ThermoLIB Documentation

Release v1.0.0

Louis Vanduyfhuys

Sep 15, 2021

CONTENTS

ThermoLIB is a library developed at the Center for Molecular Modeling (CMM)for the application of Statistical Physics, Thermodynamics and/or kinetic theory to molecular simulations. The library consists of several sub modules:

- **Thermodynamics** - Module for reading, constructing, transforming and manipulating free energy profiles. Functions include construction of free energy profiles (FEPs) from histogram(s) (including error estimation), identification of (meta)stable macrostates and computation of their free energy, transformation of FEPs from one collective variable (CV) to another, (de)projection of an free energy surface (FES) to a lower/higher dimensional FES.
- **Kinetics** - Module for computing the rate factor required in transition state theory (TST) required to compute the reaction rate constant. This factor is related to the time derivative of the collective variable in the transition state. Together with thermodynamic observables such as the free energy of the reactant state and transition state, the TST-reaction rate can be computed.

CHAPTER

ONE

HOW TO CITE THERMOLIB

If you used ThermoLIB in your research, please refer to ThermoLIB as follows:

CHAPTER

TWO

INSTALLATION GUIDE

ThermoLIB is developed and tested on modern Linux environments. The installation instructions given in *this guide* are given for a Linux system only. If you want to use ThermoLIB on other operating systems such as Windows or OSX, you should have a minimal computer geek status to get it working. We are always interested in hearing from your installation adventures.

2.1 Installation Guide

Here you can find information on the packages required for a successfull installation of ThermoLIB as well as details on how to download and install ThermoLIB on a Linux system. On Windows 10, ThermoLIB was tested for Windows Subsystem for Linux - version 2. For a pure Windows-based installation, we cannot garantue any assistance but we are offcourse interested to hear about your adventures.

2.1.1 Dependencies

2.1.1.1 MolMod Dependency

MolMod is a Python library used by most Python programs developed at the CMM. It must be installed before ThermoLIB can be used. Installation and download instructions can be found in the molmod documentation. The instructions below only work if the MolMod package is installed.

2.1.1.2 External Dependencies

Some software packages should be installed before ThermoLIB can be installed and/or used. It is recommended to use the software package management of your Linux distribution to install these dependencies.

The following software must be installed:

- Python3 <= 3.7 (including the development files): http://www.python.org/
- Numpy >= 1.0: http://numpy.scipy.org/
- Scipy >= 0.14: http://www.scipy.org/
- matplotlib >= 1.0.0: http://matplotlib.sourceforge.net
- scikit-learn >= 0.24.2: https://scikit-learn.org/

On Ubuntu distributions, this can be installed with the following command:

```
sudo apt-get install python3 python3-numpy python3-scipy python3-matplotlib␣
↪python3-sklearn
```

In order to get the LaTeX support in the plots made by ThermoLIB, you also need the following packages installed

- latex: https://www.latex-project.org/
- cm-super: https://www.ctan.org/tex-archive/fonts/ps-type1/cm-super/

which can be installed on Ubuntu using the following command:

```
sudo apt-get install texlive texlive-latex-extra texlive-fonts-recommended␣
↪dvipng cm-super
```

2.1.2 Downloading ThermoLIB

2.1.2.1 Stable releases

A stable release of ThermoLIB, which includes all features described on this website, can be downloaded through various means.

- For members of Ghent University with an active UGent GitHUB account, the latest releases of ThermoLIB can be downloaded from the UGent GitHUB page.
- For GitHUB users not member of Ghent University, stable ThermoLIB releases can be downloaded from the general GitHUB page (**TODO**).
- Various main releases can also be downloaded from the links below (**TODO**):

After downloading the archive, choose a suitable directory, e.g. `~/build`, and unpack the archive:

```
mkdir -p ~/build
cd ~/build
tar -xvzf ThermoLIB-X.X.X.tar.gz
cd ThermoLIB-X.X.X
```

2.1.2.2 Latest development version (experts only)

The latest development version of QuickFF can only be downloaded using Git. This also allows you to upload your own changes in the form of a pull request. Git is free and open-source distributed revision control system to easily handle programming projects shared between several people. Further information about git (including downloads and tutorials) can be found here. The official git URL for ThermoLIB is **TODO**. To *clone* the ThermoLIB repository, go to your favorite directory for source code, e.g. `~/build`, and execute the following commands:

```
git clone git://github.com/TODO.git thermolib
cd thermolib
```

The source code can be updated with the latest patches with the following command:

```
git pull
```

This will also update the version history so that the progress can easily be tracked. This version history can be visualized using the *gitk* program. This program can be downloaded with the following command (Ubuntu):

```
sudo apt-get install gitk
```

Once, *gitk* is installed, the version history can be visualized by going to the directory containing the quickff repository and running the command:

```
gitk
```

2.1.3 Installing ThermoLIB

Once you downloaded the source code and installed all required packages, ThermoLIB can be installed. To do so, simply go to the directory in which you extracted the ThermoLIB source files (the directory containing the file *setup.py*) and run the following command:

```
python setup.py install
```

CHAPTER

THREE

USER GUIDE

A manual elaborating on how to use ThermoLIB for the construction, transformation and projection of free energy profiles, computation of kinetic rate constants and more.

3.1 User Guide - Thermodynamics

In this user guide we will outline how to use the various features of ThermoLIB related to equilibrium thermodynamics. The central thermodynamic object in ThermoLIB is a *free energy profile (FEP) in one dimension (1D)* or a *free energy surface (FES) in two dimensions (2D)*. In the following subsections, we discuss both in detail separately.

3.1.1 Constructing a 1D Histogram

In many cases, a free energy profile will be computed by means of first constructing a corresponding histogram as model for the probability density. We first provide further details on how such histogram can be constructed (including an estimation of its error).

3.1.1.1 without error estimation

A histogram can be computed from a trajectory of CV values generated by a molecular simulation (such as molecular dynamics or Monte Carlo). To this end, we use `the Histogram1D class` as follows:

```
from thermolib.thermodynamics.histogram import Histogram1D
histogram = Histogram1D.from_single_trajectory(cv_data, bins)
```

Herein, *cv_data* represents a numpy array containig the CV values along the trajectory. This array can be computed from an XYZ trajectory file using the routine `trajectory_xyz_to_CV`. Furthermore, *bins* represents the argument of the same name defined in the numpy.histogram routine, hence for more information on its meaning and allowed values we refer to its documentation. We can plot the resulting histogram using `its plot routine`:

```
histogram.plot('histogram_noerror.png')
```

One can even add an additional pane in the plot with the corresponding free energy profile by specifying the temperature at which the simulation trajectory was generated:

```
histogram.plot('histogram_noerror.png', temp=temp)
```

More information on the construction of a histogram and its optional features can be found in the reference guide of *Histogram1D*.

3.1.1.2 with error estimation

With a few extra keyword arguments, we can include an estimation of the error bar. Two methods are distinguished:

- Estimating the error using the asymptotic normality of the maximum likelihood estimator applied to the entire trajectory.
- First divide the trajectory into blocks, construct a histogram for each block and estimate total histogram and its error through the average and standard deviation of the block histograms.

More details on the theory behind these methods can be found in the theory section. In the following subsections, we describe how to perform such error estimates using ThermoLIB.

from asymptotic normality of the MLE

The construction of a histogram including an error estimation from the entire trajectory (using the asymptotic normality of the maximum likelihood estimator) is done by setting the keyword `error_estimate='single'`:

```
from thermolib.thermodynamics.histogram import Histogram1D
histogram = Histogram1D.from_single_trajectory(cv_data, bins, error_estimate=
↪'single')
histogram.plot('histogram_single.png')
```

Specifying the `temp` argument in the histogram plot routine will include an additional pane with the correspondig free energy profile and its error estimate.

from block analysis

Alternative error estimation can be based on block analysis. Herein, we first divide the trajectory in a number of blocks (as defined by the `nblocks` keyword) and compute a histogram for each block. Afterwards, the total histogram and its error are estimated through the average and standard deviation of these histograms. This can be done as illustrated in the following code:

```
from thermolib.thermodynamics.histogram import Histogram1D
histogram = Histogram1D.from_single_trajectory(cv_data, bins, error_estimate=
↪'blocked', nblocks=6)
histogram.plot('histogram_blocked.png', temp=temp)
```

3.1.2 Constructing a 1D free energy profile

The base object of a 1D free energy surface is an instance of the `BaseFreeEnergyProfile`. This class implements all basic features of 1D free energy profiles such as:

- making a plot of the free energy profile using the `BaseFreeEnergyProfile.plot` routine or the one of a deriving class
- cropping the range of a collective variable using the `BaseFreeEnergyProfile.crop` routine.
- recollecting the collective variable into newly defined intervals using the `BaseFreeEnergyProfile.recollect` routine.
- *transformation towards other collective variables*
- *deprojection of a 1D FEP towards a 2D FES*
- …

as is included in this documentation. This class serves as a master class from which deriving classes will inherit all these features. The deriving classes will further implement additional features that require assumptions of the free energy profile. At this moment, only one deriving class is implemented, the `SimpleFreeEnergyProfile` which represents the FEP of a simple process with a single reactant, transition and product state. As such these states can be detected by means of the `SimpleFreeEnergyProfile.process_states` routine and will then be automatically shown on the plot produced by the `SimpleFreeEnergyProfile.plot` plot routine.

There are several ways to construct a free energy profile. The following sections in this will use the routines of BaseFreeEnergyProfile and can hence also be applied to SimpleFreeEnergyProfile.

3.1.2.1 using the constructor

```
from thermolib.thermodynamics.fep import BaseFreeEnergyProfile
fep = BaseFreeEnergyProfile(cv, f, temp)
```

Herein, *cv* and *f* represent equal-length numpy arrays containing respectively the collective variable and free energy on a grid. By default these arrays should be defined in atomic units. More control over these units is possible using the optional arguments as described in `the reference guide entry of this routine`. Furthermore, *temp* represents the temperature (in atomic units, i.e. Kelvin) at which the free energy profile is evaluated.

3.1.2.2 reading from a text file

```
from thermolib.thermodynamics.fep import BaseFreeEnergyProfile
fep = BaseFreeEnergyProfile.from_txt(fn, temp)
```

Herein, *fn* represents the name of a text file containing the values of the collective variable and the corresponding free energy on a grid. By default, the *cv* and *f* values should be respectively in the first and second column separated by one or multiple whitespaces. More control on the columns from which to read the data as well as the delimiter and more can be found in `the reference guide entry of this routine`. Furthermore, *temp* represents the temperature (in atomic units, i.e. Kelvin) at which the free energy profile is evaluated.

3.1.2.3 from a trajectory histogram

The free energy profile can also be constructed from a histogram that was computed from a trajectory of CV values generated by a molecular simulation (such as molecular dynamics or Monte Carlo). To this end, we first contruct a histogram as outlined in *the user guide on histogram construction* and compute the free energy profile from this histogram as follows:

```
from thermolib.thermodynamics.fep import BaseFreeEnergyProfile
from thermolib.thermodynamics.histogram import Histogram1D
histogram = Histogram1D.from_single_trajectory(cv_data, bins)
temp = 300*kelvin
fep = BaseFreeEnergyProfile.from_histogram(histogram, temp)
```

As outlined previously, the routine `from_single_trajectory` can also estamate the error on the histogram. If so, the free energy profile generated from the resulting histogram will also contain the correpsonding error which in turn will be included in any plots of the free energy profile made with its plot routine.

3.1.3 2D free energy surface

3.1.4 Transformations

3.1.4.1 Transforming FEP between collective variables

3.1.4.2 Projecting 2D FES to 1D FEP

3.1.4.3 Deprojecting 1D FEP to 2D FES

3.2 User Guide - Kinetics

In this user guide we will outline how to use the various features of ThermoLIB related to kinetics.

3.2.1 Reaction rate from equilibrium simulations

3.2.2 Alternative computation of reaction rate

CHAPTER FOUR

TUTORIALS

Various tutorials giving practical examples on how to use ThermoLIB.

4.1 Tutorials

Here we will give some practical examples on how to use ThermoLIB.

CHAPTER FIVE

REFERENCE GUIDE

A reference guide generated from the documentation strings in the source code. It includes an overview of all modules included in QuickFF. This information is based on the documentation provided in the source code.

5.1 Reference Guide

An overview is provived of all modules included in ThermoLIB. The information below is based on the documentation provided in the source code.

5.1.1 Thermodynamics Modules

5.1.1.1 Free energy profiles – `thermodynamics.fep`

1D Free energy profile

class `thermolib.thermodynamics.fep.`**`BaseFreeEnergyProfile`**(*cvs*, *fs*, *temp*, *fupper=None*, *flower=None*, *cv_unit='au'*, *f_unit='kjmol'*, *cv_label='CV'*)

Base class to define a free energy profile F(q) (stored in self.fs) as function of a certain collective variable (CV) denoted by q (stored in self.cvs).

Parameters

- **cv** (*np.ndarray*) – the collective variable values
- **f** (*np.ndarray*) – the free energy values
- **temp** (*float*) – the temperature at which the free energy is constructed
- **flower** (*np.ndarray*) – the lower limit of the error bar on the free energy values
- **fupper** (*np.ndarray*) – the upper limit of the error bar on the free energy values
- **cv_unit** (*str, default=au*) – the units for printing of CV values. Units are defined using the molmod routine.
- **f_unit** (*str, default=kjmol*) – the units for printing of free energy values. Units are defined using the molmod routine molmod.units.

compute_probdens()

Compute the probability density profile associated with the free energy profile:

$$p(q) = \frac{1}{q_0 Z}\exp\left(-\beta F(q)\right)$$

with

$$Z = \frac{1}{q_0}\int_{-\infty}^{+\infty}\exp\left(-\beta F(q)\right)dq$$

Herein, q_0 represents an arbitrary constant to account for the fact that the partition function should in principle be dimensionless. However, it can be chosen freely and has no impact on the final free energy profile (appart from meaningless a vertical shift). Therefore, in this implementation it is chosen as $q_0 = 1$.

crop(*cvrange*, *return_new_fes=False*)

Crop the free energy profile to the limits given by cvrange and throw away cropped data. If return_new_fes is set to false, a copy of the cropped profile will be returns, otherwise the current profile will be cropped and overwritten.

Parameters

- **cvrange** (*tuple*) – the range of the collective variable defining the new range to which the FEP will be cropped.
- **return_new_fes** (*bool, optional*) – If set to False, the cropped data will be written to the current instance (overwritting the original data). If set to True, a new instance will be initialized with the cropped data, defaults to False

Returns an instance of the current class if the keyword argument *return_new_fes* is set to True. Otherwise, None is returned.

classmethod from_average(*feps*, *error_estimate=None*, *nsigma=2*)

Start from a set of free energy profiles and compute and return the averaged free energy profile. If error_estimate is set to 'std', an error on the freee energy profile will be computed from the standard deviation within the set of profiles.

Parameters

- **feps** – set of free energy profiles to be averaged
- **error_estimate** – indicate if and how to perform error analysis. One of following options is available:
  - **std** – compute error from the standard deviation within the set of profiles.
  - **None** – do not estimate the error.

Defaults to None, i.e. no error estimate. :type error_estimate: str, optional

Parameters nsigma (*int, optional*) – only relevant when error estimation is turned on (i.e. when keyword `error_estimate` is not None), this option defines how large the error interval should be in terms of the standard deviation sigma. A `nsigma=2` implies a 2-sigma error bar (corresponding to 95% confidence interval) will be returned. Defaults to 2

Returns averages free energy profile

Return type identical as the mother class of the current routine `classmethod from_histogram`(*histogram*, *temp*)

Use the estimated probability histogram to construct the corresponding free energy profile at the given temperature.

Parameters

- **histogram** (`histogram.Histogram1D`) – histogram from which the free energy profile is computed
- **temp** (*float*) – the temperature at which the histogram input data was simulated

Returns free energy profile corresponding to the estimated probability histogram

Return type cls

`classmethod from_txt`(*fn*, *temp*, *cvcol=0*, *fcol=1*, *cv_unit='au'*, *f_unit='kjmol'*, *cvrange=None*, *delimiter=None*, *reverse=False*, *cut_constant=False*)

Read the free energy profile as function of a collective variable from a txt file.

Parameters

- **fn** (*str*) – the name of the txt file containing the data
- **temp** (*float*) – the temperature at which the free energy is constructed
- **cvcol** (*int, default=0*) – the column in which the collective variable is stored
- **fcol** (*int, default=1*) – the column in which the free energy is stored.
- **cv_unit** (*str or float, default=au*) – the units in which the CV are stored. Units are defined using the molmod routine.
- **f_unit** (*str or float, default=kjmol*) – the units in which the free energy are stored. Units are defined using the molmod routine
- **cvrange** (*tuple or list, default=None*) – only read free energy for CVs in the given range
- **delimiter** (*str, default=None*) – The delimiter used in the txt input file to separate columns. The default is set to None, corresponding to the default of the numpy.loadtxt routine (i.e. whitespace).
- **reverse** (*bool, default=False*) – if set to True, reverse the X axis (usefull to make sure reactant is on the left)
- **cut_constant** (*bool, default=False*) – if set to True, the data points at the start and end of the data array that are constant will be cut. Usefull to cut out unsampled areas for large and small CV values.

`macrostate`(*cvrange=None*, *indexes=None*, *verbose=False*)

Return the contribution to the partition function and free energy corresponding to the macrostate in the given range of cvs. This contribution is computed as follows.

$$Z_A = \frac{1}{q_0}\int_A \exp\left(-\beta F(q)\right) dq$$

$$= Z \cdot \int_A p(q) dq$$

$$F_A = -k_B T \log Z_A$$

Parameters

- **`cvrange`** (*`np.ndarray, optional`*) – the range of the collective variable defining the macrostate. Either cvrange or indices should be defined, defaults to None
- **`indexes`** (*`list, optional`*) – the indexes of the collective variable defining the macrostate. Either cvrange or indices should be defined, defaults to None
- **`verbose`** (*`bool, optional`*) – set to true to turn on verbosity, defaults to False

Returns

- **mean** (float) - the expected (=mean) value of the collective variable in the macrostate
- **std** (float) - the thermal fluctuation (=standard deviation) of the CV in the macrostate
- **Z** (float) - the contribution of the macrostate to the partition function
- **F** (float) - the free energy of the given macrostate

`plot`(*fn*, *flim=None*)

Make a plot of the free energy profile.

Parameters **`fn`** (*`str`*) – Name of the file of the figure. Supported file formats are determined by the supported formats of the matplotlib.pyplot.savefig routine

`recollect`(*new_cvs*, *fn_plt=None*, *return_new_fes=False*)

Redefine the CV array to the new given array. For each interval of new CV values, collect all old free energy values for which the corresponding CV value falls in this new interval and average out. As such, this routine can be used to filter out noise on a given free energy profile by means of averaging.

Parameters

- **`new_cvs`** (*`np.ndarray`*) – Array of new CV values
- **`fn_plt`** (*`str, optional`*) – File name for comparison plot of old and new profile., defaults to None
- **`return_new_fes`** (*`bool, optional`*) – If set to False, the recollected data will be written to the current instance (overwritting the original data). If set to True, a new instance will be initialized with the recollected data., defaults to False

Returns Returns an instance of the current class if the keyword argument *return_new_fes* is set to True. Otherwise, None is returned.

`savetxt`(*fn_txt*)
Save the free energy profile as txt file

`set_ref`(*ref='min'*)
Set the energy reference of the free energy profile.

Parameters `ref` (*str*) – the choice for the energy reference, currently only 'min' or 'm' is implemented resulting in setting the reference to the minimum in the free energy profile. Defaults to 'min'

Raises `IOError` – invalid value for keyword argument ref is given. See doc above for choices.

`class thermolib.thermodynamics.fep.SimpleFreeEnergyProfile`(*cvs*, *fs*, *temp*, *fupper=None*, *flower=None*, *cv_unit='au'*, *f_unit='kjmol'*, *cv_label='CV'*)

Class implementing a 1D FEP representing a simple bi-stable profile with 2 minima representing the reactant and process states and 1 local maximum representing the transition state.

As such, this class offers all features of the parent class `BaseFreeEnergyProfile()` as well as the additional feature to automatically identify the micro/macrostates corresponding to reactant state, transition state and product state (the `process_states()` routine).

See `BaseFreeEnergyProfile()` for constructor arguments and documentation.

`_find_R_TS_P`(*ts_range=[- inf, inf]*)
Internal routine called by `process_states()` to find:

- the transition state (TS) as the local maximum within the given ts_range
- the reactant (R) as local minimum left of TS
- the product (P) as local minimum right of TS

Parameters `ts_range` (*list, optional*) – range for the cv in which to look for the transition state as a local maximum, defaults to [-np.inf,np.inf]

Raises `ValueError` – the transition state cannot be found in the range defined by ts_range.

`plot`(*fn*, *rate=None*, *micro_marker='s'*, *micro_color='r'*, *micro_size='4'*, *macro_linestyle='-'*, *macro_color='b'*)
Plot the free energy profile including visualization of the microstates (markers) and macrostates (lines) defined by `set_microstates()` and `set_macrostates()` respectively

Parameters

- **fn** (*str*) – file name of the resulting plot, the extension of the file name will determine the format (png or pdf)
- **rate** (`thermolib.kinetics.rate.RateFactorEquilibrium` *or* *thermolib.kinetics.rate.RateFactorAlternative, optional*) – rate factor instance from `thermolib.kinetics` module which will allow

to include indication of reaction rate and phenomenological free energy barriers, defaults to None

- **micro_marker** (*str, optional*) – matplotlib marker style for indicating microstates, defaults to 's'
- **micro_color** (*str, optional*) – matplotlib marker color for indicating microstates, defaults to 'r'
- **micro_size** (*str, optional*) – matplotlib marker size for indicating microstates, defaults to '4'
- **macro_linestyle** (*str, optional*) – matplotlib line style for indicating macrostates, defaults to '-'
- **macro_color** (*str, optional*) – matplotlib line color for indicating macrostates, defaults to 'b'

`process_states`(*ts_range=[- inf, inf]*, *verbose=False*)

Routine to find the reactant (R), transition state (TS) and product state (P) through application of the `_find_R_TS_P()` routine and add the corresponding micro and macrostates. These will afterwards be shown on the free energy plot using the `plot()` routine.

Parameters

- **ts_range** (*list, optional*) – range for the cv in which to look for the transition state as a local maximum, defaults to [-np.inf,np.inf]
- **verbose** (*bool, optional*) – set to True to increase verbosity, defaults to False

`set_macrostates`(*cvs*, *verbose=False*)

Routine to define macrostates, i.e. regions on the 1D FEP. This routine is called by `process_states()` to add the macrostates found by the latter. These macrostates will be visualized as horizontal linesegments on a plot made by the `plot()` routine with free energy contributions as computed by the `macrostate()` routine.

