MCPostFit: a Python code to perform MC Posterior Fits of Markov chains and the subsequent sampling

Adrià Gómez-Valent[1, *] and Luca Amendola[1, †]

[1] *Institut für Theoretische Physik, Ruprecht-Karls-Universität Heidelberg, Philosophenweg 16, D-69120 Heidelberg, Germany*

In arXiv:2007.02615 [1] we developed a novel method, coined MonteCarlo Posterior Fit, to boost the MonteCarlo sampling of posterior functions. In these very short notes we provide some instructions to use `MCPostFit`, the Python code in which we have implemented the method. It fits an analytical multidimensional non-Gaussian distribution to the Markov chain introduced as an input. It also samples the resulting fitted distribution with a modified Metropolis-Hastings algorithm to obtain the final Markov chain from which to compute the marginalized one and two-dimensional histograms, and the corresponding confidence regions. The code has been publicly released, and is available at the GitHub.

I. INTRODUCTION

The Bayesian exploration of parameter spaces is a very important part of many cosmological studies aiming to extract constraints from observations in the context of particular cosmological models or to forecast the constraining power of future experimental setups. Since both the cosmological models and the data sets are increasing with time, there is a clear need of developing faster algorithms to sample the posterior distributions, i.e. a way of improving the standard MonteCarlo (MC) sampling processes. The latter can be quite time-consuming when the evaluation of the likelihood requires e.g. the solution of the full Einstein-Boltzmann equations or running N-body simulations.

In our paper [1] we presented a new method, called Montecarlo Posterior Fit, which improves the efficiency of the sampling of likelihoods found in the usual MC approaches. We use a multivariate non-Gaussian function to fit a much smaller number of sampling points than the one needed to reconstruct the posterior with the usual methods. The second-parameters that characterize the fitting function can be easily computed by solving a simple linear system of equations, and then a marginalization MC is needed to sample the fitted distribution and to subsequently obtain the 1D posteriors and 2D confidence regions for the various parameters of the model under study.

In these short notes we provide some guidelines for users of our Python [2] code, `MCPostFit`, to fit their own MonteCarlo Markov chains (MCMC) and sample the resulting fitted distribution. This guide does not pretend to explain the code line by line, but only the most important aspects of it, as e.g. the input files and variables needed to compile the program, and also provide a very general description of its main parts. The aim of this guide hence is purely practical. For an exhaustive description of the mathematics behind the code and more dedicated explanations about the method itself we refer the reader to [1].

Our code has been publicly released, and can be found in the following GitHub repository:

https://github.com/adriagova/MCPostFit

II. A PRACTICAL GUIDE TO MCPOSTFIT

This is the list of input files and variables that the user needs to introduce as arguments in the compilation command line, in the console:

- `MCPostFit` needs as input the .txt file that contains the MonteCarlo Markov chain previously generated with `MontePython`[1] [3] using the desired likelihood and priors for the various parameters. Notice that we can employ the criteria explained in Appendix C of [1] to decide when to stop the MontePython sampling process.
- The second argument the user has to provide is the total number of parameters that have been varied in the first MonteCarlo, i.e. the sum of cosmological plus nuisance parameters (no derived parameter has to be considered here). This sets the dimensions of our parameter space.

[*] gomez-valent@thphys.uni-heidelberg.de
[†] l.amendola@thphys.uni-heidelberg.de
[1] http://baudren.github.io/montepython.html

- We can fit the posterior distribution employing all the points contained in the chain or just a fraction of them, but the number of parameters that enter the fit (called "second-parameters" in [1]) has to be in any case lower or equal than the selected number of points in the MCMC because otherwise the problem is underdetermined [1]. We can specify the number of lines that we want to use in the third argument. This number has to be, of course, an integer and lower or equal to the total number of lines contained in the MCMC file. If we want to employ all of them we just have to write "full" in the third argument. This is actually the option that will be normally used in a real application, but we also allow the user to select subchains just for testing purposes.

