BICePs

Release 2.0

unknown

Jun 13, 2023

CONTENTS

CHAPTER

ONE

INSTALLATION

1.1 Dependencies

1.2 Testing Your Installation

Coming soon.

1.3 Versions

CHAPTER

TWO

THEORY

2.1 Bayesian inference

Bayesian inference approaches for modeling conformational ensembles generally seek to model a *posterior* distribution $P(X|D)$ of conformational states X, given some experimental data D. According to Bayes' theorem, the posterior distribution is proportional to a product of (1) a *likelihood* function $Q(D|X)$ of observing the experimental data given a conformational state X, and (2) a *prior* distribution $P(X)$ representing any prior knowledge about conformational states.

$$P(X|D) \propto Q(D|X)P(X)$$

Here, the prior $P(X)$ comes from theoretical modeling, while the likelihood $Q(D|X)$ corresponds to experimental restraints, typically in the form of some error model reflecting how well a given conformation X agrees with experimental measurements. One can think of $Q(D|X)$ as a reweighting factor for the population of each state X, and the BICePs algorithm can be thought of as a way to reweight conformational populations to agree with experimental knowledge.

In BICePs, the error model reflects both uncertainty in the experimental measurements and heterogeneity in the conformational ensemble, (i.e. how likely are the observables for conformation X to be away from the ensemble-average measurement, either due to experimental noise or conformational heterogeneity). These uncertainties are usually not known *a priori*, and must be treated as nuisance parameters σ which can be modeled using some prior model $P(\sigma)$:

$$P(X, \sigma|D) \propto Q(D|X, \sigma)P(X)P(\sigma)$$

Posterior sampling over X and σ using Markov Chain Monte Carlo (MCMC) can then be used to determine the conformational populations given the experimental restraints as $P(X|D) = \int P(X, \sigma|D)d\sigma$. Similarly, the posterior distribution of the experimental uncertainty $P(\sigma|D) = \int P(X, \sigma|D)dX$ gives information about how well the posterior conformational distribution agrees with the experimental restraints.

2.2 Reference potentials

The correct implementation of reference potentials is an importance advantage of the BICePs algorithm. The experimental data used to construct the likelihood function comes from some set of ensemble-averaged experimental observables $\mathbf{r} = (r_1, r_2, ..., r_N)$. Such observables are low-dimensional projections of some high-dimensional state space X, and therefore these restraints in the space of observables need to be treated as *potentials of mean force*.[1][2][3]

$$P(X|D) \propto \left[\frac{Q(\mathbf{r}(X)|D)}{Q_{\text{ref}}(\mathbf{r}(X))}\right] P(X)$$

[1] Olsson, S.; Frellsen, J.; Boomsma, W.; Mardia, K. V.; Hamelryck, T. Inference of Structure Ensembles of Flexible Biomolecules from Sparse, Averaged Data. PLoS One 2013, 8, e79439

[2] Olsson, S.; Boomsma, W.; Frellsen, J.; Bottaro, S.; Harder, T.; Ferkinghoff-Borg, J.; Hamelryck, T. Generative Probabilistic Models Extend the Scope of Inferential Structure Determination. J. Magn. Reson. 2011, 213, 182186.

[3] Hamelryck, T.; Borg, M.; Paluszewski, M.; Paulsen, J.; Frellsen, J.; Andreetta, C.; Boomsma, W.; Bottaro, S.; Ferkinghoff-Borg, J. Potentials of Mean Force for Protein Structure Prediction Vindicated, Formalized and Generalized. PLoS One 2010, 5, e13714.

The bracketed weighting function is now a ratio, with the numerator $Q(\mathbf{r}|D)$ being a likelihood function enforcing experimental restraints in the space of observables, while the denominator $Q_{\text{ref}}(\mathbf{r})$ reflects some reference distribution for possible values of the observables $\mathbf{r}$ in the absence of any experimental restraint information, such that $-\ln[Q(\mathbf{r}|D)/Q_{\text{ref}}(\mathbf{r})]$ is a potential of mean force. Without reference potentials, a great deal of unnecessary bias is introduced when many non-informative restraints are used.

