LiPyphilic: A Python toolkit for the analysis of lipid membrane simulations

Paul Smith* and Christian D. Lorenz*

Department of Physics, King's College London, London, WC2R 2LS, UK

E-mail: paul.smith@kcl.ac.uk; chris.lorenz@kcl.ac.uk

Abstract

Molecular dynamics simulations are now widely used to study emergent phenomena in lipid membranes with complex compositions. Here, we present LiPyphilic - a fast, fully tested, and easy to install Python package for analysing such simulations. Analysis tools in LiPyphilic include the identification of cholesterol flip-flop events, the classification of local lipid environments, and the degree of interleaflet registration. LiPyphilic is both force field and resolution agnostic, and thanks to the powerful atom selection language of MDAnalysis it can handle membranes with highly complex compositions. LiPyphilic also offers two on-the-fly trajectory transformations to i) fix membranes split across periodic boundaries and ii) perform nojump coordinate unwrapping. Our implementation of nojump unwrapping accounts for fluctuations in box volume under the NPT ensemble — an issue that most current implementations have overlooked. The full documentation of LiPyphilic, including installation instructions, is available at <https://lipyphilic.readthedocs.io/en/latest>.

1 Introduction

The plasma membrane was once thought to be a passive divider between a cell and its external environment. We now understand that it is in fact a dynamic interface upon which many cellular processes, from cell-signaling to membrane transport, depend. These processes are emergent phenomena that arise from a complex interplay between the molecular species that comprise the plasma membrane. As such, there is a great interest in understanding how the lipids, proteins, and carbohydrates of the plasma membrane interact with one another.

Molecular dynamics (MD) simulations are routinely used to study lipid-lipid and lipid-protein interactions at a molecular level, and there exists many excellent tools for analysing the trajectories of such simulations. FATSLiM¹ and MemSurfer,² for example, both specialise in the analysis of non-planar membranes such as buckled bilayers or vesicles. PyLipID³ and ProLint⁴ are designed for the easy and efficient analysis of lipid-protein interactions. ML-LPA is a recently developed Python package that employs various machine learning algorithms to identify the phase — L_o or L_d — of lipids in a bilayer.⁵ LOOS, on the other hand, is a C++ library with a Python interface for analysing MD simulations.^{6,7} Unlike the above packages, LOOS handles the trajectory reading internally whilst also offering a large set of analysis tools, some of which are for lipid membranes. Between them, these software packages provide an extensive analysis suite for MD simulations of lipid membranes.

There are, however, some non-trivial analyses that are frequently employed but are not yet available in any analysis software we are aware of. These include the identification of cholesterol flip-flop events,^{8–26} the classification of local lipid environments,^{17,27–34} and calculating the degree of interleaflet registration.^{11,34–42} These analyses provide important information about the structure and dynamics of lipid membranes, but they currently require the writing of in-house scripts. Here, we present LiPyphilic — a fast, fully tested, and easy to install Python package that can perform these analyses, among others.

2 LiPyphilic

LiPyphilic is an object-oriented Python package for analysing MD simulations of lipid membranes. It is built directly on top of MDAnalysis, and makes use of NumPy⁴³ and SciPy⁴⁴ for efficient computation. It is both resolution and force field agnostic, working with any file format that MDAnalysis can load so long as the topology contains residue names. All analysis classes in LiPyphilic share the same application programming interface (API) as those in MDAnalysis. This shared API makes it simple for users of MDAnalysis to learn how to use LiPyphilic.

At its core, LiPyphilic is designed to easily integrate with the wider scientific Python stack. Results are typically stored in a two dimensional NumPy array of shape N_{lipids} by N_{frames} (Figure 1), making it simple to post-process the results for further analysis. Some analysis tools also take a two dimensional NumPy array of the same shape as input. This input array may contain information about each lipid, such as which leaflet they belong to or their phase (L_o or L_d). Alternatively, the input array may be a boolean mask — an array of True and False values specifying which lipids to include in the analysis. As these inputs are generic NumPy arrays instead of types specific to LiPyphilic, it is possible to use the output from other membrane analysis tools as input to LiPyphilic. For example, you may assign lipids to leaflets using FATS LiM,¹ determine their phase state using ML-LPA,⁵ or calculate local membrane normals using MemSurfer,² and extract the results to perform further analysis with LiPyphilic.

Figure 1: In LiPyphilic, analysis results are typically stored in a NumPy array of shape (N_{Lipids}, N_{frames})

The workflow for using LiPyphilic generally involves the following steps:

1. import MDAnalysis along with the required LiPyphilic analysis modules
2. load a topology and trajectory as an MDAnalysis universe
3. create an analysis object using the MDAnalysis universe and specifying the relevant input options
4. use the `run()` method to perform the analysis
5. store the results, either by serialising the analysis object itself or saving the results data as a NumPy array

Below we will discuss the implementation and usage of some of the analysis tools currently available in LiPyphilic. We will then discuss the on-the-fly transformations that LiPyphilic can perform on MDAnalysis trajectories. We then provide benchmarks of the analysis tools and transformations. Finally, we will briefly discuss the software engineering best practices used in developing LiPyphilic. If you are more interested in learning how to use LiPyphilic, rather than how LiPyphilic works per se, we recommend working through the examples in the documentation at <https://lippyphilic.readthedocs.io/en/latest/reference/analyses.html>.

2.1 Assign leaflets

For many analyses, such as calculating the area per lipid, it is necessary to know the leaflet within which a lipid is found. LiPyphilic has two tools for assigning lipids to leaflets. The class `lippyphilic.lib.assign_leaflets.AssignLeaflets` assigns each lipid to a leaflet based on the distance in z to its local membrane midpoint. This is suitable only for planar bilayers. On the other hand, the class `lippyphilic.lib.assign_leaflets.AssignCurvedLeaflets` can be used to identify leaflets in a buckled bilayer or a micelle. This uses the MDAnalysis leaflet finder^{45,46} to assign non-translocating lipids to leaflets, then at each frame assigns the remaining lipids

based on their minimum distance to each leaflet. `AssignLeaflets` remains useful for planar bilayers, especially if the rate of cholesterol translocation is of interest, which is typically measured by assigning lipids to leaflets based on their z -coordinate.

LiPyphilic can assign molecules not just to the upper or lower leaflet, but also to the midplane. This is useful for studying, for example, the local lipid environment of midplane cholesterol²⁴ or its role in the registration of nanodomains.⁴⁷ Assigning cholesterol to the midplane also creates a buffer zone for determining whether a flip-flop event was successful (i.e. crossed the buffer zone) or not.²⁵

`AssignLeaflets` and `AssignCurvedLeaflets` have opposing approaches to assigning molecules to the midplane. The former considers a molecule's distance to the midplane whereas the latter considers its distance to each leaflet. There is naturally an inverse relationship between these two measures - the further a molecule is from the midplane the closer it is to a leaflet. However, the distance to the midplane is typically employed when studying flip-flop in planar bilayers,^{17,21-25,31} whereas the distance to each leaflet is used for studying flip-flop in undulating bilayers.⁴⁷

As with all analysis tools in LiPyphilic, the assigning of lipids to leaflets is both resolution and force field agnostic. Instead of reading atom selections hard coded into the package, the analysis tools rely on the powerful selection language of MDAnalysis. Figure 2A shows how `AssignLeaflets` may be used to determine the leaflet membership of all lipids in the 58 component neuronal plasma membrane studied by Ingólfsson *et alii*.³¹ First, an MDAnalysis Universe must be created. Lipids are then assigned to leaflets by passing this Universe to `AssignLeaflets` along with an atom selection of lipids in the bilayer, using the `universe` and `lipid_sel` arguments respectively. Optionally, to allow molecules to be in the midplane, we can use the `midplane_sel` and `midplane_cutoff` arguments. In the example shown in Figure 2A, cholesterol will be assigned to the midplane if its ROH bead is within 8 Å of its local midpoints. Local midpoints are computed by first splitting the membrane into an n by n grid in xy , where n is specified using the `n_bins` argument. The local midpoint of a grid cell

A

```
1 import MDAnalysis as mda
2 from lipyphilic.lib.assign_leaflets import AssignLeaflets
3
4 u = mda.Universe("production.tpr", "production.xtc")
5
6 leaflets = AssignLeaflets(
7     universe=u,
8     lipid_sel="name GL1 GL2 AM1 AM2 ROH",
9     midplane_sel="name ROH",
10    midplane_cutoff=8,
11    n_bins=20
12 )
13
14 leaflets.run(start=None, stop=None, step=None, verbose=True)
```

B

Figure 2: LiPyphilic can assign lipids to the upper leaflet, lower leaflet or midplane. **A.** Workflow for assigning leaflets. **B.** Lipids in the neuronal plasma membrane studied by Ingólfsson et al.³¹ are assigned to the upper leaflet (blue), lower leaflet (red) or midplane (yellow).

is then given by the center of mass of all atoms selected by `lipid_sel` that are in the grid cell. Through calculating local membrane midpoints, this algorithm can account for small undulations in a bilayer. However, for bilayers with large undulations or for non-bilayer membranes, `AssignCurvedLeaflets` should be used.

