EPIC Documentation

Release 1.1

Rafael J. F. Marcondes

May 19, 2018

Contents

1 Welcome to the EPIC user's guide!

Easy Parameter Inference in Cosmology (EPIC) is my implementation in Python of a MCMC code for Bayesian inference of parameters of cosmological models and model comparison via the computation of Bayesian evidences.

1.1 Why EPIC?

I started to develop EPIC as a means of learning how inference can be made with Markov Chain Monte Carlo, rather than trying to decipher other codes or using them as black boxes. The program has fulfilled this purposed and went on to incorporate a few cosmological observables that I have actually employed in some of my publications. Now I release this code in the hope it can be useful to students seeking to learn some of the methods used in Observational Cosmology and even to use it for their own work. It still lacks some important features. A Boltzmann solver is not available. It is possible that I will integrate it with CLASS[1] to make it more useful for more advanced research. Stay tuned for more. Meanwhile, enjoy these nice features:

- Cross-platform: the code runs on Python 3 in any operating system.
- It uses Python's `multiprocessing` library for evolution of chains in parallel. The separate processes can communicate with each other through some `multiprocessing` utilities, which made possible the implementation of the Parallel Tempering algorithm.[2] This method is capable of detecting and accurately sampling posterior distributions that present two or more separated peaks.
- Convergence between independent chains is tested with the multivariate version of the Gelman and Rubin test, a very robust method.
- Also, the plots are beautiful and can be customized to a great extent directly from the command line, without having to change the code. You can view triangle plots with marginalized distributions of parameters, predefined derived parameters, two-dimensional joint-posterior distributions, autocorrelation plots, cross-correlation plots, sequence plots, convergence diagnosis and more.

Try it now!

2 How to install

There are two ways to install this program. You can download and install from PyPi or you can clone it from BitBucket. But first, it is recommended that you make these changes inside a virtual

[1] Lesgourgues, J. "The Cosmic Linear Anisotropy Solving System (CLASS) I: Overview". arXiv:1104.2932 [astro-ph.IM]; Blas, D., Lesgourgues, J., Tram, T. "The Cosmic Linear Anisotropy Solving System (CLASS). Part II: Approximation schemes". Journal of Cosmology and Astroparticle Physics 07 (2011) 034.

[2] Removed in current version. If you need to use Parallel Tempering, please use version 1.0.4 of EPIC.

environment.

2.1 Setting up a virtual environment

if you are on Python 3.3 or superior, you can run:

```
$ python3 -m venv vEPIC
```

to create a virtual Python environment inside a folder named `vEPIC`. Activate it with:

```
$ source vEPIC/bin/activate
```

When you finish using the environment and want to leave it you can just use `deactivate`. To activate it again, which you need in a new session, just run the activation command above. More details about Python3's venv here.

With inferior versions of Python, you can install pyenv and pyenv-virtualenv, which let you create a virtual environment and even choose another version of Python to install. This is done with:

```
$ pyenv virtualenv 3.6.1 vEPIC # or other version you like.
```

Then activate it running:

```
$ pyenv activate vEPIC # use pyenv deactivate vEPIC to deactivate it.
```

Note that this version of EPIC is not compatible with Python 2 anymore.

2.2 Installing with pip

The easiest way to install this program is to get it from PyPi. It is also the most recommended since it can be easily updated when new versions come out. Inside your virtual environment, run:

```
$ pip install epic-code
```

Then, if you are using Linux, create a bind mount from your python path to an external directory for your convenience. For example, in your home, create the directories `cosmology_codes/EPIC` and `cosmology_codes/EPIC-simulations`. Run:

```
$ sudo mount --bind $PYTHONPATH/lib/python3.5/site-packages/EPIC/ cosmology_
↪codes/EPIC
```

This will make your newly created folder reflect the contents of the installation directory and you will be able to run the scripts and access the associated files from there, rather than only importing the modules from inside the python interactive interpreter. If you do not have the privileges to mount the directory then follow the instructions below for downloading and extracting manually to your home directory.

Note: If you are on Mac OS X or macOS, you can achieve the same effect but will need to install osxfuse and bindfs to create the link:

```
$ brew install osxfuse # or brew cask install osxfuse
$ brew install bindfs
$ bindfs $PYTHONPATH/lib/python3.5/site-packages/EPIC/ cosmology_codes/EPIC
```

Next, `cd` into `cosmology_codes/EPIC`, run:

```
$ python define_altdir.py
```

and pass the full path to `cosmology_codes/EPIC-simulations` to define this folder as the location for saving simulations results. You are now good to go. To check and install updates if available, just run:

```
$ pip install --upgrade epic-code
```

On a Windows computer, if you have Windows 10 Pro, I recommend enabling the Windows Subsystem for Linux and installing Bash on Ubuntu on Windows 10, then proceeding with the instructions above for installation on Linux. If not possible, then use the `zip` file as indicated below.

2.3 Downloading and extracting the tarball

If you rather not install it to your system you can just extract the `tar.gz` files from the Python Package Index at https://pypi.python.org/pypi/epic-code. Extract the file with:

```
$ tar -xzf epic-code-[version].tar.gz
```

`cd` into the root folder and install to your (virtual) environment:

```
$ python setup.py install
```

This is necessary to guarantee that all modules will be loaded correctly. It also applies to the source code extracted from the `zip` file or other compression format. You can then run the program from the `EPIC` folder. To update the program you will have to download the new tarball/zip file and execute this process again.

2.4 Cloning the repository

If you plan to contribute to this program you can clone the git repository at https://bitbucket.org/rmarcondes/epic.

3 The Cosmology module

Starting with version 1.1, there is now a module available that makes it easy for the user to perform calculations on the background cosmology given a specific model. A few classes of models are predefined. Each of these have also defined what are their components, but also allowing some variations. For example, the ΛCDM model requires at least cold dark matter (`cdm`) and cosmological constant (`lambda`), but one can also include baryonic fluid (`baryons`), or treat both matter components together by specifying the composed species `matter`. It is also possible to include `photons` or `radiation`. For models of interaction between dark matter and dark energy (`cde`), the former is labelled `idm` and the latter `ide` or `ilambda` in the case that its equation-of-state parameter is still -1 (in which the model is then labelled `cde_lambda`) rather than a free parameter or presents evolution described by some function. The models available and their components specification are defined in the file `EPIC/cosmology/model_recipes.ini`. The fluids contained in this file are in turn defined in `EPIC/cosmology/available_species.ini`, where properties like the type of equation of state, the parameters and other relevant informations are set.

3.1 Initializing a model

We start (preferably on jupyter notebook) importing the module and creating our cosmology object:

Only the label of the model is really needed here, since the essentials are already predefined in the program, as mentioned above. With this, one can explore the properties assigned to the object. For example, `LCDM.model` will print `lcdm`. `LCDM.species` is a dictionary of `Fluid` objects identified by the components labels, in this case `cdm` and `lambda`. There is also a dedicated class for an equation-of-state parameter or function, which becomes an attribute of its fluid. We can assess its value, type, etc. `LCDM.species['lambda'].EoS.value` will print `-1`.

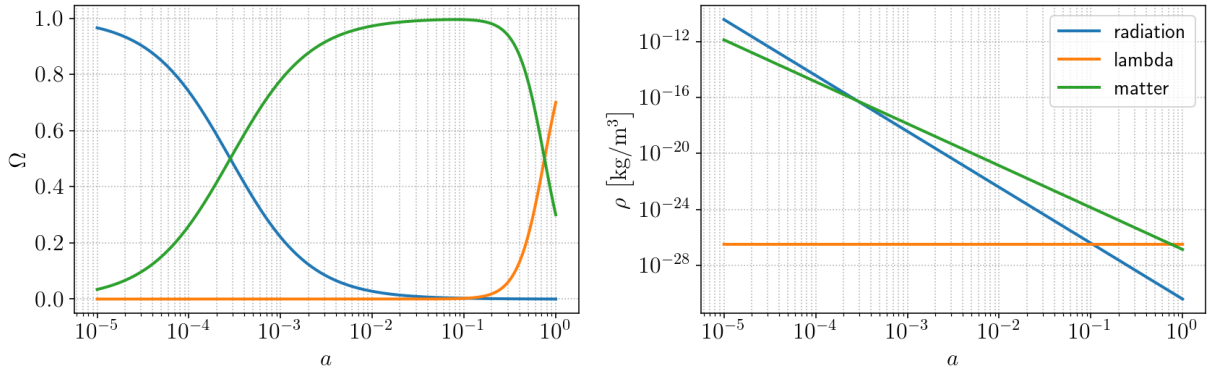
But let us proceed in a slightly different way, setting up our model with some options. Since we predominantly work with flat cosmologies (in fact, curvature is not supported yet in the current version), the flatness condition is imposed in the density parameter of one of the fluids. We will choose the dark energy density parameter to be the derived parameter, but we could have chosen dark matter as well. Also, by default, the code prefers to work with physical densities (for example $\Omega_{c0}h^2$) rather than the common Ω_{c0}. You can change this with the option `physical=False`. We will add the radiation and matter fluids. Note that this will override the optional inclusion of baryons and remove them, if given. The radiation fluid is parametrized by the temperature of the cosmic microwave background. The model will have three free parameters: the density parameter of matter (Ω_{m0}), the CMB temperature ($T_{\rm CMB}$), which we usually keep fixed) and the Hubble parameter H_0; and one derived parameter, which is the density parameter of the cosmological constant, Ω_Λ.