As an example to illustrate the importance of reference potentials, consider an experimental distance restraint applied to two residues of a polypeptide chain (Figure 1). In the absence of any experimental information, we assume a reference potential $Q_{\text{ref}}(\mathbf{r})$ corresponding to the end-to-end distance of a random-coil polymer with a chain length equal to that of the intervening residues. For residues near each other along the chain, a short-distance restraint may have $[Q(\mathbf{r}|D)/Q_{\text{ref}}(\mathbf{r})] \sim 1$, contributing little or no information to refine the conformational ensemble. If the residues are far apart along the chain, however, a short-distance restraint can be highly informative, with $[Q(\mathbf{r}|D)/Q_{\text{ref}}(\mathbf{r})]$ greatly rewarding small distances where the reference potential $Q_{\text{ref}}(\mathbf{r})$ is small.

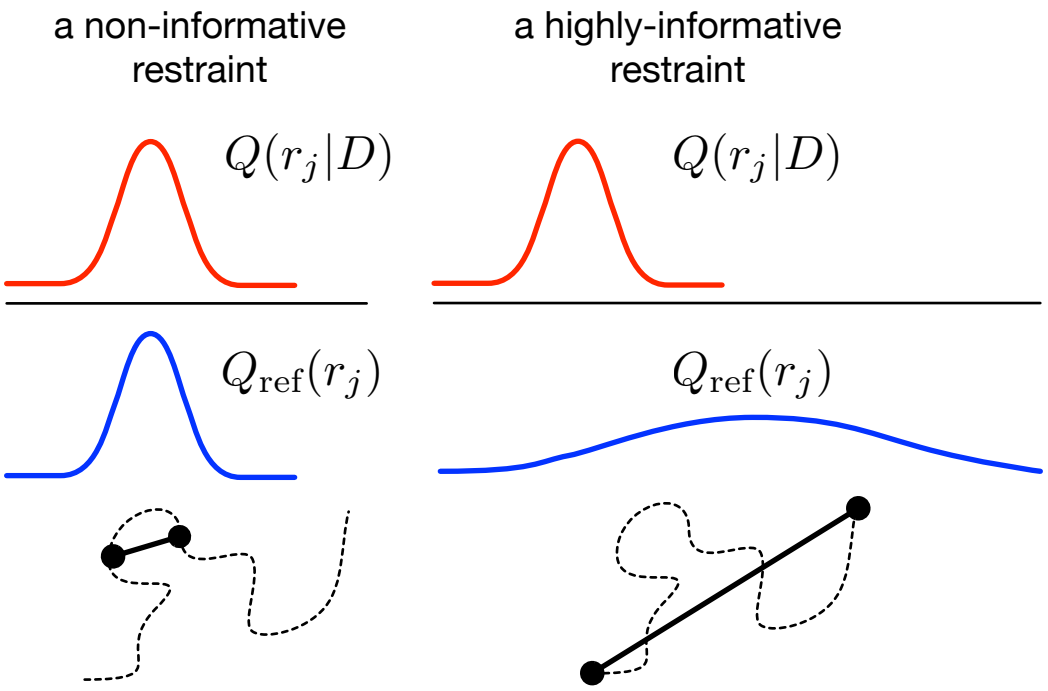

Fig. 1: Figure 1.

2.3 BICePs scores for quantitative model selection

Another key advantage of the BICePs algorithm is its the ability to use the sampled posterior probability function to perform quantitative model selection. This is done through a quantity we will call the *BICePs score*.

Consider K different theoretical models $P^{(k)}(X)$, $k = 1, ..., K$, that we may wish to use as our prior distribution. Perhaps these models come from simulations using different potential energy functions. Or perhaps we want to compare a model shaped only experimental restraints, to a model using both simulation and experimental information. Which model is more consistent with the experimental data? To determine this, we compute the posterior likelihood of the

model, $Z^{(k)}$, by integrating the k^{th} posterior distribution across all conformations X and values of σ:

