MAICoS Documentation

Release 0.4

see the file AUTHORS for the full list of names

Sep 23, 2022

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.

This is a simple example showing how to use MAICoS to extract the density profile from a molecular dynamics simulation. The files `conf.gro` and `traj.trr` correspond to a water slab in vacuum that was simulated in this case using the GROMACS simulation package. In a Python environment, type:

```
import MDAnalysis as mda
import maicos
u = mda.Universe('conf.gro', 'traj.trr')
grpH2O = u.select_atoms('type O or type H')
dplan = maicos.density_planar(grpH2O)
dplan.run()
```

Results can be accessed from `dplan.results`.

CHAPTER

ONE

INSTALLATION

Python3 and a C-compiler are needed to build the underlying libraries.

1.1 Using pip

If you have root access, install the package for all users by typing in a terminal:

```
pip3 install numpy
pip3 install maicos
```

Alternatively, if you don't have special privileges, install the package in your home directory by using the `--user` flag.

CHAPTER

TWO

LIST OF ANALYSIS MODULES

Module Name	Description
density_planar	Compute partial densities/temperature profiles in the Cartesian systems.
density_cylinder	Compute partial densities across a cylinder.
epsilon_bulk	Compute dipole moment fluctuations and static dielectric constant.
epsilon_planar	Calculates a planar dielectric profile.
epsilon_cylinder	Calculate cylindrical dielectric profiles.
dielectric_spectrum	Computes the linear dielectric spectrum.
saxs	Compute SAXS scattering intensities.
diporder	Calculation of dipolar order parameters.
debyer	Calculate scattering intensities using the debye equation. The debyer library needs to be downloaded and build.
dipole_angle	Calculate angle timeseries of dipole moments with respect to an axis.
kinetic_energy	Calculate the timeseries of energies.
velocity	Mean velocity analysis.

2.1 Installation

Python3 and a C-compiler are needed to build the underlying libraries.

2.1.1 Using pip

If you have root access, install the package for all users by typing in a terminal:

```
pip3 install numpy
pip3 install maicos
```

Alternatively, if you don't have special privileges, install the package in your home directory by using the `--user` flag:

```
pip3 install --user numpy
pip3 install --user maicos
```

2.1.1.1 Bash autocompletion

You can include MAICoS to BASH suggestions by oppening your `.bashrc` or `.profile` file with your favorite text editor (here vim is used):

```
vim ~/.bashrc
```

and by adding

```
source $(maicos --bash_completion)
```

2.1.2 Development version

The development version of MAICoS can be compiled from source. NumPy and Cython are required:

```
pip3 install numpy
pip3 install cython
```

Then type in a terminal:

```
git clone git@gitlab.com:maicos-devel/maicos.git
pip3 install -e maicos/
```

2.1.2.1 Testing

You can run the tests from the `maicos/tests/` directory. The tests rely on the pytest library, and use some work flows from NumPy and MDAnalysisTests. In a terminal, type:

```
pip3 install MDAnalysisTests
```

Then, type:

```
cd maicos/tests
pytest  --disable-pytest-warnings
```

2.2 Usage

2.2.1 From the command line

MAICoS can be used directly from the command line, by typing in a terminal:

```
maicos <module> <paramaters>
```

You can get the general help page, or a package-specific page by typing, respectively:

```
maicos -h

maicos <package> -h
```

For example, to get the help page for the `density_planar` module, type:

```
maicos density_planar -h
```

2.2.2 From the Python interpreter

MAICoS can be used within the python interpreter. In a python environment, create an `analysis` object by supplying an atom group from MDAnalysis as well as some (optional) parameters, then use the `run` method:

```
import maicos

ana_obj = maicos.<module>(atomgroup, <paramaters>)
ana_obj.run()
```

Results are available through the objects *results* dictionary.

2.3 Getting involved

Contribution via pull requests are always welcome. Source code is available from GitLab. Before submitting a pull request, please open an issue to discuss your changes. Use the main feature branch *develop* for submitting your requests. The master branch contains all commits of the latest release. More information on the branching model we used is given in this nice post blog.

2.3.1 Testing

You can run the tests from the `maicos/tests/` directory. The tests rely on the **`pytest`_** library, and use some work flows from NumPy and **`MDAnalysisTests`_**. In a terminal, type:

```
pip3 install MDAnalysisTests
```

Then, type:

```
cd maicos/tests
pytest  --disable-pytest-warnings
```

2.3.2 Writing your own analysis module

Example code for an analysis module can be found in the example folder. To deploy the script, follow the steps in examples/README.md.