After creating `leaflets` as described above, the analysis is performed by calling the `run` method. Here the `start`, `stop` and `step` arguments are used to specify which frames of the trajectories to use, and a progress bar can be displayed on the screen by setting `verbose=True`. Leaflet data are then stored in the `leaflets.leaflets` attribute as a two-dimensional NumPy array. Each row in the results array corresponds to an individual lipid and each column to an individual frame. For example, `leaflets.leaflets[i, j]` contains the leaflet membership

of lipid i at frame j . `leaflets.leaflets[i, j]` is equal to 1 if the lipid is in the upper leaflet, -1 if the lipid is in the lower leaflet, or 0 if the lipid is in the midplane.

2.2 Flip flop

Cholesterol is unevenly distributed across the plasma membrane, although the precise distribution is still under debate.⁴⁸ This uneven distribution plays an important role in numerous cellular processes, and is maintained through the ultrafast spontaneous translocation, or flip-flop, of cholesterol across leaflets.

With recent advances in computing power, sterol flip-flop can now be studied directly using either coarse-grained, united-atom, or all-atom simulations. Such simulations can be used to study the flip-flop process itself or to extract rates directly from the number of observed flip-flop events. Below we describe a general analysis tool in LiPyphilic for identifying such flip-flop events in MD simulations.

The class `lipypyphilic.lib.flip_flop.FlipFlop` can be used to identify successful and aborted flip-flop events. Figure 3A illustrates how to do so using the output from `AssignLeaflets`. The same MDAnalysis Universe that is used for assigning leaflets is passed to `FlipFlop`. An atom selection that specifies which molecules to consider when identifying flip-flop events is passed to the `lipid_sel` argument. The leaflet membership of each lipid selected by `lipid_sel` is passed to the `leaflets` argument. In this example, this is achieved by filtering the results array of `AssignLeaflets` to include only the leaflet membership of cholesterol molecules. The leaflet membership must be a NumPy array, of shape $((N_{lipids}, N_{frames}))$, in which each element is equal to:

- 1 if the lipid is in the upper/outer leaflet
- -1 if the lipid is in the lower/inner leaflet
- 0 if the lipid is in the midplane

A

```
1 from lipophilic.lib.flip_flop import FlipFlop
2
3 flip_flop = FlipFlop(
4     universe=u,
5     lipid_sel="name ROH",
6     leaflets=leaflets.filter_leaflets("name ROH"),
7     frame_cutoff=2
8 )
9
10 flip_flop.run(start=None, stop=None, step=None, verbose=True)
```

B

```
1 n_flip_flops = sum(flip_flop.flip_flop_success == "Success")
2
3 flip_flop_rate = n_flip_flops / (n_cholesterol * total_simulation_time)
```

Figure 3: **A.** Identify all cholesterol flip-flop events, based on the leaflet membership of cholesterol at each frame. **B.** Determine the cholesterol flip-flop rate directly from the number of flip-flop events.

In this example, the `frame_cutoff` argument is used to specify that a molecule must remain in its new leaflet for at least two consecutive frames in order for the flip-flop to be considered successful.

We again call the `run` method to perform the analysis. For each molecule, LiPyphilic will then identify the frames at which it leaves one leaflet and enters another for at least `frame_cutoff` frames. If the new leaflet is different to the original leaflet, the flip-flop was successful.

The success or failure of each flip-flop event is stored in a one-dimensional NumPy array, accessible via the `flip_flop.flip_flop_success` attribute. Elements in this array are strings, equal to either "Success" or "Failure". From this results array, it is easy to calculate the flip-flop rate based on the number of observed events, the number of cholesterol molecules in the membrane, and the total simulation time (Figure 3B).

The example in Figure 3A shows how to use the results from `AssignLeaflets` to identify flip-flop events. However, leaflets need not be identified using `LiPyphilic` in order to use the `FlipFlop` analysis tool. `FlipFlop` expects a NumPy array of leaflet membership as described above. Flip-flop events can thus be found even if leaflets were assigned using, for example, `FATSLiM`¹ or user-written scripts⁴⁹ based on the MDAnalysis `LeafletFinder` tool.

Gu et al. showed that translocation is highly influenced by the local lipid environment of a sterol.²⁴ `FlipFlop` therefore not only returns the success or failure of each event, but also the frame at which the flip-flop process begins and ends along with the residue index of the flip-flopping molecule. This information is stored as a two dimensional NumPy array in the `flip_flop.flip_flops` attribute, where each row corresponds to an individual event and each column contains the:

- residue index of the flip-flopping molecule
- frame at which the molecule left its original leaflet
- frame at which the molecule entered its new leaflet
- numerical identifier of its new leaflet: 1 for the upper leaflet and -1 for the lower leaflet

This information enables further analysis, such as a consideration of the local lipid environment before and after translocation.

Cholesterol is typically the only molecule to flip-flop during an MD simulation. However, ceramides and diacylglycerols, as well as fatty acids such as docosahexaenoic acid, also have fast flip-flop rates. To find flip-flop events of a molecule other than cholesterol, simply change the lipid selection passed to `FlipFlop`. For example, to find all flip-flop events for ceramides in the neuronal plasma membrane, change `lipid_sel="resname ROH"` to `lipid_sel="resname ??CE"` and pass the corresponding leaflet membership NumPy array to the `leaflets` argument. This again makes use of the powerful selection language of MDAnalysis, and the fact that all ceramides in the MARTINI force field have residue names that are four characters long and end in 'CE'.

2.3 Registration

The translocation of cholesterol across leaflets is thought to be important in several cellular process, including the modulation of the lateral heterogeneity of the membrane.⁴⁷ Recently, transient nanodomains of L_O phase lipids were observed in live mammalian cell plasma membranes.⁵⁰ These nanodomains are thought to be enriched in sphingomyelin and cholesterol, and to act as functional platforms for cell signaling. However, their nature, formation, and roles in cellular processes are still not fully understood.

There is particular interest in understanding under what conditions nanodomains in apposing leaflets are spatially aligned. Such alignment is known as interleaflet registration. This has been the subject of several MD simulations, which have revealed registration to be a complex process.^{11,34-42} Registration is modulated by many factors, including the length and saturation of lipid tails as well as the relative affinity of cholesterol for the different lipid species in a domain-forming mixture.