$$Z^{(k)} = \int P^{(k)}(X, \sigma|D) dX d\sigma = \int P^{(k)}(X) Q(X) dX$$

total evidence for model $P^{(k)}$

overlap integral

The quantity $Z^{(k)}$ can be interpreted as the total evidence in favor of a given model. Equivalently, we can think of this term as an overlap integral between the prior $P^{(k)}(X)$ and a likelihood function $Q(X) = \int [Q(\mathbf{r}(X)|D, \sigma)/Q_{\text{ref}}(\mathbf{r}(X))] P(\sigma) d\sigma$. The overlap integral quantifies how well the theoretical modeling agrees with the experimental data (Figure 2); i.e. the value of $Z^{(k)}$ is maximal when $P^{(k)}(X)$ most closely matches the likelihood distribution $Q(X)$ specified by the experimental restraints.

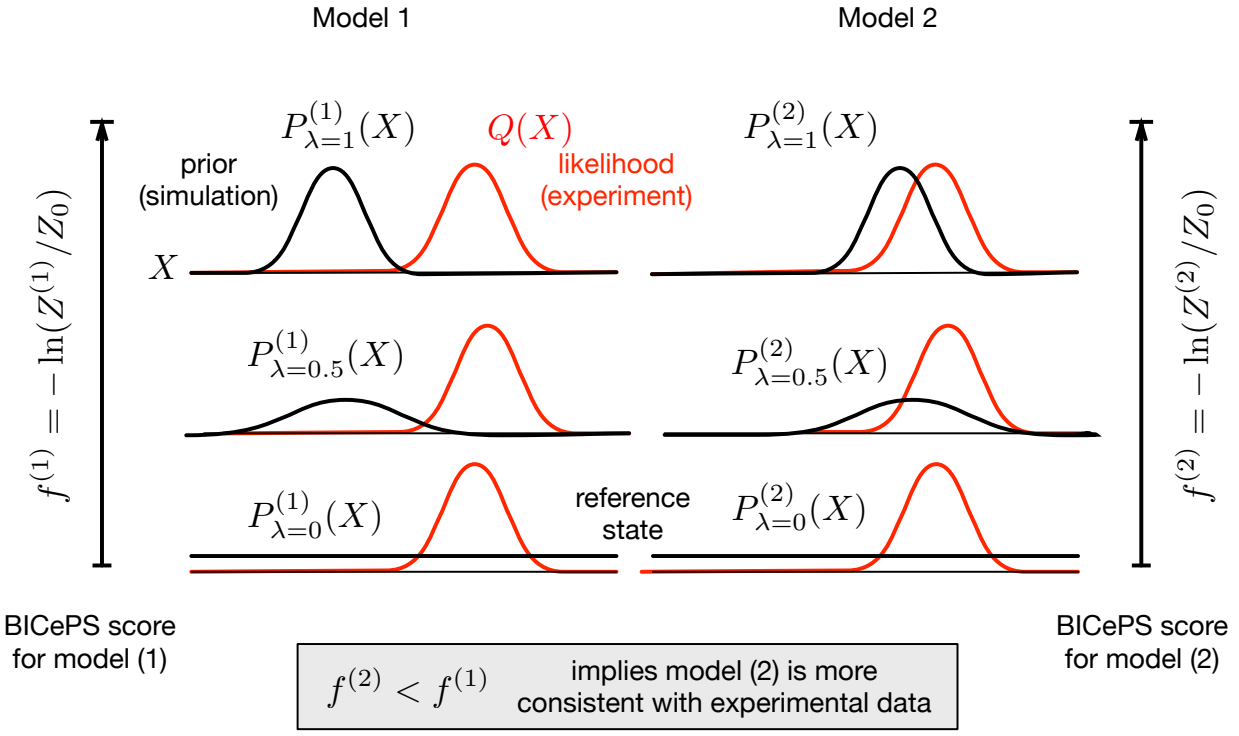

Fig. 2: Figure 2.