Parameters

- **cvs** (*list(int)*) – list of integers giving defining the index of the boundaries between subsequent macrostates.
- **verbose** (*bool, optional*) – set to True to increase verbosity, defaults to False

`set_microstates`(*indices*)

Routine to define microstates, i.e. points on the 1D FEP. This routine is called by `process_states()` to add the microstates found by the latter. These microstates will be visualized as points with corresponding free energy value on a plot made by the `plot()` routine.

Parameters **indices** (*list(int)*) – list of indices corresponding to the index of the microstates in the self.cvs and self.fs arrays

`set_ref`(*ref='min'*)

Set the energy reference of the free energy profile.

Parameters **ref** (*str*) – the choice for the energy reference, should be one of:

- *m* or *min* for the global minimum

- *r* or *reactant* for the reactant minimum
- *ts*, *trans_state* or *transition* for the transition state maximum
- *p* or *product* for the product minimum

The options r, ts and p are only available if the reactant, transition state and product have already been found by the routine find_states, defaults to 'min'

Raises **IOError** – invalid value for keyword argument ref is given. See doc above for choices.

2D Free energy surface

`class thermolib.thermodynamics.fep.FreeEnergySurface2D(cv1s, cv2s, fs, temp, fupper=None, flower=None, cv1_unit='au', cv2_unit='au', f_unit='kjmol', cv1_label='CV1', cv2_label='CV2')`

Class implementing a 2D free energy surface F(cv1,cv2) (stored in self.fs) as function of two collective variables (CV) denoted by cv1 (stored in self.cv1s) and cv2 (stored in self.cv2s).

Parameters

- **cv1s** (*np.ndarray*) – array containing the values for the first collective variable CV1
- **cv2s** (*np.ndarray*) – array the values for the second collective variable CV2
- **fs** (*np.ndarray*) – 2D array containing the free energy values corresponding to the given values of CV1 and CV2
- **temp** (*float*) – temperature at which the free energy is given
- **fupper** (*np.ndarray*) – upper value of error bar on free energy
- **flower** (*np.ndarray*) – lower value of error bar on free energy
- **cv1_unit** (*str, optional*) – unit in which the input array cv1 is given (units are defined using the molmod routine), defaults to 'au'
- **cv2_unit** (*str, optional*) – unit in which the input array cv2 is given (units are defined using the molmod routine), defaults to 'au'
- **f_unit** (*str, optional*) – unit in which the input array f is given (units are defined using the molmod routine), defaults to 'kjmol'
- **cv1_label** (*str, optional*) – label for CV1 axis on plots, defaults to 'CV1'
- **cv2_label** (*str, optional*) – label for CV2 axis on plots, defaults to 'CV2'

compute_probdens()

Compute the probability density profile associated with the free energy profile as given below and store internally in *self.ps*

$$p(q) = \frac{\exp\left(-\beta F(q)\right)}{\int_{-\infty}^{+\infty} \exp\left(-\beta F(q)\right) dq}$$

copy()

Make and return a copy of the current FreeEnergySurface2D instance.

Returns a copy of the current instance

Return type *FreeEnergySurface2D*

crop(*cv1range=None*, *cv2range=None*, *return_new_fes=False*)

Crop the free energy surface by removing all data for which either cv1 or cv2 is beyond a given range.

Parameters

- **cv1range** (*[type], optional*) – range of cv1 that will remain after cropping, defaults to None
- **cv2range** (*[type], optional*) – range of cv2 that will remain after cropping, defaults to None
- **return_new_fes** (*bool, optional*) – if set to false, the cropping process will be applied on the existing instance, otherwise a copy will be returned, defaults to False

Returns None or new instance of `FreeEnergySurface2D` representing cropped FES, depending on `return_new_fes`

detect_clusters(*eps=1.5*, *min_samples=8*, *metric='euclidean'*, *fn_plot=None*, *delete_clusters=[- 1]*)

Routine to apply the DBSCAN clustering algoritm as implemented in the Scikit Learn package to the (CV1,CV2) grid points that correspond to finite free energies (i.e. not nan or inf) to detect clusters of neighboring points.

The DBSCAN algorithm first identifies the core samples, defined as samples for which at least `min_samples` other samples are within a distance of `eps`. Next, the data is divided into clusters based on these core samples. A cluster is defined as a set of core samples that can be built by recursively taking a core sample, finding all of its neighbors that are core samples, finding all of their neighbors that are core samples, and so on. A cluster also has a set of non-core samples, which are samples that are neighbors of a core sample in the cluster but are not themselves core samples. Intuitively, these samples are on the fringes of a cluster. Each cluster is given an integer as label.

Any sample that is not a core sample, and is at least `eps` in distance from any core sample, is considered an outlier by the algorithm and is what we here consider an isolated point/region. These points get the cluster label of -1.

Finally, all data points belonging to a cluster with label specified in `delete_clusters` will have theire free energy set to nan. A safe choice here is to just delete isolated regions, i.e. the point in cluster with label -1 (which is the default).

Parameters

- **eps** (*float, optional*) – DBSCAN parameter representing maximum distance between two samples for them to be considered neighbors (for more, see DBSCAN documentation), defaults to 1.5
- **min_samples** (*int, optional*) – DBSCAN parameter representing the number of samples in a neighborhood for a point to be considered a core point (for more, see DBSCAN documentation), defaults to 8

- **metric** (*str or callable, optional*) – DBSCAN parameter representing the metric used when calculating distance (for more, see DBSCAN documentation), defaults to 'euclidean'
- **fn_plot** (*str, optional*) – if specified, a plot will be made (and written to `fn_plot`) visualizing the resulting clusters, defaults to None
- **delete_clusters** (*list, optional*) – list of cluster names whos members will be deleted from the free energy surface data, defaults to [-1] meaning only isolated points (not belonging to a cluster) will be deleted.

`classmethod` **from_histogram**(*histogram*, *temp*)

Use the estimated 2D probability histogram to construct the corresponding 2D free energy surface at the given temperature.

Parameters

- **histogram** (*histogram.Histogram2D*) – histogram from which the free energy profile is computed
- **temp** (*float*) – the temperature at which the histogram input data was simulated

Returns free energy profile corresponding to the estimated probability histogram

Return type cls

`classmethod` **from_txt**(*fn*, *temp*, *cv1_col=0*, *cv2_col=1*, *f_col=2*, *cv1_unit='au'*, *cv2_unit='au'*, *f_unit='kjmol'*, *cv1_label='CV1'*, *cv2_label='CV2'*, *cv1_range=None*, *cv2_range=None*, *delimiter=None*, *verbose=False*)

Read the free energy surface on a 2D grid as function of two collective variables from a txt file.

Parameters

- **fn** (*str*) – the name of the txt file containing the data
- **temp** (*float*) – the temperature at which the free energy is constructed
- **cv1_col** (*int, optional*) – the column in which the first collective variable is stored, defaults to 0
- **cv2_col** (*int, optional*) – the column in which the second collective variable is stored, defaults to 1
- **f_col** (*int, optional*) – the column in which the free energy is stored, defaults to 2
- **cv1_unit** (*str, optional*) – the unit in which the first CV values are stored, defaults to 'au'
- **cv2_unit** (*str, optional*) – the unit in which the second CV values are stored, defaults to 'au'
- **f_unit** (*str, optional*) – the unit in which the free energy values are stored, defaults to 'kjmol'
- **cv1_label** (*str, optional*) – the label for the CV1 axis in plots, defaults to 'CV1'

- **cv2_label** (*str, optional*) – the label for the CV2 axis in plots, defaults to 'CV2'
- **cv1_range** (*tuple/list, optional*) – [CVmin,CVmax] indicating to only read free energy for which the first CV in the given range, defaults to None
- **cv2_range** (*tuple/list, optional*) – [CVmin,CVmax] indicating to only read free energy for which the second CV in the given range, defaults to None
- **delimiter** (*[type], optional*) – [description], defaults to None

Returns 2D free energy surface

Return type *FreeEnergySurface2D*

`plot`(*fn_png*, *obs='F'*, *cv1_lims=None*, *cv2_lims=None*, *lims=None*, *ncolors=8*, *scale='lin'*)
Simple routine to make a 2D contour plot of either the free energy F or probability distribution P as specified in `obs`.

Parameters

- **fn_png** (*str*) – File name to write the plot to. The extension determines the format (PNG or PDF).
- **obs** (*str, optional, choices=('F', 'P')*) – Specification which observable should be plotted, should be 'F' for free energy or 'P' for probability. Defaults to 'F'
- **cv1_lims** (*tupple/list, optional*) – range defining the plot limits of CV1, defaults to None
- **cv2_lims** (*tupple/list, optional*) – range defining the plot limits of CV1, defaults to None
- **lims** (*tupple/list, optional*) – range defining the plot limits for the observable, defaults to None
- **ncolors** (*int, optional*) – number of different colors included in contour plot, defaults to 8
- **scale** (*str, optional, choices=('lin', 'log')*) – scal for the observable, should be 'lin' for linear or 'log' for logarithmic. Ddefaults to 'lin'

Raises

- **IOError** – if invalid observable is given, see doc above for possible choices.
- **IOError** – if invalid scale is given, see doc above for possible choices.

`project_average`(*cv_unit='au'*, *return_class=<class 'thermolib.thermodynamics.fep.BaseFreeEnergyProfile'>*)
Construct a 1D free energy profile representing the projection of the 2D FES F2(CV1,CV2) onto the average q=(CV1+CV2)/2 of the collective variables:

$$F1(q) = -k_BT \log \left(2 \int_{-\infty}^{+\infty} e^{-\beta F2(x,2q-x} dx \right)$$

with $q = 0.5\dot{(}CV1{+}CV2)$. This projection is implemented by first projecting the probability density and afterwards reconstructing the free energy.

Parameters

- **cv_unit** (*str, optional*) – unit for the new CV for plotting purposes, defaults to au.
- **return_class** (*python class object, optional*) – The class of which an instance will finally be returned. Defaults to BaseFreeEnergyProfile

Returns projected 1D free energy profile

Return type see return_class argument

`project_cv1`(*return_class=<class 'thermolib.thermodynamics.fep.BaseFreeEnergyProfile'>*)

Construct a 1D free energy profile representing the projection of the 2D FES F2(CV1,CV2) onto q=CV1. This is implemented as follows:

$$F1(q) = -k_B T \log \left(\int_{-\infty}^{+\infty} e^{-\beta F2(q,y} dy \right)$$

Parameters **return_class** (*python class object, optional*) – The class of which an instance will finally be returned. Defaults to BaseFreeEnergyProfile

Returns projected 1D free energy profile

Return type see return_class argument

`project_cv2`(*return_class=<class 'thermolib.thermodynamics.fep.BaseFreeEnergyProfile'>*)

Construct a 1D free energy profile representing the projection of the 2D FES F2(CV1,CV2) onto q=CV2. This is implemented as follows:

$$F1(q) = -k_B T \log \left(\int_{-\infty}^{+\infty} e^{-\beta F2(x,q} dx \right)$$

Parameters **return_class** (*python class object, optional*) – The class of which an instance will finally be returned. Defaults to BaseFreeEnergyProfile

Returns projected 1D free energy profile

Return type see return_class argument

`project_difference`(*sign=1, cv_unit='au', return_class=<class 'thermolib.thermodynamics.fep.BaseFreeEnergyProfile'>*)

Construct a 1D free energy profile representing the projection of the 2D FES onto the difference of collective variables:

$$F1(q) = -k_B T \log \left(\int_{-\infty}^{+\infty} e^{-\beta F2(x,x+q)} dx \right)$$

with $q = CV2 - CV1$. This projection is implemented by first projecting the probability density and afterwards reconstructing the free energy.

Parameters

- **sign** (*int, optional*) – If sign is set to 1, the projection is done on q=CV2-CV1, if it is set to -1, projection is done to q=CV1-CV2 instead. Defaults to 1

- **cv_unit** (*str, optional*) – unit for the new CV for plotting purposes, defaults to au.
- **return_class** (*python class object, optional*) – The class of which an instance will finally be returned. Defaults to BaseFreeEnergyProfile

Returns projected 1D free energy profile

Return type see return_class argument

project_function(*function*, *cvs*, *delta=0.001*, *cv_label='CV'*, *cv_unit='au'*, *return_class=<class 'thermolib.thermodynamics.fep.BaseFreeEnergyProfile'>*)

Routine to implement the general projection of a 2D FES onto a collective variable defined by the given function (which takes the original two CVs as arguments).

Parameters

- **function** (*callable*) – function in terms of the original CVs to define the new CV to project upon
- **cvs** (*np.ndarray*) – grid for the new CV
- **delta** (*float, optional*) – width of the single-bin approximation of the delta function applied in the projection formula. Defaults to 1e-3 a.u.
- **cv_label** (*str, optional*) – label for the new CV, defaults to CV.
- **cv_unit** (*str, optional*) – unit for the new CV for plotting purposes, defaults to au.
- **return_class** (*python class object, optional*) – The class of which an instance will finally be returned. Defaults to BaseFreeEnergyProfile

Returns projected 1D free energy profile

Return type see return_class argument

rotate(*interpolate=True*)

Transform the free energy profile in terms of the following two new collective variables:

$$u = \frac{CV_1 + CV_2}{2}$$
$$v = CV_2 - CV_1$$

This transformation represents a simple rotation (and mirroring). From probability theory we find the transformation formula:

$$F_{\text{rot}}(u,v) = F\left(u - \frac{v}{2}, u + \frac{v}{2}\right))$$

The uniform (u,v)-grid introduces new grid points in between the original (cv1,cv2) grid points. If interpolate is True, the free energy for these points is interpolated if all four neighbors have defined (i.e. not nan) free energies.

Parameters **interpolate** (*bool, optional*) – if set to true, interpolate undefined grid points (arrising due to rotation) between neighbors, defaults to True

Returns rotated free energy surface

Return type *FreeEnergySurface2D*

`savetxt`(*fn_txt*)
Save the free energy profile as txt file

`set_ref`(*ref='min'*)
Set the energy reference of the free energy surface.

Parameters **`ref`** (*`str`*) – the choice for the energy reference. Currently only one possibility is implemented, i.e. *m* or *min* for the global minimum. Defaults to 'min'.

Raises `IOError` – invalid value for keyword argument ref is given. See doc above for choices.

5.1.1.2 Histograms – `thermodynamics.histogram`

`class thermolib.thermodynamics.histogram.`**`Histogram1D`**(*cvs*, *ps*, *plower=None*, *pupper=None*, *cv_unit='au'*, *cv_label='CV'*)
Class to implement the estimation of a probability histogram in terms of a collective variable from a trajectory series corresponding to that collective variable.

Parameters

- **`cvs`** – array corresponding the collective variable grid points
- **`ps`** – array corresponding to the histogram probability values at the grid points
- **`plower`** – lower bound of the error on the probability histogram values, defaults to None
- **`pupper`** (*`np.ndarray, optional`*) – upper bound of the error on the probability histogram values, defaults to None
- **`cv_unit`** (*`str, optional`*) – the unit in which cv will be plotted/printed, defaults to 'au'
- **`cv_label`** (*`str, optional`*) – the label of the cv that will be used on plots, defaults to 'CV'

`classmethod` **`from_average`**(*histograms*, *error_estimate=None*, *nsigma=2*)
Start from a set of histograms and compute and return the averaged histogram. If error_estimate is set to 'std', an error on the histogram will be computed from the standard deviation within the set of histograms.

Parameters

- **`histograms`** (*`list(`*`Histogram1D`*`)`*) – set of histrograms to be averaged
- **`error_estimate`** – indicate if and how to perform error analysis. One of following options is available:
  - **std** – compute error from the standard deviation within the set of histograms.
  - **None** – do not estimate the error.

Defaults to None, i.e. no error estimate. :type error_estimate: str, optional

Parameters **nsigma** (*int, optional*) – only relevant when error estimation is turned on (i.e. when keyword `error_estimate` is not None), this option defines how large the error interval should be in terms of the standard deviation sigma. A `nsigma=2` implies a 2-sigma error bar (corresponding to 95% confidence interval) will be returned. Defaults to 2

Returns averages histogram

Return type Histrogram1D

`classmethod from_fep`(*fep*, *temp*)

Use the given free energy profile to construct the corresponding probability histogram at the given temperature.

Parameters

- **fep** (*fep.BaseFreeEnergyProfile/fep.SimpleFreeEnergyProfile*) – free energy profile from which the probability histogram is computed
- **temp** (*float*) – the temperature at which the histogram input data was simulated

Returns probability histogram corresponding to the given free energy profile

Return type *Histogram1D*

`classmethod from_single_trajectory`(*data*, *bins*, *error_estimate=None*, *nsigma=2*, *cv_unit='au'*, *cv_label='CV'*)

Routine to estimate of a probability histogram in terms of a collective variable from a trajectory series corresponding to that collective variable.

Parameters

- **data** (*np.ndarray*) – array corresponding to the trajectory series of the collective variable
- **bins** (*np.ndarray*) – array representing the edges of the bins for which a histogram will be constructed. This argument is parsed to the numpy.histogram routine. Hence, more information on its meaning and allowed values can be found there.
- **error_estimate** – indicate if and how to perform error analysis. One of following options is available:
  - **mle_p** – compute error through the asymptotic normality of the maximum likelihood estimator for the probability itself. WARNING: due to positivity constraint of the probability, this only works for low variance. Otherwise the standard error interval for the normal distribution (i.e. mle +- n*sigma) might give rise to negative lower error bars.
  - **mle_f** – compute error through the asymptotic normality of the maximum likelihood estimator for -log(p) (hence for the beta-scaled free energy). This estimation does not suffor from the same WARNING as for `mle_p`. Furthermore, in case of low variance, the error estimation using `mle_f` and `mle_p` are consistent (i.e. one can be computed from the other using f=-log(p)).
  - **None** – do not estimate the error.

Defaults to None, i.e. no error estimate. :type error_estimate: str, optional

Parameters

- **nsigma** (*int, optional*) – only relevant when error estimation is turned on (i.e. when keyword `error_estimate` is not None), this option defines how large the error interval should be in terms of the standard deviation sigma. A `nsigma=2` implies a 2-sigma error bar (corresponding to 95% confidence interval) will be returned. Defaults to 2
- **cv_unit** (*str, optional*) – the unit in which cv will be plotted/printed, defaults to 'au'
- **cv_label** (*str, optional*) – the label of the cv that will be used on plots, defaults to 'CV'

classmethod from_wham(*bins*, *trajectories*, *biasses*, *temp*, *error_estimate=None*, *nsigma=2*, *bias_subgrid_num=20*, *Nscf=1000*, *convergence=1e-06*, *verbose=False*, *cv_unit='au'*, *cv_label='CV'*)

Routine that implements the Weighted Histogram Analysis Method (WHAM) for reconstructing the overall probability histogram from a series of biased molecular simulations.

Parameters

- **bins** (*int or np.ndarray(float)*) – number of bins for the CV grid or array representing the bin edges for th CV grid.
- **trajectories** (*list(np.ndarray)/np.ndarray(np.ndarray)*) – list or array of numpy arrays containing the CV trajectory data for each simulation. Alternatively, a list of PLUMED file names containing the trajectory data can be specified as well. The arguments trajectories and biasses should be of the same length.
- **biasses** (*list(callable)*) – list of callables, each representing a function to compute the bias at a given value of the collective variable. The arguments trajectories and biasses should be of the same length.
- **temp** (*float*) – the temperature at which all simulations were performed
- **error_estimate** – indicate if and how to perform error analysis. One of following options is available:
  - **mle_p** – compute error through the asymptotic normality of the maximum likelihood estimator for the probability itself. WARNING: due to positivity constraint of the probability, this only works for high probability and low variance. Otherwise the standard error interval for the normal distribution (i.e. mle +- n*sigma) might give rise to negative lower error bars.
  - **mle_f** – compute error through the asymptotic normality of the maximum likelihood estimator for -log(p) (hence for the beta-scaled free energy). This estimation does not suffor from the same WARNING as for `mle_p`. Furthermore, in case of high probability and low variance, the error estimation using `mle_f` and `mle_p` are consistent (i.e. one can be computed from the other using f=-log(p)).
  - **None** – do not estimate the error.

Defaults to None, i.e. no error estimate. :type error_estimate: str, optional

Parameters

- **nsigma** (*int, optional*) – only relevant when error estimation is turned on (i.e. when keyword `error_estimate` is not None), this option defines how large the error interval should be in terms of the standard deviation sigma. A `nsigma=2` implies a 2-sigma error bar (corresponding to 95% confidence interval) will be returned. Defaults to 2
- **bias_subgrid_num** (*optional, defaults to 20*) – the number of grid points for the sub-grid used to compute the integrated boltzmann factor of the bias in each CV bin.
- **Nscf** (*int, defaults to 1000*) – the maximum number of steps in the self-consistent loop to solve the WHAM equations
- **convergence** (*float, defaults to 1e-6*) – convergence criterium for the WHAM self consistent solver. The SCF loop will stop whenever the integrated absolute difference between consecutive probability densities is less then the specified value.
- **verbose** (*bool, optional*) – set to True to turn on more verbosity during the self consistent solution cycles of the WHAM equations, defaults to False.
- **cv_unit** (*str, optional*) – the unit in which cv will be plotted/printed, defaults to 'au'
- **cv_label** (*str, optional*) – the label of the cv that will be used on plots, defaults to 'CV'

Returns probability histogram

Return type *Histogram1D*

`plot`(*fn*, *temp=None*, *flim=None*)
Make a plot of the probability histogram and possible the corresponding free energy (if the argument `temp` is specified).

Parameters

- **fn** (*str*) – file name of the resulting plot
- **temp** (*float, optional*) – the temperature for conversion of histogram to free energy profile. Specifying this number will add a free energy plot. Defaults to None
- **flim** (*float, optional*) – upper limit of the free energy axis in plots. Defaults to None

`histogram.`**`plot_histograms`**(*histograms*, *temp=None*, *labels=None*, *flims=None*, *colors=None*, *linestyles=None*, *linewidths=None*, *set_ref='min'*)
Make a plot to compare multiple probability histograms and possible the corresponding free energy (if the argument `temp` is specified).

Parameters

- **fn** (*str*) – file name to write the figure to, the extension determines the format (PNG or PDF).
- **histograms** (*list(*`Histogram1D`*)*) – list of histrograms to plot

- **temp** (*float/list(float)/np.ndarray, optional*) – if defined, the free energy profile corresponding to each histogram will be computed and plotted with its corresponding temperature. If a single float is given, all histograms are assumed to be at the same temperature, if a list or array of floats is given, each entry is assumed to be the temperature of the corresponding entry in the input list of histograms. Defaults to None
- **labels** (*list(str), optional*) – list of labels for the legend, one for each histogram. Defaults to None
- **flims** (*float, optional*) – [lower,upper] limit of the free energy axis in plots. Defaults to None
- **colors** (*List(str), optional*) – List of matplotlib color definitions for each entry in histograms. If an entry is None, a color will be chosen internally. Defaults to None, implying all colors are chosen internally.
- **linestyles** (*List(str), optional*) – List of matplotlib line style definitions for each entry in histograms. If an entry is None, the default line style of '-' will be chosen . Defaults to None, implying all line styles are set to the default of '-'.
- **linewidths** (*List(str), optional*) – List of matplotlib line width definitions for each entry in histograms. If an entry is None, the default line width of 1 will be chosen. Defaults to None, implying all line widths are set to the default of 2.
- **set_ref** (*str, optional*) – set the reference of each corresponding free energy profile, see documentation of the set_ref routine of the free energy profile class for more information on the allowed values. Defaults to min, implying each profile is shifted vertically untill the minimum value equals zero.

