MAICoS Documentation

Release 0.5

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.

CHAPTER

ONE

BASIC EXAMPLE

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.DensityPlanar(grpH2O)
dplan.run()
```

Results can be accessed from `dplan.results`.

CHAPTER

TWO

INSTALLATION

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

2.1 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

THREE

LIST OF ANALYSIS MODULES

Module Name	Description
DensityPlanar	Compute partial densities/temperature profiles in the Cartesian systems.
DensityCylinder	Compute partial densities across a cylinder.
EpsilonBulk	Compute dipole moment fluctuations and static dielectric constant.
EpsilonPlanar	Calculate planar dielectric profiles.
EpsilonCylinder	Calculate cylindrical dielectric profiles.
DielectricSpectrum	Compute the linear dielectric spectrum.
Saxs	Compute SAXS scattering intensities.
Diporder	Calculate dipolar order parameters.
Debyer	Calculate scattering intensities using the debye equation. The debyer library needs to be downloaded and build.
DipoleAngle	Calculate angle timeseries of dipole moments with respect to an axis.
KineticEnergy	Calculate the timeseries of energies.
Velocity	Analyse mean velocity.

3.1 Installation

Python3 and a C-compiler are needed to build the underlying libraries. Install the package using pip with:

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

Alternatively, if you don't have special privileges, install the package only for the current using the `--user` flag:

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

3.2 Usage

3.2.1 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 `maicos.DensityPlanar` module, type:

```
maicos densityplanar -h
```

3.2.2 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.

3.3 Tutorials

This section contains self-contained examples of using MAICoS for various tasks. The examples are structured in the form of Jupyter notebooks, rendered for viewing here and available for interactive execution in the top-level `/docs/source/tutorials` directory of the repository.

3.3.1 Density planar tutorial

To follow this tutorial, the data test files `airwater` of MAICoS are needed. You can obtain it by cloning MAICoS repository:

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

The `airwater` data files are located in `tests/data/airwater/`. First, let us ignore unnecessary warnings:

[1]:
```
import warnings
warnings.filterwarnings("ignore")
```

First, import MAICoS, NumPy, MDAnalysis, and PyPlot:

[2]:
```
import maicos
import numpy as np
import MDAnalysis as mda
import matplotlib.pyplot as plt
from matplotlib.ticker import AutoMinorLocator
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
Cell In [2], line 1
----> 1 import maicos
```

(continues on next page)

(continued from previous page)

```
      2 import numpy as np
      3 import MDAnalysis as mda

File ~/checkouts/readthedocs.org/user_builds/maicos/envs/v0.5/lib/python3.10/site-
↪packages/maicos/__init__.py:36
     29 from .modules.density import DensityPlanar, DensityCylinder
     30 from .modules.epsilon import (
     31     EpsilonBulk,
     32     EpsilonPlanar,
     33     EpsilonCylinder,
     34     DielectricSpectrum
     35 )
---> 36 from .modules.structure import Saxs, Debye, Diporder
     37 from .modules.timeseries import DipoleAngle, KineticEnergy
     38 from .modules.transport import Velocity

File ~/checkouts/readthedocs.org/user_builds/maicos/envs/v0.5/lib/python3.10/site-
↪packages/maicos/modules/structure.py:24
     18 from .. import tables
     19 from ..decorators import (
     20     make_whole,
     21     set_planar_class_doc,
     22     set_verbose_doc,
     23 )
---> 24 from ..lib import sfactor
     25 from ..utils import check_compound, savetxt
     27 logger = logging.getLogger(__name__)

ImportError: dynamic module does not define module export function (PyInit_sfactor)
```

and set a few parameters for plotting purpose:

[3]:
```
fontsize = 25
font = {'family': 'sans', 'color':  'black',
        'weight': 'normal', 'size': fontsize}
my_color_1 = np.array([0.090, 0.247, 0.560])
my_color_2 = np.array([0.235, 0.682, 0.639])
my_color_3 = np.array([1.000, 0.509, 0.333])
my_color_4 = np.array([0.588, 0.588, 0.588])
```

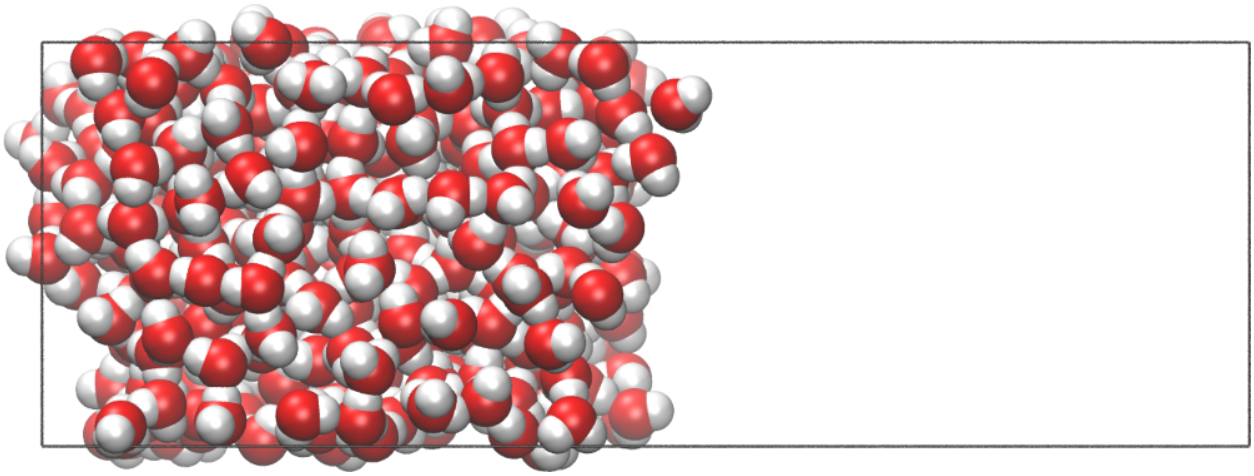
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [3], line 4
      1 fontsize = 25
      2 font = {'family': 'sans', 'color':  'black',
      3         'weight': 'normal', 'size': fontsize}
----> 4 my_color_1 = np.array([0.090, 0.247, 0.560])
      5 my_color_2 = np.array([0.235, 0.682, 0.639])
      6 my_color_3 = np.array([1.000, 0.509, 0.333])

NameError: name 'np' is not defined
```

