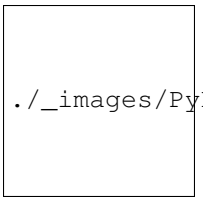
PyBILT Documentation

Release 0.1.0

Blake A. Wilson

Sep 17, 2019

CONTENTS:


./_images/PyBILT_logo.png

CHAPTER

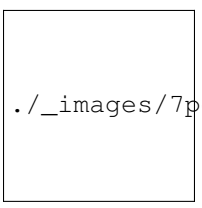
ONE

*PY*THON BASED LIPID *BIL*AYER MOLECULAR SIMULATION ANALYSIS *T*OOLKIT

1.1 PyBILT is a Python toolkit developed to analyze molecular simulation trajectories of lipid bilayers systems. The toolkit includes a variety of analyses from various lipid bilayer molecular simulation publications.

The analyses include:

- Mean Squared Displacement (MSD)
- Diffusion coefficent estimators (from MSD curves) - includes Einstein relation, linear fit, and anomalous diffusion fit.
- Area per lipid estimators
- Bilayer thickness
- Displacement Vector (flow) maps and correlations
- Deuterium order parameter
- Orientation parameters
- Mass and Electron Density Estimators
- and more!



CHAPTER

TWO

INSTALL

2.1 Core dependencies

PyBILT has the following core dependencies:

- MDAnalysis
- NumPy
- SciPy
- Matplotlib
- Seaborn
- six
- future

2.2 Python version support

The `pybilt` package has been tested using Anaconda Python 2.7, 3.6, and 3.7.

2.2.1 Sunsetting of Python 2

Please be aware that Python 2 is scheduled to be sunset on January 1 2020. You can read about it here: https://www.python.org/doc/sunset-python-2/ Parallel to the sunsetting of Python 2 many open source packages are also dropping support for Python 2 (https://python3statement.org/), including some of **PyBILT**'s core dependencies. As such, after January 1, 2020, **PyBILT** will also likely sunset its support for Python 2.7.

2.3 pip install

You can install the latest version of the `pybilt` package using `pip` sourced from the GitHub repo:

```
pip install -e git+https://github.com/LoLab-VU/PyBILT@v0.2.0#egg=pybilt
```

However, this will not automatically install the core dependencies. You will have to do that separately:

```
pip install MDAnalysis numpy scipy matplotlib seaborn six future
```

2.4 conda install

First make sure you have the `conda-forge` channel in your channel list; that is the channel from which MDAnalysis is installed. You can use the following command to add it to the bottom of your channel list:

```
conda config --append channels conda-forge
```

Then you can install the `pybilt` package from the `blakeaw` Anaconda Cloud channel,

```
conda install -c blakeaw pybilt
```

The core dependencies will be automatically installed.

2.5 Recommended additional software

The following software is not required for the basic operation of **PyBILT**, but provides extra capabilities and features when installed.

2.5.1 pytest

The `pybilt` test suite is designed to be run with pytest, so if you want to run the tests then you will need to install pytest.

2.5.2 Jupyter

PyBILT comes with a set of Jupyter IPython notebooks which supplement the doc pages. If you want to run these notebooks locally then you will need to intall Jupyter (or at least the IPython kernel).

Note that the notebooks have not been updated for Python 3 yet.

2.5.3 sphinx, sphinx_rtd_theme, and recommonmark

If you want to build local versions of doc pages install the following additional packages:

- sphinx
- sphinx_rtd_theme
- recommonmark

CHAPTER

THREE

DOCUMENTATION AND USAGE

3.1 Quick overview of PyBILT

PyBILT is composed of 2 primary analysis packages:

- bilayer_analyzer – The bilayer_analyzer is an analysis package that is designed to analyze (quasi) planar lipid bilayer systems. It is accessed through the BilayerAnalyzer object, which can be imported via: `from pybilt.bilayer_analyzer import BilayerAnalyzer`. The BilayerAnalyzer features automatic dynamic unwrapping of coordinates and leaflet detection. The bilayer_analyzer works on a multiple-representation model, whereby the various analyses are conducted using different representations of the bilayer lipids. Bilayer lipids can be represented using the following four representations:
  - All atom
  - Centers-of-mass – Each lipid (or selection of atoms from each lipid) is reduced to a center-of-mass.
  - Grid (or lipid grid) – The lipids are mapped to two-dimensional grids (one for each leaflet) in the style of the GridMAT-MD method
  - Vectors - Each lipid is converted to a vector representation using select reference atoms (or sets of reference atoms) that are used to compute the head and tail of the vector; e.g., a lipid tail atom to lipid head atom, or P-N vectors.

The bilayer_analyzer features various types of analyses and the use of different representations is handled internally based the requirements and design of each analysis type. See the documentation for list of analyses that can be added to intances of the BilayerAnalyzer.

- mda_tools – This package includes various modules and functions for directly analyzing and operating on MDAnalysis trajectories and objects. e.g. functions to compute density profiles.

Additional packages include:

- lipid_grid – The lipid grid module can be used construct "lipid grid" grid representations of lipid bilayers, which can be used to accurately estimate quantities such as area per lipid.
- com_trajectory – This module can be used to construct a center of mass trajectory (COMTraj) out of an MDAnalysis trajectory, which is useful for computing quantities like mean squared displacement. The COMTraj is designed to work with bilayers.
- plot_generation – This module has several pre-written plotting functions (using matplotlib and seaborn) for some of the properties that can be computed from functions in the other modules. e.g. mean squared displacement and area per lipid.

3.2 Docs

Visit the **PyBILT** docs on Read the Docs. Docs can also be viewed offline/locally by opening the PyBILT/docs/build/html/index.html file from the repo in a web browser; however, this build of the docs is not updated often.

3.3 Jupyter IPython notebooks

In addition to the Docs, there are currently a few Jupyter IPython notebooks that provide some examples and show some basic usage (these have not been updated/tested for/with python 3 yet); updates and more of these are in the pipeline.

CHAPTER

FOUR

CONTACT

To report problems or bugs please open a GitHub Issue. Additionally, any comments, suggestions, or feature requests for **PyBILT** can also be submitted as a GitHub Issue.

CHAPTER

FIVE

CONTRIBUTING

If you would like to contribute directly to **PyBILT**'s development please

1. Fork the repo (https://github.com/LoLab-VU/PyBILT/fork)
2. Create a new branch for your feature (git checkout -b feature/foo_bar)
3. Create test code for your feature
4. Once your feature passes its own test, run all the tests using pytest (python -m pytest)
5. Once your feature passes all the tests, commit your changes (git commit -am 'Add the foo_bar feature.')
6. Push to the branch (git push origin feature/foo_bar)
7. Create a new Pull Request

CHAPTER

SIX

LICENSE

This project is licensed under the MIT License - see the LICENSE file for details

CHAPTER

SEVEN

ACKNOWLEDGMENTS

- A special thanks to James Pino (https://github.com/JamesPino) for his inciteful comments and suggestions that have helped improve the quality of this code, and thanks to him for pointing out some very useful coding tools.
- Thanks to my advisors, Carlos F. Lopez and Arvind Ramanathan, for catalyzing this project and for providing me with the space and means to pursue it.

CHAPTER

EIGHT

CITING

If you use the **PyBILT** software as a part of your research, please cite the its use. You can export the **PyBILT** software citation in your preferred format from its Zenodo DOI entry.

Also, please cite the following references as appropriate for scientific/research software used with/via **PyBILT**:

8.1 MDAnalysis

See: https://www.mdanalysis.org/pages/citations/

8.2 Packages from the SciPy ecosystem

These include NumPy, SciPy, and Matplotlib for which references can be obtained from: https://www.scipy.org/citing.html

8.3 seaborn

Reference can be exported from the seaborn Zeondo DOI entry

8.4 Jupyter

See: https://github.com/jupyter/jupyter/issues/190

8.5 IPython

See: https://ipython.org/citing.html

CHAPTER

NINE

ANALYSES AVAILABLE TO THE BILAYERANALYZER

9.1 ac - Isothermal area compressibility.

9.1.1 BilayerAnalyzer analysis: `ac` - Isothermal area compressibility.

9.1.2 Description

Estimate the isothermal area compressibility.

This protocol is used to estimate the area compressibility modulus, K_A^-1 = [(kT) / var(A)]^-1 , where A is the area in the lateal dimension of the bilayer.

This protocol is identified by the analysis key: 'ac'

Initiated by instance of:

```
<class 'pybilt.bilayer_analyzer.analysis_protocols.
↪AreaCompressibilityProtocol'>
```

9.1.3 Syntax

```
ac analysis-ID keyword value
```

- ac = analysis-Key - keyword/name for this analysis.
- analysis-ID = The unique name/ID being assigned to this analysis.
- keyword value = settings keyword value pairs
  - temperature (float): The absolute temperature that the simulation was run at (i.e. in Kelvin). Default: 298.15 K

Examples

Construct analyzer:

```
analyzer = BilayerAnalyzer(structure='name_of_structure_file',
                           trajectory='name_of_traj_file',
                           selection='resname POPC DOPC')
```

Add by string - use default settings:

```
analyzer.add_analysis('ac ac_1')
```

Add by string - adjust a setting:

```
analyzer.add_analysis('ac ac_1 temperature 298.15')
```

Add by list:

```
analyzer.add_analysis(list(['ac', 'ac_1', dict({'temperature':298.15})]))
```

Add by dict:

```
analyzer.add_analysis(dict({'analysis_key': 'ac', 'analysis_id': 'ac_1',
↪'analysis_settings':dict({'temperature':298.15})}))
```

To remove from analyzer:

```
analyzer.remove_analysis('ac_1')
```

Output Info:

Retrieve output after running analyses:

```
output = analyzer.get_analysis_data('ac_1')
```

The output is type `<type 'numpy.ndarray'>`

9.1.4 Related analyses

- acm
- vcm

9.1.5 References

1. Yoshimichi Andoha, Susumu Okazakia, Ryuichi Ueokab, "Molecular dynamics study of lipid bilayers modeling the plasma membranes of normal murine thymocytes and leukemic GRSL cells", Biochimica et Biophysica Acta (BBA) - Biomembranes, Volume 1828, Issue 4, April 2013, Pages 1259-1270. https://doi.org/10.1016/j.bbamem.2013.01.005

9.2 acm - Area compressibility modulus.

9.2.1 BilayerAnalyzer analysis: `acm` - Area compressibility modulus.

9.2.2 Description

Estimate the isothermal area compressibility modulus.

This protocol is used to estimate the area compressibility modulus, K_A = (kT) / var(A), where A is the area in the lateal dimension of the bilayer.

This protocol is identified by the analysis key: 'acm'

Initiated by instance of:

```
<class 'pybilt.bilayer_analyzer.analysis_protocols.
↪AreaCompressibilityModulusProtocol'>
```

9.2.3 Syntax

```
acm analysis-ID keyword value
```

- acm = analysis-Key - keyword/name for this analysis.
- analysis-ID = The unique name/ID being assigned to this analysis.
- keyword value = settings keyword value pairs
  - temperature (float): The absolute temperature that the simulation was run at (i.e. in Kelvin). Default: 298.15 K

Examples

Construct analyzer:

```
analyzer = BilayerAnalyzer(structure='name_of_structure_file',
                           trajectory='name_of_traj_file',
                           selection='resname POPC DOPC')
```

Add by string - use default settings:

```
analyzer.add_analysis('acm acm_1')
```

Add by string - adjust a setting:

```
analyzer.add_analysis('acm acm_1 temperature 298.15')
```

Add by list:

```
analyzer.add_analysis(list(['acm', 'acm_1', dict({'temperature':298.15})]))
```

Add by dict:

```
analyzer.add_analysis(dict({'analysis_key': 'acm', 'analysis_id': 'acm_1',
↪'analysis_settings':dict({'temperature':298.15})}))
```

To remove from analyzer:

```
analyzer.remove_analysis('acm_1')
```

Output Info:

Retrieve output after running analyses:

```
output = analyzer.get_analysis_data('acm_1')
```

The output is type <type 'numpy.ndarray'>

9.2.4 Related analyses

- ac
- vcm

9.2.5 References

1. Christofer Hofsab, Erik Lindahl, and Olle Edholm, "Molecular Dynamics Simulations of Phospholipid Bilayers with Cholesterol", Biophys J. 2003 Apr; 84(4): 2192-2206. doi: 10.1016/S0006-3495(03)75025-5
2. L. Janosi and A. A. Gorfe, J. Chem. Theory Comput. 2010, 6, 3267-3273
3. D. Aguayo, F. D. Gonzalez-Nilo, and C. Chipot, J. Chem. Theory Comput. 2012, 8, 1765-1773

9.3 ald - Average lateral displacement.

9.3.1 BilayerAnalyzer analysis: `ald` - Average lateral displacement.

9.3.2 Description

Estimate the average lateral displacement of lipids.

This protocol is identified by the analysis key: 'ald'

Initiated by instance of:

```
<class 'pybilt.bilayer_analyzer.analysis_protocols.ALDProtocol'>
```

9.3.3 Syntax

```
ald analysis-ID keyword value
```

- ald = analysis-Key - keyword/name for this analysis.
- analysis-ID = The unique name/ID being assigned to this analysis.
- keyword value = settings keyword value pairs
  - leaflet (str: 'both', 'upper', or 'lower'): Specifies the bilayer leaflet to include in the estimate. Default: 'both'
  - resname (str): Specify the resname of the lipid type to include in this analysis. Default: 'all', includes all lipid types.

Examples

Construct analyzer:

```
analyzer = BilayerAnalyzer(structure='name_of_structure_file',
                           trajectory='name_of_traj_file',
                           selection='resname POPC DOPC')
```

Add by string - use default settings:

```
analyzer.add_analysis('ald ald_1')
```

Add by string - adjust a setting:

```
analyzer.add_analysis('ald ald_1 leaflet both')
```

Add by list:

```
analyzer.add_analysis(list(['ald', 'ald_1', dict({'leaflet':'both'})]))
```

Add by dict:

```
analyzer.add_analysis(dict({'analysis_key': 'ald', 'analysis_id': 'ald_1',
↪'analysis_settings':dict({'leaflet':'both'})}))
```

To remove from analyzer:

```
analyzer.remove_analysis('ald_1')
```

Output Info:

Retrieve output after running analyses:

```
output = analyzer.get_analysis_data('ald_1')
```

The output is type `<type 'numpy.ndarray'>`

9.3.4 Related analyses

- msd

9.3.5 References

1. Kenichiro Koshiyama, Tetsuya Kodama, Takeru Yano, Shigeo Fujikawa, "Molecular dynamics simulation of structural changes of lipid bilayers induced by shock waves: Effects of incident angles", Biochimica et Biophysica Acta (BBA) - Biomembranes, Volume 1778, Issue 6, June 2008, Pages 1423-1428

9.4 apl_box - Area per lipid using box dimensions.

9.4.1 BilayerAnalyzer analysis: `apl_box` - Area per lipid using box dimensions.

9.4.2 Description

Estimate the area per lipid using the lateral area.

This analysis is used to estimate the area per lipid (APL) using the lateral box dimensions, A_l = 2<A_xy>/N_l , where A_xy is area of the lateral box dimensions (used to approximate the surface area of the bilayer) and N_l is the number of lipids in the bilayer. As a molar quantity this approach is only accurate for homogenous lipid bilayers. If the bilayer is inhomogenous then this estimate represents a composite average of the area per lipid.

This protocol is identified by the analysis key: 'apl_box'

Initiated by instance of:

```
<class 'pybilt.bilayer_analyzer.analysis_protocols.APLBoxProtocol'>
```

9.4.3 Syntax

```
apl_box analysis-ID
```

- apl_box = analysis-Key - keyword/name for this analysis.
- analysis-ID = The unique name/ID being assigned to this analysis.

Examples

Construct analyzer:

```
analyzer = BilayerAnalyzer(structure='name_of_structure_file',
                           trajectory='name_of_traj_file',
                           selection='resname POPC DOPC')
```

Add by string - use default settings:

```
analyzer.add_analysis('apl_box apl_box_1')
```

Add by string - adjust a setting:

```
analyzer.add_analysis('apl_box apl_box_1 none None')
```

Add by list:

```
analyzer.add_analysis(list(['apl_box', 'apl_box_1', dict({'none':None})]))
```

Add by dict:

```
analyzer.add_analysis(dict({'analysis_key': 'apl_box', 'analysis_id': 'apl_
↪box_1','analysis_settings':dict({'none':None})}))
```

To remove from analyzer:

```
analyzer.remove_analysis('apl_box_1')
```

Output Info:

Retrieve output after running analyses:

```
output = analyzer.get_analysis_data('apl_box_1')
```

The output is type `<type 'numpy.ndarray'>`

9.4.4 Related analyses

- apl_grid

9.4.5 References

1. Preston B. Moore, Carlos F. Lopez, Michael L. Klein, Dynamical Properties of a Hydrated Lipid Bilayer from a Multinanosecond Molecular Dynamics Simulation, Biophysical Journal, Volume 81, Issue 5, 2001, Pages 2484-2494, ISSN 0006-3495, http://dx.doi.org/10.1016/S0006-3495(01)75894-8. (http://www.sciencedirect.com/science/article/pii/S0006349501758948)

9.5 apl_grid - Area per lipid using 2D lipid grids.

9.5.1 BilayerAnalyzer analysis: `apl_grid` - Area per lipid using 2D lipid grids.

9.5.2 Description

Estimate the indvidual area per lipid for each lipid type using a gridding procedure.

This protocol is identified by the analysis key: 'apl_grid'

Initiated by instance of:

```
<class 'pybilt.bilayer_analyzer.analysis_protocols.APLGridProtocol'>
```

9.5.3 Syntax

```
apl_grid analysis-ID
```

- apl_grid = analysis-Key - keyword/name for this analysis.
- analysis-ID = The unique name/ID being assigned to this analysis.

