., Hogan, C. J., Hook, I. M., Howell, D. A., Jha, S. W., Le Guillou, L., Leloudas, G., Lidman, C., Marshall, J. L., Möller, A., Mourão, A. M., Neveu, J., Nichol, R., Olmstead, M. D., Palanque-Delabrouille, N., Perlmutter, S., Prieto, J. L., Pritchet, C. J., Richmond, M., Riess, A. G., Ruhlmann-Kleider, V., Sako, M., Schahmaneche, K., Schneider, D. P., Smith, M., Sollerman, J., Sullivan, M., Walton, N. A., and Wheeler, C. J. (2014). Improved cosmological constraints from a joint analysis of the SDSS-II and SNLS supernova samples. *Astronomy and Astrophysics*, 568:A22.

Lesgourgues, J. (2011). The Cosmic Linear Anisotropy Solving System (CLASS) I: Overview. *ArXiv e-prints*.

Zumalacárregui, M., Bellini, E., Sawicki, I., Lesgourgues, J., and Ferreira, P. G. (2017). hi_class: Horndeski in the Cosmic Linear Anisotropy Solving System. *Journal of Cosmology and Astroparticle Physics*, 8:019.