We use yapf using the NumPy formatting style for our code. You can style your code from the command line or using an extension for your favorite editor. The easiest use is to install the git hook module, which will automatically format your code before committing. To install it just run the `enable_githooks.sh` from the command line. Currently, we only format python files.

MAICoS' unit testing relies on the pytest library and use some work flows from numpy and MDAnalysisTests. In order to run the tests you need those packages. To start the test process, simply type from the root of the repository

```
cd test
pytest  --disable-pytest-warnings
```

Whenever you add a new feature to the code you should also add a test case. Furthermore test cases are also useful if a bug is fixed or anything you think worthwhile. Follow the philosophy - the more the better!

2.3.3 Contributing to the documentation

The documentation of MAICoS is written in reStructuredText (rst) and uses sphinx documentation generator. In order to modify the documentation, first create a local version on your machine. Go to the MAICoS develop project page and hit the `Fork` button, then clone your forked branch to your machine:

```
git clone git@gitlab.com:your-user-name/maicos.git
```

Then, build the documentation from the `maicos/docs` folder:

```
cd maicos/docs/
make html
```

Then, still from the `maicos/docs/` folder, visualise the local documentation with your favourite internet explorer (here Mozilla Firefox is used)

```
firefox build/html/index.html
```

Each MAICoS module contains a documentation string, or docstring. Docstrings are processed by Sphinx and autodoc to generate the documentation. If you created a new module with a doctring, you can add it to the documentation by modifying the *toctree* in the `index.rst` file.

2.4 Authors list

2.4.1 Maintainer

- Philip Loche

2.4.2 Authors

- Alexander Schlaich
- Philip Loche

2.4.3 Contributors

- Maximilian Becker
- Shane Carlson
- Julian Kappler
- Julius Schulz
- Dominik Wille
- Amanuel Wolde-Kidan
- Philipp Stärk
- Simon Gravelle

- Henrik Jaeger
- Srihas Velpuri

2.5 Changelog

2.5.1 v0.4 (2021/12/13)

Philip Loche, Simon Gravelle, Philipp Staerk, Henrik Jaeger, 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.

2.5.2 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

2.5.3 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

2.5.4 v0.1 (2019/10/30)

Philip Loche

- first release out of the lab

2.6 Density modules

2.6.1 Density planar

Compute partial densities/temperature profiles in the Cartesian systems. **Tutorial**

To follow this tutorial, the data test files of MAICoS are needed. From a terminal, download MAICoS at a location of your choice:

```
cd mypath
git clone git@gitlab.com:maicos-devel/maicos.git
```

In a python environment, import MDAnalysis, MAICoS, PyPlot, and NumPy:

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

Define the path to the `airwater` data folder of MAICoS:

```
datapath = 'mypath/maicos/tests/data/airwater/'
```

The system consists of a 2D slab with 352 water molecules in vacuum, where the two water/vacuum interfaces are normal to the axis z:

documentation_pages/images/airwater.png

Create a universe using MDAnalysis and define a group containing the oxygen and the hydrogen atoms of the water molecules:

```
u = mda.Universe(datapath+'conf.gro', datapath+'traj.trr')
grpH2O = u.select_atoms('type O or type H')
```

Let us call the density_planar module:

```
dplan = maicos.density_planar(grpH2O)
dplan.run()
```

Extract the coordinate and the density profile:

```
zcoor = dplan.results['z']
dens = dplan.results['dens_mean']
```

By default the binwidth is 0.1 nanometers, the units are kg/m3, and the axis is z. Plot it using

```
fig = plt.figure(figsize = (12,6))
plt.plot(zcoor,dens,linewidth=2)
plt.xlabel("z coordinate [nanometer]")
plt.ylabel("density H2O [kg/m3]")
plt.show()
```

They are several options you can play with. To know the full list of options, have a look at the `Inputs` section below. For instance, you can increase the spacial resolution by reducing the binwidth:

```
dplan = maicos.density_planar(grp_oxy, binwidth = 0.05)
```

Inputs :param output (str): Output filename :param outfreq (int): Default time after which output files are refreshed (1000 ps). :param dim (int): Dimension for binning (0=X, 1=Y, 2=Z) :param binwidth (float): binwidth (nanometer) :param mu (bool): Calculate the chemical potential (sets dens='number') :param muout (str): Prefix for output filename

for chemical potential :param temperature (float): temperature (K) for chemical potential (Default: 300K) :param mass (float): Mass (u) for the chemical potential. By default taken from topology. :param zpos (float): position (nm) at which the chemical potential will be computed. By default average over box. :param dens (str): Density: mass, number, charge, temperature. (Default: mass) :param comgroup (str): Perform the binning relative to the center of mass of the selected group. :param center (bool): Perform the binning relative to the center of the (changing) box.