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

[4]:
```
datapath = "../../../../tests/data/airwater/"
```

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



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

[5]:
```
u = mda.Universe(datapath+'conf.gro',
                 datapath+'traj.trr')
group_H2O = u.select_atoms('type O or type H')
group_O = u.select_atoms('type O')
group_H = u.select_atoms('type H')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [5], line 1
----> 1 u = mda.Universe(datapath+'conf.gro',
      2                  datapath+'traj.trr')
      3 group_H2O = u.select_atoms('type O or type H')
      4 group_O = u.select_atoms('type O')

NameError: name 'mda' is not defined
```

Let us print a few information about the trajectory file:

[6]:
```
print(f"The number of water molecules is {group_O.n_atoms}")
timestep = np.round(u.trajectory.dt,2)
print(f"The time interval between the frames is {timestep} ps")
total_time = np.round(u.trajectory.totaltime,2)
print(f"The total simulation time is {total_time} ps")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [6], line 1
----> 1 print(f"The number of water molecules is {group_O.n_atoms}")
      2 timestep = np.round(u.trajectory.dt,2)
      3 print(f"The time interval between the frames is {timestep} ps")

NameError: name 'group_O' is not defined
```

Let us use the `DensityPlanar` class of MAICos using the `group_H2O` group:

[7]:
```
dplan = maicos.DensityPlanar(group_H2O)
dplan.run()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [7], line 1
----> 1 dplan = maicos.DensityPlanar(group_H2O)
      2 dplan.run()

NameError: name 'maicos' is not defined
```

Extract the coordinate and the density profile from the `results` attribute:

[8]:
```
zcoor = dplan.results['z']
dens = dplan.results['dens_mean']
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [8], line 1
----> 1 zcoor = dplan.results['z']
      2 dens = dplan.results['dens_mean']

NameError: name 'dplan' is not defined
```

By default the binwidth is 0.1 nanometers, the units are kg/m^3, and the axis is z. Plot the density profile using :

[9]:
```
fig = plt.figure(figsize=(13,6.5))
ax1 = plt.subplot(1, 1, 1)
plt.xlabel("z coordinate (nm)", fontdict=font)
plt.ylabel(r"density H2O (kg/m$^3$]", fontdict=font)
plt.xticks(fontsize=fontsize)
plt.yticks(fontsize=fontsize)
ax1.plot(zcoor, dens, color=my_color_1, linewidth=4)
ax1.yaxis.offsetText.set_fontsize(20)
ax1.minorticks_on()
ax1.tick_params('both', length=10, width=2, which='major', direction='in')
ax1.tick_params('both', length=6, width=1.4, which='minor', direction='in')
ax1.xaxis.set_ticks_position('both')
ax1.yaxis.set_ticks_position('both')
ax1.spines["top"].set_linewidth(2)
ax1.spines["bottom"].set_linewidth(2)
ax1.spines["left"].set_linewidth(2)
ax1.spines["right"].set_linewidth(2)
ax1.yaxis.offsetText.set_fontsize(30)
minor_locator_y = AutoMinorLocator(2)
ax1.yaxis.set_minor_locator(minor_locator_y)
minor_locator_x = AutoMinorLocator(2)
ax1.xaxis.set_minor_locator(minor_locator_x)
ax1.tick_params(axis='x', pad=10)
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
```

(continues on next page)

(continued from previous page)

```
Cell In [9], line 1
----> 1 fig = plt.figure(figsize=(13,6.5))
      2 ax1 = plt.subplot(1, 1, 1)
      3 plt.xlabel("z coordinate (nm)", fontdict=font)

NameError: name 'plt' is not defined
```

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

[10]:
```
dplan_smaller_bin = maicos.DensityPlanar(group_H2O, binwidth = 0.05)
dplan_smaller_bin.run()
zcoor_smaller_bin = dplan_smaller_bin.results['z']
dens_smaller_bin = dplan_smaller_bin.results['dens_mean']
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [10], line 1
----> 1 dplan_smaller_bin = maicos.DensityPlanar(group_H2O, binwidth = 0.05)
      2 dplan_smaller_bin.run()
      3 zcoor_smaller_bin = dplan_smaller_bin.results['z']

NameError: name 'maicos' is not defined
```

[11]:
```
fig = plt.figure(figsize=(13,6.5))
ax1 = plt.subplot(1, 1, 1)
plt.xlabel(r"$z$ coordinate (nm)", fontdict=font)
plt.ylabel(r"density H2O (kg/m$^3$]", fontdict=font)
plt.xticks(fontsize=fontsize)
plt.yticks(fontsize=fontsize)
ax1.plot(zcoor_smaller_bin, dens_smaller_bin, color=my_color_2, linewidth=4)
ax1.plot(zcoor, dens, color=my_color_1, linewidth=4)
ax1.yaxis.offsetText.set_fontsize(20)
ax1.minorticks_on()
ax1.tick_params('both', length=10, width=2, which='major', direction='in')
ax1.tick_params('both', length=6, width=1.4, which='minor', direction='in')
ax1.xaxis.set_ticks_position('both')
ax1.yaxis.set_ticks_position('both')
ax1.spines["top"].set_linewidth(2)
ax1.spines["bottom"].set_linewidth(2)
ax1.spines["left"].set_linewidth(2)
ax1.spines["right"].set_linewidth(2)
ax1.yaxis.offsetText.set_fontsize(30)
minor_locator_y = AutoMinorLocator(2)
ax1.yaxis.set_minor_locator(minor_locator_y)
minor_locator_x = AutoMinorLocator(2)
ax1.xaxis.set_minor_locator(minor_locator_x)
ax1.tick_params(axis='x', pad=10)
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [11], line 1
```

(continues on next page)

(continued from previous page)

```
----> 1 fig = plt.figure(figsize=(13,6.5))
      2 ax1 = plt.subplot(1, 1, 1)
      3 plt.xlabel(r"$z$ coordinate (nm)", fontdict=font)

NameError: name 'plt' is not defined
```

Note : less noisy profile can be obtained by running longer simulations.