We can then obtain the solution to the background cosmology with EPIC.

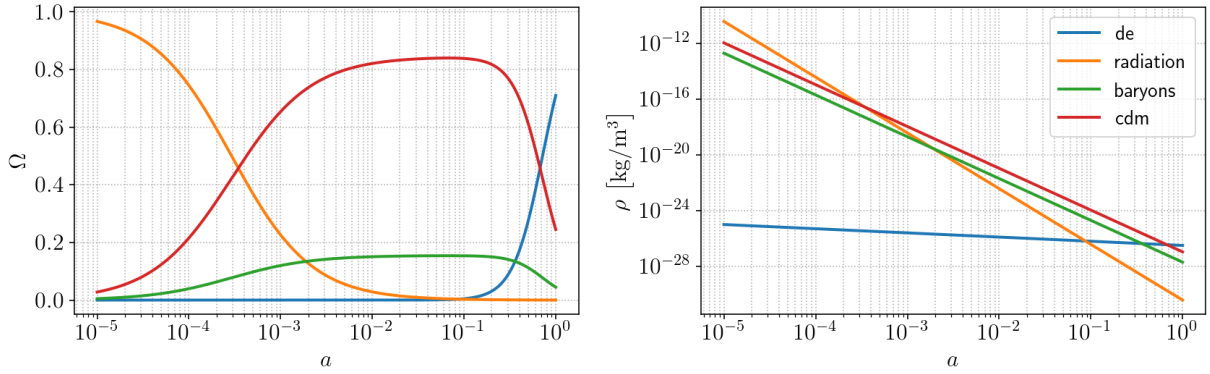
Solving the background cosmology

It is as simple as this:

Normally, a set of parameters would be given to this function in the form of a dictionary with the parameters' labels as keys, like in `parameter_space={'Oc0': 0.26, 'Ob0': 0.048, 'Or0':8e-5, 'H0':67.8}`. However, we can also ommit it and turn on the option `accepts_default` and then the default values defined in the `EPIC/cosmology/default_parameter_values.ini` file will be used for the parameters. Next, we plot the energy densities and density parameters. Here I do it in a jupyter notebook with the help of this simple function below:



Notice the matter-radiation equality moment at $a_{eq} \sim 3 \times 10^{-4}$ and the cosmological constant that just recently came to overtake matter as the dominant component. The wCDM (`wcdm`) model differs from ΛCDM only by the dark energy (`de`) equation-of-state parameter (`wd`), which although still constant can be different from -1. Note that the energy density of dark energy is not constant now:



3.2 The cosmological models

A few models are already implemented. I give a brief description below, with references for works that discuss some of them in detail and works that analyzed them with this code. The models are

objects created from the `cosmic_objects.CosmologicalSetup` class. This class has a generic module `solve_background` that calls the `Fluid`'s module `rho_over_rho0` of each fluid to obtain the solution for their energy densities. When a solution cannot be obtained directly (like in some interacting models), a fourth-order Runge-Kutta integration is done using the function `generic_runge_kutta` from `EPIC.utils`'s `integrators` and the fluids' `drho_da`. There is an intermediate function `get_Hubble_Friedmann` to calculate the Hubble rate either by just summing the energy densities, when called from the Runge-Kutta integration, or calculating them with `rho_over_rho0`.

Some new models can be introduced in the code just by editing the `model_recipes.ini`, `available_species.ini` and (optionally) `default_parameter_values.ini` configuration files, without needing to rebuild and install the EPIC's package. The format of the configuration `.ini` files is pretty straightforward and the containing information can serve as a guide for what needs to be defined.

The ΛCDM model

When baryons and radiation are included, the solution to this cosmology will require values for the parameters Ω_{c0}, Ω_{b0}, $T_{\rm CMB}$, H_0, or h, $\Omega_{c0}h^2$, $\Omega_{b0}h^2$, $T_{\rm CMB}$, and will find $\Omega_\Lambda = 1 - (\Omega_{c0} + \Omega_{b0} + \Omega_{r0})$ or $\Omega_\Lambda h^2 = h^2 - \left(\Omega_{c0}h^2 + \Omega_{b0}h^2 + \Omega_{r0}h^2\right)$ if `physical`.[1] The radiation density parameter Ω_{r0} is calculated according to the CMB temperature $T_{\rm CMB}$, including the contribution of the neutrinos (and antineutrinos) of the standard model. Extending this model to allow curvature is not completely supported yet. The Friedmann equation is

$$\frac{H(z)}{100\ \mathrm{km\ s^{-1}\ Mpc^{-1}}} = \sqrt{(\Omega_{b0}h^2 + \Omega_{c0}h^2)(1+z)^3 + \Omega_{r0}h^2(1+z)^4 + \Omega_d h^2}$$

or

$$H(z) = H_0\sqrt{(\Omega_{b0} + \Omega_{c0})(1+z)^3 + \Omega_{r0}(1+z)^4 + \Omega_d},$$

H_0 is in units of km s^{-1} Mpc^{-1}. This model is identified in the code by the label `lcdm`.

The wCDM model

Identified by `wcdm`, this is like the standard model except that the dark energy equation of state can be any constant w_d, thus having the ΛCDM model as a specific case with $w_d = -1$. The Friedmann equation is like the above but with the dark energy contribution multiplied by $(1+z)^{3(1+w_d)}$.

[1] That is, assuming `derived=lambda`, but we could also have done, for example, `physical=False, derived=matter`, specify Ω_Λ and the code would get $\Omega_{m0} = 1 - (\Omega_\Lambda + \Omega_{r0})$ or, still, without specifying the derived parameter and with `physical` true, specify all the fluids' density parameters and get $h = \sqrt{\Omega_{c0}h^2 + \Omega_{b0}h^2 + \Omega_{r0}h^2 + \Omega_\Lambda h^2}$.

The Chevallier-Polarski-Linder parametrization

The CPL parametrization[2] of the dark energy equation of state

$$w(a) = w_0 + w_a\,(1 - a)$$

is also available. In this case, the dark energy contribution in the Friedmann equation is multiplied by $(1 + z)^{3(1+w_0+w_a)}\, e^{-3w_a z/(1+z)}$ or $(a/a_0)^{-3(1+w_0+w_a)} e^{-3w_a(1-a/a_0)}$, in terms of the scale factor.

The Barboza-Alcaniz parametrization

The Barboza-Alcaniz dark energy equation of state parametrization[3]

$$w(z) = w_0 + w_1 \frac{z\,(1 + z)}{1 + z^2}$$

is implemented. This models gives a dark energy contribution in the Friedmann equation that is multiplied by the term $x^{3(1+w_0)} \left(x^2 - 2x + 2\right)^{-3w_1/2}$, where $x \equiv a_0/a$.

Interacting Dark Energy models

A comprehensive review of models that consider a possible interaction between dark energy and dark matter is given by Wang *et al.* (2016)[4]. In interacting models, the individual conservation equations of the two dark fluids are violated, although still preserving the total energy conservation:

$$\dot{\rho}_c + 3H\rho_c = Q$$
$$\dot{\rho}_d + 3H(1 + w_d)\rho_d = -Q.$$

The shape of Q is what characterizes each model. Common forms are proportional to ρ_c, to ρ_d (both supported) or to some combination of both (not supported in this version).

To create an instance of a coupled model (`cde`) with $Q \propto \rho_c$, use:

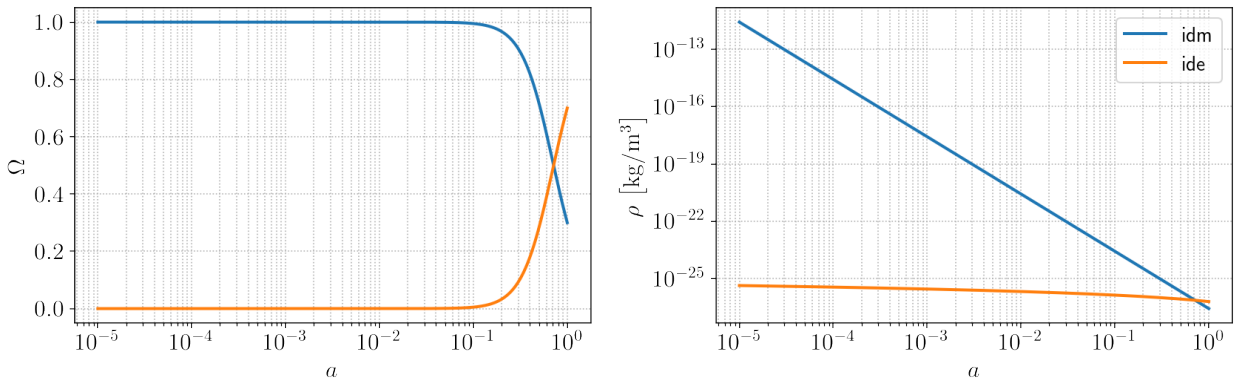
The mandatory species are `idm` and `ide`. You can add `baryons` in the `optional_species` list keyword argument, but note that `matter` is not available as a combined species for this model type since dark matter is interacting with another fluid while baryons are not. What is new here is the `interaction_setup` dictionary. This is where we tell the code which `species` are interacting (at the moment only an energy exchange within a pair is supported), to which of them (`idm`) we associate the interaction `parameter xi`, indicate the second one (`ide`) as having an interaction

[2] Chevallier M. & Polarski D., "Accelerating Universes with scaling dark matter". International Journal of Modern Physics D 10 (2001) 213-223; Linder E. V., "Exploring the Expansion History of the Universe". Physical Review Letters 90 (2003) 091301.

