Pytim: A Python Package for the Interfacial Analysis of Molecular Simulations

Marcello Sega, *[a] György Hantal,[a] Balázs Fábián,[b,c] and Pál Jedlovszky[d]

Pytim is a versatile python framework for the analysis of interfacial properties in molecular simulations. The code implements several algorithms for the identification of instantaneous interfaces of arbitrary shape, and analysis tools written specifically for the study of interfacial properties, such as intrinsic profiles. The code is written in the python language, and makes use of the numpy and scipy packages to deliver high computational performances. Pytim relies on the MDAnalysis library to analyze the trajectory file formats of popular simulation packages such as GROMACS, CHARMM, NAMD, LAMMPS or Amber, and can be used to steer OpenMM simulations. Pytim can write information about surfaces and surface atomic layers to VTK, CUBE, and PDB files for easy visualization. The classes of Pytim can be easily customized and extended to include new interfacial algorithms or analysis tools. The code is available as open source and is free of charge.

DOI:10.1002/jcc.25384

Introduction

The interface between two fluids is not a well-defined concept as soon as one leaves the continuum description and reaches the molecular detail level.[1] The determination of atoms belonging to an interface is intrinsically dependent on the scale at which the observer probes the interface itself. In addition, thermally activated capillary waves, by corrugating the interfaces, are adding further complexity to the determination of the physicochemical properties of fluids in these boundary regions.

Historically, several algorithms were devoted to the calculation of the accessible surface (to the solvent or other small molecules) of macromolecules,[2] and several approaches were developed to compute what is now known as the solvent accsssible surface area (SASA)[3,4] or the connolly surface.[5,6] More recently, the field has been stirred up by the necessity of investigating the structure of fluid/fluid interfaces from an intrinsic point of view, namely, by removing the smearing effect of thermal capillary waves,[7] and several methods focused on liquid/liquid interfaces were developed, such as Willard and Chandler instantaneous liquid interfaces method,[8] the circular variance method of Mezei,[9] the method of Škvor and colleagues[10] or the method of Yesylevskyy and Ramseyer for the calculation of the local curvature.[11]

The Pytim package includes a series of algorithms to identify phases and surfaces or surface atoms, and perform various types of analyses related to interfaces. The main algorithms implemented in Pytim are: (1) the identification of truly interfacial molecules[12] (ITIM) for macroscopically planar systems; (2) the generalized ITIM[13] (GITIM), for arbitrarily shaped systems; (3) the algorithm of Willard and Chandler,[8] to compute continuous surfaces based on a smoothed estimate of the atomic volumetric density; (4) the Lee-Richards algorithm[3] to compute the SASA and the associated solvent-exposed molecules; (5) an efficient cutoff-based clustering algorithm to distinguish liquid and vapor phases,[14] and (6) an improved, density based clustering algorithm for highly miscible systems.[15]

All algorithms that identify surface atoms can be used not only to compute the interfacial layer, but also the successive layers beneath it. The algorithms and the peculiarities related to their implementation in Pytim are briefly reviewed in the next section.

The underlying philosophy of Pytim is to provide a flexible, extendable, and easily scriptable system for the calculation of interfaces and interfacial properties in molecular systems that free the user from the burden of writing ad hoc code for different simulation packages. Pytim is built on top, and extends, the MDAnalysis library,[16] which provides the backend for reading different trajectory formats. All algorithms are implemented by

[a] *M. Sega, György Hantal*
Faculty of Physics, University of Vienna, Boltzmangasse 5, Vienna A-1090, Austria
E-mail: marcello.sega@univie.ac.at
[b] *Balázs Fábián*
Institut UTINAM (CNRS UMR 6213), Université Bourgogne Franche-Comté, 16 Route de Gray, Besançon Cedex F-25030, France
[c] *Balázs Fábián*
Department of Inorganic and Analytical Chemistry, Budapest University of Technology and Economics, Szt. Gellért tér 4, Budapest H1111, Hungary
[d] *Pál Jedlovszky*
Department of Chemistry, Eszterházy Károly University, Leányka utca 6, H-3300, Eger, Hungary
Contract Grant sponsor: H2020 Marie Skłodowska-Curie Actions; Contract Grant number: 708175; Contract Grant sponsor: UNKP-17-3-I New National Excellence Program of the Ministry of Human CapacitiesContract Grant sponsor: European Union's Horizon 2020 research and innovation programmeContract Grant sponsor: Ministry of Human CapacitiesContract Grant sponsor: Hungarian NKFH Foundation; Contract Grant number: 119732

making use of the efficient numerical libraries wrapped by **numpy** and **scipy** that take advantage of the SIMD architecture of modern processors. Multiprocess parallelization has been implemented, when possible and Cython has been used to speed up inner loops. The resulting code has a speed comparable to that of a compiled one. The user interface has been designed to be intuitive and straightforward, in the spirit of the python language, and allows to perform the fundamental analyses using just a couple of scripting lines. This task is also facilitated by providing meaningful default values for all options. However, the user has complete control over the details of the algorithms and can quickly perform complex tasks by combining the basic building blocks of Pytim. In particular, the structure of the interface is meant to encourage the user to experiment with the possibilities offered by combining Pytim with other libraries and create their own analysis tools. MDAnalysis has been chosen for its high level of abstraction, which allowed to reduce development time and simplicity in implementing other backends. In particular, the pytim code can be used in a transparent way with MDTraj,[17] and compute interfacial properties online during OpenMM[18] simulation runs.

