

# **MDORADO**

**Documentation of Version 0.2.0**

Jan Neumann,  
August 30, 2021, Rostock

# Contents

<b>1</b>	<b>Importing Testfiles</b>	<b>3</b>
1.1	File references . . . . .	3
1.2	Example . . . . .	4
<b>2</b>	<b>Correlate</b>	<b>5</b>
2.1	Function . . . . .	5
<b>3</b>	<b>Lifetimes</b>	<b>6</b>
3.1	Function . . . . .	7
3.2	Example . . . . .	8
<b>4</b>	<b>gofr</b>	<b>10</b>
4.1	Function . . . . .	11
4.2	Example . . . . .	13
<b>5</b>	<b>hb_analyze</b>	<b>15</b>
5.1	Function . . . . .	15
5.2	Example and Visualization . . . . .	17
<b>6</b>	<b>Mean Square Displacement</b>	<b>21</b>
6.1	Functions . . . . .	21
6.2	Example . . . . .	23

# 1 Importing Testfiles

A number of files for unit testing and examples are shipped with the installation of MDorado. Their absolute paths can be obtained via a python import statement using the `mdorado.data.datafilenames` module.

## 1.1 File references

<code>water_topology</code>	Gromacs <code>.tpr</code> file ( <code>water.tpr</code> ) of a small water simulation used as topology reference by MDAnalysis. Is used in a number of unit tests and examples throughout all functions.
<code>water_trajectory</code>	Gromacs <code>.xtc</code> file ( <code>water.xtc</code> ) of a small water simulation used as trajectory reference by MDAnalysis. Is used in a number of unit tests and examples throughout all functions.
<code>test_gofr_ss</code>	Textfile ( <code>gofr_ss.dat</code> ) containing the reference output of the <code>mdorado.gofr.Gofr</code> calculation using the <code>"site-site"</code> mode. Is used in the unit test of the <code>mdorado.gofr</code> module.
<code>test_gofr_cc</code>	Textfile ( <code>gofr_cc.dat</code> ) containing the reference output of the <code>mdorado.gofr.Gofr</code> calculation using the <code>"cms-cms"</code> mode. Is used in the unit test of the <code>mdorado.gofr</code> module.
<code>test_gofr_sc</code>	Textfile ( <code>gofr_sc.dat</code> ) containing the reference output of the <code>mdorado.gofr.Gofr</code> calculation using the <code>"site-cms"</code> mode. Is used in the unit test of the <code>mdorado.gofr</code> module.
<code>test_hbanalyze</code>	Textfile ( <code>hb_analyze.dat</code> ) containing the reference output of the <code>mdorado.hb_analyze.hb_analyze</code> calculation. Is used in the unit test of the <code>mdorado.hb_analyze</code> module.

<code>test_lifetime</code>	Textfile ( <code>lifetime_test.dat</code> ) containing the reference output of the <code>mdorado.lifetime.calc_lifetime</code> calculation. Is used in the unit test of the <code>mdorado.lifetime</code> module.
<code>test_wrap</code>	File ( <code>msd_molpos.npy</code> ) containing the reference output of the <code>mdorado.msd.unwrap</code> calculation. Is used in the unit test of the <code>mdorado.msd</code> module.
<code>test_msd</code>	Textfile ( <code>msd_h.dat</code> ) containing the reference output of the <code>mdoradomsd.msd</code> calculation. Is used in the unit test of the <code>mdorado.msd</code> module.

## 1.2 Example

In particular the water simulation can be used as an example input to test the functionality of MDorado. It is used in that way in many examples throughout this documentation, so the functions can be tested on an user-independent trajectory. This code, for example, is an example for the `hb_analyze` module and imports the file paths of the `water.tpr` (`water_topology`) and `water.xtc` (`water_trajectory`):

---

```

1 import MDAnalysis
2 from mdorado.hb_analyze import hb_analyze
3 from mdorado.data.datafilenames import water_topology,
   ↪ water_trajectory
4
5 u = MDAnalysis.Universe(water_topology, water_trajectory)
6 xgrp = u.select_atoms("name ow")
7 hgrp = u.select_atoms("name hw")[:,2]
8
9 hb_analyze(universe=u, xgrp=xgrp, hgrp=hgrp, rmin=1.5, rmax=5,
   ↪ cosalphamin=-1, cosalphamax=1, bins=50)

```

---

## 2 Correlate

This function uses `scipy.signal.correlate` to cross correlate two discrete functions  $a(t)$  and  $b(t)$ . The function computes  $\langle a(0)b(t) \rangle$  directly via sums or using a Fast Fourier Transform algorithm, depending on which is faster (see `scipy.signal.convolve`). In addition to the original `scipy` functionality, the function tailors the correlation function so that only non-negative time values ( $\langle a(0)b(t) \rangle$  for  $t \geq 0$ ) are returned. It may only be sensible to calculate such a function if  $a(t)$  and  $b(t)$  (and the corresponding array elements `a[t]` and `b[t]`) reference the same point in time and the arrays `a` and `b` are of equal length. The autocorrelation function  $\langle a(0)a(t) \rangle$  is computed if `b=None` (default).

