MAICoS Documentation

see the file AUTHORS for the full list of names

Mar 14, 2024

CONTENTS

This documentation covers everything you need to know about MAICoS, the Molecular Analysis for Interfacial and Confined Systems toolkit. There are five sections:

- *Getting started*
- *How-to guides*
- *Reference guides*
- *Explanations*
- *Developer documentation*

If you are new to MAICoS, we recommend starting with the *Getting started* section. If you want to contribute to the development of the library, please have a look at our *developer documentation*.

CHAPTER

ONE

GETTING STARTED

The *Getting started* section is for MAICoS beginners. It will help you install and familiarize yourself with MAICoS and its approach to analyse molecular dynamics simulations.

CHAPTER

TWO

HOW-TO GUIDES

The *How-to guides* section is for MAICoS intermediate and advanced users. It contains guides taking you through series of steps involved in addressing key problems and use-cases in MAICoS.

CHAPTER

THREE

REFERENCE GUIDES

The *Reference guides* section is for MAICoS intermediate and advanced users, it contains technical references and parameter list for each MAICoS modules (*How-to guides*) as well as the APIs. It describes the various functionalities provided by MAICoS. You can always refer to this section to learn more about classes, functions, modules, and other aspects of MAICoS machinery you may come across.

CHAPTER

FOUR

EXPLANATIONS

The *Explanations* section discusses key topics and concepts at a fairly high level and provides explanations to expand your knowledge of MAICoS. It requires at least basic to intermediate knowledge of MAICoS.

CHAPTER

FIVE

DEVELOPER DOCUMENTATION

The *Developer documentation* helps you contributing to the code base or the documentation of MAICoS.

5.1 Getting started

This sections describes MAICoS, how to install it, and its most basic commands.

5.1.1 What is MAICoS

MAICoS is the acronym for Molecular Analysis for Interfacial and Confined Systems. It is an object-oriented python toolkit for analysing the structure and dynamics of interfacial and confined fluids from molecular simulations. Combined with MDAnalysis, MAICoS can be used to extract density profiles, dielectric constants, structure factors, or transport properties from trajectories files, including LAMMPS, GROMACS, CHARMM or NAMD data. MAICoS is open source and is released under the GNU general public license v3.0.

MAICoS is a tool for beginners of molecular simulations with no prior Python experience. For these users MAICoS provides a descriptive command line interface. Also experienced users can use the Python API for their day to day analysis or use the provided infrastructure to build their own analysis for interfacial and confined systems.

Keep up to date with MAICoS news by following us on Twitter. If you find an issue, you can report it on Gitlab. You can also join the developer team on Discord to discuss possible improvements and usages of MAICoS.

Currently, MAICoS supports the following analysis modules in alphabetical order:

Module Name	Description
DensityCylinder	Compute cylindrical partial densitiy profiles
DensityPlanar	Compute cartesian partial density profiles
DensitySphere	Compute spherical partial density profiles
DielectricCylinder	Compute cylindrical dielectric profiles
DielectricPlanar	Compute planar dielectric profiles
DielectricSpectrum	Compute the linear dielectric spectrum
DielectricSphere	Compute spherical dielectric profiles
DipoleAngle	Compute angle timeseries of dipole moments
DiporderCylinder	Compute cylindrical dipolar order parameters
DiporderPlanar	Compute planar dipolar order parameters
RDFDiporder	Spherical Radial Distribution function between dipoles
DiporderSphere	Compute spherical dipolar order parameters
DiporderStructureFactor	Structure factor for dipoles
KineticEnergy	Compute the timeseries of energies
PDFCylinder	Compute cylindrical shell-wise 1D pair distribution functions
PDFPlanar	Compute slab-wise planar 2D pair distribution functions
Saxs	Compute small angle X-Ray structure factors and scattering intensities (SAXS)
TemperaturePlanar	Compute temperature profiles in a cartesian geometry
VelocityCylinder	Compute the cartesian velocity profile across a cylinder
VelocityPlanar	Compute the velocity profile in a cartesian geometry

5.1.2 Installation

Install MAICoS using pip with:

```
pip install maicos
```

or using conda with:

```
conda install -c conda-forge maicos
```

5.1.3 Usage - Python interpreter

To follow this tutorial, it is assumed that MAICoS has been *installed* on your computer.

MAICoS heavily depends on the MDAnalysis infrastructure for trajectory loading and atom selection. Here we will only cover a small aspects of the capabilities of MDAnalysis. If you want to learn more about the library, take a look at their documentation.

Only three MAICoS analysis modules are used in this tutorial *`maicos.DensityPlanar`*, *`maicos.VelocityPlanar`* and *`maicos.DiporderPlanar`* but all modules follow the same structure:

1. load your simulation data into an `MDAnalysis.core.universe.Universe`
2. define analysis parameters like bin width or the direction of the analysis
3. after the analysis was succesful, access all results in a `MDAnalysis.analysis.base.Results` of the analysis object.

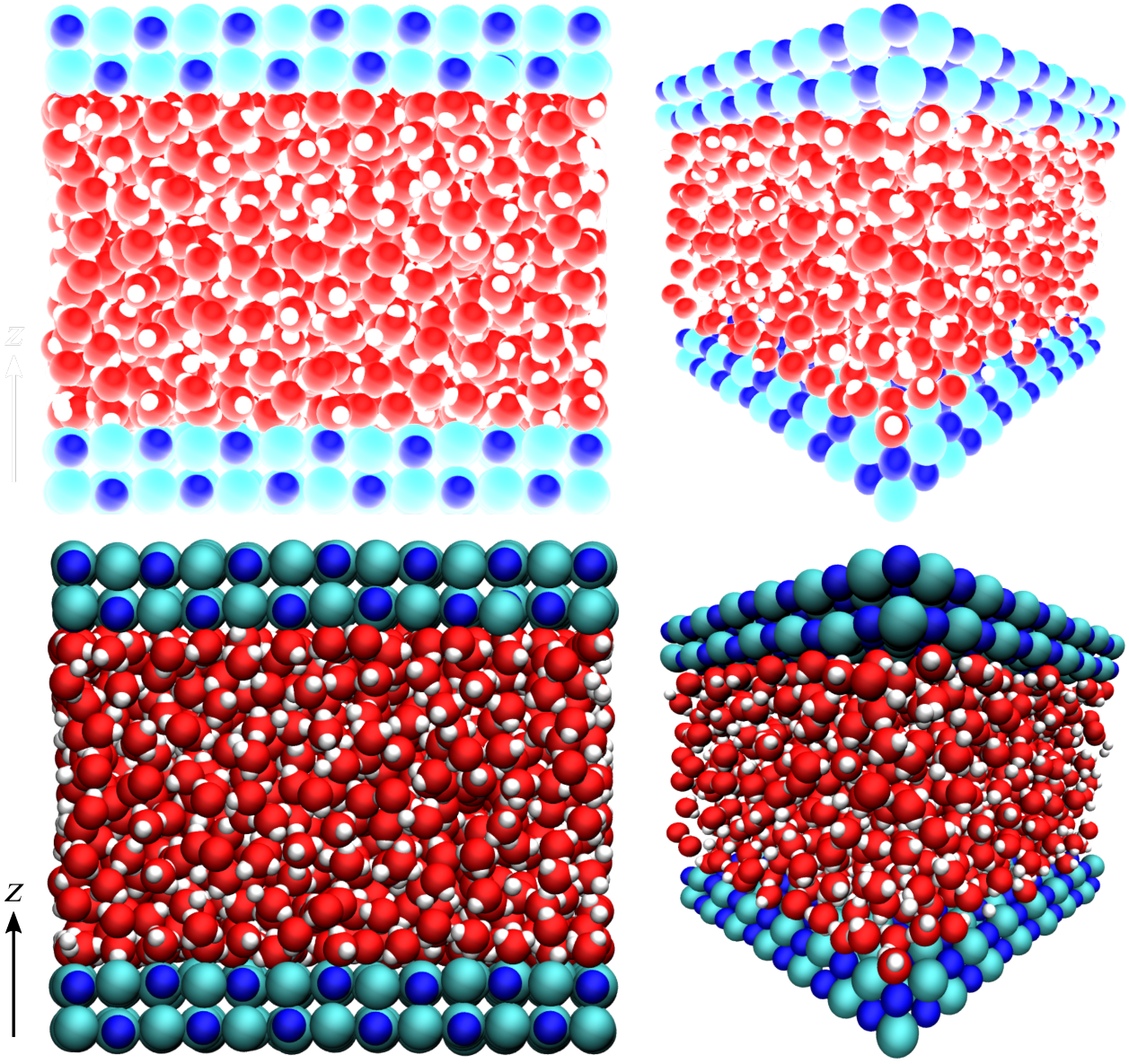
Note that some of the calculations may contain pitfall, such as dielectric profiles calculation. Potential pitfalls and best practices are listed in the *How-to guides* section.

To start, let us first import Matplotlib, MDAnalysis and MAICoS

```
import matplotlib.pyplot as plt
import MDAnalysis as mda

import maicos
```

Load Simulation Data

For this tutorial we use a system consisting of a 2D slab with 1176 water molecules confined in a 2D slit made of NaCl atoms, where the two water/solid interfaces are normal to the axis z as shown in the snapshot below:



An acceleration $a = 0.05\,\mathrm{nm\,ps^{-2}}$ was applied to the water molecules in the $\boldsymbol{e}_x$ direction parallel to the NaCl wall, and the atoms of the wall were maintained frozen along $\boldsymbol{e}_x$.

We first create an `MDAnalysis.core.universe.Universe` by loading a topology and trajectory from disk. You can

download the `topology` and the `trajectory` from our website.

```
u = mda.Universe("slit_flow.tpr", "slit_flow.trr")
```

Let us print a few information about the trajectory:

```
print(f"Number of frames in the trajectory is {u.trajectory.n_frames}.")
timestep = round(u.trajectory.dt, 2)
print(f"Time interval between two frames is {timestep} ps.")
total_time = round(u.trajectory.totaltime, 2)
print(f"Total simulation time is {total_time} ps.")
```

```
Number of frames in the trajectory is 201.
Time interval between two frames is 10.0 ps.
Total simulation time is 2000.0 ps.
```

Now, we define four atom groups containing repectively:

1. the oxygen and the hydrogen atoms (of the water molecules),
2. the oxygen atoms (of the water molecules),
3. the hydrogen atoms (of the water molecules),
4. the Na and Cl atoms (of the wall):

```
group_H2O = u.select_atoms("type OW HW")
group_O = u.select_atoms("type OW")
group_H = u.select_atoms("type HW")
group_NaCl = u.select_atoms("type SOD CLA")
```

Let us print a few information about the groups

```
print(f"Number of water molecules is {group_O.n_atoms}.")
print(f"Number of NaCl atoms is {group_NaCl.n_atoms}.")
```

```
Number of water molecules is 1176.
Number of NaCl atoms is 784.
```

Density Profiles

Let us use the *maicos.DensityPlanar* class to extract the density profile of the `group_H2O` along the (default) z axis by running the analysis:

```
dplan = maicos.DensityPlanar(group_H2O).run()
```

```
Unwrapping in combination with the `wrap_compound='atoms` is superfluous. `unwrap` will␣
↪be set to `False`.
```

The warning starting with *Unwrapping* is perfectly normal and can be ignored for now.

Let us extract the bin coordinates z and the averaged density profile from the `results` attribute:

```
zcoor = dplan.results.bin_pos
dens = dplan.results.profile
```

The density profile is given as a 1D array, let us look at the 10 first lines:

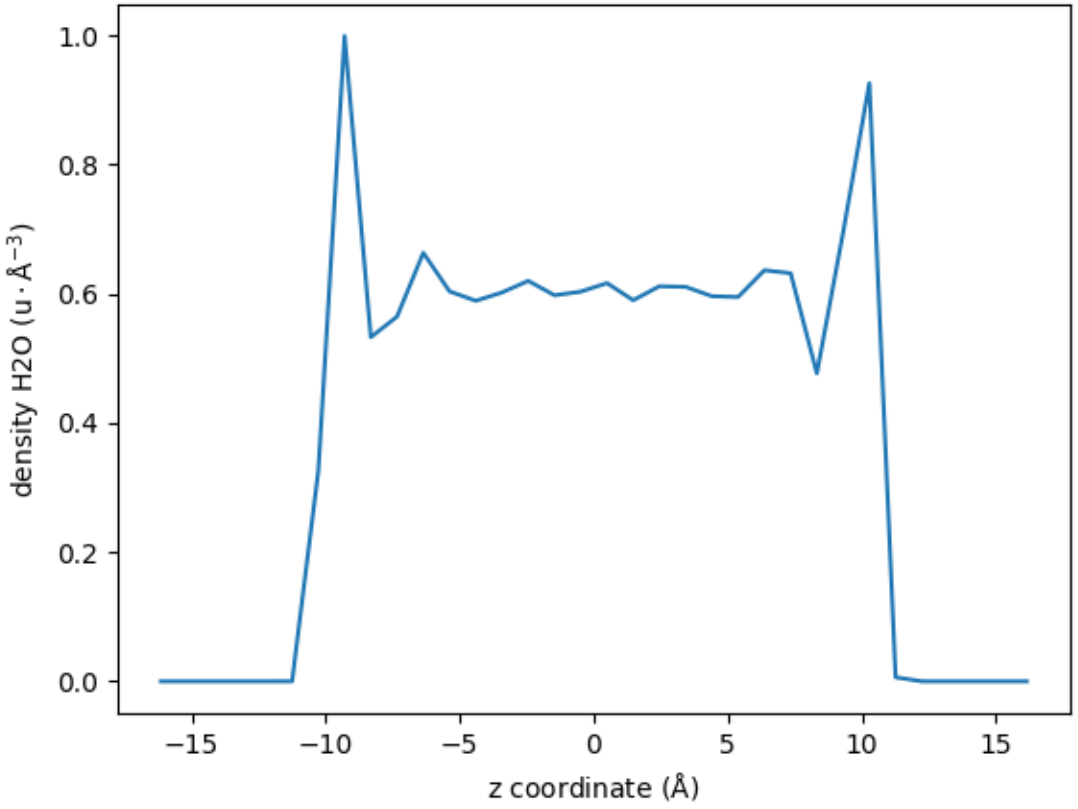
```
print(dens[:10])
```

```
[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 2.91778317e-05 3.26228271e-01 9.99460378e-01
 5.32837117e-01 5.64829859e-01]
```

By default the `bin_width` is 1 Å, and the unit is atomic mass per ^{3} (u/Å^3).

Let us plot the density profile using Matplotlib:

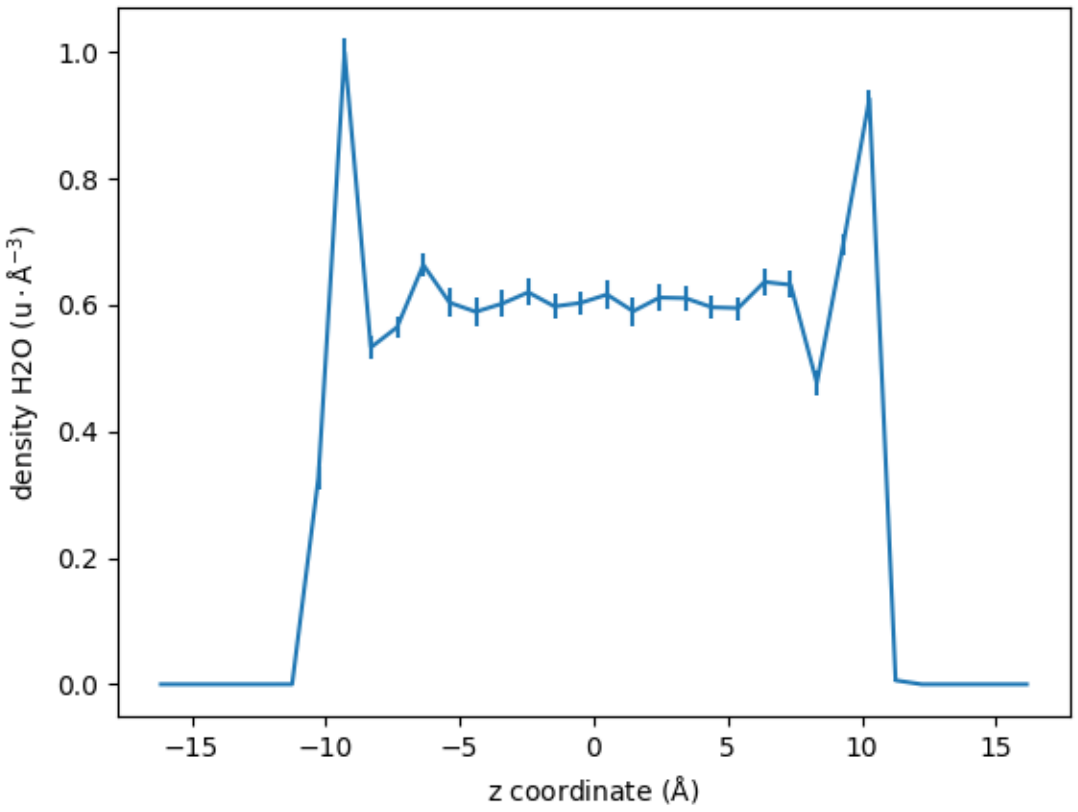
```
fig, ax = plt.subplots()

ax.plot(zcoor, dens)

ax.set_xlabel(r"z coordinate ($\rm Å$)")
ax.set_ylabel(r"density H2O ($\rm u \cdot Å^{-3}$)")

fig.show()
```



Uncertainity estimates

MAICoS estimates the uncertainity for each profile. This uncertainity is stored inside the *dprofile* attribute.

```
uncertainity = dplan.results.dprofile

# Let us plot the results also showing the uncertainities

fig, ax = plt.subplots()

ax.errorbar(zcoor, dens, 5 * uncertainity)

ax.set_xlabel(r"z coordinate ($\rm Å$)")
ax.set_ylabel(r"density H2O ($\rm u \cdot Å^{-3}$)")

fig.show()
```



For this example we scale the error by 5 to be visible in the plot.

The uncertainity estimatation assumes that the trajectory data is uncorraleted. If the correlation time is too high or not reasonably computable a warning occurs that the uncertainity estimatation might be unreasonable.

```
maicos.DensityPlanar(group_H2O).run(start=10, stop=13, step=1)
```

```
Unwrapping in combination with the `wrap_compound='atoms` is superfluous. `unwrap` will␣
↪be set to `False`.
/home/docs/checkouts/readthedocs.org/user_builds/maicos/envs/main/lib/python3.9/site-
↪packages/maicos/core/base.py:456: UserWarning: Your trajectory is too short to␣
↪estimate a correlation time. Use the calculated error estimates with caution.
  self.corrtime = correlation_analysis(self.timeseries)

<maicos.modules.densityplanar.DensityPlanar object at 0x7fad7bc26430>
```

Improving the Results

By changing the value of the default parameters, one can improve the results, and perform more advanced operations.

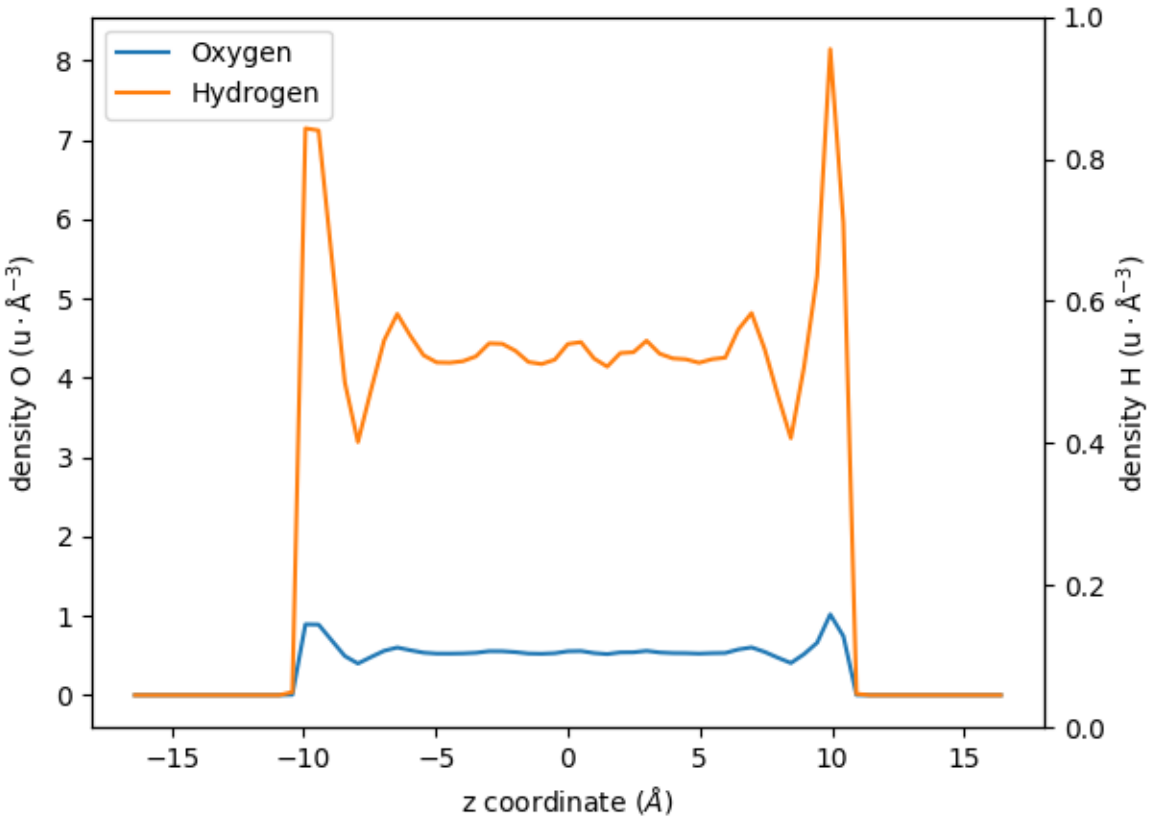
Let us increase the spatial resolution by reducing the `bin_width`, and extract two profiles instead of one:

- one for the oxygen atoms of the water molecules,
- one from the hydrogen atoms:

```
dplan_smaller_bin = []
for ag in [group_O, group_H]:
    dplan_smaller_bin.append(
        maicos.DensityPlanar(ag, bin_width=0.5, unwrap=False).run()
    )

# TODO: Intoduce AnalysisCollection here?

zcoor_smaller_bin_O = dplan_smaller_bin[0].results.bin_pos
dens_smaller_bin_O = dplan_smaller_bin[0].results.profile

zcoor_smaller_bin_H = dplan_smaller_bin[0].results.bin_pos
dens_smaller_bin_H = dplan_smaller_bin[0].results.profile
```

Let us plot the results using two differents y-axis:

```
fig, ax1 = plt.subplots()

ax1.plot(zcoor_smaller_bin_O, dens_smaller_bin_O, label=r"Oxygen")
ax1.plot(zcoor_smaller_bin_H, dens_smaller_bin_H * 8, label=r"Hydrogen")

ax1.set_xlabel(r"z coordinate ($Å$)")
ax1.set_ylabel(r"density O ($\rm u \cdot Å^{-3}$)")

ax2 = ax1.twinx()
ax2.set_ylabel(r"density H ($\rm u \cdot Å^{-3}$)")
ax1.legend()

fig.show()
```



Access to all the Module's Options

For each MAICoS module, they are several parameters similar to `bin_width`. The parameter list and default options are listed in the *module's documentation*, and can be gathered by calling the help function of Python:

```
help(maicos.DensityPlanar)
```

```
Help on class DensityPlanar in module maicos.modules.densityplanar:

class DensityPlanar(maicos.core.planar.ProfilePlanarBase)
 |  DensityPlanar(atomgroup: MDAnalysis.core.groups.AtomGroup, dens: str = 'mass', dim:␣
↪int = 2, zmin: Optional[float] = None, zmax: Optional[float] = None, bin_width: float␣
↪= 1, refgroup: Optional[MDAnalysis.core.groups.AtomGroup] = None, sym: bool = False,␣
↪grouping: str = 'atoms', unwrap: bool = True, bin_method: str = 'com', output: str =
↪'density.dat', concfreq: int = 0, jitter: float = 0.0) -> None
 |
 |  Cartesian partial density profiles.
 |
 |  Calculations are carried out for
 |  ``mass`` :math:`(\rm u \cdot Å^{-3})`, ``number`` :math:`(\rm Å^{-3})` or ``charge``
 |  :math:`(\rm e \cdot Å^{-3})` density profiles along certain cartesian axes ``[x, y,
 |  z]`` of the simulation cell. Cell dimensions are allowed to fluctuate in time.
```

(continues on next page)

(continued from previous page)

```
|
|  For grouping with respect to ``molecules``, ``residues`` etc., the corresponding
|  centers (i.e., center of mass), taking into account periodic boundary conditions,
|  are calculated. For these calculations molecules will be unwrapped/made whole.
|  Trajectories containing already whole molecules can be run with ``unwrap=False`` to
|  gain a speedup. For grouping with respect to atoms, the `unwrap` option is always
|  ignored.
|
|  For the correlation analysis the central bin
|  (:math:`N \backslash 2`) of the 0th's group profile is used. For further information␣
↪on the correlation analysis please
|  refer to :class:`maicos.core.base.AnalysisBase` or the :ref:`general-design`
|  section.
|
|  Parameters
|  ----------
|  atomgroup : MDAnalysis.core.groups.AtomGroup
|      A :class:`~MDAnalysis.core.groups.AtomGroup` for which the calculations are
|      performed.
|  unwrap : bool
|      When :obj:`True`, molecules that are broken due to the periodic boundary
|      conditions are made whole.
|
|      If the input contains molecules that are already whole, speed up the calculation
|      by disabling unwrap. To do so, use the flag ``-no-unwrap`` when using MAICoS
|      from the command line, or use ``unwrap=False`` when using MAICoS from the Python
|      interpreter.
|
|      Note: Molecules containing virtual sites (e.g. TIP4P water models) are not
|      currently supported in MDAnalysis. In this case, you need to provide unwrapped
|      trajectory files directly, and disable unwrap. Trajectories can be unwrapped,
|      for example, using the ``trjconv`` command of GROMACS.
|  refgroup : MDAnalysis.core.groups.AtomGroup
|      Reference :class:`~MDAnalysis.core.groups.AtomGroup` used for the calculation.
|      If ``refgroup`` is provided, the calculation is performed relative to the center
|      of mass of the AtomGroup. If ``refgroup`` is :obj:`None` the calculations are
|      performed with respect to the center of the (changing) box.
|  jitter : float
|      Magnitude of the random noise to add to the atomic positions.
|
|      A jitter can be used to stabilize the aliasing effects sometimes appearing when
|      histogramming data. The jitter value should be about the precision of the
|      trajectory. In that case, using jitter will not alter the results of the
|      histogram. If ``jitter = 0.0`` (default), the original atomic positions are kept
|      unchanged.
|
|      You can estimate the precision of the positions in your trajectory with
|      :func:`maicos.lib.util.trajectory_precision`. Note that if the precision is not
|      the same for all frames, the smallest precision should be used.
|  concfreq : int
|      When concfreq (for conclude frequency) is larger than ``0``, the conclude
|      function is called and the output files are written every ``concfreq``
```

(continues on next page)

(continued from previous page)

```
|      frames.
|  dim : {0, 1, 2}
|      Dimension for binning (``x=0``, ``y=1``, ``z=1``).
|  zmin : float
|      Minimal coordinate for evaluation (in Å) with respect to the center of mass of
|      the refgroup.
|
|      If ``zmin=None``, all coordinates down to the lower cell boundary are taken into
|      account.
|  zmax : float
|      Maximal coordinate for evaluation (in Å) with respect to the center of mass of
|      the refgroup.
|
|      If ``zmax = None``, all coordinates up to the upper cell boundary are taken into
|      account.
|  bin_width : float
|      Width of the bins (in Å).
|  sym : bool
|      Symmetrize the profile. Only works in combination with
|      ``refgroup``.
|  grouping : {``"atoms"``, ``"residues"``, ``"segments"``, ``"molecules"``, ``
↪"fragments"``}
|      Atom grouping for the calculations.
|
|      The possible grouping options are the atom positions (in the case where
|      ``grouping="atoms"``) or the center of mass of the specified grouping unit (in
|      the case where ``grouping="residues"``, ``"segments"``, ``"molecules"`` or
|      ``"fragments"``).
|  bin_method : {``"com"``, ``"cog"``, ``"coc"``}
|      Method for the position binning.
|
|      The possible options are center of mass (``"com"``),
|      center of geometry (``"cog"``), and center of charge (``"coc"``).
|  output : str
|      Output filename.
|  dens : {``"mass"``, ``"number"``, ``"charge"``}
|      density type to be calculated.
|
|  Attributes
|  ----------
|  results.bin_pos : numpy.ndarray
|      Bin positions (in Å) ranging from ``zmin`` to ``zmax``.
|  results.profile : numpy.ndarray
|      Calculated profile.
|  results.dprofile : numpy.ndarray
|      Estimated profile's uncertainity.
|
|  Notes
|  -----
|  Partial mass density profiles can be used to calculate the ideal component of the
|  chemical potential. For details, take a look at the corresponding :ref:`How-to
|  guide<howto-chemical-potential>`.
```

(continues on next page)

(continued from previous page)

```
 |
 |  Method resolution order:
 |      DensityPlanar
 |      maicos.core.planar.ProfilePlanarBase
 |      maicos.core.planar.PlanarBase
 |      maicos.core.base.AnalysisBase
 |      maicos.core.base._Runner
 |      MDAnalysis.analysis.base.AnalysisBase
 |      maicos.core.base.ProfileBase
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, atomgroup: MDAnalysis.core.groups.AtomGroup, dens: str = 'mass', dim:␣
↪int = 2, zmin: Optional[float] = None, zmax: Optional[float] = None, bin_width: float␣
↪= 1, refgroup: Optional[MDAnalysis.core.groups.AtomGroup] = None, sym: bool = False,␣
↪grouping: str = 'atoms', unwrap: bool = True, bin_method: str = 'com', output: str =
↪'density.dat', concfreq: int = 0, jitter: float = 0.0) -> None
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Readonly properties inherited from maicos.core.planar.PlanarBase:
 |
 |  odims
 |      Other dimensions perpendicular to dim i.e. (0,2) if dim = 1.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from maicos.core.base.AnalysisBase:
 |
 |  run(self, start: Optional[int] = None, stop: Optional[int] = None, step:␣
↪Optional[int] = None, frames: Optional[int] = None, verbose: Optional[bool] = None,␣
↪progressbar_kwargs: Optional[dict] = None) -> typing_extensions.Self
 |      Iterate over the trajectory.
 |
 |  savetxt(self, fname: str, X: numpy.ndarray, columns: Optional[List[str]] = None) ->␣
↪None
 |      Save to text.
 |
 |      An extension of the numpy savetxt function. Adds the command line input to the
 |      header and checks for a doubled defined filesuffix.
 |
 |      Return a header for the text file to save the data to. This method builds a
 |      generic header that can be used by any MAICoS module. It is called by the save
 |      method of each module.
 |
 |      The information it collects is:
 |        - timestamp of the analysis
 |        - name of the module
 |        - version of MAICoS that was used
 |        - command line arguments that were used to run the module
 |        - module call including the default arguments
 |        - number of frames that were analyzed
```

(continues on next page)

(continued from previous page)

```
|        - atomgroup that was analyzed
|        - output messages from modules and base classes (if they exist)
|
|  ----------------------------------------------------------------------
|  Readonly properties inherited from maicos.core.base.AnalysisBase:
|
|  box_center
|      Center of the simulation cell.
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from maicos.core.base._Runner:
|
|  __dict__
|      dictionary for instance variables (if defined)
|
|  __weakref__
|      list of weak references to the object (if defined)
|
|  ----------------------------------------------------------------------
|  Methods inherited from maicos.core.base.ProfileBase:
|
|  save(self) -> None
|      Save results of analysis to file specified by ``output``.
```

Here we can see that for *maicos.DensityPlanar*, there are several possible options such as `zmin`, `zmax` (the minimal and maximal coordinates to consider), or `refgroup` (to perform the binning with respect to the center of mass of a certain group of atoms).

Knowing this, let us re-calculate the density profile of H_2O, but this time using the group `group_H2O` as a reference for the center of mass:

```
dplan_centered_H2O = maicos.DensityPlanar(
    group_H2O, bin_width=0.5, refgroup=group_H2O, unwrap=False
)
dplan_centered_H2O.run()
zcoor_centered_H2O = dplan_centered_H2O.results.bin_pos
dens_centered_H2O = dplan_centered_H2O.results.profile
```

Let us also extract the density profile for the NaCl walls, but centered with respect to the center of mass of the H_2O group:

```
dplan_centered_NaCl = maicos.DensityPlanar(
    group_NaCl, bin_width=0.5, refgroup=group_H2O, unwrap=False
)
dplan_centered_NaCl.run()
zcoor_centered_NaCl = dplan_centered_NaCl.results.bin_pos
dens_centered_NaCl = dplan_centered_NaCl.results.profile
```

```
/home/docs/checkouts/readthedocs.org/user_builds/maicos/envs/main/lib/python3.9/site-
↪packages/maicos/lib/math.py:303: RuntimeWarning: invalid value encountered in divide
  (1 - np.arange(1, cutoff) / len(timeseries)) * corr[1:cutoff] / corr[0]
/home/docs/checkouts/readthedocs.org/user_builds/maicos/envs/main/lib/python3.9/site-
↪packages/maicos/core/base.py:456: UserWarning: Your trajectory does not provide␣
```

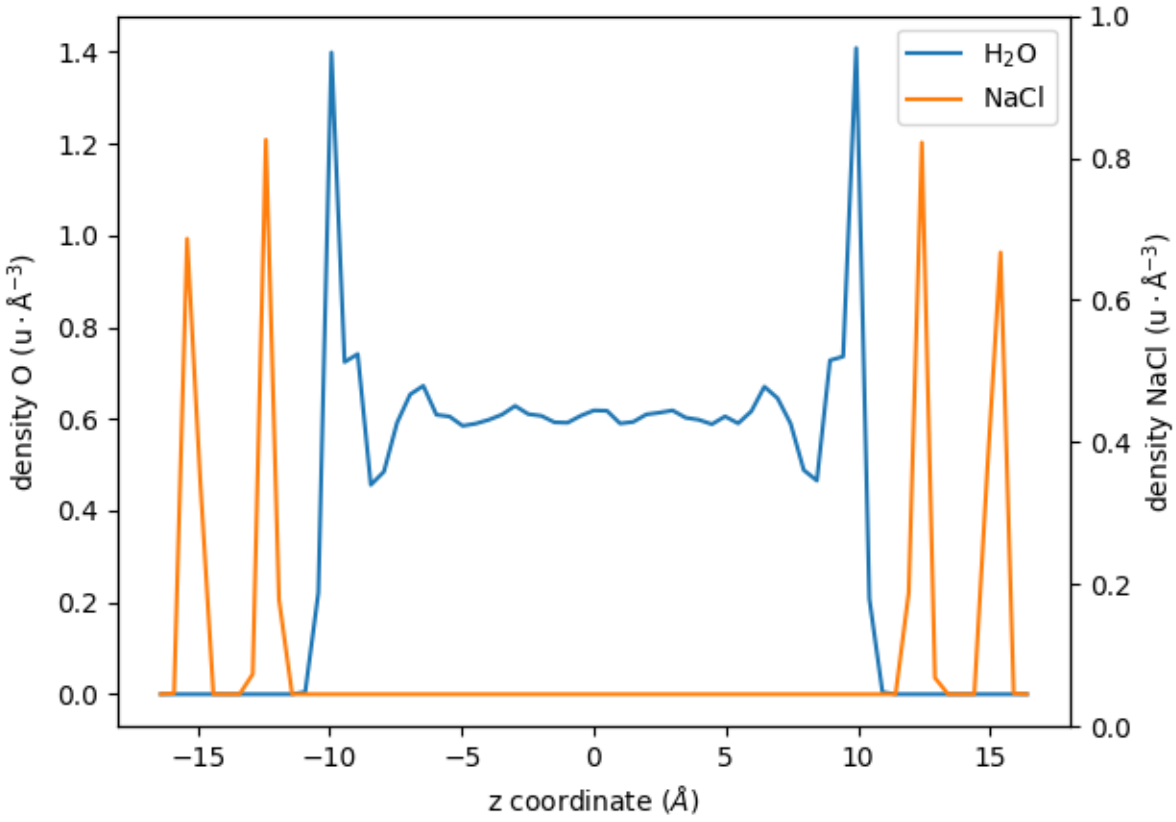
(continues on next page)

(continued from previous page)

```
↪sufficient statistics to estimate a correlation time. Use the calculated error␣
↪estimates with caution.
  self.corrtime = correlation_analysis(self.timeseries)
```

An plot the two profiles with different y-axis:

```
fig, ax1 = plt.subplots()

ax1.plot(zcoor_centered_H2O, dens_centered_H2O, label=r"$\rm H_2O$")
ax1.plot(zcoor_centered_NaCl, dens_centered_NaCl / 5, label=r"$\rm NaCl$")

ax1.set_xlabel(r"z coordinate ($Å$)")
ax1.set_ylabel(r"density O ($\rm u \cdot Å^{-3}$)")
ax1.legend()

ax2 = ax1.twinx()
ax2.set_ylabel(r"density NaCl ($\rm u \cdot Å^{-3}$)")

fig.show()
```



Additional Options

Use `verbose=True` to display a progress bar:

```
dplan_verbose = maicos.DensityPlanar(group_H2O)
dplan_verbose.run(verbose=True)
```

```
Unwrapping in combination with the `wrap_compound='atoms` is superfluous. `unwrap` will␣
↪be set to `False`.

  0%|          | 0/201 [00:00<?, ?it/s]
 18%|        | 37/201 [00:00<00:00, 360.88it/s]
 37%|      | 74/201 [00:00<00:00, 360.00it/s]
 55%|    | 111/201 [00:00<00:00, 362.00it/s]
 74%|  | 148/201 [00:00<00:00, 362.96it/s]
 92%|| 185/201 [00:00<00:00, 362.81it/s]
100%|| 201/201 [00:00<00:00, 361.47it/s]

<maicos.modules.densityplanar.DensityPlanar object at 0x7fad78e1cbe0>
```

To analyse only a subpart of a trajectory file, for instance to analyse only frames 2, 4, 6, 8, and 10, use the `start`, `stop`, and `step` keywords as follow:

```
dplan = maicos.DensityPlanar(group_H2O).run(start=10, stop=20, step=2)
```

```
Unwrapping in combination with the `wrap_compound='atoms` is superfluous. `unwrap` will␣
↪be set to `False`.
```

Velocity Profile

Here we use the same trajectory file, but extract the velocity profile instead of the density profile. Do to so, the *maicos.VelocityPlanar* is used.

Let us call the velocity module:

```
tplan = maicos.VelocityPlanar(group_H2O, bin_width=0.5, vdim=0, flux=False).run()

zcoor = tplan.results.bin_pos
vel = tplan.results.profile
```

```
Unwrapping in combination with the `wrap_compound='atoms` is superfluous. `unwrap` will␣
↪be set to `False`.
```

0`` option, but the binning is made along the default z axis.

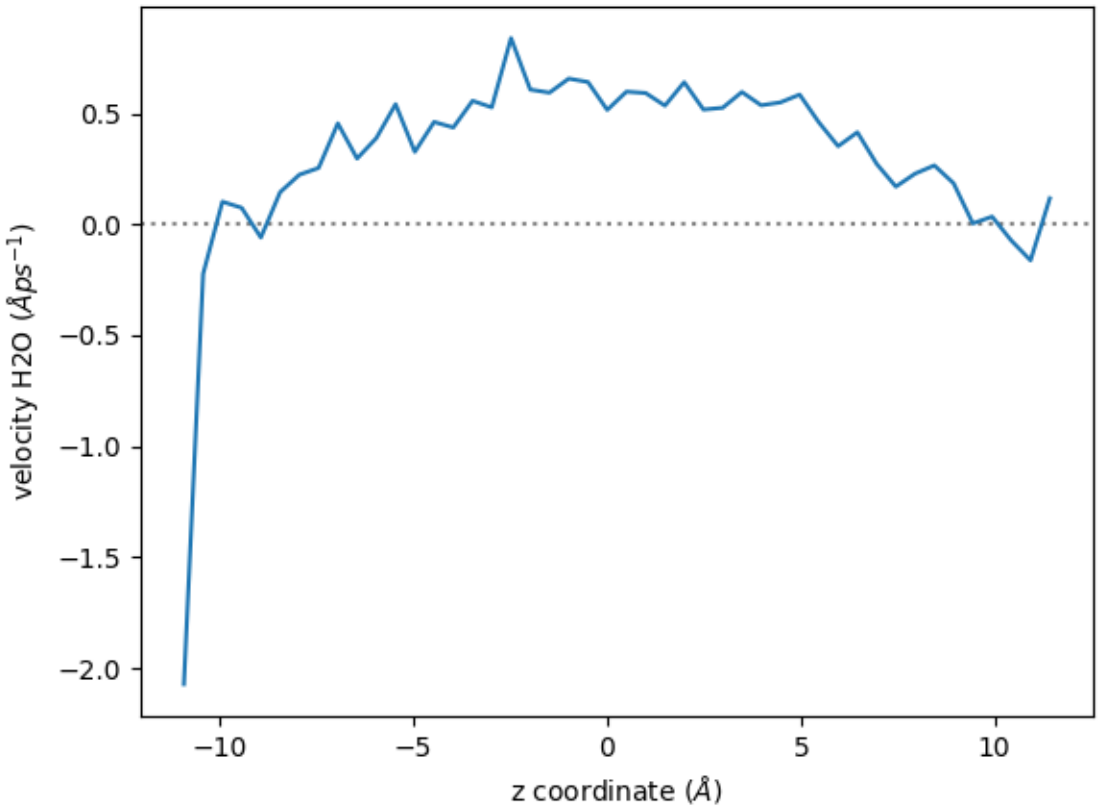
And plot the velocity profile:

```
fig, ax = plt.subplots()

ax.axhline(0, linestyle="dotted", color="gray")
ax.plot(zcoor, vel)

ax.set_xlabel(r"z coordinate ($Å$)")
```

(continues on next page)

(continued from previous page)

```
ax.set_ylabel(r"velocity H2O ($Å ps^{-1}$)")

fig.show()
```



Finally, still using the same trajectory file, we extract the average orientation of the water molecules.

Let us call the *maicos.DiporderPlanar* to extract the average orientation of the water molecules:

```
mydiporder = maicos.DiporderPlanar(
    group_H2O, refgroup=group_H2O, order_parameter="cos_theta"
).run()
```

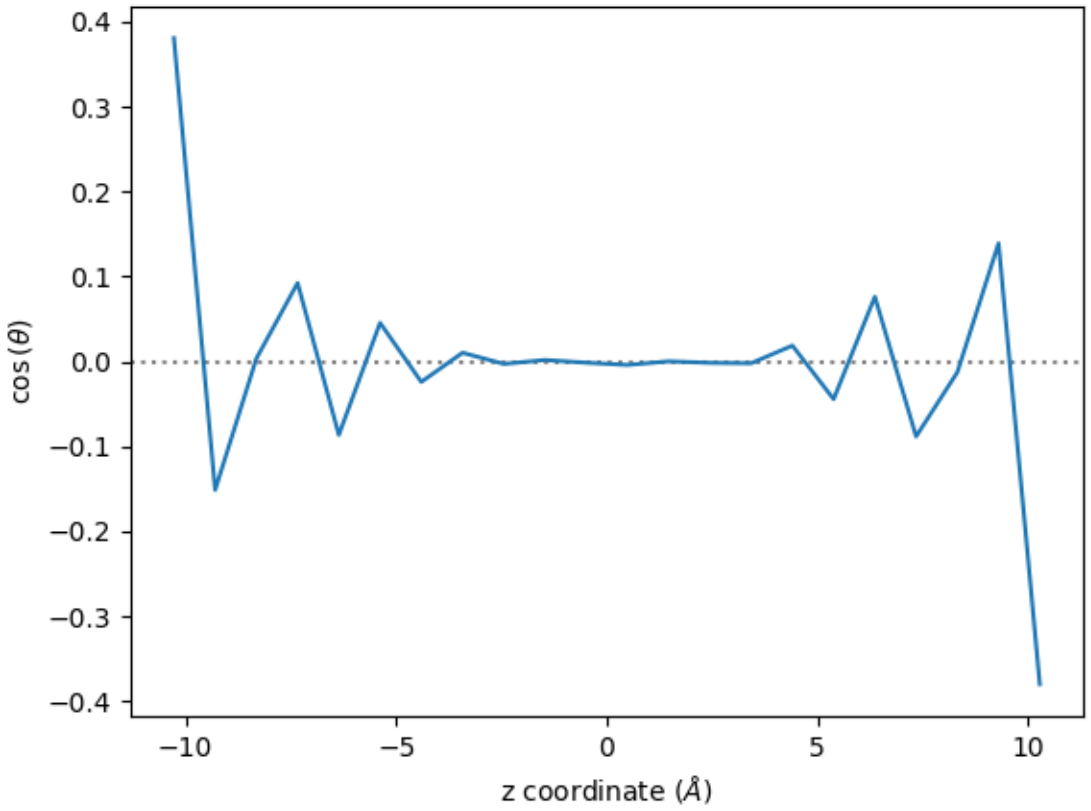
Then, let us extract the cosinus of the angle of the molecules, $\cos(\theta)$:

```
zcoor = mydiporder.results.bin_pos
cos_theta = mydiporder.results.profile

fig, ax = plt.subplots()

ax.axhline(0, linestyle="dotted", color="gray")
ax.plot(zcoor, cos_theta)
```

(continues on next page)

(continued from previous page)

```
ax.set_xlabel(r"z coordinate ($Å$)")
ax.set_ylabel(r"$\cos$($\theta$)")

plt.show()
```



5.1.4 Usage - command line

MAICoS can be used directly from the command line (cli). Using cli instead of a Jupyter notebook can sometimes be more comfortable, particularly for lengthy analysis. The cli in particular is handy because it allows for updating the analysis results during the run. You can specify the number of frames after the output is updated with the `-concfreq` flag. See below for details.

Note that in this documentation, we almost exclusively describe the use of MAICoS from the python interpreter, but all operations can be equivalently performed from the cli.

```
#!/bin/bash
# -*- Mode: bash; tab-width: 4; indent-tabs-mode:nil; coding:utf-8 -*-
#
# Copyright (c) 2024 Authors and contributors
# (see the AUTHORS.rst file for the full list of names)
#
# Released under the GNU Public Licence, v3 or any higher version
```

(continues on next page)

(continued from previous page)

```
# SPDX-License-Identifier: GPL-3.0-or-later

maicos densityplanar -s slit_flow.tpr \
                     -f slit_flow.trr \
                     -atomgroup 'type OW HW'

# The density profile has been written in a file named ``density.dat`` in the current
# directory. The written file starts with the following lines

head -n 20 density.dat

# For lengthy analysis, use the ``concfreq`` option to update the result during the run

maicos densityplanar -s slit_flow.tpr \
                    -f slit_flow.trr \
                    -atomgroup 'type OW HW' \
                    -concfreq '10'

# The general help of MAICoS can be accessed using

maicos -h

# Package-specific page can also be accessed from the cli

maicos densityplanar -h
```

5.1.5 Some conventions

The base units of MAICoS are consistent with those of MDAnalysis. Keeping inputs and outputs consistent with this set of units reduces ambiguity, so we encourage everyone to use them exclusively.