The class `liplyphilic.lib.registration.Registration` can be used to quantify the degree of interleaflet registration in a planar bilayer. `Registration` is an implementation of the registration analysis described by Thallmair *et alii*.⁴⁰ The degree of registration is calculated as the Pearson correlation coefficient of molecular densities in the upper and lower leaflets. First, the two-dimensional density of each leaflet is calculated:

$$\rho(x, y)_L = \int_{-\infty}^{\infty} \frac{1}{2\pi\sigma^2} \exp\left(-\frac{1}{2}\left[\left(\frac{x' - x}{\sigma}\right)^2 + \left(\frac{y' - y}{\sigma}\right)^2\right]\right) dx' dy'$$

where the (x, y) positions of lipid atoms in leaflet L are binned into two-dimensional histograms with bin lengths of 1 Å. L is either the upper (u) or lower (l) leaflet. The two-dimensional density is then convolved with a circular Gaussian density of standard deviation σ . The registration between the two leaflets, $r_{u/l}$, is then calculated as the Pearson correlation coefficient between $\rho(x, y)_u$ and $\rho(x, y)_l$. Values of $r_{u/l} = 1$ correspond to perfectly registered domains and values of $r_{u/l} = -1$ correspond to perfectly anti-registered domains.

```

1 from lipophilic.lib.registration import Registration
2
3 registration = Registration(
4     universe=u,
5     upper_sel="name ROH",
6     lower_sel="name ROH",
7     leaflets=leaflets.filter_leaflets("name ROH")
8 )
9
10 registration.run(start=None, stop=None, step=None, verbose=True)

```

Figure 4: Calculate the interleaflet registration of cholesterol at each frame.

The atoms used in calculating the interleaflet registration are specified by passing selection strings to the `upper_sel` and `lower_sel` arguments of `Registration` (Figure 4). The leaflet membership of all atoms in the two selections must be passed to the `leaflets` argument. As before, this must be a two-dimensional NumPy array of shape (N_{lipids}, N_{frames}) . The results are stored in the `registration.registration` attribute as a one-dimensional NumPy array of length N_{frames} . The array contains the Pearson correlation coefficient of the two-dimensional leaflet densities at each frame.

The example in Figure 4 demonstrates how to compute the registration of cholesterol across the upper and lower leaflets. However, in simulations of phase-separating mixtures it is useful to know the degree of registration of L_o domains, rather than the registration of a specific molecular species. `Registration` can be used to calculate this, providing the phase of each lipid at each frame has been determined using for example, ML-LPA,⁵ a hidden Markov Model,^{27,33} or the deuterium order parameter of lipid tails. If the lipid phase data is stored in a two-dimensional NumPy array of shape (N_{lipids}, N_{frames}) , it can be used to create a boolean mask that will tell `Registration` which lipids to include in the analysis. For example, if our array is named `lipid_phase_data`, and its elements are strings of either "Lo" or "Ld", then we can select the L_o lipids for analysis by passing the boolean mask `lipid_phase_data == "Lo"` to the `filter_by` argument of `Registration`.

2.4 Neighbour matrix

The plasma membrane is comprised of hundreds of different lipid species. In this complex mixture, lateral heterogeneities and aggregates of specific lipid species arise spontaneously. Over the past decade, this compositional complexity has begun to feature in MD simulations of membranes.^{17,25,31,51–56} In these simulations, the lateral organization of the membrane is typically quantified via a consideration of local lipid environments. Specifically, the lipid enrichment index of species B around species A , E_{AB} , may be defined as:¹⁷

$$E_{AB} = N_{AB} / \langle N_B \rangle$$

where N_{AB} is the number of molecules of species B around species A , and $\langle N_B \rangle$ is the mean number of species B around any species.

The class `liiphilic.lib.neighbours.Neighbours` provides methods for computing the lipid enrichment index and for identifying the largest cluster of a specific species of lipids over time. Both of these analyses first require the construction of an adjacency matrix, A , that describes whether each pair of lipid molecules are neighbouring one another or not. Two lipids are considered neighbours if they have any atoms within a given cutoff distance, d_{cutoff} , of one another. The adjacency matrix can be created by passing an atom selection and a value of d_{cutoff} to the `lipid_sel` and `cutoff` arguments, respectively, of `Neighbours` (Figure 5A). The `run` method is called to construct an adjacency matrix for the frames of the trajectory specified using the `start`, `stop`, and `step` arguments. The results are available in the `neighbours.neighbours` attribute as a two-dimensional SciPy sparse matrix of shape $(N_{Lipids}, N_{Lipids}N_{frames})$. This matrix can be used for further analysis, either via helper methods of `Neighbours` or via user-written scripts.

`Neighbours` initially constructs the output matrix as a SciPy `lil_matrix`, a row-based list-of-lists sparse matrix. At the end of the analysis, this is converted into a SciPy `csc_matrix`. This is because the `lil_matrix` is efficient at adding new neighbours on-the-fly, whereas the

A

```
1 from lipphilic.lib.neighbours import Neighbours
2
3 neighbours = Neighbours(
4     universe=u,
5     lipid_sel="name GL1 GL2 AM1 AM2 ROH",
6     cutoff=12
7 )
8
9 neighbours.run(start=None, stop=None, step=None, verbose=True)
```

B

```
1 largest_cluster, lipid_indices = neighbours.largest_cluster(
2     cluster_sel="name GM*",
3     filter_by=leaflets.leaflets == 1,
4     return_indices=True
5 )
```

C

```
1 counts, enrichment = neighbours.count_neighbours(return_enrichment=True)
```

Figure 5: **A** Create the neighbour adjacency matrix. **B** Find the largest cluster of glycolipids at each frame as well as the residue indices of lipids in the largest cluster. **C** Calculate the enrichment/depletion index of each lipid species.

`csc_matrix` is efficient for slicing columns of the matrix. The latter is used frequently in the `largest_cluster` and `count_neighbours` helper methods of `Neighbours`.

2.4.1 Largest cluster

Some glycolipids in the plasma membrane are known to aggregate, forming platforms for cell-signalling.^{57–59} The size of the largest cluster of glycolipids in the neuronal plasma membrane³¹ can be calculated using the `largest_cluster` method of `Neighbours` (Figure 5B). For this, an atom selection must be provided to the `cluster_sel` argument. In the example in Figure 5B, it is specified that only lipids in the upper leaflet should be included in the cal-

ulation by passing a boolean mask to the `filter_by` keyword. The `return_indices` argument is used to specify that the residue indices of the lipid molecules in the largest cluster at each frame are also to be returned. There is no need to call a `run` method nor to specify which frames of the analysis to use — the same frames specified in 5A will be used for the cluster analysis.

The results are not stored in an attribute in our `neighbours` object. Instead, the largest cluster size and the residue indices of the lipid molecules in the largest cluster are each returned as a NumPy array. The former is a one-dimensional array containing the number of lipids in the largest cluster at each frame. The latter is a list of NumPy arrays. Each array in the list corresponds to a single frame and contains the residue indices of the lipid molecules in the largest cluster at that frame. Knowing the indices of the lipids in the largest cluster allows for further analysis, such as calculating the lateral diffusion coefficient of lipid molecules in the cluster.³⁴

To find the largest cluster at a given frame, the `neighbours.neighbours` sparse adjacency matrix is first sliced to give a matrix for the current frame only, A_{frame} . The `connected_components` function of SciPy is then used to find all connected components at the current frame. NumPy’s `unique` function, with `return_counts` set to `True`, is then used to identify the largest connected component and thus the largest cluster size.

2.4.2 Enrichment index

After constructing the adjacency matrix, the `count_neighbours` method can be used to determine the local environment of each lipid molecule (Figure 5C). For a single lipid, its local lipid environment is defined as the number of neighbours of each species. In the example in Figure 5C, `return_enrichment=True` is set to specify that the lipid enrichment index is also to be returned.

As with `neighbours`, the results are not stored in an attribute in our `neighbours` object. The neighbour counts and the enrichment index are each returned as a Pandas `DataFrame`.

The `DataFrame` of neighbour counts contains — for each lipid at each frame — the residue name, lipid residue index, frame number, number of neighbours of each species, and total number of neighbours. The lipid enrichment `DataFrame` contains the enrichment index of each lipid species at each frame. This data can easily be used to calculate the mean enrichment of each species, or it can be plotted over time to determine whether the lateral mixing of lipids has equilibrated.