In Bayesian statistics, the so-called Bayes factor, $Z^{(1)}/Z^{(2)}$ is a likelihood ratio that can be used to choose between competing models (1) and (2), similar to a likelihood-ratio test in classical statistics. In order to assign each model a unique score, we define a free energy-like quantity,

$$f^{(k)} = -\ln \frac{Z^{(k)}}{Z_0}$$

to compare model $P^{(k)}(X, \sigma|D)$ against some reference model Z_0. In practice, we choose the reference model to be the posterior for a uniform prior $P(X)$, i.e. no information from theoretical modeling. We call $f^{(k)}$ the BICePs score. The BICePs score provides an unequivocal measure of model quality, to be used for objective model selection. The lower the BICePs score, the better the model (Figure 2). This useful property means that the BICePs score can be used a metric for force field validation and parameterization. In addition, the BICePs score has a useful physical interpretation: it reflects the improvement (or disimprovement) of the posterior distribution when going from a conformation ensemble shaped only experimental constraints, to a new distribution additionally shaped by a theoretical model.

2.4 Summary of the key advantages of BICePs.

To put the BICePs algorithm in a larger context, we summarize the its key advantages as follows:

- BICePs can be used with molecular dynamics (MD) or quantum mechanics (QM) methods. Conformational states can be individual conformations (like single-point QM minima) or collections of conformations (e.g. from clustering of trajectory data), as long as experimental observables can be attached to each conformational state.
- BICePs performs reweighting of conformational states derived from modeling; it is currently a post-processing algorithm (MCMC) with no additional MD or QM required.
- Bayesian inference offers a rigorous statistical framework for achieving the correct balance of theoretical modeling and experimental data.
- BICePs correctly uses reference potentials, which is essential to proper weighing of experimental restraints
- With proper reference potentials, BICePs scores can be used for unambiguous, objective model selection.

For more details about theory beneath BICePs, please check these work.[4][5]

2.5 References

[4] Voelz, V. A.; Zhou, G. Bayesian Inference of Conformational State Populations from Computational Models and Sparse Exper- imental Observables. J. Comput. Chem. 2014, 35, 22152224.

[5] Yunhui Ge and Vincent A. Voelz, Model selection using BICePs: A Bayesian approach to force field validation and parameterization Journal of Physical Chemistry B (2018) 122 (21): 5610–5622

CHAPTER

THREE

TUTORIALS & EXAMPLES

```
$ jupyter notebook
```

3.1 The workflow

3.1.1 Preparation

3.1.2 Ensemble

3.1.3 PosteriorSampler

3.1.4 Analysis

3.2 Additional tools

3.2.1 Convergence

3.2.2 Multiprocessing Lambda Values

3.2.3 Toolbox

3.3 Full examples

3.3.1 Cineromycin B

3.3.2 Albocycline

3.3.3 Apomyoglobin

CHAPTER

FOUR

API REFERENCE

4.1 Preparation

class biceps.Restraint.**Preparation**(*nstates=0*, *top_file=None*, *outdir=None*)

A class to prepare **input_data** for the *biceps.Ensemble.initialize_restraints* method.

Parameters

- **nstates** (*int*) – number of conformational states
- **top_file** (*str*) – relative path to the structure topology file
- **outdir** (*str*) – relative path for output files

4.2 Ensemble

class biceps.**Ensemble**(*lam*, *energies*, *debug=False*)

Container class for *biceps.Restraint.Restraint* objects.

Parameters

- **lam** (*float*) – lambda value to scale energies
- **energies** (*np.ndarray*) – numpy array of energies for each state

Ensemble.**initialize_restraints**(*input_data*, *options=None*)

Initialize corresponding *biceps.Restraint.Restraint* classes based on experimental observables from **input_data** for each conformational state.

Parameters

- **input_data** (*list of str*) – a sorted collection of filenames (files contain *exp* (experimental) and *model* (theoretical) observables)
- **options** (*list of dict*) – dictionary containing keys that match *biceps.Restraint.Restraint* parameters and values are lists for each restraint.

```
# In general:
parameters = [dict(**kwargs),...,dict(**kwargs)]
# More specifically, for J and NOE data restraints, respectively:
parameters = [dict(ref='uniform', sigma=(0.05, 20.0, 1.02)),
              dict(ref='exp', sigma=(0.05, 5.0, 1.02), gamma=(0.2, 5.0, 1.02))]
```

Tip: See the following parent *biceps.Restraint.Restraint* and child class methods for the full list of keyword arguments (**kwargs) for each restraint used inside *parameters*:

biceps.Restraint.Restraint_cs.init_restraint

biceps.Restraint.Restraint_J.init_restraint

biceps.Restraint.Restraint_noe.init_restraint

biceps.Restraint.Restraint_pf.init_restraint

Print possible restraints with: *biceps.toolbox.list_possible_restraints*

Print possible extensions with: *biceps.toolbox.list_possible_extensions*

Ensemble.**to_list**()

Converts the *Ensemble* class to a list.