Methods

Algorithms to compute interfaces and interfacial atoms

ITIM. This algorithm[12] can be loosely described as a molecular version of the popular pinscreen toy. It consists in determining the atoms in contact with a virtual surface made of probe spheres bound to move perpendicularly to the surface plane along test lines. The present implementation follows that of Jorge et al.,[19] which is based on sorting atoms according to their distance from the center of mass of the slab. Then, starting from the furthermost atom, one finds which test lines allow a sphere of radius α, which moves along the line, to touch the atom with a radius $\sigma/2$. Here, σ corresponds typically to the Lennard-Jones parameter.[20] The sorting of N atoms can be performed in $\mathcal{O}(N\log N)$ steps, while finding which of the N_{tl} test lines are within a given radius requires $\mathcal{O}(\log N_{\mathrm{tl}})$ steps using a kd-tree.[21] As the test lines are usually 10 times more than the surface atoms (for the algorithm to work accurately), one has typically $N_{\mathrm{tl}} \simeq 10N^{2/3}$, and in the worst case $N_{\mathrm{tl}} \simeq 10N$. Therefore, the algorithm scales globally like $\mathcal{O}(N\log N)$. By assuming convergence at a large number of test lines, this method is left with one free parameter, the probe sphere radius α, which sets the scale at which the interface is probed.

Figure 1 reports an example of how to use the ITIM algorithm in Pytim.

GITIM. In this generalization of the ITIM algorithm, no macroscopic orientation is assumed. The touching spheres, instead of "raining down" along probe lines, can be thought of as being "inflated" at all points in space, up to their maximum radius α (in which case the surrounding atoms are considered to be interfacial ones) or until they start touching neighboring atoms. This procedure is realized in GITIM using a modification of the alpha-shapes algorithm[22] that takes into account the excluded volume of the atoms,[13] and implements in its essence a filter of the Delaunay[23] triangulation of atomic positions. In other words, all the triangular faces belonging to those tetrahedra, whose inner touching sphere has a radius larger than α, are considered to be interfacial ones, and so are considered the atoms at the vertices of those triangular faces. The present implementation of GITIM can calculate the Delaunay triangulation using the **scipy** library, which uses, in turn, a wrapper to the Quickhull[24] implementation **Qhull**. In addition, it can use the faster **pytetgen** python wrapper to the **tetgen** software.[25] Both algorithms scale on average as $\mathcal{O}(N\log N)$ and in the worst case as $\mathcal{O}(N^2)$, setting, therefore, the scaling of GITIM. Also in this case the only free parameter is the probe sphere radius α.

Figure 2 reports an example of how to use the GITIM algorithm in Pytim.

```
import MDAnalysis as mda
import pytim
from pytim.datafiles import WATER_GRO
# load the configuration in MDAnalysis
u = mda.Universe(WATER_GRO)
# compute the interface using ITIM. Identify 4 layers.
inter = pytim.ITIM(u,max_layers=4)
# save a pdb file (including layers information)
# for visualization
inter.writepdb('system.pdb')
```

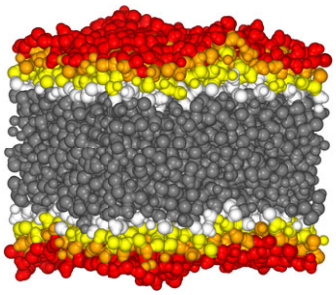

Figure 1. Usage of the ITIM class to identify the first four molecular layers in a water/vapor interface. The image has been generated in a jupyter notebook using **nglview**. [Color figure can be viewed at wileyonlinelibrary.com]

Willard and Chandler's instantaneous liquid interface. This algorithm[8] defines a continuous interface as the isodensity surface of a Gaussian kernel density estimate.[26,27] A continuous Gaussian density function, with given height h_G and width w_G, is associated with each atom. Summing over the contributions of all atoms, one obtains a continuous density field, which is usually sampled on a regular grid. The isosurface where the density is in the neighborhood of a given, target density (typically half between the maximum and the minimum) is then used as the definition of the interface itself. In the present implementation, we use both an exact method, by deriving a class from **scipy.stat.gaussian_kde** that implements periodic boundary conditions, and an approximated one that calculates the Gaussian contribution from points closer than $2.5w_G$. We make use of the topologically consistent version[28] of the marching cubes algorithm[29] as implemented in **scikit-image**[30] to extract the isosurface from the kernel density estimate. The computation of the kernel density estimate scales like $\mathcal{O}(NN_g)$. Here, N_g is either the

```
import MDAnalysis as mda
import pytim
from   pytim.datafiles import GLUCOSE_PDB
u       = mda.Universe(GLUCOSE_PDB)
solvent = u.select_atoms('name␣OW')
glc     = u.atoms - solvent.residues.atoms
# alpha is the probe-sphere radius
inter = pytim.GITIM(u, group=solvent, max_layers=3, alpha=2)
for i in [0,1,2]:
    print "Layer␣"+str(i),repr(inter.layers[i])

Layer 0 <AtomGroup with 54 atoms>
Layer 1 <AtomGroup with 117 atoms>
Layer 2 <AtomGroup with 216 atoms>
```