2.4.3 User-defined counts

By default, the `count_neighbours` method will calculate the number of neighbouring species around each individual lipid. This is done using the residue name of each lipid. However, it is also possible to use any ordinal or string data for counting lipid neighbours. For example, the enrichment index of lipids in the neuronal plasma membrane can be calculated based on their tail saturation (Figure 6). For this, first a two-dimensional NumPy array of shape (N_{lipids}, N_{frames}) that contains the saturation of each lipid needs to be created (Figure S1). Then this array is passed to the `count_by` argument of `count_neighbours`, and the local lipid environment and enrichment index will be determined based on the information in this array.

Figure 6: The enrichment/depletion index of lipids in the neuronal plasma membrane based on their tail saturation.

B

```
1 from lipphilic.transformations import center_membrane
2
3 membrane = u.select_atoms("same residue as name GL1 GL2 AM1 AM2 ROH")
4
5 u.trajectory.add_transformations(
6     center_membrane(
7         ag=membrane,
8         shift=10,
9         center_x=False,
10        center_y=False,
11        center_z=True
12    )
13 )
```

Figure 7: **A** A membrane split across periodic boundaries can be made whole and centered by iteratively translating the system of particles. After each translation, a check is performed to determine whether the membrane is still split across boundaries. If the extent of the membrane in z , H , is approximately equal to box length in z , L_z , then the membrane is split across the periodic boundary. **B** Code snippet for applying the transformation to an MDAnalysis universe. The on-the-fly transformation can be applied to each dimension independently.

2.5 On-the-fly transformations

MDAnalysis has a powerful set of on-the-fly trajectory transformations. These transformations can do away with the need to create multiple instances of the same trajectory using, for example, the GROMACS `trjconv` tool. Instead, the transformations are applied each time a frame is loaded into memory by MDAnalysis. LiPyphilic extends the set of transformations available in MDAnalysis to include the ability to repair membranes split across periodic boundaries and to perform nojump trajectory unwrapping. The latter prevents an atom from being wrapped into the primary unit cell when it crosses a periodic boundary.

2.5.1 Center membranes

The callable class `lipypilic.transformations.center_membrane` can be used to center a membrane — or any supramolecular structure — in a box, providing it is not self-interacting across periodic boundaries. For each frame, all atoms in the system are iteratively shifted along a specified set of dimensions until the membrane is no longer split across the periodic boundaries (Figure 7A). After each translation, all atoms are wrapped back into the primary unit cell. For example, to check if a bilayer is split across the periodic boundary in the z -dimension, its extent in z , H , could be compared to the box length in z , L_z . If H is within a user-specified cutoff value of L_z , the bilayer is split across z and is thus translated in this dimension. Once the bilayer is no longer split across boundaries, it is then moved to the center of the box in z .

This transformation can be applied to an MDAnalysis universe, `u`, using the `u.trajectory.add_transformation` method (Figure 7A). This method takes as input a transformation, or a list of transformations. The `center_membrane` callable class is passed to `add_transformation` along with the arguments for `center_membrane`. The atoms that comprise the membrane are specified using the `ag` argument. The atom selection should include all atoms in the membrane, not a subset of atoms — otherwise the extent of the membrane cannot be calculated accurately. In the example in Figure 7A, the bilayer is centered in only the z -dimension;

however, the membrane can be made whole in each dimension independently. This is controlled using the `center_x`, `center_y` and `center_z` arguments. The `shift` argument is used to specify the distance in Å that the membrane will be translated at each iteration. Too small a value of `shift` would require many iterations to make a membrane whole. Too large a value, on the other hand, may result in a membrane being translated nearly the length of the unit cell and thus remaining broken. We have found a translation of 10 Å to be suitable for bilayers, but the optimal value will depend on the membrane structure and the size of the system.

2.5.2 NoJump trajectory unwrapping

`liplyphilic.transformations.nojump` can be used to prevent atoms from jumping across periodic boundaries. It is analogous, but not equivalent, to using the GROMACS command `trjconv` with the flag `-pbc nojump`. This transformation can be applied to an MDAnalysis universe in much the same way as `liplyphilic.transformations.center_membrane`. We must pass an atom selection to the `ag` argument of `nojump`, and specify the dimensions to which the transformation should be applied (Figure 8).

```
1 from liplyphilic.transformations import nojump
2
3 membrane = u.select_atoms("name GL1 GL2 AM1 AM2 ROH")
4
5 u.trajectory.add_transformations(
6     nojump(
7         ag=membrane,
8         nojump_x=True,
9         nojump_y=True,
10        nojump_z=False
11    )
12 )
```

Figure 8: A "nojump" on-the-fly transformation can be applied to any AtomGroup in an MDAnalysis Universe. The transformation can be applied to each dimension independently.

Upon adding this transformation to your trajectory, `nojump` will perform an initial pass over the trajectory. It will determine the frames at which each atom crosses a boundary, keeping a record of the net movement of each atom at each boundary. This net movement across each boundary is used to determine the distance an atom must be translated in order to be moved from its wrapped position to its unwrapped position. Subsequently, every time a new frame is loaded into memory by MDAnalysis, such as when iterating over the trajectory, the relevant translation is applied to each atom to move it to its unwrapped coordinates.

Below we describe the `nojump` unwrapping algorithm implemented in `LiPyphilic`. We also explain how this algorithm avoids the artefacts introduced by the standard unwrapping scheme that, to our knowledge, is employed by all molecular dynamics simulation related software. Specifically, the standard unwrapping scheme fails to account for fluctuating systems sizes caused by barostats in NPT ensemble simulations.

Unwrapping scheme

The unwrapped position of a particle at frame N , denoted x_N^u , is given by:

$$x_N^u = x_N^w + \sum_{n=0}^N L_n c_n$$

where x_N^w is the wrapped position of the particle at frame N , and $\sum_{n=0}^N L_n c_n$ accounts for the displacement that results from all jumps across periodic boundaries from frame 0 to frame N . L_n is the box length at frame n , with the box centered at $L/2$. This box length is multiplied by a factor c_n , which depends on whether an atom crossed a periodic boundary from frame $n - 1$ to frame n :

$$c_n = \begin{cases} -1 & x_n^w - x_{n-1}^w > L_n/2 \\ 1 & x_n^w - x_{n-1}^w < -L_n/2 \\ 0 & \text{otherwise} \end{cases}$$

Figure 9: To correctly unwrap atomic coordinates we must know size of the box at the frame at which a jump across periodic boundaries occurred. The standard unwrapping scheme produces an incorrect unwrapped coordinate x_2^u .

where x_{n-1}^w is the particle's wrapped position at frame $n - 1$. At frame $n = 0$, for which there is no previous position, we take x_{-1}^w to be the particles raw atomic coordinate at frame $n = 0$. That is, if the particle is not in the primary unit cell at frame $n = 0$, we calculate the displacement required to move from x_0^w to x_0^u .

This unwrapping scheme is equivalent to that recently described by von Bulow *et al.*,⁶⁰ although it was derived independently. Both our unwrapping algorithm and that of von Bulow *et al.* avoid the problems that the standard unwrapping scheme suffers. To calculate x_N^u , the standard unwrapping scheme iteratively adds the box length at frame N to the wrapped coordinated at frame N until $|x_n^w - x_{n-1}^u| < L_n/2$. However, in the example in Figure 9, this would result in $x_2^u = x_2^w - L_2 = -0.5$ instead of the correct value $x_2^u = -1.5$. von Bulow *et al.* demonstrated clearly the effect that this inaccurate unwrapping of atomic

coordinates has on the calculated diffusion coefficient.⁶⁰

The unwrapping scheme described above, and previously by von Bulow *et al.*,² correctly accounts for the fluctuating box size in the NPT ensemble. However, it is only accurate in the case where coordinates are stored every timestep. In fact, it is impossible to correctly unwrap coordinates unless we store them at every timestep. This logically follows from the same argument made above - to correctly unwrap coordinates we must know the length of the box at the timestep at which the jump occurred. See the Supporting Information for further details.