Examples

Construct analyzer:

```
analyzer = BilayerAnalyzer(structure='name_of_structure_file',
                           trajectory='name_of_traj_file',
                           selection='resname POPC DOPC')
```

Add by string - use default settings:

```
analyzer.add_analysis('apl_grid apl_grid_1')
```

Add by string - adjust a setting:

```
analyzer.add_analysis('apl_grid apl_grid_1 none None')
```

Add by list:

```
analyzer.add_analysis(list(['apl_grid', 'apl_grid_1', dict({'none':None})]))
```

Add by dict:

```
analyzer.add_analysis(dict({'analysis_key': 'apl_grid', 'analysis_id': 'apl_
↪grid_1','analysis_settings':dict({'none':None})}))
```

To remove from analyzer:

```
analyzer.remove_analysis('apl_grid_1')
```

Output Info:

Retrieve output after running analyses:

```
output = analyzer.get_analysis_data('apl_grid_1')
```

The output is type `<type 'dict'>`

9.5.4 Related analyses

- apl_box

9.5.5 References

1. Allen et al. Vol. 30, No. 12 Journal of Computational Chemistry
2. Gapsys et al. J Comput Aided Mol Des (2013) 27:845-858

9.6 area_fluctuation - Bilayer lateral box area fluctuation.

9.6.1 BilayerAnalyzer analysis: `area_fluctuation` - Bilayer lateral box area fluctuation.

9.6.2 Description

Estimate the area fluctuation in the box along the bilayer laterals.

This protocol is identified by the analysis key: 'area_fluctuation'

Initiated by instance of:

```
<class 'pybilt.bilayer_analyzer.analysis_protocols.AreaFluctuationProtocol'>
```

9.6.3 Syntax

```
area_fluctuation analysis-ID
```

- area_fluctuation = analysis-Key - keyword/name for this analysis.
- analysis-ID = The unique name/ID being assigned to this analysis.

Examples

Construct analyzer:

```
analyzer = BilayerAnalyzer(structure='name_of_structure_file',
                           trajectory='name_of_traj_file',
                           selection='resname POPC DOPC')
```

Add by string - use default settings:

```
analyzer.add_analysis('area_fluctuation area_fluctuation_1')
```

Add by string - adjust a setting:

```
analyzer.add_analysis('area_fluctuation area_fluctuation_1 none None')
```

Add by list:

```
analyzer.add_analysis(list(['area_fluctuation', 'area_fluctuation_1', dict({
↪'none':None})]))
```

Add by dict:

```
analyzer.add_analysis(dict({'analysis_key': 'area_fluctuation', 'analysis_id
↪': 'area_fluctuation_1','analysis_settings':dict({'none':None})}))
```

To remove from analyzer:

```
analyzer.remove_analysis('area_fluctuation_1')
```

Output Info:

Retrieve output after running analyses:

```
output = analyzer.get_analysis_data('area_fluctuation_1')
```

The output is type `<type 'numpy.ndarray'>`

9.6.4 Related analyses

- acm
- ac

9.6.5 References

None

9.7 com_lateral_rdf - Lipid-lipid RDF in the bilayer lateral plane.

9.7.1 BilayerAnalyzer analysis: `com_lateral_rdf` - Lipid-lipid RDF in the bilayer lateral plane.

9.7.2 Description

Estimate the 2-d radial pair distribution function in the bilayer lateral plane using the lipid centers of mass.

This analysis protocol uses the 'com_frame' representation.

This protocol is identified by the analysis key: 'com_lateral_rdf'

Initiated by instance of:

```
<class 'pybilt.bilayer_analyzer.analysis_protocols.COMLateralRDFProtocol'>
```

9.7.3 Syntax

```
com_lateral_rdf analysis-ID keyword value
```

- com_lateral_rdf = analysis-Key - keyword/name for this analysis.
- analysis-ID = The unique name/ID being assigned to this analysis.
- keyword value = settings keyword value pairs
  - n_bins (int): Specifies the number of bins to use when estimating the RDF. Default: 25

- range_outer (float): Specify the outer distance cutoff for the RDF. Default: 25.0
- leaflet (str: 'both', 'upper', or 'lower'): Specifies the bilayer leaflet to include in the estimate. Default: 'both'
- range_inner (float): Specify the inner distance cutoff for the RDF. Default: 0.0
- resname_2 (str): Specify the resname of the target lipid type to include in this analysis. Default: 'first', the first lipid in the list pulled from the com_frame representation.
- resname_1 (str): Specify the resname of the reference lipid type to include in this analysis. Default: 'first', the first lipid in the list pulled from the com_frame representation.

Examples

Construct analyzer:

```
analyzer = BilayerAnalyzer(structure='name_of_structure_file',
                           trajectory='name_of_traj_file',
                           selection='resname POPC DOPC')
```

Add by string - use default settings:

```
analyzer.add_analysis('com_lateral_rdf com_lateral_rdf_1')
```

Add by string - adjust a setting:

```
analyzer.add_analysis('com_lateral_rdf com_lateral_rdf_1 n_bins 25')
```

Add by list:

```
analyzer.add_analysis(list(['com_lateral_rdf', 'com_lateral_rdf_1', dict({'n_
↪bins':25})]))
```

Add by dict:

```
analyzer.add_analysis(dict({'analysis_key': 'com_lateral_rdf', 'analysis_id
↪': 'com_lateral_rdf_1','analysis_settings':dict({'n_bins':25})}))
```

To remove from analyzer:

```
analyzer.remove_analysis('com_lateral_rdf_1')
```

Output Info:

Retrieve output after running analyses:

```
output = analyzer.get_analysis_data('com_lateral_rdf_1')
```

The output is type `<type 'tuple'>`

9.7.4 Related analyses

- nnf

9.7.5 References

1. Microsecond Molecular Dynamics Simulations of Lipid Mixing Chunkit Hong, D. Peter Tieleman, and Yi Wang Langmuir 2014 30 (40), 11993-12001 DOI: 10.1021/la502363b http://pubs.acs.org/doi/abs/10.1021/la502363b

9.8 dc_cluster - Hiearchical clustering of lipids based on distance.

9.8.1 BilayerAnalyzer analysis: `dc_cluster` - Hiearchical clustering of lipids based on distance.

9.8.2 Description

Compute lipid clusters using a hiearchical distance based method.

This analysis uses a type of hiearchical clustering where points (lipid centers of mass) are are added to a cluster if they are within a specified distance of any other point within the cluster.

This protocol is identified by the analysis key: 'dc_cluster'

Initiated by instance of:

```
<class 'pybilt.bilayer_analyzer.analysis_protocols.DCClusterProtocol'>
```

9.8.3 Syntax

```
dc_cluster analysis-ID keyword value
```

- dc_cluster = analysis-Key - keyword/name for this analysis.
- analysis-ID = The unique name/ID being assigned to this analysis.
- keyword value = settings keyword value pairs
  - leaflet (str: 'both', 'upper', or 'lower'): Specifies the bilayer leaflet to include in the estimate. Default: 'both'
  - resname (str): Specify the resname of the lipid type to include in this analysis. Default: 'all', includes all lipid types.
  - cutoff (float): The cutoff distance to use for the clustering. Default: 12.0

Examples

Construct analyzer:

```
analyzer = BilayerAnalyzer(structure='name_of_structure_file',
                           trajectory='name_of_traj_file',
                           selection='resname POPC DOPC')
```

Add by string - use default settings:

```
analyzer.add_analysis('dc_cluster dc_cluster_1')
```

Add by string - adjust a setting:

```
analyzer.add_analysis('dc_cluster dc_cluster_1 resname POPC')
```

Add by list:

```
analyzer.add_analysis(list(['dc_cluster', 'dc_cluster_1', dict({'resname':
↪'POPC'})]))
```

Add by dict:

```
analyzer.add_analysis(dict({'analysis_key': 'dc_cluster', 'analysis_id': 'dc_
↪cluster_1','analysis_settings':dict({'resname':'POPC'})}))
```

To remove from analyzer:

```
analyzer.remove_analysis('dc_cluster_1')
```

Output Info:

Retrieve output after running analyses:

```
output = analyzer.get_analysis_data('dc_cluster_1')
```

The output is type `<type 'dict'>`

9.8.4 Related analyses

- None

9.8.5 References

None

9.9 disp_vec_corr_avg - Weighted average of displacement vector correlations.

9.9.1 BilayerAnalyzer analysis: `disp_vec_corr_avg` - Weighted average of displacement vector correlations.

9.9.2 Description

Compute the pair-wise cross correlation between pairs of the displacement vectors for each lipid in the specified leaflet(s) of bilayer and do a inverse-distance weighted averaging.

This analysis computes the displacement vectors as in DispVecProtocol, but then continues to compute the pair-wise cross correlations between each vector (i.e. the cos(theta) for the angle theta between the

vectors) and averages the values using the inverse of the distance between the vector starting points as a weight.

This protocol is identified by the analysis key: 'disp_vec_corr_avg'

Initiated by instance of:

```
<class 'pybilt.bilayer_analyzer.analysis_protocols.
↪DispVecCorrelationAverageProtocol'>
```

9.9.3 Syntax

```
disp_vec_corr_avg analysis-ID keyword value
```

- disp_vec_corr_avg = analysis-Key - keyword/name for this analysis.
- analysis-ID = The unique name/ID being assigned to this analysis.
- keyword value = settings keyword value pairs
  - leaflet (str: 'both', 'upper', or 'lower'): Specifies the bilayer leaflet to include in the estimate. Default: 'both'
  - resname (str): Specify the resname of the lipid type to include in this analysis. Default: 'all', includes all lipid types.
  - interval (int): Sets the frame interval over which to compute the displacement vectors.
  - wrapped (bool): Specify whether to use the wrapped ('True') or un-wrapped ('False') coordintes for the base of the vectors. Default: False

Examples

Construct analyzer:

```
analyzer = BilayerAnalyzer(structure='name_of_structure_file',
                           trajectory='name_of_traj_file',
                           selection='resname POPC DOPC')
```

Add by string - use default settings:

```
analyzer.add_analysis('disp_vec_corr_avg disp_vec_corr_avg_1')
```

Add by string - adjust a setting:

```
analyzer.add_analysis('disp_vec_corr_avg disp_vec_corr_avg_1 leaflet both')
```

Add by list:

```
analyzer.add_analysis(list(['disp_vec_corr_avg', 'disp_vec_corr_avg_1', dict(
↪{'leaflet':'both'})]))
```

Add by dict:

```
analyzer.add_analysis(dict({'analysis_key': 'disp_vec_corr_avg', 'analysis_id
↪': 'disp_vec_corr_avg_1','analysis_settings':dict({'leaflet':'both'})}))
```

To remove from analyzer:

```
analyzer.remove_analysis('disp_vec_corr_avg_1')
```

Output Info:

Retrieve output after running analyses:

```
output = analyzer.get_analysis_data('disp_vec_corr_avg_1')
```

The output is type `<type 'numpy.ndarray'>`

9.9.4 Related analyses

- disp_vec
- disp_vec_corr
- disp_vec_nncorr

9.9.5 References

None

9.10 disp_vec_corr - Displacement vector correlation matrix.

9.10.1 BilayerAnalyzer analysis: `disp_vec_corr` - Displacement vector correlation matrix.

9.10.2 Description

Comute the pair-wise cross correlation matrix for the displacement vectors for each lipid in the specified leaflet(s) of bilayer.

This analysis computes the displacement vectors as in 'disp_vec' analysis (DispVecProtocol), but then continues to compute the pair-wise cross correlation matrix between each vector. i.e. the cos(theta) for the angle theta between the vectors.

This protocol is identified by the analysis key: 'disp_vec_corr'

Initiated by instance of:

```
<class 'pybilt.bilayer_analyzer.analysis_protocols.DispVecCorrelationProtocol
↪'>
```

9.10.3 Syntax

```
disp_vec_corr analysis-ID keyword value
```

- disp_vec_corr = analysis-Key - keyword/name for this analysis.
- analysis-ID = The unique name/ID being assigned to this analysis.
- keyword value = settings keyword value pairs
  - leaflet (str: 'both', 'upper', or 'lower'): Specifies the bilayer leaflet to include in the estimate. Default: 'both'
  - resname (str): Specify the resname of the lipid type to include in this analysis. Default: 'all', includes all lipid types.
  - interval (int): Sets the frame interval over which to compute the displacement vectors. f
  - wrapped (bool): Specify whether to use the wrapped ('True') or un-wrapped ('False') coordintes for the base of the vectors. Default: False

Examples

Construct analyzer:

```
analyzer = BilayerAnalyzer(structure='name_of_structure_file',
                           trajectory='name_of_traj_file',
                           selection='resname POPC DOPC')
```

Add by string - use default settings:

```
analyzer.add_analysis('disp_vec_corr disp_vec_corr_1')
```

Add by string - adjust a setting:

```
analyzer.add_analysis('disp_vec_corr disp_vec_corr_1 leaflet both')
```

Add by list:

```
analyzer.add_analysis(list(['disp_vec_corr', 'disp_vec_corr_1', dict({
↪'leaflet':'both'})]))
```

Add by dict:

```
analyzer.add_analysis(dict({'analysis_key': 'disp_vec_corr', 'analysis_id':
↪'disp_vec_corr_1','analysis_settings':dict({'leaflet':'both'})}))
```

To remove from analyzer:

```
analyzer.remove_analysis('disp_vec_corr_1')
```

Output Info:

Retrieve output after running analyses:

```
output = analyzer.get_analysis_data('disp_vec_corr_1')
```

The output is type <type 'list'>

9.10.4 Related analyses

- disp_vec
- disp_vec_nncorr
- disp_vec_corr_avg
- spatial_velocity_corr

9.10.5 References

None

9.11 disp_vec_nncorr - Displacement vector nearest neigbor correlations.

9.11.1 BilayerAnalyzer analysis: `disp_vec_nncorr` - Displacement vector nearest neigbor correlations.

9.11.2 Description

Comute the pair-wise cross correlations for the displacement vectors for each lipid in the specified leaflet(s) of bilayer and its nearest neighbor.

This analysis computes the displacement vectors as in the 'disp_vec' analysis (DispVecProtocol), but then continues to compute the pair-wise cross correlation between each vector and its nearest neighbor. i.e. the cos(theta) for the angle theta between the vectors.

This protocol is identified by the analysis key: 'disp_vec_nncorr'

Initiated by instance of:

```
<class 'pybilt.bilayer_analyzer.analysis_protocols.
↪DispVecNNCorrelationProtocol'>
```

9.11.3 Syntax

```
disp_vec_nncorr analysis-ID keyword value
```

- disp_vec_nncorr = analysis-Key - keyword/name for this analysis.
- analysis-ID = The unique name/ID being assigned to this analysis.
- keyword value = settings keyword value pairs
  - leaflet (str: 'both', 'upper', or 'lower'): Specifies the bilayer leaflet to include in the estimate. Default: 'both'
  - resname (str): Specify the resname of the lipid type to include in this analysis. Default: 'all', includes all lipid types.
  - interval (int): Sets the frame interval over which to compute the displacement vectors. f

– wrapped (bool): Specify whether to use the wrapped ('True') or un-wrapped ('False') coordintes for the base of the vectors. Default: False

Examples

Construct analyzer:

```
analyzer = BilayerAnalyzer(structure='name_of_structure_file',
                           trajectory='name_of_traj_file',
                           selection='resname POPC DOPC')
```

Add by string - use default settings:

```
analyzer.add_analysis('disp_vec_nncorr disp_vec_nncorr_1')
```

Add by string - adjust a setting:

```
analyzer.add_analysis('disp_vec_nncorr disp_vec_nncorr_1 leaflet both')
```

Add by list:

```
analyzer.add_analysis(list(['disp_vec_nncorr', 'disp_vec_nncorr_1', dict({
↪'leaflet':'both'})]))
```

Add by dict:

```
analyzer.add_analysis(dict({'analysis_key': 'disp_vec_nncorr', 'analysis_id
↪': 'disp_vec_nncorr_1','analysis_settings':dict({'leaflet':'both'})}))
```

To remove from analyzer:

```
analyzer.remove_analysis('disp_vec_nncorr_1')
```

Output Info:

Retrieve output after running analyses:

```
output = analyzer.get_analysis_data('disp_vec_nncorr_1')
```

The output is type `<type 'list'>`

9.11.4 Related analyses

- disp_vec
- disp_vec_corr
- disp_vec_corr_avg
- spatial_velocity_corr

9.11.5 References

None

9.12 disp_vec - Displacement vectors.

9.12.1 BilayerAnalyzer analysis: `disp_vec` - Displacement vectors.

9.12.2 Description

Comute displacement vectors for each lipid in the specified leaflet(s) of bilayer.

This protocol is identified by the analysis key: 'disp_vec'

Initiated by instance of:

```
<class 'pybilt.bilayer_analyzer.analysis_protocols.DispVecProtocol'>
```

9.12.3 Syntax

```
disp_vec analysis-ID keyword value
```

- disp_vec = analysis-Key - keyword/name for this analysis.
- analysis-ID = The unique name/ID being assigned to this analysis.
- keyword value = settings keyword value pairs
  - scale (bool): Specify whether to scale the coordinates by the box dimensions of the reference frame. Default: False
  - interval (int): Sets the frame interval over which to compute the displacement vectors.
  - leaflet (str: 'both', 'upper', or 'lower'): Specifies the bilayer leaflet to include in the estimate. Default: 'both'
  - wrapped (bool): Specify whether to use the wrapped ('True') or un-wrapped ('False') coordintes for the base of the vectors. Default: False
  - resname (str): Specify the resname of the lipid type to include in this analysis. Default: 'all', includes all lipid types.
  - scale_to_max (bool): Specify whether to scale the coordinates by the box dimensions of the maximum box size in the anlysis. Default: False.