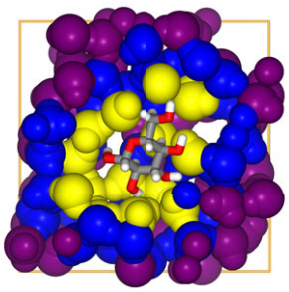

Figure 2. Usage of the **GITIM** class to identify three solvation shells of a glucose molecule. The image is a section cut of the system, as visualized using **nglview**. [Color figure can be viewed at wileyonlinelibrary.com]

number of grid points in the simulation box (in the exact implementation) or the number of grid points within the sphere of radius $2.5w_G$ (in the approximated implementation). The marching cube algorithm scales like $\mathcal{O}(N_g)$, so that $\mathcal{O}(NN_g)$ is the global scaling. In the most common scenario, the grid spacing is kept constant, independently on the system size, meaning that the computational complexity of the exact implementation scales like $\mathcal{O}(N^2)$, while the approximated one scales like $\mathcal{O}(N)$.

The algorithm has in principle three free parameters, namely w_G, h_G, and the target density. However, by choosing the target density as half of the maximum one, h_G becomes irrelevant. The algorithm does not directly determine surface atoms, but one could define them as those within a certain distance from the isodensity surface. The Willard–Chandler algorithm is not restricted to planar surfaces.

Lee-Richards SASA algorithm. The algorithm first creates, for each of the atoms, a list of neighbors within a cutoff radius, which is twice the sum of the largest atom radius and the probe sphere radius. The spherical volume associated to each atom that has as radius the sum of the atomic and of the probe sphere radii is then sliced in several discs along a chosen direction. Using simple geometrical considerations, one can calculate the total arc length of each disc, which happens to overlap with the corresponding discs associated to neighboring atoms. In this way, it is possible to compute an approximation of the area that is (or is not) accessible by a probe sphere. Atoms with nonzero accessible surface area are considered to belong to the interfacial layer. As the neighbor search algorithm scales like $\mathcal{O}(N\log N)$, and the calculation of the exposed area scales like $\mathcal{O}(1)$ for each atom, the global scaling is $\mathcal{O}(N\log N)$.

Algorithms to determine phases

To determine the interface between two phases containing the same chemical compound, it is, of course, necessary to assign which atoms belong to that phase. This task can be in principle trivial for liquid/vapor interfaces far from the critical temperature, as typical simulated system sizes are small, and often the vapor phase is just empty. For liquid/liquid interfaces, the equivalent condition is that of a perfectly demixing system. In these cases, it is in principle not necessary to perform any particular prefiltering of the particles before proceeding to the identification of surfaces or surface molecules using any of the above algorithms. One has to be careful, however, that the presence of a single molecule either in the vapor phase or solvated in the opposite phase, can jeopardize the whole interfacial determination. This problem is, of course, even more serious when getting closer to the critical temperature.

One of the most straightforward ways to separate molecules in the vapor phase from those in the liquid is to group all atoms in clusters and consider the biggest cluster as the liquid phase.

Simple clustering. A simple yet effective strategy is to consider molecules to be in the same cluster if any of their atoms are closer than a chosen cutoff distance.[14] Usually, a reasonable choice for the cutoff value is the position of the minimum after the first peak of the pair distribution function in the bulk liquid. This way, one is sure to include in the cluster all atoms belonging to the first solvation shell. In this case, the liquid fraction will form a percolating network of connected molecules, whereas the molecules in the vapor phase will be part of smaller, separated clusters.

```
import MDAnalysis as mda
import pytim
from pytim.datafiles import MICELLE_PDB

u = mda.Universe(MICELLE_PDB)
g = u.select_atoms('resname␣DPC')
interface = pytim.WillardChandler(u, group=g,
                    mesh=1.5, alpha=3.0)
interface.writecube('data.cube')
```

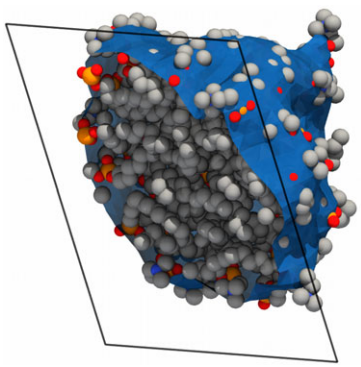

Figure 3. Usage of the **WillardChandler** class to identify the isodensity continuous surface of a solvated dodecylphosphocholine (DPC) micelle. The image (water not shown) is a section cut of the micelle along with the continuous surface, as visualized using paraview. [Color figure can be viewed at wileyonlinelibrary.com]