- The user can also select the lines by order of appearance in the MCMC file, or pick them randomly, by writing "ordered" or "random", respectively, in the fourth argument. If we select the full chain (writing "full" in the third argument) there will be no difference between these two options. If we select a large enough number of points we expect both options to provide very similar results too, since the MC will have already explored a large volume of parameter space and the chain will have no memory about the starting point. Some differences can appear, though, when the typical steps in the MC are not sufficiently large and the selected number of points is small and lower than the full chain, since in this case the "ordered" option can contain points sampled only in a small region of parameter space and, therefore, we do not introduce as much information in the fitting analysis about the underlying distribution as with the "random" option. With the latter we pick points located in basically all the regions explored by the MonteCarlo that has been employed to obtain the full MCMC contained in the input file. Hence, the output of the fit in these cases is better. As mentioned before, in a real application we will normally employ the "full" option and hence we will not find any difference between "ordered" and "random", but we have added the second option again for testing purposes.

- As explained in [1], when we deal with a very high-dimensional parameter space it is sometimes convenient not to use the cubic and quartic corrections of the non-Gaussian fitting distribution associated to all the N parameters, but only to a subset of them. This is to ease the inversion of the matrix A of Eq. (37) of [1]. In the fifth argument the user can choose the number of parameters that are employed to compute not only up to the quadratic correction, but also the third and fourth order ones. They are taken in the same order of appearance of the MCMC file. This number has to be obviously an integer, and lower or equal than the total number of parameters.

- The number of sampling points generated in the marginalization MonteCarlo, making use of the fitted posterior distribution, is specified in the sixth argument. It is of course, an integer. Typically, one needs to produce a number of points ten times larger than the one contained in the original MCMC (cf. Secs. V and VI of [1])[2]. Notice that in the marginalization MonteCarlo there is no burn-in period, since we have configured the code to start the sampling process at the location of the best-fit found in the file that contains the original MCMC, which is already very close to the true peak of the posterior. This means that we do not need to remove any initial set of sampling points to avoid the typical bias introduced by the usual burn-in phase.

- In the marginalization MonteCarlo routine we employ a multivariate Gaussian as the proposal distribution to perform the jumps in the parameter space. We employ the covariance matrix generated by `MontePython`, and specify the name and location of this file in the seventh argument. Alternatively, we could have also configured our code to compute the covariance matrix directly from the selected MCMC, but this is not needed because `MontePython` already provides it.

Some additional comments are now in order:

– The code performs both the MC Posterior Gaussian and non-Gaussian fits. For the Gaussian case, `MCPostFit` generates two files in the output: (i) the location of the peak in the N-dimensional parameter space, in a file called "theta_corrected.txt"; and (ii) the corresponding covariance matrix, in a file called "Cov_matrix.txt". The names of these files and their output location can be modified in lines 428 and 429 of the code, respectively.

– In lines 480-482 we reduce the standard deviations of the N parameters by a factor 10 in order to shorten the steps in the MCMC and improve its efficiency [1]. This number can be, of course, changed by the user.

– Current lines 460-473 are only valid to process the covariance matrix that we provide in the example discussed below. A rescaling of some elements is in this particular case needed to match the units employed for some of

[2] As explained in [1], one could alternatively produce several chains of the same length and apply the Gelman-Rubin convergence criterion [4] to decide whether more chains are needed or not. This option has not been implemented in our code.

the parameters in the MCMC file with the ones employed in the file that contains the covariance matrix. If the user uses the same units/scaling in both files these lines can be deleted. If not, they can be modified and adapted to the user needs.

– Between lines 519 and 525 the code looks for the maximum of the fitted non-Gaussian posterior distribution. This is needed to implement the modified Metropolis-Hastings algorithm that allows to avoid the problems associated to the bad behavior of the fitted distribution in those regions of parameter space at which the density of sampling points obtained with `MontePython` is not high enough [1, 5, 6].