Examples

Construct analyzer:

```
analyzer = BilayerAnalyzer(structure='name_of_structure_file',
                           trajectory='name_of_traj_file',
                           selection='resname POPC DOPC')
```

Add by string - use default settings:

```
analyzer.add_analysis('disp_vec disp_vec_1')
```

Add by string - adjust a setting:

```
analyzer.add_analysis('disp_vec disp_vec_1 scale False')
```

Add by list:

```
analyzer.add_analysis(list(['disp_vec', 'disp_vec_1', dict({'scale':False}
↪)]))
```

Add by dict:

```
analyzer.add_analysis(dict({'analysis_key': 'disp_vec', 'analysis_id': 'disp_
↪vec_1','analysis_settings':dict({'scale':False})}))
```

To remove from analyzer:

```
analyzer.remove_analysis('disp_vec_1')
```

Output Info:

Retrieve output after running analyses:

```
output = analyzer.get_analysis_data('disp_vec_1')
```

The output is type `<type 'list'>`

9.12.4 Related analyses

- disp_vec_corr
- disp_vec_nncorr
- disp_vec_corr_avg
- spatial_velocity_corr

9.12.5 References

1. Emma Falck, Tomasz Rog, Mikko Karttunen, and Ilpo Vattulainen, Lateral Diffusion in Lipid Membranes through Collective Flows, Journal of the American Chemical Society, 2008 130 (1), 44-45 DOI: 10.1021/ja7103558

9.13 flip_flop - Count lipid flip flops.

9.13.1 BilayerAnalyzer analysis: `flip_flop` - Count lipid flip flops.

9.13.2 Description

Count any lipid flips flops between the leaflets.

This protocol is identified by the analysis key: 'flip_flop'

Initiated by instance of:

```
<class 'pybilt.bilayer_analyzer.analysis_protocols.FlipFlopProtocol'>
```

9.13.3 Syntax

```
flip_flop analysis-ID
```

- flip_flop = analysis-Key - keyword/name for this analysis.
- analysis-ID = The unique name/ID being assigned to this analysis.

Examples

Construct analyzer:

```
analyzer = BilayerAnalyzer(structure='name_of_structure_file',
                           trajectory='name_of_traj_file',
                           selection='resname POPC DOPC')
```

Add by string - use default settings:

```
analyzer.add_analysis('flip_flop flip_flop_1')
```

Add by string - adjust a setting:

```
analyzer.add_analysis('flip_flop flip_flop_1 none None')
```

Add by list:

```
analyzer.add_analysis(list(['flip_flop', 'flip_flop_1', dict({'none':None}
↪)]))
```

Add by dict:

```
analyzer.add_analysis(dict({'analysis_key': 'flip_flop', 'analysis_id':
↪'flip_flop_1','analysis_settings':dict({'none':None})}))
```

To remove from analyzer:

```
analyzer.remove_analysis('flip_flop_1')
```

Output Info:

Retrieve output after running analyses:

```
output = analyzer.get_analysis_data('flip_flop_1')
```

The output is type `<type 'dict'>`

9.13.4 Related analyses

- None

9.13.5 References

1. Andrey A. Gurtovenko, and Ilpo Vattulainen, Molecular Mechanism for Lipid Flip-Flops, The Journal of Physical Chemistry B, 2007 111 (48), 13554-13559, DOI: 10.1021/jp077094k http://pubs.acs.org/doi/abs/10.1021/jp077094k?journalCode=jpcbfk
2. Nicolas Sapay, W. F. Drew Bennett, and D. Peter Tieleman, Molecular Simulations of Lipid Flip-Flop in the Presence of Model Transmembrane Helices, Biochemistry, 2010 49 (35), 7665-7673, DOI: 10.1021/bi100878q http://pubs.acs.org/doi/abs/10.1021/bi100878q

9.14 halperin_nelson - Halperin and Nelson's rotational invariant.

9.14.1 BilayerAnalyzer analysis: `halperin_nelson` - Halperin and Nelson's rotational invariant.

9.14.2 Description

Estimate the mean hexagonal packing orientation parameter.

This analysis implements Halperin and Nelson's rotational invariant to estimate the hexagonal packing orientation parameter. The value for lipid l is given by phi_l = | (1/6) * sum_{j element nn(l)} exp(6i*theta_{lj}) |^2 , where i is complex and nn(l) are the 6 nearest neighbors of lipid l; theta_{lj} is the angle between the vector formed by beads representing lipid l and j and an arbitrary axis. The value is unity for perfect hexagonal packing, and it is zero to the extent that hexagonal packing is entirely absent. This protocol uses the 'com_frame' representation of the bilayer.

This protocol is identified by the analysis key: 'halperin_nelson'

Initiated by instance of:

```
<class 'pybilt.bilayer_analyzer.analysis_protocols.HalperinNelsonProtocol'>
```

9.14.3 Syntax

```
halperin_nelson analysis-ID keyword value
```

- halperin_nelson = analysis-Key - keyword/name for this analysis.
- analysis-ID = The unique name/ID being assigned to this analysis.
- keyword value = settings keyword value pairs
  - leaflet (str: 'upper', or 'lower'): Specifies the bilayer leaflet to include in the estimate. Default: 'upper'

Examples

Construct analyzer:

```
analyzer = BilayerAnalyzer(structure='name_of_structure_file',
                           trajectory='name_of_traj_file',
                           selection='resname POPC DOPC')
```

Add by string - use default settings:

```
analyzer.add_analysis('halperin_nelson halperin_nelson_1')
```

Add by string - adjust a setting:

```
analyzer.add_analysis('halperin_nelson halperin_nelson_1 leaflet upper')
```

Add by list:

```
analyzer.add_analysis(list(['halperin_nelson', 'halperin_nelson_1', dict({
↪'leaflet':'upper'})]))
```

Add by dict:

```
analyzer.add_analysis(dict({'analysis_key': 'halperin_nelson', 'analysis_id
↪': 'halperin_nelson_1','analysis_settings':dict({'leaflet':'upper'})}))
```

To remove from analyzer:

```
analyzer.remove_analysis('halperin_nelson_1')
```

Output Info:

Retrieve output after running analyses:

```
output = analyzer.get_analysis_data('halperin_nelson_1')
```

The output is type `<type 'numpy.ndarray'>`

9.14.4 Related analyses

- None

9.14.5 References

1. Shachi Katira, Kranthi K. Mandadapu, Suriyanarayanan Vaikuntanathan, Berend Smit, and David Chandler, The order-disorder transition in model lipid bilayers is a first-order hexatic to liquid phase transition, arXiv preprint [cond-mat.soft] 2015, arXiv:1506.04310. https://arxiv.org/pdf/1506.04310.pdf

9.15 lipid_collinearity - Lipid-lipid collinearity.

9.15.1 BilayerAnalyzer analysis: `lipid_collinearity` - Lipid-lipid collinearity.

9.15.2 Description

Estimate the lipid-lipid collinearity angles.

This analysis computes the mean lipid-lipid collinearity angle (or order parameter) using the vector representation of the specified lipids.

This protocol is identified by the analysis key: 'lipid_collinearity'

Initiated by instance of:

```
<class 'pybilt.bilayer_analyzer.analysis_protocols.LipidCollinearityProtocol
↪'>
```

9.15.3 Syntax

```
lipid_collinearity analysis-ID keyword value
```

- lipid_collinearity = analysis-Key - keyword/name for this analysis.
- analysis-ID = The unique name/ID being assigned to this analysis.
- keyword value = settings keyword value pairs
  - leaflet (str: 'upper', or 'lower'): Specifies the bilayer leaflet to include in the estimate. Default: 'upper'
  - style (str: 'angle', 'order'): Specify whether to compute the tilt angle ('angle') or the tilt angle order parameter ('order'). Default: 'angle'
  - resname_2 (str): Specify the resname of the target lipid type to include in this analysis. Default: 'first', the first lipid in the list pulled from the com_frame representation.
  - resname_1 (str): Specify the resname of the reference lipid type to include in this analysis. Default: 'first', the first lipid in the list pulled from the com_frame representation.

Examples

Construct analyzer:

```
analyzer = BilayerAnalyzer(structure='name_of_structure_file',
                           trajectory='name_of_traj_file',
                           selection='resname POPC DOPC')
```

Add by string - use default settings:

```
analyzer.add_analysis('lipid_collinearity lipid_collinearity_1')
```

Add by string - adjust a setting:

```
analyzer.add_analysis('lipid_collinearity lipid_collinearity_1 leaflet upper
↪')
```

Add by list:

```
analyzer.add_analysis(list(['lipid_collinearity', 'lipid_collinearity_1',␣
↪dict({'leaflet':'upper'})]))
```

Add by dict:

```
analyzer.add_analysis(dict({'analysis_key': 'lipid_collinearity', 'analysis_
↪id': 'lipid_collinearity_1','analysis_settings':dict({'leaflet':'upper'})}
↪))
```

To remove from analyzer:

```
analyzer.remove_analysis('lipid_collinearity_1')
```

Output Info:

Retrieve output after running analyses:

```
output = analyzer.get_analysis_data('lipid_collinearity_1')
```

The output is type `<type 'numpy.ndarray'>`

9.15.4 Related analyses

- lipid_tilt

9.15.5 References

1. Anton O. Chugunov, Pavel E. Volynsky, Nikolay A. Krylov, Ivan A. Boldyrev, and Roman G. Efremov, Liquid but Durable: Molecular Dynamics Simulations Explain the Unique Properties of Archaeal-Like Membranes, Scientific Reports, 4:7462, 2014, doi:10.1038/srep07462 (https://www.nature.com/articles/srep07462)

9.16 lipid_length - Lipid length using the lipid vectors.

9.16.1 BilayerAnalyzer analysis: `lipid_length` - Lipid length using the lipid vectors.

9.16.2 Description

Estimate the lipids length using the defined lipid vector.

This analysis is used to compute the mean lipid length using the vector represetation of the specified lipids.

This protocol is identified by the analysis key: 'lipid_length'

Initiated by instance of:

```
<class 'pybilt.bilayer_analyzer.analysis_protocols.LipidLengthProtocol'>
```

9.16.3 Syntax

```
lipid_length analysis-ID keyword value
```

- lipid_length = analysis-Key - keyword/name for this analysis.
- analysis-ID = The unique name/ID being assigned to this analysis.
- keyword value = settings keyword value pairs
  - leaflet (str: 'both', 'upper', or 'lower'): Specifies the bilayer leaflet to include in the estimate. Default: 'both'
  - resname (str): Specify the resname of the lipid type to include in this analysis. Default: 'all', averages over all lipid types.

Examples

Construct analyzer:

```
analyzer = BilayerAnalyzer(structure='name_of_structure_file',
                           trajectory='name_of_traj_file',
                           selection='resname POPC DOPC')
```

Add by string - use default settings:

```
analyzer.add_analysis('lipid_length lipid_length_1')
```

Add by string - adjust a setting:

```
analyzer.add_analysis('lipid_length lipid_length_1 leaflet both')
```

Add by list:

```
analyzer.add_analysis(list(['lipid_length', 'lipid_length_1', dict({'leaflet
↪':'both'})]))
```

Add by dict:

```
analyzer.add_analysis(dict({'analysis_key': 'lipid_length', 'analysis_id':
↪'lipid_length_1','analysis_settings':dict({'leaflet':'both'})}))
```

To remove from analyzer:

```
analyzer.remove_analysis('lipid_length_1')
```

Output Info:

Retrieve output after running analyses:

```
output = analyzer.get_analysis_data('lipid_length_1')
```

The output is type <type 'numpy.ndarray'>

9.16.4 Related analyses

- None

9.16.5 References

1. Anton O. Chugunov, Pavel E. Volynsky, Nikolay A. Krylov, Ivan A. Boldyrev, and Roman G. Efremov, Liquid but Durable: Molecular Dynamics Simulations Explain the Unique Properties of Archaeal-Like Membranes, Scientific Reports, 4:7462, 2014, doi:10.1038/srep07462 (https://www.nature.com/articles/srep07462)

9.17 lipid_tilt - Lipid tilt using the lipid vectors.

9.17.1 BilayerAnalyzer analysis: `lipid_tilt` - Lipid tilt using the lipid vectors.

9.17.2 Description

Estimate the lipids tilt angles using the defined lipid vector.

This analyis estimates the mean lipid tilt using the vector represetation of the specified lipids in reference to a particular axis, typically the bilayer normal.

This protocol is identified by the analysis key: 'lipid_tilt'

Initiated by instance of:

```
<class 'pybilt.bilayer_analyzer.analysis_protocols.LipidTiltProtocol'>
```

9.17.3 Syntax

```
lipid_tilt analysis-ID keyword value
```

- lipid_tilt = analysis-Key - keyword/name for this analysis.
- analysis-ID = The unique name/ID being assigned to this analysis.
- keyword value = settings keyword value pairs
  - leaflet (str: 'upper', or 'lower'): Specifies the bilayer leaflet to include in the estimate. Default: 'upper'
  - resname (str): Specify the resname of the lipid type to include in this analysis. Default: 'all', averages over all lipid types.
  - style (str: 'angle', 'order'): Specify whether to compute the tilt angle ('angle') or the tilt angle order parameter ('order'). Default: 'angle'
  - ref_axis (str: 'x', 'y', or 'z'): Specify the reference axis that should be used to estimate the tilt. This is typically the axis along the bilayer normal. Default: 'z'

Examples

Construct analyzer:

```
analyzer = BilayerAnalyzer(structure='name_of_structure_file',
                           trajectory='name_of_traj_file',
                           selection='resname POPC DOPC')
```

Add by string - use default settings:

```
analyzer.add_analysis('lipid_tilt lipid_tilt_1')
```

Add by string - adjust a setting:

```
analyzer.add_analysis('lipid_tilt lipid_tilt_1 leaflet upper')
```

Add by list:

```
analyzer.add_analysis(list(['lipid_tilt', 'lipid_tilt_1', dict({'leaflet':
↪'upper'})]))
```

Add by dict:

```
analyzer.add_analysis(dict({'analysis_key': 'lipid_tilt', 'analysis_id':
↪'lipid_tilt_1','analysis_settings':dict({'leaflet':'upper'})}))
```

To remove from analyzer:

```
analyzer.remove_analysis('lipid_tilt_1')
```

Output Info:

Retrieve output after running analyses:

```
output = analyzer.get_analysis_data('lipid_tilt_1')
```

The output is type `<type 'numpy.ndarray'>`

9.17.4 Related analyses

- loa
- lipid_collinearity

9.17.5 References

1. Anton O. Chugunov, Pavel E. Volynsky, Nikolay A. Krylov, Ivan A. Boldyrev, and Roman G. Efremov, Liquid but Durable: Molecular Dynamics Simulations Explain the Unique Properties of Archaeal-Like Membranes, Scientific Reports, 4:7462, 2014, doi:10.1038/srep07462 (https://www.nature.com/articles/srep07462)

9.18 loa - Lateral lipid orientation angle.

9.18.1 BilayerAnalyzer analysis: `loa` - Lateral lipid orientation angle.

9.18.2 Description

Estimate the angle for lipid orientation vectors relative to the bilayer normal.

This analysis is based on the P-N vector-normal angle analysis discussed in Ref. 1. Using two reference atoms from the specified lipid type a vector for each lipid is computed and the orientation relative to the bilayer normal is computed, i.e. theta, or the angle between the two vectors.

This protocol is identified by the analysis key: 'loa'

Initiated by instance of:

```
<class 'pybilt.bilayer_analyzer.analysis_protocols.
↪LateralOrientationAngleProtocol'>
```

9.18.3 Syntax

```
loa analysis-ID keyword value
```

- loa = analysis-Key - keyword/name for this analysis.
- analysis-ID = The unique name/ID being assigned to this analysis.
- keyword value = settings keyword value pairs
  - leaflet (str: 'both', 'upper', or 'lower'): Specifies the bilayer leaflet to include in the estimate. Default: 'both'
  - resname (str): Specify the resname of the lipid type to include in this analysis. Default: 'first', the first lipid type as stored in the com_frame.
  - ref_atom_2 (str): The atom name of the reference atom to use as the head of the lipid orientation vector.
  - ref_atom_1 (str): The atom name of the reference atom to use as the base of the lipid orientation vector.

Examples

Construct analyzer:

```
analyzer = BilayerAnalyzer(structure='name_of_structure_file',
                           trajectory='name_of_traj_file',
                           selection='resname POPC DOPC')
```

Add by string - use default settings:

```
analyzer.add_analysis('loa loa_1')
```

Add by string - adjust a setting:

```
analyzer.add_analysis('loa loa_1 resname POPC')
```

Add by list:

```
analyzer.add_analysis(list(['loa', 'loa_1', dict({'resname':'POPC'})]))
```

Add by dict:

```
analyzer.add_analysis(dict({'analysis_key': 'loa', 'analysis_id': 'loa_1',
↪'analysis_settings':dict({'resname':'POPC'})}))
```

To remove from analyzer:

```
analyzer.remove_analysis('loa_1')
```

Output Info:

Retrieve output after running analyses:

```
output = analyzer.get_analysis_data('loa_1')
```

The output is type `<type 'numpy.ndarray'>`

9.18.4 Related analyses

- lop
- lipid_tilt

9.18.5 References

1. Zheng Li, Richard M. Venable, Laura A. Rogers, Diana Murray, and Richard W. Pastor, "Molecular Dynamics Simulations of PIP2 and PIP3 in Lipid Bilayers: Determination of Ring Orientation, and the Effects of Surface Roughness on a Poisson-Boltzmann Description", Biophys J. 2009 Jul 8; 97(1): 155-163. doi: 10.1016/j.bpj.2009.04.037

9.19 lop - Lateral lipid orientation parameter.

9.19.1 BilayerAnalyzer analysis: `lop` - Lateral lipid orientation parameter.

9.19.2 Description

Estimate the order parameter for lipid orientation vectors relative to the bilayer normal.