DBSCAN. The simple cutoff clustering scheme is limited to the case of low temperatures and low miscibilities. At high enough temperatures, for example, the density of the vapor phase can become high that the whole system percolates using any choice of cutoff. In this case, it is more effective to look at the local density rather than at the connectivity. The DBSCAN density-based cluster algorithm[31] is well-suited for this task, and has been already used to study supercritical fluids[32] and a two phase system of highly miscible fluids.[15] The main idea of the algorithm is to include in a cluster all atoms with overlapping neighborhoods where the density is higher than a given threshold. In Pytim, we implemented an approach for the automatic determination of the density threshold, based on a k-mean analysis[33,34] of the density distribution.[15]

Algorithms to compute interfacial properties

Once the surface atoms are determined, these are available through the python interface for further analysis. In the Pytim package, we provide the user with an extendable set of classes for the computation of observables, suited both for the calculation of bulk properties, and interfacial ones. In particular, one can combine each observable with the calculation of intrinsic and non-intrinsic density profiles, as well as pair distribution and time correlation functions of atoms within the interfacial layers. The (non-intrinsic) profile of observable O across a macroscopically planar interface is computed in the simulation box-fixed reference frame as

$$\rho_O(z) = \left\langle \frac{1}{S}\sum_i O_i \delta(z - z_i) \right\rangle,$$

where $\langle \ldots \rangle$ represents an ensemble average, $\delta(z)$ is the Dirac delta function, and the sum is extended over the set of atoms of interest. This corresponds to the ordinary notion of a density profile, where the bin width is given by $S\Delta z$. Here the discretized version of the Dirac delta function is $\delta(z) \simeq 1/\Delta z$ if $|z| < \Delta z/2$.

The intrinsic profile, on the contrary, is obtained by referring the position of the particles to the surface capillary waves, obtaining in this way the local, interfacial structure of the fluid without the smearing effect of the capillary waves themselves. If a particle is located at $\mathbf{r}_i = (x_i, y_i, z_i)$ in the box reference frame, we denote the position of the local reference frame on the corrugated surface as $(x_i, y_i, \xi(x_i, y_i))$. In this way, the intrinsic profile can be written as[7,35]

$$\rho^I_O(z) = \left\langle \frac{1}{S}\sum_i O_i \delta(z - z_i + \xi(x_i, y_i)) \right\rangle.$$

Notice that z can be either negative or positive, depending on the location with respect to the interface, and can be, therefore, thought of as a signed distance.

This concept can be generalized to nonplanar macroscopic interfaces by choosing the distance of an atom i located at $\mathbf{r}_i$ from the surface as its distance to the closest surface atom, j. However, care has to be taken in determining the bin width and its sign. Although there is, in the general case, no analytical expression for the area of the iso-distance surfaces, we estimate the bin volume numerically, by using a simple Monte Carlo scheme.[13] The second point one has to take care of is the determination of the sign of the distance that determines whether a point is located below or above the interface. While in the planar case this is straightforward, in the general case, we resort to the following. We compute the center of mass of the local environment around the closest surface atom, $\mathbf{r}_{cm}$, and determine the sign of the distance as the sign of the scalar product $(\mathbf{r}_i - \mathbf{r}_j) \cdot (\mathbf{r}_{cm} - \mathbf{r}_j)$. This choice is made because if both $\mathbf{r}_i$ and $\mathbf{r}_{cm}$ are on the same side with respect to the atom representing the location $\mathbf{r}_j$ of the local surface, one could consider the i-th atom to be within the liquid phase.

Results

Computational performances

We have analyzed the computational complexity of the three main algorithms implemented in Pytim, and of the cluster search. As a test system, we have taken a macroscopically planar water/vapor interface. Starting from an initial configuration of 216 water molecules, we generated larger systems by replicating the simulation box in the two directions of the surface plane, obtaining systems up to 55.296 water molecules. In this way, we increased by the same proportion both the system size and the surface area determined by the different algorithms.

In Figures 4–6, we report the time needed to perform the surface analysis with **`itim`**, **`gitim`**, and the Willard–Chandler algorithm. In Figure 4, we report also the time needed to assign molecules to the liquid cluster. All algorithms are roughly following the linear scaling for large system sizes, as expected from the theoretical analysis of the average cases. For small system sizes, sublinear scaling is sometimes observed. The exact calculation of the kernel density estimate in the Willard–Chandler algorithm scales also, as expected, worse than its fast, approximated version, although slightly better than quadratic.