5.1.1.3 Collective variable – `thermodynamics.cv`

`class thermolib.thermodynamics.cv.`**`Average`**(*cv1*, *cv2*, *name=None*)

Class to implement a collective variable representing the average of two other collective variables:

CV = 0.5*(CV1 + CV2)

`class thermolib.thermodynamics.cv.`**`CoordinationNumber`**(*pairs*, *r0=3.779452267842504*, *nn=6*, *nd=12*, *name=None*, *unit_cell_pars=None*)

Class to implement a collective variable representing the coordination number of a certain atom pair or a set of atom pairs. If n atom pairs are defined, the coordination number should be a number between 0 and n.

CN = sum((1-x_ij^nn)/(1-x_ij^nd), ij in pairs)

with

x_ij = r_ij/r0

in which r_ij is the distance between the atoms of pair ij, r0 is a reference distance set to 2 angstrom by default but can ge defined by the user and nn and nd are integers that are set to 6,12 by default but can also be defined by the user.

class `thermolib.thermodynamics.cv.`**`Difference`**(*cv1*, *cv2*, *name=None*)

Class to implement a collective variable representing the difference between two other collective variables:

CV = CV2 - CV1

class `thermolib.thermodynamics.cv.`**`Distance`**(*index1*, *index2*, *name=None*, *unit_cell_pars=None*)

Class to implement a collective variable representing the distance between two atoms given by index1 and index2.

`compute`(*coords*, *deriv=True*)

Compute the value of the collective variable given the coordinates. If deriv is set to True, also compute and return the analytical derivative of Q towards the cartesian coordinates.

class `thermolib.thermodynamics.cv.`**`DistanceCOP`**(*index1*, *index2a*, *index2b*, *name=None*, *unit_cell_pars=None*)

Class to implement a collective variable representing the distance between an atom (index1) and the center of position (i.e. the geometric center) of two other atoms (index2a and index2b).

class `thermolib.thermodynamics.cv.`**`Minimum`**(*cv1*, *cv2*, *name=None*)

Class to implement a collective variable representing the minimum of two other collective variables:

CV = min(CV1,CV2)

class `thermolib.thermodynamics.cv.`**`OrthogonalDistanceToPore`**(*ring_indices*, *guest_indices*, *masses*, *unit_cell_pars=None*, *name=None*)

Class to implement a collective variable that represents the orthogonal distance between the center of mass of a guest molecule defined by its atom indices (guest_indices) on the one hand, and a pore ring defined by the atom indices of its constituting atoms (ring_indices) on the other hand.

5.1.1.4 Bias potentials – `thermodynamics.bias`

class `thermolib.thermodynamics.bias.`**`BiasPotential1D`**(*name*, *inverse_cv=False*)

A base class for 1-dimensional bias potentials. This abstract class serves as a parent for inheriting child classes which should implement the __call__ routine.

class `thermolib.thermodynamics.bias.`**`BiasPotential2D`**(*name*, *inverse_cv1=False*, *inverse_cv2=False*)

A base class for 2-dimensional bias potentials. This abstract class serves as a parent for inheriting child classes which should implement the __call__ routine.

class `thermolib.thermodynamics.bias.`**`MultipleBiasses1D`**(*biasses*, *coeffs=None*)

A class to add multiple bias potentials together, possibly weighted by given coefficients.

Parameters

- **biasses** (*list*(`BiasPotential1D`)) – list of bias potentials
- **coeffs** (*list/np.ndarray, optional*) – array of weigth coefficients. If not given, defaults to an array of ones (i.e. no weighting).

class `thermolib.thermodynamics.bias.`**`Parabola1D`**(*name*, *q0*, *kappa*, *inverse_cv=False*)

A 1-dimensional parabolic bias potential.

Parameters

- **name** (*str*) – a name for the bias potential
- **q0** (*float*) – the value of the parabola equilibruim (i.e. its minimum)
- **kappa** (*float*) – the force constant of the parabola
- **inverse_cv** (*bool, optional, defaults to False*) – If set to True, the CV-axis will be inverted prior to bias evaluation

class thermolib.thermodynamics.bias.**Parabola2D**(*name*, *q01*, *q02*, *kappa1*, *kappa2*, *inverse_cv1=False*, *inverse_cv2=False*)

A 2-dimensional parabolic bias potential.

Parameters

- **name** (*str*) – a name for the bias potential
- **q01** (*float*) – the value of the first collective variable corresponding to the parabola minimum
- **q02** (*float*) – the value of the second collective variable corresponding to the parabola minimum
- **kappa1** – the force constant of the parabola in the direction of the first collective variable
- **kappa2** – the force constant of the parabola in the direction of the second collective variable
- **inverse_cv1** (*bool, optional, defaults to False*) – If set to True, the CV1-axis will be inverted prior to bias evaluation
- **inverse_cv2** (*bool, optional, defaults to False*) – If set to True, the CV2-axis will be inverted prior to bias evaluation

class thermolib.thermodynamics.bias.**PlumedSplinePotential1D**(*name*, *fn*, *inverse_cv=False*, *unit='au'*, *scale=1.0*)

A bias potential read from a PLUMED file, which is spline-interpolated.

Parameters

- **name** (*str*) – name for the external bias potential
- **fn** (*str*) – specifies the filename of an external potential written on a grid and acting on the collective variable, as used with the EXTERNAL keyword in PLUMED.
- **unit** (*str, optional*) – unit used to express the external potential, defaults to 'au'
- **scale** (*float, optional*) – scaling factor for the external potential (useful to invert free energy surfaces), default to 1.0

class thermolib.thermodynamics.bias.**Polynomial1D**(*name*, *coeffs*, *inverse_cv=False*, *unit='au'*)

Bias potential given by general polynomial of any degree.

Parameters

- **name** (*str*) – name of the bias
- **coeffs** (*list/np.ndarray*) – list of expansion coefficients of the polynomial in increasing order starting with the coefficient of power 0. The degree of the polynomial is given by len(coeffs)-1

5.1.1.5 Conditional probability – `thermodynamics.condprob`

class thermolib.thermodynamics.condprob.**ConditionalProbability1D1D**(*q1s*, *cvs*, *q1_label='Q'*, *cv_label='CV'*)

Routine to store and compute conditional probabilities of the form $p(q_1|cv)$ which allow to convert a free energy profile in terms of the collective variable cv to a free energy profile in terms of the collective variable q_1.

process_trajectory_cvs(*fns*, *col_q1=1*, *col_cv=2*, *finish=True*)

Compute the conditional probability p(q1|cv) (and norm for final normalisation) by processing a series of CV trajectory files. Each CV trajectory file contains rows of the form

time q1 cv

If the trajectory file contains this data in a different order, it can be accounted for using the col_xx keyword arguments.

process_trajectory_xyz(*fns*, *Q1*, *CV*, *finish=True*)

Compute the conditional probability p(q1|cv) (and norm for final normalisation) by processing a series of XYZ trajectory files. The final probability is estimated as the average over all given files. These files may also contain data from biased simulations.

transform(*fep*, *cv_unit='au'*, *f_unit='kjmol'*, *cv_label='CV'*)

Transform the provided 1D FES to a different 1D FES using the current conditional probability according to the formula

$$F(q) = -kT \ln\left(\int p(q|v) \cdot e^{-\beta F(v)} dv\right)$$

Parameters

- **fep** (*(child of)* `BaseFreeEnergyProfile`) – the free energy profile F(v) which will be transformed
- **cv_unit** (*str, optional, defaults to atomic units*) – the unit to be used in plotting and printing of the new cv
- **f_unit** (*str, optional, defaults to kJ/mol*) – the unit of the the transformed free energy profile to be used in plotting and printing of energies
- **cv_label** (*str, optional, defaults to CV*) – the label of the new collective variable to be used in plot labels

class thermolib.thermodynamics.condprob.**ConditionalProbability1D2D**(*q1s*, *q2s*, *cvs*, *q1_label='Q1'*, *q2_label='Q2'*, *cv_label='CV'*)

Class to store and compute conditional probabilities of the form $p(q_1, q_2|cv)$ which can be used to transform a 1D free energy profile in terms of the collective variable cv towards a 2D free energy surface in terms of the collective variables q_1 and q_2.

process_trajectory_cvs(*fns*, *col_q1=1*, *col_q2=2*, *col_cv=3*, *finish=True*)

Routine to update conditional probability $p(q_1, q_2|v)$ (and norm for final normalisation) by processing a series of CV trajectory file. Each CV trajectory file contains rows of the form

> time q1 q2 v

If the trajectory file contains this data in a different order, it can be accounted for using the col_xx keyword arguments. Similar constraints apply to these CV trajectory files as specified in the routine `process_trajectory_xyz()`.

Warning: after all trajectories have been processes, you need to manually call the finish routine!

process_trajectory_xyz(*fns*, *Q1*, *Q2*, *CV*, *finish=True*)

Compute the conditional probability $p(q_1, q_2|v)$ (and norm for final normalisation) by processing a series of XYZ trajectory files. The final probability is estimated as the average over all given files. These files may also contain data from biased simulations as long as the bias is constant over the simulation. For example, data from Umbrella Sampling is OK, while data from metadynamica itself is not. Data obtained from a regular MD with the final MTD profile as bias is OK.

transform(*fep*, *cv1_unit='au'*, *cv2_unit='au'*, *f_unit='kjmol'*, *label1=None*, *label2=None*)

Transform the provided 1D FEP to a 2D FES using the current conditional probability according to the formula

$$F(q_1, q_2) = -kT \cdot \ln\left(\int p(q_1, q_2|v) \cdot e^{-\beta F(v)} dv\right)$$

5.1.2 Kinetic Modules

5.1.2.1 Rate constant from transition state theory – `kinetics.rate`

class thermolib.kinetics.rate.**RateFactorEquilibrium**(*CV*, *CV_TS_lims*, *temp*, *CV_unit='au'*)

Class to compute the factor A required for the computation of the rate constant k of the process of crossing a transition state:

$$k = A \cdot \frac{e^{-\beta F(q_{TS})}}{\int_{-\infty}^{q_{TS}} e^{-\beta F(q)} dq}$$

$$A = \frac{1}{2} \left\langle |\dot{Q}| \right\rangle_{TS}$$

Herein, the subscript TS refers to the fact that the average has to be taken only at configurational states corresponding to the transition state (TS). Furthermore, the average contains an integral over configurational phase space as well as momenta. The integral over momenta can either be performed analytical or numerical by taking momentum samples according to a certain distribution.

- When performing the momentum integration analytical, the above expression simplifies to:

$$A = \sqrt{\frac{kT}{2\pi}} \cdot \left\langle \left| |\vec{\nabla}_x Q| \right| \right\rangle_{TS}$$

where the average is now only taken in configurational space for states corresponding to the transition state (hence still the subscript TS). Furtheremore, $\vec{\nabla}_x Q$ represents the gradient of the collective variable Q towards the mass-weighted cartesian coordinates.

- When performing the momentum integral numerically, the expression for A can be rewritten as:

$$A = \frac{1}{2}\int_{\vec{x}\in TS}\left[\int_{-\infty}^{+\infty}\left|\dot{Q}(\vec{x},\vec{P})\right| p_p(\vec{P})d\vec{P}\right] p_x(\vec{x})d\vec{x}$$

where $\dot{Q}(\vec{x},\vec{P})$ indicates that the time derivative of Q depends on both configurational state indicated by its mass-weighted cartesian coordinates $\vec{x}$ as well as the mass-weighted momenta $\vec{P}$. Furthermore $p_p(\vec{P})$ represents the momentum probability distribution which will need to be specified (eg. the Maxwell-Boltzmann distribution). The integral over $\vec{x}$ is computed using the samples $\vec{x}_i$ taken from the MD trajectory while the integration over $\vec{P}$ is computed numerically by taking random samples $\vec{P}_k$ from the given momentum distribution p_p:

$$A = \frac{1}{N_s}\sum_{i=1}^{N_s} A(\vec{x}_i)$$

$$A(x_i) = \frac{1}{2N_p}\sum_{k=1}^{N_p}\left|\dot{Q}\left(\vec{x}_i,\vec{P}_k\right)\right|$$

These computational methods are implemented in the `RateFactorEquilibrium.proces_trajectory()` routine.

Parameters

- **CV** (*callable*) – a function that computes the value (and gradient) of the collective variable
- **CV_TS_lims** (*list(float)*) – the lower and upper boundaries for determining whether a certain frame of the trajectory corresponds with the transition state (TS). In other words, the condition CV(R)=CV_TS is replaced by CV_TS_lims[0]<=CV(R)<=CV_TS_lims[1]
- **masses** (*np.ndarray(float)*) – the masses of the atoms (should be consistently indexed as the coords argument parsed to the compute_contribution routine).
- **temp** (*float*) – temperature
- **CV_unit** (*str*) – the unit for printing the collective variable

process_trajectory(*fn_xyz*, *finish=True*, *fn_samples=None*, *momenta='analytical'*, *Nmomenta=500*, *verbose=False*)

Process the given XYZ trajectory and store T and N values.

Parameters

- **fn_xyz** (*str*) – filename of the trajectory from which the rate factor will be computed.

- **finish** (*bool, optional, default=True*) – when set to True, the finish routine will be called after processing the trajectory, which will finalize storing the data. If multiple trajectory from different files need to be read, set finish to False for all but the last trajectory.
- **fn_samples** (*str, optional, default=None*) – write the rate factor samples (i.e. the As array) to the given file. This feature is switched off by specifying None.
- **momenta** (*string, optional, default=analytical*) – specify how to compute the momentum part of the phase space integral in computing the As samples (see description above). The following options are available:
  - **analytical** – compute the momentum integral analytically, is hence the fastest method
  - **MB** – compute the momentum integral numerical by taking random samples for the velocity from the Maxwell-Boltzmann distribution.
- **Nmomenta** (*float, optional, default=500*) – the number of momentum samples taken from the given distribution in case of numerical momentum integration. This keyword is only relevant when momenta is not set to analytical.
- **verbose** (*bool, optional, default=False*) – increase verbosity by setting to True.

5.1.3 General tools

`thermolib.tools.`**blav**(*data*, *blocksizes=None*, *fitrange=[0, - 1]*, *exponent=1*, *fn_plot=None*, *unit='au'*, *plot_ac=False*, *ac_range=None*, *acft_plot_range=None*)

Routine to implement block averaging. This allows to estimate the sample error on correlated data by fitting a model function towards to the naive (i.e. as if uncorrelated) sample error on the block averages as function of the blocksize. This model function is based on the following model for the integrated correlation time τ between the block averages:

$$\tau = 1 + \frac{t_0 - 1}{B^n}$$

As a result, the model for the naive error estimate on the block averages becomes

$$error = TE \cdot \frac{B^n}{B^n + t_0 - 1}$$

in which B represents the block size, TE the true error, t_0 the correlation time for the original time series ($B = 1$) and n the exponential rate with which the block average integrated correlation time decreases as function of block size.

Parameters

- **data** (*np.ndarray*) – 1D array representing the data to be analyzed
- **blocksizes** (*np.ndarray, optional*) – array of block sizes, defaults to np.arange(1,len(data)+1,1)
- **fitrange** (*list, optional*) – range of blocksizes to which fit will be performed, defaults to [0,-1]

- `exponent` (*int, optional*) – the exponent of the blocksize in the models for the auto correlation time and correlated sample error, defaults to 1
- `fn_plot` (*str, optional*) – file name to which to write the plot. No plot is made if set to None. Defaults to None
- `unit` (*str, optional*) – unit in which to plot the data, defaults to 'au'
- `plot_ac` (*bool, optional*) – indicate whether or not to compute and plot the auto correlation function as well (might take some time), defaults to False
- `ac_range` (*np.ndarray, optional*) – the range for which to plot the auto correlation function, only relevant if `plot_ac` is set to True, defaults to np.arange(0,501,1)
- `acft_plot_range` (*np.ndarray, optional*) – the range for which to plot the fourier transform of the auto correlation function, only relevant if `plot_ac` is set to True, defaults to entire freq range

Returns mean, error, corrtime

- **mean** (*float*) – the sample mean
- **error** (*foat*) – the error on the sample mean
- **corrtime** (*float*) – the correlation time (in units of the timestep) of the original sample data

`thermolib.tools.`**`integrate`**(*xs*, *ys*)

A simple integration method using the trapezoid rule

Parameters

- `xs` (*np.ndarray*) – array containing function argument values on grid
- `ys` (*np.ndarray*) – array containing function values on grid

`thermolib.tools.`**`integrate2d`**(*z*, *x=None*, *y=None*, *dx=1.0*, *dy=1.0*)

Integrates a regularly spaced 2D grid using the composite trapezium rule.

Parameters

- `z` (*np.ndarray(flt)*) – 2 dimensional array containing the function values
- `x` (*np.ndarray(flt), optional*) – 1D array containing the first function argument values, is used to determine grid spacing of first function argument. If not given, optional argument dx is used to define the grid spacing. Defaults to None
- `y` (*np.ndarray(flt), optional*) – 1D array containing the second function argument values, is used to determine grid spacing of second function argument. If not given, optional argument dy is used to define the grid spacing. Defaults to None
- `dx` (*float, optional*) – grid spacing for first function argument. If not given, argument is used to determine grid spacing. Defaults to 1.
- `dy` (*float, optional*) – grid spacing for second function argument. If not given, argument is used to determine grid spacing. Defaults to 1.

Returns integral value

Return type float

thermolib.tools.**read_wham_input**(*fn*, *path_template_colvar_fns='%s'*, *colvar_cv_column_index=1*, *kappa_unit='kjmol'*, *q0_unit='au'*, *start=0*, *end=- 1*, *stride=1*, *bias_potential='Parabola1D'*, *additional_bias=None*, *verbose=False*)

Read the input for a WHAM reconstruction of the free energy profile from a set of Umbrella Sampling simulations. The file specified by fn should have the following format:

```
T = XXXK
NAME1 Q1 KAPPA1
NAME2 Q2 KAPPA2
NAME3 Q3 KAPPA3
...
```

when a line starts with a T, it is supposed to specify the temperature. If no temperature line is found, a temperature of None will be returned. All other lines define the bias potential for each simulation as a parabola centered around `Qi` and with force constant `KAPPAi` (the units used in this file can be specified the keyword arguments `kappa_unit` and `q0_unit`). For each bias with name `NAMEi` defined in this file, there should be a colvar trajectory file accessible through the path given by the string 'path_template_colvar_fns %(NAMEi)'. For example, if path_template_colvar_fns is defined as 'trajectories/%s/COLVAR' and the wham input file contains the following lines:

```
T = 300K
Window1/r1 1.40 1000.0
Window2/r2 1.45 1000.0
Window3/r1 1.50 1000.0
...
```

Then the colvar trajectory file of the first potential can be found through the path (relative to the wham input file) 'Window1/r1/COLVAR' and so on. These colvar files contain the trajectory of the relevant collective variable during the biased simulation. Finally, these trajectory files should be formatted as outputted by PLUMED:

```
time_1 cv_value_1
time_2 cv_value_2
time_3 cv_value_3
...
```

where the cv values again have a unit that can be specified by the keyword argument `q0_unit`.

Parameters

- **fn** (*str*) – file name of the wham input file
- **path_template_colvar_fns** (*str. Defaults to '%s'*) – Template for defining the path (relative to the directory containing the wham input file given by argument fn) to the colvar trajectory file corresponding to each bias. See documentation above for more details. This argument should be string containing a single '%s' substring.
- **kappa_unit** (*str, optional*) – unit used to express kappa in the wham input file, defaults to 'kjmol'

- **q0_unit** (*str, optional*) – unit used to express q0 in the wham input file as well as the cv values in the trajectory files, defaults to 'au'
- **stride** (*int, optional*) – defines the sub sampling applied to the trajectory data to deal with correlations. For example a stride of 10 means only taking 1 in 10 samples and throw away 90% of the data. Defaults to 1 (i.e. no sub sampling).
- **start** (*int, optional*) – defines the start point from which to take samples into account. This can be usefull for eliminating equilibration times as well as for taking various subsamples trajectories from the original data each starting at different timesteps. Defaults to 0.
- **end** (*int, optional*) – defines the end point to which to take samples into account. This can be usefull when it is desired to cut the original trajectory into blocks. Defaults to -1.
- **bias_potential** – mathematical form of the bias potential used, allowed values are
  - **parabola/harmonic** – harmonic bias of the form 0.5*kappa*(q-q0)**2

defaults to parabola

Parameters

- **additional_bias** (BiasPotential1D, *optional*) – A single additional bias that is added for each simulation on top of the simulation-specific biases. Defaults to None
- **verbose** (*bool, optional*) – increases verbosity if set to True, defaults to False

Returns

temp, biasses, trajectories:

- **temp** (float) – temperature at which the simulations were performed
- **biasses** (list of callables) – list of bias potentials (defined as callable functions) for all Umbrella Sampling simulations
- **trajectories** (list of np.ndarrays) – list of trajectory data arrays containing the CV trajectory for all Umbrella Sampling simulations

thermolib.tools.**read_wham_input_2D**(*fn*, *path_template_colvar_fns='%s'*, *kappa1_unit='kjmol'*, *kappa2_unit='kjmol'*, *q01_unit='au'*, *q02_unit='au'*, *start=0*, *end=- 1*, *stride=1*, *bias_potential='Parabola2D'*, *verbose=False*)

Read the input for a WHAM reconstruction of the 2D free energy surface from a set of Umbrella Sampling simulations. The file specified by fn should have the following format:

```
T = XXXK
NAME_1 Q01_1 Q02_1 KAPPA1_1 KAPPA1_1
NAME_2 Q01_2 Q02_2 KAPPA1_2 KAPPA1_2
NAME_3 Q01_3 Q02_3 KAPPA1_3 KAPPA1_3
...
```

when a line starts with a T, it is supposed to specify the temperature. If no temperature line is found, a temperature of None will be returned. All other lines define the bias potential for each simulation as a parabola centered around (`Q01_i`,``Q02_i``) and with force constants (`KAPPA1_i`,``KAPPA2_i``) (the units used in this file can be specified the keyword arguments `kappa1_unit`, `kappa2_unit`, `q01_unit` and `q02_unit`). For each bias with name `NAME_i` defined in this file, there should be a colvar trajectory file accessible through the path given by the string 'path_template_colvar_fns %(NAME_i)'. For example, if path_template_colvar_fns is defined as 'trajectories/%s/COLVAR' and the wham input file contains the following lines:

```
T = 300K
Window1/r1 1.40 -0.2 1000.0 1000.0
Window2/r2 1.45 -0.2 1000.0 1000.0
Window3/r1 1.50 -0.2 1000.0 1000.0
...
```

Then the colvar trajectory file of the first potential can be found through the path (relative to the wham input file) 'Window1/r1/COLVAR' and so on. These colvar files contain the trajectory of the relevant collective variable during the biased simulation. Finally, these trajectory files should be formatted as outputted by PLUMED:

```
time_1 cv1_value_1 cv2_value_1
time_2 cv1_value_2 cv2_value_2
time_3 cv1_value_3 cv2_value_3
...
```

where the cv1 and cv2 values again have a unit that can be specified by the keyword arguments `q01_unit` and `q02_unit` respectively.