### 2.1 Function

```
mdorado.correlations.correlate(a, b=None)
```

#### Parameters:

- a:** one-dimensional ndarray or list  
Discrete values of the function  $a(t)$ .
- b:** one-dimensional ndarray or list, optional  
Discrete values of the function  $b(t)$ . If `None` the autocorrelation function  $\langle a(0)a(t) \rangle$  is computed. Default is `None`.

#### Returns: ndarray

An ndarray containing the correlation function  $\langle a(0)b(t) \rangle$  for  $t \geq 0$  is returned.

### 3 Lifetimes

To analyze the lifetime of a hydrogen bond or any other impermanent interaction we can define a bonding operator  $h(t)$  which is unity if the criteria for the interaction are fulfilled and zero otherwise:<sup>[1,2]</sup>

$$h(t) = \begin{cases} 1, & \text{if criteria are fulfilled} \\ 0, & \text{otherwise} \end{cases} . \quad (3.1)$$

The fluctuations of  $h(t)$  can be described by the autocorrelation function  $C(t)$

$$C(t) = \frac{\langle h(0)h(t) \rangle - \langle h \rangle^2}{\langle h \rangle}, \quad (3.2)$$

which describes the probability of the bond being intact at the time  $t$  if the bond was intact at  $t = 0$ . The so-called intermittent lifetime of the interaction can be estimated from  $C(t)$ .<sup>[3-6]</sup>

The reactive flux approach<sup>[4,6-9]</sup> is another approach to estimate the lifetime of such interactions. To follow that approach we require the function  $k_{\text{in}}(t)$

$$k_{\text{in}}(t) = - \frac{\langle \dot{h}(0)[1 - h(t)]H(t) \rangle}{\langle h \rangle}, \quad (3.3)$$

where  $\dot{h}$  denotes the time-derivative of  $h(t)$ .  $H(t)$  is a vicinity operator closely related to  $h(t)$ . If the donor and acceptor of the interaction are “near” each other  $H(t)$  equals unity otherwise it equals zero.

The purpose of `calc_lifetime` is to calculate the correlation functions  $\langle h(0)h(t) \rangle$  from equation 3.2 and  $-\langle \dot{h}(0)[1 - h(t)]H(t) \rangle$  from equation 3.3. Normalizing the correlation functions will be up to the user, since several approaches are viable.<sup>[10]</sup> To obtain both correlations we first need to determine  $h(t)$  and  $H(t)$  for every donor-acceptor pair. After that, we are able to compute both correlation functions and average them over all donor-acceptor pairs.

## 3.1 Function

```
mdorado.lifetime.calc_lifetime(universe, timestep, xgrp, hgrp,  
    cutoff_hy, cutoff_xy, angle_cutoff, ygrp=None, nproc=1,  
    check_memory=True)
```

### Parameters:

<code>universe:</code>	MDAnalysis.Universe Universe containing the trajectory.
<code>timestep:</code>	int or float Timestep between configurations in the <code>universe</code> . The unit is freely selectable and will influence the units of the output.
<code>xgrp:</code>	AtomGroup from MDAnalysis AtomGroup containing all atoms X involved in the interaction X-H...Y. Has to be the same size as <code>hgrp</code> .
<code>hgrp:</code>	AtomGroup from MDAnalysis AtomGroup containing all atoms H involved in the interaction X-H...Y. Has to be the same size as <code>xgrp</code> .
<code>ygrp:</code>	AtomGroup from MDAnalysis or None, optional MDAnalysis AtomGroup containing all atoms Y involved in the interaction X-H...Y. If <code>None</code> is given, it is assumed that Y=X (interaction X-H...X) and <code>xgrp</code> is taken as acceptor group. The default is <code>None</code> .
<code>cutoff_hy</code>	int or float Criterion for the H...Y distance in Å to define $h(t)$ . The criterion is fulfilled if the distance between a HY-pair is smaller than the value specified.
<code>cutoff_xy</code>	int or float Criterion for the X...Y distance in Å to define $H(t)$ . The criterion is fulfilled if the distance between a XY-pair is smaller than the value specified.
<code>angle_cutoff</code>	int or float Criterion for the angle $\alpha\angle XHY$ in in radian to define $h(t)$ . The cutoff is set so that if $\alpha > \text{angle\_cutoff}$ the criterion is fulfilled.

<code>nproc</code>	int, optional Number of processors available to parallelize the execution of the script. The default is 1.
<code>check_memory</code>	bool, optional Perform an approximate check if the amount of memory is sufficient. The default is <code>True</code> .