The base units are:

Table 1: Base units in MDAnalysis

quantity	unit	SI units
length	Å	10^{-10} m
mass	u	$1.660538921 \times 10^{-27}$ kg
time	ps	10^{-12} s
energy	kJ/mol	$1.66053892103219 \times 10^{-21}$ J
charge	e	$1.602176565 \times 10^{-19}$ As
force	kJ/(mol·Å)	$1.66053892103219 \times 10^{-11}$ J/m
speed	Å/ps	100 m/s

5.1.6 Changelog

v0.9 (XXXX/XX/XX)

Philip Loche, Marc Sauter, Kira Fischer, Federico Grasselli, Henrtik Stooß

- Remove handling of multiple atomgroups in favor of `AnalysisCollection` (!301)
- Fix openMP detection during setup (!304)
- *maicos.Saxs* additionally provides structure factor. (!303)
- Remove default arguments from core classes (!302)
- Add an `AnalaysisCollection` class to perform multiple analyses on the same trajectory (!298)
- Remove custom module command line interface (!299)
- Add example for *maicos.core.AnalysisBase* and rework own module section in developer docs (!299)
- Allow running an analysis with a universe without a cell (!297)
- Test that `core.AnalysisBase` API and `run` method is the same as `MDAnalysis.analysis.base.AnalysisBase` (!297)
- Add `frames` and `progressbar_kwargs` argument to *maicos.core.AnalysisBase.run()* (!297)
- Update copyright year (!296)
- Add new diporder modules: `RDFDiporder`, `DiporderStructureFactor` (!296)
- Add correlation time estimate for `SAXS` module (!296)
- Added tests of the analytical error propagation (!292)
- Remove symbolic links from examples (!295)

v0.8 (2024/02/05)

Simon Gravelle, Philip Loche, Marc Sauter, Henrik Stooß, Philipp Staerk, Adyant Agrawal, Kira Fischer

- Skip test for custom modules in case the import is not working (!294)
- Change to CHANGELOG.rst update check so that it is only executed in MRs (!198)
- Rename radial distribution function to pair distribution function (!278)
- Add RDF derivation and explain role of dz. (!278)
- Implement 1D pair distribution function in RDFCylinder (!276)
- Sort format and add more atomtypes to `atomtypes.dat` (!291)
- Add grouping option to *DipoleAngle* module (!290)
- Added Support for Python 3.12 (!289)
- Remove suffixes `-linux`, `-macos`, `-windows` when building wheels. Platform will be detected automatically. (!288)
- Use default tox error for non-exsiting enviroment (!285)
- Parse documentation metadata from `pyproject.toml` (!287)
- Convert `pathlib.Path` into `str` when using in `sys.path.append` (#123, !286)
- Update dev names (!284)

- Improvements to documentation rendering (#122, !282)
- Unify Python versions in tox environments i.e. `py311-build-macos` to `build-macos` (!283)
- Remove deprecated pytest tmpdir fixture (!283)
- Remove deprecated `assert_almost_equal` in favor of `assert_allclose` (!283)
- Move from `os.path` to `pathlib.Path` (!283)
- Added Support for Python 3.11 (!283)
- Update MacOS images for CI (!281)
- Removed the obsolete option for the vacuum boundary condition in the `DielectricPlanar` module and prompt users to use tin-foil boundary conditions instead (!280).
- Add physical integration test to test that structure factor from Saxs is the same as the Fourier transformed RDF. (!279)
- Add example and explenation of how to relate the radial distribution function and the structure factor (!279)
- Add function `maicos.lib.math.rdf_structure_factor()` for converting a radial distribution function into a structure factor. (!279)
- Change default biwnwidth (`dq`) in *maicos.Saxs* to `0.1`. (!279)
- Move `cutils` to `cmath` (!279)
- Add `weight` argument to *maicos.lib._cmath.compute_structure_factor()*
- Code cleanup of *maicos.Saxs* (!279)
- Fixed markup and consistency in `correlation` function docs (!277)
- Add info for `DielectricPlanar` module for ignored combination of `vac=True` and `is_3d=False`. (!275)
- Add description for *tox* jobs (!275)
- Cleanup coverage config and move to `pyproject.toml` (!275)
- Changed the way number normalization works, introduced sums dict (!274)
- Fixed typing error in RDF modules (!273)
- Update docs to reflect changes in `mdacli` (!271)
- Add banner to MAICoS output reporting the version (!272)
- Update UML graphic (!269)
- Show warnings if set boundaries would result in wrong results (!261)
- Small corrections to the documentation and type hinting (!268)
- Add module for calculating radial distribution functions in cylinders (!242)
- Add modules for calculating cylindrical and spherical dipolar order parameters (!259)
- Fix reproducibility information in output (!263)
- Make savetxt work with Pathlib objects (!267)
- Update versionner to 0.29 (!266)
- Use `dipole_vector` methods from MDAnalysis (!265)
- Bump minimum Python version to 3.9 (!264)
- Fix dipole calculation in `DielectricCylinder` (!258)

- Add example for RDFPlanar (!256)
- Move geometry transformations to `lib.math` (!257)
- Add typehints for examples (!255)
- Add typehints for modules (!253)
- Only test minimum and maximum Python version in CI (!252)
- Add typehints for core classes (!251)
- Update documentation with parameters, returns and examples for library functions (!248)
- Update CI to use latest MacOS (!250)
- Add tables to documentation pages (!249)
- Fix links to own classes in examples (!247)
- Update install instructions for users and devs (!246)
- Show authors on website (!245)
- Add link to developer documentation in `CONTRIBUTING.rst` (!244)
- Remove Python 2.x leftover of specific `super()` call (!243)
- Use Gitlab for showing coverage and unit test reports (!241)
- Use `black` formatter and *88* chars/line for the code and rst files (!240)
- Add return values for correlation analysis to all base classes (!235)
- Added more linting for rst files (!239)
- Bump minimum version of `tqdm` to 4.60 (!238)
- Add prompt toggle to examples (!236)
- Added description to the ideal chemical potential how-to (!232)
- Added quotation marks to command in tox.ini to account for spaces in paths (!232)
- Fixed some typos and made minor modifications to the documentation (!232)
- Cleanup .gitignore (!233)
- More consistent molecule wrapping (!230)
- Added missing AnalysisBase parameters to modules (!231)
- created dark and light images and logo (!229)
- Add explicit *stacklevel* arguments to warnings in the library (!236)
- Switch to the *build* module (!234)

v0.7.2 (2023/01/09)

Philip Loche, Henrik Stooß

- Remove superfluous group wise wrapping (!225)
- Clarify unclear definition in Dieletric modules that could lead to wrong results (!228)
- Fixed windows string manipulation in test CI (!227)
- Added coverage posting on GitLab (!226)
- Corrected wrong comparison in correlation analysis and added tests
- Fixed link to changelog in pyproject.toml
- Migrated versioneer to pyproject.toml
- Added Support for Python 3.11

v0.7.1 (2023/01/01)

Henrik Stooß

- Fix upload to PyPi. This release is identical to v0.7.

v0.7 (2022/12/27)

Philip Loche, Simon Gravelle, Marc Sauter, Henrik Stooß, Kira Fischer, Alexander Schlaich, Philipp Staerk

- Make sure citation are only printed once (!260)
- Added MacOS pipeline, fixed wheels (!218)
- Fix CHANGELOG testing (!220)
- Added dielectric how-to (!208)
- Raise an error if `unwrap=False` and `refgroup != None` in dielectric modules (!215).
- Fix velocity profiles (!211)
- Added the Theory to the Dielectric docs (!201)
- Add a logger info for citations (!205)
- Rename Diporder to DiporderPlanar (!202)
- Change default behavior of DielectricPlanar: assume slab geometry by default (removing the `xy` flag and instead introduce `is_3d` for 3d-periodic systems) (!202)
- Rename `profile_mean` to `profile` (!202)
- Major improvements on the documentation (!202)
- Add a check if the CHANGELOG.rst has been updated (!198)
- Fix behaviour of refgroup (!192)
- Resolve +1 is summed for epsilon for each atom group (#101, !193)
- Flatten file structure of analysis modules (#46, !196)
- Consistent mass unit in docs
- Porting examples to sphinx-gallery (!190)

- Add `jitter` parameter to AnalysisBase (!183)
- Test output messages (!191)
- Fixed typo in `DielectricPlanar` docs (!194)
- Add Sphere modules (!175)
- Add `ProfileBase` class (!180)
- Slight restructure of the documenation (!189)
- Fix py311 windows
- Update build requirements for py310 and py311
- Merged setup.cfg into pyproject.toml (!187)
- Use versioneer for version info (!150)
- Update project urls (!185)
- Added repository link in the documentation (!184)
- Added windows CI/CD pipeline (!182)
- Update package discovery methods in setup.cfg
- Refactor CI script (!181)
- Fix `DielectricCylinder` (!165)
- Unified `n_bins` logging (#93, !179)
- Add MAICoS UML Class Diagramm (!178)
- Changed density calculation using range in np.histogram (!77)
- Update branching model in the documentation (!177)
- remove ./ from index.rst
- Improve documentation (!174)
- Added reference for SAXS calculations (!176)
- Update type of bin_pos in docs
- Added `VelocityCylinder` module
- Change behavior of `sort_atomgroup` (#88, !152)
- `get_compound`: option for returning indices of topology attributes
- Added Tutorial for non-spatial analysis module (!170)
- Check atomgroups if they contain any atoms (!172)
- New core attributes: `bin_edges`, `bin_area`, `bin_volume`, `bin_pos` & `bin_width` (!167)
- Use `frame` dict in `structure.py` (!169)
- Fix box dimensions for cylindrical boundaries (!168)
- `rmax` for cylindrical systems now uses correct dimensions
- Transport module documentation update (!164)
- Rename frame dict (!166)
- Implement `SphereBase` and `ProfileSphereBase` (!162)

- Relative path for data (!163)
- Create Linux wheels (!160)
- Fix `Diporder` tests (!161)
- `norm=number`: Declare bins with no atoms as `nan` (!157)
- Simplify weight functions (!158)

v0.6.1 (2022/09/26)

Henrik Stooß

- Fix the output of the *ChemicalPotentialPlanar* module (!173)

v0.6 (2022/09/01)

Philip Loche, Simon Gravelle, Srihas Velpuri, Henrik Stooß, Alexander Schlaich, Maximilian Becker, Kira Fischer

- Write total epsilon as defined in paper (!155)
- Introduce generic header (!149)
- Fix error estimate in `EpsilonPlanar` (!153)
- Fix sym option in `EpsilonPlanar` (!148)
- Use standard error of the mean instead of variance for error estimate (!147)
- Make all tests that write file use temporary file directory (!151)
- Rewrite `Velocity` module using `ProfilePlanarBase` (!142)
- Add `RDFPlanar` (!133)
- Refactor `EpsilonPlanar` (!139)
- Add a correlation time estimator (!137)
- Add `frame` dict to `AnalysisBase` (!138)
- Generalize `comgroup` attribute to all dimensions (!132)
- Output headers do not require residue names anymore (!134)
- Remove `Debyer` class (!130)
- Generalize `concfreq` attribute in `AnalysisBase` (!122)
- Fix broken binning in `EpsilonPlanar` (!125)
- Removed `repairMolecules` (!119)
- Added `grouping` and `bin_method` option (!117)
- Bump minimum MDAnalysis version to 2.2.0 (!117)
- Bump minimum Python version to 3.8 (!117)
- Use base units exclusively (!115)
- Higher tolerance for non-neutral systems (1E-4 instead of 1E-5)
- `charge``neutral decorator uses ``check_compound` now
- Add option to symmetrize profiles using `ProfilePlanarBase` (!116)

- Fix `comgroup` parameter working only in the z direction (!116)
- Remove `center` option from `ProfileBase` (!116)
- Introduces new `ProfilePlanarBase` (!111)
- Split new `DensityPlanar` into `ChemicalPotentialPlanar`, `DensityPlanar`, `TemperaturePlanar` (!111)
- Convert more `print` statements into logger calls (!111)
- Fix wrong `Diporder` normalization + tests (!111)
- Add `zmin` and `zmax` to DensityPlanar and Diporder (!109)
- Fix EpsilonPlanar (!108)
- More tests for `DensityPlanar`, `DensityCylinder`, `KineticEnergy` and `DipoleAngle` (!104)
- Remove `EpsilonBulk` (!107)
- Add Code of Conduct (!97)
- Fix lint errors (!95)

v0.5.1 (2022/02/21)

Henrik Stooß

- Fix pypi installation (!98)

v0.5 (2022/02/17)

Philip Loche, Srihas Velpuri, Simon Gravelle

- Convert Tutorials into notebooks (!93)
- New docs design (!93)
- Build gitlab docs only on master branch (!94, #62)
- Removed oxygen binning from diporder (!85)
- Improved CI including tests for building and linting
- Create a consistent value of `zmax` in every frame (!79)
- Corrected README for pypi (!83)
- Use Results class for attributes and improved docs (!81)
- Bump minimum Python version to 3.7 (!80)
- Remove spaghetti code in `__main__.py` and introduce `mdacli` as cli server library. (!80)
- Remove `SingleGroupAnalysisBase` and `MultiGroupAnalysisBase` classes in favour of a unified `AnalysisBase` class (!80)
- Change `planar_base` decorator to a `PlanarBase` class (!80)
- Rename modules to be consistent with PEP8 (`density_planar` -> `DensityPlanar`) (!80)
- Use Numpy's docstyle for doc formatting (!80)
- Use Python's powerful logger library instead of bare `print` (!80)
- Use Python 3.6 string formatting (!80)

- Remove `_calculate_results` methods. This method is covered by the `_conclude` method. (!80)
- Make results saving a public function (save) (!80)
- Added docstring Decorator for `PlanarDocstring` and `verbose` option (!80)
- Use `MDAnalysis`'s' `center_of_mass` function for center of mass shifting (!80)

v0.4.1 (2021/12/17)

Philip Loche

- Fixed double counting of the box length in diporder (#58, !76)

v0.4 (2021/12/13)

Philip Loche, Simon Gravelle, Philipp Staerk, Henrik Stooß, Srihas Velpuri, Maximilian Becker

- Restructure docs and build docs for develop and release version
- Include README files into sphinx doc
- Add tutorial for density_cylinder module
- Add `planar_base` decorator unifying the syntax for planar analysis modules as `denisty_planar`, `epsilon_planar` and `diporder` (!48)
- Corrected time_series module and created a test for it
- Added support for Python 3.9
- Created sphinx documentation
- Raise error if end is to small (#40)
- Add sorting of atom groups into molecules, enabling import of LAMMPS data
- Corrected plot format selection in `dielectric_spectrum` (!66)
- Fixed box dimension not set properly (!64)
- Add docs for timeseries modulees (!72)
- Fixed diporder does not compute the right quantities (#55, !75)
- Added support of calculating the chemical potentials for multiple atomgroups.
- Changed the codes behaviour of calculating the chemical potential if atomgroups contain multiple residues.

v0.3 (2020/03/03)

Philip Loche, Amanuel Wolde-Kidan

- Fixed errors occurring from changes in MDAnalysis
- Increased minimal requirements
- Use new ProgressBar from MDAnalysis
- Increased total_charge to account for numerical inaccuracy

v0.2 (2020/04/03)

Philip Loche

- Added custom module
- Less noisy DeprecationWarning
- Fixed wrong center of mass velocity in velocity module
- Fixed documentation in diporder for P0
- Fixed debug if error in parsing
- Added checks for charge neutrality in dielectric analysis
- Added test files for an air-water trajectory and the diporder module
- Performance tweaks and tests for sfactor
- Check for molecular information in modules

v0.1 (2019/10/30)

Philip Loche

- first release out of the lab

5.2 How-to guides

Like a cooking recipe, How-to guides help you solve key problems and use cases. If you are a total MAICoS beginner, you should start with the *Getting started* section.

5.2.1 Small-angle X-ray scattering

Small-angle X-ray scattering (SAXS) can be extracted using MAICoS. To follow this how-to guide, you should download the `topology` and the `trajectory` files of the water system.

For more details on the theory see *Small-angle X-ray scattering*.

First, we import Matplotlib, MDAnalysis, NumPy and MAICoS:

```
import matplotlib.pyplot as plt
import MDAnalysis as mda
from MDAnalysis.analysis.rdf import InterRDF

import maicos
from maicos.lib.math import compute_form_factor, compute_rdf_structure_factor
```

The *water* system consists of 510 water molecules in the liquid state. The molecules are placed in a periodic cubic cell with an extent of $25 \times 25 \times 25\,\text{Å}^3$.

Load Simulation Data

Create a `MDAnalysis.core.universe.Universe` and define a group containing only the oxygen atoms and a group containing only the hydrogen atoms:

```
u = mda.Universe("water.tpr", "water.trr")

group_O = u.select_atoms("type O*")
group_H = u.select_atoms("type H*")
```

Extract small angle x-ray scattering (SAXS) intensities

Let us use the *maicos.Saxs* class of MAICoS and apply it to all atoms in the system:

```
saxs = maicos.Saxs(u.atoms).run(stop=30)
```

Note: SAXS computations are extensive calculations. Here, to get an overview of the scattering intensities, we reduce the number of frames to be analyzed from `101` to `30`, by adding the `stop = 30` parameter to the `run` method. Due to the small number of analyzed frames, the scattering intensities shown in this tutorial should not be used to draw any conclusions from the data.

Extract the scattering vectors and the averaged structure factor and SAXS scattering intensities from the `results` attribute:

```
scattering_values = saxs.results.scattering_vectors
structure_factors = saxs.results.structure_factors
scattering_intensities = saxs.results.scattering_intensities
```

The scattering intensities (and structure factors) are given as a 1D array, let us look at the 10 first lines:

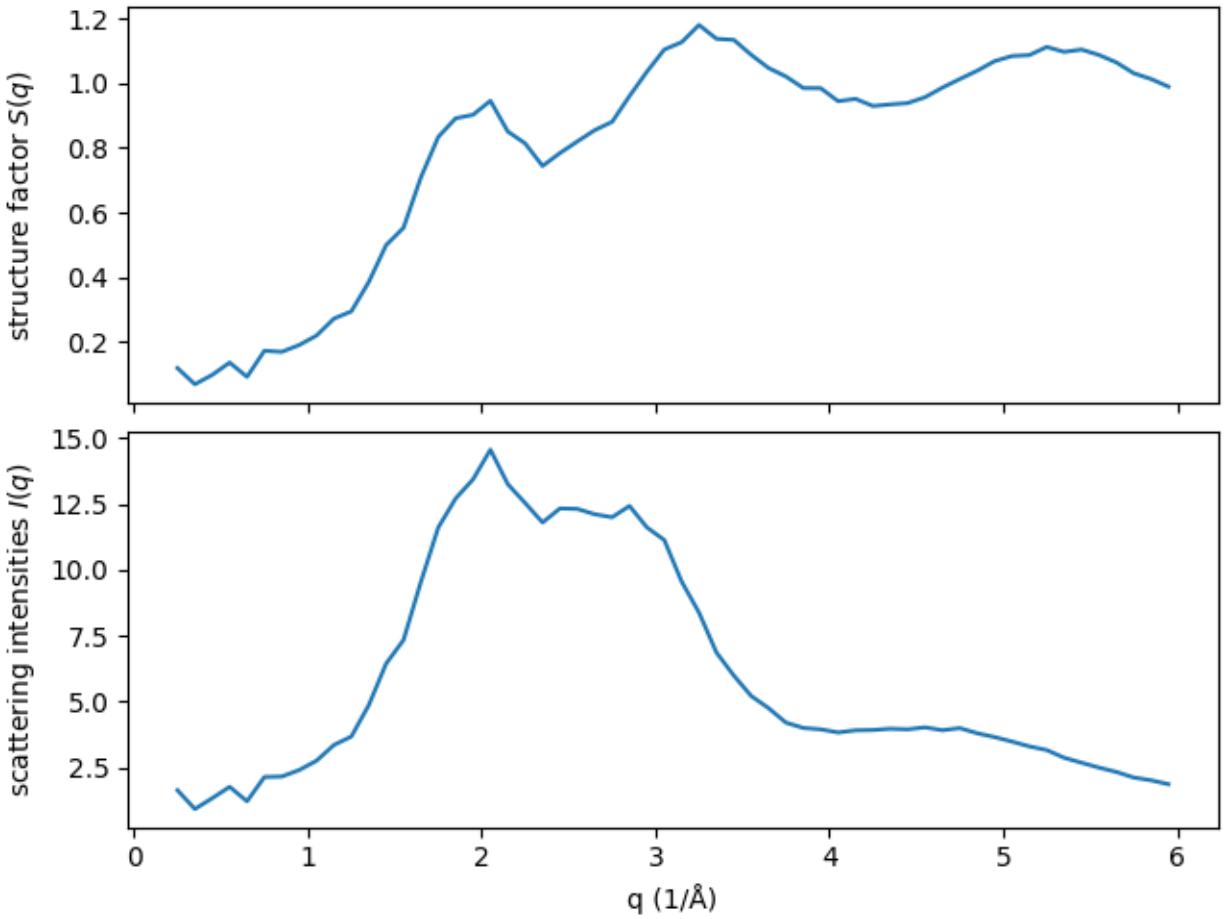
```
print(scattering_intensities[:10])
```

```
[1.62620077 0.91205581 1.32501428 1.75529088 1.20605233 2.13058505
 2.14645164 2.39313797 2.74842731 3.34680856]
```

By default, the binwidth in the recipocal (q) space is 0.1^{-1}.

We now plot the structure factor as well a the scattering intensities together.

```
fig1, ax1 = plt.subplots(nrows=2, sharex=True, layout="constrained")

ax1[0].plot(scattering_values, structure_factors)
ax1[1].plot(scattering_values, scattering_intensities)

ax1[-1].set_xlabel(r"q (1/Å)")

ax1[0].set_ylabel(r"structure factor $S(q)$")
ax1[1].set_ylabel(r"scattering intensities $I(q)$")
fig1.align_labels()

fig1.show()
```



The structure factor $S(q)$ and the scattering intensities $I(q)$ are related via

$$I(q) = [f(q)]^2 S(q)$$

where $f(q)$ are the atomic form factors. We will investigate the relation below in more details.

Computing oxygen and hydrogen contributions

An advantage of full atomistic simulations is their ability to investigate atomic contributions individually. Let us calculate both oxygen and hydrogen contributions, respectively:

```
saxs_O = maicos.Saxs(group_O).run(stop=30)
saxs_H = maicos.Saxs(group_H).run(stop=30)
```

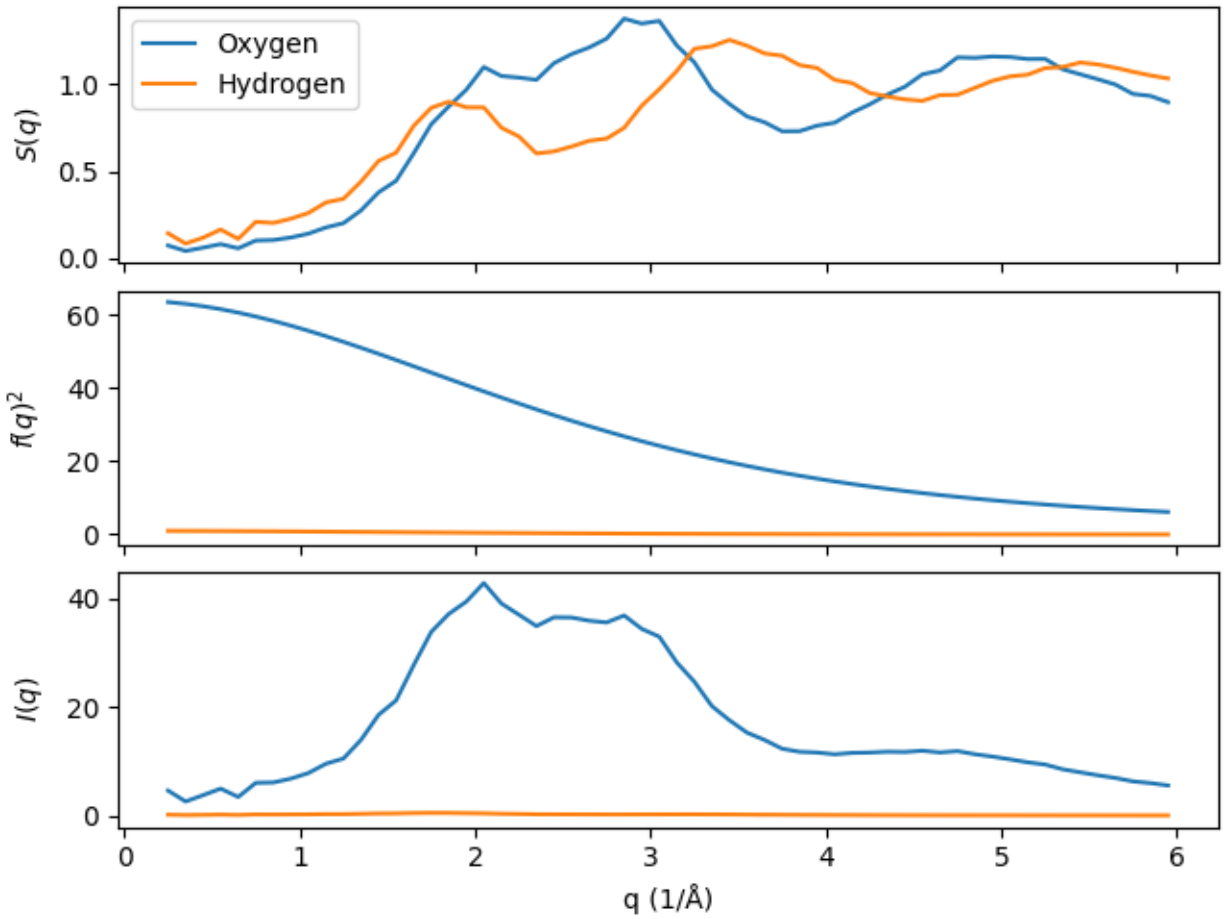
Let us plot the results for the structure factor, the squared form factor as well scattering intensities together. For computing the form factor we will use *maicos.lib.math.compute_form_factor()*. Note that here we access the results directly from the `results` attribute without storing them in individual variables before:

```
fig2, ax2 = plt.subplots(nrows=3, sharex=True, layout="constrained")

# structure factors
ax2[0].plot(
    saxs_O.results.scattering_vectors,
    saxs_O.results.structure_factors,
```

(continues on next page)

(continued from previous page)

```
    label="Oxygen",
)
ax2[0].plot(
    saxs_H.results.scattering_vectors,
    saxs_H.results.structure_factors,
    label="Hydrogen",
)

# form factors
ax2[1].plot(
    saxs_O.results.scattering_vectors,
    compute_form_factor(saxs_O.results.scattering_vectors, "O") ** 2,
)
ax2[1].plot(
    saxs_H.results.scattering_vectors,
    compute_form_factor(saxs_H.results.scattering_vectors, "H") ** 2,
)

# scattering intensitie
ax2[2].plot(saxs_O.results.scattering_vectors, saxs_O.results.scattering_intensities)
ax2[2].plot(saxs_H.results.scattering_vectors, saxs_H.results.scattering_intensities)

ax2[-1].set_xlabel(r"q (1/Å)")
ax2[0].set_ylabel(r"$S(q)$")
ax2[1].set_ylabel(r"$f(q)^2$")
ax2[2].set_ylabel(r"$I(q)$")

ax2[0].legend()
fig2.align_labels()

fig2.show()
```



The figure above nicely shows that multiplying the structure factor $S(q)$ and the squared form factor $f(q)^2$ results in the scattering intensity $I(q)$. Also, it is worth to notice that due to small form factor of hydrogen there is basically no contribution of the hydrogen atoms to the total scattering intensity of water.

Connection of the structure factor to the radial distribution function

As in details explained in *Small-angle X-ray scattering*, the structure factor can be related to the radial distribution function (RDF). We denote this structure factor by $S^{\mathrm{FT}}(q)$ since it is based on Fourier transforming the RDF. The structure factor which can be directly obtained from the trajectory is denoted by $S^{\mathrm{D}}(q)$.

To relate these two we first calculate the oxygen-oxygen RDF up to half the box length using `MDAnalysis.analysis.rdf.InterRDF` and save the result in variables for an easier access.

```
box_lengh = u.dimensions[0]

oo_inter_rdf = InterRDF(
    g1=group_O, g2=group_O, range=(0, box_lengh / 2), exclude_same="residue"
).run()

r_oo = oo_inter_rdf.results.bins
rdf_oo = oo_inter_rdf.results.rdf
```

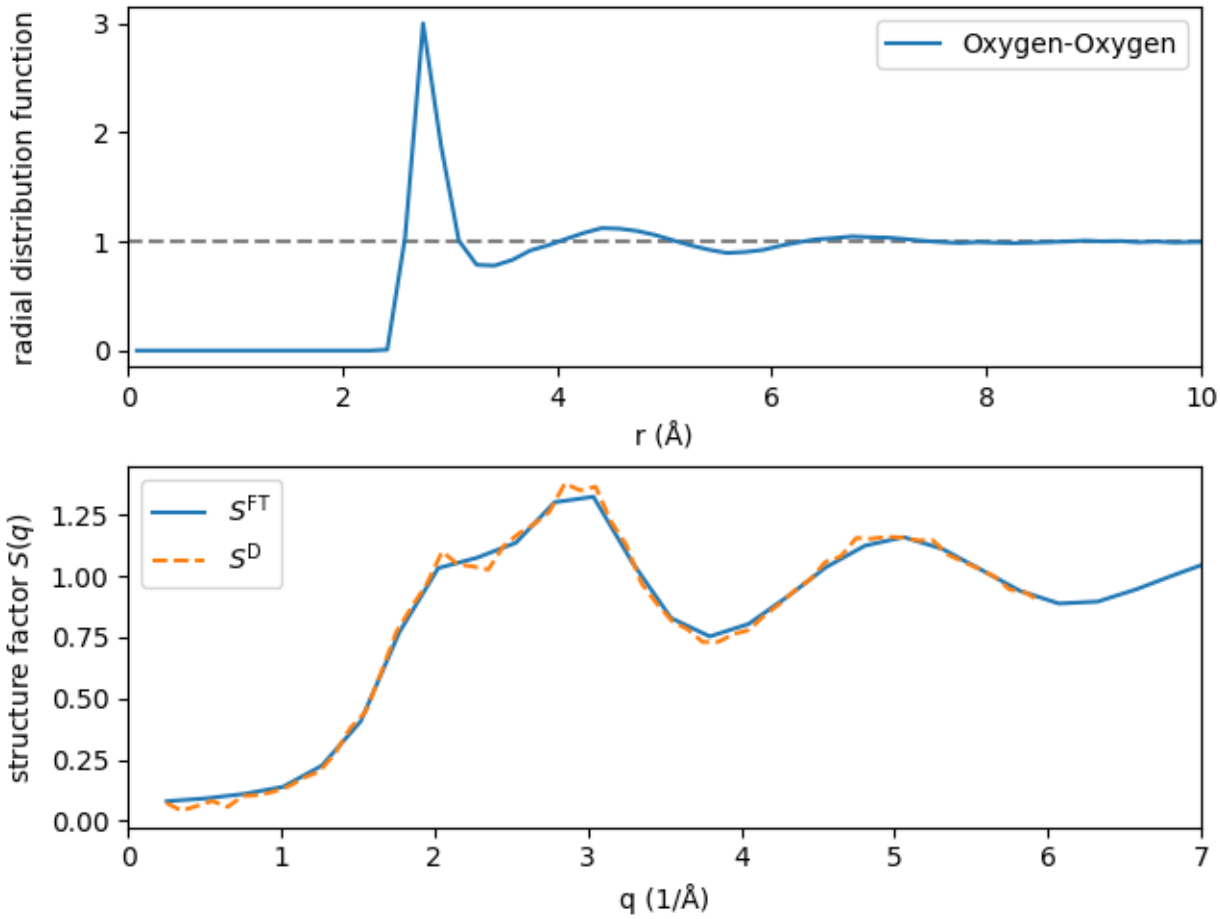
We use `exclude_same="residue"` to exclude atomic self contributions resulting in a large peak at 0. Next, we convert the RDF into a structure factor using *`maicos.lib.math.compute_rdf_structure_factor()`* and the number

density of the oxygens.

```
density = group_O.n_atoms / u.trajectory.ts.volume

q_rdf, struct_factor_rdf = compute_rdf_structure_factor(
    rdf=rdf_oo, r=r_oo, density=density
)
```

Now we can plot everything together and find that the direct evaluation from above and the transformed RDF give the same structure factor.

```
fig3, ax3 = plt.subplots(2, layout="constrained")

ax3[0].axhline(1, c="gray", ls="dashed")
ax3[0].plot(r_oo, rdf_oo, label="Oxygen-Oxygen")
ax3[0].set_xlabel("r (Å)")
ax3[0].set_ylabel("radial distribution function")
ax3[0].set_xlim(0, 10)

ax3[1].plot(q_rdf, struct_factor_rdf, label=r"$S^\mathrm{FT}$")
ax3[1].plot(
    saxs_O.results.scattering_vectors,
    saxs_O.results.structure_factors,
    label=r"$S^\mathrm{D}$",
    ls="dashed",
)

ax3[1].set_xlabel("q (1/Å)")
ax3[1].set_ylabel("structure factor $S(q)$")
ax3[1].set_xlim(0, 7)

ax3[1].legend()
ax3[0].legend()
fig3.align_labels()

fig3.show()
```



5.2.2 Ideal component of the chemical potential

What is the chemical potential?

Molecular dynamics simulations are often performed with a constant number of particles. When modelling confined systems in molecular dynamics simulations, it is often assumed that the confined geometry extends infinitely, while real systems have a finite size and are connected to a reservior many times larger than the confined space.

In this case, the number of particles in the system is not constant, but changes over time. This can be seen as a system that is exchanging particles with a reservoir. The chemical potential describes how the free energy changes when particles are added to (or removed from) the system. The chemical potential is therefore a very important quantity in molecular dynamics simulations of confined systems.

If you want to know more about what the chemical potential means you can take a look at the references below[1].

[1] G. Cook and R.H. Dickerson. Understanding the chemical potential. *American Journal of Physics*, 1995. doi:10.1119/1.17844.

How to calculate the ideal component of the chemical potential

The chemical potential can be split up into different parts

$$\mu = \mu^0 + \mu^{\text{ideal}} + \mu^{\text{excess}},$$

where μ^0 represents the standard potential of the substance, μ^{ideal} represents the component of the potential that would also occur for an ideal gas and μ^{excess} represents the excess contribution generated from the interactions between the particles. In the following calculations we are only interested in the ideal component.

For our case, we can calculate the ideal component of the potential according to

$$\mu^{\text{ideal}} = RT \ln\left(\rho \Lambda^3\right),$$

where $\Lambda = \hbar\sqrt{\frac{2\pi}{m \cdot k_B \cdot T}}$ is the thermal De-Broglie wavelength, i.e. the mean De-Broglie wavelength at temperature T. Furthermore, m is the mass of the particles and ρ is the mean density of the system. The mean density can be calculated with MAICoS by using the Density modules. We will exemplify this in the following example using the *maicos.DensityPlanar* module.

First we'll import every module we need.

```
import MDAnalysis as mda
import numpy as np
from scipy import constants as const

import maicos
```

Now we define a function that calculates μ according to the equation above. We can calculate the Volume V with MAICoS by calculating the mean density and deviding it by the mass of the particles. Therefore our function takes the density as input instead of the Volume.

```
def mu(rho: np.ndarray, T: float, m: float) -> np.ndarray:
    """Calculate the chemical potential.

    The chemical potential is calculated from the density: mu = R T log(rho. / m)
    """
    # RT in KJ/mol
    RT = T * const.Boltzmann * const.Avogadro / const.kilo

    # De Broglie (converted to angstrom)
    db = (
        np.sqrt(const.h**2 / (2 * np.pi * m * const.atomic_mass * const.Boltzmann * T))
        / const.angstrom
    )

    if np.all(rho > 0):
        return RT * np.log(rho * db**3)
    elif np.any(rho == 0):
        return np.float64("-inf") * np.ones(rho.shape)
    else:
        return np.float64("nan") * np.ones(rho.shape)
```

If you're also interested in the error of the chemical potential we can calculate it through propagation of uncertainty

from the error of the density, calculated by MAICoS. The error propagates according to

$$\begin{aligned}\Delta\mu &= \left|\frac{\partial\mu}{\partial\rho}\right| \cdot \Delta\rho \\ &= \frac{RT}{\rho} \cdot \Delta\rho.\end{aligned}$$

The implemented function looks like this.

```
def dmu(rho: np.ndarray, drho: np.ndarray, T: float) -> np.ndarray:
    """Calculate the error of the chemical potential.

    The error is calculated from the density using propagation of uncertainty.
    """
    RT = T * const.Boltzmann * const.Avogadro / const.kilo

    if np.all(rho > 0):
        return RT * (drho / rho)
    else:
        return np.float64("nan") * np.ones(rho.shape)
```

Finally, we can use those previously defined functions to calculate the chemical potential and its error for an example trajectory called *water*, whose data can be downloaded from `topology` and `trajectory`. To calculate the mean density we use the module *maicos.DensityPlanar* of MAICoS. This example uses a temperature of 300 K and a mass of 18 u.

```
water = mda.Universe("water.tpr", "water.trr")
ana = maicos.DensityPlanar(water.atoms)
ana.run()
print("μ_id =", mu(ana.results.profile.mean(), 300, 18))
print("μ_id =", dmu(ana.results.profile.mean(), ana.results.dprofile.mean(), 300))
```

```
Unwrapping in combination with the `wrap_compound='atoms` is superfluous. `unwrap` will␣
↪be set to `False`.
/home/docs/checkouts/readthedocs.org/user_builds/maicos/envs/main/lib/python3.9/site-
↪packages/maicos/core/base.py:456: UserWarning: Your data seems to be correlated with a␣
↪correlation time which is 2.19 times larger than your step size. Consider increasing␣
↪your step size by a factor of 4 to get a reasonable error estimate.
  self.corrtime = correlation_analysis(self.timeseries)
μ_id = -12.050277646766348
μ_id = 0.041243491788978626
```

References

5.2.3 Dielectric profile calculation

Basic usage

In the following example, we will show how to calculate the dielectric profiles as described in *Dielectric constant measurement*.

Before producing trajectories to calculate dielectric profiles, you will need to consider which information you will need and thus need to print out. The dielectric profile calculators need unwrapped positions and charges of **all** charged

atoms in the system. Unwrapped refers to the fact that you will need either "repaired" molecules (which in GROMACS `trjconv` with the `-pbc mol` option can do for you) or you will need to provide topology information for MAICoS to repair molecules for you during the analysis. Note, however, that unwrapping adds overhead to your calculations. Therefore, it is recommended to use a repaired trajectory if possible.

In the following, we will give an example of a trajectory of water confined by graphene sheets simulated via GROMACS. We assume that the GROMACS topology is given by *graphene_water.tpr* and the trajectory is given by *graphene_water.xtc*. Both can be downloaded under `topology` and `trajectory`, respectively.

From these files you can create a MDAnalysis universe object.

```
import matplotlib.pyplot as plt
import MDAnalysis as mda
import numpy as np

import maicos


u = mda.Universe("graphene_water.tpr", "graphene_water.xtc")
```

This universe object can then be passed to the dielectric profile analysis object, documented in *maicos.DielectricPlanar*. It expects you to pass the atom groups you want to perform the analysis for. In our example, we have graphene walls and SPC/E water confined between them, where we are interested in the dielectric behavior of the fluid. Thus, we will first select the water as an MDAnalysis atom group using `MDAnalysis.core.groups.AtomGroup.select_atoms()`. In this case we select the water by filtering for the residue named `SOL`.

According to the discussion above, we use an unwrapped trajectory and set the `unwrap = False` keyword.

The simulation trajectory that we provide was simulated using Yeh-Berkowitz dipole correction. So we don't want to include dipole corrections, because we assume that our simulation data adequately represents a 2d-periodic system. For systems that are not 2d-periodic, one should set the `is_3d` argument to `True` to include the dipole correction (see *Dielectric constant measurement* or the section on boundary conditions down below).

Since we included a large vacuum region in our simulation that is not of interest for the dielectric profile, we can set the `refgroup` to the group containing our water molecules. This will calculate the dielectric profile relative to the center of mass of the water in the region of interest.

```
water = u.select_atoms("resname SOL")
# Create the analysis object with the appropriate parameters.
analysis_obj = maicos.DielectricPlanar(water, bin_width=0.1, refgroup=water)
```

This creates the analysis object, but does not yet perform the analysis. To this end we call the member function `run`. We may set the `verbose` keyword to `True` to get additional information like a progress bar.

Here you also have the chance to set `start` and `stop` keywords to specify which frames the analysis should start at and where to end. One can also specify a `step` keyword to only analyze every `step` frames.

```
analysis_obj.run(step=5)
```

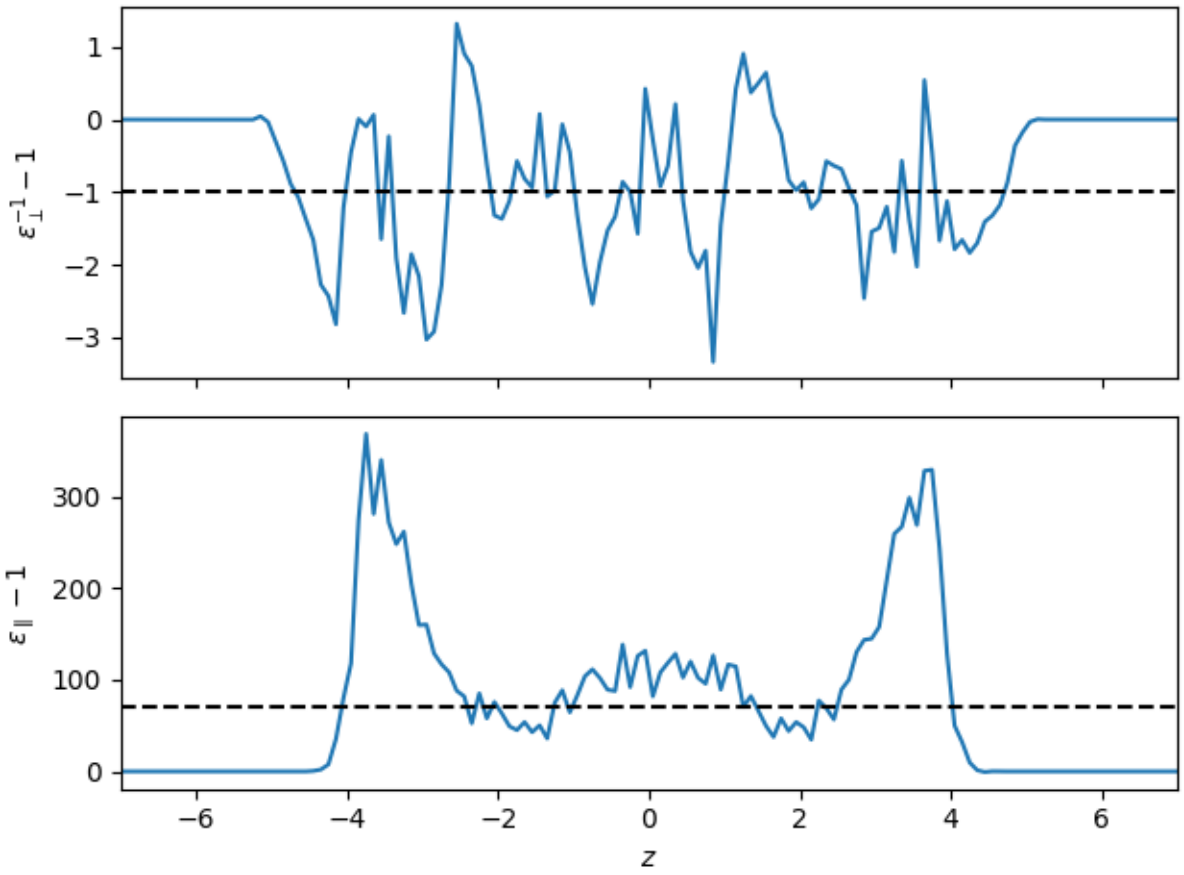
```
/home/docs/checkouts/readthedocs.org/user_builds/maicos/envs/main/lib/python3.9/site-
↪packages/maicos/core/base.py:456: UserWarning: Your data seems to be correlated with a␣
↪correlation time which is 2.55 times larger than your step size. Consider increasing␣
↪your step size by a factor of 5 to get a reasonable error estimate.
  self.corrtime = correlation_analysis(self.timeseries)

<maicos.modules.dielectricplanar.DielectricPlanar object at 0x7fad79bdfe50>
```

Here we use `step = 5` to run a fast analysis. You may reduce the `step` parameter to gain a higher accuracy. Note that the analysis issues a warning concerning the correlation time of the trajectory, which is automatically calculated as an indication of how far apart the frames should be chosen to get a statistically uncertainty indicator estimate. For small trajectories such as the one in this example, this estimate is not very reliable and one should perform the analysis for longer trajectories for actual production runs.

Hence, we will ignore the warning for the purpose of this example. Now we are ready to plot the results. MAICoS provides the outcome of the calculation as sub-attributes of the `results` attribute of the analysis object. The results object contains several attributes that can be accessed directly. For example, the bin positions are stored in the `bin_pos` attribute, the parallel and perpendicular dielectric profiles in the `eps_par` and `eps_perp` attributes respectively. (See *maicos.DielectricPlanar* for a full list of attributes.)

For this example, we plot both profiles using matplotlib. Note that MAICoS always centers the system at the origin or the selected refgroup, so here we set the limits of the x-axis to [-7, 7]. Then we can only show the relevant part of the output (the system has a width of 14 Å).

```
fig, ax = plt.subplots(2, sharex=True)

z = analysis_obj.results.bin_pos

ax[0].plot(z, analysis_obj.results.eps_perp)
ax[1].plot(z, analysis_obj.results.eps_par)

ax[0].set_ylabel(r"$\varepsilon^{-1}_{\perp} - 1$")
ax[1].set_ylabel(r"$\varepsilon_{\parallel} - 1$")

ax[1].set_xlabel(r"$z$")

# Only plot the actual physical system:
ax[0].set_xlim([-7, 7])
ax[1].set_xlim([-7, 7])

# Also plot the bulk values for reference
ax[0].axhline(1 / 71 - 1, color="black", linestyle="dashed")
ax[1].axhline(71 - 1, color="black", linestyle="dashed")

fig.tight_layout()
plt.show()
```



A few notes on the results: The perpendicular component is given as the inverse of the dielectric profile, which is the "natural" output (see *Dielectric constant measurement* for more details). Furthermore, the bulk values expected for the SPC/E water model are given as reference lines.