Returns
collection of *biceps.Restraint.Restraint* objects

Return type
list

4.3 Restraint

class biceps.Restraint.**Restraint**(*ref='uniform'*, *sigma=[0.05, 20.0, 1.02]*, *use_global_ref_sigma=True*, *verbose=False*)

The parent *biceps.Restraint.Restraint* class.

Parameters

- **ref_pot** (*str*) – referenece potential e.g., "uniform". "exp", "gau". If None, the default reference potential will be used for a given experimental observable
- **sigma** (*list*) – (sigma_min, sigma_max, dsigma)
- **use_global_ref_sigma** (*bool*) – (defaults to True)

Restraint_cs.**init_restraint**(*data*, *energy*, *extension='H'*, *weight=1*, *file_fmt='pickle'*, *verbose=False*)

Initialize the chemical shift restraints for each experimental and theoretical observable given data.

Parameters

- **data** (*str*) – filename of data
- **energy** (*float*) – The (reduced) free energy of the conformation
- **extensions** (*str*) – "H", "Ca", "N"
- **weight** (*float*) – weight for restraint

Restraint_J.**init_restraint**(*data*, *energy*, *extension='J'*, *weight=1*, *file_fmt='pickle'*, *verbose=False*)

Initialize the sclar coupling constant restraints for each **exp** (experimental) and **model** (theoretical) observable given **data**.

Parameters

- **data** (*str*) – filename of data
- **energy** (*float*) – The (reduced) free energy of the conformation

- **weight** (*float*) – weight for restraint

Restraint_noe.**init_restraint**(*data*, *energy*, *extension='noe'*, *weight=1*, *file_fmt='pickle'*, *verbose=False*, *log_normal=False*, *gamma=[0.2, 10.0, 1.01]*)

Initialize the NOE distance restraints for each experimental and theoretical observable given data. When `log_normal=True`, the modified sum of squared errors is used $\chi^2_d(X) = \sum_j w_j(\ln(r_j(X)/\gamma' r_j^{exp}))^2$:param data: filename of data :type data: str :param energy: The (reduced) free energy $f = \beta * F$ of the conformation :type energy: float :param weight: weight for restraint :type weight: float :param log_normal: use log normal distribution :type log_normal: bool :param gamma: [gamma_min, gamma_max, dgamma] in log space :type gamma: list

Restraint_pf.**init_restraint**(*data*, *energy*, *precomputed=False*, *pf_prior=None*, *Ncs_fi=None*, *Nhs_fi=None*, *beta_c=(0.05, 0.25, 0.01)*, *beta_h=(0.0, 5.2, 0.2)*, *beta_0=(-10.0, 0.0, 0.2)*, *xcs=(5.0, 8.5, 0.5)*, *xhs=(2.0, 2.7, 0.1)*, *bs=(15.0, 16.0, 1.0)*, *extension='pf'*, *weight=1*, *file_fmt='pickle'*, *states=None*, *verbose=False*)

Initialize protection factor restraints for each **exp** (experimental) and **model** (theoretical) observable given **data**.

Parameters

- **data** (*str*) – filename of data
- **energy** (*float*) – The (reduced) free energy $f = \beta * F$ of the conformation
- **weight** (*float*) – weight for restraint
- **beta_c** (*list*) – [min, max, spacing]
- **beta_h** (*list*) – [min, max, spacing]
- **beta_0** (*list*) – [min, max, spacing]
- **xcs** (*list*) – [min, max, spacing]
- **xhs** (*list*) – [min, max, spacing]
- **bs** (*list*) – [min, max, spacing]

4.4 PosteriorSampler

class biceps.**PosteriorSampler**(*ensemble*, *freq_write_traj=100.0*, *freq_save_traj=100.0*, *verbose=False*)

A class to perform posterior sampling of conformational populations.