[3] Barboza E. M. & Alcaniz J. S., "A parametric model for dark energy". Physics Letters B 666 (2008) 415-419.

[4] Wang B., Abdalla E., Atrio-Barandela F., Pavón D., "Dark matter and dark energy interactions: theoretial challenges, cosmological implications and observational signatures". Reports on Progress in Physics 79 (2016) 096901.

term proportional to the other (`idm`) and specify the sign of the interaction term for each fluid, in this case that means $Q_c = 3H\xi\rho_c$ and $Q_d = -3H\xi\rho_c$.



Here, I am exaggerating the value of the interaction parameter so we can see a variation on the dark energy density that is due to the interaction, not the equation-of-state parameter, which is -1. This same cosmology can be realized with the model type `cde_lambda` without specifying the parameter `wd`, since the `ilambda` fluid has fixed $w_d = -1$. The dark matter interacting term Q_c is positive with ξ positive, thus the lowering of the dark energy density as its energy flows towards dark matter.

Fast-varying dark energy equation-of-state models

Models of dark energy with fast-varying equation-of-state parameter have been studied in some works[5]. Three such models were implemented as described in Marcondes and Pan (2017)[6]. We used this code in that work. They have all the density parameters present in the ΛCDM model besides the dark energy parameters that we describe in the following.

Model 1

This model `fv1` has the free parameters w_p, w_f, a_t and τ characterizing the equation of state

$$w_d(a) = w_f + \frac{w_p - w_f}{1 + (a/a_t)^{1/\tau}}.$$

w_p and w_f are the asymptotic values of w_d in the past ($a \to 0$) and in the future ($a \to \infty$), respectively; a_t is the scale factor at the transition epoch and τ is the transition width. The Friedmann

[5] Corasaniti P. S. & Copeland E. J., "Constraining the quintessence equation of state with SnIa data and CMB peaks". Physical Review D 65 (2002) 043004; Basset B. A., Kunz M., Silk J., "A late-time transition in the cosmic dark energy?". Monthly Notices of the Royal Astronomical Society 336 (2002) 1217-1222; De Felice A., Nesseris S., Tsujikawa S., "Observational constraints on dark energy with a fast varying equation of state". Journal of Cosmology and Astroparticle Physics 1205, 029 (2012).

[6] Marcondes R. J. F. & Pan S., "Cosmic chronometer constraints on some fast-varying dark energy equations of state". arXiv:1711.06157 [astro-ph.CO].

equation is

$$\frac{H(a)^2}{H_0^2} = \frac{\Omega_{r0}}{a^4} + \frac{\Omega_{m0}}{a^3} + \frac{\Omega_{d0}}{a^{3(1+w_p)}} f_1(a),$$

where

$$f_1(a) = \left(\frac{a^{1/\tau} + a_t^{1/\tau}}{1 + a_t^{1/\tau}} \right)^{3\tau(w_p - w_f)}.$$

Model 2

This model `fv2` alters the previous model to allow the dark energy to feature an extremum value of the equation of state:

$$w_d(a) = w_p + (w_0 - w_p)\, a\, \frac{1 - (a/a_t)^{1/\tau}}{1 - (1/a_t)^{1/\tau}},$$

where w_0 is the current value of the equation of state and the other parameters have the interpretation as in the previous model. The Friedmann equation is

$$\frac{H(a)^2}{H_0^2} = \frac{\Omega_{r0}}{a^4} + \frac{\Omega_{m0}}{a^3} + \frac{\Omega_{d0}}{a^{3(1+w_p)}} e^{f_2(a)},$$

with

$$f_2(a) = 3(w_0 - w_p) \frac{1 + (1 - a_t^{-1/\tau})\tau + a[\{(a/a_t)^{1/\tau} - 1\}\tau - 1]}{(1 + \tau)(1 - a_t^{-1/\tau})}.$$

Model 3

Finally, we have a third model `fv3` with the same parameters as in Model 2 but with equation of state

$$w_d(a) = w_p + (w_0 - w_p)\, a^{1/\tau}\, \frac{1 - (a/a_t)^{1/\tau}}{1 - (1/a_t)^{1/\tau}}.$$

It has a Friedmann equation identical to Model 2's except that $f_2(a)$ is replaced by

$$f_3(a) = 3(w_0 - w_p)\tau \frac{2 - a_t^{-1/\tau} + a_t^{1/\tau}[(a/a_t)^{1/\tau} - 2]}{2(1 - a_t^{-1/\tau})}.$$

3.3 The datasets

Observations available for comparison with model predictions are registed in the `EPIC/cosmology/observational_data/available_observables.ini`. They are separated in sections (`Hz`, `H0`, `SNeIa`, `BAO` and `CMB`), which contain the name of the `CosmologicalSetup`'s module for the observable theoretical calculation (`predicting_function_name`). Each dataset goes below one of the sections, with the folder or text file containing the data indicated. The path is relative to the `EPIC/cosmology/observational_data/` folder. If the folder name is the same as the observable label it can be ommited. Besides, the `Dataset` subclasses are defined at the beginning of the `.ini` file. Each of these classes has its own methods for initialization and likelihood evaluation.

We choose the datasets by passing a dictionary with observables as keys and datasets as values to the function `choose_from_datasets` from `observations` in `EPIC.cosmology`:

The `WiggleZ` dataset is available but is commented out because it is correlated with the SDSS datasets and thus not supposed to be used together with them. Refer to the papers published by the authors of the observations for details. Other incompatible combinations are defined in the `conflicting_dataset_pairs.txt` file in `EPIC/cosmology/observational_data/`. The code will check for these conflicts prior to proceeding with an analysis. The returned flattened dictionary of Dataset objects will later be passed to a MCMC analysis. Now I describe briefly the datasets made available by the community.

Type Ia supernovae

Two types of analyses can be made with the JLA catalogue. One can either use the full likelihood (`JLA_full`) or a simplified version based on 30 redshift bins (`JLA_simplified`). Here we are using the binned data consisting of distance modulus estimates at 31 points (defining 30 bins of redshift). If you want to use the full dataset (which makes the analysis much slower since it involves three more nuisance parameters and requires the program to invert a 740 by 740 matrix at every iteration for the calculation of the JLA likelihood), you need to download the covariance matrix data (`covmat_v6.tgz`) from http://supernovae.in2p3.fr/sdss_snls_jla/ReadMe.html. The `covmat` folder must be extracted to the `jla_likelihood_v6` folder. This is not included in EPIC because the data files are too big.

Either way, the data location is the same data folder `jla_likelihood_v6`. Note that the binned dataset introduces one nuisance parameter `M`, representing an overall shift in the absolute magnitudes, and the full dataset introduces four nuisance parameters related to the light-curve parametrization. See Betoule et al. (2014)[1] for more details.

[1] Betoule M. et al. "Improved cosmological constraints from a joint analysis of the SDSS-II and SNLS supernova samples". Astronomy & Astrophysics 568, A22 (2014).

CMB distance priors

Constraining models with temperature or polarization anisotropy amplitudes is not currently implemented. However, you can include the CMB distance priors from Planck2015.[2] The datasets consist of an acoustic scale l_A, a shift parameter R and the physical density of baryons $\Omega_{b0}h^2$. You can choose between the data for ΛCDM and wCDM with either `Planck2015_distances_LCDM` or `Planck2015_distances_wCDM`. `Planck2015_distances_LCDM+Omega_k` is also available for When curvature is supported.

BAO data

Measurements of the baryon acoustic scales from the Six Degree Field Galaxy Survey (6dF) combined with the most recent data releases of Sloan Digital Sky Survey (SDSS-MGS),[3] the LOWZ and CMASS galaxy samples of the Baryon Oscillation Spectroscopic Survey (BOSS-LOWZ and BOSS-CMASS),[4] data from the Quasar-Lyman α cross-correlation,[5] the distribution of the Lyman α forest in BOSS[6] and from the WiggleZ Dark Energy Survey[7] are available in the `BAO` folder, as well as the latest consensus from the completed SDSS-III BOSS survey.[8] The observable is based on the value of the characteristic ratio $r_s(z_d)/D_V(z)$ between the sound horizon r_s at decoupling time (z_d) and the effective BAO distance D_V, or some variation of this. The respective references are given in the headers of the data files.

$H(z)$ data

These are the cosmic chronometer data. 30 measurements of the Hubble expansion rate $H(z)$ at redshifts between 0 and 2.[9] The values of redshift, H and the uncertainties are given in the file `Hz/Hz_Moresco_et_al_2016.txt`.

[2] Huang Q.-G., Wang K. & Wang S. "Distance priors from Planck 2015 data". Journal of Cosmology and Astroparticle Physics 12 (2015) 022.

[3] Carter P. et al. "Low Redshift Baryon Acoustic Oscillation Measurement from the Reconstructed 6-degree Field Galaxy Survey". arXiv:1803.01746v1 [astro-ph.CO].

[4] Anderson L. et al. "The clustering of galaxies in the SDSS-III Baryon Oscillation Spectroscopic Survey: measuring D_A and H at $z = 0.57$ from the baryon acoustic peak in the Data Release 9 spectroscopic Galaxy sample". Monthly Notices of the Royal Astronomical Society 438 (2014) 83-101.