### Output:

For every donor  $i$  in `xgrp` a file `ct_i.dat` will be created. The file contains the results in three columns. The first column contains the timestep  $t$  in the same unit given in the option `timestep`. The second column contains  $\langle h(0)h(t) \rangle$  for that donor. The third column contains  $-\langle \dot{h}(0)[1 - h(t)]H(t) \rangle$  for that donor in inverse units of `timestep`. As long as the amount of acceptors (`ygrp`) is constant the data of multiple donors  $i$  can be averaged by computing the arithmetic mean of the desired quantity.

## 3.2 Example

Using a water simulation from the files of the module, we take the first 20 water molecules and calculate correlation functions using one of the hydrogen atoms as donor but all water oxygen atoms as donor. The timestep of this example trajectory is 0.2 ps. The hydrogen bond was here defined by a distance cutoff  $\text{H} \cdots \text{O}$  of 2.5 Å and an angle cutoff of  $\alpha > 2.27$  rad. For  $H(t)$  the distance criterion  $\text{X} \cdots \text{Y}$  was set to 3.5 Å.

---

```

1 import MDAnalysis
2 from mdorado.lifetime import calc_lifetime
3 from mdorado.data.datafilenames import water_topology,
   ↪ water_trajectory
4
5 universe = MDAnalysis.Universe(water_topology, water_trajectory)
6
7 xgrp = universe.select_atoms("name ow")[:20]
8 hgrp = universe.select_atoms("name hw")[:40:2]
9 ygrp = universe.select_atoms("name ow")
10
```



```
11 calc_lifetime(universe=universe, timestep=0.2, xgrp=xgrp, hgrp=hgrp,  
    ↪ ygrp=ygrp, cutoff_hy=2.5, angle_cutoff=2.27, cutoff_xy=3.5)
```

---

## 4 gofr

The radial distribution function  $g_{AB}(r)$  describes the density of particle B in a spherical shell of width  $dr$  at distance  $r$  around particle A in relation to the average number density of B  $\langle\rho_B\rangle$  in the system

$$g_{AB}(r) = \frac{\langle\rho_B(r)\rangle}{\langle\rho_B\rangle} = \frac{1}{\langle\rho_B\rangle \cdot N_A} \left\langle \sum_{i \in A} \sum_{j \in B} \frac{\delta(r_{ij} - r)}{4\pi r^2} \right\rangle. \quad (4.1)$$

Here,  $N_A$  and  $N_B$  references the number of particles A and B in the system, respectively. It should be noted that  $g_{AB}(r) = g_{BA}(r)$ .

From  $g_{AB}(r)$  and  $\langle\rho_B\rangle$  the average cumulative number of neighbors  $N_B(R)$  of particles B in a sphere of radius  $R$  around a particle A is obtainable via

$$N_B(R) = \rho_B \cdot 4\pi \int_0^R g_{AB}(r) r^2 dr \quad (4.2)$$

Similarly,  $N_A(R)$  can be computed using  $\langle\rho_A\rangle$ .

Three modes (mode) are implemented at the moment: **"site-site"**, **"cms-cms"**, and **"site-cms"**. The mode **"site-site"** computes the average  $g_{AB}(r)$  between all atoms A in **agrp** and all atoms B in **bgrp**. For example, if **agrp** contains atoms A0 and A1 while **bgrp** contains atoms B0 and B1, the pairs A0B0, A0B1, A1B0, and A1B1 will contribute to  $g_{AB}(r)$ .

The mode **"cms-cms"** can be used to compute center-of-mass radial distribution functions. It will calculate the center-of-mass of atoms belonging to the same molecule in **agrp** and **bgrp** and proceed to calculate the radial distribution function of these centers-of-mass. For example, given an **agrp** containing four atoms belonging to two different molecules (A0 and A1 belonging to molecule M0, A2 and A3 belonging to M1) and the same for **bgrp** (B0 and B1 belonging to M2, B2 and B3 belonging to M3) it will first calculate the centers-of-mass  $\text{cms}_{M0}(A0,A1)$ ,  $\text{cms}_{M1}(A2,A3)$ ,  $\text{cms}_{M2}(B0,B1)$ , and  $\text{cms}_{M3}(B2,B3)$ . The radial distribution function will then contain contributions from the pairs  $\text{cms}_{M0}\text{cms}_{M2}$ ,  $\text{cms}_{M0}\text{cms}_{M3}$ ,  $\text{cms}_{M1}\text{cms}_{M2}$ , and  $\text{cms}_{M1}\text{cms}_{M3}$ .

The mode `"site-cms"` is a mix of both of the functions described above, where `agrp` is taken atom-wise as in `gofr` and for `bgrp` the center-of-mass of atoms belonging to the same molecule is calculated as in `gofr_cms`.