Parameters

- **ensemble** (*object*) – a *biceps.Ensemble* object
- **freq_write_traj** (*int*) – the frequency (in steps) to write the MCMC trajectory
- **freq_print** (*int*) – the frequency (in steps) to print status
- **freq_save_traj** (*int*) – the frequency (in steps) to store the MCMC trajectory

PosteriorSampler.**neglogP**(*states*, *parameters*, *parameter_indices*)

Return -ln P of the current configuration.

Parameters

- **state** (*list*) – the new conformational state being sampled in *PosteriorSampler.sample*
- **parameters** (*list*) – a list of the new parameters for each of the restraints

- **parameter_indices** (*list*) – parameter indices that correspond to each restraint

PosteriorSampler.**sample**(*nsteps*, *burn=0*, *print_freq=1000*, *verbose=False*, *progress=True*)

Perform n number of steps (nsteps) of posterior sampling, where Monte Carlo moves are accepted or rejected according to Metroplis criterion. Energies are computed via *neglogP*.

Parameters

- **nsteps** (*int*) – the number of steps of sampling
- **burn** (*int*) – the number of steps to burn
- **print_freq** (*int*) – the frequency of printing to the screen
- **verbose** (*bool*) – control over verbosity

Tip: Set *verbose=False* when using multiprocessing.

---

class biceps.**PosteriorSamplingTrajectory**(*ensemble*, *sampler*, *nreplicas*)

A container class to store and perform operations on the trajectories of sampling runs.

Parameters

- **ensemble** (*list*) – ensemble of *biceps.Restraint.Restraint* objects
- **nreplicas** (*int*) – number of replicas

PosteriorSamplingTrajectory.**process_results**(*filename=None*)

Process the trajectory, computing sampling statistics, ensemble-average NMR observables.

Benefits of using Numpy Z compression (npz) formatting: 1) Standardized Python library (NumPy), 2) writes a compact file of several arrays into binary format and 3) significantly smaller size over many other formats.

Parameters

filename (*str*) – relative path and filename for MCMC trajectory

---

Tip: It is possible to convert the trajectory file to a Pandas DataFrame (pickle file) with the following: *biceps.toolbox.npz_to_DataFrame*

4.5 Analysis

class biceps.**Analysis**(*outdir*, *nstates=0*, *precheck=True*, *BSdir='BS.dat'*, *popdir='populations.dat'*, *picfile='BICePs.pdf'*, *verbose=False*)

A class to perform analysis and plot figures.

Parameters

- **nstates** (*int*) – number of conformational states
- **trajs** (*str*) – relative path to glob '*.npz' trajectories (analysis files and figures will be placed inside this directory)
- **precheck** (*bool*) – find the all the states that haven't been sampled if any
- **BSdir** (*str*) – relative path for BICePs score file name
- **popdir** (*str*) – relative path for BICePs reweighted populations file name

- **picfile** (*str*) – relative path for BICePs figure

Analysis.**plot**(*plottype='hist'*, *figname='BICePs.pdf'*, *figsize=None*, *label_fontsize=12*, *legend_fontsize=10*)

Plot figures for population and sampled nuisance parameters.

Parameters

show (*bool*) – show the plot in Jupyter Notebook.

4.6 Convergence

biceps.convergence.**exponential_fit**(*autocorrelation*, *exp_function='single'*, *v0=None*, *verbose=False*)

Calls on `single_exp_decay` ('single') or `double_exp_decay` ('double') for an exponential fitting of an autocorrelation curve. See SciPy curve fit for more details.

Parameters

- **autocorrelation** (*np.ndarray*) – the autocorrelation of some timeseries
- **exp_function** (*str*) – default='single' ('single' or 'double'
- **v0** (*list*) – Initial conditions for exponential fitting. Default for 'single' is v0=[0.0, 1.0, 4000.]=[a0, a1, tau1] where $a_0 + a_1 * exp(-(x/au_1))$ and default for 'double' is v0=[0.0, 0.9, 0.1, 4000., 200.0]=[a0, a1, a2, tau1, tau2] where $f(x) = a_0 + a_1 * exp(-(x/au_1)) + a_2 * exp(-(x/au_2))$

Returns

the y-values of the fitted curve.