Parameters

- **fn** (*str*) – file name of the wham input file
- **path_template_colvar_fns** (*str. Defaults to '%s'*) – Template for defining the path (relative to the directory containing the wham input file given by argument fn) to the colvar trajectory file corresponding to each bias. See documentation above for more details. This argument should be string containing a single '%s' substring.
- **kappa1_unit** (*str, optional*) – unit used to express the CV1 force constant kappa1 in the wham input file, defaults to 'kjmol'
- **kappa2_unit** (*str, optional*) – unit used to express the CV2 force constant kappa1 in the wham input file, defaults to 'kjmol'
- **q01_unit** (*str, optional*) – unit used to express q01 in the wham input file as well as the cv values in the trajectory files, defaults to 'au'
- **q02_unit** (*str, optional*) – unit used to express q02 in the wham input file as well as the cv values in the trajectory files, defaults to 'au'
- **stride** (*int, optional*) – defines the sub sampling applied to the trajectory data to deal with correlations. For example a stride of 10 means only taking 1 in 10 samples and throw away 90% of the data. Defaults to 1 (i.e. no sub sampling).
- **start** (*int, optional*) – defines the start point from which to take samples

into account. This can be usefull for eliminating equilibration times as well as for taking various subsamples trajectories from the original data each starting at different timesteps. Defaults to 0.

- **end** (*int, optional*) – defines the end point to which to take samples into account. This can be usefull when it is desired to cut the original trajectory into blocks. Defaults to -1.
- **bias_potential** – mathematical form of the bias potential used, allowed values are
  - **parabola2D/harmonic2D** – harmonic bias of the form 0.5*kappa1*(q1-q01)**2 + 0.5*kappa2*(q2-q02)**2

defaults to parabola2D

Parameters **verbose** (*bool, optional*) – increases verbosity if set to True, defaults to False

Raises `ValueError` – when a line in the wham input file cannot be interpreted

Returns

temp, biasses, trajectories:

- **temp** (float) – temperature at which the simulations were performed
- **biasses** (list of callables) – list of bias potentials (defined as callable functions) for all Umbrella Sampling simulations
- **trajectories** (list of np.ndarrays) – list of trajectory data arrays containing the CV trajectory for all Umbrella Sampling simulations

`thermolib.tools.`**`trajectory_xyz_to_CV`**(*fns*, *CV*)

Compute the CV along an XYZ trajectory. The XYZ trajectory is assumed to be composed out of a (list of subsequent) XYZ files.

Parameters

- **fns** (*str or list(str)*) – (list of) names of XYZ trajectory file(s) containing the xyz coordinates of the system
- **CV** (*one from thermolib.thermodynamics.cv.__all__*) – collective variable defining how to compute the collective variable along the trajectory

Returns array containing the CV value along the trajectory

Return type np.ndarray(flt)

CHAPTER

SIX

THEORETICAL BACKGROUND

In this section we provide the theoretical background for all various transformations and relations used in ThermoLIB.

6.1 Theoretical background

In this section we provide the theoretical background for all various transformations and relations used in ThermoLIB.

6.1.1 General statistical physics

6.1.1.1 Partition function and total free energy

In this section, we outline the fundamental theoretical background from thermodynamics and statistical physics applied in the ThermoLIB package. The main assumption is that we can describe the dynamics of the system using classical mechanics by means of the Hamiltonian:

$$\mathcal{H}\left(\vec{r}^N,\vec{p}^N\right) = \sum_{i=1}^{N}\frac{\vec{p}_i^2}{2m_i} + V\left(\vec{r}^N\right)$$

Herein, m_i represents the mass of particle i and $V\left(\vec{r}^N\right)$ the potential energy of the N particles at positions denoted as $\vec{r}^N = (\vec{r}_1,\vec{r}_2,\ldots,\vec{r}_N)$. In order to translate this microscopic description of the molecular interactions towards macroscopically observable thermodynamic properties, we apply the theory of statistical physics and construct the partition function Z_{NVT}. We will illustrate everything in the canonical ensemble (i.e. control over particle number N, volume V and temperature T) but it should be straightforward to generalize to other ensembles (). The classical partition function in the canonical ensemble is given by:

$$Z(N,V,T) = \frac{1}{h^{3N}}\iint e^{-\beta\mathcal{H}\left(\vec{r}^N,\vec{p}^N\right)}d\vec{r}^N d\vec{p}^N$$

Note that we assume that each particle in the system is distinguishable, which is indeed the case for purely crystalline materials as the crystal position allows to distinguish between the various atoms. However, whenever interchangeability makes (some) particles indistinguishable, adequate correction factors are required. This is e.g. the case when guest molecules are adsorbed inside a nanoporous framework. If n molecules are adsorbed, a correction factor of $\frac{1}{n!}$ needs to be included in the partition function. We proceed by using the fact that the kinetic and potential degrees of freedom are fully decoupled, implying that the partition function can be factorized. Furthermore, since the kinetic energy takes on a simple quadratic form, its contribution to the partition function can be integrated analytically. Hence we arrive at:

$$\begin{aligned}
Z(N,V,T) &= \frac{1}{h^{3N}}\int e^{-\beta\sum_i\frac{\vec{p}_i^2}{2m_i}}d\vec{p}^N\int e^{-\beta V\left(\vec{r}^N\right)}d\vec{r}^N \\
&= \frac{1}{h^{3N}}\left[\prod_{i=1}^N\int e^{-\beta\frac{\vec{p}_i^2}{2m_i}}d\vec{p}_i\right]\int e^{-\beta V\left(\vec{r}^N\right)}d\vec{r}^N \\
&= \frac{1}{\prod_i\Lambda_i^3}\int e^{-\beta V\left(\vec{r}^N\right)}d\vec{r}^N
\end{aligned}$$

where we used the standard Gaussian integral in the third line and introduced the thermal wavelength for a particle with mass m_i:

$$\Lambda_i = \sqrt{\frac{h^2\beta}{2\pi m_i}}$$

We clearly see that the partition function can be factorized in a contribution due to the kinetic degrees of freedom and a contribution due to the potential energy:

$$\begin{aligned}
Z(N,V,T) &= Z^{\text{kin}}(N,V,T)Q(N,V,T) \\
Z^{\text{kin}}(N,V,T) &= \frac{V_0^N}{\prod_i\Lambda_i^3} \\
Q(N,V,T) &= \frac{1}{V_0^N}\int e^{-\beta V\left(\vec{r}^N\right)}d\vec{r}^N
\end{aligned}$$

Herein, V_0 is an arbitrary volume introduced to make the separate contributions to the partition function dimensionless and drops out when considering the total partition function or when computing differences of Z^{kin} and Q. Q is sometimes referred to as the configurational contribution to the partition function. If one chooses V_0 to be the accessible volume of each atom in absence of any forces (i.e. $V(\vec{r}^N)=0$), then Z^{kin} represents the ideal gas partition function. Once the partition function is known, the corresponding free energy can be computed from which all thermodynamic observables can be computed:

$$\begin{aligned}
F(N,V,T) &= -k_BT\ln Z(N,V,T) \\
&= F^{kin}(N,V,T) + F^{conf}(N,V,T)
\end{aligned}$$

with F^{kin} the kinetic contribution to the free energy:

$$\begin{aligned}
F^{kin}(N,V,T) &= -k_BT\ln Z^{kin}(N,V,T) \\
&= -Nk_BT\ln V_0 + \frac{3}{2}k_BT\sum_{i=1}^N\ln\left(\frac{h^2\beta}{2\pi m_i}\right)
\end{aligned}$$

and F^{conf} the configurational contribution:

$$F^{conf}(N,V,T) = -k_BT\ln Q(N,V,T)$$

6.1.1.2 Collective variables and free energy profiles

1D free energy profile

One of the basic features of ThermoLIB is the construction and manipulation of free energy profiles (FEP) as function of a well chosen collective variable to describe the thermodynamic stability of a certain process. The collective variable $X(\vec{r}^N)$ represents a degree of freedom as a function of the coordinates $\vec{r}^N$

and describes the progress of the system during the process of interest. One can then define a macrostate x of the system as the collection of microstates $\vec{r}^N$ for which the collective variable takes on a value for $X(\vec{r}^N)$ within the interval $[x, x+dx]$. The configurational partition function of this macrostate and the corresponding configurational free energy is given by:

$$\begin{aligned} Q(x;N,V,T) &= \frac{1}{V_0^N}\int \delta(X(\vec{r}^N)-x)e^{-\beta V(\vec{r}^N)}d\vec{r}^N \\ F^{conf}(x;N,V,T) &= -k_BT\ln Q(x;N,V,T) \end{aligned}$$

Herein, $\delta(x)$ represents the Dirac delta distribution. The original total partition function $Q(N,V,T)$ can be found as the integral over x which reconstructs the integral over entire phase space:

$$\begin{aligned} \int_{-\infty}^{+\infty} Q(x;N,V,T)dx &= \int_{-\infty}^{+\infty}\left[\frac{1}{V_0^N}\int \delta(X(\vec{r}^N)-x)e^{-\beta V(\vec{r}^N)}d\vec{r}^N\right]dx \\ &= \frac{1}{V_0^N}\int\left[\int_{-\infty}^{+\infty}\delta(X(\vec{r}^N)-x)dx\right]e^{-\beta V(\vec{r}^N)}d\vec{r}^N \\ &= \frac{1}{V_0^N}\int e^{-\beta V(\vec{r}^N)}d\vec{r}^N \\ &= Q(N,V,T) \end{aligned}$$

In the third line, we used the defining property of the Dirac delta distribution. As a result, we can express the probability of finding the system in a macrostate with value for $X(\vec{r}^N)$ in the interval $[x, x+dx]$ as follows:

$$p(x;N,V,T)dx = \frac{Q(x;N,V,T)}{Q(N,V,T)}dx$$

As we will only be interested in processes that can be described by collective variables depending only on the coordinates $\vec{r}^N$ and not on the momenta), we will also only be interested in the configurational contributions to partition functions and free energies (as the kinetic contribution will not change along this process). Hence, from now on we will drop the superscript *conf* and we will also drop the functional dependence on N, V and T to not over encumber the notation:

$$\begin{aligned} Q(x) &= \frac{1}{V_0^N}\int \delta(X(\vec{r}^N)-x)e^{-\beta V(\vec{r}^N)}d\vec{r}^N \\ Q &= \int_{-\infty}^{+\infty} Q(x)dx \\ F(x) &= -k_BT\ln Q(x) \\ F &= -k_BT\ln Q \\ p(x) &= \frac{Q(x)}{Q} \end{aligned} \tag{6.1}$$

2D free energy surface

In many cases, when a system is undergoing a process we require 2 collective variables , i.e. X and Y, to adequately describe the progress of the process. The corresponding 2D partition function, probability density and free energy surface, which describe the system in the state defined by $X(\vec{r}^N) \in [x, x+dx]$ and $Y(\vec{r}^N) \in [y, y+dy]$, are given by:

$$\begin{aligned} Q(x,y) &= \frac{1}{V_0^N}\int \delta(X(\vec{r}^N)-x)\delta(Y(\vec{r}^N)-y)e^{-\beta V(\vec{r}^N)}d\vec{r}^N \\ p(x,y) &= \frac{Q(x,y)}{Q} \\ F(x,y) &= -k_BT\ln Q(x,y) \end{aligned}$$

One can easily make the link towards the 1D free energy profile as follows:

$$\begin{aligned} Q(x) &= \int_{-\infty}^{+\infty} Q(x,y)dy \\ p(x) &= \int_{-\infty}^{+\infty} p(x,y)dy \\ F(x) &= -k_B T \ln\left(\int_{-\infty}^{+\infty} e^{-\beta F(x,y)}dy\right) \end{aligned}$$

In what follows, we will derive various transformation formulas used by ThermoLIB for transforming between various collective variables, general projections of higher dimensional free energy surfaces (FES) to lower dimensional FES or *deprojecting* lower dimensional FES to higher dimension FES.

6.1.2 Transformations and projections of the FES

6.1.2.1 Transforming the FEP from one CV to another

Consider a system undergoing a process that can be described by either one of two collective variables $X(\vec{r}^N)$ or $Y(\vec{r}^N)$. In other words, the progress of the system along the considered process can be adequately quantified by the value of either X or Y. For such a process, one can construct a free energy profile as a function of either x (resulting in $F_x(x)$) or y (resulting in $F_y(y)$). The resulting two free energy profiles do not necessarily represent the same physical property. More specifically, if there exists a bijection between X and Y linking the value of x uniquely to that of y through the transformation formula $y = h(x)$ (which is not always the case as we will discuss further on), then in general:

$$F_x(x) \neq F_y(h(x))$$

This is due to the fact that $F_x(x)$ represents the free energy of the macrostate corresponding to all microstates for which $X(\vec{r}^N) = x$, while $F_y(y)$ represents the free energy of the macrostate corresponding to all microstates for which $Y(\vec{r}^N) = y$. These two collections in general do not contain the same set of microstates, i.e. they do not represent the same subspace of phase space. Therefore, the free energy is not necessarily the same. As we will see later, only in the case where X and Y are connected by a linear transformation, are the free energies identical (up to a additive constant). Furthermore, it is not always the case one can identify a unique bijection between X and Y. However, one can then usually still determine a probabilistic correlation etween X and Y in the form of a conditional probability. We will go into more details for both situations in the following paragraphs.

Deterministic relation

Assume one can define a unique relation $y = h(x)$ between the value of two collective variables X and Y. To derive the transformation formula for the free energy, i.e. the formula allowing to compute $F_x(x)$ from $F_y(y)$, we start by expressing that the probability of finding the system in $X(\vec{r}^N) \in [x, x + dx]$ is equal to the probability of finding the system in $Y(\vec{r}^N) \in [y, y + dy]$ with $y = h(x)$:

$$\begin{aligned} p_x(x)dx &= p_y(y)dy \\ \frac{Q_x(x)}{Q}dx &= \frac{Q_y(y)}{Q}dy \\ Q_x(x) &= Q_y(y)\left|\frac{dy}{dx}\right| \\ Q_x(x) &= Q_y(h(x))\left|h'(x)\right| \end{aligned}$$

The absolute value in the third line is introduced because both partition functions $Q_x(x)$ and $Q_y(y)$ are fundamentally positive quantities. As a result, the corresponding free energy profiles transform as follows:

$$\begin{aligned} F_x(x) &= -k_B T \ln Q_x(x) \\ &= -k_B T \ln \big(Q_y(h(x)) h'(x) \big) \\ &= -k_B T \ln Q_y(h(x)) - k_B T \ln h'(x) \end{aligned}$$

As such, we arrive at the transformation formula between $F_x(x)$ and $F_y(y)$:

$$F_x(x) = F_y(h(x)) - k_B T \ln \left| h'(x) \right| \tag{6.2}$$

Therefore, only for a linear transformation for which $h'(x) = cte$, will the last term represent a meaningless additive constant and will the two free energy profiles coincide. However, when the transformation is not linear, an extra term $-k_B T \ln |h'(x)|$ figures in the transformation formula. This term is related to the difference in width of the intervals $[x, x+dx]$ and $[y, y+dy]$ defining the macrostates corresponding to a given value of X and Y. This can be generalized to multidimensional free energy surfaces (FES). Consider for example a 2D free energy surface $F_x(x_1, x_2)$ subject to the following transformation:

$$\begin{aligned} y_1 &= h_1(x_1, x_2) \\ y_2 &= h_2(x_1, x_2) \end{aligned}$$

The transformation formula for the FES then becomes:

$$F_x(x_1, x_2) = F_y(h_1(x_1, x_2), h_2(x_1, x_2)) - k_B T \ln J(x_1, x_2)$$

with J the Jacobian of the transformation:

$$J(x_1, x_2) = \left| \frac{\partial [y_1, y_2]}{\partial [x_1, x_2]} \right| = \begin{vmatrix} \frac{\partial h_1}{\partial x_1} & \frac{\partial h_1}{\partial x_2} \\ \frac{\partial h_2}{\partial x_1} & \frac{\partial h_2}{\partial x_2} \end{vmatrix}$$

Probabilistic correlation

A different situation arises when we cannot define a unique relation between a state of $X(\vec{r}^N) = x$ and a state of $Y(\vec{r}^N) = y$. However, one can usually still define a probabilistic correlation through the conditional probability $p_c(x|y)$ expressing the probability of the system for being observed with a value $X(\vec{r}^N) = x$ given that is was prepared in a state for which $Y(\vec{r}^N) = y$. To derive the transformation formula between $F_x(x)$ and $F_y(y)$, we start from the following result of probability theory:

$$\begin{aligned} p_x(x) &= \int_{-\infty}^{+\infty} p_{xy}(x, y) dy \\ &= \int_{-\infty}^{+\infty} p_{x|y}(x|y) p_y(y) dy \end{aligned}$$

By using the relation between probability density and the partition function as well as the relation between free energy and partition function (Eq. (**??**)), we arrive at:

$$\begin{aligned} e^{-\beta F_x(x)} &= \int_{-\infty}^{+\infty} p_{x|y}(x|y) e^{-\beta F_y(y)} dy \\ F_x(x) &= -k_B T \ln \left(\int_{-\infty}^{+\infty} p_{x|y}(x|y) e^{-\beta F_y(y)} dy \right) \end{aligned} \tag{6.3}$$

Note that this equation actually represents a generalization of the previous situation for a deterministic relation. Indeed, when the relation between X and Y can be expressed by a deterministic relation $y = h(x)$, we can express the conditional probability as:

$$p_{x|y}(x|y) = \delta(x - h^{-1}(y)) = \frac{1}{\left|\frac{d}{dy}h^{-1}(h(x))\right|}\delta(y-h(x)) = \left|h'(x)\right|\delta(y-h(x))$$

As a result, the transformation formula (**??**) becomes:

$$\begin{aligned} e^{-\beta F_x(x)} &= \left|h'(x)\right| \int_{-\infty}^{+\infty} \delta(y-h(x))e^{-\beta F_y(y)}dy \\ &= \left|h'(x)\right| e^{-\beta F_y(h(x))} \\ F_x(x) &= F_y(h(x)) - k_BT\ln\left|h'(x)\right| \end{aligned}$$

which is indeed identical to the transformation formula (**??**) for the case of a deterministic bijection.

6.1.2.2 Projecting free energy surfaces

In this section, we investigate how one could project a higher dimensional FES towards a lower dimensional FES. More specifically, we consider the situation where we start from a 2D FES $F_{xy}(x, y)$ in terms of two collective variables X and Y and want to compute the free energy of macrostates defined by a collective variable Q. Again, we can consider two situations: deterministic relation between X, Y and Q encoded by the expression $Q(x, y)$ or a probabilistic correlation encoded by the conditional probability $p_{q|xy}(q|x, y)$. However, just as it was the case for the transformations of the previous section, it will again turn out that the deterministic relation can be derived from the case of a probabilistic correlation. Therefore, we first consider the latter by starting from probability theory:

$$\begin{aligned} p_q(q) &= \int_{-\infty}^{+\infty} p_{q|xy}(q|x,y)p_{xy}(x,y)dxdy e^{-\beta F_q(q)} \\ e^{-\beta F_q(q)} &= \int_{-\infty}^{+\infty} p_{q|xy}(q|x,y)e^{-\beta F_{xy}(x,y)}dxdy F_q(q) \end{aligned}$$

Which gives the general projection formula as:

$$F_q(q) = -k_BT\ln\left(\int_{-\infty}^{+\infty} p_{q|xy}(q|x,y)e^{-\beta F_{xy}(x,y)}dxdy\right)$$

The situation for a deterministic relation encoded by the expression $Q(x, y)$ can be found by setting the conditional probability appropriately:

$$\begin{aligned} p_{q|xy}(q|x,y) &= \delta\left(q - Q(x,y)\right) \\ F_q(q) &= -k_BT\ln\left(\int_{-\infty}^{+\infty} \delta\left(q-Q(x,y)\right)e^{-\beta F_{xy}(x,y)}dxdy\right) \end{aligned}$$

Furthermore, this formula can be simplified for some special, but common, cases:

1. The difference between 2 collective variable $Q(x, y) = y - x$:

$$\begin{aligned} F_q(q) &= -k_BT\ln\left(\int_{-\infty}^{+\infty} \delta(y-x-q)e^{-\beta F_{xy}(x,y)}dxdy\right) \\ &= -k_BT\ln\left(\int_{-\infty}^{+\infty} e^{-\beta F_{xy}(x,x+q)}dx\right) \end{aligned}$$

2. The average of 2 collective variables $Q(x,y) = \frac{x+y}{2}$:

$$
\begin{aligned}
F_q(q) &= -k_B T \ln\left(\int_{-\infty}^{+\infty} \delta(\frac{x+y}{2} - q) e^{-\beta F_{xy}(x,y)} dx dy\right) \\
&= -k_B T \ln\left(2\int_{-\infty}^{+\infty} \delta(x+y-2q) e^{-\beta F_{xy}(x,y)} dx dy\right) \\
&= -k_B T \ln\left(2\int_{-\infty}^{+\infty} e^{-\beta F_{xy}(x,2q-x)} dx\right)
\end{aligned}
$$

6.1.2.3 Deprojecting free energy surfaces

Finally, one can also reconstruct a higher dimensional FES starting from a lower dimensional FES, a transformation we will herein denote as *deprojection*. As an example, consider a 1D FEP $F_q(q)$ in terms of the collective variable q and assume we are interested in reconstructing the 2D FES $F_{xy}(x,y)$ in terms of the collective variables x and y. Remark that the 2D FES generally encodes more information than the 1D FEP due to the higher number of degrees of freedom. Hence, the reconstruction of this 2D FES requires additional information. This additional information is encoded in the conditional probability $p_{xy|q}(x,y|q)$ expressing the probability of observing the system to have $X(\vec{r}^N) \in [x, x+dx]$ and $Y(\vec{r}^N) \in [y, y+dy]$ given that the system is in a macrostate given by $Q(\vec{r}^N) \in [q, q+dq]$. Using probability theory we find:

$$
\begin{aligned}
p_{xy}(x,y) &= \int_{-\infty}^{+\infty} p_{xyq}(x,y,q) dq \\
&= \int_{-\infty}^{+\infty} p_{xy|q}(x,y|q) p_q(q) dq
\end{aligned}
$$

Which we can translate to the deprojection formula for the free energy surface:

$$
F_{xy}(x,y) = -k_B T \ln\left(\int_{-\infty}^{+\infty} p_{xy|q}(x,y|q) e^{-\beta F_q(q)} dq\right)
$$

6.1.3 Conditional probabilities from (biased) simulations

In all of the transformation, projection and deprojection formulas considered in the previous sections, one either knows the relation between the various collective variables *a priori* (deterministic relation) or expressed such relation through a conditional probability. The latter conditional probability will have to be derived from molecular simulations. In that case, we want to extract the conditional probability from the molecular simulations that were already performed to construct the free energy prior to transformation, projection or deprojection. If additional simulations would be required, one could argue that we could just as well set up new simulations to directly extract the new free energy. In that case, one could question the added value of the above derived transformation formulas (and rightly so). However, in many cases the original simulations involved enhanced sampling simulations that introduce bias potentials (such as umbrella sampling). Such biasing potentials were not taking into account in the previous derivations and hence we should wonder whether the conditional probabilities can be extracted from these biases simulations. Luckily, this indeed turns out to be the case whenever the biasing potential remains constant during the simulations (as is the case in umbrella sampling). To show this, we only consider the conditional probability $p_{x|y}(x|y)$ figuring in Eq. (**??**), but the proof can easily be extended towards the other conditional probabilities. Crucial to the proof is the realization that the applied bias in the original simulations was expressed in terms of the original collective variables. In the case of $p_{x|y}(x|y)$ this means a bias $V^{(b)}(y)$ only in terms of y. We can then rewrite the conditional probability as follows:

$$\begin{aligned} p_{x|y}(x|y) &= \frac{p_{xy}(x,y)}{p_y(y)} = \frac{e^{-\beta F_{xy}(x,y)}}{e^{-\beta F_y(y)}} \\ &= \frac{e^{-\beta F_{xy}(x,y)}e^{-\beta V^{(b)}(y)}}{e^{-\beta F_y(y)}e^{-\beta V^{(b)}(y)}} \\ &= \frac{e^{-\beta\left[F_{xy}(x,y)+V^{(b)}(y)\right]}}{e^{-\beta\left[F_y(y)+V^{(b)}(y)\right]}} = \frac{p_{xy}^{(b)}(x,y)}{p_y^{(b)}(y)} \\ &= p_{x|y}^{(b)}(x|y) \end{aligned}$$

where $p_{x|y}^{(b)}(x|y)$ represents the conditional probability obtained from the biased simulation.

