MCMC-GALPROP57 Interface Documentation

Zachary Dorris*

Updated September 22 2024

THIS VERSION OF THE USER GUIDE IS INCOMPLETE. Updataed manual to be included shortly.

0 Acknowledgements

Special thanks to the cosmic-ray physics student research group at the University of Maryland, College Park for their support of this project.

1 Introduction

The Monte Carlo Markov Chain (MCMC) sampling algorithm is a robust and efficient method to automatically fit a model to experimental data by optimizing parameters and estimating their variance[1, references therein]. Various versions of the MCMC sampling algorithm are implemented in libraries and modules across several languages, for example in the python-based Cobaya[1] library which was developed from an older Fortran-based sampling library called CosmoMC [2, 3, 4]. MCMC sampling has been applied successfully to many applications in both cosmology and cosmic-ray physics, and in particular it has been applied to parameter selection for the cosmic-ray propagation code GALPROP [5, 6]. This document describes an interface I've developed that allows for Cobaya and GALPROP v57 [7] to be connected to allow for MCMC sampling of parameters in the GALPROP model. For a discussion of GALPROP and how it works see [8, 9].

2 Prerequisites

In order to run the interface, you must have a local copy of GALPROP v57 installed. You must also have Python 3.8 or newer and the libraries Cobaya, astropy, and numpy. Assuming you have Python already, Cobaya is pip-installable - see https://cobaya.readthedocs.io/en/latest/installation.html for details. Instructions for installing GALPROP v57 can be found at galprop.stanford.edu. Astropy and numpy are similarly pip-installable, though often these are already installed alongside Python.

3 Basic Use

The basic use of this interface can be broken into three steps; setting the file path configurations is described in section 3.1, adding and defining parameters via yaml files is described in section 3.2,

*zachattack@ecmtuning.com

[1] https://cobaya.readthedocs.io/en/latest/index.html

the format of the input data file is described in 3.3, and running the interface and receiving the results is described in section 3.4.

3.1 Setting the File Path Configuration

[TODO]

3.2 Defining Parameters

In Cobaya, parameters are most conveniently defined and setup using a yaml file. See their documentation for more information. This interface uses two yaml files, one to store information that Cobaya will directly use (which is in the exact same format as described in the Cobaya documentation) and a second to store information on how to input the parameters into GALPROP. For example, suppose we want to fit a diffusion-reacceleration model to the B/C ratio. We might have the parameters $(D_0, \delta, v_{Alfvén})$. We can create a yaml file for the interface that looks like this:

```
D_0:
    galpName: D_0xx
    type: galdef
delta_1:
    galpName: D_g_1
    type: galdef
v_a:
    galpName: v_Alfven
    type: galdef
```

These parameters are assigned the type "galdef", meaning that the parameters are found within the galdef file (as opposed to the source class files, etc.). The "galpName" field is used to tell the interface what the parameter is named inside of the galdef file, meaning that the name of the parameter used in Cobaya can be different from the name of the parameter in GALPROP. These three parameters would then also need to be defined, using the same top-level names "D_0, delta_1, v_a", in the main Cobaya yaml file to give them priors, reference pdfs, proposal widths, etc.

There are five types of parameters supported by default; None, galdef, source, modulation_FF, and Xsec_prod_norm. Galdef parameters exist in galdef files; source parameters exist inside source class files; modulation_FF parameters, standing for force-field approximation parameters, allow for modulation of spectra using the force-field approximation [10]; Xsec_prod_norm parameters allow for production cross sections, also called partial cross sections, of some secondary production channels to be renormalized using a multiplicative factor. Parameters with the type None are generally used as part of dependencies (described further below), and are not directly entered into any file.

Galdef parameters: All galdef parameters have a galpName and a type. Typically this is all that is needed, though the position field described below for source parameters can be used.

Source parameters: All source parameters have a galpName and a type. Additionally, source parameters - particularly injection spectrum parameters - can have a "position" field. This position, zero-indexed, allows for array-based entry of parameters. So for example, when assigning the default injection spectrum, spectral_pars, we could have a one-break un-smoothed power law using an interface yaml file like

```
gamma_1:
    galpName: spectral_pars
    type: source
    position: 0
```

R_1:
    galpName: spectral_pars
    type: source
    position: 1
gamma_2:
    galpName: spectral_pars
    type: source
    position: 2

This will enter the line "spectral_pars = {gamma_1} {R_1} {gamma_2}" into the source class file. By default, source parameters are entered into the main source class file from the configuration file, but if multiple source classes are being used then the field "src_file: filepath" can be used to specify to which file a source parameter should be entered.