## 4.1 Function

```
mdorado.gofr.Gofr(universe, agrp, bgrp, rmax, rmin=0, bins=100,  
    mode="site-site", outfilename="gofr.dat")
```

### Parameters:

<code>universe:</code>	MD.Analysis.Universe Universe containing the trajectory.
<code>agrp:</code>	AtomGroup from MDAnalysis AtomGroup containing all atoms A.
<code>bgrp:</code>	AtomGroup from MDAnalysis AtomGroup containing all atoms B.
<code>rmax:</code>	int or float The upper boundary of the A...B distance used for the $g(r)$ in units of Å.
<code>rmin:</code>	int or float, optional The lower boundary of the A...B distance used for the $g(r)$ in units of Å. The default is 0.
<code>bins:</code>	int or sequence of scalars or str, optional Specifies the number of points between <code>rmin</code> (included) and <code>rmax</code> (excluded). Will be used directly by <code>numpy.histogram</code> . From the numpy documentation: "If bins is an int, it defines the number of equal-width bins in the given range. If bins is a sequence, it defines a monotonically increasing array of bin edges, including the rightmost edge, allowing for non-uniform bin widths. If bins is a string, it defines the method used to calculate the optimal bin width, as defined by <code>histogram_bin_edges</code> ." The default is 100.
<code>mode:</code>	str, optional

Sets the mode for calculating different radial distribution functions: `"site-site"`, `"cms-cms"`, `"site-cms"`. If mode is set to `"site-site"`, the average radial distribution function of all sites in `agrp` to all sites in `bgrp` will be computed. The mode `"cms-cms"` will first compute the center-of-mass of sites belonging to the same molecule in `agrp` and `bgrp`, respectively, and then determine the radial distribution function between those centers of mass. The mode `"site-cms"` is a mix between the two, where every site in `agrp` is taken individually but for `bgrp` the center-of-mass of sites belonging to the same molecule is computed first. The default is `"site-site"`.

**outfile:** str, optional  
The name of the output file. The default is `"gofr.dat"`.

### Output:

The program creates a file named `outfile` with the distance  $r$  in Å (first column), the radial distribution function  $g_{AB}(r)$  (second column), the cumulative number of neighbors A in a sphere of radius  $r$  around particle B  $N_A(r)$  (third column), and the cumulative number of neighbors B in a sphere of radius  $r$  around particle A  $N_B(r)$  (fourth column).

### Class Methods:

**rdat:** Distance  $r$  (center of bins).  
**edges:** Edges of the bins.  
**hist:** Radial distribution function  $g_{AB}(r)$ .  
**annn:** Average number of neighbors A in a sphere of radius  $r$  around particle B  $N_A(r)$ .  
**bnnn:** Average number of neighbors B in a sphere of radius  $r$  around particle A  $N_B(r)$ .  
**avvol:** Average volume of the universe.  
**na:** Number of particles A in `agrp`. If mode is `"site-site"` or `"site-cms"`, `na` is the number of sites in `agrp`. If mode is `"cms-cms"`, `na` is the number of molecules (centers-of-mass) in `agrp`.

**nb:** Number of particles B in **bgrp**. If mode is **"site-site"**, na is the number of sites in **agrp**. If mode is **"site-cms"** or **"cms-cms"**, nb is the number of molecules (centers-of-mass) in **bgrp**.

## 4.2 Example

We start with a simulation of water from the data files shipped with the module, where all oxygen atoms are named **"ow"** and all hydrogen atoms **"hw"**. The rename of the water molecules is **"SOL"** for solvent. We will compute three different radial distribution functions to show reveal the differences in **"site-site"**, **"cms-cms"**, and **"site-cms"**: Firstly, we will use **"site-site"** to calculate the radial distribution function between all hydrogen and oxygen (H $\cdots$ O) atoms, which could for example be used to define the hydrogen bond O–H $\cdots$ O. Secondly, we will calculate the center-of-mass radial distribution function of all water molecules (cms $\cdots$ cms) using **"cms-cms"**. Thirdly, we use **"site-cms"** to compute the radial distribution function of all hydrogen atoms to the centers-of-mass of all water molecules (H $\cdots$ cms).

We first have to create a universe and select different AtomGroups to achieve the goals described above.

---

```
1 import MDAnalysis
2 from mdorado.gofr import Gofr
3 from mdorado.data.datafilenames import water_topology,
  ↪ water_trajectory
4
5 u = MDAnalysis.Universe(water_topology, water_trajectory)
6 hgrp = u.select_atoms("name hw")
7 ogrp = u.select_atoms("name ow")
8 watergrp = u.select_atoms("resname SOL")
9
10 sitesite = Gofr(universe=u, agrp=hgrp, bgrp=ogrp, rmin=1.0, rmax=6,
  ↪ bins=200, mode="site-site", outfilename="h_o.dat")
11 cmscms = Gofr(universe=u, agrp=watergrp, bgrp=watergrp, rmin=1.0,
  ↪ rmax=6, bins=200, mode="cms-cms", outfilename="cms_cms.dat")
12 sitecms = Gofr(universe=u, agrp=hgrp, bgrp=watergrp, rmin=1.0,
  ↪ rmax=6, bins=200, mode="site-cms", outfilename="h_cms.dat")
```