6.1.4 Kinetic reaction rates from transition state theory

Consider a system undergoing a process which brings it from one (meta)stable state (denoted as the reactant state) to another (meta)stable state (denoted as the product state). We also assume that the progress along the process can be adequately described by the collective variable $Q(\vec{r}^N)$ for which we have computed the free energy profile $F(q)$. Furthermore, we denote $q^\dagger$ as the value of Q in the transition state as defined in Eyrings transition state theory and assume $Q < q^\dagger$ corresponds to the reactant state, while $Q > q^\dagger$ corresponds to the product state. The reaction rate is a kinetic property as it represents the rate with which the system undergoes the process. Therefore, it cannot be derived merely from thermodynamic considerations. Eyrings transition state theory [allows us to (approximately) compute the reaction rate from output of molecular simulations in terms of the transition state as the ratio of two ensemble averages:

$$k^{TST} = \frac{\langle\dot{Q}\left(\vec{r}^N\right)\theta\left(\dot{Q}\left(\vec{r}^N\right)\right)\delta\left(Q\left(\vec{r}^N\right)-q^\dagger\right)\rangle}{\langle\theta\left(q^\dagger-Q(\vec{r}^N)\right)\rangle}$$

For a derivation of this formula, we refer to [. The meaning of the various terms in this formula is as follows:

$\dot{Q}\left(\vec{r}^N\right)$ the rate of variation (time derivative) of $Q(\vec{r}^N)$ $\theta\left(\dot{Q}\left(\vec{r}^N\right)\right)$ only include phase-space samples for which natural time evolution would direct the system towards positive Q, i.e. towards the product state. $\delta\left(Q\left(\vec{r}^N\right)-q^\dagger\right)$ only include phase-space samples in the transition state ($Q = q^\dagger$) $\theta\left(q^\dagger-Q(\vec{r}^N)\right)$ normalize by dividing by a factor relative to the probability of finding the system in the reactant state. ==
===

Since the momentum distribution is Gaussian, the integral over momentum space can be done analytically, which results in ():

$$k^{TST} = \sqrt{\frac{1}{2\pi\beta}}\frac{\int\delta\left(Q(\vec{x}^N)-q^\dagger\right)\left|\vec{\nabla}_x Q(\vec{x}^N)\right|e^{-\beta V(\vec{x}^N)}d\vec{x}^N}{\int_R e^{-\beta V(\vec{x}^N)}d\vec{x}^N}$$

in which $\vec{x}_i = \sqrt{m_i}\cdot\vec{r}_i$ represents mass-weighted coordinates. The integral in the denominator only runs over that part of phase space corresponding to the reactant state, i.e. $Q < q^\dagger$. Using the free energy profile $F(q)$ we can rewrite the expression even further:

$$\begin{aligned}k^{TST} &= \sqrt{\frac{1}{2\pi\beta}} \cdot \frac{\int \left|\vec{\nabla}_x Q(\vec{x}^N)\right| \delta\left(Q(\vec{x}^N) - q^\dagger\right) e^{-\beta V(\vec{x}^N)} d\vec{x}^N}{\int \delta\left(Q(\vec{x}^N) - q^\dagger\right) e^{-\beta V(\vec{x}^N)} d\vec{x}^N} \cdot \frac{\int \delta\left(Q(\vec{x}^N) - q^\dagger\right) e^{-\beta V(\vec{x}^N)} d\vec{x}^N}{\int_{-\infty}^{q^\dagger} \left[\int \delta\left(Q(\vec{x}^N) - q\right) e^{-\beta V(\vec{x}^N)} d\vec{x}^N\right] dq} \\ &= \sqrt{\frac{1}{2\pi\beta}} \cdot \left\langle \left|\vec{\nabla}_x Q\right| \right\rangle_{q^\dagger} \cdot \frac{e^{-\beta F(q^\dagger)}}{\int_{-\infty}^{q^\dagger} e^{-\beta F(q)} dq}\end{aligned}$$

Hence, transition state theory allows us to express the rate constant in terms of the free energy profile and an additional prefactor A which represents a (to the transition state) constrained ensemble average of the (mass-weighted) gradient of the collective variable.

$$\begin{aligned}k^{TST} &= A \cdot \frac{e^{-\beta F(q^\dagger)}}{\int_{-\infty}^{q^\dagger} e^{-\beta F(q)} dq} \\ A &= \sqrt{\frac{1}{2\pi\beta}} \cdot \left\langle \left|\vec{\nabla}_x Q\right| \right\rangle_{q^\dagger}\end{aligned}$$

6.1.5 Error on the sample average[sec:error_sample_average]

Whenever we want to calculate the population mean of a random variable, we can use the sample average as an adequate estimate:

$$\bar{X} = \frac{1}{N} \sum_{i=1}^{N} X_i$$

Such an estimate is a consistent estimate in the sense that due to the law of large numbers, its value converges to the true population average for samples sizes approaching infinity[1]:

$$\bar{X} \to E[X] \text{ as } N \to +\infty$$

Furthermore, as the sample average itself represents a random variable (since it is a function of random variables X_i), one can also express the mean and variance of this sample average. In this respect, we know that the sample average represents an unbiased estimator of the population mean, which means that the mean of the sample average is equal to the population mean itself:

$$E[\bar{X}] = E[X]$$

With respect to the variance of the sample average, this is the topic of the remainder of this section. We first make a distinction between computing the variance for uncorrelated samples (Section *[subsec:Uncorrelated-samples]*) and correlated samples (Section *[subsec:Correlated-samples]*). Finally, we discuss a well-known and practical method of dealing with these correlations, the so-called block-averaging method (Section *[subsec:Block-averaging]*).

[1] The limit used in the expression of the law of large numbers is not a conventional limit. Instead it should be interpreted as 'converges almost surely', which is a well-defined concept in probability theory.

6.1.5.1 Error bar for uncorrelated samples[subsec:Uncorrelated-samples]

In almost all cases, we are (or should be) also interested in an error bar on the estimate of the population mean. Such error bar is given by the variance of the sample average. However, we need to make a distinction between uncorrelated and correlated samples. If all samples are uncorrelated from all others, then we have maximum amount of information in the N-sample and the error bar will be at its lowest. Mathematically, this is expressed as follows:

$$\begin{aligned}\text{Var}[\bar{X}] &= \text{Var}[\frac{1}{N}\sum_{i=1}^{N} X_i] = \frac{1}{N^2}\text{Var}[\sum_{i=1}^{N} X_i]\\ &= \frac{1}{N^2}\sum_{i=1}^{N}\text{Var}[X_i]\\ &= \frac{1}{N^2}\sum_{i=1}^{N}\sigma^2 = \frac{1}{N^2}N\sigma^2\\ &= \frac{\sigma^2}{N}\end{aligned}$$

Herein, the second line is only valid due to the fact that we assume the samples are uncorrelated. In the third line, we used the fact that all samples X_i are identically distributed with variance σ^2 of the original population. As a result we see that the larger the sample size, the smaller the variance. Since the error bar is proportional to the square root of the variance (i.e. the standard deviation), we find the well-known result that the error on the sample average of uncorrelated samples decreases with increasing sample size N as $\frac{1}{\sqrt{N}}$.

6.1.5.2 Error bar for correlated samples[subsec:Correlated-samples]

However, whenever two samples are correlated they do not provide independent information and hence using the sample average as given above will overestimate the amount of information in our sample. This in turn will turn out to underestimate the error bar and hence we need to correct for this. To illustrate this mathematically, we rewrite the variance on the sample average:

$$\begin{aligned}\text{Var}[\bar{X}] &= E\left[\left(\bar{X}-E[\bar{X}]\right)^2\right] = E\left[\left(\frac{1}{N}\sum_{i=1}^{N}X_i-\mu\right)^2\right]\\ &= E\left[\left(\frac{1}{N}\sum_{i=1}^{N}X_i\right)^2 - 2\mu\frac{1}{N}\sum_{i=1}^{N}X_i+\mu^2\right]\\ &= E\left[\left(\frac{1}{N}\sum_{i=1}^{N}X_i\right)^2\right]-\mu^2 = \frac{1}{N^2}E\left[\sum_{i,j=1}^{N}X_iX_j\right]-\mu^2\\ &= \frac{1}{N^2}\sum_{i,j=1}^{N}E[X_iX_j]-\mu^2\\ &= \frac{1}{N^2}\left(\underbrace{\sum_{i=1}^{N}E[X_i^2]}_{i=j}+\underbrace{2\sum_{i<j=1}^{N}E[X_iX_j]}_{i\neq j}\right)-\mu^2\end{aligned}$$

The first term in the last line is related to the variance σ^2 of the population:

$$\sum_{i=1}^{N} E[X_i^2] = N(\sigma^2 + \mu^2)$$

Which results in:

$$\text{Var}[\bar{X}] = \frac{\sigma^2}{N} + \frac{2}{N^2} \sum_{i<j=1}^{N} E[X_i X_j] - \frac{N-1}{N}\mu^2$$

The second term can be rewritten in terms of the autocorrelation function as we will now show. We start by using the time translation invariance of the sample data. For $i < j$ this can be expressed as follows:

$$E[X_i X_j] = E[X_1 X_{j-i+1}] = E[X_1 X_{t+1}]$$

for $t = j - i$ with values in $[0, N-1]$. Using this relabeling, the second term in Eq. *[eq:variance_correlated_intermed1]* can be rewritten as follows:

$$\begin{aligned}
\sum_{i<j=1}^{N} E[X_i X_j] &= \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} E[X_i X_j] \\
&= \sum_{i=1}^{N-1} \sum_{t=1}^{N-i} E[X_1 X_{1+t}] \\
&= \sum_{t=1}^{N-1} E[X_1 X_{1+t}] + \sum_{t=1}^{N-2} E[X_1 X_{1+t}] + \cdots + \sum_{t=1}^{2} E[X_1 X_{1+t}] + \sum_{t=1}^{1} E[X_1 X_{1+t}] \\
&= (N-1)E[X_1 X_2] + \cdots + (N-t)E[X_1 X_{1+t}] + E[X_1 X_N] \\
&= \sum_{t=1}^{N-1} (N-t) E[X_1 X_{1+t}]
\end{aligned}$$

Furthermore, we also rewrite the third term in Eq. *[eq:variance_correlated_intermed1]*:

$$\frac{N-1}{N}\mu^2 = \frac{2}{N^2} \cdot \frac{N(N-1)}{2}\mu^2 = \frac{2}{N^2} \sum_{t=1}^{N-1} (N-t)\mu^2$$

As such we can further rewrite Eq. *[eq:variance_correlated_intermed1]*:

$$\begin{aligned}
\text{Var}[\bar{X}] &= \frac{\sigma^2}{N} + \frac{2}{N^2} \sum_{t=1}^{N-1} (N-t) E[X_1 X_{1+t}] - \frac{2}{N^2} \sum_{t=1}^{N-1} (N-t)\mu^2 \\
&= \frac{\sigma^2}{N} + \frac{2}{N^2} \sum_{t=1}^{N-1} (N-t) \left(E[X_1 X_{1+t}] - \mu^2 \right) \\
&= \frac{\sigma^2}{N} + \frac{2\sigma^2}{N^2} \sum_{t=1}^{N-1} (N-t) \frac{E\left[(X_1 - \mu)(X_{1+t} - \mu)\right]}{\sigma^2} \\
&= \frac{\sigma^2}{N} \left[1 + 2 \sum_{t=1}^{N-1} \left(1 - \frac{t}{N} \right) c(t) \right]
\end{aligned}$$

where we introduced the autocorrelation function:

$$c(t) = \frac{E\left[(X_1 - \mu)(X_{1+t} - \mu)\right]}{\sigma^2}$$

Finally, we define the integrated correlation time τ_{int} which allows to express the variance on the sample average more easily:

$$\begin{aligned}\text{Var}[\bar{X}] &= \frac{\sigma^2}{N}\tau_{int} \\ \tau_{int} &= 1 + 2\sum_{t=1}^{N-1}\left(1-\frac{t}{N}\right)c(t)\end{aligned}$$

As such, we can interpret the integrated correlation time τ_{int} as the number of subsequent samples in the original sample set that are correlated and hence should be treated as only 1 sample. As a result the number of independent samples is given by $\frac{N}{\tau_{int}}$ instead of N when computing the variance on the sample average. In many cases, one can assume that the autocorrelation function can be expressed as a simple exponential:

$$c(t) = e^{-\frac{t}{\tau_{exp}}}$$

The integrated correlation time for N-sample with $N \to +\infty$ (hence approaching total population) can easily be computed:

$$\begin{aligned}\tau_{int} &= 1 + 2\sum_{t=1}^{N}\left(1-\frac{t}{N}\right)e^{-\frac{t}{\tau_{exp}}} \\ &= 1 + 2\sum_{t=1}^{+\infty} e^{-\frac{t}{\tau_{exp}}} \\ &= 1 + 2\left(\sum_{t=0}^{+\infty} e^{-\frac{t}{\tau_{exp}}} - 1\right) \\ &= 1 + 2\left(\frac{1}{e^{-1/\tau_{exp}} - 1} - 1\right) \\ &= 1 + 2\frac{e^{-1/\tau_{exp}}}{e^{-1/\tau_{exp}} - 1}\end{aligned}$$

In the second line, we used the facts that (1) the function $1 - \frac{t}{N} \approx 1$ for $t < N(\to +\infty)$ and (2) the function $c(t) = e^{-\frac{t}{\tau_{exp}}} \approx 0$ for $t \to +\infty$. In the fourth line, we also assumed $\tau_{exp} > 0$. Furthermore, for large values of the 'exponential correlation time' τ_{exp} we can approximate the function

$$\frac{e^{-1/\tau_{exp}}}{e^{-1/\tau_{exp}} - 1} \approx \tau_{exp} - \frac{1}{2}$$

In practice, the approximation is valid rather quickly (for $\tau_{exp} > 8.3$ the error on the approximation is smaller than 0.01). The approximation gives rise to the following integrated correlation time

$$\tau_{int} \approx 2\tau_{exp} \text{ for } \tau_{exp} \gg 1$$

revealing the relation between the integrated correlation time (i.e. the number of subsequent samples that are correlated and hence need to be treated as one sample) and the exponential correlation time (i.e. the rate with which correlations in the samples die out).

6.1.5.3 Block averaging[subsec:Block-averaging]

As was discussed in the previous section, to estimate the variance on the sample average, we need to account for sample correlations. Assume we start from a correlated N-sample taken from a population with mean μ and variance σ^2, then we know from previous subsections that the sample average has mean and variance given by:

$$\begin{aligned}\bar{X} &= \frac{1}{N}\sum_{i=1}^{N} X_i \\ E[\bar{X}] &= \mu \\ \text{Var}[\bar{X}] &= \frac{\sigma^2}{N}\tau_{int}\end{aligned}$$

To estimate the variance, one could compute the integrated correlation time τ_{int}. Although this can be achieved by defining an estimator for the autocorrelation function, we will here discuss an alternative method to deal with the sample correlations, the so called block-averaging method. We first divide the entire sample in blocks of size B (resulting in a number of blocks equal to $N_B = \frac{N}{B}$) and compute the average of each block, which we denote as the block averages X_b:

$$\{X_i\}_{i=1}^{N} \rightarrow \{X_b\}_{b=1}^{N_B} \text{ with } X_b = \frac{1}{B}\sum_{i=1}^{B} X_{(b-1)B+i}$$

The average of the block averages ($\bar{X}_B$) is identical to the orginal full sample average ($\bar{X}$):

$$\bar{X}_B = \frac{1}{N_B}\sum_{b=1}^{N_B} X_b = \frac{1}{N_B}\sum_{b=1}^{N_B}\frac{1}{B}\sum_{i=1}^{B} X_{(b-1)B+i} = \frac{1}{N_B B}\sum_{n=1}^{N} X_n = \bar{X}$$

Hence, the average of the block averages is a good (consistent and unbiased) estimator for the population mean. However, the variance of the block averages is given by:

$$\text{Var}[\bar{X}_B] = \underbrace{\frac{\sigma_B^2}{N_B}}_{\text{Var}[\bar{X}_B]_{naive}} \cdot \tau_{int,B}$$

Herein, we indicated what would be the naive estimate of the variance when the correlations between block averages would not be accounted for. The factor $\tau_{int,B}$ corrects for these correlations between subsequent block averages. In case we choose $B = 1$ (corresponding to the original sample series without block averaging), we find $\tau_{int,B}(B = 1) = \tau_{int}$ corresponding to the integrated correlation time in the original sample seris. Furthermore, $\sigma_B^2 = \text{Var}[X_b]$ represents the variance of the block averages X_b (i.e. not the variance of the average of the block averages but of the block averages themselves), for which we have an unbiased estimator:

$$S_B^2 = \frac{1}{N_B - 1}\sum_{b=1}^{N_B}\left(X_b - \bar{X}_B\right)^2$$

If the original sample would be free of correlations, the block average integrated correlation time $\tau_{int,B}$ would be 1 independent off the block size B and the variance on the block averages would be given by:

$$\sigma_B^2 = \frac{\sigma^2}{B}$$

As a result, the variance on the average of block averages given by Eq. *[eq:blav_var_avofblavs]* would reduce to

$$\text{Var}[\bar{X}_B] = \frac{\sigma_B^2}{N_B}\tau_{int,B} \xrightarrow{uncorrelated} \text{Var}[\bar{X}_B]_{naive} = \frac{\sigma_B^2}{N_B} = \frac{\sigma^2}{BN_B} = \frac{\sigma^2}{N}$$

which is indeed consistent with Eq. *[eq:blav_sampleaverage_variance]* for $\tau_{int} = 1$ for uncorrelated samples. This implies that in case sample correlations are absent, the naive error estimate $\text{Var}[\bar{X}_B]_{naive}$ is independent of the block size and equals the true error on the sample average. This is, however, no longer the case when correlations are present in the original samples. In this case, we still consider the naive estimate of the variance $\text{Var}[\bar{X}_B]_{naive}$, which now depends on the block size B:

$$\text{Var}[\bar{X}_B]_{naive} = \frac{\sigma_B^2}{N_B} = \frac{\text{Var}[\bar{X}_B]}{\tau_{int,B}} = \frac{\text{Var}[\bar{X}]}{\tau_{int,B}(B)} = \frac{\text{True error}}{\tau_{int,B}(B)}$$

where we used the fact that (1) $\bar{X}_B = \bar{X}$ and hence so is its variance which is the true error we are interested in and (2) the integrated correlation time between block averages $\tau_{int,B}$ depends on the block size B. Furthermore, as the block size increases, the 'time-difference' between subsequent block averages increases and hence the correlation between them will decrease as well. Hence, as the block size increases, $\tau_{int,B}$ will decrease and the naive estimate $\text{Var}[\bar{X}_B]_{naive}$ will increase. Once the block size becomes larger than the integrated correlation time τ_{int}, $\tau_{int,B}$ will have converged to 1 and the naive error estimate $\text{Var}[\bar{X}_B]_{naive}$ will have converged to the true error. In order to apply the block averaging method to estimate the true (correlated) error, we execute the following steps and for each block size execute the following steps:

1. Define a range of block sizes ($B = 1 \rightarrow B_{max}$)
2. For each block size B do:
   1. compute the block averages $\{X_b\}_{b=1}^{N_B}$

      $$X_b = \frac{1}{B}\sum_{i=1}^{B} X_{(b-1)B+i}$$

   2. Estimate the variance of the block averages:

      $$S_B^2 = \frac{1}{N_B - 1}\sum_{b=1}^{N_B}\left(X_b - \bar{X}_B\right)^2$$

   3. Compute the naive estimate of the variance on the sample average:

      $$\text{Var}[\bar{X}_B]_{naive} = \frac{S_B^2}{N_B}$$

3. Propose a model function for $\tau_{int,B}(B)$. This function should satisfy following two conditions:

   $$\tau_{int,B}(B = 1) = \tau_{int}$$
   $$\lim_{B\rightarrow+\infty} \tau_{int,B}(B) = 1$$

   A possible candidate is:

   $$\tau_{int,B}(B) = 1 + \frac{\tau_{int} - 1}{B}$$

4. Use the model function to fit the naive error estimation as function of the blocksize:

   $$\text{Var}[\bar{X}_B]_{naive} = f(B) = \frac{TE}{\tau_{int,B}(B)}$$

   in which TE represents the true error we are interested in. Using the previously mentioned example for model function of $\tau_{int,B}(B)$ we find:

   $$f(B) = \frac{TE \cdot B}{B + \tau_{int} - 1}$$

   The resulting parameters of the fit, TE and τ_{int}, gives the required estimate for the true error on the sample average and the integrated correlation time respectively.