Modulation_FF parameters: These parameters have a type, but no galpName since GALPROP does not handle modulation calculations. These parameters can be assigned an "expnames" field in the form of a list to specify which experiments should be modulated with this parameter. This allows for multiple modulation levels as desired. expnames should match exactly to how they are entered in the data input file. For example, one parameter for the AMS-02 modulation level can have "expnames: [AMS-02]" and another parameter for the ACE-CRIS and ACE-SIS modulation level can have "expnames: [ACE-CRIS, ACE-SIS]", assuming these are exactly how the experiment names appear in the data input file.

Xsec_prod_norm parameters: [TODO]

A parameter of any type can be marked as a nuisance parameter using the field "nuisance: true" in the yaml file. If this field is used, the fields "mu: (number)" and "sigma: (number)" must also be used to indicate the "default", or null hypothesis, value of this parameter and its variance respectively. There will be an added term to the final chi-squared, calculated in the logp function, equal to

$$\chi^2_{nuis} = (\frac{value - \mu}{\sigma})^2 \tag{1}$$

This additional term slightly biases the sampler in favor of using the null hypothesis, or default, values. It is most common to apply this label to modulation or cross-section parameters, but it can be applied to any type.

Parameters of any type can also have dependencies, where the value of one parameter depends on the values of one or more other parameters. Suppose we are sampling parameters A and B, but we wish to enter into the galdef files parameter A and parameter C = A + B. Thus, C is not directly sampled, but depends on two sampled parameters. To accomplish this, we would use something like

A:
    galpName: a
    type: galdef
    dependents: [C]
B:
    type: None
    dependents: [C]
C:
    galpName: c
    type: galdef
    lambda: pvs['A']+pvs['B']

We assign to the sampled parameters a "dependents" field, which is a list containing all parameters that depend on this one. In this case, C depends on both A and B, so both get this field. Parameter A is entered into the galdef file, so it gets a type galdef. However, B is not directly entered into any file, so it gets a type of None. The dependent parameter, C, does not get a dependents field but rather a "lambda" field. This lambda field should contain a valid python expression that will be evaluated by the system function eval(). Other parameters should be referenced as shown above, using pvs['param_name']. This is beneficial for instance to study halo size variations, since we can use for example

```
H:
    galpName: z_max
    type: galdef
    dependents: [z_min]
z_min:
    galpName: z_min
    type: galdef
    lambda: -1*pvs['H']
```

to construct a symmetrical halo in the z-direction. It is also beneficial if a source injection spectrum is to be given to multiple isotopes, or if two source injection spectra are to be different by a constant, or if isotopic abundances for a given element are to be kept in a specific ratio to each other.

3.3 Data Input Format

Data should be input to the program in USINE format[2]. Briefly, USINE file format is a table of values separated by whitespace. Each row is a datapoint, unless it starts with "#" to indicate a comment. The columns go, from left to right,

1. Quantity name
2. Experiment name, no spaces
3. Unit: R, EKN, EK, and ETOT are supported
4. Mean energy axis value for the bin, GeV or GV units
5. Lower energy value for the bin *(currently unused in the interface)*
6. Upper energy value for the bin *(currently unused in the interface)*
7. Value, either unitless or num / (s sr m GeV)
8. Negative statistical error
9. Positive statistical error
10. Negative systematic error
11. Positive systematic error *(stat and sys errs added in quadrature)*
12. Paper reference *(currently unused in the interface)*

[2]https://dmaurin.gitlab.io/USINE/input_cr_data.html, section 5.4 details the input file format for USINE

13. Solar modulation phi value, MV units *(currently unused in the interface)*

14. Distance from the sun the data was taken at, AU units *(values $\geq$ 120 are assigned 'IS' as their modInfo, otherwise 'FF' is assigned)*

15. Time period of collection *(currently unused in the interface)*

16. Upper limit, 1 for yes and 0 for no

The quantity name is case-insensitive, and can be any of the chemical symbols for elements from Z = 1 to 28 (H-Ni). Additionally, "1H-bar" can be used for antiprotons, "electrons" and "positrons" can be used for negative electrons and positrons respectively, "allparticle" can be used to sum all species, and "subFe" can be used to sum all elements Z = 21-25. Individual isotopes can be referenced by placing a number before a chemical symbol, ie "10Be" for ^{10}Be. Sums are supported by placing + signs between symbols, ie "Sc+Ti+V". Ratios are supported using / signs, ie "B/C". Ratios of sums are also supported, but sums of ratios and compound ratios (ratios of ratios) are not. It is assumed there will be either zero or one division symbol(s), and everything to the left is the numerator and to the right is the denominator. This means for example "B/O / F/Si" is not valid, but you can use "7Be/9Be+10Be" or "positrons/electrons+positrons". Whitespace and parentheses should not be included.