MAICoS can deal with several groups at once:

[12]:
```
dplan_separate = maicos.DensityPlanar([group_O, group_H], binwidth = 0.05)
dplan_separate.run()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [12], line 1
----> 1 dplan_separate = maicos.DensityPlanar([group_O, group_H], binwidth = 0.05)
      2 dplan_separate.run()

NameError: name 'maicos' is not defined
```

In this case, respective results for each group are returned:

[13]:
```
zcoor_separate = dplan_smaller_bin.results['z']
dens_oxygen = dplan_separate.results['dens_mean'].T[0]
dens_hygrogen = dplan_separate.results['dens_mean'].T[1]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [13], line 1
----> 1 zcoor_separate = dplan_smaller_bin.results['z']
      2 dens_oxygen = dplan_separate.results['dens_mean'].T[0]
      3 dens_hygrogen = dplan_separate.results['dens_mean'].T[1]

NameError: name 'dplan_smaller_bin' is not defined
```

[14]:
```
fig = plt.figure(figsize=(13,6.5))
ax1 = plt.subplot(1, 1, 1)
plt.xlabel(r"$z$ coordinate (nm)", fontdict=font)
plt.ylabel(r"density H2O (kg/m$^3$]", fontdict=font)
plt.xticks(fontsize=fontsize)
plt.yticks(fontsize=fontsize)
ax1.plot(zcoor_separate, dens_hygrogen, color=my_color_4, linewidth=4)
ax1.plot(zcoor_separate, dens_oxygen, color=my_color_3, linewidth=4)
ax1.yaxis.offsetText.set_fontsize(20)
ax1.minorticks_on()
ax1.tick_params('both', length=10, width=2, which='major', direction='in')
ax1.tick_params('both', length=6, width=1.4, which='minor', direction='in')
ax1.xaxis.set_ticks_position('both')
ax1.yaxis.set_ticks_position('both')
ax1.spines["top"].set_linewidth(2)
ax1.spines["bottom"].set_linewidth(2)
ax1.spines["left"].set_linewidth(2)
ax1.spines["right"].set_linewidth(2)
```

(continues on next page)

(continued from previous page)

```
ax1.yaxis.offsetText.set_fontsize(30)
minor_locator_y = AutoMinorLocator(2)
ax1.yaxis.set_minor_locator(minor_locator_y)
minor_locator_x = AutoMinorLocator(2)
ax1.xaxis.set_minor_locator(minor_locator_x)
ax1.tick_params(axis='x', pad=10)
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [14], line 1
----> 1 fig = plt.figure(figsize=(13,6.5))
      2 ax1 = plt.subplot(1, 1, 1)
      3 plt.xlabel(r"$z$ coordinate (nm)", fontdict=font)

NameError: name 'plt' is not defined
```

3.3.2 Density cylinder tutorial

To follow this tutorial, the data test files `cntwater` of MAICoS are needed. You can obtain it by cloning MAICoS repository:

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

The `airwater` data files are located in `tests/data/airwater/`. First, let us ignore unnecessary warnings:

[1]:
```
import warnings
warnings.filterwarnings("ignore")
```

First, import MAICoS, NumPy, MDAnalysis, and PyPlot:

[2]:
```
import maicos
import numpy as np
import MDAnalysis as mda
import matplotlib.pyplot as plt
from matplotlib.ticker import AutoMinorLocator
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
Cell In [2], line 1
----> 1 import maicos
      2 import numpy as np
      3 import MDAnalysis as mda

File ~/checkouts/readthedocs.org/user_builds/maicos/envs/v0.5/lib/python3.10/site-
↪packages/maicos/__init__.py:36
     29 from .modules.density import DensityPlanar, DensityCylinder
     30 from .modules.epsilon import (
     31     EpsilonBulk,
     32     EpsilonPlanar,
     33     EpsilonCylinder,
     34     DielectricSpectrum
```

(continues on next page)

(continued from previous page)

```
     35 )
---> 36 from .modules.structure import Saxs, Debye, Diporder
     37 from .modules.timeseries import DipoleAngle, KineticEnergy
     38 from .modules.transport import Velocity

File ~/checkouts/readthedocs.org/user_builds/maicos/envs/v0.5/lib/python3.10/site-
↪packages/maicos/modules/structure.py:24
     18 from .. import tables
     19 from ..decorators import (
     20     make_whole,
     21     set_planar_class_doc,
     22     set_verbose_doc,
     23 )
---> 24 from ..lib import sfactor
     25 from ..utils import check_compound, savetxt
     27 logger = logging.getLogger(__name__)

ImportError: dynamic module does not define module export function (PyInit_sfactor)
```

and set a few parameters for plotting purpose:

[3]:
```
fontsize = 25
font = {'family': 'sans', 'color':  'black',
        'weight': 'normal', 'size': fontsize}
my_color_1 = np.array([0.090, 0.247, 0.560])
my_color_2 = np.array([0.235, 0.682, 0.639])
my_color_3 = np.array([1.000, 0.509, 0.333])
my_color_4 = np.array([0.588, 0.588, 0.588])
```

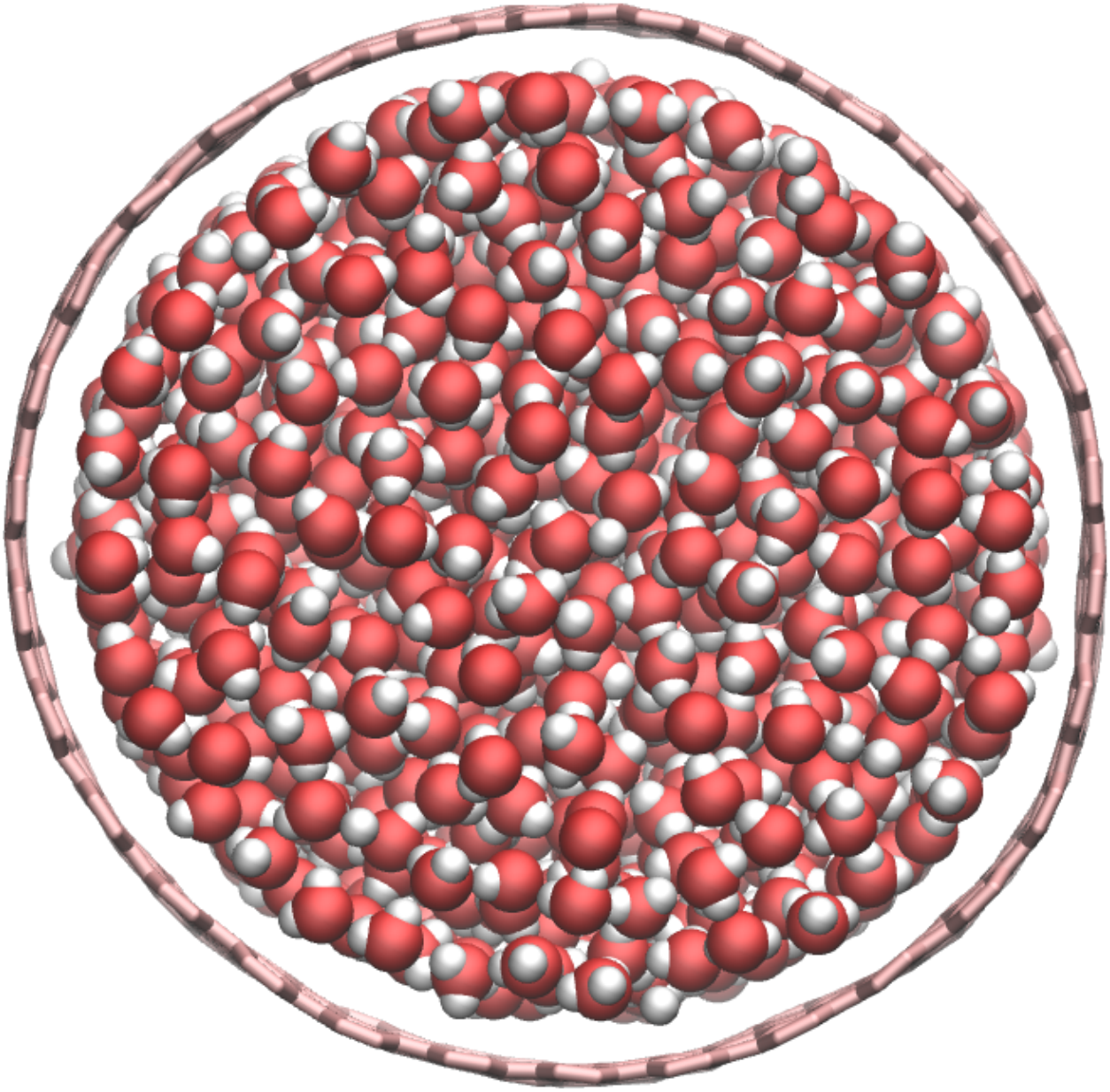
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [3], line 4
      1 fontsize = 25
      2 font = {'family': 'sans', 'color':  'black',
      3         'weight': 'normal', 'size': fontsize}
----> 4 my_color_1 = np.array([0.090, 0.247, 0.560])
      5 my_color_2 = np.array([0.235, 0.682, 0.639])
      6 my_color_3 = np.array([1.000, 0.509, 0.333])