The probe sphere radius was the same (2 Å) in all tests with **`itim`** and **`gitim`**; for the instantaneous interface calculation

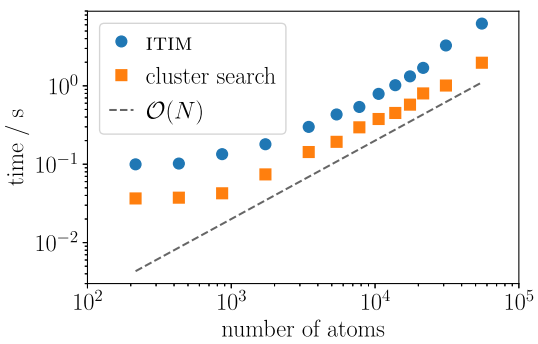

Figure 4. Computational complexity of the ITIM algorithm as implemented in **`Pytim`**. The time needed to identify the surface atoms in one configuration is separated into the time to perform the cluster analysis and the actual time for the identification. [Color figure can be viewed at wileyonlinelibrary.com]

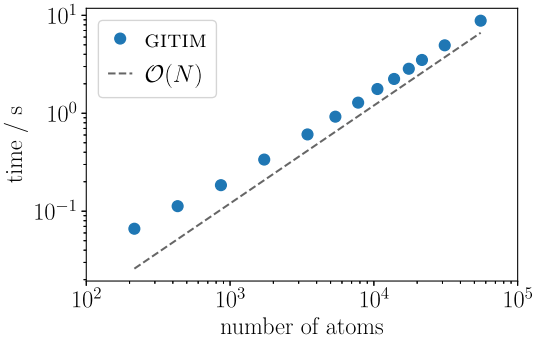

Figure 5. Computational complexity of the GITIM algorithm as implemented in **Pytim**. The calculation of the intrinsic surface atoms has been performed on the same systems as the in Figure 4: GITIM is only about 2–3 times slower than ITIM. [Color figure can be viewed at wileyonlinelibrary.com]

using the Willard–Chandler algorithm, we used a gaussian width w_G = 3 Å and a grid spacing of 2 Å.

Regarding the prefactors, it is worth noticing that, while **itim** turned out to be the fastest one, **gitim** performed comparatively well, being only a factor 2–3 slower than **itim**. The performances of the Willard–Chandler algorithm are affected by the density field discretization (i.e., linear scaling in the number of grid points, or, equivalently, cubic scaling in the inverse grid spacing), but for the reasonable choice of 2 Å for the grid spacing, the performances were only slightly worse than those of **gitim**.

All tests were performed on a laptop with an Intel(R) Core(TM) i7-4650 U CPU at 1.70 GHz.

Installation instructions

The latest development code can be retrieved from the address https://github.com/Marcello-Sega/pytim as a git repository, or as a zipped archive. From the source code, the command **python setup.py install --user** will install the package in the user's directories. Requirements for the installation are the **setuptools** and **numpy** python package and **Cython**. The **setup.py** script will automatically download and install other required dependencies.

On the other hand, the user can install the latest stable version from the Python package index (https://pypi.python.org) by invoking **pip install --user pytim**, or from Anaconda (https://anaconda.org/conda-forge/) with **conda install -c conda-forge pytim**.

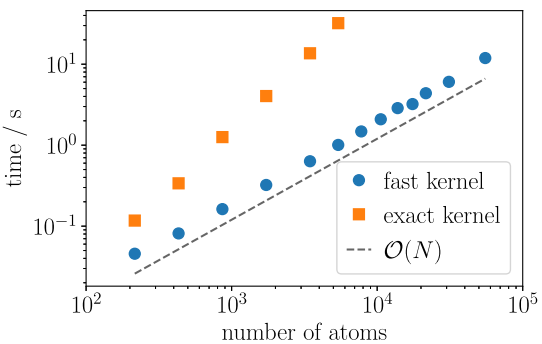

Figure 6. Computational complexity of the Willard–Chandler algorithm, with the variant of the **scipy.stats.gaussian_kde** (squares) and with the approximated fast kernel (circles). In both cases, the grid spacing was set to 2 Å. [Color figure can be viewed at wileyonlinelibrary.com]

Usage examples

The simplest script that can be used to compute the interfacial atoms is the following:

```
import MDAnalysis as mda
import pytim
from pytim.datafiles import WATER_GRO
u = mda.Universe(WATER_GRO)
inter = pytim.ITIM(u)
```

The script above starts with importing the MDAnalysis and Pytim modules, and one of the example files present in the distribution. Next, the configuration is loaded, and the interfacial atoms are computed using ITIM with default values. The interfacial atoms are accessible in a number of ways, one of which being the array **inter.atoms** that lists all of them, irrespective of the layer/side in which they are located.

In addition to the usual properties that are available for the **Atom** and **AtomGroup** classes of MDAnalysis, Pytim introduces new ones, namely, **layers**, **sides**, and **clusters**. These properties can be accessed from each atom in the system, for example, as **u.atoms.layers**, and give access to which layer the atoms have been assigned to (**layers**), to which side of a planar interface the atoms are (**sides**, ITIM only) and whether the atoms are in the main cluster or in one of the smaller ones (**clusters**). These properties can be used to quickly select the atoms of interest. For example, one can access the atoms in the first layer as **u.atoms[u.atoms.layers==1]**.

Some usage examples involving the different methods have been already presented in Figures 1–3. Here, we describe two typical scripts that can be used to compute intrinsic and non-intrinsic profiles.