Return type

yfit(np.ndarray)

biceps.convergence.**compute_autocorrelation_curves**(*data*, *max_tau*, *normalize=True*)

Calculates the autocorrelation for a list of arrays, where each array is a separate time-series.

Parameters

- **data** (*list*) – list of separate timeseries
- **maxtau** (*int*) – the upper bound of autocorrelation lag time
- **normalize** (*bool*) – to normalize

Returns: np.ndarray

biceps.convergence.**g**(*f*, *max_tau=10000*, *normalize=True*)

Calculate the autocorrelaton function for a time-series f(t).

Parameters

- **f** (*np.ndarray*) – a 1D numpy array containing the time series f(t)
- **maxtau** (*int*) – the maximum autocorrelation time to consider.
- **normalize** (*bool*) – if True, return g(tau)/g[0]

Returns: np.array: a numpy array of size (max_tau+1,) containing g(tau)

biceps.convergence.**compute_autocorrelation_time**(*autocorrelations*)

Computes the autocorrelation time $\tau_{auto} = \int C_\tau d\tau$

Parameters
autocorrelations (*np.ndarray*) – an array containing the autocorrelations for each time-series.

Returns: np.ndarray

biceps.convergence.**get_blocks**(*data*, *nblocks=5*)

Method used to partition data into blocks. The data is a list of arrays, where each array is a separate time-series or autocorrelation.

Parameters
data (*list*) – list of separate timeseries

biceps.convergence.**compute_JSD**(*T1*, *T2*, *T_total*, *ind*, *allowed_parameters*)

Compute JSD for a given part of trajectory.

$JSD = H(P_{comb}) - \pi_1 H(P_1) - \pi_2 H(P_2)$, where P_{comb} is the combined data ($P_1 \cup P_2$). H is the Shannon entropy of distribution P_i and π_i is the weight for the probability distribution P_i. $H(P_i) = \sum -\frac{r_i}{N_i} * ln(\frac{r_i}{N_i})$, where r_i and N_i represents sampled times of a specific parameter index and the total number of samples of the parameter, respectively

Variables

- **T_total** (*T1, T2,*) – part 1, part2 and total (part1 + part2)
- **rest_type** – experimental restraint type
- **allowed_parameters** – nuisacne parameters range

Return float
Jensen–Shannon divergence

class biceps.**Convergence**(*traj=None*, *filename=None*, *outdir='./'*, *verbose=False*)

Convergence submodule for BICePs.

Parameters

- **filename** (*str*) – relative path and filename to MCMC trajectory (NumPy npz file)
- **outdir** (*str*) – relative path for output files

Convergence.**plot_traces**(*figname='traj_traces.png'*, *xlim=None*)

Plot trajectory traces.

Parameters
xlim (*tuple*) – matplotlib x-axis limits

Convergence.**plot_auto_curve**(*xlim=None*, *figname='autocorrelation_curve.png'*, *std_x=None*, *std_y=None*)

Plot auto-correlation curve. This function saves a figure of auto-correlation with error bars at the 95% confidence interval (τ_{auto} is rounded to the nearest integer).

Parameters

- **xlim** (*tuple*) – matplotlib x-axis limits
- **std_x** (*np.ndarray*) –
- **std_y** (*np.ndarray*) –

Convergence.**plot_block_avg**(*nblock*, *r_max*, *figname='block_avg.png'*)

Plot block average

Parameters

- **nblock** (*int*) – is the number of partitions in the time series

- **r_max** (*np.ndarray*) – maximum sampled parameters for each restraint
- **figname** (*str*) – figure name without relative path (taken care of)

Convergence.**get_autocorrelation_curves**(*method='auto'*, *nblocks=5*, *maxtau=10000*, *plot_traces=False*)

Compute autocorrelaton function for a time-series f(t), partition the data into the specified number of blocks and plot the autocorrelation curve. Saves a figure of autocorrelation curves for each restraint.