---

In Fig. 4.1 the three different  $g(r)$  are plotted. Additionally, we obtain the neighbour numbers  $N_A(r)$   $N_B(r)$  for each pair. In case of the  $\text{H}\cdots\text{O}$  distribution  $N_A(r)$  would be the average number of hydrogen atoms in a sphere of radius  $r$  around an oxygen atom. In case of the  $\text{cms}\cdots\text{cms}$  distribution  $N_A(r) = N_B(r)$  denotes the number of water molecules in a sphere of radius  $r$  around a water molecule. In case of the  $\text{H}\cdots\text{cms}$  distribution  $N_A(r)$  is the number of water molecules around in a sphere of radius  $r$  around a hydrogen atom.

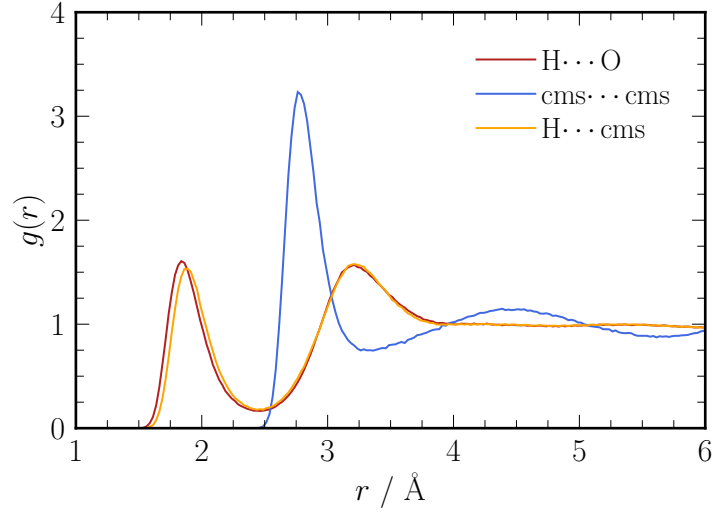


Fig. 4.1: Radial distribution functions obtained from the example above.

## 5 hb\_analyze

Geometric criteria can be used to define a hydrogen bond. Distance criteria can often be derived from pair-correlation functions but it may be required to include angular restrictions on the interaction. Twodimensional potentials of mean force (PMFs) can be used to obtain such criteria.<sup>[11,12]</sup> The PMF is calculated using the probability density of finding a donor-acceptor pair with the respective donor-acceptor ( $H \cdots Y$ ) distance  $r$  and angle  $\alpha$  ( $\alpha \angle XHY$ ).

This density can be derived from populations from equilibrium molecular dynamics trajectories. Therefore, donor-acceptor pairs with a distance  $r$  between `rmin` and `rmax` and an angle  $\cos(\alpha)$  between `cosalphamin` and `cosalphamax` will be counted in a twodimensional histogram according to the number of bins specified with the option `bins`. Each count is weighted with the respective  $r^{-2}$  to account for the growth of the spherical volume element with increasing  $r$ . The histogram is then normalized to the respective probability density function  $P(r, \cos(\alpha))$  using the area of each bin  $dr \cdot d\cos(\alpha)$  and the sum of all counts so that the integral over  $P$  is unity. At the end, the natural logarithm of  $P$  in each bin is calculated, due to the connection between  $P$  and the PMF via

$$F = -k_B T \ln(P) + c, \quad (5.1)$$

with the Boltzmann constant  $k_B$ , the temperature  $T$ , and an unknown constant  $c$ . The output is a twodimensional grid where each bin contains  $\ln(P)$  of the respective bin.

### 5.1 Function

```
mdorado.hb_analyze.hb_analyze(universe, xgrp, hgrp, rmax, ygrp=None,  
    rmin=0, cosalphamin=-1, cosalphamax=1, bins=50,  
    outfilename="hb_analyze.dat", ralphalist=False)
```

**Parameters:**

`universe:` MD.Analysis.Universe

Universe containing the trajectory.

**xgrp:** AtomGroup from MDAnalysis  
AtomGroup containing all atoms X involved in the interaction X-H...Y. Has to be the same size as **hgrp**.

**hgrp:** AtomGroup from MDAnalysis  
AtomGroup containing all atoms H involved in the interaction X-H...Y. Has to be the same size as **xgrp**.

**ygrp:** AtomGroup from MDAnalysis or None, optional  
MDAnalysis AtomGroup containing all atoms Y involved in the interaction X-H...Y. If **None** is given, it is assumed that Y=X (interaction X-H...X) and **xgrp** is taken as acceptor group. The default is **None**.