Notice that the parallel component is better converged than the perpendicular component which in this very short trajectory is still noisy. For trajectories with a duration of about 1 microsecond, the perpendicular component can be expected to be converged.

Boundary Conditions

(See *Dielectric constant measurement* for a thorough discussion of the boundary conditions). Here we only note that the `is_3d` flag has to be chosen carefully, depending on if one simulated a truly 3d periodic system or a 2d periodic one. Seldomly, vacuum boundary conditions might have been used for Ewald summations instead of the more common tin-foil boundary conditions. In this case, the `vac` flag should be set to `True`.

TIP4P Water and Molecules with Virtual Sites

One has to be careful when using the dielectric profile analysis for systems with virtual sites, such as TIP4P water. The reason is that the virtual sites might not be included in the trajectory, but instead are only constructed by the MD engine during the force calculation. (For example some LAMMPS fixes)

This problem can be circumvented by creating the virtual sites by hand. This is done by creating a transformation function that is added to the universe. This function is called for every frame and can be used to create the virtual sites. The following example shows how to do this for TIP4P/ water from a LAMMPS trajectory. Here we only shift the oxygen charge along the H-O-H angle bisector by a distance of 0.105 Å, which is the distance between the oxygen charge and the virtual site in the TIP4P/ water model.

```
def transform_lammps_tip4p(
    oxygen_index_array: np.ndarray, distance: float
) -> mda.coordinates.timestep.Timestep:
    """Creates a transformation function where for lammps tip4p molecukes.

    oxygen_index_array is the array of indices where ``atom.type == oxygen_type``.
    I.e. given by ``np.where(universe.atoms.types == oxygen_type)[0]``.

    ``distance`` defines by how much the oxygen is moved in the H-O-H plane.
    """

    def wrapped(timestep):
        # shift oxygen charge in case of tip4p
        this_pos = timestep.positions
        for j in oxygen_index_array:
            # -2 * vec_o + vec_h1 + vec_h2
            vec = np.dot(np.array([-2, 1, 1]), this_pos[j : j + 3, :])
            unit_vec = vec / np.linalg.norm(vec)
            this_pos[j] += unit_vec * distance
        return timestep

    return wrapped


oxygen_index_array = u.select_atoms("type 2").indices

shift_tip4p_lammps = transform_lammps_tip4p(oxygen_index_array, 0.105)

u.trajectory.add_transformations(shift_tip4p_lammps)
```

Preliminary Output

As the dielectric analysis is usually run for long trajectories, analysis can take a while. Hence, it is useful to get some preliminary output to see how the analysis is progressing. Use the `concfreq` keyword to specify how often the analysis should output the current results into data files on the disk. The `concfreq` keyword is given in units of frames. For example, if `concfreq = 100`, the analysis will output the current results to the data files every 100 frames.

5.2.4 Pair distribution functions

Basic usage

In the following example, we will show how to calculate the two-dimensional planar pair distribution functions.

In the following, we will give an example of a trajectory of water confined by graphene sheets simulated via GROMACS. We assume that the GROMACS topology is given by *graphene_water.tpr* and the trajectory is given by *graphene_water.xtc*. Both can be downloaded under `topology` and `trajectory`, respectively.

From these files you can create a MDAnalysis universe object.

We begin by importing the necessary modules.

```
import matplotlib.pyplot as plt
import MDAnalysis as mda
import numpy as np

import maicos
```

Next, we proceed with the creation of a MDAnalysis universe object, from which we further select the water molecules using the *resname* selector.

```
u = mda.Universe("./graphene_water.tpr", "graphene_water.xtc")
```

This universe object can then be passed to *maicos.modules.pdfplanar.PDFPlanar* analysis object. It expects you to pass the atom groups you want to perform the analysis for. In our example, we have graphene walls and SPC/E water confined between them, where we are interested in the dielectric behavior of the fluid. Thus, we will first select the water as an MDAnalysis atom group using `MDAnalysis.core.groups.AtomGroup.select_atoms()`. In this case we select the water by filtering for the residue named `SOL`.

```
water = u.select_atoms("resname SOL")

ana_obj = maicos.PDFPlanar(
    water,
    water,
    dzheight=0.25,
    dim=2,
    pdf_bin_width=0.2,
    refgroup=water,
    zmin=-5.0,
    zmax=0,
)
```

Next, we can run the analysis over the trajectory. To this end we call the member function `run`. We may set the `verbose` keyword to `True` to get additional information such a a progress bar.

Here you also have the chance to set `start` and `stop` keywords to specify which frames the analysis should start at and where to end. One can also specify a `step` keyword to only analyze every `step` frames.

```
ana_obj.run(verbose=True, step=1)
```

```
  0%|          | 0/2001 [00:00<?, ?it/s]
  0%|          | 3/2001 [00:00<01:10, 28.26it/s]
  0%|          | 6/2001 [00:00<01:10, 28.37it/s]
  0%|          | 9/2001 [00:00<01:10, 28.33it/s]
  1%|          | 12/2001 [00:00<01:10, 28.29it/s]
  1%|          | 15/2001 [00:00<01:10, 28.10it/s]
  1%|          | 18/2001 [00:00<01:10, 28.09it/s]
  1%|          | 21/2001 [00:00<01:10, 28.11it/s]
  1%|          | 24/2001 [00:00<01:10, 28.05it/s]
  1%|          | 27/2001 [00:00<01:10, 28.03it/s]
  1%|          | 30/2001 [00:01<01:10, 28.01it/s]
  2%|          | 33/2001 [00:01<01:09, 28.12it/s]
  2%|          | 36/2001 [00:01<01:10, 28.06it/s]
  2%|          | 39/2001 [00:01<01:09, 28.08it/s]
  2%|          | 42/2001 [00:01<01:09, 28.07it/s]
  2%|          | 45/2001 [00:01<01:10, 27.82it/s]
  2%|          | 48/2001 [00:01<01:09, 27.95it/s]
  3%|          | 51/2001 [00:01<01:09, 27.98it/s]
  3%|          | 54/2001 [00:01<01:09, 27.94it/s]
  3%|          | 57/2001 [00:02<01:09, 27.95it/s]
  3%|          | 60/2001 [00:02<01:09, 28.05it/s]
  3%|          | 63/2001 [00:02<01:09, 28.02it/s]
  3%|          | 66/2001 [00:02<01:08, 28.06it/s]
  3%|          | 69/2001 [00:02<01:08, 28.02it/s]
  4%|          | 72/2001 [00:02<01:08, 28.05it/s]
  4%|          | 75/2001 [00:02<01:08, 28.00it/s]
  4%|          | 78/2001 [00:02<01:08, 27.96it/s]
  4%|          | 81/2001 [00:02<01:08, 27.95it/s]
  4%|          | 84/2001 [00:02<01:08, 27.92it/s]
  4%|          | 87/2001 [00:03<01:08, 27.95it/s]
  4%|          | 90/2001 [00:03<01:08, 27.95it/s]
  5%|          | 93/2001 [00:03<01:08, 27.97it/s]
  5%|          | 96/2001 [00:03<01:08, 27.97it/s]
  5%|          | 99/2001 [00:03<01:07, 28.03it/s]
  5%|          | 102/2001 [00:03<01:07, 28.10it/s]
  5%|          | 105/2001 [00:03<01:07, 28.14it/s]
  5%|          | 108/2001 [00:03<01:07, 28.02it/s]
  6%|          | 111/2001 [00:03<01:07, 28.06it/s]
  6%|          | 114/2001 [00:04<01:07, 28.00it/s]
  6%|          | 117/2001 [00:04<01:07, 28.05it/s]
  6%|          | 120/2001 [00:04<01:07, 28.07it/s]
  6%|          | 123/2001 [00:04<01:06, 28.04it/s]
  6%|          | 126/2001 [00:04<01:06, 27.99it/s]
  6%|          | 129/2001 [00:04<01:06, 27.96it/s]
  7%|          | 132/2001 [00:04<01:06, 28.06it/s]
  7%|          | 135/2001 [00:04<01:06, 28.08it/s]
  7%|          | 138/2001 [00:04<01:06, 28.05it/s]
  7%|          | 141/2001 [00:05<01:06, 28.04it/s]
  7%|          | 144/2001 [00:05<01:06, 28.08it/s]
  7%|          | 147/2001 [00:05<01:06, 28.06it/s]
```

(continues on next page)

(continued from previous page)

```
  7%|         | 150/2001 [00:05<01:05, 28.08it/s]
  8%|         | 153/2001 [00:05<01:05, 28.05it/s]
  8%|         | 156/2001 [00:05<01:05, 28.08it/s]
  8%|         | 159/2001 [00:05<01:05, 28.08it/s]
  8%|         | 162/2001 [00:05<01:05, 28.14it/s]
  8%|         | 165/2001 [00:05<01:05, 28.18it/s]
  8%|         | 168/2001 [00:05<01:04, 28.28it/s]
  9%|         | 171/2001 [00:06<01:04, 28.17it/s]
  9%|         | 174/2001 [00:06<01:04, 28.19it/s]
  9%|         | 177/2001 [00:06<01:04, 28.15it/s]
  9%|         | 180/2001 [00:06<01:04, 28.16it/s]
  9%|         | 183/2001 [00:06<01:04, 28.03it/s]
  9%|         | 186/2001 [00:06<01:04, 28.13it/s]
  9%|         | 189/2001 [00:06<01:04, 28.12it/s]
 10%|         | 192/2001 [00:06<01:04, 28.05it/s]
 10%|         | 195/2001 [00:06<01:04, 27.96it/s]
 10%|         | 198/2001 [00:07<01:04, 27.98it/s]
 10%|         | 201/2001 [00:07<01:04, 28.04it/s]
 10%|         | 204/2001 [00:07<01:03, 28.11it/s]
 10%|         | 207/2001 [00:07<01:03, 28.13it/s]
 10%|         | 210/2001 [00:07<01:03, 28.11it/s]
 11%|         | 213/2001 [00:07<01:03, 28.02it/s]
 11%|         | 216/2001 [00:07<01:03, 27.92it/s]
 11%|         | 219/2001 [00:07<01:03, 27.96it/s]
 11%|         | 222/2001 [00:07<01:03, 28.02it/s]
 11%|         | 225/2001 [00:08<01:03, 28.11it/s]
 11%|        | 228/2001 [00:08<01:02, 28.19it/s]
 12%|        | 231/2001 [00:08<01:02, 28.15it/s]
 12%|        | 234/2001 [00:08<01:02, 28.17it/s]
 12%|        | 237/2001 [00:08<01:02, 28.07it/s]
 12%|        | 240/2001 [00:08<01:02, 28.07it/s]
 12%|        | 243/2001 [00:08<01:02, 28.05it/s]
 12%|        | 246/2001 [00:08<01:02, 28.06it/s]
 12%|        | 249/2001 [00:08<01:02, 27.98it/s]
 13%|        | 252/2001 [00:08<01:02, 27.89it/s]
 13%|        | 255/2001 [00:09<01:02, 28.02it/s]
 13%|        | 258/2001 [00:09<01:02, 28.07it/s]
 13%|        | 261/2001 [00:09<01:01, 28.10it/s]
 13%|        | 264/2001 [00:09<01:01, 28.10it/s]
 13%|        | 267/2001 [00:09<01:01, 28.10it/s]
 13%|        | 270/2001 [00:09<01:01, 28.18it/s]
 14%|        | 273/2001 [00:09<01:01, 28.07it/s]
 14%|        | 276/2001 [00:09<01:01, 28.12it/s]
 14%|        | 279/2001 [00:09<01:01, 28.19it/s]
 14%|        | 282/2001 [00:10<01:01, 28.13it/s]
 14%|        | 285/2001 [00:10<01:00, 28.14it/s]
 14%|        | 288/2001 [00:10<01:01, 28.07it/s]
 15%|        | 291/2001 [00:10<01:01, 27.97it/s]
 15%|        | 294/2001 [00:10<01:01, 27.96it/s]
 15%|        | 297/2001 [00:10<01:00, 28.05it/s]
 15%|        | 300/2001 [00:10<01:00, 27.99it/s]
 15%|        | 303/2001 [00:10<01:00, 27.94it/s]
```

(continues on next page)

(continued from previous page)

```
15%|        | 306/2001 [00:10<01:00, 27.94it/s]
15%|        | 309/2001 [00:11<01:00, 27.90it/s]
16%|        | 312/2001 [00:11<01:00, 27.92it/s]
16%|        | 315/2001 [00:11<01:00, 28.01it/s]
16%|        | 318/2001 [00:11<00:59, 28.09it/s]
16%|        | 321/2001 [00:11<00:59, 28.17it/s]
16%|        | 324/2001 [00:11<00:59, 28.27it/s]
16%|        | 327/2001 [00:11<00:59, 28.27it/s]
16%|        | 330/2001 [00:11<00:59, 28.31it/s]
17%|        | 333/2001 [00:11<00:58, 28.32it/s]
17%|        | 336/2001 [00:11<00:58, 28.30it/s]
17%|        | 339/2001 [00:12<00:58, 28.28it/s]
17%|        | 342/2001 [00:12<00:58, 28.32it/s]
17%|        | 345/2001 [00:12<00:58, 28.31it/s]
17%|        | 348/2001 [00:12<00:58, 28.22it/s]
18%|        | 351/2001 [00:12<00:58, 28.16it/s]
18%|        | 354/2001 [00:12<00:58, 28.11it/s]
18%|        | 357/2001 [00:12<00:58, 28.13it/s]
18%|        | 360/2001 [00:12<00:58, 28.09it/s]
18%|        | 363/2001 [00:12<00:58, 28.07it/s]
18%|        | 366/2001 [00:13<00:58, 28.03it/s]
18%|        | 369/2001 [00:13<00:58, 28.05it/s]
19%|        | 372/2001 [00:13<00:57, 28.17it/s]
19%|        | 375/2001 [00:13<00:57, 28.13it/s]
19%|        | 378/2001 [00:13<00:57, 28.17it/s]
19%|        | 381/2001 [00:13<00:57, 28.20it/s]
19%|        | 384/2001 [00:13<00:57, 28.12it/s]
19%|        | 387/2001 [00:13<00:57, 28.18it/s]
19%|        | 390/2001 [00:13<00:56, 28.28it/s]
20%|        | 393/2001 [00:13<00:56, 28.35it/s]
20%|        | 396/2001 [00:14<00:56, 28.29it/s]
20%|        | 399/2001 [00:14<00:56, 28.28it/s]
20%|        | 402/2001 [00:14<00:56, 28.31it/s]
20%|        | 405/2001 [00:14<00:56, 28.25it/s]
20%|        | 408/2001 [00:14<00:56, 28.18it/s]
21%|        | 411/2001 [00:14<00:56, 28.19it/s]
21%|        | 414/2001 [00:14<00:56, 28.20it/s]
21%|        | 417/2001 [00:14<00:56, 28.23it/s]
21%|        | 420/2001 [00:14<00:56, 28.07it/s]
21%|        | 423/2001 [00:15<00:56, 28.00it/s]
21%|       | 426/2001 [00:15<00:56, 28.04it/s]
21%|       | 429/2001 [00:15<00:56, 28.03it/s]
22%|       | 432/2001 [00:15<00:55, 28.06it/s]
22%|       | 435/2001 [00:15<00:55, 28.10it/s]
22%|       | 438/2001 [00:15<00:55, 28.07it/s]
22%|       | 441/2001 [00:15<00:55, 28.10it/s]
22%|       | 444/2001 [00:15<00:55, 28.17it/s]
22%|       | 447/2001 [00:15<00:57, 27.14it/s]
22%|       | 450/2001 [00:16<00:57, 26.96it/s]
23%|       | 453/2001 [00:16<00:56, 27.36it/s]
23%|       | 456/2001 [00:16<00:55, 27.61it/s]
23%|       | 459/2001 [00:16<00:55, 27.83it/s]
```

(continues on next page)

(continued from previous page)

```
23%|       | 462/2001 [00:16<00:55, 27.96it/s]
23%|       | 465/2001 [00:16<00:54, 28.05it/s]
23%|       | 468/2001 [00:16<00:54, 28.07it/s]
24%|       | 471/2001 [00:16<00:54, 28.05it/s]
24%|       | 474/2001 [00:16<00:54, 27.99it/s]
24%|       | 477/2001 [00:17<00:54, 27.79it/s]
24%|       | 480/2001 [00:17<00:55, 27.61it/s]
24%|       | 483/2001 [00:17<00:54, 27.82it/s]
24%|       | 486/2001 [00:17<00:54, 27.98it/s]
24%|       | 489/2001 [00:17<00:54, 27.91it/s]
25%|       | 492/2001 [00:17<00:53, 27.96it/s]
25%|       | 495/2001 [00:17<00:53, 28.08it/s]
25%|       | 498/2001 [00:17<00:53, 28.24it/s]
25%|       | 501/2001 [00:17<00:53, 28.15it/s]
25%|       | 504/2001 [00:17<00:53, 28.18it/s]
25%|       | 507/2001 [00:18<00:52, 28.22it/s]
25%|       | 510/2001 [00:18<00:52, 28.27it/s]
26%|       | 513/2001 [00:18<00:52, 28.25it/s]
26%|       | 516/2001 [00:18<00:52, 28.28it/s]
26%|       | 519/2001 [00:18<00:52, 28.17it/s]
26%|       | 522/2001 [00:18<00:52, 28.16it/s]
26%|       | 525/2001 [00:18<00:52, 28.10it/s]
26%|       | 528/2001 [00:18<00:52, 28.00it/s]
27%|       | 531/2001 [00:18<00:52, 28.02it/s]
27%|       | 534/2001 [00:19<00:52, 28.01it/s]
27%|       | 537/2001 [00:19<00:52, 28.03it/s]
27%|       | 540/2001 [00:19<00:51, 28.13it/s]
27%|       | 543/2001 [00:19<00:51, 28.18it/s]
27%|       | 546/2001 [00:19<00:51, 28.17it/s]
27%|       | 549/2001 [00:19<00:51, 28.19it/s]
28%|       | 552/2001 [00:19<00:51, 28.20it/s]
28%|       | 555/2001 [00:19<00:51, 28.15it/s]
28%|       | 558/2001 [00:19<00:51, 28.15it/s]
28%|       | 561/2001 [00:19<00:51, 28.16it/s]
28%|       | 564/2001 [00:20<00:50, 28.19it/s]
28%|       | 567/2001 [00:20<00:50, 28.22it/s]
28%|       | 570/2001 [00:20<00:50, 28.30it/s]
29%|       | 573/2001 [00:20<00:50, 28.21it/s]
29%|       | 576/2001 [00:20<00:50, 28.13it/s]
29%|       | 579/2001 [00:20<00:50, 28.10it/s]
29%|       | 582/2001 [00:20<00:50, 28.14it/s]
29%|       | 585/2001 [00:20<00:50, 28.02it/s]
29%|       | 588/2001 [00:20<00:50, 27.93it/s]
30%|       | 591/2001 [00:21<00:50, 27.88it/s]
30%|       | 594/2001 [00:21<00:50, 28.05it/s]
30%|       | 597/2001 [00:21<00:51, 27.13it/s]
30%|       | 600/2001 [00:21<00:52, 26.61it/s]
30%|       | 603/2001 [00:21<00:53, 26.22it/s]
30%|       | 606/2001 [00:21<00:53, 25.92it/s]
30%|       | 609/2001 [00:21<00:52, 26.51it/s]
31%|       | 612/2001 [00:21<00:51, 26.93it/s]
31%|       | 615/2001 [00:21<00:50, 27.29it/s]
```

(continues on next page)

(continued from previous page)

```
31%|       | 618/2001 [00:22<00:50, 27.49it/s]
31%|       | 621/2001 [00:22<00:49, 27.72it/s]
31%|       | 624/2001 [00:22<00:49, 27.86it/s]
31%|      | 627/2001 [00:22<00:49, 27.96it/s]
31%|      | 630/2001 [00:22<00:49, 27.94it/s]
32%|      | 633/2001 [00:22<00:48, 27.97it/s]
32%|      | 636/2001 [00:22<00:48, 27.95it/s]
32%|      | 639/2001 [00:22<00:48, 27.94it/s]
32%|      | 642/2001 [00:22<00:48, 27.99it/s]
32%|      | 645/2001 [00:23<00:48, 27.99it/s]
32%|      | 648/2001 [00:23<00:48, 28.06it/s]
33%|      | 651/2001 [00:23<00:48, 28.11it/s]
33%|      | 654/2001 [00:23<00:47, 28.17it/s]
33%|      | 657/2001 [00:23<00:47, 28.22it/s]
33%|      | 660/2001 [00:23<00:47, 28.19it/s]
33%|      | 663/2001 [00:23<00:47, 28.22it/s]
33%|      | 666/2001 [00:23<00:47, 28.20it/s]
33%|      | 669/2001 [00:23<00:47, 28.22it/s]
34%|      | 672/2001 [00:23<00:48, 27.51it/s]
34%|      | 675/2001 [00:24<00:47, 27.68it/s]
34%|      | 678/2001 [00:24<00:47, 27.83it/s]
34%|      | 681/2001 [00:24<00:47, 27.87it/s]
34%|      | 684/2001 [00:24<00:47, 27.92it/s]
34%|      | 687/2001 [00:24<00:46, 28.04it/s]
34%|      | 690/2001 [00:24<00:46, 28.14it/s]
35%|      | 693/2001 [00:24<00:46, 28.13it/s]
35%|      | 696/2001 [00:24<00:46, 28.08it/s]
35%|      | 699/2001 [00:24<00:46, 28.04it/s]
35%|      | 702/2001 [00:25<00:46, 27.94it/s]
35%|      | 705/2001 [00:25<00:46, 28.04it/s]
35%|      | 708/2001 [00:25<00:46, 28.03it/s]
36%|      | 711/2001 [00:25<00:45, 28.11it/s]
36%|      | 714/2001 [00:25<00:45, 28.11it/s]
36%|      | 717/2001 [00:25<00:45, 28.11it/s]
36%|      | 720/2001 [00:25<00:45, 28.07it/s]
36%|      | 723/2001 [00:25<00:45, 28.01it/s]
36%|      | 726/2001 [00:25<00:45, 28.09it/s]
36%|      | 729/2001 [00:26<00:45, 28.13it/s]
37%|      | 732/2001 [00:26<00:45, 28.03it/s]
37%|      | 735/2001 [00:26<00:45, 28.10it/s]
37%|      | 738/2001 [00:26<00:45, 28.06it/s]
37%|      | 741/2001 [00:26<00:44, 28.17it/s]
37%|      | 744/2001 [00:26<00:44, 28.06it/s]
37%|      | 747/2001 [00:26<00:45, 27.63it/s]
37%|      | 750/2001 [00:26<00:45, 27.76it/s]
38%|      | 753/2001 [00:26<00:45, 27.18it/s]
38%|      | 756/2001 [00:27<00:46, 26.55it/s]
38%|      | 759/2001 [00:27<00:46, 26.53it/s]
38%|      | 762/2001 [00:27<00:46, 26.93it/s]
38%|      | 765/2001 [00:27<00:45, 27.29it/s]
38%|      | 768/2001 [00:27<00:44, 27.56it/s]
39%|      | 771/2001 [00:27<00:44, 27.81it/s]
```

(continues on next page)

(continued from previous page)

```
 39%|      | 774/2001 [00:27<00:44, 27.82it/s]
 39%|      | 777/2001 [00:27<00:43, 27.83it/s]
 39%|      | 780/2001 [00:27<00:43, 27.94it/s]
 39%|      | 783/2001 [00:27<00:43, 28.14it/s]
 39%|      | 786/2001 [00:28<00:43, 28.14it/s]
 39%|      | 789/2001 [00:28<00:42, 28.25it/s]
 40%|      | 792/2001 [00:28<00:42, 28.30it/s]
 40%|      | 795/2001 [00:28<00:42, 28.27it/s]
 40%|      | 798/2001 [00:28<00:42, 28.22it/s]
 40%|      | 801/2001 [00:28<00:42, 28.10it/s]
 40%|      | 804/2001 [00:28<00:42, 28.15it/s]
 40%|      | 807/2001 [00:28<00:42, 28.14it/s]
 40%|      | 810/2001 [00:28<00:42, 28.09it/s]
 41%|      | 813/2001 [00:29<00:42, 28.09it/s]
 41%|      | 816/2001 [00:29<00:42, 28.17it/s]
 41%|      | 819/2001 [00:29<00:41, 28.18it/s]
 41%|      | 822/2001 [00:29<00:41, 28.26it/s]
 41%|      | 825/2001 [00:29<00:41, 28.30it/s]
 41%|     | 828/2001 [00:29<00:41, 28.21it/s]
 42%|     | 831/2001 [00:29<00:41, 28.15it/s]
 42%|     | 834/2001 [00:29<00:41, 28.15it/s]
 42%|     | 837/2001 [00:29<00:41, 28.20it/s]
 42%|     | 840/2001 [00:29<00:41, 28.28it/s]
 42%|     | 843/2001 [00:30<00:41, 28.22it/s]
 42%|     | 846/2001 [00:30<00:40, 28.18it/s]
 42%|     | 849/2001 [00:30<00:40, 28.14it/s]
 43%|     | 852/2001 [00:30<00:40, 28.03it/s]
 43%|     | 855/2001 [00:30<00:40, 28.06it/s]
 43%|     | 858/2001 [00:30<00:40, 28.10it/s]
 43%|     | 861/2001 [00:30<00:40, 28.10it/s]
 43%|     | 864/2001 [00:30<00:40, 28.05it/s]
 43%|     | 867/2001 [00:30<00:40, 27.98it/s]
 43%|     | 870/2001 [00:31<00:40, 27.90it/s]
 44%|     | 873/2001 [00:31<00:40, 28.02it/s]
 44%|     | 876/2001 [00:31<00:40, 28.03it/s]
 44%|     | 879/2001 [00:31<00:39, 28.13it/s]
 44%|     | 882/2001 [00:31<00:39, 28.20it/s]
 44%|     | 885/2001 [00:31<00:39, 28.30it/s]
 44%|     | 888/2001 [00:31<00:39, 28.29it/s]
 45%|     | 891/2001 [00:31<00:39, 28.32it/s]
 45%|     | 894/2001 [00:31<00:39, 28.28it/s]
 45%|     | 897/2001 [00:32<00:38, 28.37it/s]
 45%|     | 900/2001 [00:32<00:38, 28.36it/s]
 45%|     | 903/2001 [00:32<00:38, 28.26it/s]
 45%|     | 906/2001 [00:32<00:38, 28.19it/s]
 45%|     | 909/2001 [00:32<00:38, 28.15it/s]
 46%|     | 912/2001 [00:32<00:38, 28.20it/s]
 46%|     | 915/2001 [00:32<00:38, 28.21it/s]
 46%|     | 918/2001 [00:32<00:38, 28.24it/s]
 46%|     | 921/2001 [00:32<00:38, 28.20it/s]
 46%|     | 924/2001 [00:32<00:38, 28.08it/s]
 46%|     | 927/2001 [00:33<00:38, 28.00it/s]
```

(continues on next page)

(continued from previous page)

```
46%|     | 930/2001 [00:33<00:38, 28.01it/s]
47%|     | 933/2001 [00:33<00:38, 28.00it/s]
47%|     | 936/2001 [00:33<00:38, 28.02it/s]
47%|     | 939/2001 [00:33<00:37, 28.06it/s]
47%|     | 942/2001 [00:33<00:37, 28.18it/s]
47%|     | 945/2001 [00:33<00:37, 28.20it/s]
47%|     | 948/2001 [00:33<00:37, 28.26it/s]
48%|     | 951/2001 [00:33<00:37, 28.22it/s]
48%|     | 954/2001 [00:34<00:37, 28.17it/s]
48%|     | 957/2001 [00:34<00:37, 28.16it/s]
48%|     | 960/2001 [00:34<00:37, 28.09it/s]
48%|     | 963/2001 [00:34<00:36, 28.14it/s]
48%|     | 966/2001 [00:34<00:36, 28.09it/s]
48%|     | 969/2001 [00:34<00:36, 28.12it/s]
49%|     | 972/2001 [00:34<00:36, 28.16it/s]
49%|     | 975/2001 [00:34<00:36, 28.12it/s]
49%|     | 978/2001 [00:34<00:36, 28.11it/s]
49%|     | 981/2001 [00:35<00:36, 28.03it/s]
49%|     | 984/2001 [00:35<00:36, 27.99it/s]
49%|     | 987/2001 [00:35<00:36, 28.04it/s]
49%|     | 990/2001 [00:35<00:35, 28.11it/s]
50%|     | 993/2001 [00:35<00:35, 28.18it/s]
50%|     | 996/2001 [00:35<00:35, 28.20it/s]
50%|     | 999/2001 [00:35<00:35, 28.28it/s]
50%|     | 1002/2001 [00:35<00:35, 28.21it/s]
50%|     | 1005/2001 [00:35<00:35, 28.10it/s]
50%|     | 1008/2001 [00:35<00:35, 28.05it/s]
51%|     | 1011/2001 [00:36<00:35, 28.05it/s]
51%|     | 1014/2001 [00:36<00:35, 28.14it/s]
51%|     | 1017/2001 [00:36<00:35, 28.03it/s]
51%|     | 1020/2001 [00:36<00:34, 28.12it/s]
51%|     | 1023/2001 [00:36<00:34, 27.99it/s]
51%|    | 1026/2001 [00:36<00:34, 28.01it/s]
51%|    | 1029/2001 [00:36<00:34, 27.98it/s]
52%|    | 1032/2001 [00:36<00:34, 27.97it/s]
52%|    | 1035/2001 [00:36<00:34, 27.95it/s]
52%|    | 1038/2001 [00:37<00:34, 27.99it/s]
52%|    | 1041/2001 [00:37<00:34, 28.05it/s]
52%|    | 1044/2001 [00:37<00:33, 28.23it/s]
52%|    | 1047/2001 [00:37<00:33, 28.27it/s]
52%|    | 1050/2001 [00:37<00:33, 28.25it/s]
53%|    | 1053/2001 [00:37<00:33, 28.19it/s]
53%|    | 1056/2001 [00:37<00:33, 28.17it/s]
53%|    | 1059/2001 [00:37<00:33, 28.25it/s]
53%|    | 1062/2001 [00:37<00:33, 28.27it/s]
53%|    | 1065/2001 [00:37<00:33, 28.28it/s]
53%|    | 1068/2001 [00:38<00:32, 28.39it/s]
54%|    | 1071/2001 [00:38<00:32, 28.44it/s]
54%|    | 1074/2001 [00:38<00:32, 28.47it/s]
54%|    | 1077/2001 [00:38<00:32, 28.41it/s]
54%|    | 1080/2001 [00:38<00:32, 28.23it/s]
54%|    | 1083/2001 [00:38<00:32, 28.22it/s]
```

(continues on next page)

(continued from previous page)

```
54%|    | 1086/2001 [00:38<00:32, 28.11it/s]
54%|    | 1089/2001 [00:38<00:32, 28.04it/s]
55%|    | 1092/2001 [00:38<00:32, 28.03it/s]
55%|    | 1095/2001 [00:39<00:32, 27.97it/s]
55%|    | 1098/2001 [00:39<00:32, 28.00it/s]
55%|    | 1101/2001 [00:39<00:32, 28.06it/s]
55%|    | 1104/2001 [00:39<00:31, 28.18it/s]
55%|    | 1107/2001 [00:39<00:31, 28.24it/s]
55%|    | 1110/2001 [00:39<00:31, 28.17it/s]
56%|    | 1113/2001 [00:39<00:31, 28.07it/s]
56%|    | 1116/2001 [00:39<00:31, 28.25it/s]
56%|    | 1119/2001 [00:39<00:31, 28.26it/s]
56%|    | 1122/2001 [00:40<00:31, 28.31it/s]
56%|    | 1125/2001 [00:40<00:30, 28.27it/s]
56%|    | 1128/2001 [00:40<00:31, 28.12it/s]
57%|    | 1131/2001 [00:40<00:30, 28.12it/s]
57%|    | 1134/2001 [00:40<00:30, 28.14it/s]
57%|    | 1137/2001 [00:40<00:30, 28.05it/s]
57%|    | 1140/2001 [00:40<00:30, 28.10it/s]
57%|    | 1143/2001 [00:40<00:30, 28.11it/s]
57%|    | 1146/2001 [00:40<00:30, 28.07it/s]
57%|    | 1149/2001 [00:40<00:30, 28.01it/s]
58%|    | 1152/2001 [00:41<00:30, 27.95it/s]
58%|    | 1155/2001 [00:41<00:30, 28.01it/s]
58%|    | 1158/2001 [00:41<00:29, 28.14it/s]
58%|    | 1161/2001 [00:41<00:29, 28.19it/s]
58%|    | 1164/2001 [00:41<00:29, 28.15it/s]
58%|    | 1167/2001 [00:41<00:29, 28.11it/s]
58%|    | 1170/2001 [00:41<00:29, 28.11it/s]
59%|    | 1173/2001 [00:41<00:29, 28.20it/s]
59%|    | 1176/2001 [00:41<00:29, 28.16it/s]
59%|    | 1179/2001 [00:42<00:29, 28.17it/s]
59%|    | 1182/2001 [00:42<00:29, 28.11it/s]
59%|    | 1185/2001 [00:42<00:29, 28.10it/s]
59%|    | 1188/2001 [00:42<00:28, 28.12it/s]
60%|    | 1191/2001 [00:42<00:29, 27.76it/s]
60%|    | 1194/2001 [00:42<00:28, 27.86it/s]
60%|    | 1197/2001 [00:42<00:28, 27.91it/s]
60%|    | 1200/2001 [00:42<00:28, 27.89it/s]
60%|    | 1203/2001 [00:42<00:28, 27.82it/s]
60%|    | 1206/2001 [00:43<00:28, 27.76it/s]
60%|    | 1209/2001 [00:43<00:28, 27.86it/s]
61%|    | 1212/2001 [00:43<00:28, 28.01it/s]
61%|    | 1215/2001 [00:43<00:27, 28.08it/s]
61%|    | 1218/2001 [00:43<00:27, 28.18it/s]
61%|    | 1221/2001 [00:43<00:27, 28.29it/s]
61%|    | 1224/2001 [00:43<00:27, 28.30it/s]
61%|   | 1227/2001 [00:43<00:27, 28.22it/s]
61%|   | 1230/2001 [00:43<00:27, 28.20it/s]
62%|   | 1233/2001 [00:43<00:27, 28.19it/s]
62%|   | 1236/2001 [00:44<00:27, 28.24it/s]
62%|   | 1239/2001 [00:44<00:26, 28.30it/s]
```

(continues on next page)

(continued from previous page)

```
62%|   | 1242/2001 [00:44<00:26, 28.41it/s]
62%|   | 1245/2001 [00:44<00:26, 28.32it/s]
62%|   | 1248/2001 [00:44<00:26, 28.19it/s]
63%|   | 1251/2001 [00:44<00:26, 28.22it/s]
63%|   | 1254/2001 [00:44<00:26, 28.22it/s]
63%|   | 1257/2001 [00:44<00:26, 28.20it/s]
63%|   | 1260/2001 [00:44<00:26, 28.18it/s]
63%|   | 1263/2001 [00:45<00:26, 27.96it/s]
63%|   | 1266/2001 [00:45<00:26, 27.90it/s]
63%|   | 1269/2001 [00:45<00:26, 28.03it/s]
64%|   | 1272/2001 [00:45<00:25, 28.14it/s]
64%|   | 1275/2001 [00:45<00:25, 28.18it/s]
64%|   | 1278/2001 [00:45<00:25, 28.18it/s]
64%|   | 1281/2001 [00:45<00:25, 28.17it/s]
64%|   | 1284/2001 [00:45<00:25, 28.20it/s]
64%|   | 1287/2001 [00:45<00:25, 28.26it/s]
64%|   | 1290/2001 [00:45<00:25, 28.22it/s]
65%|   | 1293/2001 [00:46<00:25, 28.16it/s]
65%|   | 1296/2001 [00:46<00:25, 28.19it/s]
65%|   | 1299/2001 [00:46<00:24, 28.18it/s]
65%|   | 1302/2001 [00:46<00:24, 28.09it/s]
65%|   | 1305/2001 [00:46<00:24, 27.94it/s]
65%|   | 1308/2001 [00:46<00:24, 27.98it/s]
66%|   | 1311/2001 [00:46<00:24, 28.03it/s]
66%|   | 1314/2001 [00:46<00:24, 28.00it/s]
66%|   | 1317/2001 [00:46<00:24, 27.97it/s]
66%|   | 1320/2001 [00:47<00:24, 27.71it/s]
66%|   | 1323/2001 [00:47<00:24, 27.74it/s]
66%|   | 1326/2001 [00:47<00:24, 27.93it/s]
66%|   | 1329/2001 [00:47<00:23, 28.00it/s]
67%|   | 1332/2001 [00:47<00:23, 28.04it/s]
67%|   | 1335/2001 [00:47<00:23, 27.98it/s]
67%|   | 1338/2001 [00:47<00:23, 28.06it/s]
67%|   | 1341/2001 [00:47<00:23, 28.12it/s]
67%|   | 1344/2001 [00:47<00:23, 28.20it/s]
67%|   | 1347/2001 [00:48<00:23, 28.18it/s]
67%|   | 1350/2001 [00:48<00:23, 28.14it/s]
68%|   | 1353/2001 [00:48<00:23, 28.10it/s]
68%|   | 1356/2001 [00:48<00:22, 28.09it/s]
68%|   | 1359/2001 [00:48<00:22, 28.02it/s]
68%|   | 1362/2001 [00:48<00:22, 28.02it/s]
68%|   | 1365/2001 [00:48<00:22, 28.03it/s]
68%|   | 1368/2001 [00:48<00:22, 28.11it/s]
69%|   | 1371/2001 [00:48<00:22, 28.11it/s]
69%|   | 1374/2001 [00:48<00:22, 28.05it/s]
69%|   | 1377/2001 [00:49<00:22, 28.02it/s]
69%|   | 1380/2001 [00:49<00:22, 28.00it/s]
69%|   | 1383/2001 [00:49<00:22, 28.05it/s]
69%|   | 1386/2001 [00:49<00:21, 28.14it/s]
69%|   | 1389/2001 [00:49<00:21, 28.23it/s]
70%|   | 1392/2001 [00:49<00:21, 28.30it/s]
70%|   | 1395/2001 [00:49<00:21, 28.17it/s]
```

(continues on next page)

(continued from previous page)

```
70%|   | 1398/2001 [00:49<00:21, 28.16it/s]
70%|   | 1401/2001 [00:49<00:21, 28.06it/s]
70%|   | 1404/2001 [00:50<00:21, 28.09it/s]
70%|   | 1407/2001 [00:50<00:21, 28.04it/s]
70%|   | 1410/2001 [00:50<00:21, 28.14it/s]
71%|   | 1413/2001 [00:50<00:20, 28.24it/s]
71%|   | 1416/2001 [00:50<00:20, 28.29it/s]
71%|   | 1419/2001 [00:50<00:20, 28.21it/s]
71%|   | 1422/2001 [00:50<00:20, 28.10it/s]
71%|   | 1425/2001 [00:50<00:20, 28.11it/s]
71%|  | 1428/2001 [00:50<00:20, 28.00it/s]
72%|  | 1431/2001 [00:51<00:20, 28.04it/s]
72%|  | 1434/2001 [00:51<00:20, 28.02it/s]
72%|  | 1437/2001 [00:51<00:20, 28.08it/s]
72%|  | 1440/2001 [00:51<00:19, 28.19it/s]
72%|  | 1443/2001 [00:51<00:19, 28.19it/s]
72%|  | 1446/2001 [00:51<00:19, 28.09it/s]
72%|  | 1449/2001 [00:51<00:19, 28.06it/s]
73%|  | 1452/2001 [00:51<00:19, 28.04it/s]
73%|  | 1455/2001 [00:51<00:19, 27.95it/s]
73%|  | 1458/2001 [00:51<00:19, 28.07it/s]
73%|  | 1461/2001 [00:52<00:19, 28.17it/s]
73%|  | 1464/2001 [00:52<00:19, 28.12it/s]
73%|  | 1467/2001 [00:52<00:18, 28.11it/s]
73%|  | 1470/2001 [00:52<00:18, 28.08it/s]
74%|  | 1473/2001 [00:52<00:18, 28.02it/s]
74%|  | 1476/2001 [00:52<00:18, 28.01it/s]
74%|  | 1479/2001 [00:52<00:18, 28.00it/s]
74%|  | 1482/2001 [00:52<00:18, 27.93it/s]
74%|  | 1485/2001 [00:52<00:18, 27.93it/s]
74%|  | 1488/2001 [00:53<00:18, 27.90it/s]
75%|  | 1491/2001 [00:53<00:18, 27.99it/s]
75%|  | 1494/2001 [00:53<00:18, 28.08it/s]
75%|  | 1497/2001 [00:53<00:17, 28.09it/s]
75%|  | 1500/2001 [00:53<00:17, 28.18it/s]
75%|  | 1503/2001 [00:53<00:17, 28.07it/s]
75%|  | 1506/2001 [00:53<00:17, 28.15it/s]
75%|  | 1509/2001 [00:53<00:17, 28.22it/s]
76%|  | 1512/2001 [00:53<00:17, 28.27it/s]
76%|  | 1515/2001 [00:54<00:17, 28.26it/s]
76%|  | 1518/2001 [00:54<00:17, 28.28it/s]
76%|  | 1521/2001 [00:54<00:16, 28.34it/s]
76%|  | 1524/2001 [00:54<00:16, 28.33it/s]
76%|  | 1527/2001 [00:54<00:16, 28.27it/s]
76%|  | 1530/2001 [00:54<00:16, 28.21it/s]
77%|  | 1533/2001 [00:54<00:16, 28.23it/s]
77%|  | 1536/2001 [00:54<00:16, 28.18it/s]
77%|  | 1539/2001 [00:54<00:16, 28.19it/s]
77%|  | 1542/2001 [00:54<00:16, 28.13it/s]
77%|  | 1545/2001 [00:55<00:16, 28.13it/s]
77%|  | 1548/2001 [00:55<00:16, 28.15it/s]
78%|  | 1551/2001 [00:55<00:15, 28.21it/s]
```

(continues on next page)

(continued from previous page)

```
78%|  | 1554/2001 [00:55<00:15, 28.31it/s]
78%|  | 1557/2001 [00:55<00:15, 28.35it/s]
78%|  | 1560/2001 [00:55<00:15, 28.31it/s]
78%|  | 1563/2001 [00:55<00:15, 28.43it/s]
78%|  | 1566/2001 [00:55<00:15, 28.39it/s]
78%|  | 1569/2001 [00:55<00:15, 28.38it/s]
79%|  | 1572/2001 [00:56<00:15, 28.29it/s]
79%|  | 1575/2001 [00:56<00:15, 28.17it/s]
79%|  | 1578/2001 [00:56<00:14, 28.20it/s]
79%|  | 1581/2001 [00:56<00:14, 28.24it/s]
79%|  | 1584/2001 [00:56<00:14, 28.29it/s]
79%|  | 1587/2001 [00:56<00:14, 28.03it/s]
79%|  | 1590/2001 [00:56<00:14, 28.14it/s]
80%|  | 1593/2001 [00:56<00:14, 27.95it/s]
80%|  | 1596/2001 [00:56<00:14, 27.97it/s]
80%|  | 1599/2001 [00:56<00:14, 27.95it/s]
80%|  | 1602/2001 [00:57<00:14, 27.85it/s]
80%|  | 1605/2001 [00:57<00:14, 27.88it/s]
80%|  | 1608/2001 [00:57<00:14, 28.05it/s]
81%|  | 1611/2001 [00:57<00:13, 28.09it/s]
81%|  | 1614/2001 [00:57<00:13, 28.10it/s]
81%|  | 1617/2001 [00:57<00:13, 28.09it/s]
81%|  | 1620/2001 [00:57<00:13, 28.13it/s]
81%|  | 1623/2001 [00:57<00:13, 28.05it/s]
81%| | 1626/2001 [00:57<00:13, 28.05it/s]
81%| | 1629/2001 [00:58<00:13, 27.96it/s]
82%| | 1632/2001 [00:58<00:13, 28.01it/s]
82%| | 1635/2001 [00:58<00:13, 28.05it/s]
82%| | 1638/2001 [00:58<00:12, 28.01it/s]
82%| | 1641/2001 [00:58<00:12, 27.99it/s]
82%| | 1644/2001 [00:58<00:12, 28.01it/s]
82%| | 1647/2001 [00:58<00:12, 28.07it/s]
82%| | 1650/2001 [00:58<00:12, 28.06it/s]
83%| | 1653/2001 [00:58<00:12, 28.04it/s]
83%| | 1656/2001 [00:59<00:12, 28.03it/s]
83%| | 1659/2001 [00:59<00:12, 28.06it/s]
83%| | 1662/2001 [00:59<00:12, 28.16it/s]
83%| | 1665/2001 [00:59<00:11, 28.25it/s]
83%| | 1668/2001 [00:59<00:11, 28.29it/s]
84%| | 1671/2001 [00:59<00:11, 28.21it/s]
84%| | 1674/2001 [00:59<00:11, 28.17it/s]
84%| | 1677/2001 [00:59<00:11, 28.16it/s]
84%| | 1680/2001 [00:59<00:11, 28.08it/s]
84%| | 1683/2001 [00:59<00:11, 28.09it/s]
84%| | 1686/2001 [01:00<00:11, 28.18it/s]
84%| | 1689/2001 [01:00<00:11, 28.17it/s]
85%| | 1692/2001 [01:00<00:10, 28.17it/s]
85%| | 1695/2001 [01:00<00:10, 28.25it/s]
85%| | 1698/2001 [01:00<00:10, 28.20it/s]
85%| | 1701/2001 [01:00<00:10, 28.18it/s]
85%| | 1704/2001 [01:00<00:10, 28.04it/s]
85%| | 1707/2001 [01:00<00:10, 27.99it/s]
```

(continues on next page)

