MICS

Release 0.2.0

Feb 03, 2019

Contents

CHAPTER 1

Overview

Mixtures of Independently Collected Samples

- Free software: MIT license

1.1 Installation

```
pip install mics
```

1.2 Documentation

https://mics.readthedocs.io/

1.3 Development

To run the all tests run:

```
tox
```

Note, to combine the coverage data from all the tox environments run:

Windows	`set PYTEST_ADDOPTS=--cov-append` `tox`
Other	`PYTEST_ADDOPTS=--cov-append tox`

CHAPTER 2

Installation

At the command line:

```
pip install mics
```

CHAPTER 3

Usage

To use MICS in a project:

```
import mics
```

CHAPTER 4

Python API

4.1 sample

class mics.**sample**(*dataset*, *potential*, *acfun=None*, *batchsize=None*, ***constants*)

A sample of configurations distributed according to a *PDF* proportional to `exp(-u(x))`. Each configuration `x` is represented by a set of collective variables from which one can evaluate the reduced potential `u(x)`, as well as other properties of interest.

Parameters

- **dataset** (*pandas.DataFrame*) – A data frame whose column names are collective variables used to represent the sampled comfigurations. The rows must contain a time series of these variables, obtained by simulating the system at a state with known reduced potential.
- **potential** (*str*) – A mathematical expression defining the reduced potential of the simulated state. This must be a function of the column names in *dataset* and can also depend on external parameters passed as keyword arguments (see below).
- **acfun** (*str, optional, default=potential*) – A mathematical expression defining a property to be used for *OBM* autocorrelation analysis and effective sample size calculation. It must be a function of the column names in *dataset* and can also depend on external parameters passed as keyword arguments (see below).
- **batchsize** (*int, optional, default=sqrt(len(dataset))*) – The size of each batch (window) to be used in the *OBM* analysis. If omitted, then the batch size will be the integer part of the square root of the sample size.
- ****constants** (*keyword arguments*) – A set of keyword arguments passed as name=value, aimed to define external parameter values for the evaluation of the mathematical expressions in *potential* and *acfun*. They can also be used as labels to distinguish samples from each other, in this case not necessary being present in the mentioned expressions.

averaging(*properties*, *combinations={}*, ***constants*)

Computes averages and uncertainties of configurational properties. In addition, computes combinations among these averages while automatically handling uncertainty propagation.

Parameters

- **properties** (*dict(str: str)*) – A dictionary associating names to mathematical expressions. This is used to define functions of the collective variables included in the samples. Then, averages of these functions will be evaluated at all sampled states, along with their uncertainties. The expressions might also depend on parameters passed as keyword arguments (see below).
- **combinations** (*dict(str: str), optional, default={}*) – A dictionary associating names to mathematical expressions. This is used to define functions of the names passed as keys in the *properties* dictionary. The expressions might also depend on parameters passed as keyword arguments (see below).
- ****constants** (*optional keyword arguments*) – A set of arguments passed as `name=value`, used to define parameter values for evaluating the mathematical expressions in both *properties* and *combinations*.

Returns *pandas.DataFrame* – A data frame containing the computed averages and combinations, as well as their estimated standard errors.

subsampling(*integratedACF=True*)

Performs inline subsampling based on the statistical inefficiency `g` of the specified attribute *acfun* of *sample*, aiming at obtaining a sample of *IID* configurations. Subsampling is done via jumps of varying sizes around `g`, so that the sample size decays by a factor of approximately `1/g`.

Parameters **integratedACF** (*bool, optional, default=True*) – If true, the integrated *ACF* method *[2]* will be used for computing the statistical inefficiency. Otherwise, the *OBM* method will be used instead.

Returns *sample* – Although the subsampling is done inline, the new sample is returned for chaining purposes.

4.2 pooledsample

class mics.**pooledsample**(*iterable=0*)

A python list, but with special extensions for dealing with collections of *sample* objects. For instance, *subsampling()* and *averaging()* can be called for all samples simultaneously. There is also a method for creating a *mixture* object directly from a pooledsample.

averaging(*properties*, *combinations={}*, ***constants*)

Calls *averaging()* for all samples in the list.

Parameters Same as in *sample.averaging()*.

Returns *pandas.DataFrame* – A data frame containing the computed averages and combinations, as well as their estimated standard errors, for all samples.

histograms(*property='potential'*, *bins=100*, ***constants*)

mixture(*engine*)

Generates a *mixture* object.

Parameters **engine** (*MICS* or *MBAR*)

Returns *mixture*

subsampling(*integratedACF=True*)

Calls *subsampling()* for all samples in the list.

Parameters Same as in *sample.subsampling()*.

Returns *pooledsample* – Although the subsampling is done in line, the new pooled sample is returned for chaining purposes.

4.3 mixture

class mics.**mixture**(*samples*, *engine*)

A mixture of independently collected samples (MICS).

Parameters

- **samples** (*pooledsample* or list(*sample*)) – A list of samples.
- **engine** (*MICS* or *MBAR*) – A method for mixture-model analysis.

free_energies(*reference=0*)

Computes the free energies of all sampled states relative to a given reference state, as well as their standard errors.

Parameters **reference** (*int, optional, default=0*) – Specifies which sampled state will be considered as a reference for computing free-energy differences.