[5] Font-Ribera A. et al. "Quasar-Lyman α forest cross-correlation from BOSS DR11: Baryon Acoustic Oscillations". Journal of Cosmology and Astroparticle Physics 05 (2014) 027.

[6] Bautista J. E. et al. "Measurement of baryon acoustic oscillation correlations at $z = 2.3$ with SDSS DR12 Ly α-Forests". Astronomy & Astrophysics 603 (2017) A12.

[7] Kazin E. A. et al. "The WiggleZ Dark Energy Survey: improved distance measurements to $z = 1$ with reconstruction of the baryonic acoustic feature". Monthly Notices of the Royal Astronomical Society 441 (2014) 3524-3542.

[8] Alam S. et al. "The clustering of galaxies in the completed SDSS-III Baryon Oscillation Spectroscopic Survey: cosmological analysis of the DR12 galaxy sample". Monthly Notices of the Royal Astronomical Society 470 (2017) 2617-2652.

[9] Moresco M. et al. "A 6% measurement of the Hubble parameter at $z \sim 0.45$: direct evidence of the epoch of cosmic re-acceleration". Journal of Cosmology and Astroparticle Physics 05 (2016) 014.

H_0 data

The 2.4% precision local measure[10] of H_0 is present in `H0/local_H0_Riess_et_al_2016.txt`.

4 The MCMC module

This module is the original part of the program, although massively rewritten from version 1.0.4 to 1.1. Originally, EPIC could run with standard MCMC sampler or Parallel-Tempering MCMC. The former has been temporarily removed to give place to a new and cleaner implementation attempting to solve some bugs. Use version 1.0.4 in case you need it. The MCMC sampler comes with an experimental adaptive routine that adjusts the covariance of the proposal multivariate Gaussian probability density for optimal efficiency, aiming at an acceptance rate around 0.234. In the following sections I briefly introduce the MCMC method and show how to use this program to perform simulations, illustrating with examples.

4.1 Introduction to MCMC and the Bayesian method

Users familiar with the Markov Chain Monte Carlo (MCMC) method may want to skip to the next section. The typical problem the user will want to tackle with this program is the problem of parameter estimation of a given theoretical model confronted with one or more sets of observational data. This is a very common task in Cosmology these days, specially in the light of numerous data from several surveys, with increasing quality. Important discoveries are expected to be made with the data from new generation telescopes in the next decade.

In the following I give a very brief introduction to the MCMC technique and describe how to use program.

The Bayes Theorem

Bayesian inference is based on the inversion of the data-parameters probability relation, which is the Bayes theorem[1]. This theorem states that the posterior probability $P(\theta \mid D, \mathcal{M})$ of the parameter set θ given the data D and other information from the model $\mathcal{M}$ can be given by

$$P(\theta \mid D, \mathcal{M}) = \frac{\mathcal{L}(D \mid \theta, \mathcal{M})\, \pi(\theta \mid \mathcal{M})}{P(D, \mathcal{M})},$$

where $\mathcal{L}(D \mid \theta, \mathcal{M})$ is the likelihood of the data given the model parameters, $\pi(\theta \mid \mathcal{M})$ is the prior probability, containing any information known *a priori* about the distribution of the parameters, and $P(D, \mathcal{M})$ is the marginal likelihood, also popularly known as the evidence, giving the normalization

[10] Riess A. G. et al. "A 2.4% determination of the local value of the Hubble constant". The Astrophysical Journal 826 (2016) 56.

[1] Hobson M. P., Jaffe A. H., Liddle A. R., Mukherjee P. & Parkinson D., "Bayesian methods in cosmology". (Cambridge University Press, 2010).

of the posterior probability. The evidence is not required for the parameter inference but is essential in problems of selection model, when comparing two or more different models to see which of them is favored by the data.

Direct evaluation of $P(\theta \mid D, \mathcal{M})$ is generally a difficult integration in a multiparameter space that we do not know how to perform. Usually we do know how to compute the likelihood $\mathcal{L}(D \mid \theta, \mathcal{M})$ that is assigned to the experiment (most commonly a distribution that is Gaussian on the data or the parameters), thus the use of the Bayes theorem to give the posterior probability. Flat priors are commonly assumed, which makes the computation of the right-hand side of the equation above trivial. Remember that the evidence is a normalization constant not necessary for us to learn about the most likely values of the parameters.

The Metropolis-Hastings sampler

The MCMC method shifts the problem of calculating the unknown posterior probability distribution in the entire space, which can be extremly expensive for models with large number of parameters, to the problem of sampling from the posterior distribution. This is possible, for example, by growing a Markov chain with new states generated by the Metropolis sampler[2].

The Markov chain has the property that every new state depends on its current state, and only on this current state. Dependence on more previous states or on some statistics involving all states is not allowed. That can be done and can even also be useful for purposes like ours, but then the chain can not be called Markovian.

The standard MCMC consists of generating a random state y according to a proposal probability $Q(\cdot \mid x_t)$ given the current state x_t at time t. Then a random number u is drawn from a uniform distribution between 0 and 1. The new state is accepted if $r \geq u$, where

$$r = \min\left[1, \frac{P(y \mid D, \mathcal{M})Q(x_t \mid y)}{P(x_t \mid D, \mathcal{M})Q(y \mid x_t)}\right].$$

The fraction is the Metropolis-Hastings ratio. When the proposal function is symmetrical, $\frac{Q(x_t|y)}{Q(y|x_t)}$ reduces to 1 and the ratio is just the original Metropolis ratio of the posteriors. If the new state is accepted, we set $x_{t+1} := y$, otherwise we repeat the state in the chain by setting $x_{t+1} := x_t$.

The acceptance rate $\alpha = \frac{\text{number of accepted states}}{\text{total number of states}}$ of a chain should be around 0.234 for optimal efficiency[3]. This can be obtained by tuning the parameters of the function Q. In this implementation, I use a multivariate Gaussian distribution with a diagonal covariance matrix S.

The Parallel Tempering algorithm (removed in this version)

Standard MCMC is powerful and works in most cases but there are some problems where the user may be better off using some other method. Due to the characteristic behavior of a Markov

[2] Gayer C., "Introduction to Markov Chain Monte Carlo". in "Handbook of Markov Chain Monte Carlo" http://www.mcmchandbook.net/

[3] Roberts G. O. & Rosenthal J. S., "Optimal scaling for various Metropolis-Hastings algorithms". Statistical Science 16 (2001) 351-367.

chain, it is possible (and even likely) that a chain become stuck in a single mode of a multimodal distribution. If two or more peaks are far away from each other, the proposal function tuned for good performance in a peak may have difficulty escaping that peak to explore the other, because the jump may be too short. To overcome this inefficiency, a neat variation of MCMC, called Parallel Tempering[4], favors a better exploration of the entire parameter space in such cases thanks to an arrangement of multiple chains that are run in parallel, each one with a different ''temperature" T. The posterior is calculated as $\mathcal{L}^{\beta}\pi$, with $\beta = 1/T$. The first chain is the one that corresponds to the real life posterior we are interested in; the other chains, at higher temperatures, will have wider distributions, which makes it easier to jump between peaks, thus exploring more properly the parameter space. Periodically, a swap of states between neighboring chains is proposed and accepted or rejected according to a Hastings-like ratio.

4.2 Before starting

In the section *The Cosmology module* we learned how to use EPIC to set up a cosmological model and load some datasets. The next logical step is to calculate the probability density at a given point of the parameter space, given that model and according to the chosen data. This can be done as follows:

In this example I am choosing the cosmic chronometers dataset, the Hubble constant local measurement and the simplified version of the JLA dataset. The `Analysis` object is created from the dictionary of datasets, the model and a dictionary of priors in the model parameters (including nuisance parameters related to the data). The probability density at any point can then be calculated with the module `log_posterior`, which returns the logarithm of the posterior probability density and the logarithm of the likelihood. Setting the option `chi2` to `True` (It is `False` by default) makes the calculation of the likelihood as $\log \mathcal{L} = -\chi^2/2$, dropping the usual multiplicative terms from the normalized Gaussian likelihood. When false, the results include the contribution of the factors $1/\sqrt{2\pi}\sigma_i$ or the factor $1/\sqrt{2\pi|\mathbf{C}|}$. These are constant in most cases, making no difference to the analysis, but in other cases, depending on the data set, the covariance matrix $\mathbf{C}$ can depend on nuisance parameters and thus vary at each point.

Now that we know how to calculate the posterior probability at a given point, we can perform a Monte Carlo Markov Chain simulation to assess the confidence regions of the model parameters. The main script `epic.py` accomplishes this making use of the objects and modules here presented.

The configuration of the analysis (choice of model, datasets, priors, etc) is defined in a `.ini` configuration file that the program reads. The program creates a folder in the working directory with the same name of this `.ini` file, if it does not already exist. Another folder is created with the date and time for the output of each run of the code, but you can always continue a previous run from where it stopped, just giving the folder name instead of the `.ini` file. The script is stored in the `EPIC` source folder, where the `.ini` files should also be placed. The default working directory is the `EPIC`'s parent directory, i.e., the `epic` repository folder.

[4] Gregory P. C., "Bayesian logical data analysis for the physical sciences: a comparative approach with Mathematica support". (Cambridge University Press, 2005).