(continued from previous page)

```
85%| | 1710/2001 [01:00<00:10, 27.98it/s]
86%| | 1713/2001 [01:01<00:10, 27.84it/s]
86%| | 1716/2001 [01:01<00:10, 27.76it/s]
86%| | 1719/2001 [01:01<00:10, 27.92it/s]
86%| | 1722/2001 [01:01<00:09, 28.07it/s]
86%| | 1725/2001 [01:01<00:09, 28.12it/s]
86%| | 1728/2001 [01:01<00:09, 28.06it/s]
87%| | 1731/2001 [01:01<00:09, 28.01it/s]
87%| | 1734/2001 [01:01<00:09, 27.98it/s]
87%| | 1737/2001 [01:01<00:09, 28.02it/s]
87%| | 1740/2001 [01:02<00:09, 28.12it/s]
87%| | 1743/2001 [01:02<00:09, 28.12it/s]
87%| | 1746/2001 [01:02<00:09, 28.24it/s]
87%| | 1749/2001 [01:02<00:08, 28.23it/s]
88%| | 1752/2001 [01:02<00:08, 28.25it/s]
88%| | 1755/2001 [01:02<00:08, 28.26it/s]
88%| | 1758/2001 [01:02<00:08, 28.28it/s]
88%| | 1761/2001 [01:02<00:08, 28.16it/s]
88%| | 1764/2001 [01:02<00:08, 28.12it/s]
88%| | 1767/2001 [01:02<00:08, 28.10it/s]
88%| | 1770/2001 [01:03<00:08, 28.08it/s]
89%| | 1773/2001 [01:03<00:08, 27.92it/s]
89%| | 1776/2001 [01:03<00:08, 28.00it/s]
89%| | 1779/2001 [01:03<00:07, 28.11it/s]
89%| | 1782/2001 [01:03<00:07, 28.15it/s]
89%| | 1785/2001 [01:03<00:07, 28.20it/s]
89%| | 1788/2001 [01:03<00:07, 28.20it/s]
90%| | 1791/2001 [01:03<00:07, 28.30it/s]
90%| | 1794/2001 [01:03<00:07, 28.26it/s]
90%| | 1797/2001 [01:04<00:07, 28.30it/s]
90%| | 1800/2001 [01:04<00:07, 28.22it/s]
90%| | 1803/2001 [01:04<00:07, 28.28it/s]
90%| | 1806/2001 [01:04<00:06, 28.38it/s]
90%| | 1809/2001 [01:04<00:06, 28.31it/s]
91%| | 1812/2001 [01:04<00:06, 28.25it/s]
91%| | 1815/2001 [01:04<00:06, 28.22it/s]
91%| | 1818/2001 [01:04<00:06, 28.20it/s]
91%| | 1821/2001 [01:04<00:06, 28.16it/s]
91%| | 1824/2001 [01:04<00:06, 28.21it/s]
91%|| 1827/2001 [01:05<00:06, 28.14it/s]
91%|| 1830/2001 [01:05<00:06, 28.11it/s]
92%|| 1833/2001 [01:05<00:05, 28.19it/s]
92%|| 1836/2001 [01:05<00:05, 28.29it/s]
92%|| 1839/2001 [01:05<00:05, 28.46it/s]
92%|| 1842/2001 [01:05<00:05, 28.47it/s]
92%|| 1845/2001 [01:05<00:05, 28.38it/s]
92%|| 1848/2001 [01:05<00:05, 28.43it/s]
93%|| 1851/2001 [01:05<00:05, 28.29it/s]
93%|| 1854/2001 [01:06<00:05, 28.32it/s]
93%|| 1857/2001 [01:06<00:05, 28.22it/s]
93%|| 1860/2001 [01:06<00:04, 28.29it/s]
93%|| 1863/2001 [01:06<00:04, 28.26it/s]
```

(continues on next page)

(continued from previous page)

```
 93%|| 1866/2001 [01:06<00:04, 28.22it/s]
 93%|| 1869/2001 [01:06<00:04, 28.20it/s]
 94%|| 1872/2001 [01:06<00:04, 28.26it/s]
 94%|| 1875/2001 [01:06<00:04, 28.15it/s]
 94%|| 1878/2001 [01:06<00:04, 28.12it/s]
 94%|| 1881/2001 [01:07<00:04, 28.12it/s]
 94%|| 1884/2001 [01:07<00:04, 28.08it/s]
 94%|| 1887/2001 [01:07<00:04, 28.06it/s]
 94%|| 1890/2001 [01:07<00:03, 28.07it/s]
 95%|| 1893/2001 [01:07<00:03, 28.20it/s]
 95%|| 1896/2001 [01:07<00:03, 28.15it/s]
 95%|| 1899/2001 [01:07<00:03, 28.11it/s]
 95%|| 1902/2001 [01:07<00:03, 28.15it/s]
 95%|| 1905/2001 [01:07<00:03, 28.17it/s]
 95%|| 1908/2001 [01:07<00:03, 28.23it/s]
 96%|| 1911/2001 [01:08<00:03, 28.26it/s]
 96%|| 1914/2001 [01:08<00:03, 28.21it/s]
 96%|| 1917/2001 [01:08<00:02, 28.24it/s]
 96%|| 1920/2001 [01:08<00:02, 28.34it/s]
 96%|| 1923/2001 [01:08<00:02, 28.32it/s]
 96%|| 1926/2001 [01:08<00:02, 28.25it/s]
 96%|| 1929/2001 [01:08<00:02, 28.24it/s]
 97%|| 1932/2001 [01:08<00:02, 28.23it/s]
 97%|| 1935/2001 [01:08<00:02, 28.07it/s]
 97%|| 1938/2001 [01:09<00:02, 28.00it/s]
 97%|| 1941/2001 [01:09<00:02, 27.87it/s]
 97%|| 1944/2001 [01:09<00:02, 27.95it/s]
 97%|| 1947/2001 [01:09<00:01, 28.01it/s]
 97%|| 1950/2001 [01:09<00:01, 27.90it/s]
 98%|| 1953/2001 [01:09<00:01, 27.97it/s]
 98%|| 1956/2001 [01:09<00:01, 28.07it/s]
 98%|| 1959/2001 [01:09<00:01, 28.06it/s]
 98%|| 1962/2001 [01:09<00:01, 28.17it/s]
 98%|| 1965/2001 [01:09<00:01, 28.12it/s]
 98%|| 1968/2001 [01:10<00:01, 28.18it/s]
 99%|| 1971/2001 [01:10<00:01, 28.17it/s]
 99%|| 1974/2001 [01:10<00:00, 28.09it/s]
 99%|| 1977/2001 [01:10<00:00, 28.13it/s]
 99%|| 1980/2001 [01:10<00:00, 28.02it/s]
 99%|| 1983/2001 [01:10<00:00, 28.13it/s]
 99%|| 1986/2001 [01:10<00:00, 28.18it/s]
 99%|| 1989/2001 [01:10<00:00, 28.07it/s]
100%|| 1992/2001 [01:10<00:00, 28.03it/s]
100%|| 1995/2001 [01:11<00:00, 28.04it/s]
100%|| 1998/2001 [01:11<00:00, 27.91it/s]
100%|| 2001/2001 [01:11<00:00, 28.04it/s]
100%|| 2001/2001 [01:11<00:00, 28.08it/s]

<maicos.modules.pdfplanar.PDFPlanar object at 0x7fad7be05ca0>
```

We also calculate the density profile of the water molecules in order to compare the different slabs with the layering visible in the density.

```
dana_obj = maicos.DensityPlanar(
    water, dim=2, refgroup=water, bin_width=0.1, sym=True, zmin=-7, zmax=7
)

dana_obj.run(step=10)
```

```
Unwrapping in combination with the `wrap_compound='atoms` is superfluous. `unwrap` will␣
↪be set to `False`.

<maicos.modules.densityplanar.DensityPlanar object at 0x7fad7be354c0>
```

The results of the analysis are stored in the `results` member variable. As per the documentation of `PDFPlanar`, we get three different arrays: `bin_pos`, `bins`, and `pdf`. Here, `bin_pos` is the position of the center of the slices in the z-direction, `bins` contains the bin positions of the pair distribution, which are shared by all slices and correspondingly `pdf` contains each profile that our code produced.

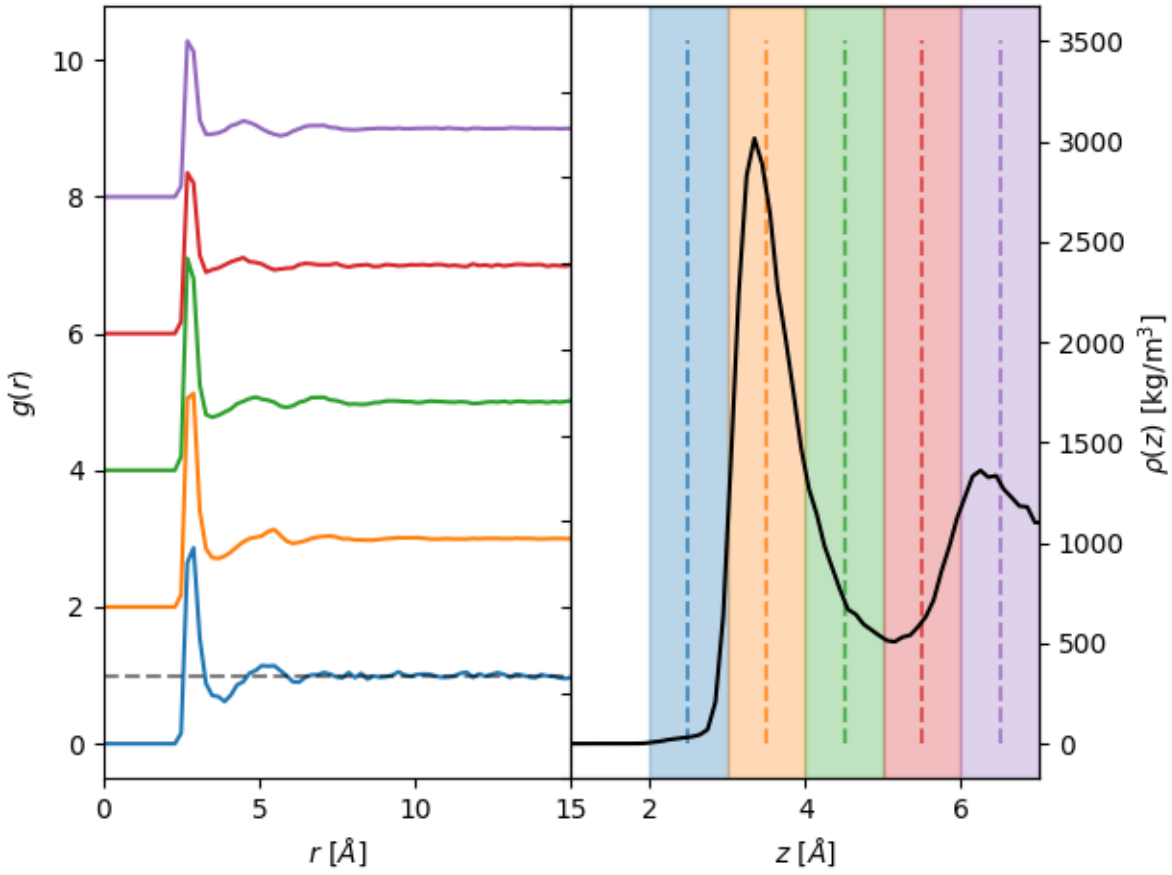
In the following, we loop over all the pdf slices and plot each of them. Furthermore, in a separate subplot, we also show the density profile of the water molecules and highlight the slices that each pdf is calculated for. Hence, the same color in both plots corresponds to the same slice for the pair distribution function and the density profile. %%

```
# u per cubic angstrom to kg per cubic meter factor
u2kg = 1660.5390665999998

fig, ax = plt.subplots(1, 2)
print(ax)

tax = ax[1].twinx()
shift = 0
shift_amount = 2
for i in range(0, len(ana_obj.results.pdf[0])):
    bin_pos = ana_obj.results.bin_pos[i]

    pdf_prof = ana_obj.results.pdf[:, i]
    mean_bulk = np.mean(pdf_prof[ana_obj.results.bins > 10])

    line = ax[0].plot(
        ana_obj.results.bins, ana_obj.results.pdf[:, i] / mean_bulk + shift
    )
    tax.vlines(
        7 + bin_pos, 0, 3500, alpha=0.7, color=line[0].get_color(), linestyles="dashed"
    )

    tax.axvspan(
        7 + bin_pos - 0.25 * 2,
        7 + bin_pos + 0.25 * 2,
        color=line[0].get_color(),
        alpha=0.3,
    )
    shift += shift_amount

ax[0].set_ylabel(r"$g(r)$")
ax[0].set_xlabel(r"$r$ [$\AA$]")
ax[0].set_xlim((0, 15))
```

(continues on next page)

(continued from previous page)

```
ax[0].hlines(1, 0, 15, color="black", linestyles="dashed", alpha=0.5)

tax.plot(
    7 + dana_obj.results.bin_pos,
    dana_obj.results.profile * u2kg,
    color="black",
    label="Density",
)
tax.set_xlim((1, 7))

ax[1].set_yticks(tax.get_yticks())

ax[1].set_yticklabels([])

tax.set_ylabel(r"$\rho(z)$ [kg/m$^3$]")
ax[1].set_xlabel(r"$z$ [$\AA$]")

# Set the padding between the axis to zero
plt.tight_layout()

fig.subplots_adjust(wspace=0, hspace=0)
fig.dpi = 200
```



```
[<Axes: > <Axes: >]
```

Total running time of the script: (1 minutes 12.400 seconds)

5.2.5 Calculating and interpreting dipolar pair correlation functions

In this examples we will calculate dipolar pair correlation functions in real and Fourier space using the maicos modules *maicos.RDFDiporder* and *maicos.DiporderStructureFactor*. We will show how these pair correlation functions are connected to each other and electrostatic properties like the dielectric constant ε and the Kirkwood factor g_K.

We start by importing the necessarary modules

```
import matplotlib.pyplot as plt
import MDAnalysis as mda
import numpy as np
import scipy
from MDAnalysis.analysis.dielectric import DielectricConstant

import maicos
from maicos.lib.math import compute_rdf_structure_factor
```

Our example system is $N = 512$ rigid SPC/E water molecules simulated in an NVT ensemble at $300\,\mathrm{K}$ in a cubic cell of $L = 24.635$. To follow this how-to guide, you should download the `topology` and the `trajectory` files of the system. Below we load the system, report and store some system properties for later usage.

```
u = mda.Universe("water_nvt.tpr", "water_nvt.xtc")

volume = u.trajectory.ts.volume
density = u.residues.n_residues / volume
dipole_moment = u.atoms.dipole_moment(compound="residues", unwrap=True).mean()

print(f"_n = {density:.3f} Å^-3")
print(f"μ = {dipole_moment:.2f} eÅ")
```

```
_n = 0.034 Å^-3
μ = 0.49 eÅ
```

The results of our first property calculations show that the number density as well as the dipole moment of a single water molecule is consistent with the literature[1].

[1] Carlos Vega and Jose L. F. Abascal. Simulating water with rigid non-polarizable models: a general perspective. *Phys. Chem. Chem. Phys.*, 13(44):19663–19688, November 2011. doi:10.1039/C1CP22168J.

Static dielectric constant

To start with the analysis we first look at the dielectric constant of the system. If you run a simulation using an Ewald simulation technique as usually done, the dielectric constant for such system with metallic boundary conditions is given according to Neumann[2] by

$$\varepsilon = 1 + \frac{\langle M^2 \rangle_{\mathrm{MBE}} - \langle M \rangle^2_{\mathrm{MBE}}}{3\varepsilon_0 V k_B T}$$

where

$$\boldsymbol{M} = \sum_{i=1}^{N} \boldsymbol{\mu}_i$$

is the total dipole moment of the box, V its volume and ε_0 the vacuum permittivity. We use the subscript in the expectation value MBE indicating that the equation only holds for simulations with **M**etallic **B**oundary conditions in an **E**wald simulation style. As shown in the equation for $\varepsilon(\mathrm{MBE})$ the dielectric constant here is a *total cell* quantity connecting the fluctuations of the total dipole moment to the dielectric constant. We can calculate $\varepsilon_{\mathrm{MBE}}$ using the `MDAnalysis.analysis.dielectric.DielectricConstant` module of MDAnalysis.

```
epsilon_mbe = DielectricConstant(atomgroup=u.atoms).run()
print(f"_MBE = {epsilon_mbe.results.eps_mean:.2f}")
```

```
_MBE = 69.21
```

The value of 70 is the same as reported in the literature for the rigid SPC/E water model[Page 65, 1].

Kirkwood factor

Knowing the dielectric constant we can also calculate the Kirkwood factor g_K which is a measure describing molecular correlations. I.e a Kirkwood factor greater than 1 indicates that neighboring molecular dipoles are more likely to align in the same direction, enhancing the material's polarization and, consequently, its dielectric constant. Based on the dielectric constant ε Kirkwood and Fröhlich derived the relation for the factor g_K according to

$$\frac{N\mu^2 g_K}{\varepsilon_0 V k_B T} = \frac{(\varepsilon - 1)(2\varepsilon + 1)}{\varepsilon}$$

This relation is valid for a sample in an infinity, homogenous medium of the same dielectric constant. Below we implement this equation and calculate the factor for our system.

```
def kirkwood_factor_KF(
    dielectric_constant: float,
    volume: float,
    n_dipoles: float,
    molecular_dipole_moment: float,
    temperature: float = 300,
) -> float:
    """Kirkwood factor in the Kirkwood-Fröhlich way.

    For the sample in an infinity, homogenous medium of the same dielectric constant.

    Parameters
```

(continues on next page)

[2] Martin Neumann. Dipole moment fluctuation formulas in computer simulations of polar systems. *Molecular Physics*, 50(4):841–858, November 1983. doi:10.1080/00268978300102721.

(continued from previous page)

```
    ----------
    dielectric_constant : float
        the static dielectric constant
    volume : float
        system volume in Å^3
    n_dipoles : float
        number of dipoles
    molecular_dipole_moment : float
        dipole moment of a molecule (eÅ)
    temperature : float
        temperature of the simulation K
    """
    dipole_moment_sq = (
        molecular_dipole_moment
        * scipy.constants.elementary_charge
        * scipy.constants.angstrom
    ) ** 2
    factor = (
        scipy.constants.epsilon_0
        * (volume * scipy.constants.angstrom**3)
        * scipy.constants.Boltzmann
        * temperature
    )

    return (
        factor
        / (dielectric_constant * n_dipoles * dipole_moment_sq)
        * (dielectric_constant - 1)
        * (2 * dielectric_constant + 1)
    )


kirkwood_KF = kirkwood_factor_KF(
    dielectric_constant=epsilon_mbe.results.eps_mean,
    volume=volume,
    n_dipoles=u.residues.n_residues,
    molecular_dipole_moment=dipole_moment,
)

print(f"g_K = {kirkwood_KF:.2f}")
```

```
g_K = 2.39
```

This value means there is a quite strong correlation between neighboring water molecules. The dielectric constant ε is a material property and does not depend on the boundary condition. Instead, the Kirkwood factor is indicative of dipole-dipole correlations which instead depend on the boundary condistions in the simulation. This relation is described and shown below.

Connecting the Kirkwood factor to real space dipolar pair-correlation functions

The r-dependent Kirkwood factor can also be calculated from real space dipole-dipole pair correlation function[3]

$$g_{\hat{\boldsymbol{\mu}},\hat{\boldsymbol{\mu}}}(r) = \frac{1}{N}\left\langle \sum_i \frac{1}{n_i(r)} \sum_{j=1}^{n_i(r)} (\hat{\boldsymbol{\mu}}_i \cdot \hat{\boldsymbol{\mu}}_j) \right\rangle$$

where $\hat{\boldsymbol{\mu}}$ is the normalized dipole moment and $n_i(r)$ is the number of dipoles within a spherical shell of distance r and $r + \delta r$ from dipole i. We compute the pair correlation function using the *maicos.RDFDiporder* module up to half of the length of cubic simulation box. We drop a delta like contribution in $r = 0$ caused by interaction of the dipole with itself.

```
L_half = u.dimensions[:3].max() / 2

rdf_diporder = maicos.RDFDiporder(g1=u.atoms, rmax=L_half, bin_width=0.01)
rdf_diporder.run()
```

```
<maicos.modules.rdfdiporder.RDFDiporder object at 0x7fad78d82fa0>
```

Based on this correlation function we can calculate the radially resolved Kirkwood factor via[4]

$$G_K(r) = \rho_n 4\pi \int_0^r \mathrm{d}r' r'^2 g_{\hat{\mu},\hat{\mu}}(r') + 1$$

where the "+1" accounts for the integration of the delta function at $r = 0$. Here $\rho_n = N/V$ is the density of dipoles.

```
radial_kirkwood = 1 + (
    density
    * 4
    * np.pi
    * scipy.integrate.cumulative_trapezoid(
        x=rdf_diporder.results.bins,
        y=rdf_diporder.results.bins**2 * rdf_diporder.results.rdf,
        initial=0,
    )
)
```

While, for a truly infinite system, the r- dependent Kirkwood factor, $G_\mathrm{K}(r)$ is short range[5 Page 68, 4], the boundary conditions on a finite system introduce long-range effects. In particular, within MBE, Caillol[6] has shown that $G_\mathrm{K}(r)$ has a spurious asymptotic growth proportional to r^3/V. This effect is stil present at $r = r_K$, where r_K (here approximately 6 Å) indicates a distance after which all the physical features of $g_{\hat{\boldsymbol{\mu}},\hat{\boldsymbol{\mu}}}(r)$ are extinct. For more details see the original literature. Below we show the pair correlation function as well as the radial and the (static) Kirkwood factor as gray dashed line.

```
fig, ax = plt.subplots(2)
```

(continues on next page)

[3] Cui Zhang and Giulia Galli. Dipolar correlations in liquid water. *The Journal of Chemical Physics*, 141(8):084504, August 2014. doi:10.1063/1.4893638.

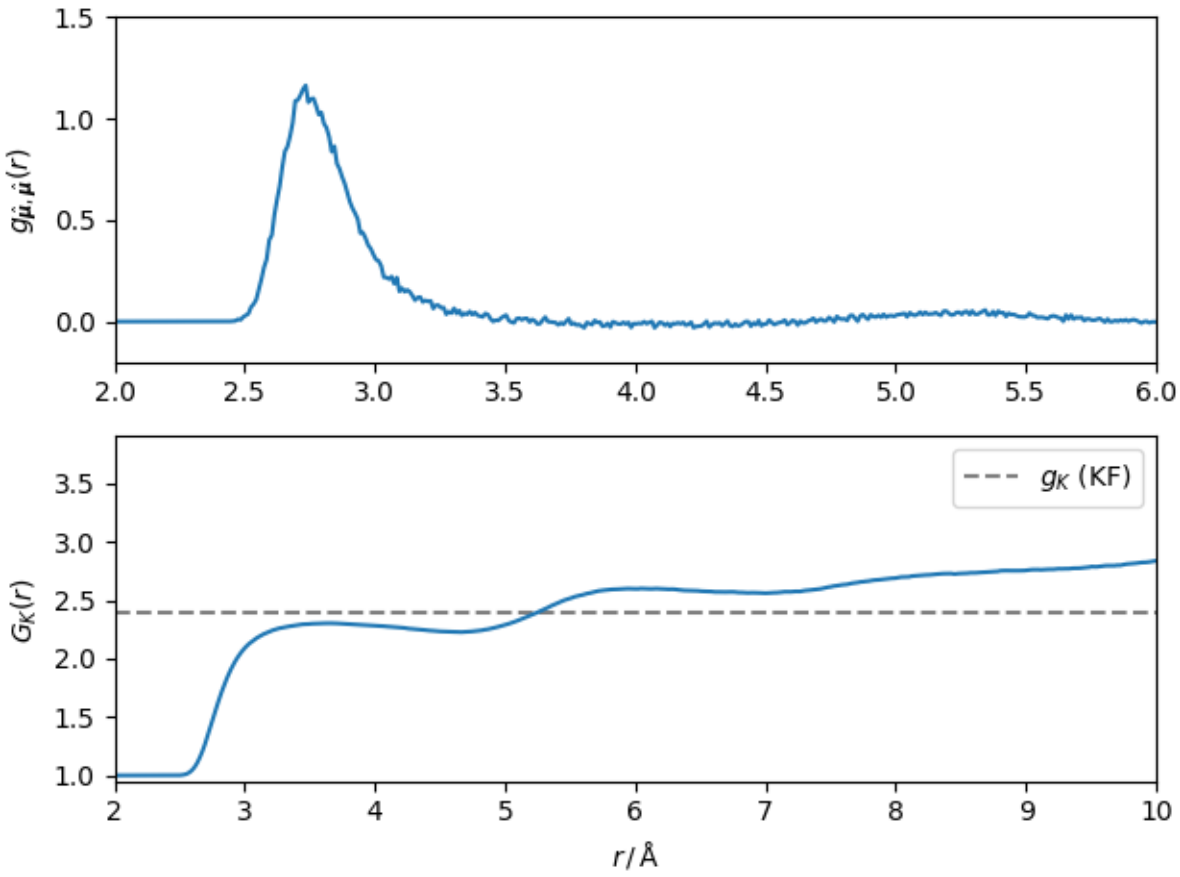
[4] Chao Zhang, Jürg Hutter, and Michiel Sprik. Computing the Kirkwood g-Factor by Combining Constant Maxwell Electric Field and Electric Displacement Simulations: Application to the Dielectric Constant of Liquid Water. *J. Phys. Chem. Lett.*, 7(14):2696–2701, July 2016. doi:10.1021/acs.jpclett.6b01127.

[5] H. Fröhlich. *Theory of Dielectrics: Dielectric Constant and Dielectric Loss*. Monographs on the Physics and Chemistry of Materials. Oxford University Press, 2 edition, 1958.

[6] J. M. Caillol. Asymptotic behavior of the pair-correlation function of a polar liquid. *The Journal of Chemical Physics*, 96(9):7039–7053, May 1992. doi:10.1063/1.462536.

(continued from previous page)

```
ax[0].plot(
    rdf_diporder.results.bins,
    rdf_diporder.results.rdf,
)
ax[1].axhline(kirkwood_KF, ls="--", c="gray", label="$g_K$ (KF)")

ax[1].plot(rdf_diporder.results.bins, radial_kirkwood)

ax[0].set(
    xlim=(2, 6),
    ylim=(-0.2, 1.5),
    ylabel=r"$g_\mathrm{\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\mu}}}(r)$",
)

ax[1].set(
    xlim=(2, 10),
    ylim=(0.95, 3.9),
    xlabel=r"$r\,/\,\mathrm{Å}$",
    ylabel=r"$G_K(r)$",
)

ax[1].legend()

fig.align_labels()
fig.tight_layout()
```



Notice that the Kirkwood Fröhlich estimator for the Kirkwood factors differs from the value of $G_K(r = r_K)$ obtained from simulations in the MBE ensemble.

Dipole Structure factor

An alternative approach to calculate the dielectric constant is via the dipole structure factor which is given by

$$S(q)_{\hat{\boldsymbol{\mu}}\hat{\boldsymbol{\mu}}} = \left\langle \frac{1}{N} \sum_{i,j=1}^{N} \hat{\mu}_i \hat{\mu}_j \, \exp(-i\boldsymbol{q} \cdot [\boldsymbol{r}_i - \boldsymbol{r}_j]) \right\rangle$$

We compute the structure factor using the *maicos.DiporderStructureFactor* module.

```
diporder_structure_factors = maicos.DiporderStructureFactor(atomgroup=u.atoms, dq=0.05)
diporder_structure_factors.run()
```

```
<maicos.modules.diporderstructurefactor.DiporderStructureFactor object at 0x7fad78f6e2b0>
```

As also shown *how to on SAXS calculations* the structure factor can also be obtained directly from the real space correlation functions using Fourier transformation via

$$S^{\mathrm{FT}}_{\hat{\boldsymbol{\mu}}\hat{\boldsymbol{\mu}}}(q) = 1 + 4\pi\rho \int_0^\infty \mathrm{d}r r \frac{\sin(qr)}{q} g_{\hat{\mu}\hat{\mu}}(r) \,,$$

which can be obtained by the function *maicos.lib.math.compute_rdf_structure_factor()*. We have assumed an isotropic system so that $S(\boldsymbol{q}) = S(q)$. Note that we added a one to the dipole pair correlation function due to the implementation of the Fourier transformation inside *maicos.lib.math.compute_rdf_structure_factor()*.

```
q_rdf, struct_fac_rdf = compute_rdf_structure_factor(
    rdf=1 + rdf_diporder.results.rdf, r=rdf_diporder.results.bins, density=density
)
```

Before we plot the structure factors we first also fit the low q limit according to a quadratic function as

$$S_{\hat{\mu}\hat{\mu}}(q \to 0) \approx S_0 + S_2 q^2$$

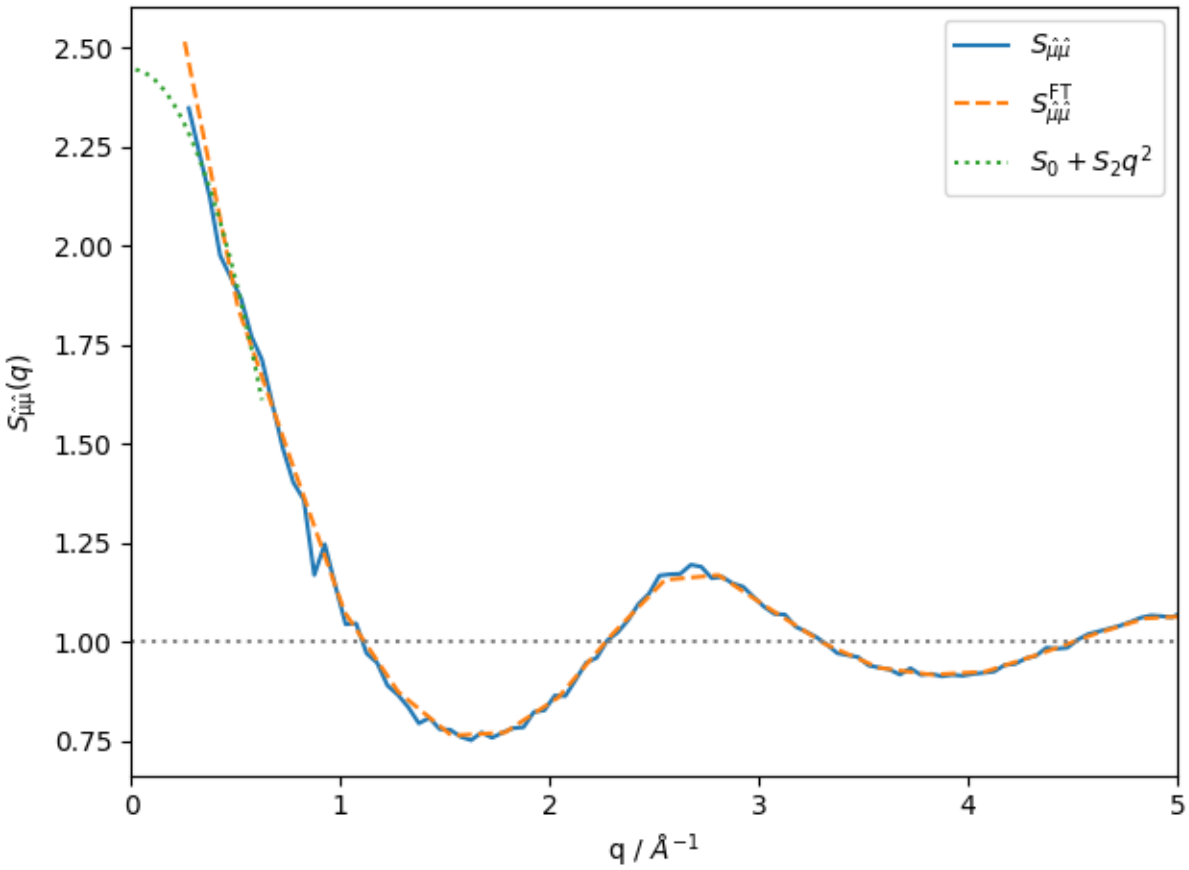
The fit contains no linear term because of the structure factors' symmetry around 0.

```
n_max = 5  # take `n_max` first data points of the structure factor for the fit

# q_max is the maximal q value corresponding to the last point taken for the fit
q_max = diporder_structure_factors.results.scattering_vectors[n_max]
print(f"q_max = {q_max:.2f} Å")

eps_fit = np.polynomial.Polynomial.fit(
    x=diporder_structure_factors.results.scattering_vectors[:n_max],
    y=diporder_structure_factors.results.structure_factors[:n_max],
    deg=(0, 2),
    domain=(-q_max, q_max),
)

print(
    f"Best fit parameters: S_0 = {eps_fit.coef[0]:.2f}, S_2 = {eps_fit.coef[2]:.2f} Å^2"
)
```

```
q_max = 0.63 Å
Best fit parameters: S_0 = 2.45, S_2 = -0.84 Å^2
```

Now we can finally plot the structure factor

```
plt.plot(
    diporder_structure_factors.results.scattering_vectors,
    diporder_structure_factors.results.structure_factors,
    label=r"$S_{\hat \mu\hat \mu}$",
)
plt.plot(
    q_rdf, struct_fac_rdf, ls="dashed", label=r"$S_{\hat \mu\hat \mu}^\mathrm{FT}$"
)
plt.plot(*eps_fit.linspace(50), ls="dotted", label=r"$S_0 + S_2 q^2$")

plt.axhline(1, ls=":", c="gray")
plt.ylabel(r"$S_\mathrm{\hat\mu \hat\mu}(q)$")
plt.xlabel(r"q / $Å^{-1}$")
plt.tight_layout()
plt.xlim(0, 5)
plt.legend()
plt.show()
```



You see that the orange and the blue curve agree. We also add the fit as a green dotted line. From S_0 we can extract the dielectric constant via[7]

$$\frac{\mu^2}{\varepsilon_0} S_0 = \frac{(\varepsilon - 1)(2\varepsilon + 1)}{\varepsilon}$$

This formula can be inverted and an estimator for ε_S can be obtained as we show below.

```
def dielectric_constant_struc_fact(S_0: float, molecular_dipole_moment: float) -> float:
    """The dielectric constant calculated from the q->0 limit of the structure factor.

    Parameters
    ----------
    q_0_limit : float
        the q -> 0 limit if the dipololar structure factor
    molecular_dipole_moment : float
        dipole moment of a molecule (eÅ)
    """
    dipole_moment_sq = (
        molecular_dipole_moment
        * scipy.constants.angstrom
        * scipy.constants.elementary_charge
    ) ** 2
```

(continues on next page)

[7] Jean-Pierre Hansen and Ian. R. McDonald. *Theory of Simple Liquids*. Elsevier / Academic Press, 3rd ed edition, 2006. ISBN 9780080455075.

(continued from previous page)

```
    S_limit = (
        dipole_moment_sq
        * S_0
        / scipy.constants.epsilon_0
        / scipy.constants.elementary_charge
        / scipy.constants.angstrom**3
    )

    return (np.sqrt((S_limit) ** 2 + 2 * S_limit + 9) + S_limit + 1) / 4


epsilon_struct_fac = dielectric_constant_struc_fact(
    S_0=eps_fit.coef[0], molecular_dipole_moment=dipole_moment
)
print(f"_S = {epsilon_struct_fac:.2f}")
```

```
_S = 53.53
```

Which is quite close the value calculated directly from the total dipole fluctuations of the simulations $\varepsilon_{\mathrm{MBE}} \approx 69$. This difference may result in the very crude fit that is performed and it could be drastically improved by a Bayesian fitting method as for example for fitting the Seebeck coefficient from a similar structure factor[8].

References

Total running time of the script: (2 minutes 2.492 seconds)

5.3 Reference guides

The reference guides contain details for all the analysis modules. The API documentation gives details on how the calculators and additional functions can be used from each language.

5.3.1 Analysis Modules

This is a list of all analysis modules provided by MAICoS. Note that we do not have an example section for each module. Instead, we refer to our *tutorial* and *How-to guides*. Once you understand the basic structure taught there, you can work with all modules. If not, feel free to raise an issue in Gitlab or ask us on Discord.

[8] Enrico Drigo and Stefano Baroni. Seebeck Coefficient of Liquid Water from Equilibrium Molecular Dynamics. *J. Chem. Theory Comput.*, 19(23):8855–8860, December 2023. doi:10.1021/acs.jctc.3c00760.

DensityCylinder

class maicos.**DensityCylinder**(*atomgroup: AtomGroup*, *dens: str = 'mass'*, *dim: int = 2*, *zmin: float | None = None*, *zmax: float | None = None*, *bin_width: float = 1*, *rmin: float = 0*, *rmax: float | None = None*, *refgroup: AtomGroup | None = None*, *grouping: str = 'atoms'*, *unwrap: bool = True*, *bin_method: str = 'com'*, *output: str = 'density.dat'*, *concfreq: int = 0*, *jitter: float = 0.0*)

Bases: *ProfileCylinderBase*

Cylindrical partial density profiles.

Calculations are carried out for `mass` ($\mathrm{u} \cdot {}^{-3}$), `number` (${}^{-3}$) or `charge` ($\mathrm{e} \cdot {}^{-3}$) density profiles along certain cartesian axes `[x, y, z]` of the simulation cell. Cell dimensions are allowed to fluctuate in time.

For grouping with respect to `molecules`, `residues` etc., the corresponding centers (i.e., center of mass), taking into account periodic boundary conditions, are calculated. For these calculations molecules will be unwrapped/made whole. Trajectories containing already whole molecules can be run with `unwrap=False` to gain a speedup. For grouping with respect to atoms, the *unwrap* option is always ignored.

For the correlation analysis the 0th bin of the 0th's group profile is used. For further information on the correlation analysis please refer to *maicos.core.base.AnalysisBase* or the *General design* section.

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A `AtomGroup` for which the calculations are performed.
- **unwrap** (*bool*) – When `True`, molecules that are broken due to the periodic boundary conditions are made whole.

  If the input contains molecules that are already whole, speed up the calculation by disabling unwrap. To do so, use the flag `-no-unwrap` when using MAICoS from the command line, or use `unwrap=False` when using MAICoS from the Python interpreter.

  Note: Molecules containing virtual sites (e.g. TIP4P water models) are not currently supported in MDAnalysis. In this case, you need to provide unwrapped trajectory files directly, and disable unwrap. Trajectories can be unwrapped, for example, using the `trjconv` command of GROMACS.
- **refgroup** (*MDAnalysis.core.groups.AtomGroup*) – Reference `AtomGroup` used for the calculation. If `refgroup` is provided, the calculation is performed relative to the center of mass of the AtomGroup. If `refgroup` is `None` the calculations are performed with respect to the center of the (changing) box.
- **jitter** (*float*) – Magnitude of the random noise to add to the atomic positions.

  A jitter can be used to stabilize the aliasing effects sometimes appearing when histogramming data. The jitter value should be about the precision of the trajectory. In that case, using jitter will not alter the results of the histogram. If `jitter = 0.0` (default), the original atomic positions are kept unchanged.

  You can estimate the precision of the positions in your trajectory with *maicos.lib.util.trajectory_precision()*. Note that if the precision is not the same for all frames, the smallest precision should be used.
- **concfreq** (*int*) – When concfreq (for conclude frequency) is larger than `0`, the conclude function is called and the output files are written every `concfreq` frames.
- **dim** (*{0, 1, 2}*) – Dimension for binning (`x=0`, `y=1`, `z=1`).
- **zmin** (*float*) – Minimal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

If `zmin=None`, all coordinates down to the lower cell boundary are taken into account.

- **zmax** (*float*) – Maximal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

  If `zmax = None`, all coordinates up to the upper cell boundary are taken into account.

- **bin_width** (*float*) – Width of the bins (in Å).
- **rmin** (*float*) – Minimal radial coordinate relative to the center of mass of the refgroup for evaluation (in Å).
- **rmax** (*float*) – Maximal radial coordinate relative to the center of mass of the refgroup for evaluation (in Å).

  If `rmax=None`, the box extension is taken.

- **grouping** ({`"atoms"`, `"residues"`, `"segments"`, `"molecules"`, `"fragments"`}) – Atom grouping for the calculations.

  The possible grouping options are the atom positions (in the case where `grouping="atoms"`) or the center of mass of the specified grouping unit (in the case where `grouping="residues"`, `"segments"`, `"molecules"` or `"fragments"`).

- **bin_method** ({`"com"`, `"cog"`, `"coc"`}) – Method for the position binning.

  The possible options are center of mass (`"com"`), center of geometry (`"cog"`), and center of charge (`"coc"`).

- **output** (*str*) – Output filename.
- **dens** ({`"mass"`, `"number"`, `"charge"`}) – density type to be calculated.

`results.`**bin_pos**

Bin positions (in Å) ranging from `rmin` to `rmax`.

Type
numpy.ndarray

`results.`**profile**

Calculated profile.

Type
numpy.ndarray

`results.`**dprofile**

Estimated profile's uncertainity.

Type
numpy.ndarray

DensityPlanar

class maicos.**DensityPlanar**(*atomgroup: AtomGroup*, *dens: str = 'mass'*, *dim: int = 2*, *zmin: float | None = None*, *zmax: float | None = None*, *bin_width: float = 1*, *refgroup: AtomGroup | None = None*, *sym: bool = False*, *grouping: str = 'atoms'*, *unwrap: bool = True*, *bin_method: str = 'com'*, *output: str = 'density.dat'*, *concfreq: int = 0*, *jitter: float = 0.0*)

Bases: *ProfilePlanarBase*

Cartesian partial density profiles.

Calculations are carried out for `mass` ($\mathrm{u} \cdot {}^{-3}$), `number` ($^{-3}$) or `charge` ($\mathrm{e} \cdot {}^{-3}$) density profiles along certain cartesian axes `[x, y, z]` of the simulation cell. Cell dimensions are allowed to fluctuate in time.

For grouping with respect to `molecules`, `residues` etc., the corresponding centers (i.e., center of mass), taking into account periodic boundary conditions, are calculated. For these calculations molecules will be unwrapped/made whole. Trajectories containing already whole molecules can be run with `unwrap=False` to gain a speedup. For grouping with respect to atoms, the *unwrap* option is always ignored.

For the correlation analysis the central bin ($N \backslash 2$) of the 0th's group profile is used. For further information on the correlation analysis please refer to *maicos.core.base.AnalysisBase* or the *General design* section.

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A `AtomGroup` for which the calculations are performed.
- **unwrap** (*bool*) – When `True`, molecules that are broken due to the periodic boundary conditions are made whole.

  If the input contains molecules that are already whole, speed up the calculation by disabling unwrap. To do so, use the flag `-no-unwrap` when using MAICoS from the command line, or use `unwrap=False` when using MAICoS from the Python interpreter.

  Note: Molecules containing virtual sites (e.g. TIP4P water models) are not currently supported in MDAnalysis. In this case, you need to provide unwrapped trajectory files directly, and disable unwrap. Trajectories can be unwrapped, for example, using the `trjconv` command of GROMACS.
- **refgroup** (*MDAnalysis.core.groups.AtomGroup*) – Reference `AtomGroup` used for the calculation. If `refgroup` is provided, the calculation is performed relative to the center of mass of the AtomGroup. If `refgroup` is `None` the calculations are performed with respect to the center of the (changing) box.
- **jitter** (*float*) – Magnitude of the random noise to add to the atomic positions.

  A jitter can be used to stabilize the aliasing effects sometimes appearing when histogramming data. The jitter value should be about the precision of the trajectory. In that case, using jitter will not alter the results of the histogram. If `jitter = 0.0` (default), the original atomic positions are kept unchanged.

  You can estimate the precision of the positions in your trajectory with *maicos.lib.util.trajectory_precision()*. Note that if the precision is not the same for all frames, the smallest precision should be used.
- **concfreq** (*int*) – When concfreq (for conclude frequency) is larger than `0`, the conclude function is called and the output files are written every `concfreq` frames.
- **dim** (*{0, 1, 2}*) – Dimension for binning (`x=0`, `y=1`, `z=1`).
- **zmin** (*float*) – Minimal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

  If `zmin=None`, all coordinates down to the lower cell boundary are taken into account.
- **zmax** (*float*) – Maximal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

  If `zmax = None`, all coordinates up to the upper cell boundary are taken into account.
- **bin_width** (*float*) – Width of the bins (in Å).
- **sym** (*bool*) – Symmetrize the profile. Only works in combination with `refgroup`.

- **grouping** ({"atoms", "residues", "segments", "molecules", "fragments"}) – Atom grouping for the calculations.

  The possible grouping options are the atom positions (in the case where grouping="atoms") or the center of mass of the specified grouping unit (in the case where grouping="residues", "segments", "molecules" or "fragments").

- **bin_method** ({"com", "cog", "coc"}) – Method for the position binning.

  The possible options are center of mass ("com"), center of geometry ("cog"), and center of charge ("coc").

- **output** (*str*) – Output filename.

- **dens** ({"mass", "number", "charge"}) – density type to be calculated.

results.**bin_pos**

Bin positions (in Å) ranging from zmin to zmax.

Type
numpy.ndarray

results.**profile**

Calculated profile.

Type
numpy.ndarray

results.**dprofile**

Estimated profile's uncertainity.

Type
numpy.ndarray

Notes

Partial mass density profiles can be used to calculate the ideal component of the chemical potential. For details, take a look at the corresponding *How-to guide*.

DensitySphere

class maicos.**DensitySphere**(*atomgroup: AtomGroup*, *dens: str = 'mass'*, *bin_width: float = 1*, *rmin: float = 0*, *rmax: float | None = None*, *refgroup: AtomGroup | None = None*, *grouping: str = 'atoms'*, *unwrap: bool = True*, *bin_method: str = 'com'*, *output: str = 'density.dat'*, *concfreq: int = 0*, *jitter: float = 0.0*)

Bases: *ProfileSphereBase*

Spherical partial density profiles.

Calculations are carried out for mass ($\mathrm{u} \cdot {}^{-3}$), number (${}^{-3}$) or charge ($\mathrm{e} \cdot {}^{-3}$) density profiles along certain cartesian axes [x, y, z] of the simulation cell. Cell dimensions are allowed to fluctuate in time.

For grouping with respect to molecules, residues etc., the corresponding centers (i.e., center of mass), taking into account periodic boundary conditions, are calculated. For these calculations molecules will be unwrapped/made whole. Trajectories containing already whole molecules can be run with unwrap=False to gain a speedup. For grouping with respect to atoms, the *unwrap* option is always ignored.

For the correlation analysis the 0th bin of the 0th's group profile is used. For further information on the correlation analysis please refer to *maicos.core.base.AnalysisBase* or the *General design* section.

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A AtomGroup for which the calculations are performed.
- **unwrap** (*bool*) – When True, molecules that are broken due to the periodic boundary conditions are made whole.

  If the input contains molecules that are already whole, speed up the calculation by disabling unwrap. To do so, use the flag `-no-unwrap` when using MAICoS from the command line, or use `unwrap=False` when using MAICoS from the Python interpreter.

  Note: Molecules containing virtual sites (e.g. TIP4P water models) are not currently supported in MDAnalysis. In this case, you need to provide unwrapped trajectory files directly, and disable unwrap. Trajectories can be unwrapped, for example, using the `trjconv` command of GROMACS.
- **refgroup** (*MDAnalysis.core.groups.AtomGroup*) – Reference AtomGroup used for the calculation. If `refgroup` is provided, the calculation is performed relative to the center of mass of the AtomGroup. If `refgroup` is None the calculations are performed with respect to the center of the (changing) box.
- **jitter** (*float*) – Magnitude of the random noise to add to the atomic positions.

  A jitter can be used to stabilize the aliasing effects sometimes appearing when histogramming data. The jitter value should be about the precision of the trajectory. In that case, using jitter will not alter the results of the histogram. If `jitter = 0.0` (default), the original atomic positions are kept unchanged.