This analysis is based on the P-N vector-normal angle analysis discussed in Ref. 1. Using two reference atoms from the specified lipid type a vector for each lipid is computed and the orientation relative to the bilayer normal is computed, i.e. cos(theta), for the angle between the two vectors.

This protocol is identified by the analysis key: 'lop'

Initiated by instance of:

```
<class 'pybilt.bilayer_analyzer.analysis_protocols.
↪LateralOrientationParameterProtocol'>
```

9.19.3 Syntax

```
lop analysis-ID keyword value
```

- lop = analysis-Key - keyword/name for this analysis.
- analysis-ID = The unique name/ID being assigned to this analysis.
- keyword value = settings keyword value pairs
  - leaflet (str: 'both', 'upper', or 'lower'): Specifies the bilayer leaflet to include in the estimate. Default: 'both'
  - resname (str): Specify the resname of the lipid type to include in this analysis. Default: 'first', the first lipid type as stored in the com_frame.
  - ref_atom_2 (str): The atom name of the reference atom to use as the head of the lipid orientation vector.
  - ref_atom_1 (str): The atom name of the reference atom to use as the base of the lipid orientation vector.

Examples

Construct analyzer:

```
analyzer = BilayerAnalyzer(structure='name_of_structure_file',
                           trajectory='name_of_traj_file',
                           selection='resname POPC DOPC')
```

Add by string - use default settings:

```
analyzer.add_analysis('lop lop_1')
```

Add by string - adjust a setting:

```
analyzer.add_analysis('lop lop_1 resname POPC')
```

Add by list:

```
analyzer.add_analysis(list(['lop', 'lop_1', dict({'resname':'POPC'})]))
```

Add by dict:

```
analyzer.add_analysis(dict({'analysis_key': 'lop', 'analysis_id': 'lop_1',
↪'analysis_settings':dict({'resname':'POPC'})}))
```

To remove from analyzer:

```
analyzer.remove_analysis('lop_1')
```

Output Info:

Retrieve output after running analyses:

```
output = analyzer.get_analysis_data('lop_1')
```

The output is type <type 'numpy.ndarray'>

9.19.4 Related analyses

- loa
- lipid_tilt

9.19.5 References

1. Zheng Li, Richard M. Venable, Laura A. Rogers, Diana Murray, and Richard W. Pastor, "Molecular Dynamics Simulations of PIP2 and PIP3 in Lipid Bilayers: Determination of Ring Orientation, and the Effects of Surface Roughness on a Poisson-Boltzmann Description", Biophys J. 2009 Jul 8; 97(1): 155-163. doi: 10.1016/j.bpj.2009.04.037

9.20 mass_dens - Mass density profile.

9.20.1 BilayerAnalyzer analysis: `mass_dens` - Mass density profile.

9.20.2 Description

Estimate the mass density profile for the specified selection.

This protocol is used to estimate the 1-dimensional mass density profile for a selection of atoms along the bilayer normal. The profile is automatically centered on the bilayer's center of mass along the bilayer normal.

This protocol is identified by the analysis key: 'mass_dens'

Initiated by instance of:

```
<class 'pybilt.bilayer_analyzer.analysis_protocols.MassDensProtocol'>
```

9.20.3 Syntax

```
mass_dens analysis-ID keyword value
```

- mass_dens = analysis-Key - keyword/name for this analysis.
- analysis-ID = The unique name/ID being assigned to this analysis.
- keyword value = settings keyword value pairs
  - n_bins (int): Set the number of bins to divide the normal dimensions into for binning. Defalt: 25

- selection_string (str): Provide the MDAnalysis compatible selection for the atoms to include in this analysis. Default: 'BILAYER', use all the lipids of the bilayer as recovered from the selection given to the external BilayerAnalyzer.

Examples

Construct analyzer:

```
analyzer = BilayerAnalyzer(structure='name_of_structure_file',
                           trajectory='name_of_traj_file',
                           selection='resname POPC DOPC')
```

Add by string - use default settings:

```
analyzer.add_analysis('mass_dens mass_dens_1')
```

Add by string - adjust a setting:

```
analyzer.add_analysis('mass_dens mass_dens_1 n_bins 25')
```

Add by list:

```
analyzer.add_analysis(list(['mass_dens', 'mass_dens_1', dict({'n_bins':25}
↪)]))
```

Add by dict:

```
analyzer.add_analysis(dict({'analysis_key': 'mass_dens', 'analysis_id':
↪'mass_dens_1','analysis_settings':dict({'n_bins':25})}))
```

To remove from analyzer:

```
analyzer.remove_analysis('mass_dens_1')
```

Output Info:

Retrieve output after running analyses:

```
output = analyzer.get_analysis_data('mass_dens_1')
```

The output is type `<type 'tuple'>`

9.20.4 Related analyses

- None

9.20.5 References

None

9.21 msd_dist - Mean squared displacement as a function of distance.

9.21.1 BilayerAnalyzer analysis: `msd_dist` - Mean squared displacement as a function of distance.

9.21.2 Description

Compute the MSD over a set time interval as function of nearest distance.

This protocol is identified by the analysis key: 'disp_vec'

Initiated by instance of:

```
<class 'pybilt.bilayer_analyzer.analysis_protocols.MSDDistProtocol'>
```

9.21.3 Syntax

```
msd_dist analysis-ID keyword value
```

- msd_dist = analysis-Key - keyword/name for this analysis.
- analysis-ID = The unique name/ID being assigned to this analysis.
- keyword value = settings keyword value pairs
  - range_inner (float): Specify the inner distance cutoff for the RDF. Default: 0.0
  - n_bins (int): Specifies the number of bins to use when estimating the RDF. Default: 25
  - interval (int): Sets the frame interval over which to compute the displacement vectors.
  - range_outer (float): Specify the outer distance cutoff for the RDF. Default: 25.0
  - leaflet (str: 'both', 'upper', or 'lower'): Specifies the bilayer leaflet to include in the estimate. Default: 'both'
  - wrapped (bool): Specify whether to use the wrapped ('True') or un-wrapped ('False') coordintes for the base of the vectors. Default: False
  - resname (str): Specify the resname of the lipid type to include in this analysis. Default: 'all', includes all lipid types.
  - resname_2 (str): Specify the resname of the target lipid type to include in this analysis. Special names are 'first' and 'all', which use the first and all lipid types respectively. Default: 'first', the first lipid in the list pulled from the com_frame representation.
  - resname_1 (str): Specify the resname of the reference lipid type to include in this analysis. Special names are 'first' and 'all', which use the first and all lipid types respectively. Default: 'first', the first lipid in the list pulled from the com_frame representation.

Examples

Construct analyzer:

```
analyzer = BilayerAnalyzer(structure='name_of_structure_file',
                           trajectory='name_of_traj_file',
                           selection='resname POPC DOPC')
```

Add by string - use default settings:

```
analyzer.add_analysis('msd_dist msd_dist_1')
```

Add by string - adjust a setting:

```
analyzer.add_analysis('msd_dist msd_dist_1 n_bins 25')
```

Add by list:

```
analyzer.add_analysis(list(['msd_dist', 'msd_dist_1', dict({'n_bins':25})]))
```

Add by dict:

```
analyzer.add_analysis(dict({'analysis_key': 'msd_dist', 'analysis_id': 'msd_
↪dist_1','analysis_settings':dict({'n_bins':25})}))
```

To remove from analyzer:

```
analyzer.remove_analysis('msd_dist_1')
```

Output Info:

Retrieve output after running analyses:

```
output = analyzer.get_analysis_data('msd_dist_1')
```

The output is type `<type 'tuple'>`

9.21.4 Related analyses

- disp_vec
- msd

9.21.5 References

None

9.22 msd_multi - Multiple time origin mean squared displacement.

9.22.1 BilayerAnalyzer analysis: `msd_multi` - Multiple time origin mean squared displacement.

9.22.2 Description

Estimate the mean squared displacement using multiple time origins.

This analysis computes the average mean squared displacement (MSD) of the centers of mass of the specified lipids using multiple time origins and thus multiple time blocks: MSD = <<(r(tau) - r_0)^2>_i>_tau where the inner angle brackets denote averaging over all lipids i and the outer brackets denote averaging over all time origins tau. The diffusion coefficient is estimated from the MSD using a simplified version Einstein's relation: D_i ~ MSD_i/(4.0*tau)

This protocol is identified by the analysis key: 'msd_multi'

Initiated by instance of:

```
<class 'pybilt.bilayer_analyzer.analysis_protocols.MSDMultiProtocol'>
```

9.22.3 Syntax

```
msd_multi analysis-ID keyword value
```

- msd_multi = analysis-Key - keyword/name for this analysis.
- analysis-ID = The unique name/ID being assigned to this analysis.
- keyword value = settings keyword value pairs
  - leaflet (str: 'both', 'upper', or 'lower'): Specifies the bilayer leaflet to include in the estimate. Default: 'both'
  - resname (str): Specify the resname of the lipid type to include in this analysis. Default: 'all', averages over all lipid types.
  - n_tau (int): Specify the time block size in number of frames. Default: 50 (1000 picoseconds for timestep of 2 fs and frame output ever 100 timesteps).
  - n_sigma (int): Specify the time between origins in number of frames. Default: 50.

Examples

Construct analyzer:

```
analyzer = BilayerAnalyzer(structure='name_of_structure_file',
                           trajectory='name_of_traj_file',
                           selection='resname POPC DOPC')
```

Add by string - use default settings:

```
analyzer.add_analysis('msd_multi msd_multi_1')
```

Add by string - adjust a setting:

```
analyzer.add_analysis('msd_multi msd_multi_1 leaflet both')
```

Add by list:

```
analyzer.add_analysis(list(['msd_multi', 'msd_multi_1', dict({'leaflet':'both
↪'})]))
```

Add by dict:

```
analyzer.add_analysis(dict({'analysis_key': 'msd_multi', 'analysis_id': 'msd_
↪multi_1','analysis_settings':dict({'leaflet':'both'})}))
```

To remove from analyzer:

```
analyzer.remove_analysis('msd_multi_1')
```

Output Info:

Retrieve output after running analyses:

```
output = analyzer.get_analysis_data('msd_multi_1')
```

The output is type `<type 'numpy.ndarray'>`

9.22.4 Related analyses

- msd

9.22.5 References

1. Christofer Hofsab, Erik Lindahl, and Olle Edholm, "Molecular Dynamics Simulations of Phospholipid Bilayers with Cholesterol", Biophys J. 2003 Apr; 84(4): 2192-2206. doi: 10.1016/S0006-3495(03)75025-5
2. Orsi, Mario, Julien Michel, and Jonathan W. Essex. "Coarse-grain modelling of DMPC and DOPC lipid bilayers." Journal of Physics: Condensed Matter 22.15 (2010): 155106. http://iopscience.iop.org/article/10.1088/0953-8984/22/15/155106/meta
3. Gaurav Pranami and Monica H. Lamm, Estimating Error in Diffusion Coefficients Derived from Molecular Dynamics Simulations, Journal of Chemical Theory and Computation 2015 11 (10), 4586-4592, DOI: 10.1021/acs.jctc.5b00574, http://pubs.acs.org/doi/full/10.1021/acs.jctc.5b00574

9.23 msd - Single time origin mean squared displacement.

9.23.1 BilayerAnalyzer analysis: `msd` - Single time origin mean squared displacement.

9.23.2 Description

Estimate the mean squared displacement from a single time origin.

This analysis computes the mean squared displacement (MSD) versus time of the centers of mass of the specified lipids for a single time origin (i.e. the first frame in the trajectory analysis): MSD(t) = <(r(t) - r_0)^2>_i where the angle brackets denote averaging over all lipids of type i; the lipid type used for the analysis is controlled by the `leaflet` setting.

This protocol is identified by the analysis key: 'msd'

Initiated by instance of:

```
<class 'pybilt.bilayer_analyzer.analysis_protocols.MSDProtocol'>
```

9.23.3 Syntax

```
msd analysis-ID keyword value
```

- msd = analysis-Key - keyword/name for this analysis.
- analysis-ID = The unique name/ID being assigned to this analysis.
- keyword value = settings keyword value pairs
  - leaflet (str: 'both', 'upper', or 'lower'): Specifies the bilayer leaflet to include in the estimate. Default: 'both'
  - resname (str): Specify the resname of the lipid type to include in this analysis. Default: 'all', averages over all lipid types.

Examples

Construct analyzer:

```
analyzer = BilayerAnalyzer(structure='name_of_structure_file',
                           trajectory='name_of_traj_file',
                           selection='resname POPC DOPC')
```

Add by string - use default settings:

```
analyzer.add_analysis('msd msd_1')
```

Add by string - adjust a setting:

```
analyzer.add_analysis('msd msd_1 leaflet both')
```

Add by list:

```
analyzer.add_analysis(list(['msd', 'msd_1', dict({'leaflet':'both'})]))
```

Add by dict:

```
analyzer.add_analysis(dict({'analysis_key': 'msd', 'analysis_id': 'msd_1',
↪'analysis_settings':dict({'leaflet':'both'})}))
```

To remove from analyzer:

```
analyzer.remove_analysis('msd_1')
```

Output Info:

Retrieve output after running analyses:

```
output = analyzer.get_analysis_data('msd_1')
```

The output is type <type 'numpy.ndarray'>

9.23.4 Related analyses

- msd_multi

9.23.5 References

1. Preston B. Moore, Carlos F. Lopez, Michael L. Klein, Dynamical Properties of a Hydrated Lipid Bilayer from a Multinanosecond Molecular Dynamics Simulation, Biophysical Journal, Volume 81, Issue 5, 2001, Pages 2484-2494, ISSN 0006-3495, http://dx.doi.org/10.1016/S0006-3495(01)75894-8. (http://www.sciencedirect.com/science/article/pii/S0006349501758948)
2. Yoshimichi Andoh, Susumu Okazaki, Ryuichi Ueoka, Molecular dynamics study of lipid bilayers modeling the plasma membranes of normal murine thymocytes and leukemic GRSL cells, Biochimica et Biophysica Acta (BBA) - Biomembranes, Volume 1828, Issue 4, April 2013, Pages 1259-1270, ISSN 0005-2736, https://doi.org/10.1016/j.bbamem.2013.01.005. (http://www.sciencedirect.com/science/article/pii/S0005273613000096)
3. Section 8.7, http://manual.gromacs.org/documentation/5.1.4/manual-5.1.4.pdf

9.24 ndcorr - Correlation between bilayer surface curvature and lipid clustering.

9.24.1 BilayerAnalyzer analysis: `ndcorr` - Correlation between bilayer surface curvature and lipid clustering.

9.24.2 Description

Correlation between bilayer surfucace curvature and the clustering of lipid molecules.

This protocol is used to estimate the cross correlation between the normal dimension deflection of lipids and the lipid types in local blocks of the bilayer. This serves as a measure of the correlation between the local curvature and composition of the bilayer.

This protocol is identified by the analysis key: 'ndcorr'

Initiated by instance of:

```
<class 'pybilt.bilayer_analyzer.analysis_protocols.NDCorrProtocol'>
```

9.24.3 Syntax

```
ndcorr analysis-ID
```

- ndcorr = analysis-Key - keyword/name for this analysis.

- analysis-ID = The unique name/ID being assigned to this analysis.

Examples

Construct analyzer:

```
analyzer = BilayerAnalyzer(structure='name_of_structure_file',
                           trajectory='name_of_traj_file',
                           selection='resname POPC DOPC')
```

Add by string - use default settings:

```
analyzer.add_analysis('ndcorr ndcorr_1')
```

Add by string - adjust a setting:

```
analyzer.add_analysis('ndcorr ndcorr_1 none None')
```

Add by list:

```
analyzer.add_analysis(list(['ndcorr', 'ndcorr_1', dict({'none':None})]))
```

Add by dict:

```
analyzer.add_analysis(dict({'analysis_key': 'ndcorr', 'analysis_id': 'ndcorr_
↪1','analysis_settings':dict({'none':None})}))
```

To remove from analyzer:

```
analyzer.remove_analysis('ndcorr_1')
```

Output Info:

Retrieve output after running analyses:

```
output = analyzer.get_analysis_data('ndcorr_1')
```

The output is type `<type 'dict'>`

9.24.4 Related analyses

- None

9.24.5 References

1. Koldso H, Shorthouse D, He Lie Sansom MSP (2014) "Lipid Clustering Correlates with Membrane Curvature as Revealed by Molecular Simulations of Complex Lipid Bilayers." PloS Comput Biol 10(10): e1003911. doi.10.1371/journal.pcbi.1003911

9.25 nnf - Lateral order nearest neighbor fraction.

9.25.1 BilayerAnalyzer analysis: `nnf` - Lateral order nearest neighbor fraction.

9.25.2 Description

Estimate the fraction of one lipid type within the nearest neighbors of another.

This analysis picks a specified number (n_neighbors) of nearest neighbors centered on a lipid of reference lipid type and then counts the number of lipids (M) of target lipid type and estimates the fraction, nnf = <M/n_neighbors> , where angle brackets denote averaging over lipids of specified by settings resname_1. This is the nearest neighbor analysis described in Ref. 1, and subsequently used in Ref. 2. This metric isn't exactly the same but is similar to the 'fractional interations' analysis of Ref. 3.

This protocol is identified by the analysis key: 'nnf'

Initiated by instance of:

```
<class 'pybilt.bilayer_analyzer.analysis_protocols.NNFProtocol'>
```

9.25.3 Syntax

```
nnf analysis-ID keyword value
```

- nnf = analysis-Key - keyword/name for this analysis.
- analysis-ID = The unique name/ID being assigned to this analysis.
- keyword value = settings keyword value pairs
  - leaflet (str: 'both', 'upper', or 'lower'): Specifies the bilayer leaflet to include in the estimate. Default: 'both'
  - n_neighbors (int): Specifies the number of nearest neighbors to to include in computation. Default: 5
  - resname_2 (str): Specify the resname of the target lipid type to include in this analysis. Default: 'first', the first lipid in the list pulled from the com_frame representation.
  - resname_1 (str): Specify the resname of the reference lipid type to include in this analysis. Default: 'first', the first lipid in the list pulled from the com_frame representation.