Changing the default working directory

By default, the folders with the name of the `.ini` files are created at the repository root level. But the chains can get very long and you might want to have them stored in a different drive. In order to set a new default location for all the new files, run:

```
$ python define_altdir.py
```

This will ask for the path of the folder where you want to save all the output of the program and keep this information in a file `altdir.txt`. If you want to revert this change you can delete the `altdir.txt` file or run again the command above and leave the answer empty when prompted. To change this directory temporarily you can use the argument `--alt-dir` when running the main script.

The structure of the `.ini` file

Let us work with an example, with a simple flat ΛCDM model. Suppose we want to constrain its parameters with $H(z)$, supernovae data, CMB shift parameters and BAO data. The model parameters are the reduced Hubble constant h, the present-day values of the physical density parameters of dark matter $\Omega_{c0}h^2$, baryons $\Omega_{b0}h^2$ and radiation $\Omega_{r0}h^2$. We will not consider perturbations, we are only constraining the parameters at the background level. Since we are using supernovae data we must include a nuisance parameter M, which represents a shift in the absolute magnitudes of the supernovae. Use of the full JLA catalogue requires the inclusion of the nuisance parameters α, β and ΔM from the light-curve fit. The first section of `.ini` is required to specify the `type` of the model, whether to use physical density parameters or not, and which species has the density parameter derived from the others (e. g. from the flatness condition):

```
[model]
type = lcdm
physical = yes
optional species = ['baryons', 'radiation']
derived = lambda
```

The `lcdm` model will always have the two species `cdm` and `lambda`. We are including the optional baryonic fluid and radiation, which being a `combined species` replaces `photons` and `neutrinos`. The configurations and options available for each model are registered in the `EPIC/cosmology/model_recipes.ini` file. This section can still received the `interaction setup` dictionary to set the configuration of an interacting dark sector model. Details on this are given in the previous section *Interacting Dark Energy models*.

The second section defines the analysis: a label, datasets and specifications about the priors ranges and distributions. The optional property `prior distributions` can receive a dictionary with either `Flat` or `Gaussian` for each parameter. When not specified, the code will assume flat priors by default and interpret the list of two numbers as an interval prior range. When `Gaussian`, these numbers are interpreted as the parameters μ and σ of the Gaussian distribution. In the `simulation`

section, we specify the parameters of the diagonal covariance matrix to be used with the proposal probability distribution in the sampler. Values comparable to the expected standard deviation of the parameter distributions are recommended.

```
[analysis]
label = $H(z)$ + $H_0$ + SNeIa + BAO + CMB
datasets = {
    'Hz':   'cosmic_chronometers',
    'H0':   'HST_local_H0',
    'SNeIa': 'JLA_simplified',
    'BAO':   [
        '6dF+SDSS_MGS',
        'SDSS_BOSS_CMASS',
        'SDSS_BOSS_LOWZ',
        'SDSS_BOSS_QuasarLyman',
        'SDSS_BOSS_consensus',
        'SDSS_BOSS_Lyalpha-Forests',
        ],
    'CMB':   'Planck2015_distances_LCDM',
    }
priors = {
    'Och2' :  [0.08, 0.20],
    'Obh2' :  [0.02, 0.03],
    'h' : [0.5, 0.9],
    'M' : [-0.3, 0.3],
    }
prior distributions =
fixed = {
    'T_CMB' : 2.7255
    }

[simulation]
proposal covariance = {
    'Och2' : 1e-3,
    'Obh2' : 1e-5,
    'h' : 1e-3,
    'M': 1e-3,
    }
```

4.3 Running MCMC

This is the vanilla Monte Carlo Markov Chain with the Metropolis algorithm, as introduced in the previous section *The Metropolis-Hastings sampler*.

We will now proceed to run MCMC for the ΛCDM model. The `epic.py` script accepts four commands: `run`, `analyze`, `plot` and `monitor`. The commands must be issued from your terminal

at the EPIC directory.

Sampling the posterior distribution

Standard MCMC is the default option for sampling. We start a MCMC simulation with the run command:

```
$ python epic.py run LCDM.ini 12 10000 --sim-full-name MyFirstRun --check-
↪interval 6h
```

where the first argument is the .ini file, the second is the number of chains to be run in parallel and the third is the number of steps to be run in each MCMC iteration. This number of steps means that the chains will be written to the disk (the new states are appended to the chains files) after each steps states in all chains. A large number prevents frequent writing operations, which could impact overall performance unnecessarily. Each chain will create an independent process with Python's multiprocessing, so you should not choose a number higher than the number of CPUs multiprocessing.cpu_count() of your machine. The number of chains should not be less than 2.

If correlations in the initial proposal covariance matrix are needed or desired, the user can overwrite the diagonal proposal covariance matrix with the option --proposal-covariance followed by the name of the file containing the desired covariance matrix in a table format.

The program creates a directory for this new simulation. Inside this directory, another one named chains receives the files that will store the states of the chains, named chain_0.txt, chain_1.txt, etc. Other relevant files are equally named but stored in the folder current_states. These store only the last state of each chain, to allow fast resuming from where the chain stopped, without need to loading the entire chains. The chains start at points randomly sampled from the priors, unless --no-multi-start is given, in which case all chains start at the same point, with coordinates given by the default values of each parameter. A good starting point may help to obtain convergence faster, besides eliminating the need for burn-in[1]. The name of the folder is the date-time of creation (in UTC), unless the option --sim-full-name MyFirstRun is used, then MyFirstRun will be the folder name. A custom label or tag can also be prepended with the --sim-tag my_label option, for example, the folder my_label-171110-173000. It will be stored within another folder with the same name of the .ini file, thus in this case LCDM/MyFirstRun. The full path of the simulation will be displayed in the first line of the output.

An existing simulation can be resumed from where it last saved information to disk with the same command, just giving the path of the simulation instead of the .ini file name, and omitting the number of chains, which has already been defined in the first run, for example:

```
$ python epic.py run <FULL-OR-RELATIVE-PATH-TO>/LCDM/MyFirstRun/ 5000
```

Another important parameter is the tolerance accepted for convergence assessment. By default, the program will stop when the convergence check finds, according to the Gelman-Rubin method,

[1] When checking convergence with Gelman-Rubin method, however, burn-in is still applied.

convergence with $\epsilon < 0.01$. To change this value, you can use `--tolerance 0.002` or just `-t 2e-3`, for example. In the MCMC mode, the code will periodically check for convergence according to the Gelman-Rubin method (by default it is done every two hours but can be specified differently as `--check-interval 12h` or `-c 45min` in the arguments, for example. This does not need to (and should not) be a small time interval, but the option to specify this time in minutes or even in seconds (`30sec`) is implemented and available for testing purposes.

The relevant information will be displayed, in our example case looking similar to the following:

```
Simulation at <FULL-PATH-TO>/LCDM/MyFirstRun
Mode MCMC.
The following datasets will be used:
    6dF+SDSS_MGS
    HST_local_H0
    JLA_simplified
    Planck2015_distances_LCDM
    SDSS_BOSS_CMASS
    SDSS_BOSS_LOWZ
    SDSS_BOSS_Lyalpha-Forests
    SDSS_BOSS_QuasarLyman
    SDSS_BOSS_consensus
    cosmic_chronometers
Initiating MCMC...
```

and the MCMC will start.

The MCMC run stops if convergence is achieved with a tolerance smaller than the default or given `tolerance`.

The following is the output of our example after the MCMC has started. The 10000 steps take a bit more than thirty minutes in my workstation running 12 chains in parallel. The number of chains will not make much impact on this unless we use too many steps by iteration and work close to the machine's memory limit. After approximately six hours, convergence is checked. Since it is larger than our required `tolerance`, the code continues with new iterations for another six hours before checking convergence again and so on. When convergence smaller than `tolerance` is achieved the code makes the relevant plots and quits.

```
Initiating MCMC...
i 1, 10000 steps, 12 ch; 32m45s, Sat Mar 24 00:29:13 2018. Next: ~5h27m.
i 2, 20000 steps, 12 ch; 32m53s, Sat Mar 24 01:02:07 2018. Next: ~4h54m.
i 3, 30000 steps, 12 ch; 32m52s, Sat Mar 24 01:34:59 2018. Next: ~4h21m.
i 4, 40000 steps, 12 ch; 32m54s, Sat Mar 24 02:07:54 2018. Next: ~3h48m.
i 5, 50000 steps, 12 ch; 32m51s, Sat Mar 24 02:40:46 2018. Next: ~3h15m.
i 6, 60000 steps, 12 ch; 32m53s, Sat Mar 24 03:13:39 2018. Next: ~2h42m.
i 7, 70000 steps, 12 ch; 33m3s, Sat Mar 24 03:46:43 2018. Next: ~2h9m.
i 8, 80000 steps, 12 ch; 32m52s, Sat Mar 24 04:19:35 2018. Next: ~1h36m.
i 9, 90000 steps, 12 ch; 32m55s, Sat Mar 24 04:52:30 2018. Next: ~1h3m.
```

(continues on next page)

(continued from previous page)

```
i 10, 100000 steps, 12 ch; 32m52s, Sat Mar 24 05:25:23 2018. Next: ~31m3s.
i 11, 110000 steps, 12 ch; 32m46s, Sat Mar 24 05:58:10 2018. Checking now...
Loading chains...                                          [##########]  12/12
Monitoring convergence...                                  [##########]  100%
R-1 tendency: 7.993e-01, 8.185e-01, 7.745e-01
i 12, 120000 steps, 12 ch; 32m49s, Sat Mar 24 06:31:21 2018. Next: ~5h27m.
i 13, 130000 steps, 12 ch; 32m53s, Sat Mar 24 07:04:15 2018. Next: ~4h54m.
i 14, 140000 steps, 12 ch; 32m49s, Sat Mar 24 07:37:04 2018. Next: ~4h21m.
...
```

After the first loop, the user can inspect the acceptance ratio of the chains, updated after every loop in the section `acceptance rates` of the `simulation_info.ini` file. Chains presenting bad performance based on this acceptance rate will be discarded. The values considered as good performance are any rate in the interval from 0.1 to 0.5. This can be changed with `--acceptance-limits 0.2 0.5`, for example. If you want to completely avoid chain removal use `--acceptance-limits 0 1`.