  You can estimate the precision of the positions in your trajectory with *maicos.lib.util.trajectory_precision()*. Note that if the precision is not the same for all frames, the smallest precision should be used.
- **concfreq** (*int*) – When concfreq (for conclude frequency) is larger than `0`, the conclude function is called and the output files are written every `concfreq` frames.
- **rmin** (*float*) – Minimal radial coordinate relative to the center of mass of the refgroup for evaluation (in Å).
- **rmax** (*float*) – Maximal radial coordinate relative to the center of mass of the refgroup for evaluation (in Å).

  If `rmax=None`, the box extension is taken.
- **bin_width** (*float*) – Width of the bins (in Å).
- **grouping** ({"atoms", "residues", "segments", "molecules", "fragments"}) – Atom grouping for the calculations.

  The possible grouping options are the atom positions (in the case where `grouping="atoms"`) or the center of mass of the specified grouping unit (in the case where `grouping="residues"`, `"segments"`, `"molecules"` or `"fragments"`).
- **bin_method** ({"com", "cog", "coc"}) – Method for the position binning.

  The possible options are center of mass (`"com"`), center of geometry (`"cog"`), and center of charge (`"coc"`).
- **output** (*str*) – Output filename.
- **dens** ({"mass", "number", "charge"}) – density type to be calculated.

`results.bin_pos`

Bin positions (in Å) ranging from `rmin` to `rmax`.

Type
numpy.ndarray

`results.profile`

Calculated profile.

Type
numpy.ndarray

`results.dprofile`

Estimated profile's uncertainity.

Type
numpy.ndarray

DielectricCylinder

class maicos.**DielectricCylinder**(*atomgroup: AtomGroup*, *bin_width: float = 0.1*, *temperature: float = 300*, *single: bool = False*, *output_prefix: str = 'eps_cyl'*, *refgroup: AtomGroup | None = None*, *concfreq: int = 0*, *jitter: float = 0.0*, *dim: int = 2*, *rmin: float = 0*, *rmax: float | None = None*, *zmin: float | None = None*, *zmax: float | None = None*, *vcutwidth: float = 0.1*, *unwrap: bool = True*)

Bases: *CylinderBase*

Cylindrical dielectric profiles.

Components are calculated along the axial (z) and radial (r) direction either with respect to the center of the simulation box or the center of mass of the refgroup, if provided. The axial direction is selected using the `dim` parameter.

For correlation analysis, the component along the z axis is used. For further information on the correlation analysis please refer to *maicos.core.base.AnalysisBase* or the *General design* section.

For usage please refer to *How-to: Dielectric constant* and for details on the theory see *Dielectric constant measurement*.

Also, please read and cite[1].

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A `AtomGroup` for which the calculations are performed.
- **unwrap** (*bool*) – When `True`, molecules that are broken due to the periodic boundary conditions are made whole.

  If the input contains molecules that are already whole, speed up the calculation by disabling unwrap. To do so, use the flag `-no-unwrap` when using MAICoS from the command line, or use `unwrap=False` when using MAICoS from the Python interpreter.

  Note: Molecules containing virtual sites (e.g. TIP4P water models) are not currently supported in MDAnalysis. In this case, you need to provide unwrapped trajectory files directly, and disable unwrap. Trajectories can be unwrapped, for example, using the `trjconv` command of GROMACS.

[1] Philip Loche, Cihan Ayaz, Alexander Schlaich, Yuki Uematsu, and Roland R. Netz. Giant Axial Dielectric Response in Water-Filled Nanotubes and Effective Electrostatic Ion–Ion Interactions from a Tensorial Dielectric Model. *J. Phys. Chem. B*, 123(50):10850–10857, December 2019. doi:10.1021/acs.jpcb.9b09269.

- **refgroup** (*MDAnalysis.core.groups.AtomGroup*) – Reference AtomGroup used for the calculation. If refgroup is provided, the calculation is performed relative to the center of mass of the AtomGroup. If refgroup is None the calculations are performed with respect to the center of the (changing) box.
- **jitter** (*float*) – Magnitude of the random noise to add to the atomic positions.

  A jitter can be used to stabilize the aliasing effects sometimes appearing when histogramming data. The jitter value should be about the precision of the trajectory. In that case, using jitter will not alter the results of the histogram. If jitter = 0.0 (default), the original atomic positions are kept unchanged.

  You can estimate the precision of the positions in your trajectory with *maicos.lib.util.trajectory_precision()*. Note that if the precision is not the same for all frames, the smallest precision should be used.
- **concfreq** (*int*) – When concfreq (for conclude frequency) is larger than 0, the conclude function is called and the output files are written every concfreq frames.
- **dim** (*{0, 1, 2}*) – Dimension for binning (x=0, y=1, z=1).
- **zmin** (*float*) – Minimal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

  If zmin=None, all coordinates down to the lower cell boundary are taken into account.
- **zmax** (*float*) – Maximal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

  If zmax = None, all coordinates up to the upper cell boundary are taken into account.
- **bin_width** (*float*) – Width of the bins (in Å).
- **rmin** (*float*) – Minimal radial coordinate relative to the center of mass of the refgroup for evaluation (in Å).
- **rmax** (*float*) – Maximal radial coordinate relative to the center of mass of the refgroup for evaluation (in Å).

  If rmax=None, the box extension is taken.
- **temperature** (*float*) – Reference temperature (K)
- **single** (*bool*) – For a single chain of molecules the average of M is zero. This flag sets <M> = 0.

results.**bin_pos**

Bin positions (in Å) ranging from rmin to rmax.

Type
numpy.ndarray

results.**eps_z**

Reduced axial dielectric profile $(\varepsilon_z - 1)$ of the selected atomgroup

Type
numpy.ndarray

results.**deps_z**

Estimated uncertainty of axial dielectric profile

Type
numpy.ndarray

`results.eps_r`

Reduced inverse radial dielectric profile $(\varepsilon_r^{-1} - 1)$

Type
numpy.ndarray

`results.deps_r`

Estimated uncertainty of inverse radial dielectric profile

Type
numpy.ndarray

References

`save()` → None

Save results of analysis to file specified by `output`.

DielectricPlanar

`class maicos.DielectricPlanar(`*atomgroup: AtomGroup, dim: int = 2, zmin: float | None = None, zmax: float | None = None, bin_width: float = 0.5, refgroup: AtomGroup | None = None, is_3d: bool = False, sym: bool = False, unwrap: bool = True, temperature: float = 300, output_prefix: str = 'eps', concfreq: int = 0, jitter: float = 0.0, vcutwidth: float = 0.1*`)`

Bases: `PlanarBase`

Planar dielectric profiles.

For usage please refer to *How-to: Dielectric constant* and for details on the theory see *Dielectric constant measurement*.

For correlation analysis, the norm of the parallel total dipole moment is used. For further information on the correlation analysis please refer to `maicos.core.base.AnalysisBase` or the *General design* section.

Also, please read and cite Schlaich *et al.*[1] and Refs.[2],[3].

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A `AtomGroup` for which the calculations are performed.
- **unwrap** (*bool*) – When `True`, molecules that are broken due to the periodic boundary conditions are made whole.

  If the input contains molecules that are already whole, speed up the calculation by disabling unwrap. To do so, use the flag `-no-unwrap` when using MAICoS from the command line, or use `unwrap=False` when using MAICoS from the Python interpreter.

  Note: Molecules containing virtual sites (e.g. TIP4P water models) are not currently supported in MDAnalysis. In this case, you need to provide unwrapped trajectory files directly, and disable unwrap. Trajectories can be unwrapped, for example, using the `trjconv` command of GROMACS.

[1] Alexander Schlaich, Ernst W. Knapp, and Roland R. Netz. Water Dielectric Effects in Planar Confinement. *Phys. Rev. Lett.*, 117(4):048001, July 2016. doi:10.1103/PhysRevLett.117.048001.

[2] Philip Loche, Cihan Ayaz, Amanuel Wolde-Kidan, Alexander Schlaich, and Roland R. Netz. Universal and Nonuniversal Aspects of Electrostatics in Aqueous Nanoconfinement. *J. Phys. Chem. B*, 124(21):4365–4371, May 2020. doi:10.1021/acs.jpcb.0c01967.

[3] Douwe Jan Bonthuis, Stephan Gekle, and Roland R. Netz. Profile of the Static Permittivity Tensor of Water at Interfaces: Consequences for Capacitance, Hydration Interaction and Ion Adsorption. *Langmuir*, 28(20):7679–7694, 2012. doi:10.1021/la2051564.

- **refgroup** (*MDAnalysis.core.groups.AtomGroup*) – Reference AtomGroup used for the calculation. If refgroup is provided, the calculation is performed relative to the center of mass of the AtomGroup. If refgroup is None the calculations are performed with respect to the center of the (changing) box.
- **jitter** (*float*) – Magnitude of the random noise to add to the atomic positions.

  A jitter can be used to stabilize the aliasing effects sometimes appearing when histogramming data. The jitter value should be about the precision of the trajectory. In that case, using jitter will not alter the results of the histogram. If jitter = 0.0 (default), the original atomic positions are kept unchanged.

  You can estimate the precision of the positions in your trajectory with *maicos.lib.util.trajectory_precision()*. Note that if the precision is not the same for all frames, the smallest precision should be used.
- **concfreq** (*int*) – When concfreq (for conclude frequency) is larger than 0, the conclude function is called and the output files are written every concfreq frames.
- **dim** (*{0, 1, 2}*) – Dimension for binning (x=0, y=1, z=1).
- **zmin** (*float*) – Minimal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

  If zmin=None, all coordinates down to the lower cell boundary are taken into account.
- **zmax** (*float*) – Maximal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

  If zmax = None, all coordinates up to the upper cell boundary are taken into account.
- **bin_width** (*float*) – Width of the bins (in Å).
- **is_3d** (*bool*) – Use 3d-periodic boundary conditions, i.e., include the dipole correction for the interaction between periodic images[4].
- **sym** (*bool*) – Symmetrize the profile. Only works in combination with refgroup.
- **temperature** (*float*) – Reference temperature (K)
- **output_prefix** (*str*) – Prefix for output files.
- **vcutwidth** (*float*) – Spacing of virtual cuts (bins) along the parallel directions.

results.**bin_pos**

Bin positions (in Å) ranging from zmin to zmax.

Type
numpy.ndarray

results.**eps_par**

Reduced parallel dielectric profile $(\varepsilon_{\parallel} - 1)$ of the selected AtomGroup

Type
numpy.ndarray

results.**deps_par**

Uncertainty of parallel dielectric profile

Type
numpy.ndarray

[4] Harry A. Stern and Scott E. Feller. Calculation of the dielectric permittivity profile for a nonuniform system: Application to a lipid bilayer simulation. *The Journal of Chemical Physics*, 118(7):3401–3412, February 2003. doi:10.1063/1.1537244.

results.**eps_par_self**

Reduced self contribution of parallel dielectric profile ($\varepsilon_{\parallel,\mathrm{self}} - 1$)

Type
numpy.ndarray

results.**eps_par_coll**

Reduced collective contribution of parallel dielectric profile ($\varepsilon_{\parallel,\mathrm{coll}} - 1$)

Type
numpy.ndarray

results.**eps_perp**

Reduced inverse perpendicular dielectric profile ($\varepsilon_{\perp}^{-1} - 1$)

Type
numpy.ndarray

results.**deps_perp**

Uncertainty of inverse perpendicular dielectric profile

Type
numpy.ndarray

results.**eps_perp_self**

Reduced self contribution of the inverse perpendicular dielectric profile ($\varepsilon_{\perp,\mathrm{self}}^{-1} - 1$)

Type
numpy.ndarray

results.**eps_perp_coll**

Reduced collective contribution of the inverse perpendicular dielectric profile ($\varepsilon_{\perp,\mathrm{coll}}^{-1} - 1$)

Type
numpy.ndarray

References

save() → None

Save results of analysis to file specified by `output`.

DielectricSpectrum

class maicos.**DielectricSpectrum**(*atomgroup: AtomGroup*, *refgroup: AtomGroup | None = None*, *unwrap: bool = True*, *concfreq: int = 0*, *temperature: float = 300*, *output_prefix: str = ''*, *segs: int = 20*, *df: float | None = None*, *bins: int = 200*, *binafter: float = 20*, *nobin: bool = False*, *jitter: float = 0.0*)

Bases: *AnalysisBase*

Linear dielectric spectrum.

This module, given a molecular dynamics trajectory, produces a *.txt* file containing the complex dielectric function as a function of the (linear, not radial - i.e., ν or f, rather than ω) frequency, along with the associated standard deviations. The algorithm is based on the Fluctuation Dissipation Relation: $\chi(f) = -1/(3Vk_BT\varepsilon_0)\mathcal{L}[\theta(t)\langle P(0)dP(t)/dt\rangle]$, where $\mathcal{L}$ is the Laplace transformation.

Note: The polarization time series and the average system volume are also saved.

Please read and cite[1].

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A AtomGroup for which the calculations are performed.
- **unwrap** (*bool*) – When True, molecules that are broken due to the periodic boundary conditions are made whole.

  If the input contains molecules that are already whole, speed up the calculation by disabling unwrap. To do so, use the flag `-no-unwrap` when using MAICoS from the command line, or use `unwrap=False` when using MAICoS from the Python interpreter.

  Note: Molecules containing virtual sites (e.g. TIP4P water models) are not currently supported in MDAnalysis. In this case, you need to provide unwrapped trajectory files directly, and disable unwrap. Trajectories can be unwrapped, for example, using the `trjconv` command of GROMACS.
- **refgroup** (*MDAnalysis.core.groups.AtomGroup*) – Reference AtomGroup used for the calculation. If `refgroup` is provided, the calculation is performed relative to the center of mass of the AtomGroup. If `refgroup` is None the calculations are performed with respect to the center of the (changing) box.
- **jitter** (*float*) – Magnitude of the random noise to add to the atomic positions.

  A jitter can be used to stabilize the aliasing effects sometimes appearing when histogramming data. The jitter value should be about the precision of the trajectory. In that case, using jitter will not alter the results of the histogram. If `jitter = 0.0` (default), the original atomic positions are kept unchanged.

  You can estimate the precision of the positions in your trajectory with *maicos.lib.util.trajectory_precision()*. Note that if the precision is not the same for all frames, the smallest precision should be used.
- **concfreq** (*int*) – When concfreq (for conclude frequency) is larger than `0`, the conclude function is called and the output files are written every `concfreq` frames.
- **temperature** (*float*) – Reference temperature (K)
- **output_prefix** (*str*) – Prefix for output files.
- **segs** (*int*) – Sets the number of segments the trajectory is broken into.
- **df** (*float*) – The desired frequency spacing in THz. This determines the minimum frequency about which there is data. Overrides *segs* option.
- **bins** (*int*) – Determines the number of bins used for data averaging; (this parameter sets the upper limit). The data are by default binned logarithmically. This helps to reduce noise, particularly in the high-frequency domain, and also prevents plot files from being too large.
- **binafter** (*int*) – The number of low-frequency data points that are left unbinned.
- **nobin** (*bool*) – Prevents the data from being binned altogether. This can result in very large plot files and errors.

results

[1] Shane Carlson, Florian N. Brünig, Philip Loche, Douwe Jan Bonthuis, and Roland R. Netz. Exploring the Absorption Spectrum of Simulated Water from MHz to Infrared. *J. Phys. Chem. A*, 124(27):5599–5605, July 2020. doi:10.1021/acs.jpca.0c04063.

References

save() → None

Save results of analysis to file specified by `output`.

DielectricSphere

class maicos.**DielectricSphere**(*atomgroup: AtomGroup*, *bin_width: float = 0.1*, *temperature: float = 300*, *output_prefix: str = 'eps_sph'*, *refgroup: AtomGroup | None = None*, *concfreq: int = 0*, *jitter: float = 0.0*, *rmin: float = 0*, *rmax: float | None = None*, *unwrap: bool = True*)

Bases: *SphereBase*

Spherical dielectric profiles.

Components are calculated along the radial (r) direction either with respect to the center of the simulation box or the center of mass of the refgroup, if provided.

For usage, please refer to *How-to: Dielectric constant* and for details on the theory see *Dielectric constant measurement*.

For correlation analysis, the radial (r) component is used. For further information on the correlation analysis please refer to *maicos.core.base.AnalysisBase* or the *General design* section.

Also, please read and cite[1].

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A `AtomGroup` for which the calculations are performed.
- **unwrap** (*bool*) – When `True`, molecules that are broken due to the periodic boundary conditions are made whole.

  If the input contains molecules that are already whole, speed up the calculation by disabling unwrap. To do so, use the flag `-no-unwrap` when using MAICoS from the command line, or use `unwrap=False` when using MAICoS from the Python interpreter.

  Note: Molecules containing virtual sites (e.g. TIP4P water models) are not currently supported in MDAnalysis. In this case, you need to provide unwrapped trajectory files directly, and disable unwrap. Trajectories can be unwrapped, for example, using the `trjconv` command of GROMACS.
- **refgroup** (*MDAnalysis.core.groups.AtomGroup*) – Reference `AtomGroup` used for the calculation. If `refgroup` is provided, the calculation is performed relative to the center of mass of the AtomGroup. If `refgroup` is `None` the calculations are performed with respect to the center of the (changing) box.
- **jitter** (*float*) – Magnitude of the random noise to add to the atomic positions.

  A jitter can be used to stabilize the aliasing effects sometimes appearing when histogramming data. The jitter value should be about the precision of the trajectory. In that case, using jitter will not alter the results of the histogram. If `jitter = 0.0` (default), the original atomic positions are kept unchanged.

  You can estimate the precision of the positions in your trajectory with *maicos.lib.util.trajectory_precision()*. Note that if the precision is not the same for all frames, the smallest precision should be used.

[1] Christian Schaaf and Stephan Gekle. Dielectric response of the water hydration layer around spherical solutes. *Phys. Rev. E*, 92(3):032718, September 2015. doi:10.1103/PhysRevE.92.032718.

- **concfreq** (*int*) – When concfreq (for conclude frequency) is larger than `0`, the conclude function is called and the output files are written every `concfreq` frames.
- **rmin** (*float*) – Minimal radial coordinate relative to the center of mass of the refgroup for evaluation (in Å).
- **rmax** (*float*) – Maximal radial coordinate relative to the center of mass of the refgroup for evaluation (in Å).

  If `rmax=None`, the box extension is taken.
- **bin_width** (*float*) – Width of the bins (in Å).
- **temperature** (*float*) – Reference temperature (K)
- **output_prefix** (*str*) – Prefix for output files.

`results.`**bin_pos**

Bin positions (in Å) ranging from `rmin` to `rmax`.

Type
numpy.ndarray

`results.`**eps_rad**

Reduced inverse radial dielectric profile ($\varepsilon_r^{-1} - 1$)

Type
numpy.ndarray

`results.`**deps_rad**

Uncertainty of inverse radial dielectric profile

Type
numpy.ndarray

References

save() → None

Save results of analysis to file specified by `output`.

DipoleAngle

class maicos.**DipoleAngle**(*atomgroup: AtomGroup*, *unwrap: bool = False*, *refgroup: AtomGroup | None = None*, *concfreq: int = 0*, *grouping: str = 'residues'*, *pdim: int = 2*, *output: str = 'dipangle.dat'*, *jitter: float = 0.0*)

Bases: *AnalysisBase*

Angle timeseries of dipole moments with respect to an axis.

The analysis can be applied to study the orientational dynamics of water molecules during an excitation pulse. For more details read Elgabarty *et al.*[1].

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A `AtomGroup` for which the calculations are performed.

[1] Hossam Elgabarty, Tobias Kampfrath, Douwe Jan Bonthuis, Vasileios Balos, Naveen Kumar Kaliannan, Philip Loche, Roland R. Netz, Martin Wolf, Thomas D. Kühne, and Mohsen Sajadi. Energy transfer within the hydrogen bonding network of water following resonant terahertz excitation. *Science Advances*, 6(17):eaay7074, April 2020. doi:10.1126/sciadv.aay7074.

- **unwrap** (*bool*) – When `True`, molecules that are broken due to the periodic boundary conditions are made whole.

  If the input contains molecules that are already whole, speed up the calculation by disabling unwrap. To do so, use the flag `-no-unwrap` when using MAICoS from the command line, or use `unwrap=False` when using MAICoS from the Python interpreter.

  Note: Molecules containing virtual sites (e.g. TIP4P water models) are not currently supported in MDAnalysis. In this case, you need to provide unwrapped trajectory files directly, and disable unwrap. Trajectories can be unwrapped, for example, using the `trjconv` command of GROMACS.

- **refgroup** (*MDAnalysis.core.groups.AtomGroup*) – Reference `AtomGroup` used for the calculation. If `refgroup` is provided, the calculation is performed relative to the center of mass of the AtomGroup. If `refgroup` is `None` the calculations are performed with respect to the center of the (changing) box.

- **jitter** (*float*) – Magnitude of the random noise to add to the atomic positions.

  A jitter can be used to stabilize the aliasing effects sometimes appearing when histogramming data. The jitter value should be about the precision of the trajectory. In that case, using jitter will not alter the results of the histogram. If `jitter = 0.0` (default), the original atomic positions are kept unchanged.

  You can estimate the precision of the positions in your trajectory with *maicos.lib.util.trajectory_precision()*. Note that if the precision is not the same for all frames, the smallest precision should be used.

- **concfreq** (*int*) – When concfreq (for conclude frequency) is larger than `0`, the conclude function is called and the output files are written every `concfreq` frames.

- **grouping** ({"atoms", "residues", "segments", "molecules", "fragments"}) – Atom grouping for the calculations.

  The possible grouping options are the atom positions (in the case where `grouping="atoms"`) or the center of mass of the specified grouping unit (in the case where `grouping="residues"`, `"segments"`, `"molecules"` or `"fragments"`).

- **pdim** (*{0, 1, 2}*) – direction of the projection

- **output** (*str*) – Output filename.

`results.t`

time (ps).

Type
numpy.ndarray

`resulst.cos_theta_i`

Average cos between dipole and axis.

Type
numpy.ndarray

`resulst.cos_theta_ii`

Average cos of the dipoles and axis.

Type
numpy.ndarray

`resulst.cos_theta_ij`

Product cos of dipole i and cos of dipole j (`i != j`).

Type
numpy.ndarray

References

`save()` → None

Save results of analysis to file specified by `output`.

DiporderCylinder

`class maicos.DiporderCylinder(`*atomgroup: AtomGroup, dim: int = 2, zmin: float | None = None, zmax: float | None = None, bin_width: float = 1, rmin: float = 0, rmax: float | None = None, refgroup: AtomGroup | None = None, grouping: str = 'residues', unwrap: bool = True, bin_method: str = 'com', output: str = 'diporder_cylinder.dat', concfreq: int = 0, pdim: str = 'r', order_parameter: str = 'P0', jitter: float = 0.0*`)`

Bases: *ProfileCylinderBase*

Cylindrical dipolar order parameters.

Calculations include the projected dipole density $P_0(z)\cos([z])$, the dipole orientation $\cos([z])$, the squared dipole orientation $\cos([z])$ and the number density (z).

For the correlation analysis the 0th bin of the 0th's group profile is used. For further information on the correlation analysis please refer to *maicos.core.base.AnalysisBase* or the *General design* section.

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A `AtomGroup` for which the calculations are performed.
- **unwrap** (*bool*) – When `True`, molecules that are broken due to the periodic boundary conditions are made whole.

  If the input contains molecules that are already whole, speed up the calculation by disabling unwrap. To do so, use the flag `-no-unwrap` when using MAICoS from the command line, or use `unwrap=False` when using MAICoS from the Python interpreter.

  Note: Molecules containing virtual sites (e.g. TIP4P water models) are not currently supported in MDAnalysis. In this case, you need to provide unwrapped trajectory files directly, and disable unwrap. Trajectories can be unwrapped, for example, using the `trjconv` command of GROMACS.
- **refgroup** (*MDAnalysis.core.groups.AtomGroup*) – Reference `AtomGroup` used for the calculation. If `refgroup` is provided, the calculation is performed relative to the center of mass of the AtomGroup. If `refgroup` is `None` the calculations are performed with respect to the center of the (changing) box.
- **jitter** (*float*) – Magnitude of the random noise to add to the atomic positions.

  A jitter can be used to stabilize the aliasing effects sometimes appearing when histogramming data. The jitter value should be about the precision of the trajectory. In that case, using jitter will not alter the results of the histogram. If `jitter = 0.0` (default), the original atomic positions are kept unchanged.

You can estimate the precision of the positions in your trajectory with *maicos.lib.util.trajectory_precision()*. Note that if the precision is not the same for all frames, the smallest precision should be used.

- **concfreq** (*int*) – When concfreq (for conclude frequency) is larger than `0`, the conclude function is called and the output files are written every `concfreq` frames.
- **dim** (*{0, 1, 2}*) – Dimension for binning (`x=0`, `y=1`, `z=1`).
- **zmin** (*float*) – Minimal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

  If `zmin=None`, all coordinates down to the lower cell boundary are taken into account.
- **zmax** (*float*) – Maximal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

  If `zmax = None`, all coordinates up to the upper cell boundary are taken into account.
- **bin_width** (*float*) – Width of the bins (in Å).
- **rmin** (*float*) – Minimal radial coordinate relative to the center of mass of the refgroup for evaluation (in Å).
- **rmax** (*float*) – Maximal radial coordinate relative to the center of mass of the refgroup for evaluation (in Å).

  If `rmax=None`, the box extension is taken.
- **grouping** ({`"atoms"`, `"residues"`, `"segments"`, `"molecules"`, `"fragments"`}) – Atom grouping for the calculations.

  The possible grouping options are the atom positions (in the case where `grouping="atoms"`) or the center of mass of the specified grouping unit (in the case where `grouping="residues"`, `"segments"`, `"molecules"` or `"fragments"`).
- **bin_method** ({`"com"`, `"cog"`, `"coc"`}) – Method for the position binning.

  The possible options are center of mass (`"com"`), center of geometry (`"cog"`), and center of charge (`"coc"`).
- **output** (*str*) – Output filename.
- **pdim** ({`"r"`, `"z"`}) – direction of the projection
- **order_parameter** ({`"P0"`, `"cos_theta"`, `"cos_2_theta"`}) –

  Order parameter to be calculated:

  - `"P0"`: total dipole moment projected on an axis
  - `"cos_theta"`: cosine of the dipole moment with an axis
  - `"cos_2_theta"`: squred cosine with an axis.

`results.`**bin_pos**

Bin positions (in Å) ranging from `rmin` to `rmax`.

Type
numpy.ndarray

`results.`**profile**

Calculated profile.

Type
numpy.ndarray

`results.dprofile`

Estimated profile's uncertainity.

Type
numpy.ndarray

DiporderPlanar

class maicos.**DiporderPlanar**(*atomgroup: AtomGroup*, *dim: int = 2*, *zmin: float | None = None*, *zmax: float | None = None*, *bin_width: float = 1*, *refgroup: AtomGroup | None = None*, *sym: bool = False*, *grouping: str = 'residues'*, *unwrap: bool = True*, *bin_method: str = 'com'*, *output: str = 'diporder_planar.dat'*, *concfreq: int = 0*, *pdim: int = 2*, *order_parameter: str = 'P0'*, *jitter: float = 0.0*)

Bases: *ProfilePlanarBase*

Cartesian dipolar order parameters.

Calculations include the projected dipole density $P_0(z)\cos([z])$, the dipole orientation $\cos([z])$, the squared dipole orientation $\cos\left([z]\right)$ and the number density (z).

For the correlation analysis the central bin ($N\backslash 2$) of the 0th's group profile is used. For further information on the correlation analysis please refer to *maicos.core.base.AnalysisBase* or the *General design* section.

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A AtomGroup for which the calculations are performed.
- **unwrap** (*bool*) – When True, molecules that are broken due to the periodic boundary conditions are made whole.

  If the input contains molecules that are already whole, speed up the calculation by disabling unwrap. To do so, use the flag `-no-unwrap` when using MAICoS from the command line, or use `unwrap=False` when using MAICoS from the Python interpreter.

  Note: Molecules containing virtual sites (e.g. TIP4P water models) are not currently supported in MDAnalysis. In this case, you need to provide unwrapped trajectory files directly, and disable unwrap. Trajectories can be unwrapped, for example, using the `trjconv` command of GROMACS.
- **refgroup** (*MDAnalysis.core.groups.AtomGroup*) – Reference AtomGroup used for the calculation. If `refgroup` is provided, the calculation is performed relative to the center of mass of the AtomGroup. If `refgroup` is None the calculations are performed with respect to the center of the (changing) box.
- **jitter** (*float*) – Magnitude of the random noise to add to the atomic positions.

  A jitter can be used to stabilize the aliasing effects sometimes appearing when histogramming data. The jitter value should be about the precision of the trajectory. In that case, using jitter will not alter the results of the histogram. If `jitter = 0.0` (default), the original atomic positions are kept unchanged.

  You can estimate the precision of the positions in your trajectory with *maicos.lib.util.trajectory_precision()*. Note that if the precision is not the same for all frames, the smallest precision should be used.
- **concfreq** (*int*) – When concfreq (for conclude frequency) is larger than `0`, the conclude function is called and the output files are written every `concfreq` frames.
- **dim** (*{0, 1, 2}*) – Dimension for binning (`x=0`, `y=1`, `z=1`).

- **zmin** (*float*) – Minimal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

  If `zmin=None`, all coordinates down to the lower cell boundary are taken into account.

- **zmax** (*float*) – Maximal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

  If `zmax = None`, all coordinates up to the upper cell boundary are taken into account.

- **bin_width** (*float*) – Width of the bins (in Å).

- **sym** (*bool*) – Symmetrize the profile. Only works in combination with `refgroup`.

- **grouping** ({"atoms", "residues", "segments", "molecules", "fragments"}) – Atom grouping for the calculations.

  The possible grouping options are the atom positions (in the case where `grouping="atoms"`) or the center of mass of the specified grouping unit (in the case where `grouping="residues"`, `"segments"`, `"molecules"` or `"fragments"`).

- **bin_method** ({"com", "cog", "coc"}) – Method for the position binning.

  The possible options are center of mass (`"com"`), center of geometry (`"cog"`), and center of charge (`"coc"`).

- **output** (*str*) – Output filename.

- **pdim** (*{0, 1, 2}*) – direction of the projection

- **order_parameter** ({"P0", "cos_theta", "cos_2_theta"}) –

  Order parameter to be calculated:

  - `"P0"`: total dipole moment projected on an axis
  - `"cos_theta"`: cosine of the dipole moment with an axis
  - `"cos_2_theta"`: squred cosine with an axis.

`results.bin_pos`

Bin positions (in Å) ranging from `zmin` to `zmax`.

Type
numpy.ndarray

`results.profile`

Calculated profile.

Type
numpy.ndarray

`results.dprofile`

Estimated profile's uncertainity.

Type
numpy.ndarray

DiporderSphere

class maicos.**DiporderSphere**(*atomgroup: AtomGroup*, *bin_width: float = 1*, *rmin: float = 0*, *rmax: float | None = None*, *refgroup: AtomGroup | None = None*, *grouping: str = 'residues'*, *unwrap: bool = True*, *bin_method: str = 'com'*, *output: str = 'diporder_sphere.dat'*, *concfreq: int = 0*, *order_parameter: str = 'P0'*, *jitter: float = 0.0*)

Bases: *ProfileSphereBase*

Spherical dipolar order parameters.

Calculations include the projected dipole density $P_0(z)\cos([z])$, the dipole orientation $\cos([z])$, the squared dipole orientation $\cos([z])$ and the number density (z).

For the correlation analysis the 0th bin of the 0th's group profile is used. For further information on the correlation analysis please refer to *maicos.core.base.AnalysisBase* or the *General design* section.

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A AtomGroup for which the calculations are performed.
- **unwrap** (*bool*) – When True, molecules that are broken due to the periodic boundary conditions are made whole.

  If the input contains molecules that are already whole, speed up the calculation by disabling unwrap. To do so, use the flag `-no-unwrap` when using MAICoS from the command line, or use `unwrap=False` when using MAICoS from the Python interpreter.

  Note: Molecules containing virtual sites (e.g. TIP4P water models) are not currently supported in MDAnalysis. In this case, you need to provide unwrapped trajectory files directly, and disable unwrap. Trajectories can be unwrapped, for example, using the `trjconv` command of GROMACS.
- **refgroup** (*MDAnalysis.core.groups.AtomGroup*) – Reference AtomGroup used for the calculation. If `refgroup` is provided, the calculation is performed relative to the center of mass of the AtomGroup. If `refgroup` is None the calculations are performed with respect to the center of the (changing) box.
- **jitter** (*float*) – Magnitude of the random noise to add to the atomic positions.

  A jitter can be used to stabilize the aliasing effects sometimes appearing when histogramming data. The jitter value should be about the precision of the trajectory. In that case, using jitter will not alter the results of the histogram. If `jitter = 0.0` (default), the original atomic positions are kept unchanged.

  You can estimate the precision of the positions in your trajectory with *maicos.lib.util.trajectory_precision()*. Note that if the precision is not the same for all frames, the smallest precision should be used.
- **concfreq** (*int*) – When concfreq (for conclude frequency) is larger than `0`, the conclude function is called and the output files are written every `concfreq` frames.
- **dim** (*{0, 1, 2}*) – Dimension for binning (`x=0`, `y=1`, `z=1`).
- **zmin** (*float*) – Minimal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

  If `zmin=None`, all coordinates down to the lower cell boundary are taken into account.
- **zmax** (*float*) – Maximal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

If `zmax = None`, all coordinates up to the upper cell boundary are taken into account.

- **bin_width** (*float*) – Width of the bins (in Å).
- **rmin** (*float*) – Minimal radial coordinate relative to the center of mass of the refgroup for evaluation (in Å).
- **rmax** (*float*) – Maximal radial coordinate relative to the center of mass of the refgroup for evaluation (in Å).

  If `rmax=None`, the box extension is taken.
- **grouping** ({"atoms", "residues", "segments", "molecules", "fragments"}) – Atom grouping for the calculations.

  The possible grouping options are the atom positions (in the case where `grouping="atoms"`) or the center of mass of the specified grouping unit (in the case where `grouping="residues"`, `"segments"`, `"molecules"` or `"fragments"`).
- **bin_method** ({"com", "cog", "coc"}) – Method for the position binning.

  The possible options are center of mass (`"com"`), center of geometry (`"cog"`), and center of charge (`"coc"`).
- **output** (*str*) – Output filename.
- **order_parameter** ({"P0", "cos_theta", "cos_2_theta"}) –

  Order parameter to be calculated:

  - `"P0"`: total dipole moment projected on an axis
  - `"cos_theta"`: cosine of the dipole moment with an axis
  - `"cos_2_theta"`: squred cosine with an axis.

`results.bin_pos`

Bin positions (in Å) ranging from `rmin` to `rmax`.

Type
numpy.ndarray

`results.profile`

Calculated profile.

Type
numpy.ndarray

`results.dprofile`

Estimated profile's uncertainity.

Type
numpy.ndarray

DiporderStructureFactor

class maicos.**DiporderStructureFactor**(*atomgroup: AtomGroup*, *bin_method: str = 'com'*, *grouping: str = 'molecules'*, *refgroup: AtomGroup | None = None*, *unwrap: bool = True*, *jitter: float = 0.0*, *concfreq: int = 0*, *qmin: float = 0*, *qmax: float = 6*, *dq: float = 0.01*, *output: str = 'sq.dat'*)

Bases: *AnalysisBase*

Structure factor for dipoles.

Extension the standard structure factor $S(q)$ by weighting it with different the normalized dipole moment $\hat{\boldsymbol{\mu}}$ of a `group` according to

$$S(q)_{\hat{\boldsymbol{\mu}}\hat{\boldsymbol{\mu}}} = \left\langle \frac{1}{N} \sum_{i,j=1}^{N} \hat{\mu}_i \hat{\mu}_j \, \exp(-i\boldsymbol{q} \cdot [\boldsymbol{r}_i - \boldsymbol{r}_j]) \right\rangle$$

For the correlation time estimation the module will use the value of the structure factor with the smallest possible q value.

For an detailed example on the usage refer to the *how-to on dipolar correlation functions*. For general details on the theory behind the structure factor refer to *Small-angle X-ray scattering*.

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A `AtomGroup` for which the calculations are performed.
- **unwrap** (*bool*) – When `True`, molecules that are broken due to the periodic boundary conditions are made whole.

  If the input contains molecules that are already whole, speed up the calculation by disabling unwrap. To do so, use the flag `-no-unwrap` when using MAICoS from the command line, or use `unwrap=False` when using MAICoS from the Python interpreter.

  Note: Molecules containing virtual sites (e.g. TIP4P water models) are not currently supported in MDAnalysis. In this case, you need to provide unwrapped trajectory files directly, and disable unwrap. Trajectories can be unwrapped, for example, using the `trjconv` command of GROMACS.
- **refgroup** (*MDAnalysis.core.groups.AtomGroup*) – Reference `AtomGroup` used for the calculation. If `refgroup` is provided, the calculation is performed relative to the center of mass of the AtomGroup. If `refgroup` is `None` the calculations are performed with respect to the center of the (changing) box.
- **jitter** (*float*) – Magnitude of the random noise to add to the atomic positions.

  A jitter can be used to stabilize the aliasing effects sometimes appearing when histogramming data. The jitter value should be about the precision of the trajectory. In that case, using jitter will not alter the results of the histogram. If `jitter = 0.0` (default), the original atomic positions are kept unchanged.

  You can estimate the precision of the positions in your trajectory with *maicos.lib.util.trajectory_precision()*. Note that if the precision is not the same for all frames, the smallest precision should be used.
- **concfreq** (*int*) – When concfreq (for conclude frequency) is larger than `0`, the conclude function is called and the output files are written every `concfreq` frames.
- **qmin** (*float*) – Starting q (1/Å)
- **qmax** (*float*) – Ending q (1/Å)

- **dq** (*float*) – bin_width (1/Å)
- **output** (*str*) – Output filename.

`results.q`

length of binned q-vectors

Type
numpy.ndarray

`results.structure_factors`

Structure factor

Type
numpy.ndarray

save() → None

Save results of analysis to file specified by `output`.

KineticEnergy

class maicos.**KineticEnergy**(*atomgroup: AtomGroup*, *unwrap: bool = False*, *refgroup: AtomGroup | None = None*, *jitter: float = 0.0*, *concfreq: int = 0*, *output: str = 'ke.dat'*, *refpoint: str = 'com'*)

Bases: *AnalysisBase*

Kinetic energy timeseries.

The kinetic energy function computes the translational and rotational kinetic energy with respect to molecular center (center of mass, center of charge) of a molecular dynamics simulation trajectory.

The analysis can be applied to study the dynamics of water molecules during an excitation pulse. For more details read Elgabarty *et al.*[1].

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A `AtomGroup` for which the calculations are performed.
- **unwrap** (*bool*) – When `True`, molecules that are broken due to the periodic boundary conditions are made whole.

  If the input contains molecules that are already whole, speed up the calculation by disabling unwrap. To do so, use the flag `-no-unwrap` when using MAICoS from the command line, or use `unwrap=False` when using MAICoS from the Python interpreter.

  Note: Molecules containing virtual sites (e.g. TIP4P water models) are not currently supported in MDAnalysis. In this case, you need to provide unwrapped trajectory files directly, and disable unwrap. Trajectories can be unwrapped, for example, using the `trjconv` command of GROMACS.
- **refgroup** (*MDAnalysis.core.groups.AtomGroup*) – Reference `AtomGroup` used for the calculation. If `refgroup` is provided, the calculation is performed relative to the center of mass of the AtomGroup. If `refgroup` is `None` the calculations are performed with respect to the center of the (changing) box.

[1] Hossam Elgabarty, Tobias Kampfrath, Douwe Jan Bonthuis, Vasileios Balos, Naveen Kumar Kaliannan, Philip Loche, Roland R. Netz, Martin Wolf, Thomas D. Kühne, and Mohsen Sajadi. Energy transfer within the hydrogen bonding network of water following resonant terahertz excitation. *Science Advances*, 6(17):eaay7074, April 2020. doi:10.1126/sciadv.aay7074.

- **jitter** (*float*) – Magnitude of the random noise to add to the atomic positions.

  A jitter can be used to stabilize the aliasing effects sometimes appearing when histogramming data. The jitter value should be about the precision of the trajectory. In that case, using jitter will not alter the results of the histogram. If `jitter = 0.0` (default), the original atomic positions are kept unchanged.

  You can estimate the precision of the positions in your trajectory with *maicos.lib.util.trajectory_precision()*. Note that if the precision is not the same for all frames, the smallest precision should be used.

- **concfreq** (*int*) – When concfreq (for conclude frequency) is larger than `0`, the conclude function is called and the output files are written every `concfreq` frames.

- **refpoint** (*str*) – reference point for molecular center: center of mass (`"com"`) or center of charge (`"coc"`).

- **output** (*str*) – Output filename.

`results.t`

time (ps).

Type
numpy.ndarray

`results.trans`

translational kinetic energy (kJ/mol).

Type
numpy.ndarray

`results.rot`

rotational kinetic energy (kJ/mol).

Type
numpy.ndarray

References

save() → None

Save results of analysis to file specified by `output`.

PDFCylinder

class maicos.modules.pdfcylinder.**PDFCylinder**(*g1: AtomGroup*, *g2: AtomGroup | None = None*, *bin_width_pdf_z: float = 0.3*, *bin_width_pdf_phi: float = 0.1*, *drwidth: float = 0.1*, *dmin: float | None = None*, *dmax: float | None = None*, *density: bool = False*, *origin: ndarray | None = None*, *bin_method: str = 'com'*, *unwrap: bool = False*, *refgroup: AtomGroup | None = None*, *jitter: float = 0.0*, *concfreq: int = 0*, *dim: int = 2*, *zmin: float | None = None*, *zmax: float | None = None*, *rmin: float = 0*, *rmax: float | None = None*, *bin_width: float = 1*, *output: str = 'pdf.dat'*)

Bases: *CylinderBase*

Shell-wise one-dimensional (cylindrical) pair distribution functions.

The one-dimensional pair distribution functions $g_{1\mathrm{d}}(\phi)$ and $g_{1\mathrm{d}}(z)$ describes the pair distribution to particles which lie on the same cylinder along the angular and axial directions respectively. These functions can be used in cylindrical systems that are inhomogeneous along radial coordinate, and homogeneous in the angular and axial directions. It gives the average number density of $g2$ as a function of angular and axial distances respectively from a $g1$ atom. Then the angular pair distribution function is

$$g_{1\mathrm{d}}(\phi) = \left\langle \sum_{i}^{N_{g1}} \sum_{j}^{N_{g2}} \delta(\phi - \phi_{ij}) \delta(R_{ij}) \delta(z_{ij}) \right\rangle$$

And the axial pair distribution function is

$$g_{1\mathrm{d}}(z) = \left\langle \sum_{i}^{N_{g1}} \sum_{j}^{N_{g2}} \delta(z - z_{ij}) \delta(R_{ij}) \delta(\phi_{ij}) \right\rangle$$

Even though due to consistency reasons the results are called pair distribution functions the output is not unitless. The default output is is in dimension of number/volume in $^{-3}$. If `density` is set to `True`, the output is normalised by the density of $g2$.

Parameters

- **g1** (*MDAnalysis.core.groups.AtomGroup*) – First AtomGroup.
- **g2** (*MDAnalysis.core.groups.AtomGroup*) – Second AtomGroup.
- **pdf_z_bin_width** (*float*) – Binwidth of bins in the histogram of the axial PDF (Å).
- **pdf_phi_bin_width** (*float*) – Binwidth of bins in the histogram of the angular PDF (Å).
- **drwidth** (*float*) – radial width of a PDF cylindrical shell (Å), and axial or angular (arc) slices.
- **dmin** (*float*) – the minimum pairwise distance between 'g1' and 'g2' (Å).
- **dmax** (*float*) – the maximum pairwise distance between 'g1' and 'g2' (Å).
- **density** (*bool*) – normalise the PDF by the density of 'g2' ($^{-3}$).
- **origin** (*numpy.ndarray*) – Set origin of the cylindrical coordinate system (x,y,z). If `None` the origin will be set according to the `refgroup` parameter.
- **bin_method** ({"com", "cog", "coc"}) – Method for the position binning.

  The possible options are center of mass ("com"), center of geometry ("cog"), and center of charge ("coc").
- **unwrap** (*bool*) – When `True`, molecules that are broken due to the periodic boundary conditions are made whole.

  If the input contains molecules that are already whole, speed up the calculation by disabling unwrap. To do so, use the flag `-no-unwrap` when using MAICoS from the command line, or use `unwrap=False` when using MAICoS from the Python interpreter.

  Note: Molecules containing virtual sites (e.g. TIP4P water models) are not currently supported in MDAnalysis. In this case, you need to provide unwrapped trajectory files directly, and disable unwrap. Trajectories can be unwrapped, for example, using the `trjconv` command of GROMACS.
- **refgroup** (*MDAnalysis.core.groups.AtomGroup*) – Reference `AtomGroup` used for the calculation. If `refgroup` is provided, the calculation is performed relative to the center of mass of the AtomGroup. If `refgroup` is `None` the calculations are performed with respect to the center of the (changing) box.

- **jitter** (*float*) – Magnitude of the random noise to add to the atomic positions.

  A jitter can be used to stabilize the aliasing effects sometimes appearing when histogramming data. The jitter value should be about the precision of the trajectory. In that case, using jitter will not alter the results of the histogram. If `jitter = 0.0` (default), the original atomic positions are kept unchanged.

  You can estimate the precision of the positions in your trajectory with *maicos.lib.util.trajectory_precision()*. Note that if the precision is not the same for all frames, the smallest precision should be used.

- **concfreq** (*int*) – When concfreq (for conclude frequency) is larger than `0`, the conclude function is called and the output files are written every `concfreq` frames.

- **dim** (*{0, 1, 2}*) – Dimension for binning (`x=0`, `y=1`, `z=1`).

- **zmin** (*float*) – Minimal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

  If `zmin=None`, all coordinates down to the lower cell boundary are taken into account.

- **zmax** (*float*) – Maximal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

  If `zmax = None`, all coordinates up to the upper cell boundary are taken into account.

- **bin_width** (*float*) – Width of the bins (in Å).

- **rmin** (*float*) – Minimal radial coordinate relative to the center of mass of the refgroup for evaluation (in Å).

- **rmax** (*float*) – Maximal radial coordinate relative to the center of mass of the refgroup for evaluation (in Å).

  If `rmax=None`, the box extension is taken.

- **output** (*str*) – Output filename.

`results.`**bin_pos**

Bin positions (in Å) ranging from `rmin` to `rmax`.

Type
numpy.ndarray

`results.`**phi_bins**

Angular distances to which the PDF is calculated with shape (*pdf_nbins*) (Å)

Type
numpy.ndarray

`results.`**z_bins**

axial distances to which the PDF is calculated with shape (*pdf_nbins*) (Å)

Type
numpy.ndarray

`results.`**phi_pdf**

Angular PDF with shape (*pdf_nbins*, *n_bins*) ($Å^{-3}$)

Type
numpy.ndarray

`results.z_pdf`

Axial PDF with shape (*pdf_nbins*, *n_bins*) ($Å^{-3}$)

Type
numpy.ndarray

`save()` → None

Save results of analysis to file specified by `output`.