6.1.6 Error on the estimation of the probability density

In this section we outline the mathematical details on how to derive a probability density (as well as the corresponding free energy profile) as function of a collective variable from a histogram including an estimation of the corresponding error bar. The discussion is divided into two parts, the first part deals with the construction of a probability density from a single (possibly biased) equilibrium simulation while the second part deals with the wheighted histogram analysis method for reconstructing probability densities from multiple (possibly biased) equilibrium simulations as applied in the umbrella sampling technique.

6.1.6.1 Estimation from a single equilibrium simulation

Probability model[subsec:Probability-model]

Assume we performed a molecular dynamics simulation and computed the value of the collective variable $Q(\vec{r}^N)$ at each step resulting in the sample series q_i. From this series, one can easily construct a histogram H_k representing the number of q-values in the series that are within the bin $[q_k - \frac{\Delta}{2}, q_k + \frac{\Delta}{2}]$ for a given bin width Δ. After proper normalization, this histogram can serve as a model for the probability density. However, in order to estimate the error on each value in the resulting probability distribution, we apply the theory of maximum likelihood estimators and with corresponding error estimate from the Fisher information. Herein, we start from a model for the probability density of finding a value q. Such model will depend on certain model parameters which we will condense in the parameter vector $\vec{a}$ and we will therefore denote it as $p(q|\vec{a})$ indicating the probability is conditional on a certain value of the parameter vector $\vec{a}$. For example, a model for the probability density corresponding to a histogram can be expressed as

$$p(q|\vec{a}) = \sum_{k=1}^{K} a_k \phi_k(q)$$

Herein, ϕ_k represents a function corresponding to the bin centered around value Q_k:

$$\phi_k(q) = \begin{cases} \frac{1}{\Delta} & q \in [Q_k - \frac{\Delta}{2}, Q_k + \frac{\Delta}{2}] \\ 0 & elsewhere \end{cases}$$

with Δ an a priori chosen bin width and Q_k the a priori chosen bin centers. Note that each bin function ϕ_k is normalized to 1, which implies that normalization of the probability density requires:

$$\sum_{k=1}^{K} a_k = 1$$

Likelihood and maximum likelihood estimator[subsec:Likelihood-and-MLE]

In order to be able to use such a model for further analysis such as the construction of the free energy profile, we need an accurate estimate for the model parameters $\vec{a}$, which we want to derive from the simulation data given by the samples q_i. These parameters can be estimated using the likelihood function. We define the likelihood $\mathcal{L}(\vec{a})$ as the probability of $\vec{a}$ being the true parameters given that we observed samples $\vec{q} = (q_1, \ldots, q_N)$ that were taken from the true probability distribution[2]. Using Bayes theorem for conditional probabilities we find:

$$\mathcal{L}(\vec{a}) = p(\vec{a}|\vec{q}) = \frac{p(\vec{a}, \vec{q})}{p(\vec{q})} = \frac{p(\vec{q}|\vec{a})p(\vec{a})}{p(\vec{q})}$$

[2] We assume uncorrelated samples. See section *[sec:error_sample_average]* for more information on error estimation of sample averages with sample correlations.

Herein, we can express $p(\vec{q}|\vec{a})$ in terms of the previously defined probabilty function $p(q|\vec{a})$ (assuming uncorrelated samples):

$$p(\vec{q}|\vec{a}) = \prod_{n=1}^{N} p(q_n|\vec{a})$$

Furthermore, the probability $p(\vec{a})$ represents a prior knowledge about the parameters, which we assume we don't have, i.e. $p(\vec{a}) = cte$. Finally, since $p(\vec{q})$ does not depend on the parameters, it is only relevant for proper normalization. However, such normalisation will not be important for our purposes, hence we write:

$$\mathcal{L}(\vec{a}) = p(\vec{q}|\vec{a}) = \prod_{n=1}^{N} p(q_n|\vec{a})$$

We also define the log-likelihood and the related function l_N:

$$\log \mathcal{L} = \sum_{n=1}^{N} \log\left(p(q_n|\vec{a})\right)$$

$$l_N(\vec{a}) = \frac{1}{N}\sum_{n=1}^{N} \log\left(p(q_n|\vec{a})\right)$$

The function l_N can be interpreted as an N-sample estimate of the mean of the random variable $\log\left(p(q|\vec{a})\right)$, therefore we also define the true mean l:

$$l(\vec{a}) = E_{\vec{a}_0}\left[\log\left(p(q|\vec{a})\right)\right] = \int_{-\infty}^{+\infty} \log\left[p(q|\vec{a})\right] p(q|\vec{a}_0) dq$$

where the notation $E_{\vec{a}_0}[\cdot]$ denotes the average computed using the probability density $p(q|\vec{a}_0)$ with the true parameters $\vec{a}_0$. We are now ready to define the so-called maximum likelihood estimator (MLE) $\hat{\vec{a}}$ as an estimator for the parameters which maximizes the likelihood $\mathcal{L}(\vec{a})$ (and as a result $\log \mathcal{L}(\vec{a})$ and $l_N(\vec{a})$ as well):

$$\frac{\partial l_N}{\partial \vec{a}}(\hat{\vec{a}}) = 0$$

Consistency of the MLE

We continue by formulating and prooving the following theorem:

The true parameters $\vec{a}_0$ maximize the funtion $l(\vec{a})$

$$l(\vec{a}) \leq l(\vec{a}_0) \;\; \forall \vec{a} \text{ and } l(\vec{a}) = l(\vec{a}_0) \text{ only if } \vec{a} = \vec{a}_0$$

To prove this, we rewrite the difference $l(\vec{a}) - l(\vec{a}_0)$

$$\begin{aligned}
l(\vec{a}) - l(\vec{a}_0) &= \int_{-\infty}^{+\infty} \log\left[p(q|\vec{a})\right] p(q|\vec{a}_0) dq - \int_{-\infty}^{+\infty} \log\left[p(q|\vec{a}_0)\right] p(q|\vec{a}_0) dq \\
&= \int_{-\infty}^{+\infty} \log\left[\frac{p(q|\vec{a})}{p(q|\vec{a}_0)}\right] p(q|\vec{a}_0) dq \\
&\leq \int_{-\infty}^{+\infty} \left[\frac{p(q|\vec{a})}{p(q|\vec{a}_0)} - 1\right] p(q|\vec{a}_0) dq \\
&= \int_{-\infty}^{+\infty} \left[p(q|\vec{a}) - p(q|\vec{a}_0)\right] dq = 0
\end{aligned}$$

In the third line we used the inequality $\log x \leq x - 1$ valid $\forall x > 0$. Furthermore, since the probability density $p(q|\vec{a}_0)$ is postive everywhere, the equality can only hold if $\log\left[\frac{p(q|\vec{a})}{p(q|\vec{a}_0)}\right]$ is zero everywhere, which implies that $p(q|\vec{a}) = p(q|\vec{a}_0)$ everywhere and hence $\vec{a} = \vec{a}_0$.

$$Q.E.D.$$

As such, we know now that the true parameters $\vec{a}_0$ maximize the function $l(\vec{a})$ while the maximum likelihood estimator $\hat{\vec{a}}$ maximizes the corresponding estimator function $l_N(\vec{a})$. Since $l(\vec{a})$ represents the mean of a random variable, while $l_N(\vec{a})$ represents the sample average of this random variable, we can apply the law of large numbers (LLN). Hence, the estimator function $l_N(\vec{a})$ will approach the true mean $l(\vec{a})$ in the limit for sample sizes approaching infinity. As a result we can conclude the same thing about the maxima of these functions, i.e. the MLE will approach the true parameters in the same limit:

$$l_N(\vec{a}) \to l(\vec{a}) \text{ as } N \to +\infty$$
$$\hat{\vec{a}} \to \vec{a}_0 \text{ as } N \to +\infty$$

The result of the last line is referred to as *consistency* of the MLE, which is not the same as being unbiased, but is important for its interpretation as an adequate estimator of the true parameters.

Fisher information

In order to derive an estimate for the error on the MLE, we first need to introduce the Fisher information.

The Fisher information is defined as the following matrix:

$$\bar{\bar{I}}(\vec{a}_0) = E_{\vec{a}_0}\left[\left(\frac{\partial}{\partial\vec{a}}\log p(q|\vec{a}_0)\right)\left(\frac{\partial}{\partial\vec{a}}\log p(q|\vec{a}_0)\right)^T\right]$$

with matrix elements:

$$I_{kl}(\vec{a}_0) = E_{\vec{a}_0}\left[\left(\frac{\partial}{\partial a_k}\log p(q|\vec{a}_0)\right)\left(\frac{\partial}{\partial a_l}\log p(q|\vec{a}_0)\right)\right]$$

Herein, the notation $\frac{\partial}{\partial a_k}\log p(q|\vec{a}_0)$ indicates we first take the derivatieve of $\log p(q|\vec{a})$ with respect to a_k and then evaluate at $\vec{a} = \vec{a}_0$. We proceed by formulating and proving an important theorem about the Fisher information.

The Fisher information can be computed from the expectation value of the second order derivative of $\log p(q|\vec{a})$ as follows:

$$\bar{\bar{I}}(\vec{a}_0) = -E_{\vec{a}_0}\left[\frac{\partial^2}{\partial\vec{a}^2}\log p(q|\vec{a}_0)\right]$$
$$I_{kl}(\vec{a}_0) = -E_{\vec{a}_0}\left[\frac{\partial^2}{\partial a_k\partial a_l}\log p(q|\vec{a}_0)\right]$$

We start by rewriting the second order derivative

$$\begin{aligned}\frac{\partial^2}{\partial a_k\partial a_l}\left[\log p(q|\vec{a})\right] &= \frac{\partial}{\partial a_l}\left[\frac{1}{p(q|\vec{a})}\frac{\partial p(q|\vec{a})}{\partial a_k}\right] = \frac{1}{p(q|\vec{a})}\frac{\partial^2 p(q|\vec{a})}{\partial a_k\partial a_l} - \frac{1}{(p(q|\vec{a}))^2}\frac{\partial p(q|\vec{a})}{\partial a_k}\frac{\partial p(q|\vec{a})}{\partial a_l}\\ &= \frac{1}{p(q|\vec{a})}\frac{\partial^2 p(q|\vec{a})}{\partial a_k\partial a_l} - \frac{\partial\log p(q|\vec{a})}{\partial a_k}\frac{\partial\log p(q|\vec{a})}{\partial a_l}\end{aligned}$$

Next, we can compute the expected value

$$\begin{aligned}
E_{\vec{a}_0}\left[\frac{\partial^2}{\partial a_k \partial a_l}\log p(q|\vec{a}_0)\right] &= \int_{-\infty}^{+\infty}\frac{\partial^2}{\partial a_k \partial a_l}\left[\log p(q|\vec{a}_0)\right]p(q|\vec{a}_0)dq \\
&= \int_{-\infty}^{+\infty}\left[\frac{1}{p(q|\vec{a}_0)}\frac{\partial^2 p(q|\vec{a}_0)}{\partial a_k \partial a_l} - \frac{\partial \log p(q|\vec{a}_0)}{\partial a_k}\frac{\partial \log p(q|\vec{a}_0)}{\partial a_l}\right]p(q|\vec{a}_0)dq \\
&= \int_{-\infty}^{+\infty}\frac{1}{p(q|\vec{a}_0)}\frac{\partial^2 p(q|\vec{a}_0)}{\partial a_k \partial a_l}p(q|\vec{a}_0)dq - \int_{-\infty}^{+\infty}\frac{\partial \log p(q|\vec{a}_0)}{\partial a_k}\frac{\partial \log p(q|\vec{a}_0)}{\partial a_l}p(q|\vec{a}_0)dq \\
&= \frac{\partial^2}{\partial a_k \partial a_l}\left[\underbrace{\int_{-\infty}^{+\infty}p(q|\vec{a}_0)dq}_{=1}\right] - E_{\vec{a}_0}\left[\left(\frac{\partial}{\partial a_k}\log p(q|\vec{a}_0)\right)\left(\frac{\partial}{\partial a_l}\log p(q|\vec{a}_0)\right)\right] \\
&= -I_{kl}(\vec{a}_0)
\end{aligned}$$

$$Q.E.D$$

As a result, we now know the following about the function $l(\vec{a})$:

$$\begin{aligned}
l(\vec{a}) &= E_{\vec{a}_0}\left[\log\left(p(q|\vec{a})\right)\right] \\
\frac{\partial l}{\partial \vec{a}}(\vec{a}_0) &= E_{\vec{a}_0}\left[\frac{\partial}{\partial \vec{a}}\log\left(p(q|\vec{a}_0)\right)\right] = 0 \text{ (as } \vec{a}_0 \text{ represents the maximum of } l) \\
\frac{\partial^2 l}{\partial \vec{a}\partial \vec{a}}(\vec{a}_0) &= E_{\vec{a}_0}\left[\frac{\partial^2}{\partial a_k \partial a_l}\log p(q|\vec{a}_0)\right] = -\bar{\bar{I}}(\vec{a}_0)
\end{aligned}$$

These relations represent properties of the population mean of $\log\left(p(q|\vec{a})\right)$ which we will now use to say something about the properties of the sample average l_N and the distribution of its maximum, i.e. the MLE.

Asymptotic normality of the MLE[subsec:Asymptotic-normality]

Asymptotic normality expresses that when the sample size N approaches infinity, the distribution of the MLE will approach a multivariate normal distribution with a mean given by the true parameters $\vec{a}_0$ and a covariance matrix related to the inverseof the Fisher information. Mathematically, this is expressed in the following theorem.

The MLE converges in probability towards a normal distribution centered around the true parameters with (co)variance related to the Fisher information

$$\sqrt{N}\left(\hat{\vec{a}} - \vec{a}_0\right) \xrightarrow{distribution} \mathcal{N}(\vec{0}, \left[\bar{\bar{I}}(\vec{a}_0)\right]^{-1})$$

or equivalently

$$\hat{\vec{a}} \xrightarrow{distribution} \mathcal{N}\left(\vec{a}_0, \frac{\left[\bar{\bar{I}}(\vec{a}_0)\right]^{-1}}{N}\right)$$

where the second argument of the normal distribution denotes the covariance matrix (which is why $\sqrt{N}$ gets squared).

The proof starts with an application of the mean value theorem. We consider the function

$$f_k(t) = \frac{\partial}{\partial a_k}l_N(\hat{\vec{a}} \cdot t + \vec{a}_0 \cdot (1-t))$$

and apply the mean value theorem to it:

$$\exists c_k \in [0,1] : f_k'(c_k) = \frac{f_k(1) - f_k(0)}{1-0}$$

We first define

$$\vec{a}_1(t) = \hat{\vec{a}} \cdot t + \vec{a}_0 \cdot (1-t)$$

and evaluate the left hand side:

$$\begin{aligned} f_k'(c_k) &= \frac{\partial}{\partial \vec{a}} \left[\frac{\partial}{\partial a_k} l_N(\vec{a}) \right]_{\vec{a}=\vec{a}_1(c_k)} \cdot \frac{d}{dt} \left(\vec{a}_1(t) \right)_{t=c_k} \\ &= \frac{\partial}{\partial \vec{a}} \left[\frac{\partial}{\partial a_k} l_N(\vec{a}_1(c_k)) \right] \cdot \left(\hat{\vec{a}} - \vec{a}_0 \right) \end{aligned}$$

The right-hand side becomes:

$$f_k(1) - f_k(0) = \frac{\partial}{\partial a_k} l_N(\hat{\vec{a}}) - \frac{\partial}{\partial a_k} l_N(\vec{a}_0)$$

We know consider the limit $N \to +\infty$. In this limit, $\hat{\vec{a}} \to \vec{a}_0$ which implies that $\vec{a}_1(c_k) \to \vec{a}_0$ for each value of k. Hence, we find that in this limit

$$\frac{\partial}{\partial \vec{a}} \left[\frac{\partial}{\partial a_k} l_N(\vec{a}_0) \right] \cdot \left(\hat{\vec{a}} - \vec{a}_0 \right) = \frac{\partial}{\partial a_k} l_N(\hat{\vec{a}}) - \frac{\partial}{\partial a_k} l_N(\vec{a}_0)$$

Since this is valid for each value of k and further more $\frac{\partial}{\partial a_k} l_N(\hat{\vec{a}}) = 0$ per definition of the MLE, we find for $N \to +\infty$:

$$\frac{\partial^2}{\partial \vec{a} \partial \vec{a}} l_N(\vec{a}_0) \cdot \left(\hat{\vec{a}} - \vec{a}_0 \right) = -\frac{\partial}{\partial \vec{a}} l_N(\vec{a}_0)$$

Or equivalently

$$\begin{aligned} \bar{\bar{I}}(\vec{a}_0) \cdot \left(\hat{\vec{a}} - \vec{a}_0 \right) &= \frac{\partial}{\partial \vec{a}} l_N(\vec{a}_0) \\ \sqrt{N} \left(\hat{\vec{a}} - \vec{a}_0 \right) &= \sqrt{N} \left[\bar{\bar{I}}(\vec{a}_0) \right]^{-1} \cdot \frac{\partial}{\partial \vec{a}} l_N(\vec{a}_0) \end{aligned}$$

We now focus on the distribution of $\frac{\partial}{\partial a_k} l_N(\vec{a}_0)$ in the limit $N \to +\infty$:

$$\begin{aligned} \frac{\partial}{\partial a_k} l_N(\vec{a}_0) &= \frac{\partial}{\partial a_k} \left[\frac{1}{N} \sum_{n=1}^N \log p(q_n|\vec{a}) \right]_{\vec{a}=\vec{a}_0} \\ &= \frac{1}{N} \sum_{n=1}^N \frac{\partial}{\partial a_k} \log p(q_n|\vec{a}_0) \end{aligned}$$

Since we already know that $E_{\vec{a}_0} \left[\frac{\partial}{\partial \vec{a}} \log \left(p(q|\vec{a}_0) \right) \right] = 0$ we can rewrite this as:

$$\frac{\partial}{\partial a_k} l_N(\vec{a}_0) = \bar{X} - E\left[X \right]$$

with

$$\bar{X} = \frac{1}{N} \sum_{n=1}^N X_n$$

the sample average of the random variable $X = \frac{\partial}{\partial \vec{a}} \log\left(p(q|\vec{a}_0)\right)$ with corresponding samples $X_n = \frac{\partial}{\partial \vec{a}} \log\left(p(q_n|\vec{a}_0)\right)$. As a result, the central limit theorem can be applied to Eq *[eq:asnorm_X]*, which tells us:

$$\sqrt{N} \cdot \frac{\partial}{\partial a_k} l_N(\vec{a}_0) \xrightarrow{distribution} \mathcal{N}\left(0, \bar{\bar{\sigma}}\right)$$

in which $\bar{\bar{\sigma}}$ represents the covariance matrix of the normal distribution, which we will now determine.

$$\begin{aligned}
\sigma_{kl} &= E_{\vec{a}_0}\left[\left(\frac{\partial}{\partial a_k}\log\left(p(q|\vec{a}_0)\right) - \underbrace{E_{\vec{a}_0}\left[\frac{\partial}{\partial a_k}\log p(q|\vec{a}_0)\right]}_{=0}\right)\left(\frac{\partial}{\partial a_l}\log\left(p(q|\vec{a}_0)\right) - \underbrace{E_{\vec{a}_0}\left[\frac{\partial}{\partial a_l}\log p(q|\vec{a}_0)\right]}_{=0}\right)\right] \\
&= E_{\vec{a}_0}\left[\frac{\partial}{\partial a_k}\log\left(p(q|\vec{a}_0)\right)\frac{\partial}{\partial a_l}\log\left(p(q|\vec{a}_0)\right)\right] \\
&= I_{kl}(\vec{a}_0)
\end{aligned}$$

which result in:

$$\sqrt{N} \cdot \frac{\partial}{\partial a_k} l_N(\vec{a}_0) \xrightarrow{distribution} \mathcal{N}\left(0, \bar{\bar{I}}(\vec{a}_0)\right)$$

Combining Eq. *[eq:asnorm_rel1]* with Eq. *[eq:asnorm_rel2]* we find:

$$\sqrt{N}\left(\hat{\vec{a}} - \vec{a}_0\right) \xrightarrow{distribution} \left[\bar{\bar{I}}(\vec{a}_0)\right]^{-1} \cdot \mathcal{N}\left(0, \bar{\bar{I}}(\vec{a}_0)\right) = \mathcal{N}\left(0, \left[\bar{\bar{I}}(\vec{a}_0)\right]^{-2} \bar{\bar{I}}(\vec{a}_0)\right) = \mathcal{N}\left(0, \left[\bar{\bar{I}}(\vec{a}_0)\right]^{-1}\right)$$

$$Q.E.D.$$

As a final remark, we note that usually, one does not know the true parameters $\vec{a}_0$ and hence one also does not know the Fisher matrix. Therefore, in order to estimate the (co)variance of the MLE estimator, we can use an important property of the MLE estimator that expresses that for any function $f(\vec{a})$, the estimator $f(\hat{\vec{a}})$ is a good estimator for $f(\vec{a}_0)$. Hence, we can estimate the Fisher matrix as $\bar{\bar{I}}(\hat{\vec{a}})$.

Example 1 - Bernoulli experiment

Consider performing an experiment[3] who's outcome can be either succes, with probability $a > 0$, or failure with probability $1 - a > 0$. If we define a random variable Q with outcome $q = 1$ for success and $q = 0$ for failure, we can write down the probability distribution such an experiment as:

$$\begin{aligned}
p(q|a) &= a^q(1-a)^{1-q} \\
\log p(q|a) &= q\log(a) + (1-q)\log(1-a)
\end{aligned}$$

which gives rise to the population average:

$$E_{a_0}[q] = a_0$$

We can also define $l(a)$:

$$\begin{aligned}
l(a) &= E_{a_0}[\log p(q|a)] \\
&= E_{a_0}[q] \cdot \log(a) + (1 - E_{a_0}[q])\log(1-a) \\
&= a_0 \cdot \log(a) + (1-a_0)\log(1-a)
\end{aligned}$$

[3] An experiment has to be interpreted in the broad sense here, i.e. it can also be a computational experiment or thought experiment.