Adaptation (beta)

When starting a new run, use the option `--adapt FREE [ADAPT [STEPS]]`, where `FREE` is an integer representing the number of loops of free random-walk before adaptation, `ADAPT` is the actual number of loops during which adaptation will take place, and `STEPS` is the number of steps for both, pre-adapting and adapting phases. Note that the last two are optional. If `STEPS` is omitted, the number of steps of the regular loops will be used (from the mandatory `steps` argument); if only one argument is given, it is assigned to both `FREE` and `ADAPT`. At least one loop of free random-walk is necessary, so the starting states can serve as input for the adaptation phase. The total number of states in the two phases will be registered in the `simulation_info.ini` file (`adaptive burn-in`) as the minimal necessary burn-in size if one wants to keep the chains Markovian. The adaptation is an iterative process that updates the covariance matrix $\mathbf{S}_t$ of the proposal function for the current t-th state based on the chains history and goes as follows. Let $\mathbf{\Sigma}_t$ be a matrix derived from $\mathbf{S}_t$, defined by $\mathbf{\Sigma}_t \equiv \mathbf{S}_t\, m/2.38^2$, where m is the number of free parameters. After the initial free random-walk phase, we calculate, for a given chain, the chain mean

$$\bar{\mathbf{x}}_t = \frac{1}{t}\sum_{i=1}^{t} \mathbf{x}_i,$$

where $\mathbf{x}_i = (x_{1,i}, \ldots, x_{m,i})$ is the i-th state vector of the chain. The covariance matrix for the sampling of the next state $t+1$ will then be given by

$$\mathbf{S}_{t+1} = e^{2\theta_{t+1}} \mathbf{\Sigma}_{t+1},$$

where $\theta_{t+1} = \theta_t + \gamma_t\,(\eta_t - 0.234)$, $\gamma_t = t^{-\alpha}$, θ_t is a parameter that we set to zero initially, α is a number between 0.5 and 1, here set to 0.6, η_t is the current acceptance rate, targeted at 0.234, and

$$\mathbf{\Sigma}_{t+1} = (1-\gamma_t)\,\mathbf{\Sigma}_t + \gamma_t\,(\mathbf{x}_t - \bar{\mathbf{x}}_t)^T\,(\mathbf{x}_t - \bar{\mathbf{x}}_t)\,.$$

In this program, the covariance matrix is updated based on the first chain and applied to all chains. This method is mostly based on the adaptation applied to the parallel tempering algorithm by Łącki & Miasojedow (2016)[2]. The idea is to obtain a covariance matrix such that the asymptotic acceptance rate is 0.234, considered to be a good value. Note, however, that this feature is in beta and may require some trial and error with the choice of both free random-walk and adaptation phases lengths. It might take too long for the proposal covariance matrix to converge and the simulation is not guaranteed to keep good acceptance rates with the final matrix obtained after the adaptation phase.

Analyzing the chains

Once convergence is achieved and MCMC is finished, the code reads the chains and extract the information for the parameter inference. But this can be done to assess the results while MCMC is still going on. The distributions are compiled for a nice plot with the command `analyze`:

```
$ python epic.py analyze <FULL-OR-RELATIVE-PATH-TO>/LCDM/MyFirstRun/
```

Histograms are generated for the marginalized distributions using 20 bins or any other number given with `--bins` or `-b`. At any time one can run this command, optionally with `--convergence` or `-c` to check the state of convergence. By default, this will calculate $\hat{R}^p - 1$ for twenty different sizes (which you can change with `--GR-steps`) considering the size of the chains to provide an idea of the evolution of the convergence.

Convergence is assessed based on the Gelman-Rubin criterium. I will not enter into details of the method here, but I refer the reader to the original papers[4][5] for more information. The variation of the original method for multivariate distribution is implemented. When using `MCMC`, all the chains are checked for convergence and the final resulting distribution which is analyzed and plotted is the concatenation of all the chains, since they are essentially all the same once they have converged.

Additional options

If for some reason you want to view the results for an intermediate point of the simulation, you can tell the script to `--stop-at 18000`, everything will be analyzed until that point.

If you want to check the random walk of the chains you can plot the sequences with `--plot-sequences`. This will make a grid plot containing all the chains and all parameters. Keep in mind that this can take some time and generate a big output file if the chains are very long. You can contour this problem by thinning the distributions by some factor `--thin 10`, for

[2] Łącki M. K. & Miasojedow B. "State-dependent swap strategies and automatic reduction of number of temperatures in adaptive parallel tempering algorithm". Statistics and Computing 26, 951–964 (2016).

[4] Gelman A & Rubin D. B. "Inference from Iterative Simulation Using Multiple Sequences". Statistical Science 7 (1992) 457-472.

[5] Brooks S. P. & Gelman A. "General Methods for Monitoring Convergence of Iterative Simulations". Journal of Computational and Graphical Statistics 7 (1998) 434.

example. This also applies for the calculation of the correlation of the parameters in each chain, enabled with the `--correlation-function` option.

We generally represent distributions by their histograms but sometimes we may prefer to exhibit smooth curves. Although it is possible to choose a higher number of histogram bins, this may not be sufficient and may required much more data. Much better (although possibly slow when there are too many states) are the kernel density estimates (KDE) tuned for Gaussian-like distributions[6]. To obtain smoothed shapes use `--kde`. This will compute KDE curves for the marginalized parameter distributions and also the two-parameter joint posterior probability distributions.

Making the triangle plots

The previous command will also produce the plots automatically (unless supressed with `--dont-plot`), but you can always redraw everything when you want, maybe you would like to tweak some colors? Loading the chains again is not necessary since the analysis already saves the information for the plots anyway. This is done with:

```
$ python epic.py plot <FULL-OR-RELATIVE-PATH-TO>/LCDM/MyFirstRun/
```

The code will find the longest chain analysis results and plot them. In this plot, you can, for example, choose the main color with `--color m` for magenta, or with any other Python color name, and omit the best-fit point mark with `--no-best-fit-marks`. The default color is `C0` (a shade of blue, from the default Matplotlib palette that ranges from `C0` to `C9`).

Plot ranges and eventually necessary factors of power of 10 are automatically handled by the code, unless you use `--no-auto-range` and `--no-auto-factors` or choose your own ranges or powers of 10 by specifying a dictionary with parameter label as keys and a pair of floats in a list or an integer power of 10 for the entries `custom range` and `custom factors` under the section `analysis` in the `.ini` file.

If you have used the option `--kde` previously in the analysis you need to specify it here too to make the corresponding plots, otherwise the histograms will be drawn.

The 1σ and 2σ confidence levels are shown and are written to the file `hist_table.tex` (and `kde_table.tex` if it is the case) inside the folder with the size of the chain preceeded by the letter `n`. These LaTeX-ready files can be compiled to make a nice table in a PDF file or included in your paper as you want. To view and save information for more levels[3], use `--levels 1 2 3 4 5`.

Additional options

You can further tweak your plots and tables. `--use-tex` will make the program render the plot using LaTeX, fitting nicely to the rest of your paper. You can plot Gaussian fits together with the

[6] Kristan M., Leonardis A. and Skocaj D. "Multivariate online kernel density estimation with Gaussian kernels". Pattern Recognit 44 (2011) 2630-2642.

[3] Choice limited up to the fifth level. Notice that high sigma levels might be difficult to find due to the finite sample sizes when the sample is not sufficiently large.

histograms (the Gaussian curve and the two first sigma levels), using `--show-gaussian-fits`, but probably only for the sake of comparison since this is not usually done in publications. `--fmt` can be used to set the number of figures to report the results (default is `5`). There is `--font-size` to choose the size of the fonts, `--show-hist` to plot the histograms together with the smooth curves when `--kde` is used, `--no-best-fit-marks` to omit the indication of the coordinates of the best-fit point, `--png` to save images in png format besides the pdf files.

If you have nuisance parameters, you can opt to not show them with `--exclude nuisance`. This option can actually take the label of any parameter you choose not to include in the plot.

Below we see the triangle plot of the histograms, with the default settings, in comparison with a perfected version using the smoothed distributions, the Python color `C9`, the LaTeX renderer, including the 3σ confidence level and excluding the nuisance parameter M.

Combining two or more simulations in one plot

You just need to run `epic.py plot` with two or more paths in the arguments. I illustrate this with two simulation for the same simplified ΛCDM model, with cold dark matter and Λ only, one with $H(z)$ and H_0 data, the other with the same data plus the simplified supernovae dataset from JLA. It is then interesting to plot both realizations together so we can see the effect that including a dataset has on the results:

```
$ python epic.py plot \
<FULL-OR-RELATIVE-PATH-TO>/HLCDM+SNeIa/H_and_SN/ \
<FULL-OR-RELATIVE-PATH-TO>/HLCDM/H_only/ \
--kde --use-tex --group-name comparison --no-best-fit-marks
```

You can combine as many results as you like. When this is done, the list of parameters will be determined from the first simulation given in the command by its path. Automatic ranges for the plots are determined from the constraints of the first simulation as well. Since different simulations might give considerably different results that could go outside of the ranges of one of them, consider using `--no-auto-range` or specifying custom intervals in the `.ini` file if needed. The `--group-name` option specifies the prefix for the name of the pdf file that will be generated. If you omit this option you will be prompted to type it. All other settings are optional.