NameError: name 'np' is not defined
```

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

[4]:
```
datapath = "../../../../tests/data/cntwater/"
```

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



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

[5]:
```
u = mda.Universe(datapath + 'lammps.data',
                 datapath + 'traj.xtc')
group_H2O = u.select_atoms('type 1 or type 2')
group_CNT = u.select_atoms('type 3')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [5], line 1
----> 1 u = mda.Universe(datapath + 'lammps.data',
      2                  datapath + 'traj.xtc')
      3 group_H2O = u.select_atoms('type 1 or type 2')
      4 group_CNT = u.select_atoms('type 3')
```

(continues on next page)

(continued from previous page)

```
NameError: name 'mda' is not defined
```

Let us print a few information about the trajectory file:

[6]:
```
print(f"The number of water molecules is {np.int32(group_H2O.n_atoms/3)}")
print(f"The number of carbon atoms is {np.int32(group_CNT.n_atoms)}")
print(f"The CNT length is {np.round(u.dimensions[2]/10,2)} nm")
timestep = np.round(u.trajectory.dt,2)
print(f"The time intervl between the frames is {timestep} ps")
total_time = np.round(u.trajectory.totaltime,2)
print(f"The total simulation time is {total_time} ps")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [6], line 1
----> 1 print(f"The number of water molecules is {np.int32(group_H2O.n_atoms/3)}")
      2 print(f"The number of carbon atoms is {np.int32(group_CNT.n_atoms)}")
      3 print(f"The CNT length is {np.round(u.dimensions[2]/10,2)} nm")

NameError: name 'np' is not defined
```

Call the DensityCylinder module for the two groups at once:

[7]:
```
dcyl = maicos.DensityCylinder([group_H2O, group_CNT], center='all', binwidth = 0.01)
dcyl.run()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [7], line 1
----> 1 dcyl = maicos.DensityCylinder([group_H2O, group_CNT], center='all', binwidth = 0.
↪01)
      2 dcyl.run()

NameError: name 'maicos' is not defined
```

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:

[8]:
```
rcoor = dcyl.results['r']
dens_H2O = dcyl.results['dens_mean'].T[0]
dens_CNT = dcyl.results['dens_mean'].T[1]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [8], line 1
----> 1 rcoor = dcyl.results['r']
      2 dens_H2O = dcyl.results['dens_mean'].T[0]
      3 dens_CNT = dcyl.results['dens_mean'].T[1]

NameError: name 'dcyl' is not defined
```

By default the binwidth is 0.1 nanometers, the units are kg/m^3, and the axis is z. Plot the density profile using :

[9]:
```
fig = plt.figure(figsize=(13,6.5))
ax1 = plt.subplot(1, 1, 1)
plt.xlabel(r"$r$ coordinate (nm)", fontdict=font)
plt.ylabel(r"density (kg/m$^3$]", fontdict=font)
plt.xticks(fontsize=fontsize)
plt.yticks(fontsize=fontsize)
ax1.plot(rcoor, dens_H2O, color=my_color_1, linewidth=4)
ax1.plot(rcoor, dens_CNT, color=my_color_2, linewidth=4)
ax1.yaxis.offsetText.set_fontsize(20)
ax1.minorticks_on()
ax1.tick_params('both', length=10, width=2, which='major', direction='in')
ax1.tick_params('both', length=6, width=1.4, which='minor', direction='in')
ax1.xaxis.set_ticks_position('both')
ax1.yaxis.set_ticks_position('both')
ax1.spines["top"].set_linewidth(2)
ax1.spines["bottom"].set_linewidth(2)
ax1.spines["left"].set_linewidth(2)
ax1.spines["right"].set_linewidth(2)
ax1.yaxis.offsetText.set_fontsize(30)
minor_locator_y = AutoMinorLocator(2)
ax1.yaxis.set_minor_locator(minor_locator_y)
minor_locator_x = AutoMinorLocator(2)
ax1.xaxis.set_minor_locator(minor_locator_x)
ax1.tick_params(axis='x', pad=10)
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [9], line 1
----> 1 fig = plt.figure(figsize=(13,6.5))
      2 ax1 = plt.subplot(1, 1, 1)
      3 plt.xlabel(r"$r$ coordinate (nm)", fontdict=font)