Examples

Construct analyzer:

```
analyzer = BilayerAnalyzer(structure='name_of_structure_file',
                           trajectory='name_of_traj_file',
                           selection='resname POPC DOPC')
```

Add by string - use default settings:

```
analyzer.add_analysis('nnf nnf_1')
```

Add by string - adjust a setting:

```
analyzer.add_analysis('nnf nnf_1 leaflet both')
```

Add by list:

```
analyzer.add_analysis(list(['nnf', 'nnf_1', dict({'leaflet':'both'})]))
```

Add by dict:

```
analyzer.add_analysis(dict({'analysis_key': 'nnf', 'analysis_id': 'nnf_1',
↪'analysis_settings':dict({'leaflet':'both'})}))
```

To remove from analyzer:

```
analyzer.remove_analysis('nnf_1')
```

Output Info:

Retrieve output after running analyses:

```
output = analyzer.get_analysis_data('nnf_1')
```

The output is type `<type 'numpy.ndarray'>`

9.25.4 Related analyses

- None

9.25.5 References

1. A. H. de Vries, A. E. Mark and S. J. Marrink, J. Phys. Chem. B, 2004, 108, 2454-2463
2. M. Orsi and J. W. Essex, Faraday Discuss., 2013, 161, 249-272
3. Koldso H, Shorthouse D, He Lie Sansom MSP (2014) "Lipid Clustering Correlates with Membrane Curvature as Revealed by Molecular Simulations of Complex Lipid Bilayers." PloS Comput Biol 10(10): e1003911. doi.10.1371/journal.pcbi.1003911

9.26 spatial_velocity_corr - Weighted average of displacement vector correlations.

9.26.1 BilayerAnalyzer analysis: `spatial_velocity_corr` - Weighted average of displacement vector correlations.

9.26.2 Description

Compute spatial correlation function between lipid displacements/pseudo-velocities.

This analysis computes the displacement vectors as in the 'disp_vec' analysis, and uses those to estimated a spatial correlation function between the displacments/pseudo-velocities; the function is of the radial separation between the base of each vector in the bilayers lateral plane. The correlation is the cos(theta) value for the angle theta between the vectors.

This protocol is identified by the analysis key: 'spatial_velocity_corr'

Initiated by instance of:

```
<class 'pybilt.bilayer_analyzer.analysis_protocols.
↪SpatialVelocityCorrelationFunctionProtocol'>
```

9.26.3 Syntax

```
spatial_velocity_corr analysis-ID keyword value
```

- spatial_velocity_corr = analysis-Key - keyword/name for this analysis.
- analysis-ID = The unique name/ID being assigned to this analysis.
- keyword value = settings keyword value pairs
  - n_bins (int): Specifies the number of bins to use when estimating the RDF. Default: 25
  - interval (int): Sets the frame interval over which to compute the displacement vectors.
  - range_outer (float): Specify the outer distance cutoff for the RDF. Default: 25.0
  - leaflet (str: 'both', 'upper', or 'lower'): Specifies the bilayer leaflet to include in the estimate. Default: 'both'
  - range_inner (float): Specify the inner distance cutoff for the RDF. Default: 0.0
  - resname_2 (str): Specify the resname of the target lipid type to include in this analysis. Special names are 'first' and 'all', which use the first and all lipid types respectively. Default: 'first', the first lipid in the list pulled from the com_frame representation.
  - resname_1 (str): Specify the resname of the reference lipid type to include in this analysis. Special names are 'first' and 'all', which use the first and all lipid types respectively. Default: 'first', the first lipid in the list pulled from the com_frame representation.

Examples

Construct analyzer:

```
analyzer = BilayerAnalyzer(structure='name_of_structure_file',
                           trajectory='name_of_traj_file',
                           selection='resname POPC DOPC')
```

Add by string - use default settings:

```
analyzer.add_analysis('spatial_velocity_corr spatial_velocity_corr_1')
```

Add by string - adjust a setting:

```
analyzer.add_analysis('spatial_velocity_corr spatial_velocity_corr_1 n_bins␣
↪25')
```

Add by list:

```
analyzer.add_analysis(list(['spatial_velocity_corr', 'spatial_velocity_corr_1
↪', dict({'n_bins':25})]))
```

Add by dict:

```
analyzer.add_analysis(dict({'analysis_key': 'spatial_velocity_corr',
↪'analysis_id': 'spatial_velocity_corr_1','analysis_settings':dict({'n_bins
↪':25})}))
```

To remove from analyzer:

```
analyzer.remove_analysis('spatial_velocity_corr_1')
```

Output Info:

Retrieve output after running analyses:

```
output = analyzer.get_analysis_data('spatial_velocity_corr_1')
```

The output is type `<type 'tuple'>`

9.26.4 Related analyses

- disp_vec
- disp_vec_corr

9.26.5 References

None

9.27 thickness_grid - Bilayer thickness using lipid_grid.

9.27.1 BilayerAnalyzer analysis: `thickness_grid` - Bilayer thickness using lipid_grid.

9.27.2 Description

Estimate the bilayer thickness using a gridding procedure.

This analysis uses a lipid grid representation (lipid positions are mapped to 2-D grids, one grid per leaflet) of the bilayer to estimate the bilayer thickness. This is done by measuring the difference in 'z' position between corresponding grid points of opposing bilayer leaflets.

This protocol is identified by the analysis key: 'thickness_grid'

Initiated by instance of:

```
<class 'pybilt.bilayer_analyzer.analysis_protocols.BTGridProtocol'>
```

9.27.3 Syntax

```
thickness_grid analysis-ID
```

- thickness_grid = analysis-Key - keyword/name for this analysis.
- analysis-ID = The unique name/ID being assigned to this analysis.

Examples

Construct analyzer:

```
analyzer = BilayerAnalyzer(structure='name_of_structure_file',
                           trajectory='name_of_traj_file',
                           selection='resname POPC DOPC')
```

Add by string - use default settings:

```
analyzer.add_analysis('thickness_grid thickness_grid_1')
```

Add by string - adjust a setting:

```
analyzer.add_analysis('thickness_grid thickness_grid_1 none None')
```

Add by list:

```
analyzer.add_analysis(list(['thickness_grid', 'thickness_grid_1', dict({'none
↪':None})]))
```

Add by dict:

```
analyzer.add_analysis(dict({'analysis_key': 'thickness_grid', 'analysis_id':
↪'thickness_grid_1','analysis_settings':dict({'none':None})}))
```

To remove from analyzer:

```
analyzer.remove_analysis('thickness_grid_1')
```

Output Info:

Retrieve output after running analyses:

```
output = analyzer.get_analysis_data('thickness_grid_1')
```

The output is type `<type 'numpy.ndarray'>`

9.27.4 Related analyses

- lipid_length
- thickness_leaflet_distance
- thickness_peak_distance
- thickness_mass_weighted_std

9.27.5 References

1. Allen et al. Vol. 30, No. 12 Journal of Computational Chemistry
2. Gapsys et al. J Comput Aided Mol Des (2013) 27:845-858

9.28 thickness_leaflet_distance - Distance between the leaflets' centers of mass along the normal.

9.28.1 BilayerAnalyzer analysis: `thickness_leaflet_distance` - Distance between the leaflets' centers of mass along the normal.

9.28.2 Description

Estimate the bilayer thickness using the distance between the COMs of each leaflet.

This analysis uses the COM postions from the com_frame representation. At each frame the centers-of-mass of each leaflet is computed and the distance along the bilayer normal is computed. This is another to estimate the bilayer thickness.

This protocol is identified by the analysis key: 'thickness_leaflet_distance'

Initiated by instance of:

```
<class 'pybilt.bilayer_analyzer.analysis_protocols.LeaftoLeafDistProtocol'>
```

9.28.3 Syntax

```
thickness_leaflet_distance analysis-ID
```

- thickness_leaflet_distance = analysis-Key - keyword/name for this analysis.
- analysis-ID = The unique name/ID being assigned to this analysis.

Examples

Construct analyzer:

```
analyzer = BilayerAnalyzer(structure='name_of_structure_file',
                           trajectory='name_of_traj_file',
                           selection='resname POPC DOPC')
```

Add by string - use default settings:

```
analyzer.add_analysis('thickness_leaflet_distance thickness_leaflet_distance_
↪1')
```

Add by string - adjust a setting:

```
analyzer.add_analysis('thickness_leaflet_distance thickness_leaflet_distance_
↪1 none None')
```

Add by list:

```
analyzer.add_analysis(list(['thickness_leaflet_distance', 'thickness_leaflet_
↪distance_1', dict({'none':None})]))
```

Add by dict:

```
analyzer.add_analysis(dict({'analysis_key': 'thickness_leaflet_distance',
↪'analysis_id': 'thickness_leaflet_distance_1','analysis_settings':dict({
↪'none':None})}))
```

To remove from analyzer:

```
analyzer.remove_analysis('thickness_leaflet_distance_1')
```

Output Info:

Retrieve output after running analyses:

```
output = analyzer.get_analysis_data('thickness_leaflet_distance_1')
```

The output is type `<type 'numpy.ndarray'>`

9.28.4 Related analyses

- lipid_length
- thickness_grid
- thickness_peak_distance
- thickness_mass_weighted_std

9.28.5 References

None

9.29 thickness_mass_weighted_std - Estimates the thickness using two times the mass-weighted standard deviation of coordinates along the normal.

9.29.1 BilayerAnalyzer analysis: `thickness_mass_weighted_std` - Estimates the thickness using two times the mass-weighted standard deviation of coordinates along the normal.

9.29.2 Description

Estimate the thickness via the mass weighted standard deviation along the normal dimension. This protocol is used to estimate the thickness by computing the standard deviation of the reference atoms (e.g. phosphorous atoms) coordinates along the bilayer normal dimension weighted by their masses: thickness = 2xmass_weighted_std This is same method used by MEMBPLUGIN to estimate bilayer thickness (Blake: I had to examine the MEMBPLUGIN source code to see that this is what it actually does.).

This protocol is identified by the analysis key: 'mass_weighted_std_thickness'

Initiated by instance of:

```
<class 'pybilt.bilayer_analyzer.analysis_protocols.
↪MassWeightedStdDistProtocol'>
```

9.29.3 Syntax

```
thickness_mass_weighted_std analysis-ID keyword value
```

- thickness_mass_weighted_std = analysis-Key - keyword/name for this analysis.
- analysis-ID = The unique name/ID being assigned to this analysis.
- keyword value = settings keyword value pairs
  - selection_string (str): Provide the MDAnalysis compatible selection for the atoms to include in this analysis. Default: 'BILAYER', use all the lipids of the bilayer as recovered from the selection given to the external BilayerAnalyzer.

Examples

Construct analyzer:

```
analyzer = BilayerAnalyzer(structure='name_of_structure_file',
                           trajectory='name_of_traj_file',
                           selection='resname POPC DOPC')
```

Add by string - use default settings:

```
analyzer.add_analysis('thickness_mass_weighted_std thickness_mass_weighted_
↪std_1')
```

Add by string - adjust a setting:

```
analyzer.add_analysis('thickness_mass_weighted_std thickness_mass_weighted_
↪std_1 selection_string name P')
```

Add by list:

```
analyzer.add_analysis(list(['thickness_mass_weighted_std', 'thickness_mass_
↪weighted_std_1', dict({'selection_string':'name P'})]))
```

Add by dict:

```
analyzer.add_analysis(dict({'analysis_key': 'thickness_mass_weighted_std',
↪'analysis_id': 'thickness_mass_weighted_std_1','analysis_settings':dict({
↪'selection_string':'name P'})}))
```

To remove from analyzer:

```
analyzer.remove_analysis('thickness_mass_weighted_std_1')
```

Output Info:

Retrieve output after running analyses:

```
output = analyzer.get_analysis_data('thickness_mass_weighted_std_1')
```

The output is type `<type 'numpy.ndarray'>`

9.29.4 Related analyses

- None

9.29.5 References

1. R. Guixa-Gonzalez; I. Rodriguez-Espigares; J. M. Ramirez-Anguita; P. Carrio-Gaspar; H. Martinez-Seara; T. Giorgino; J. Selent. MEMBPLUGIN: studying membrane complexity in VMD. Bioinformatics 2014; vol. 30 (10) p. 1478-1480 doi:10.1093/bioinformatics/btu037

9.30 thickness_peak_distance - Estimate the bilayer thickness using distance between corresponding peaks in a mass density profile.

9.30.1 BilayerAnalyzer analysis: `thickness_peak_distance` - Estimate the bilayer thickness using distance between corresponding peaks in a mass density profile.

9.30.2 Description

Estimate the mass density profile for the specified selection.

This protocol is used to estimate the bilayer thickness by computing the 1-dimensional mass density profile (with two peaks) for a selection of atoms along the bilayer normal and then estimating the distance between between the two peaks.

This protocol is identified by the analysis key: 'thickness_peak_distance'

Initiated by instance of:

```
<class 'pybilt.bilayer_analyzer.analysis_protocols.PeakToPeakProtocol'>
```

9.30.3 Syntax

```
thickness_peak_distance analysis-ID keyword value
```

- thickness_peak_distance = analysis-Key - keyword/name for this analysis.
- analysis-ID = The unique name/ID being assigned to this analysis.
- keyword value = settings keyword value pairs
  - n_bins (int): Set the number of bins to divide the normal dimensions into for binning. Defalt: 25
  - selection_string (str): Provide the MDAnalysis compatible selection for the atoms to include in this analysis. Default: 'BILAYER', use all the lipids of the bilayer as recovered from the selection given to the external BilayerAnalyzer.

Examples

Construct analyzer:

```
analyzer = BilayerAnalyzer(structure='name_of_structure_file',
                           trajectory='name_of_traj_file',
                           selection='resname POPC DOPC')
```

Add by string - use default settings:

```
analyzer.add_analysis('thickness_peak_distance thickness_peak_distance_1')
```

Add by string - adjust a setting:

```
analyzer.add_analysis('thickness_peak_distance thickness_peak_distance_1 n_
↪bins 25')
```

Add by list:

```
analyzer.add_analysis(list(['thickness_peak_distance', 'thickness_peak_
↪distance_1', dict({'n_bins':25})]))
```

Add by dict:

```
analyzer.add_analysis(dict({'analysis_key': 'thickness_peak_distance',
↪'analysis_id': 'thickness_peak_distance_1','analysis_settings':dict({'n_
↪bins':25})}))
```

To remove from analyzer:

```
analyzer.remove_analysis('thickness_peak_distance_1')
```

Output Info:

Retrieve output after running analyses:

```
output = analyzer.get_analysis_data('thickness_peak_distance_1')
```

The output is type `<type 'numpy.ndarray'>`

9.30.4 Related analyses

- None

9.30.5 References

None

9.31 vcm - Volume compressibility modulus.

9.31.1 BilayerAnalyzer analysis: `vcm` - Volume compressibility modulus.

9.31.2 Description

Estimate the isothermal volume compressibility modulus.

This protocol is used to estimate the volume compressibility modulus, K_V = (kT) / var(V), where V is the volume.

This protocol is identified by the analysis key: 'vcm'

Initiated by instance of:

```
<class 'pybilt.bilayer_analyzer.analysis_protocols.
↪VolumeCompressibilityModulusProtocol'>
```

9.31.3 Syntax

```
vcm analysis-ID keyword value
```

- vcm = analysis-Key - keyword/name for this analysis.
- analysis-ID = The unique name/ID being assigned to this analysis.
- keyword value = settings keyword value pairs
  - temperature (float): The absolute temperature that the simulation was run at (i.e. in Kelvin). Default: 298.15 K

Examples

Construct analyzer:

```
analyzer = BilayerAnalyzer(structure='name_of_structure_file',
                           trajectory='name_of_traj_file',
                           selection='resname POPC DOPC')
```

Add by string - use default settings:

```
analyzer.add_analysis('vcm vcm_1')
```

Add by string - adjust a setting:

```
analyzer.add_analysis('vcm vcm_1 temperature 298.15')
```

Add by list:

```
analyzer.add_analysis(list(['vcm', 'vcm_1', dict({'temperature':298.15})]))
```

Add by dict:

```
analyzer.add_analysis(dict({'analysis_key': 'vcm', 'analysis_id': 'vcm_1',
↪'analysis_settings':dict({'temperature':298.15})}))
```

To remove from analyzer:

```
analyzer.remove_analysis('vcm_1')
```

Output Info:

Retrieve output after running analyses:

```
output = analyzer.get_analysis_data('vcm_1')
```

The output is type `<type 'numpy.ndarray'>`

9.31.4 Related analyses

- acm
- ac

9.31.5 References

1. Christofer Hofsab, Erik Lindahl, and Olle Edholm, "Molecular Dynamics Simulations of Phospholipid Bilayers with Cholesterol", Biophys J. 2003 Apr; 84(4): 2192-2206. doi: 10.1016/S0006-3495(03)75025-5

CHAPTER

TEN

PYBILT

10.1 pybilt package

10.1.1 Subpackages

pybilt.bilayer_analyzer package

Submodules

pybilt.bilayer_analyzer.analysis_protocols module

pybilt.bilayer_analyzer.bilayer_analyzer module

pybilt.bilayer_analyzer.com_frame module

pybilt.bilayer_analyzer.leaflet module

pybilt.bilayer_analyzer.mda_data module

pybilt.bilayer_analyzer.plot_protocols module

pybilt.bilayer_analyzer.prefab_analysis_protocols module

pybilt.bilayer_analyzer.vector_frame module

Module contents

pybilt.com_trajectory package

Submodules

pybilt.com_trajectory.COMTraj module

Module contents

pybilt.common package

Submodules

pybilt.common.distance_cutoff_clustering module

Function to compute hiearchical distance cutoff clusters.

pybilt.common.distance_cutoff_clustering.**distance_cutoff_clustering**(*vectors*, *cutoff*, *dist_func*, *min_size=1*, **df_args*, ***df_kwargs*)

Hiearchical distance cutoff clustering.