A legend will be included in the top right corner using the labels defined in the `.ini` files, under the `analysis` section. The legend title uses the model name of the first simulation in the arguments. This is intended for showing, at the same time, results from different datasets with the same model.

Visualizing chain sequences and convergence

If you include the option `--sequences` with the `analyze` command you will get a plot like this

When monitoring convergence, the values of $\hat{R}^p - 1$ at twenty (or `--GR-steps`) different lengths for the multivariate analysis and separate one-dimensional analyses for each parameter are plotted

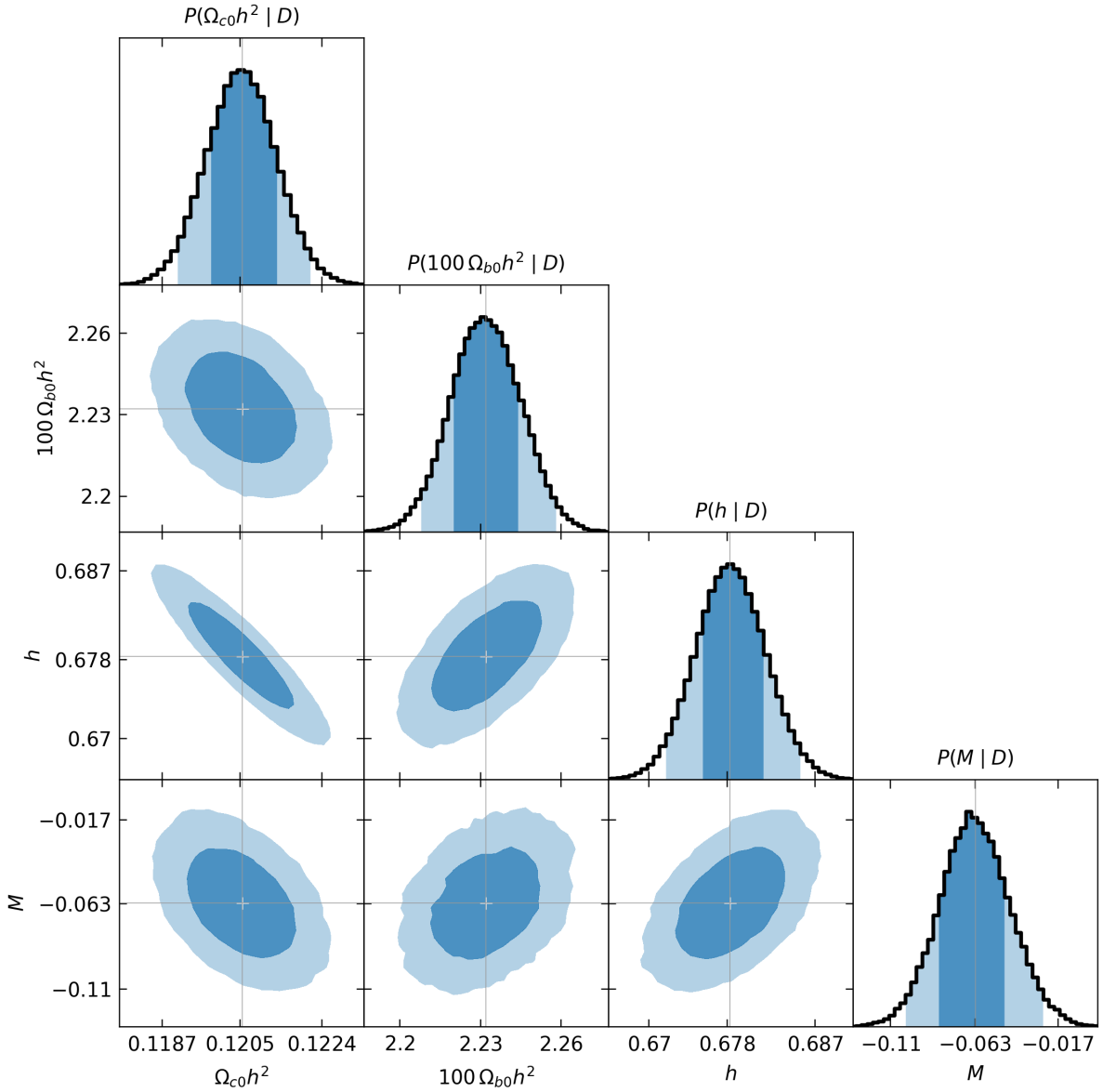


Fig. 1: Triangle plot with default configurations for histograms.

in the files `monitor_convergence_<N>.pdf` and `monitor_each_parameter_<N>.pdf`, where `N` is the total utilized length of the chains. The absolute values of the between-chains and within-chains variances, $\hat{V}$ and W are also shown. For our ΛCDM example, we got

To generate these plots, one needs first run the script with the command `analyze` and the `--convergence` option; then, run:

```
$ python epic.py monitor <FULL-OR-RELATIVE-PATH-TO>/LCDM/MyFirstRun
```

The first argument can be multiple paths to simulations, in which case the first plot above will

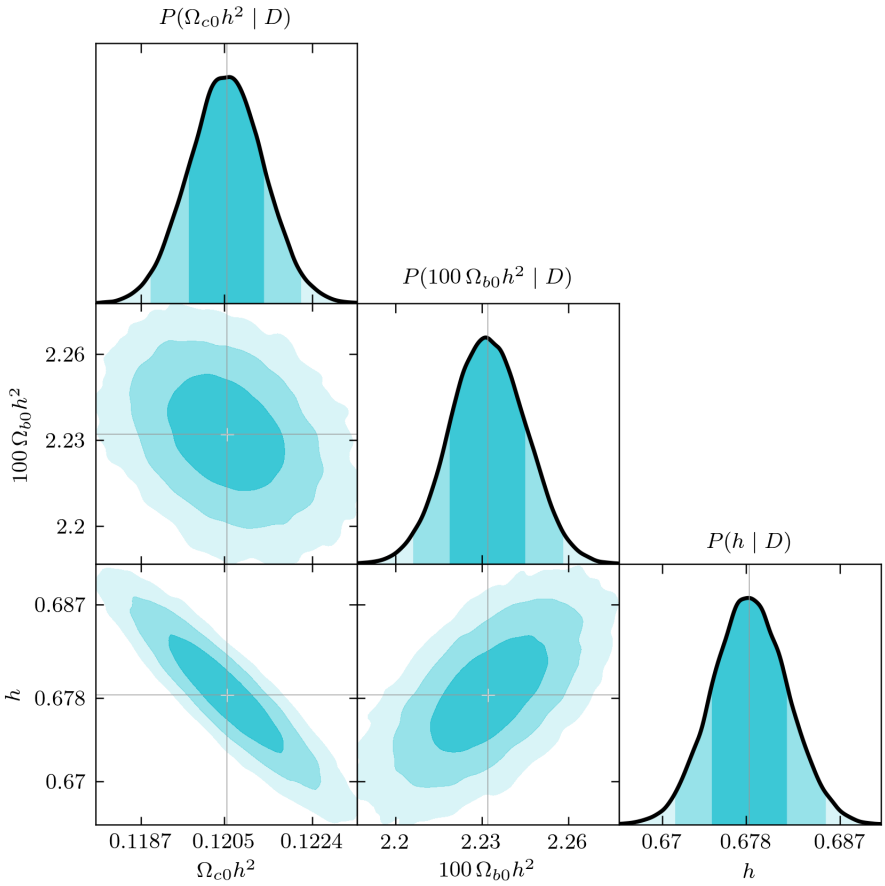


Fig. 2: Customized plot with smoothed distributions.

have panes for each simulation, one below each other. There are still the options `--use-tex` and `--png`, as with the `plot` command. Finally, the correlation of the chains can also be inspected if we use the option `--correlation-function`. This includes all the cross- and auto-correlations.

5 Acknowledgments

I want to thank Elcio Abdalla, Gabriel Marcondes and Luiz Irber for their help. This work has made use of the computing facilities of the Laboratory of Astroinformatics (IAG/USP, NAT/Unicsul), whose purchase was made possible by the Brazilian agency FAPESP (grant 2009/54006-4) and the INCT-A.