Figure 7 shows an example of how one can compute an intrinsic density profile in Pytim. Note that the Pytim distribution provides several sample configurations and trajectories (in this case, the configuration of the Lennard-Jones system is pointed at by the label **LJ_GRO**). Some larger trajectories, which would be too big for the distribution, can be retrieved from the online repository using the **pytim_data.fetch()** function, as, in this case, for the **LJ_BIG_XTC** binary trajectory. This script initializes first the interface calculation using ITIM, then instantiates a **Profile** object. The **interface** option signals that the profile has to be computed relative to the interface itself, that is, it will be an intrinsic profile. Because no observable is passed to the **Profile** constructor, this defaults to the number density profile calculation. In the successive lines, the code iterates over the trajectory frames, computing the surface atoms automatically every time a new frame is loaded, and sampling the intrinsic profile. The profile is stored internally with 0.01 Å resolution, and can be accessed through the **get_values()** function with the desired binning, resulting in the plot shown in Figure 7, with the liquid part located on the left (negative position values) of the interface (a delta function centered at zero, not shown) and the vapor part on its

```
import numpy as np
import MDAnalysis as mda
import pytim
from   pytim.datafiles import LJ_GRO
from   pytim.observables import Profile
XTC = pytim.datafiles.pytim_data.fetch('LJ_BIG_XTC')
u = mda.Universe(LJ_GRO,XTC)

# passing the cluster_cut option switches on
# the cluster search.
inter = pytim.ITIM(u,alpha=2.5,cluster_cut=4.5)
profile = Profile(interface=inter)

for ts in u.trajectory:
    profile.sample(g)

low, up, avg = profile.get_values(binwidth=0.5)
np.savetxt('profile.dat',list(zip((low+up)/2.,avg)))
```

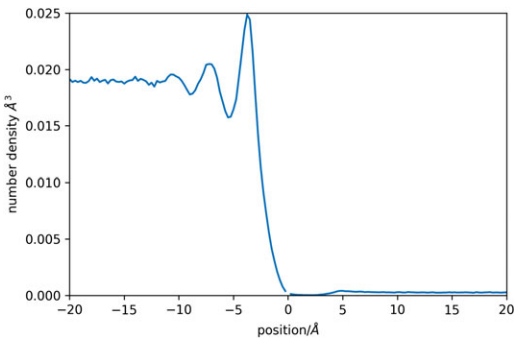

Figure 7. Calculation of the intrinic density profile of a Lennard-Jones liquid/vapor interface. [Color figure can be viewed at wileyonlinelibrary.com]

right (positive values). The distinction between liquid and vapor is performed by prefiltering the configurations used by ITIM using the simple cluster search, which is switched on by using the **`cluster_cut`** option. The DBSCAN cluster search is, instead, switched on by using, additionally, the **`cluster_threshold`** option.

Another example, on how to compute the distribution of the surface layers along the interface normal is shown in Figure 8. In this case, one instantiates five different profiles (without specifying an **`interface`** option, the code will compute the non-intrinsic profile), and passes to the **`sample()`** function a selection of atoms corresponding to those in the different layers.

As a final example, in Figure 9 we show the analysis of the intrinsic curvature at the surface of the DPC micelle, together with the calculation of the local normal vector, using the method of Yesylevskyy and Ramseyer.[11]

Other examples are provided on the github page of the project, in the online manual, and in the example files in the Pytim distribution, including details on how to compute time correlation functions or how to use the pytim tools on top of MDTraj or in an OpenMM simulation run.

List of functionalities

The code, thanks to the use of the MDAnalysis, allows to perform several surface-related analyses of many popular trajectories and configuration file formats. The full list is available on the website of MDAnalysis (http://www.mdanalysis.org). Both Python 2.7 and Python 3 are supported. The functionalities of the code can be grouped in the following classes:

```
import numpy as np
import MDAnalysis as mda
import pytim
u = mda.Universe('water.gro','water.xtc')
g = u.select_atoms('name␣OW')
# here we calculate the profiles of oxygens only
# (note molecular=False)
inter = pytim.ITIM(u,group=g,max_layers=4, molecular=False)

# We create a list of 5 profiles, one for the density
# of the whole system and 4 for the first layers.
Layers = []
for L in range(5):
    Layers.append(Profile())
# Go through the trajectory, center the liquid
# slab and sample the profiles
for ts in u.trajectory:
    inter.center()
    Layers[0].sample(g)
    for L in range(1,5):
        selection = np.where(u.atoms.layers == L)
        Layers[L].sample(u.atoms[selection])
```

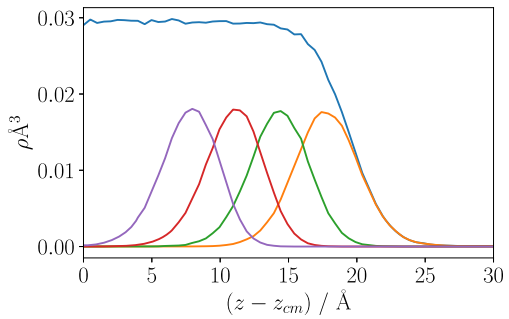

Figure 8. Calculation of the distribution of the first four molecular layers in a water/vapor interface. [Color figure can be viewed at wileyonlinelibrary.com]

Interface/surface identification.