2.6 Other analysis tools

While we have described some of the analysis tools that make LiPyphilic unique in the previous sections, there is much more functionality in LiPyphilic that is fully detailed in the documentation (<https://lipyphilic.readthedocs.io/en/latest>). This functionality includes calculating the coarse-grained lipid order parameter,⁶¹ the area per lipid for planar bilayers, the lateral diffusion coefficient, and the membrane thickness of planar bilayers. There are also tools for calculating thickness, orientations, and z -positions of lipid molecules in a bilayer. In general, LiPyphilic does not handle plotting of analysis data. However, it does have plotting utilities for visualising joint potentials of mean force (PMFs) — such as the PMF of cholesterol orientation and height — and for the projection of membrane properties onto the xy plane. There are full descriptions of all LiPyphilic tools, along with usage examples, in the documentation. Regarding the area per lipid tool, we recommend using either FATSLiM¹ or MemSurfer² if you have a curved membrane. These tools are designed specifically to deal with undulating bilayers and non-bilayer structures, whereas LiPyphilic will only produce reliable values in the case of planar bilayers.

3 Benchmarking

For benchmarking the LiPyphilic analysis tools, we have performed a simulation of an equimolar mixture of DPPC/DOPC/Cholesterol using the MARTINI 2 force field.^{62,63} We created a symmetric bilayer of 12,000 lipids in total using the CHARMM-GUI MARTINI Maker.⁶⁴ The production run was performed for 8.0 μ s and coordinates were stored every 5.0 ns, giving a trajectory of 1,600 frames. Where analogous analysis tools are available in either FATS LiM¹ 0.2.2 or GROMACS⁶⁵ 2020.4, we compare their performance with that of LiPyphilic. We benchmark against FATS LiM as this is generally the fastest membrane analysis tool available.¹ The FATS LiM benchmarks were performed using eight OpenMP threads. All other benchmarks were performed in serial. All of the benchmarks can be seen in Table 1.

LiPyphilic is generally very fast, with most analysis tools and trajectory transformations taking on the order of 10 ms per frame for the 12,000 lipid membrane. For example, both methods of assigning leaflets are faster than the corresponding implementation in FATS LiM. There are two further benefits of assigning leaflets with LiPyphilic: i) molecules may reside in the midplane and ii) the results are stored in a single NumPy array whereas FATS LiM creates a GROMACS index file for each frame of the trajectory. LiPyphilic is significantly faster in calculating the bilayer thickness, although the implementation in FATS LiM is more sophisticated. The FATS LiM thickness tool is able to handle both bilayers and non-bilayer membranes, whereas the one in LiPyphilic is only appropriate for planar bilayers. LiPyphilic is also faster than GROMACS when it comes to calculating the coarse-grained order parameter, S_{CC} . Using LiPyphilic to calculate S_{CC} is also simpler than using GROMACS, which requires the creation of a separate index group for each unique atom along a tail of each lipid species.

The calculation of interleaflet registration, construction of a neighbour matrix, and calculation of the lipid enrichment index are slower, on the order of 100 ms per frame. This is still relatively fast — although they are different analyses, these times are comparable to

Table 1: Benchmark times for analysis tools in LiPyphilic, using a MARTINI bilayer of 12,000 lipids. Where possible, performance is compared with either FATS LiM 0.2.2 or GROMACS 2020.4. The FATS LiM benchmarks were performed using eight OpenMP threads. All other benchmarks were performed in serial.

Analysis/transformation	LiPyphilic class/method	Time per frame (ms)	
		LiPyphilic	Comparison
Leaflet identification	AssignLeaflets	17.54	168.47 [†]
	AssignCurvedLeaflets	29.85	168.47 [†]
Flip-flop [§]	FlipFlop	0.5	-
Interleaflet registration	Registration	151.81	-
Construct neighbour matrix	Neighbours	260.97	-
Largest cluster [¶]	Neighbours.largest_cluster	1.85	-
Enrichment index	Neighbours.count_neighbours	100.93	-
Bilayer thickness	MembThickness	16.35	394.11 [†]
Lipid thickness	ZThickness	37.62	-
Lipid height	ZPositions	19.05	-
Lipid orientation	ZAngles	22.87	-
Area per lipid	AreaPerLipid	1589.29	206.75 [†]
Coarse-grained order parameter [*]	SCC	16.10	119.09 ^{††}
Unwrap membrane	center_membrane	23.37	-
NoJump unwarpping	nojump	23.89	12.04 ^{††}

[†] FATS LiM

^{††} GROMACS

[§] Time per molecule (ms) over 1,600 frames

[¶] Time per frame (ms) for a subset of 2,000 lipids

^{*} Time per frame (ms) for the sn-1 tail of DPPC (4,000 lipids)

the performance of the various analyses available in FATS LiM such as the area per lipid and membrane thickness calculations.

The slowest analysis in LiPyphilic is the calculation of the area per lipid. This takes over 1.5 s per frame, and is 7.6 times slower than FATS LiM. We therefore recommend using FATS LiM for the area per lipid calculation if you have a large membrane (>1,000 lipids). It should be remembered, however, that the FATS LiM analyses were run in parallel over eight cores whereas all other benchmarks were performed in serial. Parallelising the analysis in LiPyphilic could result in a similar performance boost. Whilst this is out of the scope of the package in its current state, we do have plans to parallelise the slower tools in LiPyphilic in due course. In this respect, we are particularly interested in the development of Parallel

MDAnalysis (PMDA).⁶⁶ PMDA is based on MDAnalysis and uses Dask to parallelise the analysis modules.⁶⁷ The analysis classes inherit a modified abstract base class that is specifically designed to make the parallelisation straightforward. We will wait until PMDA is out of the alpha stage of development before assessing which modules would benefit from being parallelised.

Finally, the on-the-fly transformations are fast, with `center_membrane` and `nojump` taking 23.37 ms and 23.89 ms per frame, respectively. Upon applying the `nojump` transformation, a first pass over the trajectory is performed to calculate the translations that need to be applied at each frame. This first pass takes 13.98 ms per frame for the 12,000 atoms selected in the benchmark. After this first pass, the time to load a frame into memory and apply the translations is 9.91 ms. This performance is put into perspective by considering that iterating over the trajectory with MDAnalysis takes 8.54 ms per frame itself, with no transformations applied. The total time per frame of `nojump` (23.89 ms) is approximately twice that required by the GROMACS `trjconv` tool to do the same transformation. However, using `nojump` has two benefits over GROMACS `trjconv` in that it i) prevents the need to create duplicate trajectories and ii) accounts for box size fluctuations caused by barostats.

4 Software Engineering

LiPyphilic is free, open-source software licensed under the GNU General Public License v2 or later. In developing LiPyphilic, we have followed the best practices in modern scientific software engineering.⁶⁸ We use version control, unit testing and continuous integration, and we have a fully documented API with examples of how to use each analysis tool.

The full development history and planned improvements of the project are available to view on GitHub, at <https://github.com/p-j-smith/lipyphilic>. LiPyphilic loosely follows the GitHub-flow model of software development⁶⁹ — developing directly from the master branch and releasing new versions soon after new functionality or fixes are added.

We encourage users to submit feature requests and bug reports via GitHub, and we welcome any question about usage on our discussion page at <https://github.com/p-j-smith/lipyphilic/discussions>.

Unit testing in LiPyphilic is performed using Pytest.⁷⁰ We have constructed a set of toy systems for testing each analysis tool. These systems are typically composed of two sets of atoms each arranged on a hexagonal lattice in xy , with the two lattices separated vertically in z . This setup approximates the topology of the headgroups of a lipid bilayer. Using these toy systems, we know what the results of each analysis tool should be *a priori*. We can thus test each analysis tool with full confidence in the results if the tests pass, without relying on regression tests that involve highly complex systems. Further, using Pytest-cov,⁷¹ we have ensured that all analysis tools and trajectory transformations have 100% test coverage.