NameError: name 'plt' is not defined
```

3.3.3 Dipole angle tutorial

Here we analyse a box of water molecules exposed to an alternating electric field. To follow this tutorial, the data test files `electricfwater` of MAICoS are needed.

Simulation details - The system contains $\sim$ 5000 water molecules simulated for 100 ps in the NVT ensemble at a temperature of 300K. Periodic boundary conditions were employed in all directions and long range electrostatics were modelled using the PME method. LINCS algorithm was used to constraint the H-Bonds. The alternating electric field was applied along the x-axis in the form of a Gaussian laser pulse. Please check gromacs electric field for more details.

To follow this tutorial, the data test files airwater of MAICoS are needed. You can obtain them by cloning MAICoS repository:

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

The `electricfwater` data files are located in `tests/data/electricfwater/`.

3.3.3.1 Option 1: from the Python interpreter

First, let us ignore unnecessary warnings:

[1]:
```
import warnings
warnings.filterwarnings("ignore")
```

Then, import MAICoS, NumPy, MDAnalysis, and PyPlot:

[2]:
```
import maicos
import numpy as np
import MDAnalysis as mda
import matplotlib.pyplot as plt
from matplotlib.ticker import AutoMinorLocator
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
Cell In [2], line 1
----> 1 import maicos
      2 import numpy as np
      3 import MDAnalysis as mda

File ~/checkouts/readthedocs.org/user_builds/maicos/envs/v0.5/lib/python3.10/site-
↪packages/maicos/__init__.py:36
     29 from .modules.density import DensityPlanar, DensityCylinder
     30 from .modules.epsilon import (
     31     EpsilonBulk,
     32     EpsilonPlanar,
     33     EpsilonCylinder,
     34     DielectricSpectrum
     35 )
---> 36 from .modules.structure import Saxs, Debye, Diporder
     37 from .modules.timeseries import DipoleAngle, KineticEnergy
     38 from .modules.transport import Velocity

File ~/checkouts/readthedocs.org/user_builds/maicos/envs/v0.5/lib/python3.10/site-
↪packages/maicos/modules/structure.py:24
     18 from .. import tables
     19 from ..decorators import (
     20     make_whole,
     21     set_planar_class_doc,
     22     set_verbose_doc,
     23 )
---> 24 from ..lib import sfactor
     25 from ..utils import check_compound, savetxt
     27 logger = logging.getLogger(__name__)

ImportError: dynamic module does not define module export function (PyInit_sfactor)
```

and set a few parameters for plotting purpose:

[3]:
```
fontsize = 25
font = {'family': 'sans', 'color':  'black',
        'weight': 'normal', 'size': fontsize}
```

(continues on next page)

(continued from previous page)

```
my_color_1 = np.array([0.090, 0.247, 0.560])
my_color_2 = np.array([0.235, 0.682, 0.639])
my_color_3 = np.array([1.000, 0.509, 0.333])
my_color_4 = np.array([0.588, 0.588, 0.588])
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [3], line 4
      1 fontsize = 25
      2 font = {'family': 'sans', 'color':  'black',
      3         'weight': 'normal', 'size': fontsize}
----> 4 my_color_1 = np.array([0.090, 0.247, 0.560])
      5 my_color_2 = np.array([0.235, 0.682, 0.639])
      6 my_color_3 = np.array([1.000, 0.509, 0.333])

NameError: name 'np' is not defined
```

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

[4]:
```
datapath = "../../../../tests/data/electricfwater/"
```

Create a MDAnalysis universe, and extract its atoms:

[5]:
```
u = mda.Universe(datapath+"mdelectric.tpr",
                 datapath+"mdelectric.trr")
atoms = u.atoms
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [5], line 1
----> 1 u = mda.Universe(datapath+"mdelectric.tpr",
      2                  datapath+"mdelectric.trr")
      3 atoms = u.atoms

NameError: name 'mda' is not defined
```

Important note: Because this calculation is computationaly heavy, let use isolate a small portion of the trajectory file. To perform the full calculation, just comment the following line:

[6]:
```
u.transfer_to_memory(stop = 100)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [6], line 1
----> 1 u.transfer_to_memory(stop = 100)

NameError: name 'u' is not defined
```

Let us print a few information about the trajectory file:

[7]:
```
print(f"The number of water molecules is {np.int32(u.atoms.n_atoms/3)}")
timestep = np.round(u.trajectory.dt,2)
print(f"The time interval between the frames is {timestep} ps")
total_time = np.round(u.trajectory.totaltime,2)
print(f"The total simulation time is {total_time} ps")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [7], line 1
----> 1 print(f"The number of water molecules is {np.int32(u.atoms.n_atoms/3)}")
      2 timestep = np.round(u.trajectory.dt,2)
      3 print(f"The time interval between the frames is {timestep} ps")

NameError: name 'np' is not defined
```

Run the MAICoS dipole angle function, selecting the x axis using the `dim` keyword:

[8]:
```
dipangle = maicos.DipoleAngle(atoms, dim=0)
dipangle.run()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [8], line 1
----> 1 dipangle = maicos.DipoleAngle(atoms, dim=0)
      2 dipangle.run()

NameError: name 'maicos' is not defined
```

The run produces a python dictionary named `dipangle.results` with 4 keys linked to numpy arrays as values: 1. timestep, 2. cosine of dipole and x-axis, 3. cosine squared, 4. product of cosine of dipoles i and j with $i \neq j$.

Extract the time vector and the $\cos(\theta_i)$ data from the `results` attribute:

[9]:
```
t = dipangle.results["t"]
cos_theta_i = dipangle.results["cos_theta_i"]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [9], line 1
----> 1 t = dipangle.results["t"]
      2 cos_theta_i = dipangle.results["cos_theta_i"]

NameError: name 'dipangle' is not defined
```

Plot the final results using :

[10]:
```
fig = plt.figure(figsize=(13,6.5))
ax1 = plt.subplot(1, 1, 1)
plt.xlabel(r"$t$ (ps)", fontdict=font)
plt.ylabel(r"cos($\theta_i$)", fontdict=font)
plt.xticks(fontsize=fontsize)
plt.yticks(fontsize=fontsize)
ax1.plot(t, cos_theta_i, color=my_color_1, linewidth=4)
ax1.yaxis.offsetText.set_fontsize(20)
ax1.minorticks_on()
ax1.tick_params('both', length=10, width=2, which='major', direction='in')
ax1.tick_params('both', length=6, width=1.4, which='minor', direction='in')
ax1.xaxis.set_ticks_position('both')
ax1.yaxis.set_ticks_position('both')
ax1.spines["top"].set_linewidth(2)
ax1.spines["bottom"].set_linewidth(2)
```

(continues on next page)

(continued from previous page)