This function takes a set of vector points and clusters them using a hiearchical distance based clustering algorithm. Points are clustered together whenevever a point is within the cutoff distance of any point within in a cluster.

Parameters

- **vectors** (*np.array, list like*) – The array of vector points.
- **cutoff** (*float*) – The cutoff distance.
- **dist_func** (*function*) – The function to use when computing the distance between points.
- **min_size** (*Optional[int]*) – The minimum size of a cluster. Defaults to 1.
- ***df_args** – Any additional arguments to be passed to the distance function (dist_func).
- ****df_kwargs** – Any additional keyword arguments to be passed to the distance function (dist_func).

Returns

Returns a list of clustered points where each set of clustered i points is a list of the indices of the points in that cluster.

Return type list

pybilt.common.distance_cutoff_clustering.**distance_euclidean**(*v_a*, *v_b*)

Compute the Euclidean distance between two vectors.

Parameters

- **v_a** (*numpy.array, array like*) – The first input vector.
- **v_b** (*numpy.array, array like*) – The second input vector.

Returns The Euclidean distance between the two vectors.

Return type float

Notes

The two vectors should have the same size and dimension.

pybilt.common.distance_cutoff_clustering.**distance_euclidean_pbc**(*v_a*, *v_b*, *box_lengths*, *center='zero'*)

Compute the Euclidean distance between two vectors under periodic boundaries.

Parameters

- **v_a** (*numpy.array, array like*) – The first input vector.
- **v_b** (*numpy.array, array like*) – The second input vector.
- **box_lengths** (*numpy.array, array like*) – The periodic boundary box lengths for each dimension.
- **center** (*Optional[str, array like]*) – Set the coordinate center of the periodic box dimensions. Defaults to 'zero', which sets the center to numpy.zeros(len(box_lengths)). Also accepts the string value 'box_half', which sets the center to 0.5*box_lengths.

Returns The Euclidean distance between the two vectors.

Return type float

Notes

The two vectors should have the same size and dimension, while box_lengths should have the length of the vector dimension.

pybilt.common.distance_cutoff_clustering.**vector_difference_pbc**(*v_a*, *v_b*, *box_lengths*, *center='zero'*)

Compute the Euclidean distance between two vectors under periodic boundaries.

Parameters

- **v_a** (*numpy.array, array like*) – The first input vector.
- **v_b** (*numpy.array, array like*) – The second input vector.
- **box_lengths** (*numpy.array, array like*) – The periodic boundary box lengths for each dimension.
- **center** (*Optional[str, array like]*) – Set the coordinate center of the periodic box dimensions. Defaults to 'zero', which sets the center to numpy.zeros(len(box_lengths)). Also accepts the string value 'box_half', which sets the center to 0.5*box_lengths.

Returns The Euclidean distance between the two vectors.

Return type float

Notes

The two vectors should have the same size and dimension, while box_lengths should have the length of the vector dimension.

pybilt.common.gaussian module

Define Gaussian function objects.

This module defines the Gaussian class and the GaussianRange class.

class pybilt.common.gaussian.**Gaussian**(*mean*, *std*)
Bases: object

A Gaussian function object.

mean
The mean of the Gaussian.

Type float

std
The standard deviation of the Gaussian.

Type float

Initialize a Gaussian function object.

Parameters

- **mean** (*float*) – Set the mean of the Gaussian.
- **std** (*float*) – Set the standard deviation of the Gaussian.

eval(*x_in*)
Return the Gaussian function evaluated at the input x value.

Parameters **x_in** (*float*) – The x value to evaluate the function at.

Returns The function evaluation for the Gaussian.

Return type float

reset_mean(*new_mean*)
Change the mean of the Gaussian function.

Parameters **new_mean** (*float*) – The new mean of the Gaussian function.

class pybilt.common.gaussian.**GaussianRange**(*in_range*, *mean*, *std*, *npoints=200*)
Bases: object

Define a Gaussian function over a range.

This object is used to define a Gaussian function over a defined finite range and store its values as evaluated at points evenly spaced over the range. The points can then for example be used for integrating the Gaussian function over the range using numerical quadrature.

mean
The mean of the Gaussian.

Type float

std
The standard deviation of the Gaussian.

Type float

upper
The upper boundary of the range.

Type float

lower
The lower boundary of the range.

Type float

npoints
The number of points to evaluate in the range.

Type int

Initialize the GaussianRange object.

The GaussianRange stores the values of Gaussian function with the input mean and standard deviation evaluated at evenly spaced points in the specified x-value range.

Parameters

- **in_range** (*tuple, list*) – Specify the endpoints for range, e.g. (x_start, x_end).
- **mean** (*float*) – The mean of the Gaussian function.
- **std** (*float*) – The standard deviation of the Gaussian function.
- **npoints** (*Optional[int]*) – The number of x-value points to evaluate the Gaussian function for in the specified range (i.e. in_range).

eval(*x_in*)
Return the Gaussian function evaluated at the input x value.

Parameters **x_in** (*float*) – The x value to evaluate the function at.

Returns The function evaluation for the Gaussian.

Return type float

get_values()
Return the x and y values for the Gaussian range function.

Returns The x and y values for the function, returned as (x_values, y_values).

Return type tuple

integrate_range(*lower*, *upper*)
Returns the numerical integration of the Gaussian range.

This function does a simple quadrature for the Gaussian function as evaluated on the range (or subset of the range) specified at initialization.

Parameters

- **lower** (*float*) – The lower boundary for the integration.
- **upper** (*float*) – The upper boundary for the integration.

Returns

The numerical value of the Gaussian range integrated from lower to upper.

Return type float

Notes

This function does not thoroughly check the bounds, so if upper is less than lower the function will break.

normalize()
Normalizes (by area) the Gaussian function values over the range.

reset_mean(*new_mean*)
Change the mean of the Gaussian function.

Parameters **new_mean** (*float*) – The new mean of the Gaussian function.

Notes

This function does not re-evaluate the Gaussian range and therefore only affects the output of the eval function.

sum_range(*lower*, *upper*)
Returns the over the Gaussian range.

This function sums the Gaussian function at the points that were evaluated on the range (or subset of the range) specified at initialization.

Parameters

- **lower** (*float*) – The lower boundary for the sum.
- **upper** (*float*) – The upper boundary for the sum.

Returns

The numerical value of the Gaussian range as summed from lower to upper.

Return type float

Notes

This function does not thoroughly check the bounds, so if upper is less than lower the function will break.

pybilt.common.knn_entropy module

Functions to evaluate information theoretic measures using knn approaches.

This module defines a set of functions to compute information theoretic measures (i.e. Shannon Entropy, Mutual Information, etc.) using the k-nearest neighbors (knn) approach.

pybilt.common.knn_entropy.**conditional_mutual_information**(*var_tuple*, *cond_tuple*, *k=2*)
Returns an estimate of the conditional mutual information.

This function computes an estimate of the mutual information between a set of random variables or random vectors conditioned on other random variables or vectors using knn estimators for the entropy calculations.

Parameters

- **var_tuple** (*tuple*) – A tuple of random variables or random vectors (i.e. numpy.array) to estimate the mutual information between.; e.g. var_tuple = (X, Y) where X form X = {x_1, x_2, x_3,...,x_N} and Y has form Y = {x_1, x_2, x_3,...,x_N}, or where X has form X = {(x_X1, y_X1), (x_X2, y_X2),...,(x_XN, y_XN)} and Y has form Y = {(x_Y1, y_Y1), (x_Y2, y_Y2),...,(x_YN, y_YN)}.
- **cond_tuple** (*tuple*) –

  A tuple of random variables or random vectors (i.e. numpy.array) that the mutual information is to be conditioned on; e.g. var_tuple = (X) where X has the form X = {x_1, x_2, x_3,...,x_N}.

  k (Optional[int]): The number of nearest neighbors to store for each point. Defaults to 2.

Returns The mutual information estimate.

Return type float

Notes

The information entropies used to estimate the mutual information are computed using the shannon_entropy function. All input random variable/vector arrays must have the same shape.

pybilt.common.knn_entropy.**k_nearest_neighbors**(*X*, *k=1*)
Get the k-nearest neighbors between points in a random variable/vector.

Determines the k nearest neighbors for each point in teh random variable/vector using Euclidean style distances.

Parameters

- **X** (*np.array*) – A random variable of form X = {x_1, x_2, x_3,...,x_N} or a random vector of form X = {(x_1, y_1), (x_2, y_2),...,(x_N, y_n)}.
- **k** (*Optional[int]*) – The number of nearest neighbors to store for each point. Defaults to 1.

Returns

A dictionary keyed by the indices of X and containing a list of the k nearest neighbor for each point along with the distance value between the point and the nearest neighbor.

Return type dict

pybilt.common.knn_entropy.**kth_nearest_neighbor_distances**(*X*, *k=1*)
Returns the distance for the kth nearest neighbor of each point.

Parameters

- **X (*np.array*) – A random variable of form X = {x_1, x_2, x_3,...,x_N} or**

  a random vector of form X = {(x_1, y_1), (x_2, y_2),...,(x_N, y_n)}.

  – **k** (*Optional[int]*) – The number of nearest neighbors to check for each point. Defaults to 1.

Returns: list: A list in same order as X with the distance value to the kth nearest neighbor of each point in X.

pybilt.common.knn_entropy.**mutual_information**(*var_tuple*, *k=2*)
Returns an estimate of the mutual information.

This function computes an estimate of the mutual information between a set of random variables or random vectors using knn estimators for the entropy calculations.

Parameters **var_tuple** (*tuple*) –

A tuple of random variables or random vectors (i.e. numpy.array); e.g. var_tuple = (X, Y) where X form X = {x_1, x_2, x_3,...,x_N} and Y has form Y = {x_1, x_2, x_3,...,x_N}, or where X has form X = {(x_X1, y_X1), (x_X2, y_X2),...,(x_XN, y_XN)} and Y has form Y = {(x_Y1, y_Y1), (x_Y2, y_Y2),...,(x_YN, y_YN)}.

k (Optional[int]): The number of nearest neighbors to store for each point. Defaults to 2.

Returns The mutual information estimate.

Return type float

Notes

The information entropies used to estimate the mutual information are computed using the shannon_entropy function. All input random variable/vector arrays must have the same shape.

pybilt.common.knn_entropy.**shannon_entropy**(*X*, *k=1*, *kth_dists=None*)
Return the Shannon Entropy of the random variable/vector.

This function computes the Shannon information entropy of the random variable/vector as estimated using the Kozachenko-Leonenko (KL) knn estimator.

Parameters

- **X** (*np.array*) – A random variable of form X = {x_1, x_2, x_3,...,x_N} or a random vector of form X = {(x_1, y_1), (x_2, y_2),...,(x_N, y_n)}.
- **k** (*Optional[int]*) – The number of nearest neighbors to store for each point. Defaults to 1.
- **kth_dists** (*Optional[list]*) – A list in the same order as points in X that has the pre-computed distances between the points in X and their kth nearest neighbors at. Defaults to None.

References

1. **Damiano Lombardi and Sanjay Pant, A non-parametric k-nearest**

   neighbour entropy estimator, arXiv preprint, [cs.IT] 2015, arXiv:1506.06501v1. https://arxiv.org/pdf/1506.06501v1.pdf

2. https://www.cs.tut.fi/~timhome/tim/tim/core/differential_entropy_kl_details.htm

3. **Kozachenko, L. F. & Leonenko, N. N. 1987 Sample estimate of entropy** of a random vector. Probl. Inf. Transm. 23, 95-101.

Returns The estimate of the Shannon Information entropy of X.

Return type float

pybilt.common.knn_entropy.**shannon_entropy_pc**(*X*, *k=1*, *kth_dists=None*)
Return the Shannon Entropy of the random variable/vector.

This function computes the Shannon information entropy of the random variable/vector as estimated using the Perez-Cruz knn estimator described in Reference 1.

Parameters

- **X** (*np.array*) – A random variable of form X = {x_1, x_2, x_3,...,x_N} or a random vector of form X = {(x_1, y_1), (x_2, y_2),...,(x_N, y_n)}.
- **k** (*Optional[int]*) – The number of nearest neighbors to store for each point. Defaults to 1.
- **kth_dists** (*Optional[list]*) – A list in the same order as points in X that has the pre-computed distances between the points in X and their kth nearest neighbors at. Defaults to None.

References

1. **Perez-Cruz, (2008). Estimation of Information Theoretic Measures** for Continuous Random Variables. Advances in Neural Information Processing Systems 21 (NIPS). Vancouver (Canada), December. https://papers.nips.cc/paper/3417-estimation-of-information-theoretic-measures-for-continuous-random-variables.pdf

Returns The estimate of the Shannon Information entropy of X.

Return type float

pybilt.common.running_stats module

Running stats module.

This module defines the RunningStats and BlockAverager classes, as well as the gen_running_average function.

class pybilt.common.running_stats.**BlockAverager**(*points_per_block=1000*, *min_points_in_block=500*, *store_data=False*)

Bases: object

An object that keeps track of points for block averaging.

n_blocks
The current number of active blocks.

Type int

Init a the BlockAverager

Parameters

- **points_per_block** (*int, Optional*) – The number of points to assign to a block before initiating a new block. Default: 1000
- **min_points_in_block** (*int, Optional*) – The minimum number of points that a block (typically the last block) can have and still be included in computing the final block average and standard error estimates. This value should be <= points_per_block. Default: 500

averages_of_blocks()
Return the block average and standard error.

Returns Returns a length two tuple with the block average and standard error estimates.

Return type tuple

get()
Return the block average and standard error.

Returns Returns a length two tuple with the block average and standard error estimates.

Return type tuple

n_block()

number_of_blocks()
Return the current number of blocks.

Returns The number of blocks.

Return type int

points_per_block()
Return information about the points per block.

Returns

A three element tuple containing the setting for points per block, the setting for minimum points per block, and the number of points in the last block.

Return type tuple

push_container(*data*)
Push a container (array or array like) of data points to the block averaging.

Parameters **data** (*array like*) – The container (list, tuple, np.array, etc.) of data points to add to the block averaging.

push_single(*datum*)
Push a single data point (datum) into the block averager.

Parameters **datum** (*float*) – The value to add to the block averaging.

standards_of_blocks()
Return the block average and standard error.

Returns Returns a length two tuple with the block average and standard error estimates.

Return type tuple

class pybilt.common.running_stats.**RunningStats**
Bases: object

A RunningStats object.

The RunningStats object keeps running statistics for a single value/quantity.

n
The number of points that have pushed to the running

Type int

average.

Initialize the RunningStats object.

deviation()
Return the current standard deviation.

mean()
Return the current mean.

push(*val*)
Push a new value to the running average.

Parameters **val** (*float*) – The value to be added to the running average.

Returns:

reset()
Reset the running average.

variance()
Returun the current variance.

pybilt.common.running_stats.**binned_average**(*data*, *positions*, *n_bins=25*, *position_range=None*, *min_count=0*)
Compute averages over a quantized range of histogram like bins.

Parameters

- **data** (*np.array*) – A 1d numpy array of values.
- **positions** (*np.array*) – A 1d numpy array of positions corresponding to the values in data. These are used to assign the values to the histogram like bins for averaging.
- **n_bins** (*Optional[int]*) – Set the target number of bins to quantize the position_range up into. Defaults to 25
- **position_range** (*Optional[tuple]*) – A two element tuple containing the lower and upper range to bin the postions over; i.e. (position_lower, postion_upper). Defaults to None, which uses positions.min() and positions.max().

Returns returns a tuple with two numpy arrays of form (bins, averages)

Return type tuple

Notes

The function automatically filters out bins that have a zero count, so the final value of the number of bins and values will be len(bins) <= n_bins.

pybilt.common.running_stats.**block_average_bse_v_size**(*data*)

pybilt.common.running_stats.**block_avg_hist**(*nparray_1d*, *block_size*, *in_range='auto'*, *scale=False*, **args*, ***kwargs*)

Creates histograms for each block and averages them to generate block a single block averaged historgram.

pybilt.common.running_stats.**gen_running_average**(*onednparray*)

Generates a running average

Args: onednparray (numpy.array): A 1d numpy array of measurements (e.g. over time)

Returns: numpy.array: 2d array of dim len(onednparray)x2

2dnparray[i][0] = running average at i 2dnparray[i][1] = running standard deviation at i for i in range(0,len(onednparray))

pybilt.common.running_stats.**pair_ttest**(*mean_1*, *std_err_1*, *n_1*, *mean_2*, *std_err_2*, *n_2*)

Module contents

pybilt.diffusion package

Submodules

pybilt.diffusion.diffusion_coefficients module

Estimate self diffusion coefficients from MSD(t) data.

This module provides a set of functions to estimate self diffusion coefficients from mean squared displacement (MSD) time trajectories.

pybilt.diffusion.diffusion_coefficients.**diffusion_coefficient_Einstein**(*times*, *msd_vals*, *dim=2*, *time_range=None*)

Estimate diffusion coefficient from MSD(t) via Einstein relation.

A function to estimate the diffusion constant from a mean squared displacement time series. This function uses the long time mean squared displacement approximation (Einstein relation):

MSD = lim_(t->inf) <||r_i(t) - r_i(0)||**2>_(nsels) = 2*dim*D*t

where D is the diffusion coefficient.

Parameters

- **times** (*numpy.array*) – Array of the simulation times of each MSD point.
- **msd_vals** (*numpy.array*) – Array of the MSD values.

- **dim** (*Optional[int]*) – Set the dimension of the data and fit function: 1 for 1-dimensional , 2 for 2-dimensional, or 3 for 3-dimensional. Defaults to 2.
- **time_range** (*Optional[list]*) – Specify a time range via a list with format [time_start, time_end]; the range should be given in the same units as the time data in the Arg times. Defaults to None, which uses the whole time range in Args times.