Finally, LiPyphilic is simple to install. We have packaged LiPyphilic to make it available for installation using widely-used package managers. The easiest way to install LiPyphilic along with all of its dependencies is through Anaconda.⁷² Alternatively, it can be installed via the Python Package Index using Pip.⁷³

5 Conclusion

We have developed a new Python package for the analysis of lipid membrane simulations. We have focused on providing functionality not available in other membrane analysis tools, such as calculating the lipid enrichment/depletion index, the degree of interleaflet registration, and the flip-flop rate of molecules between leaflets. LiPyphilic is a modular object-oriented package that makes extensive use of NumPy,⁴³ SciPy,⁴⁴ and MDAnalysis^{45,46} for efficient computation. For analysing a 12,000 lipid MARTINI membrane, the analysis classes typically take on the order of 10-100 ms per frame. This is comparable to the performance of analysis tools in GROMACS⁶⁵ and FATSlim¹ — a very fast package for membrane analysis. All analysis tools in LiPyphilic share the same API as those of MDAnalysis. This shared API

makes LiPyphilic simple to learn for current users of MDAnalysis.

The modularity of LiPyphilic, along with its focus on integrating with the wider scientific Python stack, means the output of other analysis tools such as FATS_{LiM}¹ or ML-LPA⁵ can be used as input for further analysis in LiPyphilic. Further, the output of LiPyphilic is in the form of NumPy arrays, Scipy sparse matrices, or Pandas Dataframes. This means the results can readily be plotted or further analysed using the standard libraries of the scientific Python stack.

LiPyphilic is built upon sound software engineering principles. It uses version control, is fully unit-tested, employs continuous integration, and has extensive documentation. LiPyphilic is also trivial to install — it can be installed using either Anaconda or Pip.^{72,73} We encourage users to submit feature requests and bug reports via GitHub, and are always open to new contributors to the project.

Acknowledgement

Via our membership of the UK's HEC Materials Chemistry Consortium, which is funded by EPSRC (EP/L000202/1, EP/R029431/1, EP/T022213), this work used the ARCHER UK National Supercomputing Service (<http://www.archer.ac.uk>) and the UK Materials and Molecular Modelling Hub (MMM Hub) for computational resources, which is partially funded by EPSRC (EP/P020194/1, EP/T022213) to carry out the MD simulations of DP-PC/DOPC/Cholesterol reported in this manuscript. P.S. acknowledges the funding provided by the EPSRC DTP Studentship Block Grant (EP/N509498/1). C.D.L. acknowledges the supportive research environment of the EPSRC center for Doctoral Training in Cross-Disciplinary Approaches to Non-Equilibrium Systems (CANES, No. EP/L015854/1).

Supporting Information Available

Python script for calculating the lipid enrichment index based on tail saturation. Discussion

of problems with nojump trajectory unwrapping.

References

- (1) Buchoux, S. FATSLiM: a fast and robust software to analyze MD simulations of membranes. *Bioinformatics* **2016**, *33*, 133–134.
- (2) Bhatia, H.; Ingólfsson, H. I.; Carpenter, T. S.; Lightstone, F. C.; Bremer, P.-T. MemSurfer: A Tool for Robust Computation and Characterization of Curved Membranes. *Journal of Chemical Theory and Computation* **2019**, *15*, 6411–6421, PMID: 31564100.
- (3) Song, W.; Corey, R. A.; Duncan, A. L.; Ansell, T. B.; Stansfeld, P. J.; Sansom, M. S. Pylipid: A Python Toolkit for Analysis of Lipid-Protein Interactions from MD Simulations. *Biophysical Journal* **2021**, *120*, 48a.
- (4) Sejdiu, B. I.; Tieleman, P. D. ProLint: A web-based framework for the automated data analysis and visualization of lipid-protein interactions. <https://github.com/bisejdiu/prolint>.
- (5) Walter, V.; Ruscher, C.; Benzerara, O.; Thalmann, F. MLLPA: A Machine Learning-assisted Python module to study phase-specific events in lipid membranes. *Journal of Computational Chemistry* **2021**, *42*, 930–943.
- (6) Romo, T. D.; Grossfield, A. LOOS: An extensible platform for the structural analysis of simulations. 2009 Annual International Conference of the IEEE Engineering in Medicine and Biology Society. 2009; pp 2332–2335.
- (7) Romo, T. D.; Leioatts, N.; Grossfield, A. Lightweight object oriented structure analysis: Tools for building tools to analyze molecular dynamics simulations. *Journal of Computational Chemistry* **2014**, *35*, 2305–2318.

- (8) Tieleman, D. P.; Marrink, S.-J. Lipids Out of Equilibrium: Energetics of Desorption and Pore Mediated Flip-Flop. *Journal of the American Chemical Society* **2006**, *128*, 12462–12467.
- (9) Gurtovenko, A. A.; Vattulainen, I. Molecular Mechanism for Lipid Flip-Flops. *The Journal of Physical Chemistry B* **2007**, *111*, 13554–13559.
- (10) Róg, T.; Stimson, L. M.; Pasenkiewicz-Gierula, M.; Vattulainen, I.; Karttunen, M. Replacing the cholesterol hydroxyl group with the ketone group facilitates sterol flip-flop and promotes membrane fluidity. *Journal of Physical Chemistry B* **2008**, *112*, 1946–1952.
- (11) Risselad, H. J.; Marrink, S. J. The molecular face of lipid rafts in model membranes. *Proceedings of the National Academy of Sciences of the United States of America* **2008**, *105*, 17367–17372.
- (12) Kučerka, N.; Perlmutter, J. D.; Pan, J.; Tristram-Nagle, S.; Katsaras, J.; Sachs, J. N. The effect of cholesterol on short- and long-chain monounsaturated lipid bilayers as determined by molecular dynamics simulations and X-ray scattering. *Biophysical Journal* **2008**, *95*, 2792–2805.
- (13) Bennett, W. F.; MacCallum, J. L.; Hinner, M. J.; Marrink, S. J.; Tieleman, D. P. Molecular view of cholesterol flip-flop and chemical potential in different membrane environments. *Journal of the American Chemical Society* **2009**, *131*, 12714–12720.
- (14) Bennett, W. F.; Tieleman, D. P. Molecular simulation of rapid translocation of cholesterol, diacylglycerol, and ceramide in model raft and nonraft membranes. *Journal of Lipid Research* **2012**, *53*, 421–429.
- (15) Ogushi, F.; Ishitsuka, R.; Kobayashi, T.; Sugita, Y. Rapid flip-flop motions of diacylglycerol and ceramide in phospholipid bilayers. *Chemical Physics Letters* **2012**, *522*, 96–102.

- (16) Choubey, A.; Kalia, R. K.; Malmstadt, N.; Nakano, A.; Vashishta, P. Cholesterol translocation in a phospholipid membrane. *Biophysical Journal* **2013**, *104*, 2429–2436.
- (17) Ingólfsson, H. I.; Melo, M. N.; Van Eerden, F. J.; Arnarez, C.; Lopez, C. A.; Wasseenaar, T. A.; Periole, X.; De Vries, A. H.; Tieleman, D. P.; Marrink, S. J. Lipid organization of the plasma membrane. *Journal of the American Chemical Society* **2014**, *136*, 14554–14559.
- (18) Marquardt, D.; Kučerka, N.; Wassall, S. R.; Harroun, T. A.; Katsaras, J. Cholesterol's location in lipid bilayers. *Chemistry and Physics of Lipids* **2016**, *199*, 17–25.
- (19) Javanainen, M.; Martinez-Seara, H. Rapid diffusion of cholesterol along polyunsaturated membranes: Via deep dives. *Physical Chemistry Chemical Physics* **2019**, *21*, 11660–11669.
- (20) Miettinen, M. S.; Lipowsky, R. Bilayer Membranes with Frequent Flip-Flops Have Tensionless Leaflets. *Nano Letters* **2019**, *19*, 5011–5016.
- (21) Oh, Y.; Sung, B. J. Facilitated and Non-Gaussian Diffusion of Cholesterol in Liquid Ordered Phase Bilayers Depends on the Flip-Flop and Spatial Arrangement of Cholesterol. *Journal of Physical Chemistry Letters* **2018**, *9*, 6529–6535.
- (22) Aghaaminiha, M.; Farnoud, A. M.; Sharma, S. Quantitative relationship between cholesterol distribution and ordering of lipids in asymmetric lipid bilayers. *Soft Matter* **2021**, *17*, 2742–2752.
- (23) Filipe, H. A. L.; Javanainen, M.; Salvador, A.; Galvã, A. M.; Vattulainen, I.; Luís, L. M. S.; João Moreno, M. Quantitative Assessment of Methods Used To Obtain Rate Constants from Molecular Dynamics Simulations - Translocation of Cholesterol across Lipid Bilayers. *J. Chem. Theory Comput* **2018**, *14*, 20.