**rmax:** int or float  
The upper boundary of the H...Y distance in units of Å.

**rmin:** int or float, optional  
The lower boundary of the H...Y distance in units of Å. The default is 0.

**cosalphamin:** int or float, optional  
The lower boundary of  $\cos(\alpha)$  ( $\alpha\angle XHY$ ). The default is -1.

**cosalphamax:** int or float, optional  
The upper boundary of  $\cos(\alpha)$  ( $\alpha\angle XHY$ ). The default is 1.

**bins:** int or array\_like or [int, int] or [array, array], optional



Bins used for the 2D-histogram. Will be used directly by `numpy.histogram2d`. For two numbers the first will specify the bins of the  $H \cdots Y$  distance (`x_edges`) and the second will specify the bins of  $\cos(\alpha)$  (`y_edges`). The default is 50. Specifications:

- If `int`, the number of bins for the two dimensions (`nx=ny=bins`).
- If `array_like`, the bin edges for the two dimensions (`x_edges=y_edges=bins`).
- If `[int, int]`, the number of bins in each dimension (`nx, ny = bins`).
- If `[array, array]`, the bin edges in each dimension (`x_edges, y_edges = bins`).
- A combination `[int, array]` or `[array, int]`, where `int` is the number of bins and `array` is the bin edges.

**outfile:** str, optional

The name of the outputfile. The default is `hb_analyze.dat`.

**ralphalist:** bool, optional

Changes the output from the weighted probability density matrix to the list containing all the  $H \cdots Y$  distances and corresponding  $\cos(\alpha)$  from which the probability density is calculated. The default is `False`.

### Output:

The program creates a file named `outfile` with the weighted twodimensional histogram. The first axis represents the  $H \cdots Y$  distance and the second axis represents  $\cos(\alpha)$  ( $\alpha \angle XHY$ ).

If `ralphalist=True` the file contains the distances and corresponding angles of  $HY$ -pairs as a list: in the first column the distances are written in units of Å and the second column indicates the cosine of the corresponding angle  $\cos(\alpha)$ , both in the respective range `rmin` to `rmax` and `cosalphamin` to `cosalphamax`.

## 5.2 Example and Visualization

To use `hb_analyze` we first have to create a universe, define `xgrp` and `hgrp`, the range of the histogram, and the amount of bins in each dimension. If no `ygrp` is given the

program will use `xgrp` as acceptor group and analyze the interaction  $X-H \cdots X$  instead. Here an example for a water simulation from the datafiles of `mdorado` where the oxygen atoms are named `"ow"` and the hydrogen atoms `"hw"`:

---

```
1 import MDAnalysis
2 from mdorado.hb_analyze import hb_analyze
3 from mdorado.data.datafilenames import water_topology,
  ↪ water_trajectory
4
5 u = MDAnalysis.Universe(water_topology, water_trajectory)
6 xgrp = u.select_atoms("name ow")
7 hgrp = u.select_atoms("name hw")[:,2]
8
9 hb_analyze(universe=u, xgrp=xgrp, hgrp=hgrp, rmin=1.5, rmax=5,
  ↪ cosalphamin=-1, cosalphamax=1, bins=50)
```

---

After execution a file `hb_analyze.dat` (changable by the option `outfilename`) can be found in the current folder. It contains the  $50 \times 50$  (bins) matrix of the weighted probability function. This matrix can be plotted by matplotlibs `contour` and similar programs. Here an example using matplotlibs `contourf`:

---

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from matplotlib import ticker
4 import matplotlib.colors as col
5
6 lowcolor = '#ffffff'
7 midcolor1 = '#6090f0'
8 midcolor2 = '#30f050'
9 midcolor3 = '#f0f000'
10 midcolor4 = '#f06000'
11 highcolor = '#b02000'
12 cmapown = col.LinearSegmentedColormap.from_list('own',
  ↪ [lowcolor,midcolor1,midcolor2,midcolor3,midcolor4,highcolor])
13 cmapown.set_over('#9e1c00')
14
```

---