PDFPlanar

`class maicos.PDFPlanar(g1: AtomGroup, g2: AtomGroup | None = None, pdf_bin_width: float = 0.3, dzheight: float = 0.1, dmin: float = 0.0, dmax: float | None = None, bin_method: str = 'com', output: str = 'pdf.dat', unwrap: bool = False, refgroup: AtomGroup | None = None, concfreq: int = 0, jitter: float = 0.0, dim: int = 2, zmin: float | None = None, zmax: float | None = None, bin_width: float = 1)`

Bases: *PlanarBase*

Slab-wise planar 2D pair distribution functions.

The pair distribution function $g_{2D}(r)$ describes the spatial correlation between atoms in g_1 and atoms in g_2, which lie in the same plane. It gives the average number density of g_2 atoms as a function of lateral distance r from a centered g_1 atom. PDFPlanar can be used in systems that are inhomogeneous along one axis, and homogeneous in a plane. In fully homogeneous systems and in the limit of small 'dzheight' Δz, it is the same as the well known three dimensional PDF.

The planar PDF is defined by

$$g_{2\mathrm{D}}(r) = \left\langle \frac{1}{N_{g1}} \cdot \sum_{i}^{N_{g1}} \sum_{j}^{N_{g2}} \frac{1}{2\pi r} \delta(r - r_{ij}) \delta(z_{ij}) \right\rangle .$$

where the brackets $\langle \cdot \rangle$ denote the ensemble average. $\delta(r - r_{ij})$ counts the g_2 atoms at distance r from atom i. $\delta(z_{ij})$ ensures that only atoms, which lie in the same plane $z_i = z_j$, are considered for the PDF.

Discretized for computational purposes the equation reads as

$$g_{2\mathrm{D}}(r) = \frac{1}{N_{g1}} \cdot \sum_{i}^{N_{g1}} \frac{\text{count } g_2 \text{ in } \Delta V_i(r)}{\Delta V_i(r)}.$$

where $\Delta V_i(r)$ is a ring around atom i, with inner radius $r - \frac{\Delta r}{2}$, outer radius $r + \frac{\Delta r}{2}$ and height $2\Delta z$.

As the density to normalise the PDF with is unknown, the output is in the dimension of number/volume in 1/Å^3.

Functionally, PDFPlanar bins all pairwise g_1-g_2 distances, where the z distance is smaller than 'dzheight' in a histogram.

For a more detailed explanation refer to *Explanation: PDF* and *PDFPlanar Derivation*

Parameters

- **g1** (*MDAnalysis.core.groups.AtomGroup*) – First AtomGroup.
- **g2** (*MDAnalysis.core.groups.AtomGroup*) – Second AtomGroup.
- **pdf_bin_width** (*float*) – Binwidth of bins in the histogram of the PDF (Å).
- **dzheight** (*float*) – dz height of a PDF slab Δz (Å). Δz is introduced to discretize the delta function $\delta(z_{ij})$. It is the maximum z distance between atoms which are considered to lie in

the same plane. In the limit of $\Delta z \to 0$, PDFPlanar reaches the continous limit. However, if Δz is too small, there are no atoms in `g2` to sample. We recommend a choice of Δz that is 1/10th of a bond length.

- **dmin** (*float*) – Minimum pairwise distance between `g1` and `g2` (Å).
- **dmax** (*float*) – Maximum pairwise distance between `g1` and `g2` (Å).
- **bin_method** ({"com", "cog", "coc"}) – Method for the position binning.

  The possible options are center of mass ("com"), center of geometry ("cog"), and center of charge ("coc").
- **output** (*str*) – Output filename.
- **unwrap** (*bool*) – When `True`, molecules that are broken due to the periodic boundary conditions are made whole.

  If the input contains molecules that are already whole, speed up the calculation by disabling unwrap. To do so, use the flag `-no-unwrap` when using MAICoS from the command line, or use `unwrap=False` when using MAICoS from the Python interpreter.

  Note: Molecules containing virtual sites (e.g. TIP4P water models) are not currently supported in MDAnalysis. In this case, you need to provide unwrapped trajectory files directly, and disable unwrap. Trajectories can be unwrapped, for example, using the `trjconv` command of GROMACS.
- **refgroup** (*MDAnalysis.core.groups.AtomGroup*) – Reference `AtomGroup` used for the calculation. If `refgroup` is provided, the calculation is performed relative to the center of mass of the AtomGroup. If `refgroup` is `None` the calculations are performed with respect to the center of the (changing) box.
- **jitter** (*float*) – Magnitude of the random noise to add to the atomic positions.

  A jitter can be used to stabilize the aliasing effects sometimes appearing when histogramming data. The jitter value should be about the precision of the trajectory. In that case, using jitter will not alter the results of the histogram. If `jitter = 0.0` (default), the original atomic positions are kept unchanged.

  You can estimate the precision of the positions in your trajectory with *maicos.lib.util.trajectory_precision()*. Note that if the precision is not the same for all frames, the smallest precision should be used.
- **concfreq** (*int*) – When concfreq (for conclude frequency) is larger than `0`, the conclude function is called and the output files are written every `concfreq` frames.
- **dim** (*{0, 1, 2}*) – Dimension for binning (`x=0`, `y=1`, `z=1`).
- **zmin** (*float*) – Minimal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

  If `zmin=None`, all coordinates down to the lower cell boundary are taken into account.
- **zmax** (*float*) – Maximal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

  If `zmax = None`, all coordinates up to the upper cell boundary are taken into account.
- **bin_width** (*float*) – Width of the bins (in Å).

`results.`**bin_pos**

Bin positions (in Å) ranging from `zmin` to `zmax`.

Type
numpy.ndarray

`results.bins`

distances to which the PDF is calculated with shape (pdf_nbins) (Å)

Type
numpy.ndarray

`results.pdf`

PDF with shape (pdf_nbins, n_bins) (1/Å^3)

Type
np.ndrray

`save()` → None

Save results of analysis to file specified by `output`.

RDFDiporder

`class maicos.RDFDiporder(`*g1: AtomGroup*, *g2: AtomGroup | None = None*, *bin_width: float = 0.1*, *rmin: float = 0.0*, *rmax: float = 15.0*, *bin_method: str = 'com'*, *norm: str = 'rdf'*, *grouping: str = 'residues'*, *unwrap: bool = True*, *refgroup: AtomGroup | None = None*, *jitter: float = 0.0*, *concfreq: int = 0*, *output: str = 'diporderrdf.dat'*`)`

Bases: *AnalysisBase*

Spherical Radial Distribution function between dipoles.

The implementation is heavily inspired by `MDAnalysis.analysis.rdf.InterRDF` and is according to Zhang and Galli[1] given by

$$g_{\hat{\boldsymbol{\mu}},\hat{\boldsymbol{\mu}}}(r) = \frac{1}{N}\left\langle \sum_i \frac{1}{n_i(r)} \sum_{j=1}^{n_i(r)} (\hat{\boldsymbol{\mu}}_i \cdot \hat{\boldsymbol{\mu}}_j) \right\rangle$$

where $\hat{\boldsymbol{\mu}}$ is the normalized dipole moment of a `grouping` and $n_i(r)$ is the number of dipoles within a spherical shell of distance r and $r + \delta r$ from dipole i.

For the correlation time estimation the module will use the value of the RDF with the largest possible r value.

For an detailed example on the usage refer to the *how-to on dipolar correlation functions*.

Parameters

- **g1** (*MDAnalysis.core.groups.AtomGroup*) – First AtomGroup.
- **g2** (*MDAnalysis.core.groups.AtomGroup*) – Second AtomGroup.
- **bin_width** (*float*) – Width of the bins (in Å).
- **rmin** (*float*) – Minimal radial coordinate relative to the center of mass of the refgroup for evaluation (in Å).
- **rmax** (*float*) – Maximal radial coordinate relative to the center of mass of the refgroup for evaluation (in Å).

  If `rmax=None`, the box extension is taken.

[1] Cui Zhang and Giulia Galli. Dipolar correlations in liquid water. *The Journal of Chemical Physics*, 141(8):084504, August 2014. doi:10.1063/1.4893638.

- **bin_method** ({"com", "cog", "coc"}) – Method for the position binning.

  The possible options are center of mass ("com"), center of geometry ("cog"), and center of charge ("coc").

- **norm** (*str, {'rdf', 'density', 'none'}*) – For 'rdf' calculate $g_{ab}(r)$. For 'density' the single group density $n_{ab}(r)$ is computed. 'none' computes the number of particles occurences in each spherical shell.

- **grouping** ({"atoms", "residues", "segments", "molecules", "fragments"}) – Atom grouping for the calculations.

  The possible grouping options are the atom positions (in the case where grouping="atoms") or the center of mass of the specified grouping unit (in the case where grouping="residues", "segments", "molecules" or "fragments").

- **unwrap** (*bool*) – When True, molecules that are broken due to the periodic boundary conditions are made whole.

  If the input contains molecules that are already whole, speed up the calculation by disabling unwrap. To do so, use the flag -no-unwrap when using MAICoS from the command line, or use unwrap=False when using MAICoS from the Python interpreter.

  Note: Molecules containing virtual sites (e.g. TIP4P water models) are not currently supported in MDAnalysis. In this case, you need to provide unwrapped trajectory files directly, and disable unwrap. Trajectories can be unwrapped, for example, using the trjconv command of GROMACS.

- **refgroup** (*MDAnalysis.core.groups.AtomGroup*) – Reference AtomGroup used for the calculation. If refgroup is provided, the calculation is performed relative to the center of mass of the AtomGroup. If refgroup is None the calculations are performed with respect to the center of the (changing) box.

- **jitter** (*float*) – Magnitude of the random noise to add to the atomic positions.

  A jitter can be used to stabilize the aliasing effects sometimes appearing when histogramming data. The jitter value should be about the precision of the trajectory. In that case, using jitter will not alter the results of the histogram. If jitter = 0.0 (default), the original atomic positions are kept unchanged.

  You can estimate the precision of the positions in your trajectory with *maicos.lib.util.trajectory_precision()*. Note that if the precision is not the same for all frames, the smallest precision should be used.

- **concfreq** (*int*) – When concfreq (for conclude frequency) is larger than 0, the conclude function is called and the output files are written every concfreq frames.

- **output** (*str*) – Output filename.

results.**bins**

radial distances to which the RDF is calculated with shape (rdf_nbins) (Å)

Type
numpy.ndarray

results.**rdf**

RDF either in eÅ^{-2} if norm is "rdf" or "density" or eÅ if norm is "none".

Type
numpy.ndarray

References

save() → None

Save results of analysis to file specified by `output`.

Saxs

class maicos.**Saxs**(*atomgroup: AtomGroup*, *unwrap: bool = False*, *refgroup: AtomGroup | None = None*, *jitter: float = 0.0*, *concfreq: int = 0*, *bin_spectrum: bool = True*, *qmin: float = 0*, *qmax: float = 6*, *dq: float = 0.1*, *thetamin: float = 0*, *thetamax: float = 180*, *output: str = 'sq.dat'*)

Bases: *AnalysisBase*

Small angle X-Ray scattering intensities (SAXS).

This module computes the structure factor $S(q)$, the scattering intensity $I(q)$ and their corresponding scattering vectors q. For a system containing only one element the structure factor and the scattering intensity are connected via the form factor $f(q)$

$$I(q) = [f(q)]^2 S(q)$$

For more details on the theory behind this module see *Small-angle X-ray scattering*.

By default the scattering vectors $\boldsymbol{q}$ are binned according to their length q using a bin width given by `dq`. Setting the option `bin_spectrum=False`, also the raw scattering vectors and their corresponding Miller indices can be saved. Saving the scattering vectors and Miller indices is only possible when the box vectors are constant in the whole trajectory (NVT) since for changing cells the same Miller indices correspond to different scattering vectors.

Analyzed scattering vectors q can be restricted by a minimal and maximal angle with the z-axis. For `0` and `180`, all possible vectors are taken into account. To obtain the scattering intensities, the structure factor is normalized by an element-specific form factor based on Cromer-Mann parameters Prince[1].

For the correlation time estimation the module will use the value of the scattering intensity with the largest possible q value.

For an example on the usage refer to *How-to: SAXS*.

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A `AtomGroup` for which the calculations are performed.
- **unwrap** (*bool*) – When `True`, molecules that are broken due to the periodic boundary conditions are made whole.

  If the input contains molecules that are already whole, speed up the calculation by disabling unwrap. To do so, use the flag `-no-unwrap` when using MAICoS from the command line, or use `unwrap=False` when using MAICoS from the Python interpreter.

  Note: Molecules containing virtual sites (e.g. TIP4P water models) are not currently supported in MDAnalysis. In this case, you need to provide unwrapped trajectory files directly, and disable unwrap. Trajectories can be unwrapped, for example, using the `trjconv` command of GROMACS.
- **refgroup** (*MDAnalysis.core.groups.AtomGroup*) – Reference `AtomGroup` used for the calculation. If `refgroup` is provided, the calculation is performed relative to the center of

[1] E. Prince. *International Tables for Crystallography,Volume C: Mathematical, Physical and Chemical Tables*. Springer, Dordrecht, 3rd ed. edition edition, January 2004. ISBN 978-1-4020-1900-5.

mass of the AtomGroup. If `refgroup` is `None` the calculations are performed with respect to the center of the (changing) box.

- **jitter** (*float*) – Magnitude of the random noise to add to the atomic positions.

  A jitter can be used to stabilize the aliasing effects sometimes appearing when histogramming data. The jitter value should be about the precision of the trajectory. In that case, using jitter will not alter the results of the histogram. If `jitter = 0.0` (default), the original atomic positions are kept unchanged.

  You can estimate the precision of the positions in your trajectory with *maicos.lib.util.trajectory_precision()*. Note that if the precision is not the same for all frames, the smallest precision should be used.

- **concfreq** (*int*) – When concfreq (for conclude frequency) is larger than `0`, the conclude function is called and the output files are written every `concfreq` frames.
- **bin_spectrum** (*bool*) – Bin the spectrum. If `False` Miller indices of q-vector are returned. Only works for NVT simulations.
- **qmin** (*float*) – Starting q (1/Å)
- **qmax** (*float*) – Ending q (1/Å)
- **dq** (*float*) – bin_width (1/Å)
- **thetamin** (*float*) – Minimal angle (°) between the q vectors and the z-axis.
- **thetamax** (*float*) – Maximal angle (°) between the q vectors and the z-axis.
- **output** (*str*) – Output filename.

`results.`**scattering_vectors**

Length of the binned scattering vectors.

Type
numpy.ndarray

`results.`**miller_indices**

Miller indices of q-vector (only available if `bin_spectrum==False`).

Type
numpy.ndarray

`results.`**struture_factors**

structure factors $S(q)$

Type
numpy.ndarray

`results.`**scattering_intensities**

scattering intensities $I(q)$

Type
numpy.ndarray

References

save() → None

Save results of analysis to file specified by `output`.

TemperaturePlanar

class maicos.**TemperaturePlanar**(*atomgroup: AtomGroup*, *dim: int = 2*, *zmin: float | None = None*, *zmax: float | None = None*, *bin_width: float = 1*, *refgroup: AtomGroup | None = None*, *sym: bool = False*, *grouping: str = 'atoms'*, *unwrap: bool = True*, *bin_method: str = 'com'*, *output: str = 'temperature.dat'*, *concfreq: int = 0*, *jitter: float = 0.0*)

Bases: *ProfilePlanarBase*

Temperature profiles in a cartesian geometry.

Currently only atomistic temperature profiles are supported. Therefore grouping per molecule, segment, residue, or fragment is not possible.

For the correlation analysis the central bin ($N \backslash 2$) of the 0th's group profile is used. For further information on the correlation analysis please refer to *maicos.core.base.AnalysisBase* or the *General design* section.

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A `AtomGroup` for which the calculations are performed.
- **unwrap** (*bool*) – When `True`, molecules that are broken due to the periodic boundary conditions are made whole.

  If the input contains molecules that are already whole, speed up the calculation by disabling unwrap. To do so, use the flag `-no-unwrap` when using MAICoS from the command line, or use `unwrap=False` when using MAICoS from the Python interpreter.

  Note: Molecules containing virtual sites (e.g. TIP4P water models) are not currently supported in MDAnalysis. In this case, you need to provide unwrapped trajectory files directly, and disable unwrap. Trajectories can be unwrapped, for example, using the `trjconv` command of GROMACS.
- **refgroup** (*MDAnalysis.core.groups.AtomGroup*) – Reference `AtomGroup` used for the calculation. If `refgroup` is provided, the calculation is performed relative to the center of mass of the AtomGroup. If `refgroup` is `None` the calculations are performed with respect to the center of the (changing) box.
- **jitter** (*float*) – Magnitude of the random noise to add to the atomic positions.

  A jitter can be used to stabilize the aliasing effects sometimes appearing when histogramming data. The jitter value should be about the precision of the trajectory. In that case, using jitter will not alter the results of the histogram. If `jitter = 0.0` (default), the original atomic positions are kept unchanged.

  You can estimate the precision of the positions in your trajectory with *maicos.lib.util.trajectory_precision()*. Note that if the precision is not the same for all frames, the smallest precision should be used.
- **concfreq** (*int*) – When concfreq (for conclude frequency) is larger than `0`, the conclude function is called and the output files are written every `concfreq` frames.
- **dim** (*{0, 1, 2}*) – Dimension for binning (`x=0`, `y=1`, `z=1`).

- **zmin** (*float*) – Minimal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

  If `zmin=None`, all coordinates down to the lower cell boundary are taken into account.

- **zmax** (*float*) – Maximal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

  If `zmax = None`, all coordinates up to the upper cell boundary are taken into account.

- **bin_width** (*float*) – Width of the bins (in Å).

- **sym** (*bool*) – Symmetrize the profile. Only works in combination with `refgroup`.

- **grouping** ({`"atoms"`, `"residues"`, `"segments"`, `"molecules"`, `"fragments"`}) – Atom grouping for the calculations.

  The possible grouping options are the atom positions (in the case where `grouping="atoms"`) or the center of mass of the specified grouping unit (in the case where `grouping="residues"`, `"segments"`, `"molecules"` or `"fragments"`).

- **bin_method** ({`"com"`, `"cog"`, `"coc"`}) – Method for the position binning.

  The possible options are center of mass (`"com"`), center of geometry (`"cog"`), and center of charge (`"coc"`).

- **output** (*str*) – Output filename.

`results.bin_pos`

Bin positions (in Å) ranging from `zmin` to `zmax`.

Type
numpy.ndarray

`results.profile`

Calculated profile.

Type
numpy.ndarray

`results.dprofile`

Estimated profile's uncertainity.

Type
numpy.ndarray

VelocityCylinder

class maicos.**VelocityCylinder**(*atomgroup: AtomGroup*, *dim: int = 2*, *zmin: float | None = None*, *zmax: float | None = None*, *bin_width: int = 1*, *rmin: float = 0*, *rmax: float | None = None*, *refgroup: AtomGroup | None = None*, *grouping: str = 'atoms'*, *unwrap: bool = True*, *bin_method: str = 'com'*, *output: str = 'velocity.dat'*, *concfreq: int = 0*, *jitter: float = 0.0*, *vdim: int = 0*, *flux: bool = False*)

Bases: *ProfileCylinderBase*

Cartesian velocity profile across a cylinder.

Reads in coordinates and velocities from a trajectory and calculates a velocity $[/\mathrm{ps}]$ or a flux per unit area $[^{-2}\,\mathrm{ps}^{-1}]$ profile along a given axis.

The `grouping` keyword gives you fine control over the velocity profile, e.g. you can choose atomar or molecular velocities. Note that if the first one is employed for complex compounds, usually a contribution corresponding to the vorticity appears in the profile.

For the correlation analysis the 0th bin of the 0th's group profile is used. For further information on the correlation analysis please refer to *maicos.core.base.AnalysisBase* or the *General design* section.

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A `AtomGroup` for which the calculations are performed.
- **unwrap** (*bool*) – When `True`, molecules that are broken due to the periodic boundary conditions are made whole.

  If the input contains molecules that are already whole, speed up the calculation by disabling unwrap. To do so, use the flag `-no-unwrap` when using MAICoS from the command line, or use `unwrap=False` when using MAICoS from the Python interpreter.

  Note: Molecules containing virtual sites (e.g. TIP4P water models) are not currently supported in MDAnalysis. In this case, you need to provide unwrapped trajectory files directly, and disable unwrap. Trajectories can be unwrapped, for example, using the `trjconv` command of GROMACS.
- **refgroup** (*MDAnalysis.core.groups.AtomGroup*) – Reference `AtomGroup` used for the calculation. If `refgroup` is provided, the calculation is performed relative to the center of mass of the AtomGroup. If `refgroup` is `None` the calculations are performed with respect to the center of the (changing) box.
- **jitter** (*float*) – Magnitude of the random noise to add to the atomic positions.

  A jitter can be used to stabilize the aliasing effects sometimes appearing when histogramming data. The jitter value should be about the precision of the trajectory. In that case, using jitter will not alter the results of the histogram. If `jitter = 0.0` (default), the original atomic positions are kept unchanged.

  You can estimate the precision of the positions in your trajectory with *maicos.lib.util.trajectory_precision()*. Note that if the precision is not the same for all frames, the smallest precision should be used.
- **concfreq** (*int*) – When concfreq (for conclude frequency) is larger than `0`, the conclude function is called and the output files are written every `concfreq` frames.
- **dim** (*{0, 1, 2}*) – Dimension for binning (`x=0`, `y=1`, `z=1`).
- **zmin** (*float*) – Minimal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

  If `zmin=None`, all coordinates down to the lower cell boundary are taken into account.
- **zmax** (*float*) – Maximal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

  If `zmax = None`, all coordinates up to the upper cell boundary are taken into account.
- **bin_width** (*float*) – Width of the bins (in Å).
- **rmin** (*float*) – Minimal radial coordinate relative to the center of mass of the refgroup for evaluation (in Å).
- **rmax** (*float*) – Maximal radial coordinate relative to the center of mass of the refgroup for evaluation (in Å).

  If `rmax=None`, the box extension is taken.

- **grouping** ({"atoms", "residues", "segments", "molecules", "fragments"}) – Atom grouping for the calculations.

  The possible grouping options are the atom positions (in the case where grouping="atoms") or the center of mass of the specified grouping unit (in the case where grouping="residues", "segments", "molecules" or "fragments").

- **bin_method** ({"com", "cog", "coc"}) – Method for the position binning.

  The possible options are center of mass ("com"), center of geometry ("cog"), and center of charge ("coc").

- **output** (*str*) – Output filename. vdim : {0, 1, 2} Dimension for velocity binning (x=0, y=1, z=1).

- **flux** (*bool*) – Calculate the flux ($[^2/\mathrm{ps}]$) instead of the velocity.

results.**bin_pos**

Bin positions (in Å) ranging from rmin to rmax.

Type
numpy.ndarray

results.**profile**

Calculated profile.

Type
numpy.ndarray

results.**dprofile**

Estimated profile's uncertainity.

Type
numpy.ndarray

VelocityPlanar

class maicos.**VelocityPlanar**(*atomgroup: AtomGroup*, *dim: int = 2*, *zmin: float | None = None*, *zmax: float | None = None*, *bin_width: int = 1*, *refgroup: AtomGroup | None = None*, *sym: bool = False*, *grouping: str = 'atoms'*, *unwrap: bool = True*, *bin_method: str = 'com'*, *output: str = 'velocity.dat'*, *concfreq: int = 0*, *vdim: int = 0*, *flux: bool = False*, *jitter: float = 0.0*)

Bases: *ProfilePlanarBase*

Velocity profiles in a cartesian geometry.

Reads in coordinates and velocities from a trajectory and calculates a velocity $[/\mathrm{ps}]$ or a flux per unit area $[^{-2}\,\mathrm{ps}^{-1}]$ profile along a given axis.

The grouping keyword gives you fine control over the velocity profile, e.g. you can choose atomar or molecular velocities. Note that if the first one is employed for complex compounds, usually a contribution corresponding to the vorticity appears in the profile.

For the correlation analysis the central bin ($N\backslash 2$) of the 0th's group profile is used. For further information on the correlation analysis please refer to *maicos.core.base.AnalysisBase* or the *General design* section.

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A AtomGroup for which the calculations are performed.

- **unwrap** (*bool*) – When `True`, molecules that are broken due to the periodic boundary conditions are made whole.

  If the input contains molecules that are already whole, speed up the calculation by disabling unwrap. To do so, use the flag `-no-unwrap` when using MAICoS from the command line, or use `unwrap=False` when using MAICoS from the Python interpreter.

  Note: Molecules containing virtual sites (e.g. TIP4P water models) are not currently supported in MDAnalysis. In this case, you need to provide unwrapped trajectory files directly, and disable unwrap. Trajectories can be unwrapped, for example, using the `trjconv` command of GROMACS.

- **refgroup** (*MDAnalysis.core.groups.AtomGroup*) – Reference `AtomGroup` used for the calculation. If `refgroup` is provided, the calculation is performed relative to the center of mass of the AtomGroup. If `refgroup` is `None` the calculations are performed with respect to the center of the (changing) box.

- **jitter** (*float*) – Magnitude of the random noise to add to the atomic positions.

  A jitter can be used to stabilize the aliasing effects sometimes appearing when histogramming data. The jitter value should be about the precision of the trajectory. In that case, using jitter will not alter the results of the histogram. If `jitter = 0.0` (default), the original atomic positions are kept unchanged.

  You can estimate the precision of the positions in your trajectory with *maicos.lib.util.trajectory_precision()*. Note that if the precision is not the same for all frames, the smallest precision should be used.

- **concfreq** (*int*) – When concfreq (for conclude frequency) is larger than `0`, the conclude function is called and the output files are written every `concfreq` frames.

- **dim** (*{0, 1, 2}*) – Dimension for binning (`x=0`, `y=1`, `z=1`).

- **zmin** (*float*) – Minimal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

  If `zmin=None`, all coordinates down to the lower cell boundary are taken into account.

- **zmax** (*float*) – Maximal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

  If `zmax = None`, all coordinates up to the upper cell boundary are taken into account.

- **bin_width** (*float*) – Width of the bins (in Å).

- **sym** (*bool*) – Symmetrize the profile. Only works in combination with `refgroup`.

- **grouping** ({`"atoms"`, `"residues"`, `"segments"`, `"molecules"`, `"fragments"`}) – Atom grouping for the calculations.

  The possible grouping options are the atom positions (in the case where `grouping="atoms"`) or the center of mass of the specified grouping unit (in the case where `grouping="residues"`, `"segments"`, `"molecules"` or `"fragments"`).

- **bin_method** ({`"com"`, `"cog"`, `"coc"`}) – Method for the position binning.

  The possible options are center of mass (`"com"`), center of geometry (`"cog"`), and center of charge (`"coc"`).

- **output** (*str*) – Output filename. vdim : {0, 1, 2} Dimension for velocity binning (`x=0`, `y=1`, `z=1`).

- **$FLUX_PARAMETER}** –

`results.bin_pos`

Bin positions (in Å) ranging from `zmin` to `zmax`.

Type
numpy.ndarray

`results.profile`

Calculated profile.

Type
numpy.ndarray

`results.dprofile`

Estimated profile's uncertainity.

Type
numpy.ndarray

5.3.2 API Documentation

Core classes

These Modules build the core of other MAICoS modules.

Base classes

AnalysisBase

class maicos.core.**AnalysisBase**(*atomgroup: AtomGroup*, *unwrap: bool*, *refgroup: None | AtomGroup*, *jitter: float*, *concfreq: int*, *wrap_compound: str*)

Bases: `_Runner`, `AnalysisBase`

Base class derived from MDAnalysis for defining multi-frame analysis.

The class is designed as a template for creating multi-frame analyses. This class will automatically take care of setting up the trajectory reader for iterating, and it offers to show a progress meter. Computed results are stored inside the *results* attribute. To define a new analysis, *AnalysisBase* needs to be subclassed and `_single_frame()` must be defined. It is also possible to define `_prepare()` and `_conclude()` for pre- and post-processing. All results should be stored as attributes of the `MDAnalysis.analysis.base.Results` container.

During the analysis, the correlation time of an observable can be estimated to ensure that calculated errors are reasonable. For this, the `_single_frame()` method has to return a single `float`. For details on the computation of the correlation and its further analysis refer to *maicos.lib.util.correlation_analysis()*.

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A `AtomGroup` for which the calculations are performed.
- **unwrap** (*bool*) – When `True`, molecules that are broken due to the periodic boundary conditions are made whole.

  If the input contains molecules that are already whole, speed up the calculation by disabling unwrap. To do so, use the flag `-no-unwrap` when using MAICoS from the command line, or use `unwrap=False` when using MAICoS from the Python interpreter.

Note: Molecules containing virtual sites (e.g. TIP4P water models) are not currently supported in MDAnalysis. In this case, you need to provide unwrapped trajectory files directly, and disable unwrap. Trajectories can be unwrapped, for example, using the `trjconv` command of GROMACS.

- **refgroup** (*MDAnalysis.core.groups.AtomGroup*) – Reference `AtomGroup` used for the calculation. If `refgroup` is provided, the calculation is performed relative to the center of mass of the AtomGroup. If `refgroup` is `None` the calculations are performed with respect to the center of the (changing) box.
- **jitter** (*float*) – Magnitude of the random noise to add to the atomic positions.

  A jitter can be used to stabilize the aliasing effects sometimes appearing when histogramming data. The jitter value should be about the precision of the trajectory. In that case, using jitter will not alter the results of the histogram. If `jitter = 0.0` (default), the original atomic positions are kept unchanged.

  You can estimate the precision of the positions in your trajectory with *maicos.lib.util.trajectory_precision()*. Note that if the precision is not the same for all frames, the smallest precision should be used.
- **concfreq** (*int*) – When concfreq (for conclude frequency) is larger than `0`, the conclude function is called and the output files are written every `concfreq` frames.
- **wrap_compound** (*str*) – The group which will be kept together through the wrap processes. Allowed values are: `"atoms"`, `"group"`, `"residues"`, `"segments"`, `"molecules"`, or `"fragments"`.

atomgroup

A `AtomGroup` for which the calculations are performed.

Type
MDAnalysis.core.groups.AtomGroup

_universe

The Universe the AtomGroup belong to

Type
MDAnalysis.core.universe.Universe

_trajectory

The trajectory the AtomGroup belong to

Type
MDAnalysis.coordinates.base.ReaderBase

times

array of Timestep times. Only exists after calling *AnalysisBase.run()*

Type
numpy.ndarray

frames

array of Timestep frame indices. Only exists after calling *AnalysisBase.run()*

Type
numpy.ndarray

_frame_index

index of the frame currently analysed

Type
int

_index

Number of frames already analysed (same as _frame_index + 1)

Type
int

results

results of calculation are stored after call to *AnalysisBase.run()*

Type
MDAnalysis.analysis.base.Results

_obs

Observables of the current frame

Type
MDAnalysis.analysis.base.Results

_obs.box_center

Center of the simulation cell of the current frame

Type
numpy.ndarray

sums

Sum of the observables across frames. Keys are the same as *_obs*.

Type
MDAnalysis.analysis.base.Results

means

Means of the observables. Keys are the same as *_obs*.

Type
MDAnalysis.analysis.base.Results

sems

Standard errors of the mean of the observables. Keys are the same as *_obs*

Type
MDAnalysis.analysis.base.Results

corrtime

The correlation time of the analysed data. For details on how this is calculated see *maicos.lib.util.correlation_analysis()*.

Type
float

Raises
ValueError – If any of the provided AtomGroups (*atomgroup* or *refgroup*) does not contain any atoms.

Example

To write your own analysis module you can use the example given below. As with all MAICoS modules, this inherits from the *maicos.core.base.AnalysisBase* class.

The example will calculate the average box volume and stores the result within the `result` object of the class.

```
>>> import logging
>>> from typing import Optional
```

```
>>> import MDAnalysis as mda
>>> import numpy as np
```

```
>>> from maicos.core import AnalysisBase
>>> from maicos.lib.util import render_docs
```

Creating a logger makes debugging easier.

```
>>> logger = logging.getLogger(__name__)
```

In the following the analysis module itself. Due to the similar structure of all MAICoS modules you can render the parameters using the *maicos.lib.util.render_docs()* decorator. The decorator will replace special keywords with a leading $ with the actual docstring as defined in *maicos.lib.util.DOC_DICT*.

```
>>> @render_docs
... class NewAnalysis(AnalysisBase):
...     "Analysis class calcuting the average box volume."
...
...     def __init__(
...         self,
...         atomgroup: mda.AtomGroup,
...         concfreq: int = 0,
...         temperature: float = 300,
...         output: str = "outfile.dat",
...     ):
...         super().__init__(
...             atomgroup=atomgroup,
...             refgroup=None,
...             unwrap=False,
...             jitter=0.0,
...             wrap_compound="atoms",
...             concfreq=concfreq,
...         )
...
...         self.temperature = temperature
...         self.output = output
...
...     def _prepare(self):
...         "Set things up before the analysis loop begins."
...         # self.atomgroup refers to the provided `atomgroup`
...         # self._universe refers to full universe of given `atomgroup`
...         self.volume = 0
...
...     def _single_frame(self):
```

(continues on next page)

(continued from previous page)

```
...         "Calculate data from a single frame of trajectory.
...
...         Don't worry about normalising, just deal with a single frame.
...         """
...         # Current frame index: self._frame_index
...         # Current timestep object: self._ts
...
...         volume = self._ts.volume
...         self.volume += volume
...
...         # Eeach module should return a characteristic scalar which is used
...         # by MAICoS to estimate correlations of an Analysis.
...         return volume
...
...     def _conclude(self):
...         "Finalise the results you've gathered.
...
...         Called at the end of the run() method to finish everything up.
...         """
...         self.results.volume = self.volume / self.n_frames
...         logger.info(
...             "Average volume of the simulation box "
...             f"{self.results.volume:.2f} Å$^3$"
...         )
...
...     def save(self) -> None:
...         "Save results of analysis to file specified by ``output``.
...
...         Called at the end of the run() method after _conclude.
...         """
...         self.savetxt(
...             self.output, np.array([self.results.volume]), columns="volume / Å$^3$"
...         )
...
```

Afterwards the new analysis can be run like this

```
>>> import MDAnalysis as mda
>>> from MDAnalysisTests.datafiles import TPR, XTC
```

```
>>> u = mda.Universe(TPR, XTC)
```

```
>>> na = NewAnalysis(u.atoms)
>>> _ = na.run(start=0, stop=10)
>>> round(na.results.volume, 2)
362631.65
```

Results can also be accessed by key

```
>>> round(na.results["volume"], 2)
362631.65
```

property box_center: ndarray

Center of the simulation cell.

run(*start: int | None = None*, *stop: int | None = None*, *step: int | None = None*, *frames: int | None = None*, *verbose: bool | None = None*, *progressbar_kwargs: dict | None = None*) → Self

Iterate over the trajectory.

savetxt(*fname: str*, *X: ndarray*, *columns: List[str] | None = None*) → None

Save to text.

An extension of the numpy savetxt function. Adds the command line input to the header and checks for a doubled defined filesuffix.

Return a header for the text file to save the data to. This method builds a generic header that can be used by any MAICoS module. It is called by the save method of each module.

The information it collects is:

- timestamp of the analysis
- name of the module
- version of MAICoS that was used
- command line arguments that were used to run the module
- module call including the default arguments
- number of frames that were analyzed
- atomgroup that was analyzed
- output messages from modules and base classes (if they exist)

AnalysisCollection

class maicos.core.**AnalysisCollection**(**analysis_instances: AnalysisBase*)

Bases: `_Runner`

Running a collection of analysis classes on the same single trajectory.

> **Warning:** `AnalysisCollection` is still experimental. You should not use it for anything important.

An analyses with `AnalysisCollection` can lead to a speedup compared to running the individual analyses, since the trajectory loop is performed only once. The class requires that each analysis is a child of *`AnalysisBase`*. Additionally, the trajectory of all `analysis_instances` must be the same. It is ensured that all analysis instances use the *same original* timestep and not an altered one from a previous analysis instance.

Parameters

***analysis_instances** (`AnalysisBase`) – Arbitrary number of analysis instances to be run on the same trajectory.

Raises

- **`AttributeError`** – If the provided `analysis_instances` do not work on the same trajectory.
- **`AttributeError`** – If an `analysis_instances` is not a child of *`AnalysisBase`*.

Example

```
>>> import MDAnalysis as mda
>>> from maicos import DensityPlanar
>>> from maicos.core import AnalysisCollection
>>> from MDAnalysisTests.datafiles import TPR, XTC
>>> u = mda.Universe(TPR, XTC)
```

Select atoms

```
>>> ag_O = u.select_atoms("name O")
>>> ag_H = u.select_atoms("name H")
```

Create the individual analysis instances

```
>>> dplan_O = DensityPlanar(ag_O)
>>> dplan_H = DensityPlanar(ag_H)
```

Create a collection for common trajectory

```
>>> collection = AnalysisCollection(dplan_O, dplan_H)
```

Run the collected analysis

```
>>> _ = collection.run(start=0, stop=100, step=10)
```

Results are stored in the individual instances see *AnalysisBase* on how to access them. You can also save all results of the analysis within one call:

```
>>> collection.save()
```

run(*start: int | None = None*, *stop: int | None = None*, *step: int | None = None*, *frames: int | None = None*, *verbose: bool | None = None*, *progressbar_kwargs: dict | None = None*) → Self

Iterate over the trajectory.

Parameters

- **start** (*int*) – start frame of analysis
- **stop** (*int*) – stop frame of analysis
- **step** (*int*) – number of frames to skip between each analysed frame
- **frames** (*array_like*) – array of integers or booleans to slice trajectory; `frames` can only be used *instead* of `start`, `stop`, and `step`. Setting *both* `frames` and at least one of `start`, `stop`, `step` to a non-default value will raise a `ValueError`.
- **verbose** (*bool*) – Turn on verbosity
- **progressbar_kwargs** (*dict*) – ProgressBar keywords with custom parameters regarding progress bar position, etc; see `MDAnalysis.lib.log.ProgressBar` for full list.

Returns

self – analysis object

Return type

object

save() → None

Save results of all analysis_instances to disk.

The methods calls the *save()* method of all analysis_instances if available. If an instance has no *save()* method a warning for this instance is issued.

ProfileBase

class maicos.core.**ProfileBase**(*atomgroup: AtomGroup*, *grouping: str*, *bin_method: str*, *output: str*, *weighting_function: Callable*, *weighting_function_kwargs: None | Dict*, *normalization: str*)

Bases: object

Base class for computing profiles.

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A AtomGroup for which the calculations are performed.
- **grouping** ({"atoms", "residues", "segments", "molecules", "fragments"}) – Atom grouping for the calculations.

  The possible grouping options are the atom positions (in the case where grouping="atoms") or the center of mass of the specified grouping unit (in the case where grouping="residues", "segments", "molecules" or "fragments").
- **bin_method** ({"com", "cog", "coc"}) – Method for the position binning.

  The possible options are center of mass ("com"), center of geometry ("cog"), and center of charge ("coc").
- **output** (*str*) – Output filename.
- **weighting_function** (*callable*) – The function calculating the array weights for the histogram analysis. It must take an AtomGroup as first argument and a grouping ("atoms", "residues", "segments", "molecules", "fragments") as second. Additional parameters can be given as weighting_function_kwargs. The function must return a numpy.ndarray with the same length as the number of group members.
- **weighting_function_kwargs** (*dict*) – Additional keyword arguments for weighting_function
- **normalization** ({"none", "number", "volume"}) – The normalization of the profile performed in every frame. If None, no normalization is performed. If *number*, the histogram is divided by the number of occurences in each bin. If *volume*, the profile is divided by the volume of each bin.

results.**profile**

Calculated profile.

Type
numpy.ndarray

results.**dprofile**

Estimated profile's uncertainity.

Type
numpy.ndarray

save() → None

Save results of analysis to file specified by `output`.

Planar classes

PlanarBase

class maicos.core.**PlanarBase**(*atomgroup: AtomGroup*, *unwrap: bool*, *refgroup: AtomGroup | None*, *jitter: float*, *concfreq: int*, *dim: int*, *zmin: None | float*, *zmax: None | float*, *bin_width: float*, *wrap_compound: str*)

Bases: *AnalysisBase*

Analysis class providing options and attributes for a planar system.

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A `AtomGroup` for which the calculations are performed.
- **unwrap** (*bool*) – When `True`, molecules that are broken due to the periodic boundary conditions are made whole.

  If the input contains molecules that are already whole, speed up the calculation by disabling unwrap. To do so, use the flag `-no-unwrap` when using MAICoS from the command line, or use `unwrap=False` when using MAICoS from the Python interpreter.

  Note: Molecules containing virtual sites (e.g. TIP4P water models) are not currently supported in MDAnalysis. In this case, you need to provide unwrapped trajectory files directly, and disable unwrap. Trajectories can be unwrapped, for example, using the `trjconv` command of GROMACS.
- **refgroup** (*MDAnalysis.core.groups.AtomGroup*) – Reference `AtomGroup` used for the calculation. If `refgroup` is provided, the calculation is performed relative to the center of mass of the AtomGroup. If `refgroup` is `None` the calculations are performed with respect to the center of the (changing) box.
- **jitter** (*float*) – Magnitude of the random noise to add to the atomic positions.

  A jitter can be used to stabilize the aliasing effects sometimes appearing when histogramming data. The jitter value should be about the precision of the trajectory. In that case, using jitter will not alter the results of the histogram. If `jitter = 0.0` (default), the original atomic positions are kept unchanged.

  You can estimate the precision of the positions in your trajectory with *maicos.lib.util.trajectory_precision()*. Note that if the precision is not the same for all frames, the smallest precision should be used.
- **concfreq** (*int*) – When concfreq (for conclude frequency) is larger than `0`, the conclude function is called and the output files are written every `concfreq` frames.
- **dim** (*{0, 1, 2}*) – Dimension for binning (`x=0`, `y=1`, `z=1`).
- **zmin** (*float*) – Minimal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

  If `zmin=None`, all coordinates down to the lower cell boundary are taken into account.
- **zmax** (*float*) – Maximal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

If `zmax = None`, all coordinates up to the upper cell boundary are taken into account.

- **bin_width** (*float*) – Width of the bins (in Å).
- **wrap_compound** (*str*) – The group which will be kept together through the wrap processes. Allowed values are: `"atoms"`, `"group"`, `"residues"`, `"segments"`, `"molecules"`, or `"fragments"`.

`results.`**bin_pos**

Bin positions (in Å) ranging from `zmin` to `zmax`.

Type
numpy.ndarray

zmin

Minimal coordinate for evaluation (Å) with in the lab frame, where 0 corresponds to the origin of the cell.

Type
float

zmax

Maximal coordinate for evaluation (Å) with in the lab frame, where 0 corresponds to the origin of the cell.

Type
float

`_obs.`**L**

Average length (in Å) along the chosen dimension in the current frame.

Type
float

`_obs.`**bin_pos**

Central bin positions (in Å) of each bin (in Å) in the current frame.

Type
numpy.ndarray, (n_bins)

`_obs.`**bin_width**

Bin width (in Å) in the current frame

Type
float

`_obs.`**bin_edges**

Edges of the bins (in Å) in the current frame.

Type
numpy.ndarray, (n_bins + 1)

`_obs.`**bin_area**

Area of the rectangle of each bin in the current frame. Calculated via $L_x \cdot L_y / N_{\mathrm{bins}}$ where L_x and L_y are the box lengths perpendicular to the dimension of evaluations given by *dim*. N_{bins} is the number of bins.

Type
numpy.ndarray, (n_bins)

`results.`**bin_volume**

Volume of an cuboid of each bin (in Å^3) in the current frame.

Type
numpy.ndarray, (n_bins)

property odims: ndarray

Other dimensions perpendicular to dim i.e. (0,2) if dim = 1.

ProfilePlanarBase

class maicos.core.**ProfilePlanarBase**(*atomgroup: AtomGroup*, *unwrap: bool*, *refgroup: AtomGroup | None*, *jitter: float*, *concfreq: int*, *dim: int*, *zmin: None | float*, *zmax: None | float*, *bin_width: float*, *sym: bool*, *grouping: str*, *bin_method: str*, *output: str*, *weighting_function: Callable*, *weighting_function_kwargs: None | Dict*, *normalization: str*)

Bases: *PlanarBase*, *ProfileBase*

Base class for computing profiles in a cartesian geometry.

For the correlation analysis the 0th bin of the 0th's group profile is used. For further information on the correlation analysis please refer to *maicos.core.base.AnalysisBase* or the *General design* section.

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A AtomGroup for which the calculations are performed.
- **unwrap** (*bool*) – When True, molecules that are broken due to the periodic boundary conditions are made whole.

  If the input contains molecules that are already whole, speed up the calculation by disabling unwrap. To do so, use the flag `-no-unwrap` when using MAICoS from the command line, or use `unwrap=False` when using MAICoS from the Python interpreter.

  Note: Molecules containing virtual sites (e.g. TIP4P water models) are not currently supported in MDAnalysis. In this case, you need to provide unwrapped trajectory files directly, and disable unwrap. Trajectories can be unwrapped, for example, using the `trjconv` command of GROMACS.
- **refgroup** (*MDAnalysis.core.groups.AtomGroup*) – Reference AtomGroup used for the calculation. If `refgroup` is provided, the calculation is performed relative to the center of mass of the AtomGroup. If `refgroup` is None the calculations are performed with respect to the center of the (changing) box.
- **jitter** (*float*) – Magnitude of the random noise to add to the atomic positions.

  A jitter can be used to stabilize the aliasing effects sometimes appearing when histogramming data. The jitter value should be about the precision of the trajectory. In that case, using jitter will not alter the results of the histogram. If `jitter = 0.0` (default), the original atomic positions are kept unchanged.

  You can estimate the precision of the positions in your trajectory with *maicos.lib.util.trajectory_precision()*. Note that if the precision is not the same for all frames, the smallest precision should be used.
- **concfreq** (*int*) – When concfreq (for conclude frequency) is larger than `0`, the conclude function is called and the output files are written every `concfreq` frames.
- **dim** (*{0, 1, 2}*) – Dimension for binning (`x=0`, `y=1`, `z=1`).
- **zmin** (*float*) – Minimal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

If zmin=None, all coordinates down to the lower cell boundary are taken into account.

- **zmax** (*float*) – Maximal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

  If zmax = None, all coordinates up to the upper cell boundary are taken into account.

- **bin_width** (*float*) – Width of the bins (in Å).
- **sym** (*bool*) – Symmetrize the profile. Only works in combination with refgroup.
- **grouping** ({"atoms", "residues", "segments", "molecules", "fragments"}) – Atom grouping for the calculations.