– In line 564 we save the generated MCMC in a .txt file. The name and location can be, again, modified. This file can later on be used to produce the marginalized one and two-dimensional histograms in all the planes of interest, and the corresponding confidence regions.

In our GitHub repository we provide, apart from the code `MCPostFit`, also the input files needed to reproduce the results reported in Sec. VI of [1] (cf. Fig. 5 therein), with the ΛCDM model with curvature and using the Planck 2018 TT,TE,EE+lowE likelihood [7]. That is, we provide the .txt files containing the MCMC and the covariance matrix employed in the marginalization sampling of the non-Gaussian fitted posterior distribution. Their names are "Omegak.txt" and "Omegak_cov.txt", respectively. These files were obtained in the output of `MontePython` [3], and using `CLASS`[3] [8] to compute the CMB temperature and polarization spectra needed to evaluate the likelihood. In this case we deal with 7 cosmological parameters, i.e. $(\omega_b, \omega_{cdm}, \ln(10^{10}A_s), H_0, \tau_{reio}, n_s, \Omega_k)$, and the 21 nuisance parameters that enter the Planck likelihood. We used 17 parameters (7 cosmological plus 10 nuisance) to build the non-Gaussian corrections of the fitted distribution accounting for the skewness and the kurtosis, cf. [1]. To produce the results shown in Fig. 5 of [1] (green curves) we used $1.7\cdot10^5$ sampling points from the first MCMC in the fitting analysis, which corresponds to the choice of the first 60000 lines of the MCMC file. We generated 10^6 points in the marginalization MonteCarlo, which were obtained in only ~ 5.5 hours of computational time (we used a computer with a processor Intel(R) Core(TM) i3-8350K CPU @ 4.00GHz).

The compilation line reads therefore:

```
python MCPostFit.py Omegak.txt 28 60000 ordered 17 1000000 Omegak_cov.txt
```

Here we assume that the user has saved the two text files in the same folder as `MCPostFit`.

III. CONCLUSIONS

In this document we provided some instructions to make use of our code, `MCPostFit`, in which we have implemented the MonteCarlo Posterior Fit method developed in [1]. The method itself does not rely on `CLASS`+`MontePython` and our code could be easily adapted to process the chains obtained with CAMB[4] and CosmoMC[5] [9] too, and actually also with any other program. Our code can be of course used in other analogous analyses, with different models and data, following the guidelines explained above. `MCPostFit` is very user friendly.

For any doubt on the use of the code or any interest in integrating it in e.g. the `MontePython` environment, please contact us.

Acknowledgements

AGV is funded by the Deutsche Forschungsgemeinschaft (DFG) - Project number 415335479.

[1] L. Amendola and A. Gómez-Valent, arXiv:2007.02615 [astro-ph.CO].
[2] G. Van Rossumn and F.L. Drake Jr, *Python reference manual.* Centrum voor Wiskunde en Informatica Amsterdam (1995).
[3] B. Audren, J. Lesgourgues, K. Benabed, and S. Prunet, JCAP **1302**, 001 (2013), arXiv:1210.7183 [astro-ph.CO].
[4] A. Gelman A. and D. Rubin, Statist. Sci. **7**, 457 (1992).

[3] http://lesgourg.github.io/class public/class.html
[4] https://camb.info/
[5] https://cosmologist.info/cosmomc/

[5] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, Journal of Chemical Physics **21**, 1087 (1953).
[6] W. Hastings, Biometrika **57**, 97 (1970).
[7] N. Aghanim *et al.* (Planck Collaboration), arXiv:1807.06209 [astro-ph.CO].
[8] D. Blas, J. Lesgourgues, and T. Tram, JCAP **1107**, 034 (2011), arXiv:1104.2933 [astro-ph.CO].
[9] A. Lewis and S. Bridle, Phys. Rev. D **66**, 103511 (2002), arXiv:astro-ph/0205436.