```
ax1.spines["left"].set_linewidth(2)
ax1.spines["right"].set_linewidth(2)
ax1.yaxis.offsetText.set_fontsize(30)
minor_locator_y = AutoMinorLocator(2)
ax1.yaxis.set_minor_locator(minor_locator_y)
minor_locator_x = AutoMinorLocator(2)
ax1.xaxis.set_minor_locator(minor_locator_x)
ax1.tick_params(axis='x', pad=10)
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [10], line 1
----> 1 fig = plt.figure(figsize=(13,6.5))
      2 ax1 = plt.subplot(1, 1, 1)
      3 plt.xlabel(r"$t$ (ps)", fontdict=font)

NameError: name 'plt' is not defined
```

3.3.3.2 Option 2: from the command line interface

Go to the `electricfwater` data folder of MAICoS, then use the `maicos` command directly from the terminal:

```
cd tests/data/electricfwater/
maicos DipoleAngle -s md.tpr -f md.trr -d 0
```

The output file `dipangle.dat` is similar to `dipangle.results` and contains the data in columns. To know the full list of options, have a look at the `Inputs`.

3.3.4 Kinetic energy tutorial

Here we analyse a box of water molecules. To follow this tutorial, the data test files `kineticenergy` of MAICoS are needed. You can obtain them by cloning MAICoS repository:

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

Simulation details - The system contains around 500 water molecules simulated for 200 ps in the NVE ensemble. Periodic boundary conditions were employed in all directions, long range electrostatics were modelled using the PME method, and the LINCS algorithm was used to constraint the H-Bonds.

3.3.4.1 Option 1: from the Python interpreter

First, let us ignore unnecessary warnings:

[1]:
```
import warnings
warnings.filterwarnings("ignore")
```

Then, import MAICoS, NumPy, MDAnalysis, and PyPlot:

[2]:
```
import maicos
import numpy as np
import MDAnalysis as mda
import matplotlib.pyplot as plt
from matplotlib.ticker import AutoMinorLocator
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
Cell In [2], line 1
----> 1 import maicos
      2 import numpy as np
      3 import MDAnalysis as mda

File ~/checkouts/readthedocs.org/user_builds/maicos/envs/v0.5/lib/python3.10/site-
↪packages/maicos/__init__.py:36
     29 from .modules.density import DensityPlanar, DensityCylinder
     30 from .modules.epsilon import (
     31     EpsilonBulk,
     32     EpsilonPlanar,
     33     EpsilonCylinder,
     34     DielectricSpectrum
     35 )
---> 36 from .modules.structure import Saxs, Debye, Diporder
     37 from .modules.timeseries import DipoleAngle, KineticEnergy
     38 from .modules.transport import Velocity

File ~/checkouts/readthedocs.org/user_builds/maicos/envs/v0.5/lib/python3.10/site-
↪packages/maicos/modules/structure.py:24
     18 from .. import tables
     19 from ..decorators import (
     20     make_whole,
     21     set_planar_class_doc,
     22     set_verbose_doc,
     23 )
---> 24 from ..lib import sfactor
     25 from ..utils import check_compound, savetxt
     27 logger = logging.getLogger(__name__)

ImportError: dynamic module does not define module export function (PyInit_sfactor)
```

and set a few parameters for plotting purpose:

[3]:
```
fontsize = 25
font = {'family': 'sans', 'color':  'black',
        'weight': 'normal', 'size': fontsize}
my_color_1 = np.array([0.090, 0.247, 0.560])
my_color_2 = np.array([0.235, 0.682, 0.639])
my_color_3 = np.array([1.000, 0.509, 0.333])
my_color_4 = np.array([0.588, 0.588, 0.588])
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [3], line 4
      1 fontsize = 25
```

(continues on next page)

(continued from previous page)

```
      2 font = {'family': 'sans', 'color':  'black',
      3         'weight': 'normal', 'size': fontsize}
----> 4 my_color_1 = np.array([0.090, 0.247, 0.560])
      5 my_color_2 = np.array([0.235, 0.682, 0.639])
      6 my_color_3 = np.array([1.000, 0.509, 0.333])

NameError: name 'np' is not defined
```

Define the path to the kineticenergy data folder of MAICoS:

[4]:
```
datapath = "../../../../tests/data/kineticenergy/"
```

Create a MDAnalysis universe:

[5]:
```
u = mda.Universe(datapath+"nve.tpr",
                 datapath+"nve.trr")
atoms = u.atoms
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [5], line 1
----> 1 u = mda.Universe(datapath+"nve.tpr",
      2                  datapath+"nve.trr")
      3 atoms = u.atoms

NameError: name 'mda' is not defined
```

Let us print a few information about the trajectory file:

[6]:
```
print(f"The number of water molecules is {np.int32(u.atoms.n_atoms/3)}")
timestep = np.round(u.trajectory.dt,2)
print(f"The time interval between the frames is {timestep} ps")
total_time = np.round(u.trajectory.totaltime,2)
print(f"The total simulation time is {total_time} ps")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [6], line 1
----> 1 print(f"The number of water molecules is {np.int32(u.atoms.n_atoms/3)}")
      2 timestep = np.round(u.trajectory.dt,2)
      3 print(f"The time interval between the frames is {timestep} ps")

NameError: name 'np' is not defined
```

Run the MAICOS dipole angle function:

[7]:
```
kenergy = maicos.KineticEnergy(atoms)
kenergy.run()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [7], line 1
----> 1 kenergy = maicos.KineticEnergy(atoms)
      2 kenergy.run()
```

(continues on next page)

(continued from previous page)

```
NameError: name 'maicos' is not defined
```

Then extract the kinetic energy:

[8]:
```
ke_trans = kenergy.results["trans"]
ke_rot = kenergy.results["rot"]
print(f"The average translational kinetic energy "
      f"is {np.round(np.mean(ke_trans),2)} kJ/mol")
print(f"The average rotational kinetic energy "
      f"is {np.round(np.mean(ke_rot),2)} kJ/mol")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [8], line 1
----> 1 ke_trans = kenergy.results["trans"]
      2 ke_rot = kenergy.results["rot"]
      3 print(f"The average translational kinetic energy "
      4       f"is {np.round(np.mean(ke_trans),2)} kJ/mol")

NameError: name 'kenergy' is not defined
```

3.3.4.2 Option 2 : from the command line interface

Go to the `kineticenergy` data folder of MAICoS, then use the `maicos` command directly from the terminal:

```
maicos KineticEnergy -s md.tpr -f md.trr -o ke.dat
```

The output file `ke.dat` is similar to `ke.results` and contains the data in columns

3.4 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.

3.4.1 Testing

Continuous Integration pipeline is based on Tox. So you need to install *tox* first:

```
pip install tox
# or
conda install tox-c conda-forge
```

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 docs  # only builds the documentation
tox -e tests  # testing
```

3.4.2 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!