  The possible grouping options are the atom positions (in the case where grouping="atoms") or the center of mass of the specified grouping unit (in the case where grouping="residues", "segments", "molecules" or "fragments").

- **bin_method** ({"com", "cog", "coc"}) – Method for the position binning.

  The possible options are center of mass ("com"), center of geometry ("cog"), and center of charge ("coc").

- **output** (*str*) – Output filename.
- **weighting_function** (*callable*) – The function calculating the array weights for the histogram analysis. It must take an AtomGroup as first argument and a grouping ("atoms", "residues", "segments", "molecules", "fragments") as second. Additional parameters can be given as weighting_function_kwargs. The function must return a numpy.ndarray with the same length as the number of group members.
- **weighting_function_kwargs** (*dict*) – Additional keyword arguments for weighting_function
- **normalization** ({"none", "number", "volume"}) – The normalization of the profile performed in every frame. If None, no normalization is performed. If *number*, the histogram is divided by the number of occurences in each bin. If *volume*, the profile is divided by the volume of each bin.

results.**bin_pos**

Bin positions (in Å) ranging from zmin to zmax.

Type
numpy.ndarray

results.**profile**

Calculated profile.

Type
numpy.ndarray

results.**dprofile**

Estimated profile's uncertainity.

Type
numpy.ndarray

Cylinder classes

CylinderBase

class maicos.core.**CylinderBase**(*atomgroup: AtomGroup*, *unwrap: bool*, *refgroup: AtomGroup | None*, *jitter: float*, *concfreq: int*, *dim: int*, *zmin: None | float*, *zmax: None | float*, *bin_width: float*, *rmin: float*, *rmax: None | float*, *wrap_compound: str*)

Bases: *PlanarBase*

Analysis class providing options and attributes for a cylinder system.

Provide the results attribute *r*.

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A AtomGroup for which the calculations are performed.
- **unwrap** (*bool*) – When True, molecules that are broken due to the periodic boundary conditions are made whole.

  If the input contains molecules that are already whole, speed up the calculation by disabling unwrap. To do so, use the flag `-no-unwrap` when using MAICoS from the command line, or use `unwrap=False` when using MAICoS from the Python interpreter.

  Note: Molecules containing virtual sites (e.g. TIP4P water models) are not currently supported in MDAnalysis. In this case, you need to provide unwrapped trajectory files directly, and disable unwrap. Trajectories can be unwrapped, for example, using the `trjconv` command of GROMACS.
- **refgroup** (*MDAnalysis.core.groups.AtomGroup*) – Reference AtomGroup used for the calculation. If `refgroup` is provided, the calculation is performed relative to the center of mass of the AtomGroup. If `refgroup` is None the calculations are performed with respect to the center of the (changing) box.
- **jitter** (*float*) – Magnitude of the random noise to add to the atomic positions.

  A jitter can be used to stabilize the aliasing effects sometimes appearing when histogramming data. The jitter value should be about the precision of the trajectory. In that case, using jitter will not alter the results of the histogram. If `jitter = 0.0` (default), the original atomic positions are kept unchanged.

  You can estimate the precision of the positions in your trajectory with *maicos.lib.util.trajectory_precision()*. Note that if the precision is not the same for all frames, the smallest precision should be used.
- **concfreq** (*int*) – When concfreq (for conclude frequency) is larger than `0`, the conclude function is called and the output files are written every `concfreq` frames.
- **dim** (*{0, 1, 2}*) – Dimension for binning (`x=0`, `y=1`, `z=1`).
- **zmin** (*float*) – Minimal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

  If `zmin=None`, all coordinates down to the lower cell boundary are taken into account.
- **zmax** (*float*) – Maximal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

  If `zmax = None`, all coordinates up to the upper cell boundary are taken into account.
- **bin_width** (*float*) – Width of the bins (in Å).

- **rmin** (*float*) – Minimal radial coordinate relative to the center of mass of the refgroup for evaluation (in Å).
- **rmax** (*float*) – Maximal radial coordinate relative to the center of mass of the refgroup for evaluation (in Å).

  If `rmax=None`, the box extension is taken.
- **wrap_compound** (*str*) – The group which will be kept together through the wrap processes. Allowed values are: `"atoms"`, `"group"`, `"residues"`, `"segments"`, `"molecules"`, or `"fragments"`.

`results.bin_pos`

Bin positions (in Å) ranging from `rmin` to `rmax`.

Type
numpy.ndarray

`pos_cyl`

positions in cylinder coordinats (r, phi, z)

Type
numpy.ndarray

`_obs.R`

Average length (in Å) along the radial dimension in the current frame.

Type
float

`_obs.bin_pos`

Central bin position of each bin (in Å) in the current frame.

Type
numpy.ndarray, (n_bins)

`_obs.bin_width`

Bin width (in Å) in the current frame

Type
float

`_obs.bin_edges`

Edges of the bins (in Å) in the current frame.

Type
numpy.ndarray, (n_bins + 1)

`_obs.bin_area`

Area of the annulus pf the each bin in the current frame. Calculated via $\pi \left(r_{i+1}^2 - r_i^2\right)$ where *i* is the index of the bin.

Type
numpy.ndarray, (n_bins)

`_obs.bin_volume`

Volume of an hollow cylinder of each bin (in Å^3) in the current frame. Calculated via $\pi L \left(r_{i+1}^2 - r_i^2\right)$ where *i* is the index of the bin.

Type
numpy.ndarray, (n_bins)

ProfileCylinderBase

class maicos.core.**ProfileCylinderBase**(*atomgroup: AtomGroup*, *unwrap: bool*, *refgroup: AtomGroup | None*, *jitter: float*, *concfreq: int*, *dim: int*, *zmin: None | float*, *zmax: None | float*, *bin_width: float*, *rmin: float*, *rmax: None | float*, *grouping: str*, *bin_method: str*, *output: str*, *weighting_function: Callable*, *weighting_function_kwargs: None | Dict*, *normalization: str*)

Bases: *CylinderBase*, *ProfileBase*

Base class for computing radial profiles in a cylindrical geometry.

For the correlation analysis the 0th bin of the 0th's group profile is used. For further information on the correlation analysis please refer to *maicos.core.base.AnalysisBase* or the *General design* section.

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A `AtomGroup` for which the calculations are performed.
- **unwrap** (*bool*) – When `True`, molecules that are broken due to the periodic boundary conditions are made whole.

  If the input contains molecules that are already whole, speed up the calculation by disabling unwrap. To do so, use the flag `-no-unwrap` when using MAICoS from the command line, or use `unwrap=False` when using MAICoS from the Python interpreter.

  Note: Molecules containing virtual sites (e.g. TIP4P water models) are not currently supported in MDAnalysis. In this case, you need to provide unwrapped trajectory files directly, and disable unwrap. Trajectories can be unwrapped, for example, using the `trjconv` command of GROMACS.
- **refgroup** (*MDAnalysis.core.groups.AtomGroup*) – Reference `AtomGroup` used for the calculation. If `refgroup` is provided, the calculation is performed relative to the center of mass of the AtomGroup. If `refgroup` is `None` the calculations are performed with respect to the center of the (changing) box.
- **jitter** (*float*) – Magnitude of the random noise to add to the atomic positions.

  A jitter can be used to stabilize the aliasing effects sometimes appearing when histogramming data. The jitter value should be about the precision of the trajectory. In that case, using jitter will not alter the results of the histogram. If `jitter = 0.0` (default), the original atomic positions are kept unchanged.

  You can estimate the precision of the positions in your trajectory with *maicos.lib.util.trajectory_precision()*. Note that if the precision is not the same for all frames, the smallest precision should be used.
- **concfreq** (*int*) – When concfreq (for conclude frequency) is larger than `0`, the conclude function is called and the output files are written every `concfreq` frames.
- **dim** (*{0, 1, 2}*) – Dimension for binning (`x=0`, `y=1`, `z=1`).
- **zmin** (*float*) – Minimal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

  If `zmin=None`, all coordinates down to the lower cell boundary are taken into account.
- **zmax** (*float*) – Maximal coordinate for evaluation (in Å) with respect to the center of mass of the refgroup.

  If `zmax = None`, all coordinates up to the upper cell boundary are taken into account.

- **bin_width** (*float*) – Width of the bins (in Å).
- **rmin** (*float*) – Minimal radial coordinate relative to the center of mass of the refgroup for evaluation (in Å).
- **rmax** (*float*) – Maximal radial coordinate relative to the center of mass of the refgroup for evaluation (in Å).

  If `rmax=None`, the box extension is taken.
- **grouping** ({"atoms", "residues", "segments", "molecules", "fragments"}) – Atom grouping for the calculations.

  The possible grouping options are the atom positions (in the case where `grouping="atoms"`) or the center of mass of the specified grouping unit (in the case where `grouping="residues"`, `"segments"`, `"molecules"` or `"fragments"`).
- **bin_method** ({"com", "cog", "coc"}) – Method for the position binning.

  The possible options are center of mass (`"com"`), center of geometry (`"cog"`), and center of charge (`"coc"`).
- **output** (*str*) – Output filename.
- **weighting_function** (*callable*) – The function calculating the array weights for the histogram analysis. It must take an `AtomGroup` as first argument and a grouping (`"atoms"`, `"residues"`, `"segments"`, `"molecules"`, `"fragments"`) as second. Additional parameters can be given as `weighting_function_kwargs`. The function must return a numpy.ndarray with the same length as the number of group members.
- **weighting_function_kwargs** (*dict*) – Additional keyword arguments for `weighting_function`
- **normalization** ({"none", "number", "volume"}) – The normalization of the profile performed in every frame. If `None`, no normalization is performed. If *number*, the histogram is divided by the number of occurences in each bin. If *volume*, the profile is divided by the volume of each bin.

`results.`**bin_pos**

Bin positions (in Å) ranging from `rmin` to `rmax`.

Type
numpy.ndarray

`results.`**profile**

Calculated profile.

Type
numpy.ndarray

`results.`**dprofile**

Estimated profile's uncertainity.

Type
numpy.ndarray

Sphere classes

SphereBase

class maicos.core.**SphereBase**(*atomgroup: AtomGroup*, *unwrap: bool*, *refgroup: AtomGroup | None*, *jitter: float*, *concfreq: int*, *rmin: float*, *rmax: None | float*, *bin_width: float*, *wrap_compound: str*)

Bases: *AnalysisBase*

Analysis class providing options and attributes for a spherical system.

Provide the results attribute *r*.

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A `AtomGroup` for which the calculations are performed.
- **unwrap** (*bool*) – When `True`, molecules that are broken due to the periodic boundary conditions are made whole.

  If the input contains molecules that are already whole, speed up the calculation by disabling unwrap. To do so, use the flag `-no-unwrap` when using MAICoS from the command line, or use `unwrap=False` when using MAICoS from the Python interpreter.

  Note: Molecules containing virtual sites (e.g. TIP4P water models) are not currently supported in MDAnalysis. In this case, you need to provide unwrapped trajectory files directly, and disable unwrap. Trajectories can be unwrapped, for example, using the `trjconv` command of GROMACS.
- **refgroup** (*MDAnalysis.core.groups.AtomGroup*) – Reference `AtomGroup` used for the calculation. If `refgroup` is provided, the calculation is performed relative to the center of mass of the AtomGroup. If `refgroup` is `None` the calculations are performed with respect to the center of the (changing) box.
- **jitter** (*float*) – Magnitude of the random noise to add to the atomic positions.

  A jitter can be used to stabilize the aliasing effects sometimes appearing when histogramming data. The jitter value should be about the precision of the trajectory. In that case, using jitter will not alter the results of the histogram. If `jitter = 0.0` (default), the original atomic positions are kept unchanged.

  You can estimate the precision of the positions in your trajectory with *maicos.lib.util.trajectory_precision()*. Note that if the precision is not the same for all frames, the smallest precision should be used.
- **concfreq** (*int*) – When concfreq (for conclude frequency) is larger than `0`, the conclude function is called and the output files are written every `concfreq` frames.
- **rmin** (*float*) – Minimal radial coordinate relative to the center of mass of the refgroup for evaluation (in Å).
- **rmax** (*float*) – Maximal radial coordinate relative to the center of mass of the refgroup for evaluation (in Å).

  If `rmax=None`, the box extension is taken.
- **bin_width** (*float*) – Width of the bins (in Å).

- **wrap_compound** (*str*) – The group which will be kept together through the wrap processes. Allowed values are: `"atoms"`, `"group"`, `"residues"`, `"segments"`, `"molecules"`, or `"fragments"`.

`results.`**bin_pos**

Bin positions (in Å) ranging from `rmin` to `rmax`.

Type
numpy.ndarray

pos_sph

positions in spherical coordinats (r, phi, theta)

Type
numpy.ndarray

`_obs.`**R**

Average length (in Å) along the radial dimension in the current frame.

Type
float

`_obs.`**bin_pos**

Central bin position of each bin (in Å) in the current frame.

Type
numpy.ndarray, (n_bins)

`_obs.`**bin_width**

Bin width (in Å) in the current frame

Type
float

`_obs.`**bin_edges**

Edges of the bins (in Å) in the current frame.

Type
numpy.ndarray, (n_bins + 1)

`_obs.`**bin_area**

Surface area (in Å^2) of the sphere of each bin with radius *bin_pos* in the current frame. Calculated via $4\pi r_i^2$ where i is the index of the bin.

Type
numpy.ndarray, (n_bins)

`results.`**bin_volume**

volume of a spherical shell of each bins (in Å^3) of the current frame. Calculated via $4\pi/3\left(r_{i+1}^3 - r_i^3\right)$ where *i* is the index of the bin.

Type
numpy.ndarray, (n_bins)

ProfileSphereBase

class maicos.core.**ProfileSphereBase**(*atomgroup: AtomGroup*, *unwrap: bool*, *refgroup: AtomGroup | None*, *jitter: float*, *concfreq: int*, *rmin: float*, *rmax: None | float*, *bin_width: float*, *grouping: str*, *bin_method: str*, *output: str*, *weighting_function: Callable*, *weighting_function_kwargs: Dict | None*, *normalization: str*)

Bases: *SphereBase*, *ProfileBase*

Base class for computing radial profiles in a spherical geometry.

For the correlation analysis the 0th bin of the 0th's group profile is used. For further information on the correlation analysis please refer to *maicos.core.base.AnalysisBase* or the *General design* section.

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A `AtomGroup` for which the calculations are performed.
- **unwrap** (*bool*) – When `True`, molecules that are broken due to the periodic boundary conditions are made whole.

  If the input contains molecules that are already whole, speed up the calculation by disabling unwrap. To do so, use the flag `-no-unwrap` when using MAICoS from the command line, or use `unwrap=False` when using MAICoS from the Python interpreter.

  Note: Molecules containing virtual sites (e.g. TIP4P water models) are not currently supported in MDAnalysis. In this case, you need to provide unwrapped trajectory files directly, and disable unwrap. Trajectories can be unwrapped, for example, using the `trjconv` command of GROMACS.
- **refgroup** (*MDAnalysis.core.groups.AtomGroup*) – Reference `AtomGroup` used for the calculation. If `refgroup` is provided, the calculation is performed relative to the center of mass of the AtomGroup. If `refgroup` is `None` the calculations are performed with respect to the center of the (changing) box.
- **jitter** (*float*) – Magnitude of the random noise to add to the atomic positions.

  A jitter can be used to stabilize the aliasing effects sometimes appearing when histogramming data. The jitter value should be about the precision of the trajectory. In that case, using jitter will not alter the results of the histogram. If `jitter = 0.0` (default), the original atomic positions are kept unchanged.

  You can estimate the precision of the positions in your trajectory with *maicos.lib.util.trajectory_precision()*. Note that if the precision is not the same for all frames, the smallest precision should be used.
- **concfreq** (*int*) – When concfreq (for conclude frequency) is larger than `0`, the conclude function is called and the output files are written every `concfreq` frames.
- **rmin** (*float*) – Minimal radial coordinate relative to the center of mass of the refgroup for evaluation (in Å).
- **rmax** (*float*) – Maximal radial coordinate relative to the center of mass of the refgroup for evaluation (in Å).

  If `rmax=None`, the box extension is taken.
- **bin_width** (*float*) – Width of the bins (in Å).
- **grouping** ({`"atoms"`, `"residues"`, `"segments"`, `"molecules"`, `"fragments"`}) – Atom grouping for the calculations.

The possible grouping options are the atom positions (in the case where `grouping="atoms"`) or the center of mass of the specified grouping unit (in the case where `grouping="residues"`, `"segments"`, `"molecules"` or `"fragments"`).

- **bin_method** ({`"com"`, `"cog"`, `"coc"`}) – Method for the position binning.

  The possible options are center of mass (`"com"`), center of geometry (`"cog"`), and center of charge (`"coc"`).

- **output** (*str*) – Output filename.
- **weighting_function** (*callable*) – The function calculating the array weights for the histogram analysis. It must take an `AtomGroup` as first argument and a grouping (`"atoms"`, `"residues"`, `"segments"`, `"molecules"`, `"fragments"`) as second. Additional parameters can be given as `weighting_function_kwargs`. The function must return a numpy.ndarray with the same length as the number of group members.
- **weighting_function_kwargs** (*dict*) – Additional keyword arguments for `weighting_function`
- **normalization** ({`"none"`, `"number"`, `"volume"`}) – The normalization of the profile performed in every frame. If `None`, no normalization is performed. If *number*, the histogram is divided by the number of occurences in each bin. If *volume*, the profile is divided by the volume of each bin.

`results.`**bin_pos**

Bin positions (in Å) ranging from `rmin` to `rmax`.

Type
numpy.ndarray

`results.`**profile**

Calculated profile.

Type
numpy.ndarray

`results.`**dprofile**

Estimated profile's uncertainity.

Type
numpy.ndarray

Library functions

Library modules of MAICoS.

This library contains additional modules, like general and mathematical helper functions, which are used in the other MAICoS modules.

Mathematical helper functions

Helper functions for mathematical and physical operations.

maicos.lib.math.**FT**(*t: ndarray*, *x: ndarray*, *indvar: bool = True*) → ndarray | Tuple[ndarray, ndarray]

Discrete Fourier transformation using fast Fourier transformation (FFT).

Parameters

- **t** (*numpy.ndarray*) – Time values of the time series.
- **x** (*numpy.ndarray*) – Function values corresponding to the time series.
- **indvar** (*bool*) – If `True`, returns the FFT and frequency values. If `False`, returns only the FFT.

Returns

If `indvar` is `True`, returns a tuple `(k, xf2)` where:

- k (numpy.ndarray): Frequency values corresponding to the FFT.
- `xf2` (numpy.ndarray): FFT of the input function, scaled by the time range and phase shifted.

If indvar is `False`, returns the FFT (`xf2`) directly as a `numpy.ndarray`.

Return type
tuple(numpy.ndarray, numpy.ndarray) or numpy.ndarray

Raises
`RuntimeError` – If the time series is not equally spaced.

Example

```
>>> t = np.linspace(0, np.pi, 4)
>>> x = np.sin(t)
>>> k, xf2 = FT(t, x)
>>> k
array([-3. , -1.5,  0. ,  1.5])
>>> np.round(xf2, 2)
array([ 0.  +0.j  , -0.68+0.68j,  1.36+0.j  , -0.68-0.68j])
```

See also:

`iFT()`
For the inverse fourier transform.

maicos.lib.math.**center_cluster**(*ag: AtomGroup*, *weights: ndarray*) → ndarray

Calculate the center of the atomgroup with respect to some weights.

Parameters

- **ag** (*MDAnalysis.core.groups.AtomGroup*) – Group of atoms to calculate the center for.
- **weights** (*numpy.ndarray*) – Weights in the shape of ag.

Returns
com – The center with respect to the weights.

Return type
numpy.ndarray

Without proper treatment of periodic boundrary conditions (PBC) most algorithms will result in wrong center calculations. As shown below without treating PBC the center of mass is located in the center of the box

```
+-----------+
|           |
| 1   x   2 |
|           |
+-----------+
```

However, the distance accross the box boundary is shorter and therefore the center with PBC should be located somwhere else. The correct way to calculate the center is described in Bai and Breen[1] where coordinates of the particles are projected on a circle and weighted by their mass in this two dimensional space. The center of mass is obtained by transforming this point back to the corresponding point in the real system. This is done seperately for each dimension.

Reasons for doing this include the analysis of clusters in periodic boundrary conditions and consistent center of mass calculation across box boundraries. This procedure results in the right center of mass as seen below

```
+-----------+
|           |
x 1       2 |
|           |
+-----------+
```

maicos.lib.math.**compute_form_factor**(*q: float*, *atom_type: str*) → float

Calculate the form factor $f(q)$.

$f(q)$ is expressed in terms of the scattering vector as

$$f(q) = \sum_{i=1}^{4} a_i e^{-b_i q^2/(4\pi)^2} + c\,.$$

The coefficients $a_{1,\dots,4}$, $b_{1,\dots,4}$ and c are also known as Cromer-Mann X-ray scattering factors and are documented in Prince[2].

For determining the elements *maicos.lib.tables.atomtypes* is used and the Cromer-Mann X-ray scattering factors are stored in *maicos.lib.tables.CM_parameters*.

Parameters

- **q** (*float*) – The magnitude of the scattering vector in reciprocal angstroms (1/Å).
- **atom_type** (*str*) – The type of the atom for which the form factor is calculated.

Returns
The calculated form factor for the specified atom type and q.

Return type
float

[1] Linge Bai and David Breen. Calculating Center of Mass in an Unbounded 2D Environment. *Journal of Graphics Tools*, 13(4):53–60, January 2008. doi:10.1080/2151237X.2008.10129266.

[2] E. Prince. *International Tables for Crystallography,Volume C: Mathematical, Physical and Chemical Tables*. Springer, Dordrecht, 3rd ed. edition edition, January 2004. ISBN 978-1-4020-1900-5.

maicos.lib.math.**compute_rdf_structure_factor**(*rdf: ndarray*, *r: ndarray*, *density: float*) → Tuple[ndarray, ndarray]

Computes the structure factor based on the radial distribution function (RDF).

The structure factor $S(q)$ based on the RDF $g(r)$ is given by

$$S(q) = 1 + 4\pi\rho \int_0^\infty \mathrm{d}r r \frac{\sin(qr)}{q}(g(r) - 1)$$

where q is the magnitude of the scattering vector. The calculation is performed via a discrete sine transform as implemented in `scipy.fftpack.dst()`.

For an *example* take a look at *Small-angle X-ray scattering*.

Parameters

- **rdf** (*numpy.ndarray*) – radial distribution function
- **r** (*numpy.ndarray*) – equally spaced distance array on which rdf is defined
- **density** (*float*) – number density of particles

Returns

- **q** (*numpy.ndarray*) – array of q points
- **struct_factor** (*numpy.ndarray*) – structure factor

maicos.lib.math.**correlation**(*a: ndarray*, *b: ndarray | None = None*, *subtract_mean: bool = False*) → ndarray

Calculate correlation or autocorrelation.

Uses fast fourier transforms to give the correlation function of two arrays, or, if only one array is given, the autocorrelation. Setting `subtract_mean=True` causes the mean to be subtracted from the input data.

Parameters

- **a** (*numpy.ndarray*) – The first input array to calculate the correlation
- **b** (*numpy.ndarray*) – The second input array. If `None`, autocorrelation of `a` is calculated.
- **subtract_mean** (*bool*) – If `True`, subtract the mean from the input data.

Returns
The correlation or autocorrelation function.

Return type
numpy.ndarray

maicos.lib.math.**correlation_time**(*timeseries: ndarray*, *method: str = 'sokal'*, *mintime: int = 3*, *sokal_factor: float = 8*) → float

Compute the integrated correlation time of a time series.

The integrated correlation time (in units of the sampling interval) is given by

$$\tau = \sum_{t=1}^{N_{\mathrm{cut}}} C(t) \left(1 - \frac{t}{N}\right)$$

where $N_{\mathrm{cut}} < N$ is a subset of the time series of length N and $C(t)$ is the discrete-time autocorrelation function. To obtain the upper limit of the sum N_{cut} two different methods are provided:

1. For "chodera"[3] N_{cut} is given by the time when $C(t)$ crosses zero the first time.

[3] John D. Chodera, William C. Swope, Jed W. Pitera, Chaok Seok, and Ken A. Dill. Use of the Weighted Histogram Analysis Method for the Analysis of Simulated and Parallel Tempering Simulations. *J. Chem. Theory Comput.*, 2007. doi:10.1021/ct0502864.

2. For "sokal"[4] N_{cut} is determined iteratively by stepwise increasing until

$$N_{\mathrm{cut}} \geq c \cdot \tau$$

where c is the constant `sokal_factor`. If the condition is never fulfilled, `-1` is returned, indicating that the time series does not provide sufficient statistics to estimate a correlation time.

While both methods give the same correlation time for a smooth time series that decays to 0, "sokal" will results in a more reasonable result for actual time series that are noisy and cross zero several times.

Parameters

- **timeseries** (*numpy.ndarray*) – The time series used to calculate the correlation time from.
- **method** (*{"sokal", "chodera"}*) – Method to choose summation cutoff N_{cut}.
- **mintime** (*int*) – Minimum possible value for N_{cut}.
- **sokal_factor** (*float*) – Cut-off factor c for the Sokal method.

Returns

tau – Integrated correlation time τ. If `-1` (only for `method="sokal"`) the provided time series does not provide sufficient statistics to estimate a correlation time.

Return type

float

Raises

- **ValueError** – If mintime is larger than the length of the timeseries.
- **ValueError** – If method is not one of "sokal" or "chodera".

References

maicos.lib.math.**iFT**(*k: ndarray*, *xf: ndarray*, *indvar: bool = True*) → ndarray | Tuple[ndarray, ndarray]

Inverse Fourier transformation using fast Fourier transformation (FFT).

Takes the frequency series and the function as arguments. By default, returns the iFT and the time series. Setting indvar=False means the function returns only the iFT.

Parameters

- **k** (*numpy.ndarray*) – The frequency series.
- **xf** (*numpy.ndarray*) – The function series in the frequency domain.
- **indvar** (*bool*) – If `True`, return both the iFT and the time series. If `False`, return only the iFT.

Returns

If indvar is `True`, returns a tuple containing the time series and the iFT. If indvar is `False`, returns only the iFT.

Return type

tuple(numpy.ndarray, numpy.ndarray) or numpy.ndarray

Raises

RuntimeError – If the time series is not equally spaced.

See also:

[4] A.D. Sokal. Monte Carlo Methods in Statistical Mechanics: Foundations and New Algorithms, lecture notes.

FT()
For the Fourier transform.

maicos.lib.math.**new_mean**(*old_mean: float*, *data: float*, *length: int*) → float

Compute the arithmetic mean of a series iteratively.

Compute the arithmetic mean of n samples based on an existing mean of n-1 and the n-th value.

Given the mean of a data series

$$\bar{x}_N = \frac{1}{N}\sum_{n=1}^{N} x_n$$

we seperate the last value

$$\bar{x}_N = \frac{1}{N}\sum_{n=1}^{N-1} x_n + \frac{x_N}{N}$$

and multiply 1 = (N - 1)/(N - 1)

$$\bar{x}_N = \frac{N-1}{N}\frac{1}{N-1}\sum_{n=1}^{N-1} x_n + \frac{x_N}{N}$$

The first term can be identified as the mean of the first N - 1 values and we arrive at

$$\bar{x}_N = \frac{N-1}{N}\bar{x}_{N-1} + \frac{x_N}{N}$$

Parameters

- **old_mean** (*float*) – arithmetic mean of the first n - 1 samples.
- **data** (*float*) – n-th value of the series.
- **length** (*int*) – Length of the updated series, here called n.

Returns
new_mean – Updated mean of the series of n values.

Return type
float

Examples

The mean of a data set can easily be calculated from the data points. However this requires one to keep all data points on hand until the end of the calculation.

```
>>> np.mean([1, 3, 5, 7])
4.0
```

Alternatively, one can update an existing mean, this requires only knowledge of the total number of samples.

```
>>> new_mean(np.mean([1, 3, 5]), data=7, length=4)
4.0
```

maicos.lib.math.**new_variance**(*old_variance: float | ndarray*, *old_mean: float | ndarray*, *new_mean: float | ndarray*, *data: float | ndarray*, *length: int*) → float | ndarray

Calculate the variance of a timeseries iteratively.

The variance of a timeseries x_n can be calculated iteratively by using the following formula:

$$S_n = S_n - 1 + (n-1) * (x_n - \bar{x}_n - 1)^2/(n-1)$$

Here, $\bar{x}_n$ is the mean of the timeseries up to the n-th value.

Floating point imprecision can lead to slight negative variances leading non defined standard deviations. Therefore a negetaive variance is set to 0.

Parameters

- **old_variance** (*float, numpy.ndarray*) – The variance of the first n-1 samples.
- **old_mean** (*float*) – The mean of the first n-1 samples.
- **new_mean** (*float, numpy.ndarray*) – The mean of the full n samples.
- **data** (*float, numpy.ndarray*) – The n-th value of the series.
- **length** (*int*) – Length of the updated series, here called n.

Returns

new_variance – Updated variance of the series of n values.

Return type

float

Examples

The data set `[1, 5, 5, 1]` has a variance of `4.0`

```
>>> np.var([1, 5, 5, 1])
4.0
```

Knowing the total number of data points, this operation can be performed iteratively.

```
>>> new_variance(
...     old_variance=np.var([1, 5, 5]),
...     old_mean=np.mean([1, 5, 5]),
...     new_mean=np.mean([1, 5, 5, 1]),
...     data=1,
...     length=4,
... )
4.0
```

maicos.lib.math.**scalar_prod_corr**(*a: ndarray*, *b: ndarray | None = None*, *subtract_mean: bool = False*) → ndarray

Give the corr. function of the scalar product of two vector timeseries.

Arguments should be given in the form a[t, i], where t is the time variable along which the correlation is calculated, and i indexes the vector components.

Parameters

- **a** (*numpy.ndarray*) – The first vector timeseries of shape (t, i).

- **b** (*numpy.ndarray*) – The second vector timeseries of shape (t, i). If `None`, correlation with itself is calculated.
- **subtract_mean** (*bool*) – If `True`, subtract the mean from the timeseries before calculating the correlation.

Returns
The correlation function of the scalar product of the vector timeseries.

Return type
numpy.ndarray

maicos.lib.math.**symmetrize**(*m: ndarray*, *axis: None | int | Tuple[int] = None*, *inplace: bool = False*) → ndarray

Symmeterize an array.

The shape of the array is preserved, but the elements are symmetrized with respect to the given axis.

Parameters

- **m** (*array_like*) – Input array to symmetrize
- **axis** (*int, tuple(int)*) – Axis or axes along which to symmetrize over. The default, `axis=None`, will symmetrize over all of the axes of the input array. If axis is negative it counts from the last to the first axis. If axis is a `tuple` of ints, symmetrizing is performed on all of the axes specified in the `tuple`.
- **inplace** (*bool*) – Do symmetrizations inplace. If `False` a new array is returned.

Returns
out – the symmetrized array

Return type
array_like

Notes

symmetrize uses `np.flip()` for flipping the indices.

Examples

```
>>> A = np.arange(10).astype(float)
>>> A
array([0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])
>>> symmetrize(A)
array([4.5, 4.5, 4.5, 4.5, 4.5, 4.5, 4.5, 4.5, 4.5, 4.5])
>>> symmetrize(A, inplace=True)
array([4.5, 4.5, 4.5, 4.5, 4.5, 4.5, 4.5, 4.5, 4.5, 4.5])
>>> A
array([4.5, 4.5, 4.5, 4.5, 4.5, 4.5, 4.5, 4.5, 4.5, 4.5])
```

It also works for arrays with more than 1 dimensions in a general dimension.

```
>>> A = np.arange(20).astype(float).reshape(2, 10).T
>>> A
array([[ 0., 10.],
       [ 1., 11.],
```

(continues on next page)

(continued from previous page)

```
       [ 2., 12.],
       [ 3., 13.],
       [ 4., 14.],
       [ 5., 15.],
       [ 6., 16.],
       [ 7., 17.],
       [ 8., 18.],
       [ 9., 19.]])
>>> symmetrize(A)
array([[9.5, 9.5],
       [9.5, 9.5],
       [9.5, 9.5],
       [9.5, 9.5],
       [9.5, 9.5],
       [9.5, 9.5],
       [9.5, 9.5],
       [9.5, 9.5],
       [9.5, 9.5],
       [9.5, 9.5]])
>>> symmetrize(A, axis=0)
array([[ 4.5, 14.5],
       [ 4.5, 14.5],
       [ 4.5, 14.5],
       [ 4.5, 14.5],
       [ 4.5, 14.5],
       [ 4.5, 14.5],
       [ 4.5, 14.5],
       [ 4.5, 14.5],
       [ 4.5, 14.5],
       [ 4.5, 14.5]])
```

maicos.lib.math.**transform_cylinder**(*positions: ndarray*, *origin: ndarray*, *dim: int*) → ndarray

Transform positions into cylinder coordinates.

The origin of th coordinate system is at *origin*, the direction of the cylinder is defined by *dim*.

Parameters

- **positions** (*numpy.ndarray*) – Cartesian coordinates (x,y,z)
- **origin** (*numpy.ndarray*) – Origin of the new cylindrical coordinate system (x,y,z).
- **dim** (*int*) – Direction of the cylinder axis (0=x, 1=y, 2=z).

Returns

Positions in cylinder coordinates (r, phi, z)

Return type

numpy.ndarray

maicos.lib.math.**transform_sphere**(*positions: ndarray*, *origin: ndarray*) → ndarray

Transform positions into spherical coordinates.

The origin of the new coordinate system is at *origin*.

Parameters

- **positions** (*numpy.ndarray*) – Cartesian coordinates (x,y,z)

- **origin** (*numpy.ndarray*) – Origin of the new spherical coordinate system (x,y,z).

Returns
Positions in spherical coordinates (r, phi, theta)

Return type
numpy.ndarray

maicos.lib._cmath.**compute_structure_factor**(*positions*, *dimensions*, *qmin*, *qmax*, *thetamin*, *thetamax*, *weights*)

Calculates scattering vectors and corresponding structure factors.

Use via `from maicos.lib.math import compute_structure_factor`

The structure factors are calculated according to

$$S(\boldsymbol{q}) = \left[\sum_{k=1}^{N} w_j(q)\cos(\boldsymbol{q}\boldsymbol{r}_j)\right]^2 + \left[\sum_{k=1}^{N} w_j(q)\sin(\boldsymbol{q}\boldsymbol{r}_j)\right]^2 .$$

where $\boldsymbol{r}_j$ is the positions vector of particle k, $\boldsymbol{q}$ is scattering vector and the w_j are optional weights. The possible scattering vectors are determined by the given cell `dimensions`.

Results are returned as arrays with three dimensions, where the index of each dimensions referers to the Miller indices hkl. Based on the Miller indices and the returned length of the scattering vector the actual scattering vector can be obtained by

$$q_{hkl} = |\boldsymbol{q}|\frac{2\pi}{L_{hkl}}$$

where $|\boldsymbol{q}|$ are the returned lengths of the scattering vector and L_{hkl} are the components of the simulation cell.

Parameters

- **positions** (*numpy.ndarray*) – position array.
- **dimensions** (*numpy.ndarray*) – dimensions of the cell.
- **qmin** (*float*) – Starting scattering vector length (1/Å).
- **qmax** (*float*) – Ending scattering vector length (1/Å).
- **thetamin** (*float*) – Minimal angle (°) between the scattering vectors and the z-axis.
- **thetamax** (*float*) – Maximal angle (°) between the scattering vectors and the z-axis.
- **weights** (*numpy.ndarray*) – Atomic quantity whose $S(|q|)$ we are computing. Provide an array of `1` that has the same size as the postions, h.e `np.ones(len(positions))`, for the standard structure factor.

Returns
The length of the scattering vectors and the corresponding structure factors.

Return type
tuple(numpy.ndarray, numpy.ndarray)

General helper functions

Small helper and utilities functions that don't fit anywhere else.

maicos.lib.util.**DOC_DICT**

Dictionary containing the keys and the actual docstring used by *maicos.lib.util.render_docs()*.

maicos.lib.util.**DOI_LIST**

References associated with MAICoS

class maicos.lib.util.**Unit_vector**(**args*, ***kwargs*)

Bases: Protocol

Protocol class for unit vector methods type hints.

maicos.lib.util.**atomgroup_header**(*AtomGroup: AtomGroup*) → str

Return a string containing infos about the AtomGroup.

Infos include the total number of atoms, the including residues and the number of residues. Useful for writing output file headers.

Parameters
AtomGroup (*MDAnalysis.core.groups.AtomGroup*) – The AtomGroup object containing the atoms.

Returns
A string containing the AtomGroup information.

Return type
str

maicos.lib.util.**bin**(*a: ndarray*, *bins: ndarray*) → ndarray

Average array values in bins for easier plotting.

Parameters

- **a** (*numpy.ndarray*) – The input array to be averaged.
- **bins** (*numpy.ndarray*) – The array containing the indices where each bin begins.

Returns
The averaged array values.

Return type
numpy.ndarray

Notes

The "bins" array should contain the INDEX (integer) where each bin begins.

maicos.lib.util.**charge_neutral**(*filter: str*) → Callable

Raise a Warning when AtomGroup is not charge neutral.

Class Decorator to raise an Error/Warning when AtomGroup in an AnalysisBase class is not charge neutral. The behaviour of the warning can be controlled with the filter attribute. If the AtomGroup's corresponding universe is non-neutral an ValueError is raised.

Parameters
filter (*str*) – Filter type to control warning filter. Common values are: "error" or "default" See *warnings.simplefilter* for more options.

maicos.lib.util.**citation_reminder**(**dois: str*) → str

Prints citations in order to remind users to give due credit.

Parameters
dois (*list*) – dois associated with the method which calls this. Possible dois are registered in *maicos.lib.util.DOI_LIST*.

Returns
cite – formatted citation reminders

Return type
str

maicos.lib.util.**correlation_analysis**(*timeseries: ndarray*) → float

Timeseries correlation analysis.

Analyses a timeseries for correlation and prints a warning if the correlation time is larger than the step size.

Parameters
timeseries (*numpy.ndarray*) – Array of (possibly) correlated data.

Returns
corrtime – Estimated correlation time of *timeseries*.

Return type
float

maicos.lib.util.**get_center**(*atomgroup: AtomGroup*, *bin_method: str*, *compound: str*) → ndarray

Center attribute for an `MDAnalysis.core.groups.AtomGroup`.

This function acts as a wrapper for the `MDAnalysis.core.groups.AtomGroup.center()` method, providing a more user-friendly interface by automatically determining the appropriate weights based on the chosen binning method.

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A `AtomGroup` for which the calculations are performed.
- **bin_method** ({"com", "cog", "coc"}) – Method for the position binning.

  The possible options are center of mass ("com"), center of geometry ("cog"), and center of charge ("coc").
- **compound** ({"group", "segments", "residues", "molecules", "fragments"}) – The compound to be used in the center calculation. For example, "residue", "segment", etc.

Returns
The coordinates of the calculated center.

Return type
np.ndarray

Raises
ValueError – If the provided `bin_method` is not one of {"com", "cog", "coc"}.

maicos.lib.util.**get_cli_input**() → str

Return a proper formatted string of the command line input.

Returns
A string representing the command line input in a proper format.

Return type
str

maicos.lib.util.**get_compound**(*atomgroup: AtomGroup*) → str

Returns the highest order topology attribute.

The order is "molecules", "fragments", "residues". If the topology contains none of those attributes, an AttributeError is raised.

Parameters
atomgroup (*MDAnalysis.core.groups.AtomGroup*) – A AtomGroup for which the calculations are performed.

Returns
Name of the topology attribute.

Return type
str

Raises
AttributeError – *atomgroup* is missing any connection information"

maicos.lib.util.**maicos_banner**(*version: str = ''*, *frame_char: str = '-'*) → str

Prints ASCII banner resembling the MAICoS Logo with 80 chars width.

Parameters

- **version** (*str*) – Version string to add to the banner.
- **frame_char** (*str*) – Character used to as framing around the banner.

Returns
banner – formatted banner

Return type
str

maicos.lib.util.**render_docs**(*func: Callable*) → Callable

Replace all template phrases in the functions docstring.

Keys for the replacement are taken from in *maicos.lib.util.DOC_DICT*.

Parameters
func (*callable*) – The callable (function, class) where the phrase old should be replaced.

Returns
callable with replaced phrase

Return type
Callable

maicos.lib.util.**trajectory_precision**(*trajectory: ReaderBase*, *dim: int = 2*) → ndarray

Detect the precision of a trajectory.

Parameters

- **trajectory** (*MDAnalysis.coordinates.base.ReaderBase*) – Trajectory from which the precision is detected.
- **dim** (*{2, 0, 1}*) – Dimension along which the precision is detected.

Returns

precision – Precision of each frame of the trajectory.

If the trajectory has a high precision, its resolution will not be detected, and a value of 1e-4 is returned.

Return type
numpy.ndarray

maicos.lib.util.**unit_vectors_cylinder**(*atomgroup: AtomGroup*, *grouping: str*, *bin_method: str*, *dim: int*, *pdim: str*) → ndarray

Calculate cylindrical unit vectors in cartesian coordinates.

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A `AtomGroup` for which the calculations are performed.
- **grouping** ({"atoms", "residues", "segments", "molecules", "fragments"}) – Atom grouping for the calculations.

  The possible grouping options are the atom positions (in the case where `grouping="atoms"`) or the center of mass of the specified grouping unit (in the case where `grouping="residues"`, `"segments"`, `"molecules"` or `"fragments"`).
- **bin_method** ({"com", "cog", "coc"}) – Method for the position binning.

  The possible options are center of mass (`"com"`), center of geometry (`"cog"`), and center of charge (`"coc"`).
- **dim** (*{0, 1, 2}*) – Dimension for binning (`x=0`, `y=1`, `z=1`).
- **pdim** ({"r", "z"}) – direction of the projection

Returns
Array of the calculated unit vectors with shape (3,) for *pdim='z'* and shape (3,n) for *pdim='r'*. The length of *n* depends on the grouping.

Return type
numpy.ndarray

maicos.lib.util.**unit_vectors_planar**(*atomgroup: AtomGroup*, *grouping: str*, *pdim: int*) → ndarray

Calculate unit vectors in planar geometry.

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A `AtomGroup` for which the calculations are performed.
- **grouping** ({"atoms", "residues", "segments", "molecules", "fragments"}) – Atom grouping for the calculations.

  The possible grouping options are the atom positions (in the case where `grouping="atoms"`) or the center of mass of the specified grouping unit (in the case where `grouping="residues"`, `"segments"`, `"molecules"` or `"fragments"`).
- **pdim** (*{0, 1, 2}*) – direction of the projection

Returns
the unit vector

Return type
numpy.ndarray

maicos.lib.util.**unit_vectors_sphere**(*atomgroup: AtomGroup*, *grouping: str*, *bin_method: str*) → ndarray

Calculate spherical unit vectors in cartesian coordinates.

Parameters

- **ATOMGROUP_PARAMETER** –
- **grouping** ({"atoms", "residues", "segments", "molecules", "fragments"}) – Atom grouping for the calculations.

  The possible grouping options are the atom positions (in the case where grouping="atoms") or the center of mass of the specified grouping unit (in the case where grouping="residues", "segments", "molecules" or "fragments").
- **bin_method** ({"com", "cog", "coc"}) – Method for the position binning.

  The possible options are center of mass ("com"), center of geometry ("cog"), and center of charge ("coc").

Returns

Array of the calculated unit vectors with shape (3,n). The length of *n* depends on the grouping.

Return type

numpy.ndarray

maicos.lib.util.**unwrap_refgroup**(*original_class*)

Class decorator error if *unwrap* = *False* and *refgroup !*= *None*.

Weighting functions

Weight functions used for spatial binned analysis modules.

maicos.lib.weights.**density_weights**(*atomgroup: AtomGroup*, *grouping: str*, *dens: str*) → ndarray

Weights for density calculations.

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A AtomGroup for which the calculations are performed.
- **grouping** ({"atoms", "residues", "segments", "molecules", "fragments"}) – Atom grouping for the calculations.

  The possible grouping options are the atom positions (in the case where grouping="atoms") or the center of mass of the specified grouping unit (in the case where grouping="residues", "segments", "molecules" or "fragments").
- **dens** ({"mass", "number", "charge"}) – density type to be calculated.

Returns

1D array of calculated weights. The length depends on the grouping.

Return type

numpy.ndarray

Raises

ValueError – if grouping or dens parameter is not supported.

maicos.lib.weights.**diporder_pair_weights**(*g1: AtomGroup*, *g2: AtomGroup*, *compound: str*) → ndarray

Normalized dipole moments as weights for general diporder RDF calculations.

maicos.lib.weights.**diporder_weights**(*atomgroup: AtomGroup*, *grouping: str*, *order_parameter: str*, *get_unit_vectors: Unit_vector*) → ndarray

Weights for general diporder calculations.

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A AtomGroup for which the calculations are performed.
- **grouping** ({"atoms", "residues", "segments", "molecules", "fragments"}) – Atom grouping for the calculations.

  The possible grouping options are the atom positions (in the case where grouping="atoms") or the center of mass of the specified grouping unit (in the case where grouping="residues", "segments", "molecules" or "fragments").
- **order_parameter** ({"P0", "cos_theta", "cos_2_theta"}) –

  Order parameter to be calculated:

  - "P0": total dipole moment projected on an axis
  - "cos_theta": cosine of the dipole moment with an axis
  - "cos_2_theta": squred cosine with an axis.
- **get_unit_vectors** (*Callable*) – Callable that returns unit vectors on which the projection is performed. Returned unit_vectors can either be of shape (3,) or of shape (n, 3). For a shape of (3,) the same unit vector is used for all calculations.

maicos.lib.weights.**temperature_weights**(*atomgroup: AtomGroup*, *grouping: str*) → ndarray

Weights for temperature calculations.

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A AtomGroup for which the calculations are performed.
- **grouping** ({"atoms", "residues", "segments", "molecules", "fragments"}) – Atom grouping for the calculations.

  The possible grouping options are the atom positions (in the case where grouping="atoms") or the center of mass of the specified grouping unit (in the case where grouping="residues", "segments", "molecules" or "fragments").

Returns
1D array of calculated weights. The length depends on the grouping.

Return type
numpy.ndarray

Raises
NotImplementedError – Currently only works for *grouping='atoms'*

maicos.lib.weights.**velocity_weights**(*atomgroup: AtomGroup*, *grouping: str*, *vdim: int*) → ndarray

Weights for velocity calculations.

The function normalises by the number of compounds.

Parameters

- **atomgroup** (*MDAnalysis.core.groups.AtomGroup*) – A AtomGroup for which the calculations are performed.

- **grouping** ({"atoms", "residues", "segments", "molecules", "fragments"}) – Atom grouping for the calculations.

  The possible grouping options are the atom positions (in the case where grouping="atoms") or the center of mass of the specified grouping unit (in the case where grouping="residues", "segments", "molecules" or "fragments"). vdim : {0, 1, 2} Dimension for velocity binning (x=0, y=1, z=1).

Returns
1D array of calculated weights. The length depends on the grouping.