Returns *pandas.DataFrame* – A data frame containing the free-energy differences and their computed standard errors for all sampled states.

reweighting(*potential*, *properties={}*, *derivatives={}*, *combinations={}*, *conditions={}*, *reference=0*, ***constants*)

Computes averages of specified properties at target states defined by a given reduced *potential* function with distinct passed parameter values, as well as the free energies of such states with respect to a sampled *reference* state. Also, computes derivatives of these averages and free energies with respect to the mentioned parameters. In addition, evaluates combinations of free energies, averages, and derivatives. In all cases, uncertainty propagation is handled automatically by means of the delta method.

Parameters

- **potential** (*str*) – A mathematical expression defining the reduced potential of the target states. It might depend on the collective variables of the mixture samples, as well as on external parameters whose values will be passed via *conditions* or *constants*, such as explained below.
- **properties** (*dict(str: str), optional, default={}*) – A dictionary associating names to mathematical expressions, thus defining a set of properties whose averages must be evaluated at the target states. If it is omitted, then only the relative free energies of the target states will be evaluated. The expressions might depend on the same collective variables and parameters mentioned above for *potential*.
- **derivatives** (*dict(str: (str, str)), optional, default={}*) – A dictionary associating names to (property, parameter) pairs, thus specifying derivatives of average properties at the target states or relative free energies of these states with respect to external parameters. For each pair, property must be either "f" (for free energy) or a name defined in *properties*, while parameter must be an external parameter such as described above for *potential*.
- **combinations** (*dict(str: str), optional, default={}*) – A dictionary associating names to mathematical expressions, thus defining combinations among average properties at the target states, the relative free energies of these states, and their derivatives with respect to external parameters. The expressions might depend on "f" (for free energy) or on the names defined in *properties*, as well as on external parameters such as described above for *potential*.

- **conditions** (*pandas.DataFrame or dict, optional, default={}*) – A data frame whose column names are external parameters present in mathematical expressions specified in arguments *potential*, *properties*, and *combinations*. The rows of the data frame contain sets of values of these parameters, in such as way that the reweighting is carried out for every single set. This is a way of defining multiple target states from a single *potential* expression. The same information can be passed as a dictionary associating names to lists of numerical values, provided that all lists are equally sized. If it is empty, then a unique target state will be considered and all external parameters in *potential*, if any, must be passed as keyword arguments.
- **reference** (*int, optional, default=0*) – The index of a sampled state to be considered as a reference for computing relative free energies.
- ****constants** (*keyword arguments*) – A set of keyword arguments passed as name=value, aimed to define external parameter values for the evaluation of mathematical expressions. These values will be repeated at all target states specified via *potential* and *conditions*.

Returns *pandas.DataFrame* – A data frame containing the computed quantities, along with their estimated uncertainties, at all target states specified via *potential* and *conditions*.

4.4 MICS

class mics.**MICS**(*composition=None*, *tol=1e-12*)

Machinery for mixture-model analysis using the MICS method.

Parameters

- **composition** (*list(Number), optional, default = None*) – A predefined composition for the mixture. If this is None, then the prior probability of each state will be considered as proportional to the effective size of the corresponding sample.
- **tol** (*real, optional, default = 1e-12*) – A tolerance for determining convergence of the self-consistent solution of the MICS equations.

4.5 MBAR

class mics.**MBAR**(*tol=1e-12*)

Machinery for mixture-model analysis using the MBAR method *[1]*.

Parameters **tol** (*real, optional, default = 1e-12*) – A tolerance for determining convergence of the self-consistent solution of the MBAR equations.

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

5.1 Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.2 Documentation improvements

MICS could always use more documentation, whether as part of the official MICS docs, in docstrings, or even on the web in blog posts, articles, and such.

5.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/craabreu/mics/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that code contributions are welcome :)

5.4 Development

To set up *mics* for local development:

1. Fork mics (look for the "Fork" button).
2. Clone your fork locally:

```
git clone git@github.com:your_name_here/mics.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

4. When you're done making changes, run all the checks, doc builder and spell checker with tox one command:

```
tox
```

5. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

5.4.1 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`)[1].
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

5.4.2 Tips

To run a subset of tests:

```
tox -e envname -- py.test -k test_myfeature
```

To run all the test environments in *parallel* (you need to `pip install detox`):

```
detox
```

[1] If you don't have all the necessary python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.
It will be slower though ...

CHAPTER 6

Authors

- Charlles R. A. Abreu - http://atoms.peq.coppe.ufrj.br

CHAPTER 7

Changelog

0.2.0 (2018-05-09)

- Implementation of classes `sample`, `pool`, `mixture`, `MICS`, and `MBAR`.

0.1.0 (2017-10-11)

- Experimental release.

CHAPTER 8

Glossary

ACF Autocorrelation function

IID Independent and identically distributed

OBM Overlapping batch mean

PDF Probability density function

CHAPTER 9

Bibliography

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

Bibliography

[1] Michael R. Shirts and John D. Chodera. Statistically optimal analysis of samples from multiple equilibrium states. *The Journal of Chemical Physics*, 129(12):124105, September 2008. doi:10.1063/1.2978177.

[2] John D. Chodera, William C. Swope, Jed W. Pitera, Chaok Seok, and Ken A. Dill. Use of the Weighted Histogram Analysis Method for the Analysis of Simulated and Parallel Tempering Simulations. *Journal of Chemical Theory and Computation*, 3(1):26–41, January 2007. doi:10.1021/ct0502864.

Index