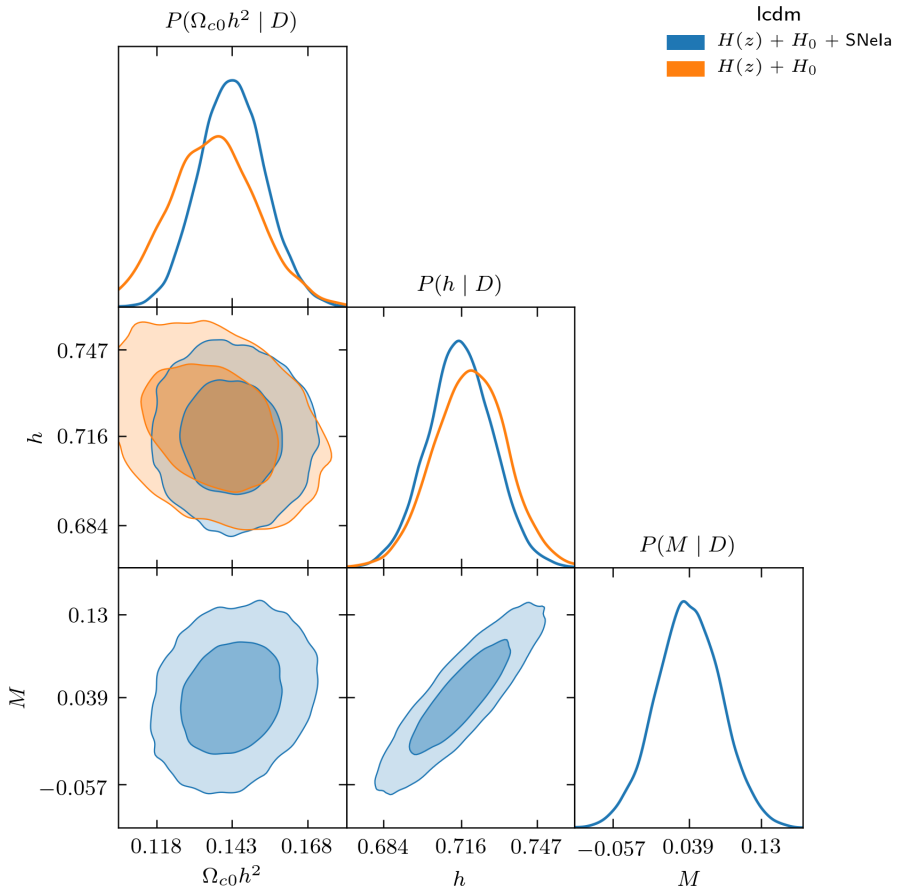


Fig. 3: Results for two different analyses.

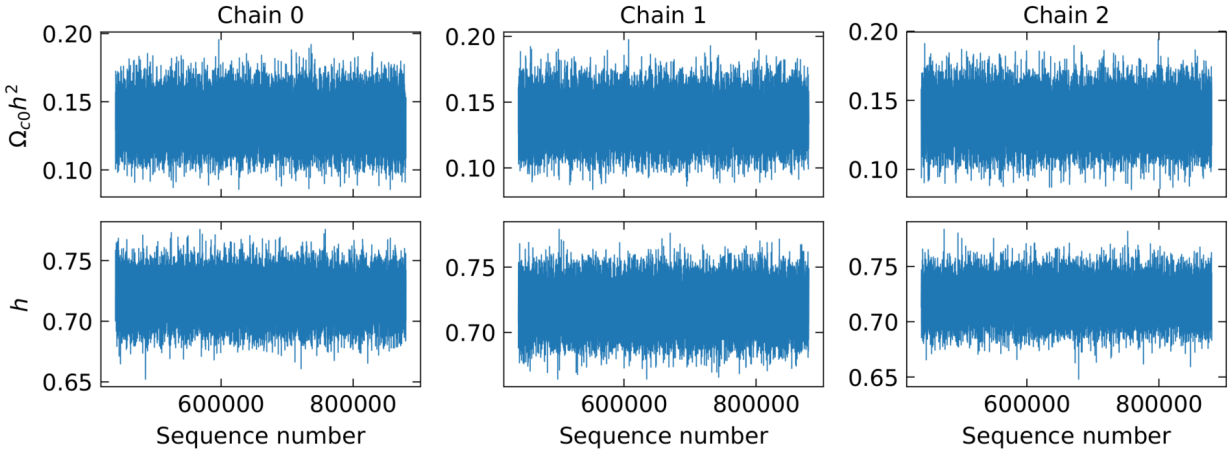


Fig. 4: Chain sequences along each parameter axis (only the first few chains shown here).

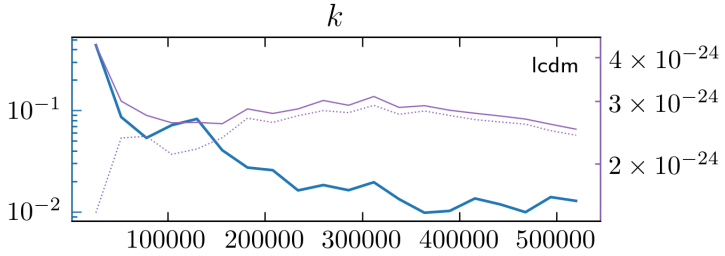


Fig. 5: Multivariate convergence analysis.

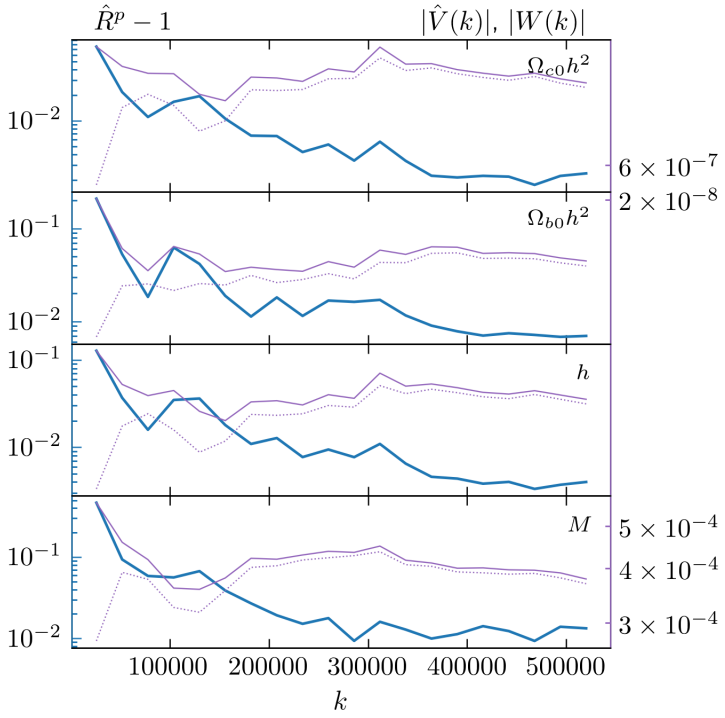


Fig. 6: Individual parameter convergence monitoring.

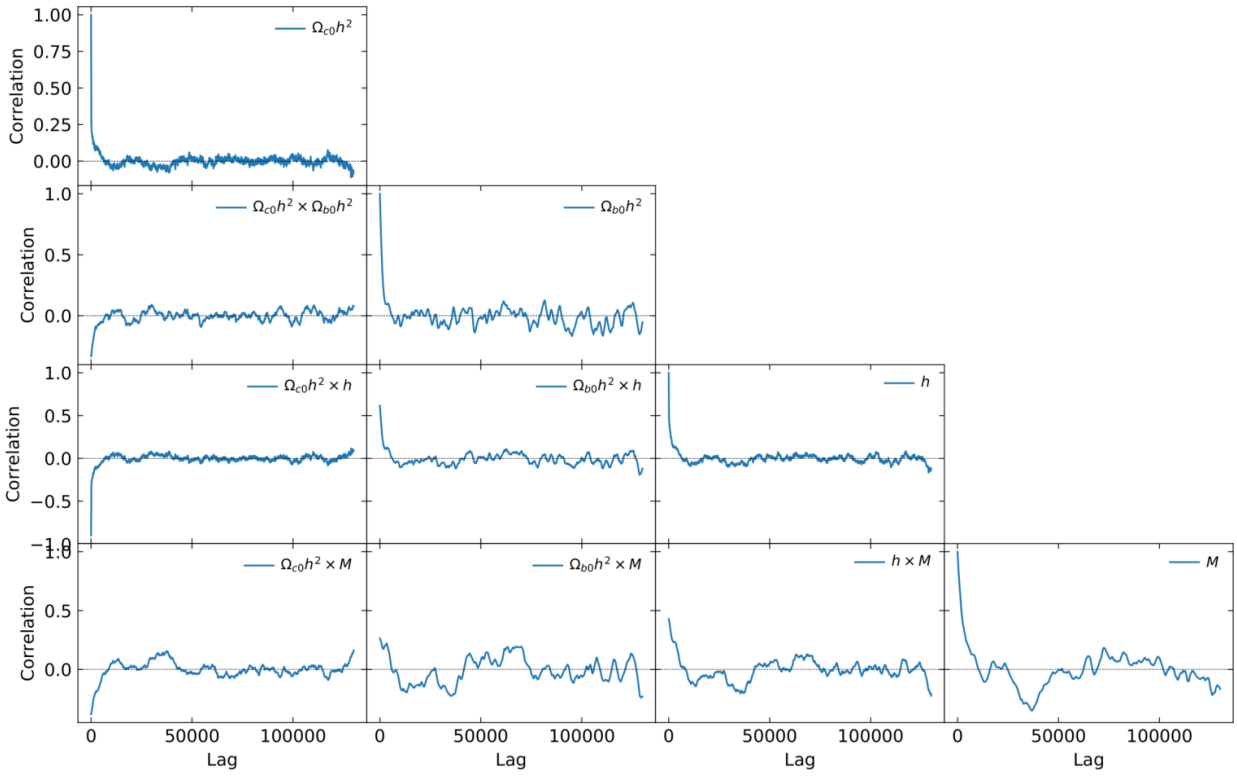


Fig. 7: Correlations in a chain (here using a factor `--thin 15`).