```

15 rmin=1.5
16 rmax=5
17 cosalphamin=-1
18 cosalphamax=1
19
20 fig, ax = plt.subplots()
21 histo_matrix = np.loadtxt("hb_analyze.dat")
22 levels = ticker.MaxNLocator(nbins=60).tick_values(-3, 2)
23 cax = ax.contourf(histo_matrix, extent=(cosalphamin, cosalphamax,
    ↪ rmin, rmax), levels=levels, extend='both', cmap=cmapown)
24 plt.xlabel('$\\cos(\\alpha)$')
25 plt.ylabel('$r$ / $\\AA$')
26 plt.axis([cosalphamin, cosalphamax, rmin, rmax])
27 cbar = fig.colorbar(cax, ticks=[-3, -2, -1, 0, 1, 2])
28 cbar.ax.set_ylabel('$\\log[W(\\cos(\\alpha), r)]$')
29 plt.tight_layout()
30 plt.savefig("histo.pdf")
31 plt.clf()

```

---

After importing the necessary modules, we first define our own colormap `cmapown` (line 6 to 13). Standard colormaps can be found here. The output of `hb_analyze` only contains the weighted probability densities for each bin and not their position, so we have to tell the program in line 15–18 in which range the histogram is plotted (option `extent` of `contourf` line 23 and  $x$ - and  $y$ -axis limits line 26).

The actual plotting happens onwards from line 20. Using numpy's `loadtxt` we load the histogram matrix into the array `histo_matrix` (line 21). In line 22 we define the amount of bins (`nbis=60`) and the range  $(-3, 2)$  of the coloraxis. The array can directly be processed by `contourf` where we also input the range, levels, and colormap. The option `extend='both'` enables the colors beyond the levels defined before (arrows above and below the coloraxis). The lines 24 to 28 are defining the axis-ticks and -labels. After that the plot is already finished and can be saved or shown directly. An example plot for a small watersimulation is shown in Fig. 5.1.

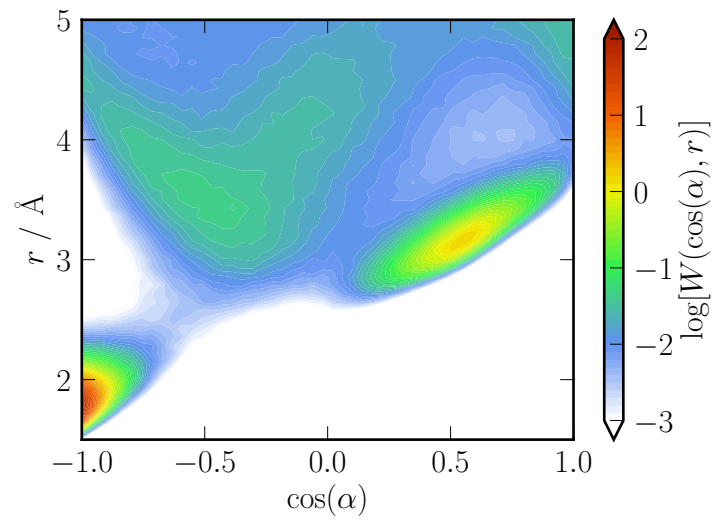


Fig. 5.1: Example plot of a twodimensional histogram computed with [hb\\_analyze](#).

## 6 Mean Square Displacement

First, to compute the mean square displacement (MSD) of a particle the trajectory has to be “unwrapped”, so that the effects of the periodic boundary conditions (PBCs) are reversed. The function `mdorado.msd.unwrap` is able to accomplish this task for cuboid boxes (all box angles are  $90^\circ$ ).

Given a trajectory (`universe`) AtomGroup (`agrp`) the positions of the atoms of interest are recomputed so that each atom starts in the origin of the coordinate system at  $t = 0$ . The displacement in each time step is then added incrementally to obtain a trajectory relative to this starting position. If the magnitude of the displacement in one dimension (x, y or z) is larger than half a box length in that dimension, indicating a jump of the atom due to PBCs, the displacement is adjusted by adding or subtracting the box length to ensure “jump-free” movement.

From these coordinates `mdorado.msd.msd` is able to compute the MSD using an efficient algorithm<sup>[13]</sup>

$$\text{MSD}(t) = \langle |\mathbf{r}(t) - \mathbf{r}(0)|^2 \rangle, \quad (6.1)$$

where  $\mathbf{r}(t)$  denotes a positional vector of the unwrapped trajectory.

### 6.1 Functions

```
mdorado.msd.unwrap(universe, agrp, dimensionskey="xyz", cms=False)
```

#### Parameters:

<code>universe:</code>	MD.Analysis.Universe Universe containing the trajectory.
<code>agrp:</code>	AtomGroup from MDAnalysis AtomGroup containing all atoms for which the trajectory should be unwrapped.
<code>dimensionskey:</code>	str, optional

Dimensions in which the trajectory is unwrapped. The keywords are:

- "xyz" for all dimensions
- "x", "y", and "z" for one of the three principal box axes
- "xy", "xz", and "yz" for a combination of two of the three box axes

Changes the shape of the output array. The default value is "xyz".

`cms:` bool, optional

If `cms=True` the program computes the movement of the center-of-mass of atoms belonging to the same molecule in `agrp`. If, for example, `agrp` would contain all atoms of water molecules in the simulation this option allows the calculation of the average center-of-mass MSD of these water molecules. If for the same case `cms=False` is chosen, the program calculates the MSD of all the various atoms individually, averaging over hydrogen as well as oxygen atoms. The default value is `False`.

**Returns:** ndarray

An ndarray containing the unwrapped positions. The shape of the array is  $(N_A, N_{\text{dim}}, N_{\text{steps}})$ , where  $N_A$  denotes the number of atoms in `agrp` (or the number of molecules if `cms=True`),  $N_{\text{dim}}$  denotes the number of dimensions according to the options `dimensionskey`, and  $N_{\text{steps}}$  is the number of timesteps in the `universe`.