Outputs :param dim (int): Dimension for binning (0=X, 1=Y, 2=Z) :param binwidth (float): binwidth (nanometer) :param comgroup (str): Perform the binning relative to the center of mass of the selected group.With *comgroup* the *center* option is also used. :param center (bool): Perform the binning relative to the center of the (changing) box.

returns (dict)

- z: bins
- dens_mean: calculated densities
- dens_err: density error
- mu: chemical potential
- dmu: error of chemical potential

2.6.2 Density cylinder

Compute partial densities across a cylinder.

Inputs

param output (str)
Output filename

param outfreq (int)
Default time after which output files are refreshed (1000 ps).

param dim (int)
Dimension for binning (0=X, 1=Y, 2=Z)

param center (str)
Perform the binning relative to the center of this selection string of teh first AtomGroup. If None center of box is used.

param radius (float)
Radius of the cylinder (nm). If None smallest box extension is taken.

param binwidth (float)
binwidth (nanometer)

param length (float)
Length of the cylinder (nm). If None length of box in the binning dimension is taken.

param dens (str)
Density: mass, number, charge, temp

Outputs

returns (dict)

- z: bins
- dens_mean: calculated densities

- dens_err: density error

Tutorial

To follow this tutorial, the data test files of MAICoS are needed. From a terminal, download MAICoS at a location of your choice:

```
cd mypath
git clone git@gitlab.com:maicos-devel/maicos.git
```

In a python environment, import MDAnalysis, MAICoS, and PyPlot:

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

Define the path to the `cntwater` data folder of MAICoS:

```
datapath = 'mypath/maicos/tests/data/cntwater/'
```

The system consists of a carbon nanotube (CNT) with axis in the z: direction, a radius of about 2 nm, a of length 2.2 nm, and filled with 810 water molecules.

Create a universe using MDAnalysis and define two groups, one containing the water molecules, one containing the carbon atoms:

```
u = mda.Universe(datapath + 'lammps.data', datapath + 'traj.xtc')
grpH2O = u.select_atoms('type 1 or type 2')
grpCNT = u.select_atoms('type 3')
```

Call the `density_cylinder` module for the two groups:

```
dcylH2O = maicos.density_cylinder(grpH2O, center='all', binwidth = 0.01)
dcylH2O.run()
dcylCNT = maicos.density_cylinder(grpCNT, center='all', binwidth = 0.01)
dcylCNT.run()
```