Parameters

- **method** (*str*) – method for computing autocorrelation time; "block-avg-auto" or "exp" or "auto"
- **nblocks** (*int*) – number of blocks to split up the trajectory
- **maxtau** (*int*) – the upper bound of autocorrelation lag time
- **plot_traces** (*bool*) – plot the trajectory traces?

Convergence.**process**(*nblock=5*, *nfold=10*, *nround=100*, *savefile=True*, *block_avg=False*, *normalize=True*)

Process the trajectory and execute `compute_JSD()` with `plot_JSD_conv()` and `plot_JSD_distribution()`. If `block_avg=True`, then block averaging will be executed and *plot_block_avg()* will be executed as well.

Parameters

- **nblock** (*int*) – is the number of partitions in the time series
- **nfold** (*int*) – is the number of partitions in the shuffled (subsampled) trajectory
- **nround** (*int*) – is the number of rounds of bootstrapping when computing JSDs
- **savefile** (*bool*) –
- **block_avg** (*bool*) – use block averaging
- **verbose** (*bool*) – verbosity

4.7 toolbox

biceps.toolbox.**sort_data**(*dataFiles*)

Sorting the data by extension into lists. Data can be located in various directories. Provide a list of paths where the data can be found. Some examples of fileextensions: {.noe,.J,.cs_H,.cs_Ha}.

Parameters

dataFiles (*list*) – list of strings where the data can be found

Raises

ValueError – if the data directory does not exist

```
>>> biceps.toolbox.sort_data()
```

biceps.toolbox.**get_files**(*path*)

Return a sorted list of files that will be globbed from the path given. First, this function can handle decimals and multiple numbers that are seperated by characters. https://pypi.org/project/natsort/

Parameters

path (*str*) –

Returns

sorted list

biceps.toolbox.**list_res**(*input_data*)

Determine the ordering of the experimental restraints that will be included in sampling.

Parameters
input_data (*list*) – see *biceps.Ensemble.initialize_restraints*

```
>>> biceps.toolbox.list_res()
```

biceps.toolbox.**list_possible_restraints**()

Function will return a list of all possible restraint classes in Restraint.py.

```
>>> biceps.toolbox.list_possible_restraints()
```

biceps.toolbox.**list_extensions**(*input_data*)

Determine the ordering of the experimental restraints that will be included in sampling.

Parameters
input_data (*list*) – see *biceps.Ensemble.initialize_restraints*

```
>>> biceps.toolbox.list_extensions()
```

biceps.toolbox.**list_possible_extensions**()

Function will return a list of all possible input data file extensions.

```
>>> biceps.toolbox.list_possible_extensions()
```

biceps.toolbox.**npz_to_DataFrame**(*file*, *out_filename='traj_lambda0.00.pkl'*, *verbose=False*)

Converts numpy Z compressed file to Pandas DataFrame (*.pkl)

```
>>> biceps.toolbox.npz_to_DataFrame(file, out_filename="traj_lambda0.00.pkl")
```

biceps.toolbox.**save_object**(*obj*, *filename*)

Saves python object as pickle file. :param obj: python object :type obj: object :param filename: relative path for ouput :type filename: str

```
>>> biceps.toolbox.save_object()
```

CHAPTER

FIVE

CITATION

Please apply BICePs in your research and cite it in any scientific publications.

```
@article{raddi2022biceps,
  title={BICePs v2. 0: Software for Ensemble Reweighting using Bayesian Inference of␣
↪Conformational Populations},
  author={Raddi, Robert and Ge, Yunhui and Voelz, Vincent},
  year={2022}
}

@article{VAV-2018,
    title = {Model selection using BICePs: A Bayesian approach to forcefield validation␣
↪and parameterization},
    author = {Yunhui Ge and Vincent A. Voelz},
    journal = {Journal of Physical Chemistry B},
    volume = {122},
    number = {21},
    pages = {5610 -- 5622},
    year = {2018},
    doi = {doi:10.1021/acs.jpcb.7b11871}
}
```

CHAPTER

SIX

LICENSE

INDEX