Tutorial - Parameter Estimation / MontePython

Jascha A. Schewtschenko

July 28, 2019

Abstract

In this tutorial, we will learn how to use the MCMC code `MontePython`. As a motivation, we are going to try to reproduce some of the results of a recent paper [Betoule et al.,2014] using supernovae data.

Exercise 0: Installing yet another missing library

We are missing one more python module for this exercise. Please execute the following command in your Linux environment before continuing:

```
pip install numexpr
```

Exercise 1: First run

In this first exercise, we are going to perform our first runs with MontePython. Our goal is to use the joint lightcurve analysis (JLA) dataset from SDSS-II and SNLS3 to constrain the parameters for ΛCDM.

(a) As a first step, we are setting a little workspace for us in our home directory

```
mkdir tutorial-montepython
cd tutorial-montepython
```

In general, you will have to write your own MontePython parameter file for each analysis you want to perform. Luckily for us, one for JLA and ΛCDM is already deployed in our MontePython installation alongside a few other examples in the `$HOME/montepython_public-3.1.0/input/` folder. So we copy this into our workspace

```
cp $HOME/montepython_public-3.1.0/input/jla.param
jla_lcdm.param
```

You may want to open the file with an editor to have a quick look at the settings. Which free parameters will we be using? What are their priors? What other parameters do you find in there? We also need a config file to run MontePython. Here, we can simply copy the existing cosmobox config file in the MontePython directory

```
cp $HOME/montepython_public-3.1.0/cosmobox.conf default.conf
```

(b) Now we are already ready to start the first run. We are going to run 4 chains (using MPI):

```
mpirun -np 4
$HOME/montepython_public-3.1.0/montepython/MontePython.py run
    -p jla_lcdm.param -o chains/jla_lcdm_testrun -N 1000
    >jla_lcdm_testrun.log
```

This shouldn't take more than a few minutes on your computer.Once this is done, we can analyse the chains:

```
$HOME/montepython_public-3.1.0/montepython/MontePython.py info
    chains/jla_lcdm_testrun
```

You can notice in the output that the stated $R-1$ values for the Gelman-Rubin convergence test are all well above 0.01 for any parameter. This would be unacceptable for a proper analysis run. Here we only used only a fraction of the steps you would normally use for an analysis, which is of course a major reason for the lack of convergence. But we also learned in the lectures, that the calibration of the proposal function can have a major impact.

(c) In order to improve the convergence, we can now use the info we got from our first poor run to better calibrate our proposal function by using the extracted covariance matrix and best-fits for the parameters as input for a second run (re-run the analysis again using `--want-covmat` to calculate the covariance matrix first).

```
mpirun -np 4
$HOME/montepython_public-3.1.0/montepython/MontePython.py run
  -p jla_lcdm.param -o chains/jla_lcdm_testrun_2 -N 1000 -c
     chains/jla_lcdm_testrun/jla_lcdm_testrun.covmat -b
      chains/jla_lcdm_testrun/jla_lcdm_testrun.bestfit
            >jla_lcdm_testrun_2.log
```

Run the analysis again on the new chains once they are done. What do you notice in the output now? (You can also check the generated log files `jla_lcdm_testrun.log` and `jla_lcdm_testrun.log` for a comparision of the acceptance ratios)

(d) Both runs we did so far were far too short to achieve a reasonable convergence. For some proper contraints we need at least about $N = 20000$. You could run this on your own computers, but it would exceed the time limit of this tutorial. Therefore, I ran this in advance using the second run as calibration and uploaded the chains for you. You can download and unpack them into your workspace

```
wget http://icg.port.ac.uk/~schewtsj/TPCosmoV/L3/jla_chains.tgz
tar xvfz jla_chains.tgz
```

You should now find these chains in `chains/jla_lcdm`. You can use them to run another analysis. What are the estimates for each parameter now?

(e) MontePython also gives you some freedom on deciding what to be plotted by the analysis and how. These customizations are stored in an additional file passed as an argument `--extra` to MontePython. We provide such a file here with the downloaded chains. So, try to run your final analysis instead with it:[1]

```
$HOME/montepython_public-3.1.0/montepython/MontePython.py info
chains/jla_lcdm --extra chains/jla_lcdm/jla_lcdm.extra --no-mean
```

Compare your results to those listed and plotted in the paper (Tab.10).

Exercise 2: Beyond ΛCDM

After our first successful parameter estimation for ΛCDM given JLA, we now want to go bit further and consider a more general model, namely o-ΛCDM, which is a one parameter extension allowing for non-zero spatial curvature Ω_k.

(a) This model is already fully implemented in the `CLASS` code, MontePython is running on. So the only thing, we need to do is to modify the parameter file to acccount for the new free parameter `Omega_k`. Do this by copying your existing parameter file

```
cp jla_lcdm.param jla_olcdm.param
```

and adding the required addition. You also have to add `Omega_Lambda` as a derived parameter (Hint: Assume upper or lower bounds, a 'sensible' mean and a proposal variance of 0.008)[2]

(b) Now follow the steps from Exercise 1 to obtain a calibrated prior and to get a full run (again, due to time limitations, you can skip the final long run and use the chains found in `chains/jla_olcdm` instead for your analysis. Compare your results and plots again to those found in the paper (use the customization file `chains/jla_olcdm/jla_olcdm.extra` for your analysis/plots).

[1] We also added another argument to the command to prevent MontePython from plotting the mean likelihood per bin on top of the marginalized likelihood plots as well.

[2] In case you really get stuck here, you can find a solution for the parameter file in `chains/jla_olcdm/jla_olcdm.param`, but please try to solve it yourself first.