Returns The value of the diffusion coefficient.

Return type float

References

1. **Preston B. Moore, Carlos F. Lopez, Michael L. Klein, Dynamical** Properties of a Hydrated Lipid Bilayer from a Multinanosecond Molecular Dynamics Simulation, Biophysical Journal, Volume 81, Issue 5, 2001, Pages 2484-2494, ISSN 0006-3495, http://dx.doi.org/10.1016/S0006-3495(01)75894-8. (http://www.sciencedirect.com/science/article/pii/S0006349501758948)
2. **Section 8.7,** http://manual.gromacs.org/documentation/5.1.4/manual-5.1.4.pdf

pybilt.diffusion.diffusion_coefficients.**diffusion_coefficient_anomalous_fit**(*times*, *msd_vals*, *dim=2*, *time_range=None*)

Fit MSD time series data to an anomalous diffusion model.

The anomalous diffusion function has the form: MSD(t) = 2 * dim * D_alpha * t**alpha

For alpha values 0 < alpha < 1 corresponds to subdiffusion, while 1 < alpha < 2 corresponds to superdiffusion.

Parameters

- **times** (*numpy.array*) – Array of the simulation times of each MSD point.
- **msd_vals** (*numpy.array*) – Array of the MSD values.
- **dim** (*Optional[int]*) – Set the dimension of the data and fit function: 1 for 1-dimensional , 2 for 2-dimensional, or 3 for 3-dimensional. Defaults to 2.
- **time_range** (*Optional[list]*) – Specify a time range via a list with format [time_start, time_end]; the range should be given in the same units as the time data in the Arg times. Defaults to None, which uses the whole time range in Args times.

Returns The return is a tuple with values (D_alpha, alpha) where D_alpha is the anomalous diffusion coefficent and alpha is the anomalous diffusion power.

Return type tuple

References

1. **Gerald R. Kneller, Krzysztof Baczynski, and Marta** Pasenkiewicz-Gierula, Consistent picture of lateral subdiffusion in lipid bilayers: Molecular dynamics simulation and exact results, The Journal of Chemical Physics 135, 141105 (2011); doi: http://dx.doi.org/10.1063/1.3651800

pybilt.diffusion.diffusion_coefficients.**diffusion_coefficient_linear_fit**(*times*, *msd_vals*, *dim=2*, *time_range=None*)

Estimate the diffusion coefficent via linear least squares fit of MSD(t) data.

Assumes the MSD data has the linear form, MSD(t) = 2*dim*D*t ,

and uses a linear least squares fit to estimate the diffusion coefficent D.

Parameters

- **times** (*numpy.array*) – Array of the simulation times of each MSD point.
- **msd_vals** (*numpy.array*) – Array of the MSD values.
- **dim** (*Optional[int]*) – Set the dimension of the data and fit function: 1 for 1-dimensional , 2 for 2-dimensional, or 3 for 3-dimensional. Defaults to 2.
- **time_range** (*Optional[list]*) – Specify a time range via a list with format [time_start, time_end]; the range should be given in the same units as the time data in the Arg times. Defaults to None, which uses the whole time range in Args times.

Returns Returns a tuple with format (D, Error_D), where D is the estimated diffusion coefficent and Error_D is the estimated error for D.

Return type tuple

References

1. **Preston B. Moore, Carlos F. Lopez, Michael L. Klein, Dynamical** Properties of a Hydrated Lipid Bilayer from a Multinanosecond Molecular Dynamics Simulation, Biophysical Journal, Volume 81, Issue 5, 2001, Pages 2484-2494, ISSN 0006-3495; http://dx.doi.org/10.1016/S0006-3495(01)75894-8 (http://www.sciencedirect.com/science/article/pii/S0006349501758948)

Module contents

pybilt.lipid_grid package

Submodules

pybilt.lipid_grid.lipid_grid module

Build lipid grids derived from COMFrame objects. Classes and functions to implement lipid COM gridding and analysis for lipid bilayers. This module defines version that build grids off of COMFrame objects and is meant primarily for internal use by the BilayerAnalyzer class. The gridding and anlaysis procedures are based on the descriptions given in Gapsys et al. J Comput Aided Mol Des (2013) 27:845-858, which is itself a modified version of the GridMAT-MD method by Allen et al. Vol. 30, No. 12 Journal of Computational Chemistry. However, I have currently left out bits of the extra functionality like the handling of an embedded proteins.

class pybilt.lipid_grid.lipid_grid.**LipidGrid2d**(*com_frame*, *com_frame_indices*, *plane*, *nxbins=50*, *nybins=50*)

Bases: object

A 2d lipid grid object.

This object is used by the LipidGrids object to construct a 2d grid for a bilayer leaflet and assign lipids to it using the coordinates derived from a COMFrame representation object.

frame

Stores a local copy of the COMFrame object from which the the coordinate data and lipid type data is derived from.

Type COMFrame

x_nbins
The number of grid bins in the 'x' dimension.

Type int

y_nbins
The number of grid bins in the 'y' dimension.

Type int

x_min
The lower boundary of the grid range in the 'x' dimension.

Type float

x_max
The upper boundary of the grid range in the 'x' dimension.

Type float

x_incr
The size of grid boxes (or spacing between grid points) in the 'x' dimension.

Type float

x_centers
The center points of the grid boxes in the 'x' dimension.

Type np.array

x_edges
The edges of the grid boxes in the 'x' dimension.

Type np.array

x_centers
The centers of the grid boxes in the 'x' dimension.

Type np.array

y_min
The lower boundary of the grid range in the 'y' dimension.

Type float

y_max
The upper boundary of the grid range in the 'y' dimension.

Type float

y_incr
The size of grid boxes (or spacing between grid points) in the 'y' dimension.

Type float

y_centers
The center points of the grid boxes in the 'y' dimension.

Type np.array

y_edges
The edges of the grid boxes in the 'y' dimension.

Type np.array

y_centers
The centers of the grid boxes in the 'y' dimension.

Type np.array

lipid_grid
A 2d array of size x_nbins*y_nbins that stores the index of lipids from the COMFrame that are assigned to each grid box.

Type np.array

lipid_grid_z
A 2d array of size x_nbins*y_nbins that stores the z coordinate of the lipids assigned to each grid box.

Type np.array

Initialize the LipidGrid2d object.

Parameters

- **com_frame** (*COMFrame*) – The instance of COMFrame from which to pull the coordinates for lipids to use when building the grid.
- **com_frame_indices** (*list*) – A list COMFrame lipid indices to include when building the grid.
- **plane** (*list*) – The indices from the 3d coordinates for the coordinates that correspond to the bilayer lateral plane.
- **nxbins** (*Optional[int]*) – The number of bins along the 'x' dimension, i.e. along the dimension corresponding to plane[0]. Defaults to 50.
- **nybins** (*Optional[int*) – The number of bins along the 'y' dimension, i.e. along the dimension corresponding to plane[1]. Defaults to 50.

get_index_at(*ix*, *iy*)
Returns the COMFrame index of the lipid at the specified position in the lipid_grid.

Parameters

- **ix** (*int*) – The 'x' index in the lipid_grid.
- **iy** () – The 'y' index in the lipid_grid.

Returns The index of the lipid.

Return type int

get_z_at(*ix*, *iy*)
Returns the z coordinate of the lipid at the specified position in the lipid_grid.

Parameters

- **ix** (*int*) – The 'x' index in the lipid_grid.
- **iy** () – The 'y' index in the lipid_grid.

Returns The z coordinate of the lipid.

Return type float

write_xyz(*xyz_name*)
Write out the lipid grid as an xyz coordinate file.

Parameters **xyz_name** (*str*) – File path and name for the output file.

z_perturb_grid()
Returns the array with z coordinates shifted by the mean.

Returns The mean shifted z coordinate array.

Return type np.array

class pybilt.lipid_grid.lipid_grid.**LipidGrids**(*com_frame*, *leaflets*, *plane*, *nxbins=50*, *nybins=50*)

Bases: object

area_per_lipid()

average_thickness(*return_grid=False*)

curvature(*use_gaussian_filter=True*, *filter_sigma=10.0*, *filter_mode='nearest'*)

get_integer_type_arrays()

get_one_array_per_leaflet()

get_xyzc(*leaflet='both'*, *zvalue_dict=None*, *color_dict=None*, *color_grid=None*, *color_type_dict=None*)

grid_to_dict(*in_grid*, *leaflet='upper'*)

map_to_grid(*com_values_dict*, *leaflet='both'*)

surface_area(*use_gaussian_filter=True*, *filter_sigma=10.0*, *filter_mode='nearest'*)

thickness_grid()

write_pdb(*pdb_name*, *leaflet='both'*, *z_grid_upper=None*, *z_grid_lower=None*, *beta_grid_upper=None*, *beta_grid_lower=None*, *use_gaussian_filter=False*, *filter_sigma=10.0*, *filter_mode='nearest'*)

Write out the lipid grid as an PDB coordinate file.

Parameters

- **pdb_name** (*str*) – File path and name for the output file.
- **leaflet** (*Optional[str]*) – Specify which leaflets to write to the PDB file. The options are 'both', 'upper', or 'lower'. Defaults to 'both'.
- **z_grid_upper** (*Optional[np.array]*) – A 2d grid of values corresponding to the elements of the upper leaflet of the lipid grid that are to be written as the z-coordinate in the PDB file for upper leaflet members. Defaults to None.
- **z_grid_lower** (*Optional[np.array]*) – A 2d grid of values corresponding to the elements of the lower leaflet of the lipid grid that are to be written as the z-coordinate in the PDB file for lower leaflet members. Defaults to None.
- **beta_grid_upper** (*Optional[np.array]*) – A 2d grid of values corresponding to the elements of the upper leaflet of the lipid grid that are to be written in the Beta column of the PDB file for upper leaflet members. Defaults to None.
- **beta_grid_lower** (*Optional[np.array]*) – A 2d grid of values corresponding to the elements of the lower leaflet of the lipid grid that are to be written in the Beta column of the PDB file for upper leaflet members. Defaults to None.
- **use_gaussian_filter** (*Optional[bool]*) – Use SciPy's Gaussian filter to filter the z-coordinates before outputting the PDB file. Defaults to False. This option overrides inputs for z_grid_upper and z_grid_lower.
- **filter_sigma** (*Optional[bool]*) – Set the sigma value for the Gaussian filter. Defaults to 10.0
- **filter_mode** (*Optional['str']*) – Set the mode for Gaussian filter. Defaults to 'nearest'

write_xyz(*leaflet='both'*, *zvalue_dict='Default'*, *out_path='./'*)

pybilt.lipid_grid.lipid_grid.**grid_curvature**(*x_vals*, *y_vals*, *zgrid*)
Compute the Mean and Gaussian curvature across a grid. :Parameters: * **x_vals** (*np.array*) – The bin labels along the x-axis of the gridded data.

- **y_vals** (*np.arrray*) – The bin labels along the y-axis of the gridded data.
- **zgrid** (*np.array*) – The 2d grid of bin values.

Returns

Returns a 2 item tuple with the 2d numpy arrays of the curvatures with format (mean curvature, Gaussian curvature).

Return type tuple

pybilt.lipid_grid.lipid_grid.**grid_surface_area**(*x_vals*, *y_vals*, *zgrid*)
Compute the surface area across a regular 2d grid. :Parameters: * **x_vals** (*np.array*) – The bin labels along the x-axis of the gridded data.

- **y_vals** (*np.arrray*) – The bin labels along the y-axis of the gridded data.
- **zgrid** (*np.array*) – The 2d grid of bin values.

Returns Returns a the surface area estimate.

Return type float

pybilt.lipid_grid.lipid_grid_curv module

clustering of lipid molecules" of Koldso H, Shorthouse D, He lie J, Sansom MSP (2014) Lipid Clustering Correlates with Membrane Curvature as Revealed by Molecular Simulations of Complex Lipid Bilayers. PLoS Comput Biol 10(10): e1003911. doi:10.1371/journal.pcbi.1003911 However, this implementation currently uses the z position (or normal position) of the lipids' centers of mass, while their implementaion uses "the z coordinate of the interface between the head groups of the lipids (excluding the current species being calculated and tails in that box."

class pybilt.lipid_grid.lipid_grid_curv.**LipidGrid_2d**(*com_frame*, *com_frame_indices*, *plane*, *nxbins=20*, *nybins=20*)
Bases: object

get_index_at(*ix*, *iy*)

get_z_at(*ix*, *iy*)

class pybilt.lipid_grid.lipid_grid_curv.**LipidGrids**(*com_frame*, *leaflets*, *plane*, *nxbins=3*, *nybins=3*)
Bases: object

norm_displacement_cross_correlation()

pybilt.lipid_grid.lipid_grid_mda module

Classes and functions to implement lipid COM gridding and analysis for lipid bilayers. Acts on MemSys objects. The gridding and anlaysis procedures are based on the decriptions given in Gapsys et al. J Comput Aided Mol Des (2013) 27:845-858, which is itself a modified version of the GridMAT-MD method by Allen et al. Vol. 30, No. 12 Journal of Computational Chemistry. However, I have currently left out several bits of the extra functionality, e.g. the handling of an embedded protein.

class pybilt.lipid_grid.lipid_grid_mda.**LipidGrid_2d**(*mda_frame*, *mda_universe*, *mda_frame_resids*, *plane*, *nxbins=50*, *nybins=50*)

Bases: object

get_index_at(*ix*, *iy*)

get_z_at(*ix*, *iy*)

write_xyz(*xyz_name*)

class pybilt.lipid_grid.lipid_grid_mda.**LipidGrids**(*mda_frame*, *mda_universe*, *leaflets*, *plane*, *nxbins=50*, *nybins=50*)

Bases: object

area_per_lipid()

average_thickness(*return_grid=False*)

curvature()

get_integer_type_arrays()

get_one_array_per_leaflet()

get_xyzc(*leaflet='both'*, *zvalue_dict=None*, *color_dict=None*, *color_grid=None*, *color_type_dict=None*)

grid_to_dict(*in_grid*, *leaflet='upper'*)

map_to_grid(*com_values_dict*, *leaflet='both'*)

thickness_grid()

write_xyz(*leaflet='both'*, *zvalue_dict='Default'*, *out_path='./'*)

Module contents

pybilt.mda_tools package

Submodules

pybilt.mda_tools.mda_area_compressibility_modulus module

Implementation of area compressibilty modulus The are compressibilty modulus is an elastic property that can be computed from the total area (in the lateral dimension) and its fluctuations. The quantity is given by

K_A = kT<A>[<(A - <A>)^2>]^-1,

where A is the area per lipid, k is Boltzmann's constant and T is the temperature (in Kelvin). See references: Coarse Grained Model for Semiquantitative Lipid Simulations Siewert J. Marrink, Alex H. de Vries, and Alan E. Mark J. Phys. Chem. B, 2004, 108 (2), pp 750-760 DOI: 10.1021/jp036508g

Molecular Dynamics Simulations of Cardiolipin Bilayers Martin Dahlberg and Arnold Maliniak J. Phys. Chem. B, 2008, 112 (37), pp 11655-11663 DOI: 10.1021/jp803414g

pybilt.mda_tools.mda_area_compressibility_modulus.**area_compressibility_modulus**(*mda_trajectory*, *temperature*, *normal='z'*, *first=0*, *last=-1*, *interval=1*)

pybilt.mda_tools.mda_density_map module

pybilt.mda_tools.mda_density_profile module

pybilt.mda_tools.mda_deuterium_order_parameter module

pybilt.mda_tools.mda_distance module

pybilt.mda_tools.mda_msd module

pybilt.mda_tools.mda_unwrap module

Module contents

pybilt.plot_generation package

Submodules

pybilt.plot_generation.plot_generation_functions module

A set of functions to generate plots/figures from the lipid bilayer analysis outputs. These functions use matplotlib (http://matplotlib.org/index.html) along with Seaborn (https://stanford.edu/~mwaskom/software/seaborn/index.html).

pybilt.plot_generation.plot_generation_functions.**gen_step_vector_ghost_tails**(*vectors_resnames*, *length=5*, *periodic_cut=75.0*)

pybilt.plot_generation.plot_generation_functions.**gen_step_vector_net_ghost_tails**(*vectors_resna*
length=6,
group_size=2
periodic_cut=75.0

pybilt.plot_generation.plot_generation_functions.**gen_step_vector_smooth_ghost_tails**(*vectors_r*
length=9,
win-
dow=3,
pe-
ri-
odic_cut=

pybilt.plot_generation.plot_generation_functions.**gen_step_vector_smooth_ghost_tails_forward**

pybilt.plot_generation.plot_generation_functions.**plot**(*dat_list*, *yerr_list=None*, *xerr_list=None*, *name_list=None*, *filename='plot.eps'*, *save=True*, *show=False*, *xlabel=None*, *ylabel=None*, *marker=None*, *linestyle=None*, *xticks=None*)

Generic plotting function for (multiple) xy datasets.