1. ITIM algorithm for macroscopically planar interfaces
2. GITIM algorithm for arbitrary interfaces
3. Willard–Chandler algorithm for arbitrary interfaces
4. Lee–Richards SASA algorithm for arbitrary interfaces

Clustering/filtering.

1. simple clustering of nearest neighbors
2. DBSCAN density based clustering with automatic threshold tuning

Output/Export. In addition to all the file formats available through MDAnalysis, Pytim can save information about the interfaces in some additional ones, namely:

1. PDB format for particles with information on layering
2. VTK format for particles
3. VTK format for volumetric densities
4. VTK format for surface triangulation
5. CUBE format for particles and volumetric density
6. Wavefront OBJ format for surface triangulation

Observables. Pytim provides also an abstract class for coding new observables. A set of basic atomic/molecular properties (density, charge, molecular orientation, ...) are already provided in the code, and we plan to expand the set of available observables with time. All observables derived from the abstract class

```
import numpy as np
import MDAnalysis as mda
import pytim
from pytim.datafiles import MICELLE_PDB
from pytim.observables import Curvature
from pytim.observables import LocalReferenceFrame
u = mda.Universe(MICELLE_PDB)
g = u.select_atoms('resname␣DPC')
inter = pytim.GITIM(u,group=g,molecular=False)
Gauss = Curvature().compute(inter.atoms)[:,0]
frame = LocalReferenceFrame().compute(inter.atoms)

normals  = frame[:,2]
positive = inter.atoms[np.where(Gauss>=0)[0]]
negative = inter.atoms[np.where(Gauss<0)[0]]
```

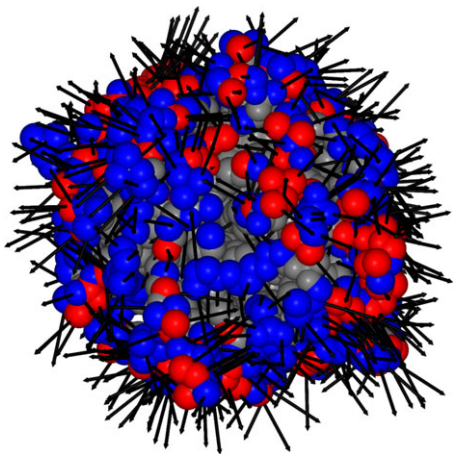

Figure 9. Calculation of the local intrinsic curvature for surface atoms (red: positive; blue: negative) and local surface normal for the DPC micelle. [Color figure can be viewed at wileyonlinelibrary.com]

are suitable to be used in the calculation of the following correlations/distributions:

1. Intrinsic and non-intrinsic profiles
2. Three- and two-dimensional radial distribution functions
3. Time correlation functions, also for variable sets of atoms (both in the continuous and intermittent variants)

Test systems. Several configurations and small trajectory files are provided for testing purposes: the present set of systems includes so far as follows:

1. Water/vapor interfaces at different temperatures
2. Water/CCl_4 interface
3. Methanol/vapor interface
4. Ionic liquid/benzene mixture
5. A glucose molecule in water
6. DPC micelle
7. DPPC bilayer
8. Fullerene

API documentation and tests

`Pytim` code has been written using docstrings for all public classes/functions, and for the most important private ones. The code reference manual, generated automatically from the code, is available at https://marcello-sega.github.io/pytim/.

Along with the reference manual come several tutorials, which cover several topics ranging from the installation to troubleshooting, and include common usage of the main modules. In the source distribution, there is a directory that includes several python example files that show how to use the main modules and the analysis tools. In addition, jupyter notebooks are also present, and can be visualized (statically) directly from a web browser, without needing to run the code, from https://github.com/Marcello-Sega/pytim.

Code snippets within the documentation of the code are used to provide a testing framework using the **`doctest`** module in combination with **`Sphinx`** (http://www.sphinx-doc.org/). Any new update of the code, to be incorporated in the main branch, has to pass all tests (more than 200, with a code coverage larger than 85%) using different versions of Python, of the underlying libraries (MDAnalysis) on Linux and Mac OS-X systems. Automated testing is performed using Travis-CI (https://travis-ci.org/Marcello-Sega/pytim).

Conclusions

We presented Pytim, a new package for the interfacial analysis of molecular systems. Pytim, is based on the python scripting language and provides the user with a number of analysis tools dedicated to the properties of fluid interfaces. Being built on top of the MDAnalysis library, Pytim allows to analyze trajectory files in the native formats of the most common simulation packages, freeing the user from the burden of conversion. The package implements three main surface analysis algorithms, namely, ITIM, GITIM, and the Willard–Chandler instantaneous liquid interface algorithm, as well as two different clustering algorithms to be used for identifying different phases. All algorithms have been carefully optimized using either fast, vectorized functions from the **`numpy`** and **`scipy`** libraries, or by writing, when necessary, Cython code, as well as parallel, multiprocess code. All algorithm have a linear or quasi-linear scaling with respect to the system size. The use of python as the main coding language for this package also allows the user to easily modify and extend the package with minimum effort.

Acknowledgments

P. J. acknowledges financial support of the Hungarian NKFH Foundation under project no. 119732. B.F. is supported by the UNKP-17-3-I New National Excellence Program of the Ministry of Human Capacities. This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement no. 708175.