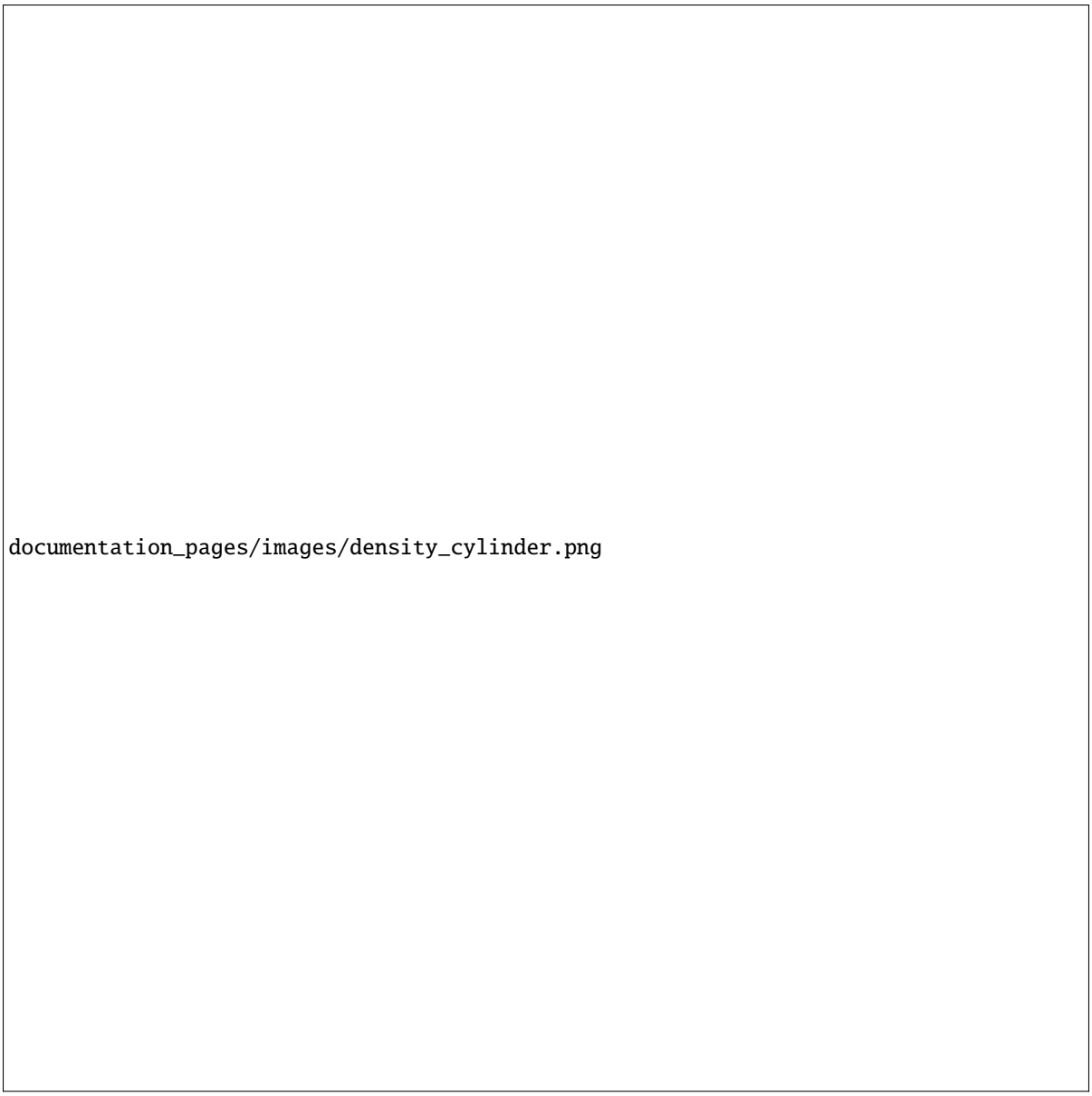
With the keyword `center='all'`, the center of mass of all the atoms of the group is used as the center of the density profile. If not specified, the center of the box is used.

Finally, extract the coordinates and the density profiles:

```
rcoor = dcylH2O.results['r']
densH2O = dcylH2O.results['dens_mean']
densCNT = dcylCNT.results['dens_mean']
```

Plot it using PyPlot:

```
fig = plt.figure(figsize = (12,6))
plt.plot(rcoor,densH2O,linewidth=2)
plt.plot(rcoor,densCNT,linewidth=2)
plt.xlabel("r coordinate [nanometer]")
plt.ylabel("density [kg/m3]")
plt.show()
```



2.7 Dielectric constant modules

2.7.1 Epsilon bulk

2.7.1.1 Description

Compute dipole moment fluctuations and static dielectric constant.

Inputs

param outfreq (float)
Number of frames after which the output is updated.

param output (str)
Output filename.

param temperature (float)
temperature (K)

param bpbc (bool)
do not make broken molecules whole again (only works if molecule is smaller than shortest

box vector

Outputs

returns (dict)

- **M: Directional dependant dipole moment**
  $\langle \boldsymbol{M} \rangle$ in e.
- **M2: Directional dependant squared dipole moment**
  $\langle \boldsymbol{M}^2 \rangle$ in $(e)^2$
- **fluct: Directional dependant dipole moment fluctuation**
  $\langle \boldsymbol{M}^2 \rangle - \langle \boldsymbol{M} \rangle^2$ in $(e)^2$
- eps: Directional dependant static dielectric constant
- eps_mean: Static dielectric constant

2.7.2 Epsilon planar

2.7.2.1 Description

Calculate planar dielectric profiles. See Schlaich, et al., Phys. Rev. Lett., vol. 117 (2016) for details