```
mdorado.msd.msd(positions, dt, outfilename="msd.dat")
```

**Parameters:**

`positions:` ndarray  
ndarray of shape  $(N_A, N_{\text{dim}}, N_{\text{steps}})$  containing an unwrapped trajectory, where  $N_A$  denotes the number of particles,  $N_{\text{dim}}$  denotes the number of dimensions, and  $N_{\text{steps}}$  denotes the number of steps (see `mdorado.msd.unwrap`).

`dt:` int or float  
Difference in time between two configurations in `positions`.

outfilename: str, optional  
The name of the outputfile. The default is `"msd.dat"`.

### Output:

The function writes a file with two columns, where the first column is the time delay  $t$  and the second column is the average MSD for that time delay  $\text{MSD}(t) = \langle |\mathbf{r}(t) - \mathbf{r}(0)|^2 \rangle$

## 6.2 Example

We are going to compute two slightly different MSDs from the example water trajectory. First, we are interested in the movement of the center of mass of the water molecules. Therefore, we create an atom group with all oxygen and hydrogen atoms (`solgrp`). The `msd.unwrap` function repairs the jumps due to the periodic boundary conditions ("unwrapping"). Because we are interested in the 3D movement of the center-of-mass, we set the option `dimensionskey` to `"xyz"` and the option `cms` to `"True"` (line 10). The function automatically computes the center-of-mass of atoms belonging to the same molecule and returns the trajectory of these centers in the ndarray `cmspos` of shape  $(128, 3, 2501)$   $[(N_A, N_{\text{dim}}, N_{\text{steps}})]$ . This array can be used as input for the `msd.msd` function along with the timestep (`dt`) to calculate the average MSD of these centers-of-mass.

The second example computes the average MSD of all hydrogen atoms in the trajectory. Therefore, we create an atom group containing the hydrogen atoms (`hgrp`) and unwrap their trajectory with `cms` set to `False` (default) (line 13). The array `hpos` is of shape  $(256, 3, 2501)$  (two hydrogen atoms per water molecule) and can again be used as an input for the `msd.msd` function to compute the MSD.

---

```
1 import MDAnalysis
2 from mdorado import msd
3 from mdorado.data.datafilenames import water_topology,
   ↪ water_trajectory
4
5 u = MDAnalysis.Universe(water_topology, water_trajectory)
6 solgrp = u.select_atoms("resname SOL")
7 hgrp = u.select_atoms("name hw")
8 dt = 0.2
9
```

```
10 cmspos = msd.unwrap(universe=u, agrp=solgrp, dimensionskey="xyz",  
    ↪  cms=True)  
11 msd.msd(positions=cmspos, dt=dt, outfilename="msd_cms.dat")  
12  
13 hpos = msd.unwrap(universe=u, agrp=hgrp, dimensionskey="xyz",  
    ↪  cms=False)  
14 msd.msd(positions=hpos, dt=dt, outfilename="msd_h.dat")
```

---



# Bibliography

- [1] D. Chandler, *J. Chem. Phys.* **1978**, *68*, 2959.
- [2] C. H. Bennett in *Algorithms for Chemical Computations*, Chapter 4, p. 63.
- [3] S. Gehrke, M. von Domaros, R. Clark, O. Hollóczki, M. Brehm, T. Welton, A. Luzar, B. Kirchner, *Faraday Discuss.* **2018**, *206*, 219.
- [4] S. Gehrke, B. Kirchner, *J. Chem. Eng. Data* **2019**, *65*, 1146.
- [5] J. Neumann, D. Paschek, A. Strate, R. Ludwig, *J. Phys. Chem. B* **2021**, *125*, 281.
- [6] J. Neumann, R. Ludwig, D. Paschek, *J. Phys. Chem. B* **2021**, *125*, 5132.
- [7] A. Luzar, D. Chandler, *Phys. Rev. Lett.* **1996**, *76*, 928.
- [8] A. Luzar, D. Chandler, *Nature* **1996**, *379*, 55.
- [9] A. Luzar, *J. Chem. Phys.* **2000**, *113*, 10663.
- [10] J. Busch, J. Neumann, D. Paschek, *J. Chem. Phys.* **2021**, *154*, 214501.
- [11] R. Kumar, J. R. Schmidt, J. L. Skinner, *J. Chem. Phys.* **2007**, *126*, 204107.
- [12] D. Trzesniak, A.-P. E. Kunz, W. F. van Gunsteren, *ChemPhysChem* **2007**, *8*, 162.
- [13] Calandrini, V., Pellegrini, E., Calligari, P., Hinsén, K., Kneller, G.R., *JDN* **2011**, *12*, 201.