- (24) Gu, R.-X.; Baoukina, S.; Tieleman, D. P. Cholesterol Flip-Flop in Heterogeneous Membranes. *Journal of Chemical Theory and Computation* **2019**, *15*, 2064–2070.
- (25) Baral, S.; Levental, I.; Lyman, E. Composition dependence of cholesterol flip-flop rates in physiological mixtures. *Chemistry and Physics of Lipids* **2020**, *232*, 104967.
- (26) Carter, J. W.; Gonzalez, M. A.; Brooks, N. J.; Seddon, J. M.; Bresme, F. Flip-flop asymmetry of cholesterol in model membranes induced by thermal gradients. *Soft Matter* **2020**, *16*, 5925–5932.
- (27) Sodt, A. J.; Sandar, M. L.; Gawrisch, K.; Pastor, R. W.; Lyman, E. The molecular structure of the liquid-ordered phase of lipid bilayers. *Journal of the American Chemical Society* **2014**, *136*, 725–732.
- (28) Ackerman, D. G.; Feigenson, G. W. Multiscale modeling of four-component lipid mixtures: Domain composition, size, alignment, and properties of the phase interface. *Journal of Physical Chemistry B* **2015**, *119*, 4240–4250.
- (29) Lin, X.; Lorent, J. H.; Skinkle, A. D.; Levental, K. R.; Waxham, M. N.; Gorfe, A. A.; Levental, I. Domain stability in biomimetic membranes driven by lipid polyunsaturation. *Journal of Physical Chemistry B* **2016**, *120*, 11930–11941.
- (30) Sodt, A. J.; Venable, R. M.; Lyman, E.; Pastor, R. W. Nonadditive Compositional Curvature Energetics of Lipid Bilayers. *Physical Review Letters* **2016**, *117*, 138104.
- (31) Ingólfsson, H. I.; Carpenter, T. S.; Bhatia, H.; Bremer, P. T.; Marrink, S. J.; Lightstone, F. C. Computational Lipidomics of the Neuronal Plasma Membrane. *Biophysical Journal* **2017**, *113*, 2271–2280.
- (32) Koshiyama, K.; Taneo, M.; Shigematsu, T.; Wada, S. Bicelle-to-Vesicle Transition of a Binary Phospholipid Mixture Guided by Controlled Local Lipid Compositions: A

- Molecular Dynamics Simulation Study. *Journal of Physical Chemistry B* **2019**, *123*, 3118–3123.
- (33) Smith, P.; Quinn, P. J.; Lorenz, C. D. Two coexisting membrane structures are defined by lateral and transbilayer interactions between sphingomyelin and cholesterol. *Langmuir* **2020**, *36*, 9786–9799.
- (34) Gu, R.-X.; Baoukina, S.; Tieleman, D. P. Phase Separation in Atomistic Simulations of Model Membranes. *Journal of the American Chemical Society* **2020**, *142*, 2844–2856.
- (35) Nickels, J. D.; Smith, J. C.; Cheng, X. Lateral organization, bilayer asymmetry, and inter-leaflet coupling of biological membranes. *Chemistry and Physics of Lipids* **2015**, *192*, 87–99.
- (36) Fowler, P. W.; Williamson, J. J.; Sansom, M. S.; Olmsted, P. D. Roles of Interleaflet Coupling and Hydrophobic Mismatch in Lipid Membrane Phase-Separation Kinetics. *Journal of the American Chemical Society* **2016**, *138*, 11633–11642.
- (37) Reigada, R. Alteration of interleaflet coupling due to compounds displaying rapid translocation in lipid membranes. *Scientific Reports* **2016**, *6*, 1–15.
- (38) Fujimoto, T.; Parmryd, I. Interleaflet coupling, pinning, and leaflet asymmetry-major players in plasma membrane nanodomain formation. *Frontiers in Cell and Developmental Biology* **2017**, *4*, 155.
- (39) Galimzyanov, T. R.; Kuzmin, P. I.; Pohl, P.; Akimov, S. A. Undulations Drive Domain Registration from the Two Membrane Leaflets. *Biophysical Journal* **2017**, *112*, 339–345.
- (40) Thallmair, S.; Ingoïfsson, H. I.; Marrink, S. J. Cholesterol Flip-Flop Impacts Domain Registration in Plasma Membrane Models. *J. Phys. Chem. Lett* **2018**, *9*, 5527–5533.

- (41) Weiner, M. D.; Feigenson, G. W. Molecular Dynamics Simulations Reveal Leaflet Coupling in Compositionally Asymmetric Phase-Separated Lipid Membranes. *The Journal of Physical Chemistry B* **2019**, *123*, 3968–3975.
- (42) Sarmiento, M. J.; Hof, M.; Šachl, R. Interleaflet Coupling of Lipid Nanodomains – Insights From in vitro Systems. *Frontiers in Cell and Developmental Biology* **2020**, *8*, 284.
- (43) Harris, C. R.; Millman, K. J.; van der Walt, S. J.; Gommers, R.; Virtanen, P.; Cournapeau, D.; Wieser, E.; Taylor, J.; Berg, S.; Smith, N. J.; Kern, R.; Picus, M.; Hoyer, S.; van Kerkwijk, M. H.; Brett, M.; Haldane, A.; del Río, J. F.; Wiebe, M.; Peterson, P.; Gérard-Marchant, P.; Sheppard, K.; Reddy, T.; Weckesser, W.; Abbasi, H.; Gohlke, C.; Oliphant, T. E. Array programming with NumPy. *Nature* **2020**, *585*, 357–362.
- (44) Virtanen, P.; Gommers, R.; Oliphant, T. E.; Haberland, M.; Reddy, T.; Cournapeau, D.; Burovski, E.; Peterson, P.; Weckesser, W.; Bright, J.; van der Walt, S. J.; Brett, M.; Wilson, J.; Millman, K. J.; Mayorov, N.; Nelson, A. R.; Jones, E.; Kern, R.; Larson, E.; Carey, C. J.; Polat, İ.; Feng, Y.; Moore, E. W.; VanderPlas, J.; Laxalde, D.; Perktold, J.; Cimrman, R.; Henriksen, I.; Quintero, E. A.; Harris, C. R.; Archibald, A. M.; Ribeiro, A. H.; Pedregosa, F.; van Mulbregt, P.; Vijaykumar, A.; Bardelli, A. P.; Rothberg, A.; Hilboll, A.; Kloeckner, A.; Scopatz, A.; Lee, A.; Rokem, A.; Woods, C. N.; Fulton, C.; Masson, C.; Häggström, C.; Fitzgerald, C.; Nicholson, D. A.; Hagen, D. R.; Pasechnik, D. V.; Olivetti, E.; Martin, E.; Wieser, E.; Silva, F.; Lenders, F.; Wilhelm, F.; Young, G.; Price, G. A.; Ingold, G. L.; Allen, G. E.; Lee, G. R.; Audren, H.; Probst, I.; Dietrich, J. P.; Silterra, J.; Webber, J. T.; Slavič, J.; Nothman, J.; Buchner, J.; Kulick, J.; Schönberger, J. L.; de Miranda Cardoso, J. V.; Reimer, J.; Harrington, J.; Rodríguez, J. L. C.; Nunez-Iglesias, J.; Kuczynski, J.; Tritz, K.; Thoma, M.; Newville, M.; Kümmerer, M.; Bolingbroke, M.; Tartre, M.; Pak, M.; Smith, N. J.; Nowaczyk, N.; Shebanov, N.; Pavlyk, O.; Brodtkorb, P. A.;