Unfortunately, we cannot not evaluate this function, as we do not know the true parameter a_0. However, we can easily check that $l(a)$ is indeed maximized by the true parameter $a = a_0$. Now assume we have actually performed the experiment N times, i.e. we have a sample consisting of N observations q_i. We can then define the function l_N:

$$l_N(a) = \frac{1}{N}\sum_{n=1}^{N}\left[q_n \log(a) + (1-q_n)\log(1-a)\right]$$

which serves as an N-estimate of the unknown function $l(a)$. Furthermore, by the law of large numbers, the estimate $l_N(a)$ converges to $l(a)$ for large sample size N. The maximum likelihood estimator (MLE) $\hat{a}$ for the true parameter can be found by maximizing $l_N(a)$:

$$\begin{aligned}\frac{dl_N}{da} &= 0 = \frac{1}{N}\sum_{n=1}^{N}\left[\frac{q_n}{\hat{a}} - \frac{1-q_n}{1-\hat{a}}\right]\\ 0 &= \frac{\sum_{n=1}^{N} q_n}{\hat{a}} - \frac{N - \sum_{n=1}^{N} q_n}{1-\hat{a}}\end{aligned}$$

Defining $n_s = \sum_{n=1}^{N} q_n$ as the number of successes, we find:

$$\hat{a} = \frac{\sum_{i=1}^{N} q_i}{N} = \frac{n_s}{N}$$

To estimate the error on the MLE, we will need to compute the Fisher information and evaluate it in the MLE.

$$\begin{aligned}I(\hat{a}) &= -E_{\hat{a}}\left[\frac{d^2}{da^2}\log p(q|\hat{a})\right]\\ &= -E_{\hat{a}}\left[\frac{d^2}{da^2}\left(q\log(a) + (1-q)\log(1-a)\right)_{a=\hat{a}}\right]\\ &= -E_{\hat{a}}\left[\frac{d}{da}\left(\frac{q}{a} - \frac{1-q}{1-a}\right)_{a=\hat{a}}\right]\\ &= -E_{\hat{a}}\left[-\frac{q}{\hat{a}^2} - \frac{1-q}{(1-\hat{a})^2}\right]\\ &= \frac{E_{\hat{a}}[q]}{\hat{a}^2} + \frac{1-E_{\hat{a}}[q]}{(1-\hat{a})^2}\\ &= \frac{\hat{a}}{\hat{a}^2} + \frac{1-\hat{a}}{(1-\hat{a})^2}\\ I(\hat{a}) &= \frac{1}{\hat{a}(1-\hat{a})}\end{aligned}$$

As a result, the variance on the MLE is given by:

$$\text{Var}[\hat{a}] = \frac{[I(\hat{a})]^{-1}}{N} = \frac{\hat{a}(1-\hat{a})}{N} = \frac{n_s}{N^2}(1-\frac{n_s}{N})$$

Finally, we can express the 95%-confidence interval (i.e. 2-sigma interval) for an estimate of the true parameters as:

$$\begin{aligned}a_0 &\approx \hat{a} \pm 2\sqrt{\text{Var}[\hat{a}]}\\ &\approx \frac{n_s}{N} \pm 2\frac{\sqrt{n_s(1-\frac{n_s}{N})}}{N}\end{aligned}$$

Example 2 - histogram estimation from MD simulation[subsec:Example2-single-histogram]

In this example, we want to estimate the probability density function $p(q)$ associated with finding a molecular system to have a configuration in which the collective variable $Q(\vec{r}^N)$ takes the value q through the construction of a histogram including an errorbar. The samples required for such estimate are provided by a molecular dynamics (or Monte Carlo) simulation (which we also assume to be uncorrelated). As we are interested in constructing a historgram, we define our probability model as was introduced in section *[subsec:Probability-model]*:

$$p(q|\vec{a}) = \sum_{k=1}^{K} a_k \phi_k(q)$$

where ϕ_k represents a function corresponding to the bin centered around value Q_k:

$$\phi_k(q) = \begin{cases} \frac{1}{\Delta} & q \in [Q_k - \frac{\Delta}{2}, Q_k + \frac{\Delta}{2}] \\ 0 & elsewhere \end{cases}$$

with Δ an a priori chosen bin width and Q_k the a priori chosen bin centers. Note that each bin function ϕ_k is normalized to 1. Hence, since the probability density function also needs to be normalized, we require:

$$\sum_{k=1}^{K} a_k = 1$$

This represents a constraint on the parameters $\vec{a}$ which we will impose on the maximization of the likelihood using the technique of Lagrange multipliers. We therefore replace the function $l_N(\vec{a})$ with its constrained alternative $l_N^{(c)}(\vec{a}, \mu)$:

$$\begin{aligned} l_N^{(c)}(\vec{a}, \mu) &= l_N(\vec{a}) + \mu \left[1 - \sum_{k=1}^{K} a_k\right] \\ &= \frac{1}{N}\sum_{n=1}^{N} \log p(q_n|\vec{a}) + \mu \left[1 - \sum_{k=1}^{K} a_k\right] \end{aligned}$$

We proceed with computing the MLE:

$$\begin{aligned} \frac{\partial}{\partial a_k} l_N^{(c)}(\vec{a}, \mu) &= 0 = \frac{1}{N}\sum_{n=1}^{N} \frac{1}{p(q_n|\vec{a})}\frac{\partial p}{\partial a_k}(q_i|\vec{a}) - \mu \\ \frac{\partial}{\partial \mu} l_N^{(c)}(\vec{a}, \mu) &= 0 = 1 - \sum_{k=1}^{K} a_k \end{aligned}$$

From Eq. *[eq:ex2_probdens]* we find:

$$\frac{\partial p}{\partial a_k}(q_n|\vec{a}) = \phi_k(q_n) = \frac{\delta_{k,k_n}}{\Delta}$$

Since $\phi_k(q_i)$ is only non-zero if $q_i \in [Q_k - \frac{\Delta}{2}, Q_k + \frac{\Delta}{2}]$, we introduce k_n which represents that k-value for which the corresponding bin contains q_n. As such, we can rewrite Eq. *[eq:ex2_MLEeq1]* as:

$$\begin{aligned}\frac{1}{\Delta}\sum_{n=1}^{N}\frac{\delta_{k,k_n}}{\sum_{k=1}^{K} a_k\phi_k(q_n)} &= N\mu \\ \sum_{n=1}^{N}\frac{\delta_{k,k_n}}{\sum_{l=1}^{K} a_l\delta_{l,k_n}} &= N\mu \\ \sum_{n=1}^{N}\frac{\delta_{k,k_n}}{a_{k_n}} &= N\mu \\ \frac{1}{a_k}\sum_{n=1}^{N}\delta_{k,k_n} &= N\mu \\ a_k &= \frac{\sum_{n=1}^{N}\delta_{k,k_n}}{N\mu}\end{aligned}$$

We define the quantity H_k as:

$$H_k = \sum_{n=1}^{N}\delta_{k,k_n}$$

which represents the number of samples with Q-value within the k-th bin $[Q_k - \frac{\Delta}{2}, Q_k + \frac{\Delta}{2}]$. As such, we find:

$$a_k = \frac{H_k}{N\mu}$$

To determine the Lagrange multiplier μ we use Eq. *[eq:ex2_MLEeq2]*:

$$1 = \sum_{k=1}^{K}\hat{a}_k = \sum_{k=1}^{K}\frac{H_k}{N\hat{\mu}} = \frac{\sum_{k=1}^{K} H_k}{N\hat{\mu}} = \frac{N}{N\hat{\mu}} \Rightarrow \hat{\mu} = 1$$

Hence, the MLE for the histogram populations can be found as:

$$\begin{aligned}\hat{a}_k &= \frac{H_k}{N} \\ p(q|\hat{\vec{a}}) &= \sum_{k=1}^{K}\frac{H_k}{N}\phi_k(q)\end{aligned}$$

Note that $\hat{a}_k$ also corresponds to the MLE estimator for the probability in a Bernoulli experiment where success is defined as "finding an MD sample with $Q(\vec{r}^N) \in [Q_k - \frac{\Delta}{2}, Q_k + \frac{\Delta}{2}]$". To compute a reliable estimate of the error bar on each histogram count, we first need to compute the Fisher information. Given the special meaning of the Lagrange multiplier μ (which isn't a true parameters we are interested in), we write the Fisher matrix formally as a block matrix (for convenience, we will drop the dependence of the Fisher matrix elements on the paramters $\vec{a}$ and μ):

$$\bar{\bar{I}} = \begin{bmatrix} \bar{\bar{I}}^{aa} & \bar{I}^{a\mu} \\ [\bar{I}^{a\mu}]^T & I^{\mu\mu}\end{bmatrix}$$

with the following matrix elements:

$$\begin{aligned}I^{aa}_{k,l} &= -E\left[\frac{\partial^2}{\partial a_k \partial a_l}\log p^{(c)}(q|\hat{\vec{a}},\hat{\mu})\right] \\ I^{a\mu}_{k} &= -E\left[\frac{\partial^2}{\partial a_k \partial \mu}\log p^{(c)}(q|\hat{\vec{a}},\hat{\mu})\right] \\ I^{\mu\mu} &= -E\left[\frac{\partial^2}{\partial \mu^2}\log p^{(c)}(q|\hat{\vec{a}},\hat{\mu})\right]\end{aligned}$$

where we introduced the probability density function $p^{(c)}(q|\vec{a},\mu)$ corresponding to the constrained function $l_N^{(c)}(\vec{a},\mu)$:

$$\begin{aligned}
l_N^{(c)}(\vec{a},\mu) &= \frac{1}{N}\sum_{n=1}^{N}\log p(q_n|\vec{a}) + \mu\left[1-\sum_{k=1}^{K}a_k\right] = \frac{1}{N}\sum_{n=1}^{N}\log p^{(c)}(q_n|\vec{a}) \\
\Rightarrow \log p^{(c)}(q_n|\vec{a}) &= \log p(q_n|\vec{a}) + \mu\left[1-\sum_{k=1}^{K}a_k\right]
\end{aligned}$$

Using this function, we can compute the Fisher matrix elements:

$$\begin{aligned}
I_{k,l}^{aa} &= -E\left[\frac{\partial}{\partial a_k}\left(\frac{1}{p(q|\hat{\vec{a}})}\frac{\partial p(q|\hat{\vec{a}})}{\partial a_l}-\mu\right)\right] = -E\left[\frac{\partial}{\partial a_k}\left(\frac{\phi_l(q)}{p(q|\hat{\vec{a}})}-\mu\right)\right] \\
&= E\left[\frac{\phi_l(q)}{\left(p(q|\hat{\vec{a}})\right)^2}\frac{\partial p(q|\hat{\vec{a}})}{\partial a_k}\right] = E\left[\frac{\phi_k(q)\phi_l(q)}{\left(\sum_{n=1}^{K}\hat{a}_n\phi_n(q)\right)^2}\right] \\
&= \int_{-\infty}^{+\infty}\frac{\phi_k(q)\phi_l(q)}{\left(\sum_{s=1}^{K}\hat{a}_s\phi_s(q)\right)^2}p^{(c)}(q|\vec{a},\mu)dq \\
&= \int_{-\infty}^{+\infty}\frac{\phi_k(q)\phi_l(q)}{\left(\sum_{s=1}^{K}\hat{a}_s\phi_s(q)\right)^2}\left(\sum_{s=1}^{K}\hat{a}_s\phi_s(q)\right)e^{\mu\overbrace{\left(1-\sum_{s=1}^{K}\hat{a}_s\right)}^{=0}}dq \\
&= \int_{-\infty}^{+\infty}\frac{\phi_k(q)\phi_l(q)}{\sum_{s=1}^{K}\hat{a}_s\phi_s(q)}dq \\
&= \sum_{m=0}^{K}\int_{Q_m-\frac{\Delta}{2}}^{Q_m+\frac{\Delta}{2}}\frac{\phi_k(q)\phi_l(q)}{\sum_{s=1}^{K}\hat{a}_s\phi_s(q)}dq = \sum_{m=0}^{K}\int_{Q_m-\frac{\Delta}{2}}^{Q_m+\frac{\Delta}{2}}\frac{\frac{\delta_{k,m}}{\Delta}\frac{\delta_{l,m}}{\Delta}}{\frac{\hat{a}_m}{\Delta}}dq \\
&= \sum_{m=0}^{K}\frac{\delta_{k,m}\delta_{l,m}}{\Delta\hat{a}_m}\underbrace{\int_{Q_m-\frac{\Delta}{2}}^{Q_m+\frac{\Delta}{2}}dq}_{=\Delta} \\
I_{k,l}^{aa} &= \frac{\delta_{k,l}}{\hat{a}_k}
\end{aligned}$$

The matrix elements coming from the other blocks are easier to compute:

$$\begin{aligned}
I_k^{a\mu} &= -E\left[\frac{\partial}{\partial\mu}\left(\frac{1}{p(q|\hat{\vec{a}})}\frac{\partial p(q|\hat{\vec{a}})}{\partial a_k}-\mu\right)\right] = -E\left[-1\right] = 1 \\
I^{\mu\mu} &= -E\left[\frac{\partial}{\partial\mu}\left(1-\sum_{k=1}^{K}a_k\right)\right] = -E\left[0\right] = 0
\end{aligned}$$

Hence we find the following Fisher matrix:

$$\bar{\bar{I}} = \begin{bmatrix} \frac{1}{\hat{a}_1} & 0 & \cdots & 0 & 1 \\ 0 & \frac{1}{a_2} & \cdots & 0 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \frac{1}{a_K} & 1 \\ 1 & 1 & \cdots & 1 & 0 \end{bmatrix}$$

The inverse of this matrix, which is proportional to the covariance we are interested in, is given by (taking into account the constraint $\sum_{k=1}^{K} a_k = 1$):

$$\left[\bar{\bar{I}}\right]^{-1} = \begin{bmatrix} \hat{a}_1(1-\hat{a}_1) & -\hat{a}_1\hat{a}_2 & \cdots & -\hat{a}_1\hat{a}_K & \hat{a}_1 \\ -\hat{a}_1\hat{a}_2 & \hat{a}_2(1-\hat{a}_2) & \cdots & -\hat{a}_2\hat{a}_K & \hat{a}_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ -\hat{a}_1\hat{a}_K & -\hat{a}_2\hat{a}_K & \cdots & \hat{a}_K(1-\hat{a}_K) & \hat{a}_K \\ \hat{a}_1 & \hat{a}_2 & \cdots & \hat{a}_K & -1 \end{bmatrix}$$

Since we are only interested in the true parameters $\vec{a}$, we find the following covariance matrix for these parameters:

$$\mathrm{Cov}\left[\hat{\vec{a}}, \hat{\vec{a}}\right] = \frac{1}{N} \begin{bmatrix} \hat{a}_1(1-\hat{a}_1) & -\hat{a}_1\hat{a}_2 & \cdots & -\hat{a}_1\hat{a}_K \\ -\hat{a}_1\hat{a}_2 & \hat{a}_2(1-\hat{a}_2) & \cdots & -\hat{a}_2\hat{a}_K \\ \vdots & \vdots & \ddots & \vdots \\ -\hat{a}_1\hat{a}_K & -\hat{a}_2\hat{a}_K & \cdots & \hat{a}_K(1-\hat{a}_K) \end{bmatrix}$$

Finally, this allows us to write down the error bar (again using 95%-confidence interval or 2-sigma) on the histogram probability for bin k:

$$a_{0,k} \approx \frac{H_k}{N} \pm 2\frac{\sqrt{H_k(1-\frac{H_k}{N})}}{N}$$

Note that this is again the same result as for the MLE estimator for the probability in a Bernoulli experiment. However, we now also have access to the covariance (and hence the correlation) between e.g. the probability in bin k and that in bin l:

$$\mathrm{Cov}\left[\hat{a}_k, \hat{a}_l\right] = \frac{-\hat{a}_k\hat{a}_l}{N} = -\frac{H_k H_l}{N^3}$$

$$\mathrm{Corr}\left[\hat{a}_k, \hat{a}_l\right] = \frac{\mathrm{Cov}\left[\hat{a}_k, \hat{a}_l\right]}{\sqrt{\mathrm{Var}\left[\hat{a}_k\right] \cdot \mathrm{Var}\left[\hat{a}_l\right]}} = \frac{\frac{-\hat{a}_k\hat{a}_l}{N}}{\sqrt{\frac{\hat{a}_k(1-\hat{a}_k)}{N}\frac{\hat{a}_l(1-\hat{a}_l)}{N}}} = -\sqrt{\frac{\hat{a}_k\hat{a}_l}{(1-\hat{a}_k)(1-\hat{a}_l)}}$$

From this, we can conclude that two large histogram probabilities are more strongly correlated than two small probabilities, which is intuïtively clear due to the constraint $\sum_{k=1}^{K} a_k = 1$.

6.1.6.2 Estimation from combining multiple simulation data sets

In the previous section, we investigated the problem of estimating the unkown parameters $\vec{a}$ (and their error bar) figuring in the probability density function $p(q|\vec{a})$ from sampled data coming from a single simulation which samples the unknown probability density. However, in many cases a regular moleculare simulation cannot access all relevant parts of phase space because of high barriers in (free) energy separating various (meta)stable equilibrium states. In that case, one can apply various so-called enhanced sampling techniques. An example is given by umbrella sampling, which involves performing not 1, but multiple equilibrium simulations each with a different additional bias potential aiming at enhancing the sampling in a certain region of phase space. As a result, each simulation will not sample the original probability density function, which we will now rename $p^{(0)}(q|\vec{a})$ for clarity, but the biased probability density function. Consider a series of M simulations, where simulation i is run with the bias potential $W^{(i)}(q)$ expressed in terms of the collective variable of interest. The biased probability density of this simulation will be given by:

$$p^{(i)}(q|\vec{a}) = f_i(\vec{a}) \cdot p^{(0)}(q|\vec{a}) \cdot e^{-\beta W^{(i)}(q)}$$

where

$$\frac{1}{f_i(\vec{a})} = \int p^{(0)}(q|\vec{a})e^{-\beta W^{(i)}(q)}dq$$

represents the normalisation factor for the biases probability density of simulation i and depends on the unknown parameters $\vec{a}$ of the unbiased probability density. We now find ourselves with a new hurdle. We have access to M simulations, each delivering N_i samples $\left\{q_{n_i}^{(i)}\right\}_{n_i=1}^{N_i}$, generated by a biases probability density $p^{(i)}(q|\vec{a})$ all related to the underlying unbiased probability density $p^{(0)}(q|\vec{a})$ we are interested in. The question we now have to answer is: how can be combine samples $\left\{q_{n_i}^{(i)}\right\}_{n_i=1}^{N_i}$ taken from different probability densities $p^{(i)}(q|\vec{a})$ to estimate a common set of parameters $\vec{a}$. We will tackle this question in the remainder of this section. In section *[subsec:Mixed-log-likelihood]* we introduce the weighted log-likelihood which is defined as the weighted sum of the log-likelihood functions of each simulation indepedently. This function will generate the MLE after maximalisation as an estimator for the true parameters. In section *[subsec:Covariance-of-MLE-mixedloglikelihood]* we derive the corresponding covariance. Finally, in section *[subsec:Example:-Multiple-histograms]* we apply the theory to estimate the unbiased probability density using a histogram estimator using input from multiple biased simulations as done in umbrella sampling.

Weighted log-likelihood functions[subsec:Mixed-log-likelihood]

In section *[subsec:Likelihood-and-MLE]* we introduced the sample-average log-likelihood function l_N of the random variable $\log p(q|\vec{a})$ using the samples taken from the simulation. However, as we now have M different simulations, each with their own samples, we can define a sample-average log-likelihood for each simulation:

$$l_{N_i}^{(i)}(\vec{a}) = \frac{1}{N_i}\sum_{n_i=1}^{N_i} \log\left[p^{(i)}(q_{n_i}^{(i)}|\vec{a})\right]$$

Similarly, we can also define a population-mean log-likelihood function corresponding to each simulation:

$$l^{(i)}(\vec{a}) = E^{(i)}\left[\log p^{(i)}(q|\vec{a})\right] = \int_{-\infty}^{+\infty} \log p^{(i)}(q|\vec{a}) \cdot p^{(i)}(q|\vec{a})dq$$

in which $E^{(i)}[\cdot]$ represents the true population mean in simulation i (with its corresponding bias). As was discussed in section *[subsec:Likelihood-and-MLE]*, the law of large numbers implies:

$$l_{N_i}^{(i)}(\vec{a}) \xrightarrow{N_i\to+\infty} l^{(i)}(\vec{a})$$

In order to be able to combine all simulation samples, we define the weighted versions of both the sample average and population mean of the log-likilihood as a weighted average of the simulation contributions:

$$l_{\vec{N}}(\vec{a}) = \sum_{i=1}^{M} c_i l_{N_i}^{(i)}(\vec{a}) = \sum_{i=1}^{M}\frac{c_i}{N_i}\sum_{n_i=1}^{N_i}\log\left[p^{(i)}(q_{n_i}^{(i)}|\vec{a})\right]$$
$$l(\vec{a}) = \sum_{i=1}^{M} c_i l^{(i)}(\vec{a}) = \sum_{i=1}^{M} c_i E^{(i)}\left[\log p^{(i)}(q|\vec{a})\right]$$

Where we used the notation $\vec{N} = (N_1, N_2, \ldots, N_M)$. The weights c_i will be determined later, for now they represent positive real numbers that sum to 1[4]. Given the fact that the true paramaters $\vec{a}_0$ will result

[4] Both $l_{\vec{N}}$ and l can be multiplied (or divided) by a constant without any impact on its derived features such as the MLE and its covariance. Therefore, we can see that the weights are determined up to a constant factor which we choose such that the sum of the weights equals 1.

in the true biased probability density $p^{(i)}(q|\vec{a}_0)$ for each simulation i, we can also state that these true parameters will maximize the population-mean log-likelihood of each simulation individually:

$$\left.\frac{\partial}{\partial\vec{a}}l^{(i)}(\vec{a})\right|_{\vec{a}=\vec{a}_0}=0$$

and hence the true parameters will also maximize the weighted population-mean log-likelihood:

$$\left.\frac{\partial}{\partial\vec{a}}l(\vec{a})\right|_{\vec{a}=\vec{a}_0}=0$$

It is trivial to see that the law of large numbers again implies:

$$l_{\vec{N}}(\vec{a})\xrightarrow{\vec{N}\to+\infty}l(\vec{a})$$

where $\vec{N}\to+\infty$ is short for $\forall i:N_i\to+\infty$. We can now compute the maximum likelihood estimator by maximizing $l_{\vec{N}}(\vec{a})$:

$$\forall k:\frac{\partial}{\partial a_k}l_{\vec{N}}(\vec{a})=0$$

its solution gives the MLE $\hat{\vec{a}}$.