Return type
numpy.ndarray

Tables

The module contains static lookup tables for atom typing etc.

The tables are dictionaries that are indexed by elements.

maicos.lib.tables.**CM_parameters = {'Al': <maicos.lib.tables.CM_parameter object>, 'Ar': <maicos.lib.tables.CM_parameter object>, 'B': <maicos.lib.tables.CM_parameter object>, 'Be': <maicos.lib.tables.CM_parameter object>, 'Br': <maicos.lib.tables.CM_parameter object>, 'C': <maicos.lib.tables.CM_parameter object>, 'Ca': <maicos.lib.tables.CM_parameter object>, 'Cl': <maicos.lib.tables.CM_parameter object>, 'F': <maicos.lib.tables.CM_parameter object>, 'H': <maicos.lib.tables.CM_parameter object>, 'He': <maicos.lib.tables.CM_parameter object>, 'I': <maicos.lib.tables.CM_parameter object>, 'K': <maicos.lib.tables.CM_parameter object>, 'Kr': <maicos.lib.tables.CM_parameter object>, 'Li': <maicos.lib.tables.CM_parameter object>, 'Mg': <maicos.lib.tables.CM_parameter object>, 'N': <maicos.lib.tables.CM_parameter object>, 'Na': <maicos.lib.tables.CM_parameter object>, 'Ne': <maicos.lib.tables.CM_parameter object>, 'O': <maicos.lib.tables.CM_parameter object>, 'P': <maicos.lib.tables.CM_parameter object>, 'Rb': <maicos.lib.tables.CM_parameter object>, 'S': <maicos.lib.tables.CM_parameter object>, 'Se': <maicos.lib.tables.CM_parameter object>, 'Si': <maicos.lib.tables.CM_parameter object>, 'Xe': <maicos.lib.tables.CM_parameter object>}**

Cromer-Mann X-ray scattering factors computed from numerical Hartree-Fock wave functions. See Acta Cryst. A 24 (1968) p. 321

maicos.lib.tables.**atomtypes** = {'AR': 'Ar', 'BR': 'Br', 'Br': 'Br', 'C': 'C', 'C*': 'C', 'C0': 'Ca', 'CA': 'C', 'CA2+': 'Ca', 'CB': 'C', 'CC': 'C', 'CCHL': 'C', 'CG2R51': 'C', 'CG2R53': 'C', 'CG2R61': 'C', 'CG311': 'C', 'CG321': 'C', 'CG324': 'C', 'CG331': 'C', 'CG334': 'C', 'CG3C51': 'C', 'CG3C52': 'C', 'CG3RC1': 'C', 'CH0': 'C', 'CH1': 'CH1', 'CH2': 'CH2', 'CH2r': 'CH2', 'CH3': 'CH3', 'CH4': 'CH4', 'CK': 'C', 'CL': 'Cl', 'CL-': 'Cl', 'CLA': 'C', 'CM': 'C', 'CMET': 'CH3', 'CN': 'C', 'CQ': 'C', 'CR': 'C', 'CR1': 'CH1', 'CT': 'C', 'CU': 'Cu', 'CU1+': 'Cu', 'CU2+': 'Cu', 'CV': 'C', 'CW': 'C', 'Cl': 'Cl', 'Cs': 'CS', 'DUM': 'DUM', 'F': 'Fe', 'FE': 'Fe', 'H': 'H', 'H1': 'H', 'H2': 'H', 'H3': 'H', 'H4': 'H', 'H5': 'H', 'HA': 'H', 'HC': 'H', 'HGA1': 'H', 'HGA2': 'H', 'HGA3': 'H', 'HGP1': 'H', 'HGR52': 'H', 'HGR53': 'H', 'HGR61': 'H', 'HO': 'H', 'HP': 'H', 'HS': 'H', 'HT': 'H', 'HW': 'H', 'HW_spc': 'H', 'HW_tip4p': 'H', 'HW_tip4pew': 'H', 'HW_tip5p': 'H', 'I': 'I', 'IB': 'Na', 'K': 'K', 'Li': 'Li', 'MCH3': 'DUM', 'MG': 'Mg', 'MG2+': 'Mg', 'MNH3': 'DUM', 'MW': 'DUM', 'N': 'N', 'N*': 'N', 'N2': 'N', 'N3': 'N', 'NA': 'N', 'NA+': 'Na', 'NB': 'N', 'NC': 'N', 'NE': 'NH', 'NG2R52': 'N', 'NL': 'NH3', 'NR': 'N', 'NT': 'NH2', 'NZ': 'NH2', 'Na': 'Na', 'O': 'O', 'O2': 'O', 'OA': 'O', 'OE': 'O', 'OG311': 'O', 'OH': 'O', 'OM': 'O', 'OMET': 'O', 'OS': 'O', 'OT': 'O', 'OW': 'O', 'OW_spc': 'O', 'OW_tip4p': 'O', 'OW_tip4p2005': 'O', 'OW_tip4pew': 'O', 'OW_tip5p': 'O', 'P': 'P', 'Rb': 'Rb', 'S': 'S', 'SH': 'Si', 'VW': 'DUM', 'ZN2+': 'Zn', 'Zn': 'Zn'}

Translation of `MDAnalysis.AtomGroup.types` to chemical elements.

5.4 Explanations

This section provides the theory behind some of the most complex analysis modules and explains the general design of MAICoS. Its purpose is to provide more clarity and understanding of what MAICoS is all about.

5.4.1 General design

Foundation

MAICoS analysis modules are built on top of stacked *Core classes* as shown in the UML chart above. For spatial dependent analysis, these are split into the geometries:

- *Planar classes*,
- *Cylinder classes*,
- and *Sphere classes*.

Each sub class inherits attributes and provides geometry-specific methods and attributes. The flow chart is shown in the figure above. The foundation for all these classes is *`maicos.core.base.AnalysisBase`*, inherited and extended from `MDAnalysis.analysis.base.AnalysisBase`. *`maicos.core.base.AnalysisBase`* takes case of the general aspects of each analysis, which will be discussed in detail below:

1. **Atom Selection -** MAICoS builds on top of the MDAnalysis Universe and atom selection system, therefore all analysis modules work only on subsets of the whole simulation. This allows investigating different species components individually, for example splitting the contributions of solvent and solute to a single observable. Moreover, many MAICoS analysis modules are able to process several atom selections from one simulation within one analysis run by providing a `list` of atom selections. This reduces I/O loads and operations and gains a speed up for the analysis.

1. **Translational coordinate transformations and unit cell wrapping -** MAICoS works with a reference structure denoted by *refgroup* which center of mass (com for short) serves as the coordinate origin for every analysis.

MDAnalysis's cell dimension and coordinates range from 0 to L where L is the dimension of the simulation box. Therefore, MAICoS defines the origin at the center of the simulation cell.

Within each frame of the analysis, the *refgroup*'s com is translated to the origin and all coordinates are wrapped into the primary unit cell. Additionally, it is possible to unwrap molecules afterwards since some analysis require whole molecules (e.g. dielectric). With this centering, the investigation of systems that translate over time is made possible, such as for example soft interfaces or moving molecules. However, users are not forced to give a *refgroup*. If no such reference structure is given, MAICoS takes the frame specific center of the simulation cell as the origin.

User-provided ranges for spatial analysis are always with respect to the *refgroup* and not in absolute box coordinates. For example, a 1-dimensional planar analysis ranging from -2 (Å) to 0 considers atoms on the left half space of the *refgroup*.

2. **Trajectory iteration -** Each module implements an initialization, a prepare, a single frame and a conclude method. The *AnalysisBase* will perform an analysis that is based on these provided methods. It is possible to provide an initial and final frame as well as a step size or to analyse individual frames.
3. **Time averaging of observables -** For observables that have to be time-averaged, `maicos.core.base.AnalysisBase` provides a frame dictionary. Each key has to be updated within the (private) *_single_frame* method and the mean and the variance of each observable will be provided within a `mean` and a `var` dictionary. Each key name within these two dictionaries is the same as within the frame dictionary.
4. **On-the-fly output -** MAICoS is able to update analysis results during the analysis. This can be particularly useful for long analysis providing a way to check the correctness of analysis parameters during the run.
5. **Correlation time estimation -** For the calculation of the mean and the standard deviation, MAICoS assumes uncorrelated data to compute reasonable error estimates. Since users may not know the correlation time within their simulation, MAICoS estimates correlation times for representative observables and warns users if their averages are obtained from correlated data. The correlation analysis gets handled by `maicos.core.base.AnalysisBase` if the *single_frame* method of the used class returns a value to perform the analysis on. You can find general info about which class uses which observable for the analysis below, and more detailed information in the *Reference guides*. The correlation time gets calculated using the `correlation time function`. The generation of warnings for the users gets handled by the `correlation analysis function`.

   For dielectric analysis, MAICoS uses the total dipole moment parallel to the direction of the analysis. For other spatial-dependant analysis, the correlation time is estimated from the central bin of the refgroup; in the center of the simulation cell. This translates to the middle bin of the profile for planar analyses and the first bin for cylindrical or spherical profiles.

Spatial Dependent Analysis

Spatial dependent analyses are crucial for interfacial and confined systems. Based on the *AnalysisBase* in combination with a `maicos.core.base.ProfileBase` class, MAICoS provides intermediate *Core classes* for the three main geometries:

- `maicos.core.planar.PlanarBase`,
- `maicos.core.cylinder.CylinderBase`,
- and `maicos.core.sphere.SphereBase`.

These modules take care of the coordinate transformations, of the spatial boundaries, and of the spatial resolution of the analysis.

A design concept of MAICoS for spatial analysis is that the user always provides the spatial resolution via the *bin_width* parameter rather than a number of bins. Therefore, the same analysis code is easily transferable to different simulation sizes without additional considerations about the spatial resolution.

Based on the three geometric base classes, three corresponding high level classes are provided:

- *maicos.core.planar.ProfilePlanarBase*,
- *maicos.core.cylinder.ProfileCylinderBase*,
- and *maicos.core.sphere.ProfileSphereBase*.

When developing a new analysis class based on one of theses three classes, only a single *weight* function has to be provided. All current *Weighting functions* are documented. For instance, the atomic weight could be the masses, thus resulting in mass density profiles as done in *DensityPlanar*, atomic or molecular velocities as for *VelocityPlanar*, or the dipolar orientations as used by the *DiporderPlanar* class.

More details on each base class are given in the *API Documentation*. For detailed information on the physical principles of each module consider the following sections.

5.4.2 Dielectric constant measurement

Dielectric Response of Homogeneous, Isotropic Fluids

The linear dielectric response of a material relates the displacement field D to the electric field E, which in the isotropic, homogenous case can be written as (in SI units)

$$\mathbf{D} = \varepsilon_0 \varepsilon \mathbf{E}$$

where ε_0 is the vacuum permittivity, and ε is the dielectric constant of the insulating medium.

One can relate the dielectric constant of a material to the fluctuations of the dipole moment of a sub-sample even without any perturbation by an external field. Relations of this sort have been known since the 1930s and follow from the fluctuation-dissipation theory[1]. Depending on the boundary conditions, this equation takes different forms, however, the most common boundary conditions of molecular dynamics simulations are tin-foil boundary conditions in conjunction with an Ewald-summation type approach. In this case, we get for a bulk material

$$\varepsilon = 1 + \frac{\langle M^2 \rangle - \langle M \rangle^2}{3\varepsilon_0 V k_\mathrm{B} T}$$

where M is the dipole moment of the sample, V is its volume, k_B is the Boltzmann constant and T is the temperature.

The dipole moment is defined by

$$M = \sum_i \mathbf{r}_i q_i$$

where $\mathbf{r}_i$ is the position of the i-th particle and q_i is the charge of the i-th particle. Notably, this allows the calculation of the dielectric response from equilibrium simulations without the need to explicitly define an external field in simulations.

This analysis - valid for isotropic and homogeneous systems - is implemented in `MDAnalysis.analysis.dielectric.DielectricConstant` and can directly be applied to trajectories of homogeneous systems.

[1] John G. Kirkwood. The Dielectric Polarization of Polar Liquids. *J. Chem. Phys.*, 7(10):911–919, October 1939. doi:10.1063/1.1750343.

Dielectric Response of Fluids at Interfaces and in Confinement

Electrostatic Theory

The relationship between the electric field and the dielectric response shown above is only valid for isotropic homogeneous systems, where the properties of the material are the same throughout. However, there is also a need for calculating the dielectric response of anistropic inhomogeneous systems. For instance, fluids confined in a porous material are of great importance for many technological processes, such as energy storage devices like batteries and capacitors. In these devices, a nano-porous electrode is used to increase the surface area and improve the capacity of the device. Another common example are catalysts, where an increased surface area is used to increase the rate of a chemical reaction, and thus porous catalysts are often utilized.

The presence of interfaces alters the dielectric response of the fluid in two ways. First, the response is not isotropic anymore, but depends on the orientation of the electric field. Second, the response varies with the distance from the surface of the porous material, i.e., it becomes inhomogeneous.

In the following discussion, we will focus on pores with planar symmetry, also known as "slit pores" and implemented in `maicos.DielectricPlanar`. However, similar concepts apply to other types of pore geometries, such as ones with cylindrical or spherical symmetries implemented in `maicos.DielectricCylinder` and `maicos.DielectricSphere`.

Without loss of generality, we will assume that the pore is aligned along the z-axis.

The non-local, anisotropic, linear dielectric response of a fluid can generally be written as[2]

$$D(\mathbf{r}) = \varepsilon_0 \int_V \mathrm{d}^3 r' \varepsilon(\mathbf{r}, \mathbf{r}') E(\mathbf{r}')$$

where $\varepsilon(\mathbf{r}, \mathbf{r}')$ is the dielectric tensor, which describes how the dielectric response of the fluid at position $\mathbf{r}$ is affected by the electric field $E(\mathbf{r}')$ throughout the volume V of the fluid. The convolution integral accounts for the non-local influences of the fluid response at other locations.

In planar symmetry, we can simplify the above expression further, because the Maxwell relations give

$$\nabla \times \mathbf{E} = 0$$

in the absence of external magnetic fields. Because of the planar symmetry, we know that the $\mathbf{E}$ only varies with respect to z. Hence, the above gives $\partial_z E_y = \partial_z E_x = 0$, implying that the parallel components of the electric field do not vary with z.

Thus, we can simplify the anisotropic, non-linear equation above in the parallel case to

$$D_\parallel = \epsilon_0 E_\parallel \int \mathrm{d}z' \epsilon_\parallel(z, z') =: \epsilon_0 \epsilon_\parallel(z) E_\parallel$$

where the marginal integration of $\varepsilon_\parallel(\mathbf{r}, \mathbf{r}')$ defines the dielectric profile $\varepsilon_\parallel(z)$. It is important to note that this derivation starts with non-local assumptions and is exact in the case of planar geometries discussed here (similar derivations apply also for cylindrical and spherical symmertries). Thus, $\varepsilon_\parallel(z)$ fully captures the non-locality of the confined fluid's response and does not require additional assumptions.

In the absence of "free charges" we can use the macroscopic Maxwell equation

$$\nabla \cdot \mathbf{D} = 0$$

to derive the perpendicular dielectric profile.

[2] Douwe Jan Bonthuis, Stephan Gekle, and Roland R. Netz. Profile of the Static Permittivity Tensor of Water at Interfaces: Consequences for Capacitance, Hydration Interaction and Ion Adsorption. *Langmuir*, 28(20):7679–7694, 2012. doi:10.1021/la2051564.

Warning: This requires that no free charges are used in simulations, which means that no ions can be included in simulations. This is a common pitfall and leads to a wrong analysis.

The above equation gives us the important relation of $\partial_z \mathbf{D}_z = 0$, which implies that the perpendicular components of the displacement field do not vary with z. Thus, if we start with the inverse dielectric response, defined as

$$E(z) = \varepsilon_0^{-1} \int \mathrm{d}z' \varepsilon^{-1}(z, z') D(z')$$

where $\varepsilon^{-1}(z, z')$ is the matrix inverse of the dielectric tensor. Similar to above, we use the fact that D does not vary with z and simplify

$$E_\perp = \epsilon_0^{-1} D_\perp \int \mathrm{d}z' \epsilon_\perp^{-1}(z, z') =: \epsilon_0^{-1} \epsilon_\perp^{-1}(z) D_\perp$$

where the marginal integration of $\varepsilon_\perp^{-1}(\mathbf{r}, \mathbf{r}')$ defines the inverse dielectric profile $\varepsilon_\perp^{-1}(z)$.

In summary, if one has no magnetic fields and no free charges, the dielectric profiles $\varepsilon_\perp^{-1}(z)$ and $\varepsilon_\parallel(z)$ fully define the linear, anisotropic, non-local response of a system in planar confinement.

Fluctuation-Dissipation Theorem

As was briefly discussed for the homogenous case, the dielectric response of a system can be calculated from equilibrium simulations without the need to explicitly define an external field in simulations, using a fluctuation dissipation theorem. This can be derived by identifying the linear response under consideration, in this case the dielectric response to a derivative of the expected value of an observable, in this case the polarization density. The expectation value is calculated using statistical mechanics. One can then show[3 Page 149, 24] that the dielectric response formalism is given by

$$\epsilon_\parallel(z) = 1 + \frac{\langle m_\parallel(z) M_\parallel \rangle - \langle m_\parallel(z) \rangle \langle M_\parallel \rangle}{\epsilon_0 k_\mathrm{B} T}$$

for the **parallel** dielectric profile, and

$$\epsilon_\perp^{-1}(z) = 1 - \frac{\langle m_\perp(z) M_\perp \rangle - \langle m_\perp(z) \rangle \langle M_\perp \rangle}{\epsilon_0 k_\mathrm{B} T},$$

for the **inverse** perpendicular dielectric profile.

Note that we still need to define how to calculate $m_\parallel(z)$ and $m_\perp(z)$. For the perpendicular polarization density, we have[Page 149, 2]

$$m_\perp(z) = -\int_0^z \mathrm{d}z' \rho(z').$$

For the parallel case, we have to derive the lateral component of the polarization density as a function of the coordinate z. This can be done by introducing multiple virtual cuts perpendicular to any lateral axis, such as the x or y axis[Page 149, 24]. During this step one has to take care to only cut molecules along this cutting plane, which requires careful treatment of the periodic boundary conditions commonly employed in simulations. Identifying the (non-zero) total charge on one side of the cut with the surface charge along the plane of the virtual cut via Gauss' theorem we can integrate out the dependency of the lateral axis of the cut and average over multiple such cuts. This gives a good estimate for the average surface charge density $\sigma(z)$ w.r.t the coordinate z. Finally, we can identify

$$m_\parallel(z) = \mp\sigma(z).$$

[3] Harry A. Stern and Scott E. Feller. Calculation of the dielectric permittivity profile for a nonuniform system: Application to a lipid bilayer simulation. *The Journal of Chemical Physics*, 118(7):3401–3412, February 2003. doi:10.1063/1.1537244.

[4] Alexander Schlaich, Ernst W. Knapp, and Roland R. Netz. Water Dielectric Effects in Planar Confinement. *Phys. Rev. Lett.*, 117(4):048001, July 2016. doi:10.1103/PhysRevLett.117.048001.

Boundary Conditions

The above equations for $\varepsilon_{\parallel}(z)$ and $\varepsilon_{\perp}^{-1}(z)$ are derived under 2d periodicity. In simulations, this entails using periodic boundary conditions only in the x and y directions. In most of the typically employed simulation codes, electrostatics are calculated using a Ewald-summation type approach. This includes direct Ewald sums or the faster meshed Ewald sums (such as P3M, and PME). However, in their usual formulation these codes calculate 3d-periodic systems and thus do not meet the assumptions of the derivation shown above.

In order to use the above, one can use the 2d Ewald sum or corrections thereof, such as the correction of Yeh and Berkovitz[5] or the ELC[6].

However, one can also correct for the 3d electrostatics of an uncorrected Ewald-sum in the fluctuation dissipation formalism directly as shown in refs.[3 Page 150, 4]

For tin-foil boundary conditions, one gets[Page 150, 4]

$$\epsilon_{\perp}^{-1}(z) = 1 - \frac{\langle m_{\perp}(z) M_{\perp}\rangle - \langle m_{\perp}(z)\rangle\langle M_{\perp}\rangle}{\epsilon_0 k_{\mathrm{B}} T + C_{\perp}/V},$$

where $C_{\perp} = \int \mathrm{d}m_{\perp}(z)$.

Note, that a very close formula[Page 150, 3] can also be derived for arbitrary boundary conditions at infinity, which some simulation codes can also utilize. As most simulations nowadays are performed using tin-foil boundary conditions, MAICoS does not provide these special cases and we do not recommend that simulations for the calculation of dielectric profiles are performed with other boundary conditions.

Note: The above equation reduces to the correct 2d periodic system if one includes an explicit vacuum layer in the z direction of infinite (sufficiently large) size, such that the influence between periodic images over the z direction can be approximated as a dipole interaction. This approach is analogous to the Yeh and Berkovitz correction[5] and may be used to calculate the dielectric profiles for physical systems with 2d-symmetry when corrections are not available. In these situations, we recommend to use a padding vacuum layer such that the system is 3x the physical system size in z direction.

However, there are systems which truly are 3d-periodic, such as stacks of lipid membranes. In these cases, one has to also use the above formula which includes the dipole corrections, but only simulate the physical system, without a padding vacuum layer.

The correction for 3d periodic systems with tin-foil boundary conditions can be turned on using the parameter `is_3d`.

References

5.4.3 Small-angle X-ray scattering

MD Simulations often complement conventional experiments, such as X-ray crystallography, Nuclear Magnetic Resonance (NMR) spectroscopy and Atomic-Force Microscopy (AFM). X-ray crystallography is a method by which the structure of molecules can be resolved. X-rays of wavelength 0.1 to 100 Å are scattered by the electrons of atoms. The intensities of the scattered rays are often amplified using by crystals containing a multitude of the studied molecule positionally ordered. The molecule is thereby no longer under physiological conditions. However, the study of structures in a solvent should be done under physiological conditions (in essence, this implies a disordered, solvated fluid system); therefore X-ray crystallography does not represent the ideal method for such systems. Small-Angle X-ray Scattering

[5] In-Chul Yeh and Max L. Berkowitz. Ewald summation for systems with slab geometry. *The Journal of Chemical Physics*, 111(7):3155–3162, August 1999. doi:10.1063/1.479595.

[6] Axel Arnold, Jason de Joannis, and Christian Holm. Electrostatics in periodic slab geometries. I. *The Journal of Chemical Physics*, 117(6):2496–2502, July 2002. doi:10.1063/1.1491955.

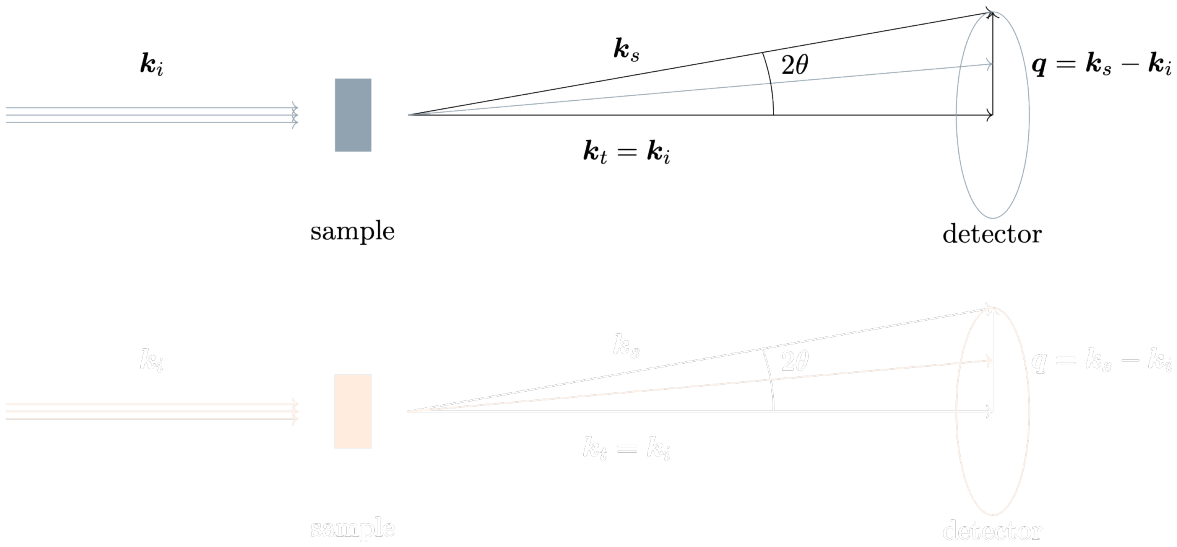
(abbreviated to SAXS) allows for measurements of molecules in solutions. With this method the shape and size of the molecule and also distances within it can be obtained. In general, for larger objects, the information provided by SAXS can be converted to information about the object's geometry via the Bragg-Equation

$$n \cdot \lambda = 2 \cdot d \cdot \sin(\theta)$$

with $n \in \mathbb{N}$, λ the wavelength of the incident wave, d the size of the diffracting object, and θ the scattering angle. For small angles, d and θ are approximately inversely proportional to each other, which means larger objects scatter X-rays at smaller angles.

Experiments

The measured quantity in SAXS experiments is the number of elastically scattered photons as a function of the scattering angle 2θ, i.e. the intensity of the scattered rays across a range of small angles. The general set-up of a SAXS experiment is shown in figure below.



The experiments are carried out by placing the sample of interest in a highly monochromatic and collimated (parallel) X-ray beam of wavelength λ. When the incident rays with wave vector $\boldsymbol{k}_i$ reach the sample they scatter. The scattered rays, with wave vector $\boldsymbol{k}_s$, are recorded by a 2D-detector revealing a diffraction pattern.

Since the scattering agents in the sample are electrons, X-Ray diffraction patterns reveal the electron density. Since the scattering is elastic, the magnitudes of the incident and scattered waves are the same: $|\boldsymbol{k}_i| = |\boldsymbol{k}_s| = 2\pi/\lambda$. The scattering vector is $\boldsymbol{q} = \boldsymbol{k}_s - \boldsymbol{k}_i$ with a magnitude of $q = |\boldsymbol{q}| = 4\pi \sin(\theta)/\lambda$. The structure factor can be obtained from the intensity of the scattered wave, $I_s(\boldsymbol{q})$, and the correspnding form factor $f(q)$, which involves a frourier transform of the element-specific local electron density and thus determines the amplitude of the scattered wave of a single element.

Simulations

In simulations, the structure factor and scattering intensities $S(\boldsymbol{q})$ can be extracted directly from the positions of the particles. `maicos.Saxs` calculates these factors. The calculated scattering intensities can be directly compared to the experimental one without any further processing. In the following we derive the essential relations. We start with the scattering intensity which is expressed as

$$I_s(\boldsymbol{q}) = A_s(\boldsymbol{q}) \cdot A_s^*(\boldsymbol{q}) \,,$$

with the amplitude of the elastically scattered wave

$$A_s(\boldsymbol{q}) = \sum_{j=1}^{N} f_j(q) \cdot e^{-i\boldsymbol{q}\boldsymbol{r}_j} \,,$$

where $f_j(q)$ is the element-specific form factor of atom j and $\boldsymbol{r}_j$ the position of the j th atom out of N atoms.

The scattering intensity can be evaluated for wave vectors $\boldsymbol{q} = 2\pi(L_x n_x, L_y n_y, L_z n_z)$, where $n \in \mathbb{N}$ and L_x, L_y, L_z are the box lengths of cubic cells.

Note: `maicos.Saxs` can analyze any cells by mapping coordinates back onto cubic cells.

The complex conjugate of the amplitude is

$$A_s^*(\boldsymbol{q}) = \sum_{j=1}^{N} f_j(q) \cdot e^{i\boldsymbol{q}\boldsymbol{r}_j} \,.$$

The scattering intensity therefore can be written as

$$I_s(\boldsymbol{q}) = \sum_{j=1}^{N} f_j(q) e^{-i\boldsymbol{q}\boldsymbol{r}_j} \cdot \sum_{k=1}^{N} f_k(q) e^{i\boldsymbol{q}\boldsymbol{r}_k} \,.$$

With Euler's formula $e^{i\phi} = \cos(\phi) + i\sin(\phi)$ the intensity is

$$I_s(\boldsymbol{q}) = \sum_{j=1}^{N} f_j(q)\cos(\boldsymbol{q}\boldsymbol{r}_j) - i\sin(\boldsymbol{q}\boldsymbol{r}_j) \cdot \sum_{k=1}^{N} f_k(q)\cos(\boldsymbol{q}\boldsymbol{r}_k) + i\sin(\boldsymbol{q}\boldsymbol{r}_k) \,.$$

Multiplication of the terms and simplifying yields the final expression for the intensity of a scattered wave as a function of the wave vector and with respect to the particle's form factor

$$I_s(\boldsymbol{q}) = \left[\sum_{j=1}^{N} f_j(q)\cos(\boldsymbol{q}\boldsymbol{r}_j)\right]^2 + \left[\sum_{j=1}^{N} f_j(q)\sin(\boldsymbol{q}\boldsymbol{r}_j)\right]^2 \,.$$

For systems containing only one kind of atom the structure factor is connected to the scattering intensity via

$$I_s(\boldsymbol{q}) = [f(q)]^2 S(\boldsymbol{q}) \,.$$

For any system the structure factor can be written as

$$S(\boldsymbol{q}) = \left\langle \frac{1}{N}\sum_{j=1}^{N}\cos(\boldsymbol{q}\boldsymbol{r}_j)\right\rangle^2 + \left\langle \frac{1}{N}\sum_{j=1}^{N}\sin(\boldsymbol{q}\boldsymbol{r}_j)\right\rangle^2 \,.$$

The limiting value $S(0)$ for $q \to 0$ is connected to the isothermal compressibility[1] and the element-specific form factors $f(q)$ of a specific atom can be approximated with

$$f(\sin\theta/\lambda) = \sum_{i=1}^{4} a_i e^{-b_i \sin^2\theta/\lambda^2} + c\,.$$

Expressed in terms of the scattering vector we can write

$$f(q) = \sum_{i=1}^{4} a_i e^{-b_i q^2/(4\pi)^2} + c\,.$$

The element-specific coefficients $a_{1,\ldots,4}$, $b_{1,\ldots,4}$ and c are documented[2].

Connection of the structure factor to the radial distribution function

If the system's structure is determined by pairwise interactions only, the density correlations of a fluid are characterized by the pair distribution function

$$g(\boldsymbol{r}, \boldsymbol{r'}) = \frac{\langle \rho^{(2)}(\boldsymbol{r}, \boldsymbol{r'})\rangle}{\langle \rho(\boldsymbol{r})\rangle\langle \rho(\boldsymbol{r'})\rangle}\,,$$

where $\rho^{(2)}(\boldsymbol{r}, \boldsymbol{r'}) = \sum_{i,j=1, i\neq j}^{N} \delta(\boldsymbol{r} - \boldsymbol{r}_i)\delta(\boldsymbol{r} - \boldsymbol{r}_j)$ and $\rho(\boldsymbol{r}) = \sum_{i=1}^{N} \delta(\boldsymbol{r} - \boldsymbol{r}_i)$ are the two- and one-particle density operators.

For a homogeneous and isotropic system, $g(r) = g(\boldsymbol{r}, \boldsymbol{r'})$ is a function of the distance $r = |\boldsymbol{r} - \boldsymbol{r'}|$ only and is called the radial distribution function (RDF). As explained above, scattering experiments measure the structure factor

$$S(\boldsymbol{q}) = \left\langle \frac{1}{N} \sum_{i,j=1}^{N} \exp(-i\boldsymbol{q} \cdot [\boldsymbol{r}_i - \boldsymbol{r}_j]) \right\rangle\,,$$

which we here normalize only by the number of particles N. For a homogeneous and isotropic system, it is a function of $q = |\boldsymbol{q}|$ only and related to the RDF by Fourier transformation (FT)

$$S^{FT}(q) = 1 + 4\pi\rho \int_0^\infty \mathrm{d}r r \frac{\sin(qr)}{q}(g(r) - 1)\,,$$

which is another way compared for the direct evaluation from trajectories which was derived above. In general this can be as accurate as the direct evaluation if the RDF implementation works for non-cubic cells and is not limited to distances $r_{\max} = L/2$, see[3] for details. However, in usual implementation the RDF can only be obtained until $r_{\max} = L/2$ which leads to a range of $q > q^{\mathrm{FT}}_{\min} = 2\pi/r_{\mathrm{rmax}} = 4\pi/L$. This means that the minimal wave vector that can be resolved is a factor of 2 larger compared compared to the direct evaluation, leading to "cutoff ripples". The direct evaluation should therefore usually be preferred[4].

To compare the RDF and the structure factor you can use `maicos.lib.math.compute_rdf_structure_factor()`. For a detailed example take a look at *Small-angle X-ray scattering*.

[1] Jean-Pierre Hansen and Ian. R. McDonald. *Theory of Simple Liquids*. Elsevier / Academic Press, 3rd ed edition, 2006. ISBN 9780080455075.

[2] E. Prince. *International Tables for Crystallography,Volume C: Mathematical, Physical and Chemical Tables*. Springer, Dordrecht, 3rd ed. edition edition, January 2004. ISBN 978-1-4020-1900-5.

[3] Johannes Zeman, Svyatoslav Kondrat, and Christian Holm. Ionic screening in bulk and under confinement. *The Journal of Chemical Physics*, 155(20):204501, November 2021. doi:10.1063/5.0069340.

[4] Felix Sedlmeier, Dominik Horinek, and Roland R. Netz. Spatial Correlations of Density and Structural Fluctuations in Liquid Water: A Comparative Simulation Study. *J. Am. Chem. Soc.*, 133(5):1391–1398, February 2011. doi:10.1021/ja1064137.

References

5.4.4 Pair distribution functions

The pair distribution function describes the spatial correlation between particles.

Two-dimensional (planar) pair distribution function

Here, we present the two-dimensional pair distribution function $g_{2\mathrm{d}}(r)$, which restricts the distribution to particles which lie on the same surface S_ξ.

Let g_1 be the group of particles which are centered, and g_2 be the group of particles whose density around a g_1 particle is calculated. Furthermore, we define a parametric surface S_ξ as a function of ξ,

$$S_\xi = \{\mathbf{r}_\xi(u,v) | u_{\min} < u < u_{\max}, v_{\min} < v < v_{\max}\}$$

which consists of all points $\mathbf{r}_\xi$. By varying u, v we can reach all points on one surface ξ. Let us additionally consider a circle on that plane $S_{i,r}$ with radius r around atom i given by

$$S_{i,r} = \{\mathbf{r}_{i,r} |\ ||(\mathbf{r}_{i,r} - \mathbf{x_i}|| = r) \wedge (\mathbf{r}_{i,r} \in S_{\xi,i})\}$$

where $S_{\xi,i}$ is the plane in which atom i lies.

Then the two-dimensional pair distribution function is

$$g_{2\mathrm{d}}(r) = \left\langle \sum_i^{N_{g_1}} \frac{1}{L(r,\xi_i)} \frac{\sum_j^{N_{g_2}} \delta(r - r_{ij})\delta(\xi_{ij})}{||\frac{\partial \mathbf{f}_i}{\partial r} \times \frac{\partial \mathbf{f}_i}{\partial \xi}||_{\phi=\phi_j}} \right\rangle$$

where $L(r,\xi_i)$ is the contour length of the circle $S_{i,r}$. $\mathbf{f}_i(r,\gamma,\phi)$ is a parametrization of the circle $S_{i,r}$.

Discretized for computational purposes we consider a volume $\Delta V_{\xi_i}(r)$, which is bounded by the surfaces $S_{\xi_i - \Delta\xi}$, $S_{\xi_i + \Delta\xi}$ and $S_{r - \frac{\Delta r}{2}}, S_{r + \frac{\Delta r}{2}}$. Then our two-dimensional pair distribution function is

$$g_{2\mathrm{d}}(r) = \left\langle \frac{1}{N_{g_1}} \sum_i^{N_{g_1}} \frac{\text{count } (g_2) \text{ in } \Delta V_{\xi_i}(r)}{\Delta V_{\xi_i}(r)} \right\rangle$$

Derivation

Let us introduce cylindrical coordinates r, z, ϕ with the origin at the position of atom i.

$$\begin{aligned} x &= r \cdot \cos\phi \\ y &= r \cdot \sin\phi \\ z &= z \end{aligned}$$

Then the two-dimensional pair distribution is given by

$$g_{2\mathrm{d}}(r, z=0) = \left\langle \sum_i^{N_{g_1}} \frac{1}{2\pi r} \sum_j^{N_{g2}} \delta(r - r_{ij})\delta(z_{ij}) \right\rangle$$

where we have followed the general derivations given above.

For discretized calculation we count the number of atoms per ring as illustrated below

The sketch shows an atom i from group g_1 at the origin in blue. Around the atom a ring volume with average distance r from atom i is shaded in light red. Atoms j from group g_2 are counted in this volume.

One-dimensional (cylindrical) pair distribution functions

Here, we present the one-dimensional pair distribution functions $g_{1\mathrm{d}}(\phi)$ and $g_{1\mathrm{d}}(z)$, which restricts the distribution to particles which lie on the same cylinder along the angular and axial directions respectively.

Let $g2$ be the group of particles whose density around a $g1$ particle is to be calculated and let $g1, g2$ lie in a cylinderical coordinate system (R, z, ϕ).

Then the angular pair distribution function is

$$g_{1\mathrm{d}}(\phi) = \left\langle \sum_{i}^{N_{g1}} \sum_{j}^{N_{g2}} \delta(\phi - \phi_{ij}) \delta(R_{ij}) \delta(z_{ij}) \right\rangle$$

And the axial pair distribution function is

$$g_{1\mathrm{d}}(z) = \left\langle \sum_{i}^{N_{g1}} \sum_{j}^{N_{g2}} \delta(z - z_{ij}) \delta(R_{ij}) \delta(\phi_{ij}) \right\rangle$$

Discretized for computational purposes we consider a volume $\Delta V_{z_i,R_i}(\phi)$, which is bounded by the surfaces $S_{z_i-\Delta z}$, $S_{z_i+\Delta z}$, $S_{R_i-\Delta R}$, $S_{R_i+\Delta R}$ and $S_{\phi-\frac{\Delta\phi}{2}}, S_{\phi+\frac{\Delta\phi}{2}}$. Then our the angular pair distribution function is

$$g_{1\mathrm{d}}(\phi) = \left\langle \frac{1}{N_{g_1}} \sum_{i}^{N_{g_1}} \frac{\text{count } (g_2) \text{ in } \Delta V_{z_i,R_i}(\phi)}{\Delta V_{z_i,R_i}(\phi)} \right\rangle$$

Similarly,

$$g_{1\mathrm{d}}(z) = \left\langle \frac{1}{N_{g_1}} \sum_{i}^{N_{g_1}} \frac{\text{count } (g_2) \text{ in } \Delta V_{\phi_i,R_i}(z)}{\Delta V_{\phi_i,R_i}(z)} \right\rangle$$

5.5 Developer documentation

5.5.1 Getting involved

Contribution via merge requests are always welcome. Source code is available from GitLab. Before submitting a merge request, please read the developer documentation and open an issue to discuss your changes. Use only the *main* branch for submitting your requests.

By contributing to MAICoS, you accept and agree to the following terms and conditions for your present and future contributions submitted to MAICoS. Except for the license granted herein to MAICoS and recipients of software distributed by MAICoS, you reserve all right, title, and interest in and to your contributions.

5.5.2 Getting started

To help with developing start by installing the development dependencies. Our continuous integration pipeline is based on Tox. So you need to install `tox` first

```
pip install tox
# or
conda install -c conda-forge tox
```

Then go to the MAICoS develop project page, hit the `Fork` button and clone your forked branch to your machine.

```
git clone git@gitlab.com:your-user-name/maicos.git
```

Now you have a local version on your machine which you can install by

```
cd maicos
pip install -e .
```

This install the package in development mode, making it importable globally and allowing you to edit the code and directly use the updated version.

5.5.3 Useful developer scripts

The following scripts can be useful to developers:

- `./developer/clean_dist_check.sh`: Clean dist files. Useful before/after `tox -e build`
- `./developer/clean_tempfiles.sh`: Remove all generated files related to Python, including all build caches.

5.5.4 Code of Conduct

As contributors and maintainers of MAICoS, we pledge to respect all people who contribute through reporting issues, posting feature requests, updating documentation, submitting merge requests or patches, and other activities.

We are committed to making participation in this project a harassment-free experience for everyone, regardless of level of experience, gender, gender identity and expression, sexual orientation, disability, personal appearance, body size, race, ethnicity, age, or religion.

Examples of unacceptable behavior by participants include the use of sexual language or imagery, derogatory comments or personal attacks, trolling, public or private harassment, insults, or other unprofessional conduct.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct. Project maintainers who do not follow the Code of Conduct may be removed from the project team.

This code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community.

This Code of Conduct is adapted from the Contributor Covenant, version 1.1.0, available at https://contributor-covenant.org/version/1/1/0/

Contributing your own analysis module

To write your module take a look at the comprehensive example in the documentation of *maicos.core.AnalysisBase*. MAICoS also has more specific base classes for different geometries that make developing modules much easier. You may take a look at the source code at `src/maicos/modules`.

After you wrote your module you can add it in a new file in `src/maicos/modules`. On top of that please also update the list in `src/maicos/modules/__init__.py` accordingly. Also, create a new `.rst` file with your module name in `docs/src/references/modules` similar to the already existing. To finally show the documentation for the other modules add an entry in `docs/src/references/modules/index.rst` in alphabetical order.

All MAICoS modules are also listed in the `README.rst` and you should add your module as well.

Finally, also provide meaningful tests for your module in `test/modules`.

For further questions feel free to ask us on our Discord server.

Testing

Whenever you add a new feature to the code you should also add a test case. Further test cases are also useful if a bug is fixed or you consider something to be worthwhile. Follow the philosophy - the more the better!

You can run all tests by:

```
tox
```

These are exactly the same tests that will be performed online in our GitLab CI workflows.

Also, you can run individual environments if you wish to test only specific functionalities, for example

```
tox -e lint  # code style
tox -e build  # packaging
tox -e tests  # testing
tox -e docs  # build the documentation
```

You can also run only a subset of the tests with `tox -e tests -- <tests/file.py>`, replacing `<tests/file.py>` with the path to the files you want to test, e.g. `tox -e tests -- tests/test_main.py` for testing only the main functions. For more details take a look at the *usage and invocation <https://docs.pytest.org/en/latest/usage.html#usage-and-invocations>* page of the pytest documentation.

You can also use `tox -e format` to use tox to do actual formatting instead of just testing it. Also, you may want to setup your editor to automatically apply the black code formatter when saving your files, there are plugins to do this with all major editors.

Contributing to the documentation

Local documentation

The documentation of MAICoS is written in reStructuredText (rst) and uses the Sphinx documentation generator. You can build the documentation from the `maicos/docs` folder:

```
tox -e docs
```

Then, visualize the local documentation with your favorite internet explorer (here Mozilla Firefox is used)

```
firefox dist/docs/index.html
```

Structure

Most of the content of the documentation is written in `.rst` files located within `docs/src/`. The content in the *Reference guides* section is directly generated from the documentation string of the source code located in `src/maicos` thanks to Sphinx and Autodoc.

After creating a new module, add it to the documentation by modifying the *toctree* in the `docs/src/references/modules/index.rst` file, and adding a new .rst file with the following format:

```
.. _ModuleName:

ModuleName
##########

```

(continues on next page)

(continued from previous page)

```
.. _label_module_name:

.. autoclass:: maicos.ModuleName
    :members:
    :undoc-members:
    :show-inheritance:
```

Note that all files located within `docs/src/examples` are generated from the Python scrips located in `examples` using Sphinx-Gallery.

Version information

The version information in *maicos.__version__* indicates the release of MAICoS using semantic versioning.

In brief:

Given a version number MAJOR.MINOR.PATCH, we increment the

1. **MAJOR** version when we make **incompatible API changes**,
2. **MINOR** version when we **add functionality** in a **backwards-compatible** manner, and
3. **PATCH** version when we make backwards-compatible **bug fixes**.

However, as long as the **MAJOR** number is **0** (i.e. the API has not stabilized), even **MINOR** increases *may* introduce incompatible API changes. As soon as we have a 1.0.0 release, the public API can only be changed in a backward-incompatible manner with an increase in MAJOR version.

Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format, following **PEP 440**.

Note: Development versions and pre-releases have a suffix after the release number, such as `0.7.0+12.gFEED2BEEF`. If you have problems, try out a full release (e.g. `0.7.0`) first.

Data

`maicos.__version__ = '0.8+23.g3ff25bb'`

Version information for MAICoS, following **PEP 440** and semantic versioning.

Release workflow

Versioneer (optional)

1. Upgrade versioneer if a new version is available.
2. Check the upgrade notes if additional steps are required
3. Upgrade versioneer

   ```
   pip3 install --upgrade versioneer
   ```

4. Remove the old versioneer.py file

```
rm versioneer.py
```

5. Install new versioneer.py file

```
python3 -m versioneer install --vendor
```

   Revert the changes in *src/maicos/__init__.py*

6. Commit changes

Create release

1. Make sure changelog is up to date and add release date and commit your changes

```
git commit -m 'Release vX.X'
```

2. Tag commit with the new version

```
git tag -m 'Release vX.X' vX.X
```

3. Test locally!!!

```
git describe
```

   and

```
pip3 install .
```

   should result in `vX.X`

4. If a new major release create release branch

```
git branch release-X-X
```

   otherwise update the release branch with respect to the branch

```
git checkout release-X-X
git rebase -i main
```

5. Push branch, tag

```
git push release-X-X
git push --tags
```

6. Go to the web interface, add changelog as release message

After the release

- Bump version (Create new section in CHANGELOG.rst)

5.6 Contributors

Thank you to all of the developers, programmers, and researchers who contributed to the creation of MAICoS.

We also thank the Institute for Computational Physics (University of Stuttgart), group members of Roland Netz at Freie Universität Berlin, and the Stuttgart Center for Simulation Science and the German Research Council (DFG) for funding through the Cluster of Excellence EXC 2075 "Data-integrated Simulation Science".

5.6.1 History

MAICoS was first developed in Roland Netz's group at the Freie University of Berlin by Alexander Schlaich and Philip Loche, and is now mostly developed and maintained at the *Institute for Computational Physics*.

5.6.2 Maintainers

- Philip Loche
- Alexander Schlaich
- Henrik Stooß

5.6.3 Developers

- Maximilian Becker
- Simon Gravelle
- Philipp Stärk
- Srihas Velpuri

5.6.4 Contributors

- Adyant Agrawal
- Shane Carlson
- Kira Fischer
- Federico Grasselli
- Julian Kappler
- Marc Sauter
- Laura Scalfi
- Julius Schulz
- Dominik Wille
- Amanuel Wolde-Kidan

PYTHON MODULE INDEX

m

INDEX

Symbols

A

B

C

D

E

F

G

I

K

L

M

N

O

P

Q

R

S

T

U

V

Z