- Lee, P.; McGibbon, R. T.; Feldbauer, R.; Lewis, S.; Tygier, S.; Sievert, S.; Vigna, S.; Peterson, S.; More, S.; Pudlik, T.; Oshima, T.; Pingel, T. J.; Robitaille, T. P.; Spura, T.; Jones, T. R.; Cera, T.; Leslie, T.; Zito, T.; Krauss, T.; Upadhyay, U.; Halchenko, Y. O.; Vázquez-Baeza, Y. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods* **2020**, *17*, 261–272.
- (45) Michaud-Agrawal, N.; Denning, E. J.; Woolf, T. B.; Beckstein, O. MDAAnalysis: A toolkit for the analysis of molecular dynamics simulations. *Journal of Computational Chemistry* **2011**, *32*, 2319–2327.
- (46) Gowers, R.; Linke, M.; Barnoud, J.; Reddy, T.; Melo, M.; Seyler, S.; Domański, J.; Dotson, D.; Buchoux, S.; Kenney, I.; Beckstein, O. MDAAnalysis: A Python Package for the Rapid Analysis of Molecular Dynamics Simulations. Proceedings of the 15th Python in Science Conference. 2016; pp 98–105.
- (47) Weiner, M. D.; Feigenson, G. W. Presence and Role of Midplane Cholesterol in Lipid Bilayers Containing Registered or Antiregistered Phase Domains. *J. Phys. Chem. B* **2018**, *122*, 21.
- (48) Doktorova, M.; Symons, J. L.; Levental, I. Structural and functional consequences of reversible lipid asymmetry in living membranes. *Nature Chemical Biology* **2020**, *16*, 1321–1330.
- (49) Bhatia, H.; Bremer, P.-T. Example usage of Memsurfer - a bilayer membrane consisting of three types of lipid. https://github.com/LLNL/MemSurfer/blob/develop/examples/ex_3lipid.py.
- (50) Heberle, F. A.; Doktorova, M.; Scott, H. L.; Skinkle, A. D.; Waxham, M. N.; Levental, I. Direct label-free imaging of nanodomains in biomimetic and biological membranes by cryogenic electron microscopy. *Proceedings of the National Academy of Sciences of the United States of America* **2020**, *117*, 19943–19952.

- (51) Koldsø, H.; Shorthouse, D.; Hélie, J.; Sansom, M. S. P. Lipid Clustering Correlates with Membrane Curvature as Revealed by Molecular Simulations of Complex Lipid Bilayers. *PLoS Computational Biology* **2014**, *10*, e1003911.
- (52) Lorent, J. H.; Levental, K. R.; Ganesan, L.; Rivera-Longworth, G.; Sezgin, E.; Doktorova, M.; Lyman, E.; Levental, I. Plasma membranes are asymmetric in lipid unsaturation, packing and protein shape. *Nature Chemical Biology* **2020**, *16*, 644–652.
- (53) Ingólfsson, H. I.; Bhatia, H.; Zeppelin, T.; Bennett, W. F.; Carpenter, K. A.; Hsu, P. C.; Dharuman, G.; Bremer, P. T.; Schjøtt, B.; Lightstone, F. C.; Carpenter, T. S. Capturing Biologically Complex Tissue-Specific Membranes at Different Levels of Compositional Complexity. *Journal of Physical Chemistry B* **2020**, *124*, 7819–7829.
- (54) Wilson, K. A.; MacDermott-Opeskin, H. I.; Riley, E.; Lin, Y.; O’Mara, M. L. Understanding the Link between Lipid Diversity and the Biophysical Properties of the Neuronal Plasma Membrane. *Biochemistry* **2020**, *59*, 3010–3018.
- (55) Wilson, K. A.; Fairweather, S. J.; Macdermott-Opeskin, H. I.; Wang, L.; Morris, R. A.; O’mara, M. L. The role of plasmalogens, Forssman lipids, and sphingolipid hydroxylation in modulating the biophysical properties of the epithelial plasma membrane. *Journal of Chemical Physics* **2021**, *154*, 095101.
- (56) Sze, M. Y.; Gillams, R. J.; McLain, S. E.; Lorenz, C. D. Effects of lipid heterogeneity on model human brain lipid membranes. *Soft Matter* **2021**, *17*, 126–135.
- (57) Sonnino, S.; Mauri, L.; Chigorno, V.; Prinetti, A. Gangliosides as components of lipid membrane domains. *Glycobiology* **2007**, *17*, 1R–13R.
- (58) Prinetti, A.; Loberto, N.; Chigorno, V.; Sonnino, S. Glycosphingolipid behaviour in complex membranes. *Biochimica et Biophysica Acta - Biomembranes* **2009**, *1788*, 184–193.

- (59) Westerlund, B.; Slotte, J. P. How the molecular features of glycosphingolipids affect domain formation in fluid membranes. *Biochimica et Biophysica Acta - Biomembranes* **2009**, *1788*, 194–201.
- (60) Von Bülow, S.; Bullerjahn, J. T.; Hummer, G. Systematic errors in diffusion coefficients from long-time molecular dynamics simulations at constant pressure. *Journal of Chemical Physics* **2020**, *153*, 021101.
- (61) Seo, S.; Murata, M.; Shinoda, W. Pivotal Role of Interdigitation in Interleaflet Interactions: Implications from Molecular Dynamics Simulations. *Journal of Physical Chemistry Letters* **2020**, *11*, 5171–5176.
- (62) Marrink, S. J.; Risselada, H. J.; Yefimov, S.; Tieleman, D. P.; De Vries, A. H. The MARTINI force field: Coarse grained model for biomolecular simulations. *Journal of Physical Chemistry B* **2007**, *111*, 7812–7824.
- (63) Melo, M. N.; Ingólfsson, H. I.; Marrink, S. J. Parameters for Martini sterols and hopanoids based on a virtual-site description. *Journal of Chemical Physics* **2015**, *143*, 243152.
- (64) Qi, Y.; Ingólfsson, H. I.; Cheng, X.; Lee, J.; Marrink, S. J.; Im, W. CHARMM-GUI Martini Maker for Coarse-Grained Simulations with the Martini Force Field. *Journal of Chemical Theory and Computation* **2015**, *11*, 4486–4494.
- (65) Abraham, M. J.; Murtola, T.; Schulz, R.; Páll, S.; Smith, J. C.; Hess, B.; Lindah, E. Gromacs: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX* **2015**, *1-2*, 19–25.
- (66) Fan, S.; Linke, M.; Paraskevakos, I.; Gowers, R.; Gecht, M.; Beckstein, O. PMDA - Parallel Molecular Dynamics Analysis. Proceedings of the 18th Python in Science Conference. 2019; pp 134–142.

- (67) Rocklin, M. Dask: Parallel computation with blocked algorithms and task scheduling. Proceedings of the 14th python in science conference. 2015.
- (68) Jarrod Millman, K.; Pérez, F. *Implementing Reproducible Research*; CRC Press, 2014; pp 149–183.
- (69) Flow, G. The best way to use Git and GitHub. <https://githubflow.github.io/>.
- (70) Krekel, H.; Oliveira, B.; Pfannschmidt, R.; Bruynooghe, F.; Laughner, B.; Bruhin, F. pytest 6.2.2. 2004; <https://github.com/pytest-dev/pytest>.
- (71) pytest-cov development team, pytest-cov 2.11.1. 2006; <https://github.com/pytest-dev/pytest-cov>.
- (72) Anaconda, Anaconda Software Distribution. <https://anaconda.com>.
- (73) PyPA, pip - The Python Package Installer. <https://pip.pypa.io/en/stable/>.

Graphical TOC Entry