3.4.3 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:

```
tox -e docs
```

Then, visualise the local documentation with your favourite internet explorer (here Mozilla Firefox is used)

```
firefox dist/docs/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.

3.5 Authors list

3.5.1 Maintainer

- Philip Loche

3.5.2 Authors

- Alexander Schlaich
- Philip Loche

3.5.3 Contributors

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

3.6 Changelog

3.6.1 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)

3.6.2 v0.4.1 (2021/12/17)

Philip Loche

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

3.6.3 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.

3.6.4 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

3.6.5 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

3.6.6 v0.1 (2019/10/30)

Philip Loche

- first release out of the lab

3.7 Density modules

The density modules of MAICoS are tools for computing density, temperature, and chemical potential profiles from molecular simulation trajectory files. Profiles can be extracted either in Cartesian or cylindrical coordinate systems. Units for the density are the same as GROMACS, i.e. mass, number or charge. See the gmx density manual for details.

From the command line

You can extract a density profile from molecular dynamics trajectory files directly from the terminal. For this example, we use the `airwater` data file of MAICoS. First, go to the directory

```
cd tests/data/airwater/
```

then type:

```
maicos DensityPlanar -s conf.gro -traj traj.trr
```

Here `conf.gro` and `traj.trr` are GROMACS configuration and trajectory files, respectively. The density profile appears in a `.dat` file. You can visualise all the options of the module `DensityPlanar` by typing

```
maicos DensityPlanar -h
```

From the Python interpreter

In order to calculate the density using MAICoS in a Python environment, first import MAICoS and MDAnalysis:

```
import MDAnalysis as mda
import maicos
```

Then create a MDAnalysis universe:

```
u = mda.Universe('conf.gro', 'traj.trr')
group_H2O = u.select_atoms('type O or type H')
```

And run MAICoS' `DensityPlanar` module:

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

Results can be accessed from `dplan.results`. More details are given in the *Tutorials*.

3.7.1 Density planar

Compute partial densities/temperature profiles in the Cartesian systems.

3.7.1.1 Parameters

atomgroups
: [list[AtomGroup]] a list of `AtomGroup` for which the densities are calculated

dens
: [str] Density: mass, number, charge, temperature.

dim
: [int] Dimension for binning (x=0, y=1, z=2)

binwidth
: [float] binwidth (nanometer)

comgroup
: [AtomGroup] Perform the binning relative to the center of mass of the selected group.

center
: [bool] Perform the binning relative to the center of the (changing) box.

mu
: [bool] Calculate the chemical potential (requires dens='number')

muout
: [str] Prefix for output filename for chemical potential

temperature
: [float] temperature (K) for chemical potential

mass
: [float] Mass (u) for the chemical potential. By default taken from topology.

zpos
[float] position (nm) at which the chemical potential will be computed. By default average over box.

output
[str] Output filename

concfreq
[int] Default number of frames after which results are calculated and files refreshed. If *0* results are only calculated at the end of the analysis and not saved by default.

verbose
[bool] Turn on more logging and debugging

3.7.1.2 Attributes

results.z
[list] bins

results.dens_mean
[np.ndarray] calculated densities

results.dens_std
[np.ndarray] density standard deviation

results.dens_err
[np.ndarray] density error

results.mu
[float] chemical potential (only if *mu=True*)

results.dmu
[float] error of chemical potential (only if *mu=True*)

3.7.2 Density cylinder

Compute partial densities across a cylinder.

3.7.2.1 Parameters

atomgroups
[list[AtomGroup]] A list of `AtomGroup` for which the densities are calculated.

dens
[str] Density: mass, number, charge, temp

dim
[int] Dimension for binning (x=0, y=1, z=2)

center
[str] Perform the binning relative to the center of this selection string of teh first AtomGroup. If *None* center of box is used.

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

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

binwidth
[float] binwidth (nanometer)

output
[str] Output filename

concfreq
[int] Default number of frames after which results are calculated and files refreshed. If *0* results are only calculated at the end of the analysis and not saved by default.

verbose
[bool] Turn on more logging and debugging

3.7.2.2 Attributes

results.r
[np.ndarray] bins

results.dens_mean
[np.ndarray] calculated densities

results.dens_mean_sq
[np.ndarray] squared calculated density

results.dens_std
[np.ndarray] density standard deviation

results.dens_err
[np.ndarray] density error

3.8 Dielectric constant modules

3.8.1 Epsilon bulk

3.8.1.1 Description

Compute dipole moment fluctuations and static dielectric constant.

Parameters

atomgroup
[AtomGroup] Atomgroup on which the analysis is executed

make_whole
[bool] Make molecules whole; If the input already contains whole molecules this can be disabled to gain speedup

temperature
[float] temperature (K)

output
[str] Output filename.

concfreq
[int] Default number of frames after which results are calculated and files refreshed. If *0* results are only calculated at the end of the analysis and not saved by default.

verbose
[bool] Turn on more logging and debugging

Attributes

results.M
[numpy.ndarray] *Directional dependant dipole moment :math:*langle boldsymbol M rangle` in e.

results.M2
[numpy.ndarray] Directional dependant squared dipole moment $\langle \boldsymbol{M}^2 \rangle$ in $(e)^2$

results.fluct
[float] Directional dependant dipole moment fluctuation $\langle \boldsymbol{M}^2 \rangle - \langle \boldsymbol{M} \rangle^2$ in $(e)^2$

results.eps
[numpy.ndarray] Directional dependant static dielectric constant

results.eps_mean
[float] Static dielectric constant

3.8.2 Epsilon planar

3.8.2.1 Description

Calculate planar dielectric profiles.

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

Parameters

atomgroups
[list[AtomGroup]] a list of `AtomGroup` for which the dielectric profiles are calculated

dim
[int] Dimension for binning (x=0, y=1, z=2)

binwidth
[float] binwidth (nanometer)

comgroup
[AtomGroup] Perform the binning relative to the center of mass of the selected group.

center
[bool] Perform the binning relative to the center of the (changing) box.

zmin
[float] minimal coordinate for evaluation (nm)

zmax
[float] maximal coordinate for evaluation (nm). If *None* the whole box is taken into account.

xy
[bool] Use 2D slab geometry

vac
[bool] Use vacuum boundary conditions instead of metallic (2D only!).

sym
[bool] symmetrize the profiles

make_whole
[bool] Make molecules whole; If the input already contains whole molecules this can be disabled to gain speedup

temperature
[float] temperature (K)

output_prefix
[str] Prefix for output files

concfreq
[int] Default number of frames after which results are calculated and files refreshed. If *0* results are only calculated at the end of the analysis and not saved by default.

verbose
[bool] Turn on more logging and debugging

Attributes

results.z
[list] bins

results.dens_mean
[np.ndarray] eps_par: Parallel dielectric profile (_ - 1)

results.deps_par
[np.ndarray] Error of parallel dielectric profile

results.eps_par_self
[np.ndarray] Self contribution of parallel dielectric profile

results.eps_par_coll
[np.ndarray] Collective contribution of parallel dielectric profile

results.eps_perp
[np.ndarray] Inverse perpendicular dielectric profile (^{-1}_ - 1)

results.deps_perp
[np.ndarray] Error of inverse perpendicular dielectric profile

results.eps_par_self
[np.ndarray] Self contribution of Inverse perpendicular dielectric profile

results.eps_perp_coll
[np.ndarray] Collective contribution of Inverse perpendicular dielectric profile

3.8.3 Epsilon cylinder

3.8.3.1 Description

Calculate cylindrical dielectric profiles.

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

Parameters

atomgroup
[AtomGroup] `AtomGroup` for which the dielectric profiles are calculated

geometry
[str] A structure file without water from which com is calculated.

radius
[float] Radius of the cylinder (nm)

binwidth
[float] Bindiwdth the binwidth (nm)

variable_dr
[bool] Use a variable binwidth, where the volume is kept fixed.

length
[float] Length of the cylinder (nm)

make_whole
[bool] Make molecules whole; If the input already contains whole molecules this can be disabled to gain speedup

temperature
[float] temperature (K)

single
[bool] "1D" line of watermolecules

output_prefix
[str] Prefix for output_prefix files

concfreq
[int] Default number of frames after which results are calculated and files refreshed. If *0* results are only calculated at the end of the analysis and not saved by default.

verbose
[bool] Turn on more logging and debugging

Attributes

results.r
[np.ndarray] bins

results.eps_ax
[np.ndarray] Parallel dielectric profile (_)

results.deps_ax
[np.ndarray] Error of parallel dielectric profile

results.eps_rad
[np.ndarray] Inverse perpendicular dielectric profile (^{-1}_)

results.deps_rad
[np.ndarray] Error of inverse perpendicular dielectric profile

3.8.4 Dielectric spectrum

3.8.4.1 Description

Compute 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. ν or f, rather than ω) frequency, along with the associated standard deviations. The algorithm is based on the Fluctuation Dissipation Relation (FDR): $\chi(f) = -1/(3Vk_BT\varepsilon_0)FT[\theta(t)\langle P(0)dP(t)/dt\rangle]$. 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.

Parameters

atomgroup
: [AtomGroup] Atomgroup on which the analysis is executed

make_whole
: [bool] Make molecules whole; If the input already contains whole molecules this can be disabled to gain speedup

temperature
: [float] Reference temperature.

output_prefix
: [str] Prefix for the 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.

noplots
: [bool] Prevents plots from being generated.

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

ymin
: [float] Manually sets the minimum lower bound for the log-log plot.

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.

verbose
: [bool] Turn on more logging and debugging

Attributes

results

3.9 Structure modules

3.9.1 Saxs

3.9.1.1 Description

3.9.2 Diporder

3.9.2.1 Description

3.9.3 Debyer

3.9.3.1 Description

3.10 Timeseries modules

3.10.1 Dipole angle

3.10.1.1 Description

Calculate angle timeseries of dipole moments with respect to an axis.

Parameters

atomgroup
: [AtomGroup] Atomgroup on which the analysis is executed

dim
: [int] refernce vector for angle (x=0, y=1, z=2)

output
: [str] Prefix for output filenames

concfreq
: [int] Default number of frames after which results are calculated and files refreshed. If *0* results are only calculated at the end of the analysis and not saved by default.

verbose
: [bool] Turn on more logging and debugging

Attributes

results.t
[np.ndarray] time (ps)

resulst.cos_theta_i
[np.ndarray] Average cos between dipole and axis

resulst.cos_theta_ii
[np.ndarray] Average cos^2 of the same between dipole and axis

resulst.cos_theta_ij
[np.ndarray] Product cos of dipole i and cos of dipole j (i!=j)

3.10.2 Kinetic energy

3.10.2.1 Description

Calculate the timeseries of energies.

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.

Parameters

atomgroup
[AtomGroup] Atomgroup on which the analysis is executed

refpoint
[str] reference point for molecular center: center of mass (COM) or center of charge (COC) Note: The oxygen position only works for systems of pure water

output
[str] Output filename

${VERBOSE_PARAMETER}

Attributes

results.t
[np.ndarray] time (ps)

results.trans
[np.ndarray] translational kinetic energy (kJ/mol)

results.rot
[np.ndarray] rotational kinetic energy (kJ/mol)

3.11 Transport modules

3.11.1 Velocity

3.11.1.1 Description

Analyse mean velocity..

Reads in coordinates and velocities from a trajectory and calculates a velocity profile along a given axis. The obtained profile is averaged over the 4 symmetric slab halfs.Error bars are estimated via block averaging as described in [1].

[1] Hess, B. Determining the shear viscosity of model liquids from molecular dynamics simulations. The Journal of Chemical Physics 116, 209-217 (2002).

Parameters

atomgroup
[AtomGroup] Atomgroup on which the analysis is executed

dim
[int] Dimension for position binning (x=0, y=1, z=2)

vdim
[int] Dimension for velocity binning (x=0, y=1, z=2)

n_bins
[int] Number of bins. For making use of symmetry must be a multiple of 4.

n_block
[int] Maximum number of blocks for block averaging error estimate; 1 results in standard error

make_whole
[bool] Make molecules whole; If the input already contains whole molecules this can be disabled to gain speedup

output_suffix
[str] Suffix for output filenames

concfreq
[int] Default number of frames after which results are calculated and files refreshed. If *0* results are only calculated at the end of the analysis and not saved by default.

verbose
[bool] Turn on more logging and debugging

Attributes

results.z
[np.ndarray] bins [nm]

results.v
[np.ndarray] velocity profile [m/s]

results.ees
[np.ndarray] velocity error estimate [m/s]

results.symz
[np.ndarray] symmetrized bins [nm]

results.symvel
[np.ndarray] symmetrized velocity profile [m/s]

results.symees
[np.ndarray] symmetrized velocity error estimate [m/s]

PYTHON MODULE INDEX

m

INDEX

M