Parameters

- **dat_list** (*list or list like*) – List of tuples of data vectors in the format [(x_0, y_0), (x_1. y_1), ...]
- **yerr_list** (*list or list like*) – List of the yerr vectors. e.g. [y_0_err, y_1_err, ...]
- **name_list** (*list or list like, Optional*) – List of string legend names to assign the curves being plotted.
- **filename** (*str, Optional*) – The string containing the path and filename for the exported plot file.
- **save** (*bool, Optional*) – Set whether to save the generated plot to disc with filename. Default: True
- **show** (*bool, Optional*) – Set whether to show the generated plot in an interactive window (i.e. plt.show()). Default: False
- **xlabel** (*str, Optional*) – Specify a x-axis label.
- **ylabel** (*str, Optional*) – Specify a y-axis label
- **marker** (*str, Optional*) – Specify a matplotlib marker type for data points.

pybilt.plot_generation.plot_generation_functions.**plot_area_per_lipid**(*apl_dat_list*, *name_list=None*, *filename='apl.pdf'*, *time_in='ps'*, *time_out='ns'*, *save=True*, *show=False*, *interval=1*, *ylim=None*, *xlim=None*)

Generates a single plot with area per lipid (apl) curves Takes outputs from:

MemSys.CalcAreaPerLipid_Box MemSys.CalcAreaPerLipid_ClosestNeighborCircle

The outputs are passed to function in a list input: apl_dat_list

pybilt.plot_generation.plot_generation_functions.**plot_average_deuterium_op**(*dop_dat_list*, *name_list=None*, *filename='dop.pdf'*, *time_in='ps'*, *time_out='ns'*, *show=False*, *interval=1*)

Generates a single plot of the average deuterium order parameter vs. time

The outputs are passed to function in a list input: dop_dat_list

pybilt.plot_generation.plot_generation_functions.**plot_bilayer_thickness**(*bt_dat_list*, *name_list=None*, *filename='bilayer_thickness.p*
time_in='ps', *time_out='ns'*, *show=False*, *interval=1*, *save=True*, *xlim=None*, *ylim=None*)

Generates a single plot with bilayer thickness curves Takes outputs from:

The outputs are passed to function in a list input: bt_dat_list

pybilt.plot_generation.plot_generation_functions.**plot_corr_mat**(*in_corrmat*, *save=True*, *filename='correlation_matrix.pdf'*, *show=False*)

pybilt.plot_generation.plot_generation_functions.**plot_corr_mat_as_scatter**(*in_corrmat*, *save=True*, *filename='correlation_mat*
show=False)

pybilt.plot_generation.plot_generation_functions.**plot_dc_cluster_dat_number**(*clust_dat_list*, *name_list=None*, *filename='clust_number.pdf'*, *time_in='ps'*, *time_out='ns'*, *save=True*, *show=False*)

Generates a plot of the average number of clusters (vs. time) This function generates a plot of the average number of clusters vs. time for data output with key 'nclusters'

from the 'dc_cluster' analysis in the BilayerAnalyzer, which corresponds to the

bilayer_analyzer.analysis_protocols.DCClusterProtocol analysis protocol class. :Parameters: * **clust_dat_list** (*list*) – This is a list of all data to plot and should be a list of tuples/lists where

each element tuple has the data arrays for each time series curve to include in the plot: e.g. [(times_1, means_1, stds_1), (times_2, means_2, stds_2)]

- **name_list** (*list, optional*) – This is a list of string names to assign each curve included in the plot. It should have len(name_list) == len(clust_dat_list) = True.

  Default: None
- **filename** (*str, optional*) – This a string containing the filename for the output plot if save is set to True. Default: 'clust_number.pdf'
- **time_in** (*str, optional*) – This is a string specifying the time units of values in the input arrays. Acceptible values are 'ps' for picosecond and 'ns' nanosecond.

  Default: 'ps'
- **time_out** (*str, optional*) – This is a string specifying the time units to use in the output plot. Acceptible values are 'ps' for picosecond and 'ns' nanosecond. If this is different than the value for time_in then the time values will be scaled accordingly.

  Default: 'ns'
- **save** (*bool, optional*) – This is boolean switch to set whether or not to save the generated plot to disc. plt.savefig is called.

  Default: True
- **show** (*bool, optional*) – This is boolean switch to set whether or not the generated plot is displayed in interactive mode using plt.show.

  Default: False

pybilt.plot_generation.plot_generation_functions.**plot_dc_cluster_dat_number_comtraj**(*clust_dat_list*, *name_list*, *filename='clust_number.pdf'*, *time_in='ps'*, *time_out='ns'*, *show=False*)

Generates a single of the average number of clusters (vs. time) using output data from:

MemSys.CheckClustering

The outputs are passed to function in a list input: clust_dat_list

pybilt.plot_generation.plot_generation_functions.**plot_dc_cluster_dat_size**(*clust_dat_list*, *name_list=None*, *filename='clust_number.pdf'*, *time_in='ps'*, *time_out='ns'*, *save=True*, *show=False*)

Generates a plot of the average cluster size (vs. time) This function generates a plot of the average cluster size vs. time for data output with key 'avg_size' from the 'dc_cluster' analysis in the BilayerAnalyzer, which corresponds to the bilayer_analyzer.analysis_protocols.DCClusterProtocol analysis protocol class. :Parameters: * **clust_dat_list** (*list*) – This is a list of all data to plot and should be a list of tuples/lists where

each element tuple has the data arrays for each time series curve to include in the plot: e.g. [(times_1, means_1, stds_1), (times_2, means_2, stds_2)]

- **name_list** (*list, optional*) – This is a list of string names to assign each curve included in the plot. It should have len(name_list) == len(clust_dat_list) = True.

  Default: None

- **filename** (*str, optional*) – This a string containing the filename for the output plot if save is set to True. Default: 'clust_number.pdf'

- **time_in** (*str, optional*) – This is a string specifying the time units of values in the input arrays. Acceptible values are 'ps' for picosecond and 'ns' nanosecond.

  Default: 'ps'

- **time_out** (*str, optional*) – This is a string specifying the time units to use in the output plot. Acceptible values are 'ps' for picosecond and 'ns' nanosecond. If this is different than the value for time_in then the time values will be scaled accordingly.

  Default: 'ns'

- **save** (*bool, optional*) – This is boolean switch to set whether or not to save the generated plot to disc. plt.savefig is called.

  Default: True

- **show** (*bool, optional*) – This is boolean switch to set whether or not the generated plot is displayed in interactive mode using plt.show.

  Default: False

pybilt.plot_generation.plot_generation_functions.**plot_dc_cluster_dat_size_comtraj**(*clust_dat_list*, *name_list=None*, *filename='clust_number.pdf'*, *time_in='ps'*, *time_out='ns'*, *show=False*)

Generates a single plot of the average cluster size (vs time) using output data from:

MemSys.CheckClustering

The outputs are passed to function in a list input: clust_dat_list

pybilt.plot_generation.plot_generation_functions.**plot_dc_cluster_maps_comtraj**(*clusters*, *filename='cluster_ma* *show=False*)

Generates a single plot of the lipid cluster map Takes a single frame of the output from:

MemSys.ExportClustersForPlotting

pybilt.plot_generation.plot_generation_functions.**plot_density_profile**(*dp_out_list*, *save=True*, *filename='density_profile.pdf'*, *show=False*, *label_list=None*, *ylabel='Density'*)

Plot density profiles This function can be used to plot the results of density profiles functions in the mda_density_profile module.

Parameters

- **dp_out_list** (*list of tuples*) – A list of the tuple outputs of the profile calculation functions
- **save** (*bool, optional*) – Default is True. Saves the plot output as an image file if True.
- **filename** (*str, optional*) – The name out the image file that will be created if save=True.
- **show** (*bool, optional*) – Default is False. Display the plot (plt.show) if True.
- **label_list (list of str** – None, optional): Default is None. Allows a list of strings used to label the plot lines.

pybilt.plot_generation.plot_generation_functions.**plot_displacement_lipid_type_cross_correla**

pybilt.plot_generation.plot_generation_functions.**plot_ebar_hists**(*center_count_err*, *name_list=None*, *filename='ebar_hist.eps'*, *save=True*, *show=False*, *xlabel=None*, *ylabel='Counts'*)

Generic plotting function for (multiple) xy datasets.

Parameters

- **dat_list** (*list or list like*) – List of tuples of data vectors in the format [(x_0, y_0), (x_1. y_1), ...]
- **yerr_list** (*list or list like*) – List of the yerr vectors. e.g. [y_0_err, y_1_err, ...]
- **name_list** (*list or list like, Optional*) – List of string legend names to assign the curves being plotted.

- **filename** (*str, Optional*) – The string containing the path and filename for the exported plot file.
- **save** (*bool, Optional*) – Set whether to save the generated plot to disc with filename. Default: True
- **show** (*bool, Optional*) – Set whether to show the generated plot in an interactive window (i.e. plt.show()). Default: False
- **xlabel** (*str, Optional*) – Specify a x-axis label.
- **ylabel** (*str, Optional*) – Specify a y-axis label
- **marker** (*str, Optional*) – Specify a matplotlib marker type for data points.

pybilt.plot_generation.plot_generation_functions.**plot_grid_as_scatter**(*in_xyzc*, *save=True*, *filename='lipid_grid.pdf'*, *show=False*, *colorbar=False*, *cmap=None*, *vmin=None*, *vmax=None*)

pybilt.plot_generation.plot_generation_functions.**plot_lipid_grid_thickness_map_2d**(*x_centers*, *y_centers*, *thickness_grid*, *save=True*, *filename='bilay*
show=False
colorbar=True
vmin=0.0
vmax=None
interpolation='none'

pybilt.plot_generation.plot_generation_functions.**plot_msd**(*msd_dat_list*, *name_list=None*, *filename='msd.pdf'*, *time_in='ps'*, *time_out='ns'*, *show=False*, *interval=1*, *save=True*)

Generates a single plot with Mean Squared Displacement curves Takes outputs from:

MemSys.CalcMSD MemSys.CalcMSD_parallel

The outputs are passed to function in a list input: apl_dat_list

pybilt.plot_generation.plot_generation_functions.**plot_position_density_map_2d**(*x_centers*, *y_centers*, *counts*, *save=True*, *filename='position_d*, *show=False*, *colorbar=True*, *vmin=0.0*, *vmax=None*, *normalized=False*, *scaled_to_max=Fa*, *interpolation='none'*)

pybilt.plot_generation.plot_generation_functions.**plot_position_density_map_2d_scatter**(*x_cent*, *y_cent*, *counts*, *save=*, *filename=*, *show=*, *colorbar=T*, *vmin=*, *vmax=*, *normalized=*, *scaled_*

pybilt.plot_generation.plot_generation_functions.**plot_spark**(*xdat*, *ydat*, *filename='plot.eps'*, *save=True*, *show=False*, *color='#47d147'*)

Generic plotting function for (multiple) xy datasets. Based on matplotlib spark_line function defined here: https://markhneedham.com/blog/2017/09/23/python-3-create-sparklines-using-matplotlib/ :Parameters: * **dat_list** (*list or list like*) – List of tuples of data vectors in the format [(x_0, y_0), (x_1. y_1), . . .]

- **yerr_list** (*list or list like*) – List of the yerr vectors. e.g. [y_0_err, y_1_err, . . .]
- **name_list** (*list or list like, Optional*) – List of string legend names to assign the curves being plotted.
- **filename** (*str, Optional*) – The string containing the path and filename for the exported plot file.
- **save** (*bool, Optional*) – Set whether to save the generated plot to disc with filename. Default: True
- **show** (*bool, Optional*) – Set whether to show the generated plot in an interactive window (i.e. plt.show()). Default: False

- **xlabel** (*str, Optional*) – Specify a x-axis label.
- **ylabel** (*str, Optional*) – Specify a y-axis label
- **marker** (*str, Optional*) – Specify a matplotlib marker type for data points.

pybilt.plot_generation.plot_generation_functions.**plot_spark_error**(*xdat*, *ydat*, *error*, *filename='plot.eps'*, *save=True*, *show=False*, *color='#47d147'*)

Generic plotting function for (multiple) xy datasets. Based on matplotlib spark_line function defined here: https://markhneedham.com/blog/2017/09/23/python-3-create-sparklines-using-matplotlib/ :Parameters: * **dat_list** (*list or list like*) – List of tuples of data vectors in the format [(x_0, y_0), (x_1. y_1), ...]

- **yerr_list** (*list or list like*) – List of the yerr vectors. e.g. [y_0_err, y_1_err, ...]
- **name_list** (*list or list like, Optional*) – List of string legend names to assign the curves being plotted.
- **filename** (*str, Optional*) – The string containing the path and filename for the exported plot file.
- **save** (*bool, Optional*) – Set whether to save the generated plot to disc with filename. Default: True
- **show** (*bool, Optional*) – Set whether to show the generated plot in an interactive window (i.e. plt.show()). Default: False
- **xlabel** (*str, Optional*) – Specify a x-axis label.
- **ylabel** (*str, Optional*) – Specify a y-axis label
- **marker** (*str, Optional*) – Specify a matplotlib marker type for data points.

pybilt.plot_generation.plot_generation_functions.**plot_spark_error_marker**(*xdat*, *ydat*, *error*, *filename='plot.eps'*, *save=True*, *show=False*, *color='#47d147'*, *marker='s'*)

Generic plotting function for (multiple) xy datasets. Based on matplotlib spark_line function defined here: https://markhneedham.com/blog/2017/09/23/python-3-create-sparklines-using-matplotlib/ :Parameters: * **dat_list** (*list or list like*) – List of tuples of data vectors in the format [(x_0, y_0), (x_1. y_1), ...]

- **yerr_list** (*list or list like*) – List of the yerr vectors. e.g. [y_0_err, y_1_err, ...]
- **name_list** (*list or list like, Optional*) – List of string legend names to assign the curves being plotted.
- **filename** (*str, Optional*) – The string containing the path and filename for the exported plot file.
- **save** (*bool, Optional*) – Set whether to save the generated plot to disc with filename. Default: True
- **show** (*bool, Optional*) – Set whether to show the generated plot in an interactive window (i.e. plt.show()). Default: False
- **xlabel** (*str, Optional*) – Specify a x-axis label.
- **ylabel** (*str, Optional*) – Specify a y-axis label
- **marker** (*str, Optional*) – Specify a matplotlib marker type for data points.

pybilt.plot_generation.plot_generation_functions.**plot_spark_marker**(*xdat*, *ydat*, *filename='plot.eps'*, *save=True*, *show=False*, *color='#47d147'*, *marker='s'*)

Generic plotting function for (multiple) xy datasets. Based on matplotlib spark_line function defined here: https://markhneedham.com/blog/2017/09/23/python-3-create-sparklines-using-matplotlib/ :Parameters: * **dat_list** (*list or list like*) – List of tuples of data vectors in the format [(x_0, y_0), (x_1. y_1), ...]

- **yerr_list** (*list or list like*) – List of the yerr vectors. e.g. [y_0_err, y_1_err, ...]
- **name_list** (*list or list like, Optional*) – List of string legend names to assign the curves being plotted.
- **filename** (*str, Optional*) – The string containing the path and filename for the exported plot file.
- **save** (*bool, Optional*) – Set whether to save the generated plot to disc with filename. Default: True
- **show** (*bool, Optional*) – Set whether to show the generated plot in an interactive window (i.e. plt.show()). Default: False
- **xlabel** (*str, Optional*) – Specify a x-axis label.
- **ylabel** (*str, Optional*) – Specify a y-axis label
- **marker** (*str, Optional*) – Specify a matplotlib marker type for data points.

pybilt.plot_generation.plot_generation_functions.**plot_spark_multi**(*xdats*, *ydats*, *filename='plot.eps'*, *save=True*, *show=False*, *colors=None*)

Generic plotting function for (multiple) xy datasets. Based on matplotlib spark_line function defined here: https://markhneedham.com/blog/2017/09/23/python-3-create-sparklines-using-matplotlib/ :Parameters: * **dat_list** (*list or list like*) – List of tuples of data vectors in the format [(x_0, y_0), (x_1. y_1), ...]

- **yerr_list** (*list or list like*) – List of the yerr vectors. e.g. [y_0_err, y_1_err, ...]
- **name_list** (*list or list like, Optional*) – List of string legend names to assign the curves being plotted.
- **filename** (*str, Optional*) – The string containing the path and filename for the exported plot file.
- **save** (*bool, Optional*) – Set whether to save the generated plot to disc with filename. Default: True
- **show** (*bool, Optional*) – Set whether to show the generated plot in an interactive window (i.e. plt.show()). Default: False
- **xlabel** (*str, Optional*) – Specify a x-axis label.
- **ylabel** (*str, Optional*) – Specify a y-axis label
- **marker** (*str, Optional*) – Specify a matplotlib marker type for data points.

pybilt.plot_generation.plot_generation_functions.**plot_step_vectors**(*vectors_resnames*, *filename='step_vectors.pdf'*, *save=True*, *show=False*, *scaled=False*, *wrapped=False*, *ghost_tails=None*, *ghost_tail_alpha=0.5*, *ghost_tail_arrow=False*, *ylim=None*, *xlim=None*)

Generates a single plot with the lipid displacement vectors (or step vectors) Takes a single frame of the output from:

MemSys.StepVector

Corresponding colors (if multiple lipid types are included) can be generated using:

MemSys.StepVectorColors

pybilt.plot_generation.plot_generation_functions.**plot_step_vectors_comtraj**(*vectors*, *colors=None*, *filename='step_vectors.pd*
show=False, *save=True*)

Generates a single plot with the lipid displacement vectors (or step vectors) Takes a single frame of the output from:

MemSys.StepVector

Corresponding colors (if multiple lipid types are included) can be generated using:

MemSys.StepVectorColors

pybilt.plot_generation.plot_generation_functions.**plot_step_vectors_stroboscopic**(*vectors_resnam*
index=0, *filename='step_ve*
save=True, *show=False*, *scaled=False*, *wrapped=False*

Generates a stroboscopic trajectory plot with the displacement vectors (or step vectors) of a single lipid. Takes the output from the 'disp_vec' analysis of the bilayer_analyzer:

pybilt.plot_generation.plot_generation_functions.**plot_xygrid_as_imshow**(*x_centers*, *y_centers*, *grid*, *filename='grid.pdf'*, *save=True*, *show=False*, *colorbar=False*, *colorbar_label=None*, *cmap=None*, *vmin=None*, *vmax=None*, *interpolation='none'*, *xlabel=None*, *ylabel=None*)

pybilt.plot_generation.plot_generation_functions.**update_rcparams**(*rcparams*)

Module contents

10.1.2 Module contents

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

I

K

L

M

N

P

R

S

T

U

V

W

X

Y

Z