3.4 Running the Interface

Once the Cobaya yaml file, the interface yaml file, and the configuration files are all set, the program can be run using the command

python3 path/to/MCMCInterfaceLibv3.py > output.txt 2>&1 &

making sure that all the file paths in the configuration file are relative to the directory you plan to run the script from. This outputs all the work into an output.txt file, which can be checked on from time to time to see the progress.

References

[1] C.P. Robert, "The Metropolis-Hastings Algorithm", arXiv:1504.01896v3 [stat.CO] (2016)

[2] A. Lewis and S. Bridle, "Cosmological parameters from CMB and other data: A Monte Carlo approach", PRD **66**, 103511 (2002)

[3] A. Lewis, "Efficient sampling of fast and slow cosmological parameters", PRD **87**, 103529 (2013)

[4] R. M. Neal, "Taking Bigger Metropolis Steps by Dragging Fast Variables", Technical Report No. 0411, Department of Statistics, University of Toronto (2004)

[5] M. J. Boschini *et al.*, "Inference of the Local Interstellar Spectra of Cosmic-Ray Nuclei Z $\leq$ 28 with the GalProp–HelMod Framework", ApJS **250** 27 (2020)

[6] G. Jóhannesson *et al.*, "BAYESIAN ANALYSIS OF COSMIC RAY PROPAGATION: EVIDENCE AGAINST HOMOGENEOUS DIFFUSION", ApJ **824** 16 (2016)

[7] T. A. Porter *et al.*, "The GALPROP Cosmic-ray Propagation and Nonthermal Emissions Framework: Release v57", ApJS **262** 30 (2022)

[8] I. Moskalenko and A. W. Strong, "Production and Propagation of Cosmic-Ray Positrons and Electrons", ApJ **493** 694 (1998)

[9] A. W. Strong and I. Moskalenko, "Propagation of Cosmic-Ray Nucleons in the Galaxy", ApJ **509** 212 (1998)

[10] L. J. Gleeson and W. I. Axford, "Solar Modulation of Galactic Cosmic Rays", ApJ **154** pp. 1011 (1968)

[11] L. Derome *et al.*, "Fitting B/C cosmic-ray data in the AMS-02 era: a cookbook", Astronomy & Astrophysics **627** A158 (2019)

A Additional Documentation of the Code

A.1 Future To-Do List

- Make compatible with 2- and 3-dimensional GALPROP runs (currently only 2-D is supported)

  Notes: only procFITSHeader() and readSpectraFITS() need to be changed

- Add HelMod support to the interface

  Notes: Need new case for solar_modulate() with a new modInfo type. Configuration file might also need some new items. If there are any free parameters associated with HelMod they'll need to be included as new types in setParameter() and GalpropLikelihood class, for instance

- Enhance the scope and variety of cross-section nuisance parameters

  Notes: Currently, only production cross-section channels present in the eval_iso_cs.dat file can be modified, and they can only be changed by a constant multiplicative factor. Additionally, production of isotopes A<4 are not affected by eval_iso_cs.dat even when data is present. Additionally, it only works if you use cross_section_option with kopt ending in 2 in GALPROP, since the final digit controls whether the eval file is considered. It would be nice to be able to play with total inelastic cross-sections and the low-mass channels, as well as supporting many cross-section options, but this is not as simple as editing a file to shove a new value in each iteration. Modifications to GALPROP are required that pass the calculated cross-sections for each channel through the interface first before the calculation of the spectra proceeds. It would also be nice to support the full nuisance parameterizations described in [11, Sec. 4.2.1]. This would be a fairly easy task, only needing small changes to setParameter().

- Extend the functionality to allow for comparisons to diffuse emissions (γ-ray and radio wave) data

  Notes: More research is needed to figure out how to best support this.

- Allow for neural network acceleration of evaluation of likelihood function, and use linear scaling to improve the speed when using source abundance parameters [6]

Notes: More research is needed to figure out how to best support this. Likely it's a fairly straightforward addition to the GalpropLikelihood.logp() and GalpropDriver.calculate() functions, but such a "straightforward addition" could have lots of implementation edge cases and hyperparameters to tune.