Inputs :param output_prefix (str): Prefix for output files :param zmin (float): minimal coordinate for evaluation (nm) :param zmax (float): maximal coordinate for evaluation (nm) :param temperature (float): temperature (K) :param outfreq (int): Default number of frames after which output files are refreshed. :param b2d (bool): Use 2d slab geometry :param vac (bool): Use vacuum boundary conditions instead of metallic (2D only!). :param bsym (bool): symmetrize the profiles :param bpbc (bool): Do not make broken molecules whole again (only works if

molecule is smaller than shortest box vector

Outputs :param dim (int): Dimension for binning (0=X, 1=Y, 2=Z) :param binwidth (float): binwidth (nanometer) :param comgroup (str): Perform the binning relative to the center of mass of the selected group.With *comgroup* the *center* option is also used. :param center (bool): Perform the binning relative to the center of the (changing) box.

returns (dict)

- z: Bin positions
- eps_par: Parallel dielectric profile (_ - 1)
- deps_par: Error of parallel dielectric profile
- eps_par_self: Self contribution of parallel dielectric profile
- eps_par_coll: Collective contribution of parallel dielectric profile
- eps_perp: Inverse perpendicular dielectric profile (^{-1}_ - 1)
- deps_perp: Error of inverse perpendicular dielectric profile

- eps_par_self: Self contribution of Inverse perpendicular dielectric profile
- eps_perp_coll: Collective contribution of Inverse perpendicular dielectric profile

2.7.3 Epsilon cylinder

2.7.3.1 Description

Calculate cylindrical dielectric profiles.

Components are calculated along the axial (z) and radial (along xy) direction at the system's center of mass.

Inputs

param output_prefix (str)
Prefix for output_prefix files

param geometry (str)
A structure file without water from which com is calculated.

param radius (float)
Radius of the cylinder (nm)

param binwidth (float)
Bindiwdth the binwidth (nm)

param variable_dr (bool)
Use a variable binwidth, where the volume is kept fixed.

param length (float)
Length of the cylinder (nm)

param outfreq (int)
Default number of frames after which output files are refreshed.

param temperature (float)
temperature (K)

param single (bool)
"1D" line of watermolecules

param bpbc (bool)
Do not make broken molecules whole again (only works if molecule is smaller than shortest box vector

Outputs

returns (dict)

- r: Bin positions
- eps_ax: Parallel dielectric profile (_)
- deps_ax: Error of parallel dielectric profile
- eps_rad: Inverse perpendicular dielectric profile (^{-1}_)
- deps_rad: Error of inverse perpendicular dielectric profile

2.7.4 Dielectric spectrum

2.7.4.1 Description

Computes the linear dielectric spectrum.

This module, given molecular dynamics trajectory data, produces a .txt file containing the complex dielectric function as a function of the (linear, not radial - i.e. nu or f, rather than omega) frequency, along with the associated standard deviations. The algorithm is based on the Fluctuation Dissipation Relation (FDR): chi(f) = -1/(3 V k_B T epsilon_0) FT{theta(t) <P(0) dP(t)/dt>}. By default, the polarization trajectory, time series array and the average system volume are saved in the working directory, and the data are reloaded from these files if they are present. Lin-log and log-log plots of the susceptibility are also produced by default.

Inputs

param recalc (bool)
Forces to recalculate the polarization, regardless if it is already present.

param temperature (float)
Reference temperature.

param output_prefix (str)
Prefix for the output files.

param segs (int)
Sets the number of segments the trajectory is broken into.

param df (float)
The desired frequency spacing in THz. This determines the minimum frequency about which there is data. Overrides -segs option.

param noplots (bool)
Prevents plots from being generated.

param plotformat (str)
Allows the user to choose the format of generated plots (choose from 'pdf', 'png', 'jpg', 'eps')

param ymin (float)
Manually sets the minimum lower bound for the log-log plot.

param 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.

param binafter (int)
The number of low-frequency data points that are left unbinned.

param nobin (bool)
Prevents the data from being binned altogether. This can result in very large plot files and errors.

Outputs :returns (dict): TODO

2.8 Structure modules

2.8.1 Saxs

2.8.1.1 Description

2.8.2 Diporder

2.8.2.1 Description

2.8.3 Debyer

2.8.3.1 Description

2.9 Timeseries modules

2.9.1 Dipole angle

2.9.1.1 Description

2.9.2 Kinetic energy

2.9.2.1 Description

2.10 Transport modules

2.10.1 Velocity

2.10.1.1 Description

PYTHON MODULE INDEX

m

INDEX

M