Keywords: interfacial analysis · molecular simulations · python

How to cite this article: M. Sega, G. Hantal, B. Fábián, P. Jedlovszky. *J. Comput. Chem.* **2018**, *39*, 2118–2125. DOI: 10.1002/jcc.25384

[1] D. L. Heath, J. K. Percus, J. Stat. Phys. **1987**, 49, 319.
[2] J. Tomasi, B. Mennucci, R. Cammi, Chem. Rev. **2005**, 105, 2999.
[3] B. Lee, F. M. Richards, J. Mol. Biol. **1971**, 55, 379.

[4] F. M. Richards, Annu. Rev. Biophys. Bioeng. **1977**, 6, 151.
[5] M. L. Connolly, Science **1983**, 221, 709.
[6] M. L. Connolly, J. Appl. Crystallogr. **1983**, 16, 548.
[7] E. Chacón, P. Tarazona, Phys. Rev. Lett. **2003**, 91, 166103.
[8] A. P. Willard, D. Chandler, J. Phys. Chem. B **2010**, 114, 1954.
[9] M. Mezei, J. Mol. Graph. Modell. **2003**, 21, 463.
[10] J. Škvor, J. Škvára, J. Jirsák, I. Nezbeda, J. Mol. Graph. Modell. **2017**, 76, 17.
[11] S. Yesylevskyy, C. Ramseyer, Phys. Chem. Chem. Phys. **2014**, 16, 17052.
[12] L. B. Pártay, G. Hantal, P. Jedlovszky, Á. Vincze, G. Horvai, J. Comput. Chem. **2008**, 29, 945.
[13] M. Sega, S. S. Kantorovich, P. Jedlovszky, M. Jorge, J. Chem. Phys. **2013**, 138, 44110.
[14] L. B. Partay, G. Horvai, P. Jedlovszky, J. Phys. Chem. C **2010**, 114, 21681.
[15] M. Sega, G. Hantal, Phys. Chem. Chem. Phys. **2017**, 19, 18968.
[16] N. Michaud-Agrawal, E. J. Denning, T. B. Woolf, O. Beckstein, J. Comput. Chem. **2011**, 32, 2319.
[17] R. T. McGibbon, K. A. Beauchamp, M. P. Harrigan, C. Klein, J. M. Swails, C. X. Hernández, C. R. Schwantes, L.-P. Wang, T. J. Lane, V. S. Pande, Biophys. J. **2015**, 109, 1528.
[18] P. Eastman, M. S. Friedrichs, J. D. Chodera, R. J. Radmer, C. M. Bruns, J. P. Ku, K. A. Beauchamp, T. J. Lane, L.-P. Wang, D. Shukla, T. Tye, M. Houston, T. Stich, C. Klein, M. R. Shirts, V. S. Pande, J. Chem. Theory Comput. **2012**, 9, 461.
[19] M. Jorge, P. Jedlovszky, M. N. D. S. Cordeiro, J. Phys. Chem. B **2010**, 114, 11169.
[20] M. Sega, Phys. Chem. Chem. Phys. **2016**, 18, 23354.
[21] J. L. Bentley, Commun. ACM **1975**, 18, 509.
[22] H. Edelsbrunner, E. P. Mücke, ACM Trans. Graph. **1994**, 13, 43.
[23] B. N. Delaunay, I. Akad, Nauk SSSR Otdelenie Matematicheskii i Estestvennyka Nauk **1934**, 7, 793.
[24] C. B. Barber, D. P. Dobkin, H. Huhdanpaa, ACM Trans. Math. Softw. **1996**, 22, 469.
[25] H. Si, A. C. M. Trans, Math. Softw. **2015**, 41, 1.
[26] M. Rosenblatt, Ann. Math. Stat. **1956**, 27, 832.
[27] E. Parzen, Ann. Math. Stat. **1962**, 33, 1065.
[28] T. Lewiner, H. Lopes, A. W. Vieira, G. Tavares, J. Graph. Tools **2003**, 8, 1.
[29] W. E. Lorensen, H. E. Cline, ACM Siggraph Computer Graphics (ACM), Vol. 21, Association for Computing Machinery (ACM), New York, **1987**, p. 163.
[30] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, Scikit-image contributors, PeerJ **2014**, 2, e453.
[31] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, KDD-96 Proceedings, Vol. 96, (Association for the Advancement of Artificial Intelligence (AAAI), Palo Alto, California, **1996**, p. 226.
[32] A. Idrissi, I. Vyalov, N. Georgi, M. Kiselev, J. Phys. Chem. B **2013**, 117, 12184.
[33] J. MacQueen, Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Vol. 1, University of California Press, Oakland, **1967**, p. 281.
[34] S. Lloyd, IEEE Trans. Inf. Theory **1982**, 28, 129.
[35] M. Sega, B. Fábián, P. Jedlovszky, J. Chem. Phys. **2015**, 143, 114709.

Received: 13 March 2018
Revised: 31 May 2018
Accepted: 2 June 2018
Published online on 10 October 2018