Covariance of the MLE for the weighted log-likelihood[subsec:Covariance-of-MLE-mixedloglikelihood]

In section *[subsec:Asymptotic-normality]* we discussed how the covariance matrix of the MLE can be computed from the Fisher matrix. Similarly as we did for the individual log-likelihood functions in the previous section, we first define the Fisher matrices for each simulation individually:

$$I_{kl}^{(i)}=-E^{(i)}\left[\frac{\partial^2}{\partial a_k\partial a_l}\left(\log p^{(i)}(q|\vec{a}_0)\right)\right]=-\frac{\partial^2}{\partial\vec{a}\partial\vec{a}}l(\vec{a}_0)$$

The final question that remains now is: how do we combine the Fisher matrices of the various simulations to obtain the covariance matrix on the MLE derived from the weighted log-likelihood? For this purpose, we start by recalling how we have proven prove the asymptotic normality of the MLE in section *[subsec:Asymptotic-normality]*. Therein, by using the Mean Value Theorem, we could derive the relation as expressed by Eq. *[eq:asnorm_rel-1]*:

$$\frac{\partial^2}{\partial\vec{a}\partial\vec{a}}l_N(\vec{a}_0)\cdot\left(\hat{\vec{a}}-\vec{a}_0\right)=-\frac{\partial}{\partial\vec{a}}l_N(\vec{a}_0)\text{ as }N\to+\infty$$

In the derivation of this relation, we used the fact that $\hat{\vec{a}}$ maximizes $l_N(\vec{a})$ as the MLE. Here, one has to be careful in the current situation of multiple simulations: we do not know at this point if $\hat{\vec{a}}$ maximizes each $l_{N_i}^{(i)}(\vec{a})$ individually. However, $\hat{\vec{a}}$ does maximize the weighted log-likelihood $l_{\vec{N}}(\vec{a})$, therefore we can write:

$$\frac{\partial^2}{\partial\vec{a}\partial\vec{a}}l_{\vec{N}}(\vec{a}_0)\cdot\left(\hat{\vec{a}}-\vec{a}_0\right)=-\frac{\partial}{\partial\vec{a}}l_{\vec{N}}(\vec{a}_0)\text{ as }\vec{N}\to+\infty$$

which we can now rewrite as:

$$\begin{aligned}\left[\frac{\partial^2}{\partial\vec{a}\partial\vec{a}}\left(\sum_{i=1}^M c_i l_{N_i}^{(i)}(\vec{a})\right)\right]\cdot\left(\hat{\vec{a}}-\vec{a}_0\right)&=-\frac{\partial}{\partial\vec{a}}\left(\sum_{i=1}^M c_i l_{N_i}^{(i)}(\vec{a})\right)\\ \left[\sum_{i=1}^M c_i\left(\frac{\partial^2}{\partial\vec{a}\partial\vec{a}}l_{N_i}^{(i)}(\vec{a}_0)\right)\right]\cdot\left(\hat{\vec{a}}-\vec{a}_0\right)&=-\sum_{i=1}^M c_i\left(\frac{\partial}{\partial\vec{a}}l_{N_i}^{(i)}(\vec{a}_0)\right)\\ \left[\sum_{i=1}^M c_i\bar{\bar{I}}^{(i)}\right]\cdot\left(\hat{\vec{a}}-\vec{a}_0\right)&=\sum_{i=1}^M c_i\left(\frac{\partial}{\partial\vec{a}}l_{N_i}^{(i)}(\vec{a}_0)\right)\end{aligned}$$

If we now focus on the right-hand side, then we can rewrite this as:

$$\begin{aligned}
\sum_{i=1}^{M} c_i \left(\frac{\partial}{\partial \vec{a}} l_{N_i}^{(i)}(\vec{a}_0)\right) &= \sum_{i=1}^{M} c_i \left[\frac{1}{N_i}\sum_{n_i=1}^{N_i}\frac{\partial}{\partial \vec{a}}\left(\log p^{(i)}(q_{n_i}^{(i)}|\vec{a}_0\right)\right] \\
&= \sum_{i=1}^{M} c_i \left[\frac{1}{N_i}\sum_{n_i=1}^{N_i} X_{n_i}^{(i)}\right] \\
&= \sum_{i=1}^{M} c_i \bar{X}^{(i)}
\end{aligned}$$

where we define the random variable $X^{(i)}$, its samples $X_{n_i}^{(i)}$ and its sample average $\bar{X}^{(i)}$ of simulation i as follows:

$$\begin{aligned}
X^{(i)} &= \frac{\partial}{\partial \vec{a}}\left(\log p^{(i)}(q|\vec{a}_0\right) \\
X_{n_i}^{(i)} &= \frac{\partial}{\partial \vec{a}}\left(\log p^{(i)}(q_{n_i}^{(i)}|\vec{a}_0\right) \\
\bar{X}^{(i)} &= \frac{1}{N_i}\sum_{n_i=1}^{N_i} X_{n_i}^{(i)} = \frac{1}{N_i}\sum_{n_i=1}^{N_i}\frac{\partial}{\partial \vec{a}}\left(\log p^{(i)}(q_{n_i}^{(i)}|\vec{a}_0\right)
\end{aligned}$$

Furthermore, we also know that the sample averages $\bar{X}^{(i)}$ satisfy:

$$\begin{aligned}
\sum_{i=1}^{M} c_i \left(E^{(i)}\left[\bar{X}^{(i)}\right]\right) &= \sum_{i=1}^{M} c_i \left(E^{(i)}\left[X^{(i)}\right]\right) \\
&= \sum_{i=1}^{M} c_i \left(E^{(i)}\left[\frac{\partial}{\partial \vec{a}}\left(\log p^{(i)}(q|\vec{a}_0\right)\right]\right) \\
&= \sum_{i=1}^{M} c_i \left(\frac{\partial}{\partial \vec{a}} l^{(i)}(\vec{a}_0)\right) \\
&= \frac{\partial}{\partial \vec{a}}\left(\sum_{i=1}^{M} c_i l^{(i)}(\vec{a}_0)\right) \\
&= \frac{\partial}{\partial \vec{a}} l(\vec{a}_0) \\
&= 0
\end{aligned}$$

where we remind the reader that the notation $\frac{\partial}{\partial x} f(x_0)$ means we first take the derivative of f towards x and then evaluate the derivative in $x = x_0$. Furthermore, in the last line we used the fact that the true parameters $\vec{a}_0$ maximize the weighted population-mean log-likelihood (see Eq. *[eq:truepars_max_popmeanloglik]*). This result can now be used to rewrite Eq. *[eq:CovMLE_mixed_rel1]* as:

$$\begin{aligned}
\sum_{i=1}^{M} c_i \left(\frac{\partial}{\partial \vec{a}} l_{N_i}^{(i)}(\vec{a}_0)\right) &= \sum_{i=1}^{M} c_i \bar{X}^{(i)} - \sum_{i=1}^{M} c_i \left(E^{(i)}\left[\bar{X}^{(i)}\right]\right) \\
&= \sum_{i=1}^{M} c_i \left(\bar{X}^{(i)} - E^{(i)}\left[\bar{X}^{(i)}\right]\right)
\end{aligned}$$

As was discussed earlier (see Eq. *[eq:asnorm_rel2]*), the central limit theorem implies for each simulation seperately:

$$\bar{X}^{(i)} - E^{(i)}\left[\bar{X}^{(i)}\right] \xrightarrow{N_i \to +\infty} \mathcal{N}\left(0, \frac{\bar{\bar{I}}^{(i)}}{N_i}\right)$$

If we now put Eqs. *[eq:CovMLE_mixed_rel0]*, *[eq:CovMLE_mixed_rel1]* and *[eq:CovMLE_mixed_rel2]* together, we arrive at:

$$\left[\sum_{i=1}^{M} c_i \bar{\bar{I}}^{(i)}\right] \cdot \left(\hat{\vec{a}} - \vec{a}_0\right) = \sum_{i=1}^{M} c_i \left(\bar{X}^{(i)} - E^{(i)}\left[\bar{X}^{(i)}\right]\right) \xrightarrow{\vec{N}\to+\infty} \sum_{i=1}^{M} c_i \cdot \mathcal{N}\left(0, \frac{\bar{\bar{I}}^{(i)}}{N_i}\right) = \mathcal{N}\left(0, \sum_{i=1}^{M} \frac{c_i^2}{N_i}\bar{\bar{I}}^{(i)}\right)$$

or equivalently with covariance matrix $\bar{\bar{\sigma}}$:

$$\hat{\vec{a}} \xrightarrow{\vec{N}\to+\infty} \mathcal{N}\left(\vec{a}_0, \bar{\bar{\sigma}}\right)$$
$$\bar{\bar{\sigma}} = \left[\sum_{i=1}^{M} c_i \bar{\bar{I}}^{(i)}\right]^{-2} \cdot \left[\sum_{i=1}^{M} \frac{c_i^2}{N_i}\bar{\bar{I}}^{(i)}\right]$$

At this point, it is interesting to discuss the impact of the weights c_i. The covariance of the MLE clearly depends on the choice of the weights[5] . To illustrate this depende, we consider a few limiting cases:

1. $c_i = \delta_{i,j}$, corresponding to the situation we actually only consider a single simulation, that is simulation j. In this case the covariance becomes:

$$\bar{\bar{\sigma}} = \left[\bar{\bar{I}}^{(j)}\right]^{-2} \cdot \left[\frac{1}{N_j}\bar{\bar{I}}^{(j)}\right] = \left[N_j \bar{\bar{I}}^{(j)}\right]^{-1}$$

   corresponding indeed to the previous results of section *[subsec:Asymptotic-normality]*.

2. $c_i = \frac{1}{M}$, corresponding to the same weight given to each simulation. The covariance matrix becomes:

$$\bar{\bar{\sigma}} = \left[\sum_{i=1}^{M} \bar{\bar{I}}^{(i)}\right]^{-2} \cdot \left[\sum_{i=1}^{M} \frac{1}{N_i}\bar{\bar{I}}^{(i)}\right]$$

   If the number of steps in one particular simulation would approach zero, say $N_j \to 0$, we found that $\bar{\bar{\sigma}}$ would be predominantly determined by the Fisher matrix of that simulation and moreover, the covariance would approach infinity. This is offcourse something we want to avoid at all cost, hence setting all weights equal is not a good idea.

3. $c_i = \frac{N_i}{N}$ with $N = \sum_i N_i$ the accumulated number of simulation steps. This choice corresponds to giving a weight proportional to the size of the simulation, i.e. its number of simulation steps. This makes sens *a priori* as larger simulations generally mean more information. The resulting covariance matrix for this choice takes on a much more simple form:

$$\begin{aligned}\bar{\bar{\sigma}} &= \left[\sum_{i=1}^{M} \frac{N_i}{N}\bar{\bar{I}}^{(i)}\right]^{-2} \cdot \left[\sum_{i=1}^{M} \frac{N_i}{N^2}\bar{\bar{I}}^{(i)}\right] \\ &= \left[\sum_{i=1}^{M} N_i\bar{\bar{I}}^{(i)}\right]^{-2} \cdot \left[\sum_{i=1}^{M} N_i\bar{\bar{I}}^{(i)}\right] \\ &= \left[\sum_{i=1}^{M} N_i\bar{\bar{I}}^{(i)}\right]^{-1}\end{aligned}$$

   This expression is makes much sense as it represents the inverse of the accumulated information, giving larger weight to larger simulations. Furthermore, if one simulation approaches zero in size ($N_j \to 0$), its contribution simply drops out. Furthermore, it can be proven that this choice of weights minimizes the variance, i.e. diagonal covariance matrix elements, and hence error on each parameters [[6]].

[5] However, it can easily be seen that multiplying each weight with the same factor does not influence the covariance. Hence, we choose this factor so that the sum of the weights is 1 as was already discussed earlier.

[6] I have not proven this (yet), but as this choice of weights corresponds to WHAM (see Example 3 in next section) and WHAM is known to correspond to the choice of weights that minimizes tha variance, this makes sense.

Example 3 - Multiple histograms from umbrella sampling[subsec:Example:-Multiple-histograms]

In this section, we apply the relations derived in the previous sections to estimate the optimal histogram-based probability density function using samples taken from various biased simulations. The proposed model for the unbiased probability density is identical as in section *[subsec:Example2-single-histogram]*:

$$p^{(0)}(q|\vec{a}) = \sum_{k=1}^{K} a_k \phi_k(q)$$

where ϕ_k represents a function corresponding to the bin centered around value Q_k:

$$\phi_k(q) = \begin{cases} \frac{1}{\Delta} & q \in [Q_k - \frac{\Delta}{2}, Q_k + \frac{\Delta}{2}] \\ 0 & elsewhere \end{cases}$$

with Δ an a priori chosen bin width and Q_k the a priori chosen bin centers. Note that each bin function ϕ_k is normalized to 1. Hence, since the probability density function also needs to be normalized, we require:

$$\sum_{k=1}^{K} a_k = 1$$

which we will address using Lagrange multipliers. The biased probability density of simulation i will be given by:

$$p^{(i)}(q|\vec{a}) = f_i(\vec{a}) \cdot p^{(0)}(q|\vec{a}) \cdot e^{-\beta W^{(i)}(q)}$$

where we can now compute the normalisation factors:

$$\begin{aligned} \frac{1}{f_i(\vec{a})} &= \int p^{(0)}(q|\vec{a}) e^{-\beta W^{(i)}(q)} dq \\ &= \sum_{k=1}^{K} a_k \int_{-\infty}^{+\infty} \phi_k(q) e^{-\beta W^{(i)}(q)} dq \\ &= \sum_{k=1}^{K} a_k \underbrace{\frac{1}{\Delta} \int_{Q_k - \frac{\Delta}{2}}^{Q_k + \frac{\Delta}{2}} e^{-\beta W^{(i)}(q)} dq}_{\triangleq b_{ik}} \\ \frac{1}{f_i(\vec{a})} &= \sum_{k=1}^{K} b_{ik} a_k \end{aligned}$$

and where we introduced the overlap b_{ik} of the bias in simulation i with bin k. In order to compute the MLE, we construct the mixed sample-average log-likelihood function:

$$\begin{aligned} l_{\vec{N}}(\vec{a}) &= \sum_{i=1}^{M} \frac{c_i}{N_i} \sum_{n_i=1}^{N_i} \log p^{(i)}(q_{n_i}^{(i)}|\vec{a}) \\ &= \sum_{i=1}^{M} \frac{c_i}{N_i} \sum_{n_i=1}^{N_i} \left[\log p^{(0)}(q_{n_i}^{(i)}|\vec{a}) + \log f_i(\vec{a}) - \beta W^{(i)}(q_{n_i}^{(i)}) \right] \\ &= \sum_{i=1}^{M} \frac{c_i}{N_i} \sum_{n_i=1}^{N_i} \left[\log \left(\sum_{k=1}^{K} a_k \phi_k(q_{n_i}^{(i)}) \right) - \log \left(\sum_{k=1}^{K} b_{ik} a_k \right) - \beta W^{(i)}(q_{n_i}^{(i)}) \right] \end{aligned}$$

This function needs to be maximized with respect to the parameters $\vec{a}$ under the constraint $\sum_{k=1}^K a_k = 1$. Since the last term in the equation above, i.e. the contribution of the bias, does not depend on the parameters, we can ommit it when doing the maximization. The constraint is implemented using Lagrange multipliers:

$$\forall s: \frac{\partial}{\partial a_k}\left(\sum_{i=1}^M \frac{c_i}{N_i}\sum_{n_i=1}^{N_i}\left[\log\left(\sum_{s=1}^K a_s\phi_s(q_{n_i}^{(i)})\right) - \log\left(\sum_{s=1}^K b_{is}a_s\right)\right] + \mu\left[1-\sum_{s=1}^K a_s\right]\right) = 0$$

$$\sum_{s=1}^K a_s = 1$$

We now proceed by rewriting the first set of equations:

$$\sum_{i=1}^M \frac{c_i}{N_i}\sum_{n_i=1}^{N_i}\left[\frac{\phi_k(q_{n_i}^{(i)})}{\sum_{s=1}^K a_s\phi_s(q_{n_i}^{(i)})} - \frac{b_{ik}}{\sum_{s=1}^K b_{is}a_s}\right] - \mu = 0$$

Since $\phi_k(q_{n_i}^{(i)})$ is only non-zero if $q_{n_i}^{(i)} \in [Q_k - \frac{\Delta}{2}, Q_k + \frac{\Delta}{2}]$, we introduce $k_{n_i}^{(i)}$ which represents the bin k-value for which the corresponding bin contains $q_{n_i}^{(i)}$. We can rewrite the above equation as:

$$\sum_{i=1}^M \frac{c_i}{N_i}\left[\sum_{n_i=1}^{N_i}\frac{\delta_{k,k_{n_i}^{(i)}}/\Delta}{\sum_{s=1}^K a_s\delta_{s,k_{n_i}^{(i)}}/\Delta} - N_i\frac{b_{ik}}{\sum_{s=1}^K b_{is}a_s}\right] = \mu$$

$$\sum_{i=1}^M \frac{c_i}{N_i}\left[\sum_{n_i=1}^{N_i}\frac{\delta_{k,k_{n_i}^{(i)}}}{a_{k_{n_i}^{(i)}}} - N_i\frac{b_{ik}}{\sum_{s=1}^K b_{is}a_s}\right] = \mu$$

$$\sum_{i=1}^M \frac{c_i}{N_i}\left[\frac{1}{a_k}\sum_{n_i=1}^{N_i}\delta_{k,k_{n_i}^{(i)}} - N_i\frac{b_{ik}}{\sum_{s=1}^K b_{is}a_s}\right] = \mu$$

We define H_{ik} as the histogram count in bin k from simulation i:

$$H_{ik} = \sum_{n_i=1}^{N_i}\delta_{k,k_{n_i}^{(i)}}$$

which allows us to further rewrite the expression as:

$$\frac{1}{a_k}\sum_{i=1}^M \frac{c_i}{N_i}H_{ik} - \sum_{i=1}^M c_i\frac{b_{ik}}{\sum_{s=1}^K b_{is}a_s} = \mu$$

We now first proceed by computing the value of the lagrange multiplier μ, which we do by multiplying both sides of the above equation by a_k a,d summing over k:

$$\sum_{i=1}^M \frac{c_i}{N_i}\sum_{k=1}^K H_{ik} - \sum_{i=1}^M c_i\frac{\sum_{k=1}^K b_{ik}a_k}{\sum_{s=1}^K b_{is}a_s} = \mu\sum_{k=1}^K a_k$$

Taking into account the constraint $\sum_{k=1}^K a_k = 1$ as well as the relation $\sum_{k=1}^K H_{ik} = N_i$ (i.e. the total histogram count in simulation i equals the number of simulation steps), we find:

$$\sum_{i=1}^M c_i - \sum_{i=1}^M c_i = \mu$$

In other words: $\hat{\mu} = 0$. Substituting this result in Eq. *[eq:ex3_rel1]* we find the MLE estimator for the parameters:

$$\hat{a}_k = \frac{\sum_{i=1}^{M} \frac{c_i}{N_i} H_{ik}}{\sum_{i=1}^{M} c_i \frac{b_{ik}}{\sum_{s=1}^{K} b_{is} a_s}}$$

which we rewrite using the definition of the normalisation factors f_i:

$$\hat{a}_k = \frac{\sum_{i=1}^{M} \frac{c_i}{N_i} H_{ik}}{\sum_{i=1}^{M} c_i f_i b_{ik}}$$
$$f_i^{-1} = \sum_{k=1}^{K} b_{ik} \hat{a}_k$$

These two coupled equations need to be solved iteratively to obtain the MLE estimator. As we recall from Section *[subsec:Covariance-of-MLE-mixedloglikelihood]* an optimal choice of the weight factors was given by $c_i = \frac{N_i}{N}$, which results in:

$$\begin{aligned} \hat{a}_k &= \frac{\sum_{i=1}^{M} H_{ik}}{\sum_{i=1}^{M} N_i f_i b_{ik}} \\ f_i^{-1} &= \sum_{k=1}^{K} b_{ik} \hat{a}_k \end{aligned} \tag{6.4}$$

These equations represent the WHAM (i.e. Weighted-Histogram Analysis Method) equations. Finally, we also compute the covariance matrix of the MLE. Therefore, we first compute the Fisher matrix for each biases simulation separately:

$$\bar{\bar{I}}^{(i)} = \begin{bmatrix} \bar{\bar{I}}^{(i),(aa)} & \bar{1} \\ \bar{1}^T & 0 \end{bmatrix}$$

where we already applied the block matrix structure resulting from the Lagrange multiplier μ for the applied constraint $\sum_{k=1}^{K} a_k = 1$ as was discussed in section *[subsec:Example2-single-histogram]* with $\bar{1}$ a K-dimensional column matrix with only ones. The matrix elements of the aa-block corresponding to the paramaters $\vec{a}$ can be computed as follows:

$$\begin{aligned}
I_{kl}^{(i),(aa)} &= -E^{(i)}\left[\frac{\partial^2}{\partial a_k \partial a_l}\log p^{(i)}(q|\vec{a})\right] \\
&= -E^{(i)}\left[\frac{\partial}{\partial a_k}\left(\frac{\phi_l(q)}{\sum_{s=1}^K a_s\phi_s(q)} - \frac{b_{il}}{\sum_{s=1}^K b_{is}a_s} - \mu\right)\right] \\
&= E^{(i)}\left[\frac{\phi_k(q)\phi_l(q)}{\left[\sum_{s=1}^K a_s\phi_s(q)\right]^2} - \frac{b_{ik}b_{il}}{\left[\sum_{s=1}^K b_{is}a_s\right]^2}\right] \\
&= \int_{-\infty}^{+\infty}\frac{\phi_k(q)\phi_l(q)}{\left[\sum_{s=1}^K a_s\phi_s(q)\right]^2}p^{(i)}(q|\vec{a})dq - \frac{b_{ik}b_{il}}{\left[\sum_{s=1}^K b_{is}a_s\right]^2} \\
&= \int_{-\infty}^{+\infty}\frac{\phi_k(q)\phi_l(q)}{\left[\sum_{s=1}^K a_s\phi_s(q)\right]^2}f_i\left(\sum_{s=1}^K a_s\phi_s(q)\right)e^{-\beta U^{(i)}(q)}dq - \frac{b_{ik}b_{il}}{\left[\sum_{s=1}^K b_{is}a_s\right]^2} \\
&= f_i\int_{-\infty}^{+\infty}\frac{\phi_k(q)\phi_l(q)}{\sum_{s=1}^K a_s\phi_s(q)}e^{-\beta U^{(i)}(q)}dq - f_i^2 b_{ik}b_{il} \\
&= f_i\int_{Q_k-\frac{\Delta}{2}}^{Q_k+\frac{\Delta}{2}}\frac{\frac{1}{\Delta}\frac{\delta_{kl}}{\Delta}}{\frac{a_k}{\Delta}}e^{-\beta U^{(i)}(q)}dq - f_i^2 b_{ik}b_{il} \\
&= f_i\frac{\delta_{kl}}{a_k}\frac{1}{\Delta}\int_{Q_k-\frac{\Delta}{2}}^{Q_k+\frac{\Delta}{2}}e^{-\beta U^{(i)}(q)}dq - f_i^2 b_{ik}b_{il} \\
I_{kl}^{(i),(aa)} &= \frac{f_i b_{ik}}{a_k}\delta_{kl} - f_i^2 b_{ik}b_{il}
\end{aligned}$$

CHAPTER

SEVEN

CONTACT

For support and bug reports, please contact us at louis.vanduyfhuys@ugent.be.

CHAPTER EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX