MDAnalysis: A Python Package for the Rapid Analysis of Molecular Dynamics Simulations

Richard J. Gowers[†][††][‡‡][†], Max Linke[**][†], Jonathan Barnoud[¶][†], Tyler J. E. Reddy[§], Manuel N. Melo[¶], Sean L. Seyler[‡], Jan Domański[§], David L. Dotson[‡], Sébastien Buchoux[||], Ian M. Kenney[‡], Oliver Beckstein[‡][*]

https://youtu.be/zVQGFysYDew

Abstract—MDAnalysis (http://mdanalysis.org) is a library for structural and temporal analysis of molecular dynamics (MD) simulation trajectories and individual protein structures. MD simulations of biological molecules have become an important tool to elucidate the relationship between molecular structure and physiological function. Simulations are performed with highly optimized software packages on HPC resources but most codes generate output trajectories in their own formats so that the development of new trajectory analysis algorithms is confined to specific user communities and widespread adoption and further development is delayed. MDAnalysis addresses this problem by abstracting access to the raw simulation data and presenting a uniform object-oriented Python interface to the user. It thus enables users to rapidly write code that is portable and immediately usable in virtually all biomolecular simulation communities. The user interface and modular design work equally well in complex scripted work flows, as foundations for other packages, and for interactive and rapid prototyping work in IPython / Jupyter notebooks, especially together with molecular visualization provided by nglview and time series analysis with pandas. MDAnalysis is written in Python and Cython and uses NumPy arrays for easy interoperability with the wider scientific Python ecosystem. It is widely used and forms the foundation for more specialized biomolecular simulation tools. MDAnalysis is available under the GNU General Public License v2.

Index Terms—molecular dynamics simulations, science, chemistry, physics, biology

Introduction

Molecular dynamics (MD) simulations of biological molecules have become an important tool to elucidate the relationship between molecular structure and physiological function [DDG+12], [Oro14]. Simulations are performed with highly optimized software packages on HPC resources but most codes generate output trajectories in their own formats so that the development of new trajectory analysis algorithms is confined to specific user communities and widespread adoption and further development is delayed. Typical trajectory sizes range from gigabytes to terabytes so it is typically not feasible to convert trajectories into a range of different formats just to use a tool that requires this specific format. Instead, a framework is required that provides a common interface to raw simulation data. Here we describe the MDAnalysis library [MADWB11] that addresses this problem by abstracting access to the raw simulation data. MDAnalysis presents a uniform object-oriented Python interface to the user. Since its original publication in 2011 [MADWB11], MDAnalysis has been widely adopted and has undergone substantial changes. Here we provide a short introduction to MDAnalysis and its capabilities and an overview over recent improvements.

MDAnalysis was initially inspired by MDTools for Python (J.C. Phillips, unpublished) and MMTK [Hin00]. MDTools pioneered the key idea to use an extensible and object-oriented language, namely, Python, to provide a high-level interface for the construction and analysis of molecular systems for MD simulations. MMTK became a tool kit to build MD simulation applications on the basis of a concise object model of a molecular system. MDAnalysis was built on an object model similar to that of MMTK with a strong focus on providing universal high-level building blocks for the analysis of MD trajectories, but for a much wider range of formats than previously available. MDAnalysis has been publicly available since January 2008 and is one of the longest actively maintained Python packages for the analysis of molecular simulations. Since then many other packages have appeared that primarily function as libraries for providing access to simulation data from within Python. Three popular examples are PyLOOS [RLG14], mdtraj [MBH+15], and pytraj [NRSC16]. PyLOOS [RLG14] consists of Python bindings to the C++ LOOS library [RG09]; in order to aid novice users, LOOS also provides about 140 small stand-alone tools that each focus on a single task. mdtraj [MBH+15] is similar to MDAnalysis in many aspects but focuses even more on being a light-weight building block for other packages; it also includes a number of innovative performance optimizations. pytraj [NRSC16] is a versatile Python frontend to the popular and powerful `cpptraj` tool [RCI13] and is particularly geared towards users of the Amber MD package [CCD+05]. These three packages and MDAnalysis have in common that they are built on an object model of the underlying data (such as groups of particles or a trajectory), use compiled code in C, C++ or Cython to accelerate time critical bottlenecks, and have a "Pythonic" user interface. LOOS and MDAnalysis share a similar object-oriented

† These authors contributed equally.
†† University of Manchester, Manchester, UK
‡‡ University of Edinburgh, Edinburgh, UK
∗∗ Max Planck Institut für Biophysik, Frankfurt, Germany
¶ University of Groningen, Groningen, The Netherlands
§ University of Oxford, Oxford, UK
‡ Arizona State University, Tempe, Arizona, USA
|| Université de Picardie Jules Verne, Amiens, France
∗ Corresponding author: oliver.beckstein@asu.edu

philosophy in their user interface design. In contrast, mdtraj and pytraj expose a functional user interface. Both approaches have advantages and the existence of different "second generation" Python packages for the analysis of MD simulations provides many good choices for users and a fast moving and stimulating environment for developers.

Overview

MDAnalysis is specifically tailored to the domain of molecular simulations, in particularly in biophysics, chemistry, and biotechnology as well as materials science. The user interface provides physics-based abstractions (e.g., atoms, bonds, molecules) of the data that can be easily manipulated by the user. It hides the complexity of accessing data and frees the user from having to implement the details of different trajectory and topology file formats (which by themselves are often only poorly documented and just adhere to certain community expectations that can be difficult to understand for outsiders). MDAnalysis currently supports more than 25 different file formats and covers the vast majority of data formats that are used in the biomolecular simulation community, including the formats required and produced by the most popular packages such as NAMD [PBW+05], Amber [CCD+05], Gromacs [AMS+15], CHARMM [BBIM+09], LAMMPS [Pli95], DL_POLY [TSTD06], HOOMD [GNA+15] as well as the Protein Data Bank PDB format [BWF+00] and various other specialized formats.

Since the original publication [MADWB11], improvements in speed and data structures make it now possible to work with terabyte-sized trajectories containing up to ~10 million particles. MDAnalysis also comes with specialized analysis classes in the `MDAnalysis.analysis` module that are unique to MDAnalysis such as *LeafletFinder* (in the `leaflet` module), a graph-based algorithm for the analysis of lipid bilayers [MADWB11], or *Path Similarity Analysis* (`psa`) for the quantitative comparison of macromolecular conformational changes [SKTB15].

Code base

MDAnalysis is written in Python and Cython with about 42k lines of code and 24k lines of comments and documentation. It uses NumPy arrays [VCV11] for easy interoperability with the wider scientific Python ecosystem. Although the primary dependency is NumPy, other Python packages such as netcdf4 and BioPython [HM03] also provide specialized functionality to the core of the library (Figure 1).

Availability

MDAnalysis is available in source form under the GNU General Public License v2 from GitHub as MDAnalysis/mdanalysis, and as PyPi and conda packages. The documentation is extensive and includes an introductory tutorial.

Development process

The development community is very active with more than five active core developers and many community contributions in every release. We use modern software development practices [WAB+14], [SM14] with continuous integration (provided by Travis CI) and an extensive automated test suite (containing over 3500 tests with >92% coverage for our core modules). Development occurs on GitHub through pull requests that are reviewed by core developers and other contributors, supported by

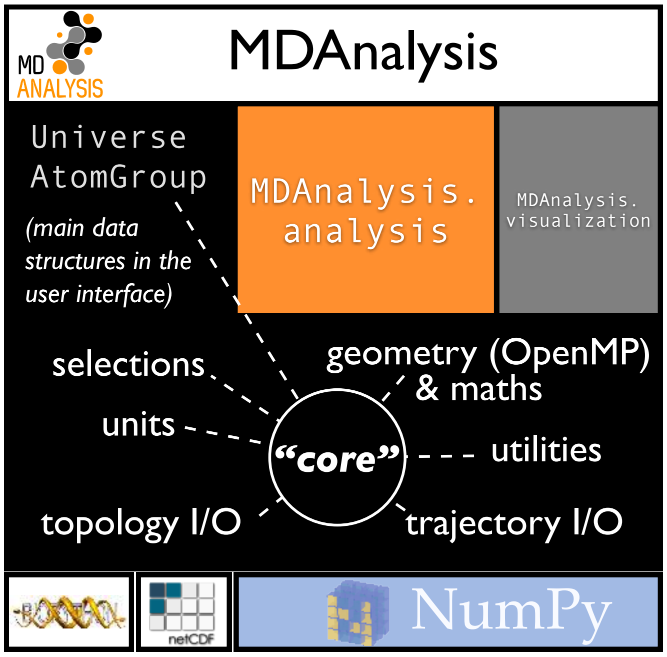


Fig. 1: Structure of the MDAnalysis package. MDAnalysis consists of the core *with the* Universe *class as the primary entry point for users. The* `MDAnalysis.analysis` *package contains independent modules that make use of the core to implement a wide range of algorithms to analyze MD simulations. The* `MDAnalysis.visualization` *package contains a growing number of tools that are specifically geared towards calculating visual representations such as, for instance, streamlines of molecules.*

the results from the automated tests, test coverage reports provided by Coveralls, and QuantifiedCode code quality reports. Users and developers communicate extensively on the community mailing list (*Google* groups) and the GitHub issue tracker; new users and developers are very welcome and most user contributions are eventually integrated into the code base. The development and release process is transparent to users through open discussions and announcements and a full published commit history and changes. Releases are numbered according to the semantic versioning convention so that users can immediately judge the impact of a new release on their existing code base, even without having to consult the `CHANGELOG` documentation. Old code is slowly deprecated so that users have ample opportunity to update the code although we generally attempt to break as little code as possible. When backwards-incompatible changes are inevitable, we provide tools (based on the Python standard library's *lib2to3*) to automatically refactor code or warn users of possible problems with their existing code.

Basic usage

The core object in MDAnalysis is the Universe which acts as a nexus for accessing all data contained within a simulation. It is initialized by passing the file names of the topology and trajectory files, with a multitude of different formats supported in these roles. The topology acts as a description of all the particles in the system while the trajectory describes their behavior over time.

```python
import MDAnalysis as mda

# Create a Universe based on simulation results
u = mda.Universe('topol.tpr', 'traj.trr')
```

```
# Create a selection of atoms to work with
ag = u.atoms.select_atoms('backbone')
```

The select_atoms method allows for AtomGroups to be created using a human readable syntax which allows queries according to properties, logical statements and geometric criteria.

```
# Select all solvent within a set distance from protein atoms
ag = u.select_atoms('resname SOL and around 5.0 protein')

# Select all heavy atoms in the first 20 residues
ag = u.select_atoms('resid 1:20 and not prop mass < 10.0')

# Use a preexisting AtomGroup as part of another selection
sel1 = u.select_atoms('name N and not resname MET')
sel2 = u.select_atoms('around 2.5 group Nsel', Nsel=sel1)

# Perform a selection on another AtomGroup
sel1 = u.select_atoms('around 5.0 protein')
sel2 = sel1.select_atoms('type O')
```

The AtomGroup acts as a representation of a group of particles, with the properties of these particles made available as NumPy arrays.

```
ag.names
ag.charges
ag.positions
ag.velocities
ag.forces
```

The data from MD simulations comes in the form of a trajectory which is a frame by frame description of the motion of particles in the simulation. Today trajectory data can often reach sizes of hundreds of GB. Reading all these data into memory is slow and impractical. To allow the analysis of such large simulations on an average workstation (or even laptop) MDAnalysis will only load a single frame of a trajectory into memory at any time.

The trajectory data can be accessed through the trajectory attribute of a Universe. Changing the frame of the trajectory object updates the underlying arrays that AtomGroups point to. In this way the positions attribute of an AtomGroup within the iteration over a trajectory will give access to the positions at each frame. Through this approach only a single frame of data is present in memory at any time, allowing for large data sets, from half a million particles to tens of millions (see also section Analysis of large systems), to be dissected with minimal resources.

```
# the trajectory is an iterable object
len(u.trajectory)

# seek to a given frame
u.trajectory[72]
# iterate through every 10th frame
for ts in u.trajectory[::10]:
    ag.positions
```

In some cases it is necessary to access frames of trajectories in a random access pattern or at least be able to rapidly access a starting frame anywhere in the trajectory. Examples for such usage are the calculation of time correlation functions, skipping of frames (as in the iterator `u.trajectory[5000::1000]`), or parallelization over trajectory blocks in a map/reduce pattern [TRB+08]. If the underlying trajectory reader only implements linear sequential reading from the beginning, searching for specific frames becomes extremely inefficient, effectively prohibiting random access to time frames on disk. Many trajectory formats suffer from this shortcoming, including the popular Gromacs XTC and TRR formats, but also commonly used multi-frame PDB files and other text-based formats such as XYZ. LOOS [RG09] implemented a mechanism by which the trajectory was read once on loading and frame offsets on disk were computed that could be used to directly seek to individual frames. Based on this idea, MDAnalysis implements a fast frame scanning algorithm for TRR and XTC files and also saves the offsets to disk (as a compressed NumPy array). When a trajectory is loaded again then instead of reading the whole trajectory, only the persistent offsets are read (provided they have not become stale as checked by conservative criteria such as changes in file name, modification time, and size of the original file, which are all saved with the offsets). In cases of terabyte-sized trajectories, the persistent offset approach can save hundreds of seconds for the initial loading of the `Universe` (after an initial one-time cost of scanning the trajectory). Current development work is extending the persistent offset scheme to all trajectory readers, which will provide random access for all trajectories in a completely automatic and transparent manner to the user.

Example: Per-residue RMSF

As a complete example consider the calculation of the C_α root mean square fluctuation (RMSF) ρ_i that characterizes the mobility of a residue i in a protein:

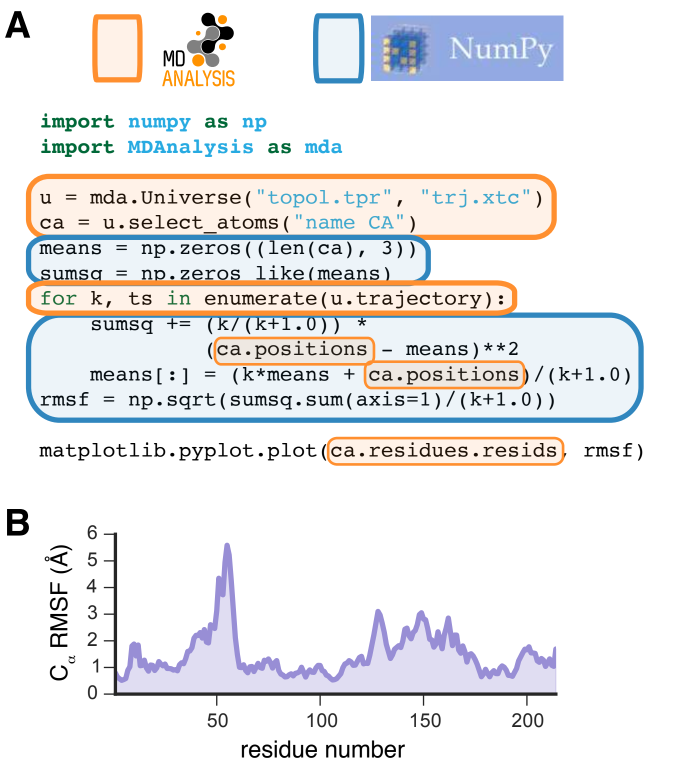
$$\rho_i = \sqrt{\left\langle \left(\mathbf{x}_i(t) - \langle \mathbf{x}_i \rangle\right)^2 \right\rangle} \tag{1}$$

The code in Figure 2 A shows how MDAnalysis in combination with NumPy can be used to implement Eq. 1. The topology information and the trajectory are loaded into a `Universe` instance; C_α atoms are selected with the MDAnalysis selection syntax and stored as the `AtomGroup` instance `ca`. The main loop iterates through the trajectory using the MDAnalysis trajectory iterator. The coordinates of all selected atoms become available in a NumPy array `ca.positions` that updates for each new time step in the trajectory. Fast operations on this array are then used to calculate variance over the whole trajectory. The final result is plotted with matplotlib [Hun07] as the RMSF over the residue numbers, which are conveniently provided as an attribute of the `AtomGroup` (Figure 2 B).

The example demonstrates how the abstractions that MDAnalysis provides enable users to write concise code where the computations on data are cleanly separated from the task of extracting the data from the simulation trajectories. These characteristics make it easy to rapidly prototype new algorithms. In our experience, most new analysis algorithms are developed by first prototyping a simple script (like the one in Figure 2), often inside a Jupyter notebook (see section Interactive Use and Visualization). Then the code is cleaned up, tested and packaged into a module. In section Analysis Module, we describe the analysis code that is included as modules with MDAnalysis.

Interactive use and visualization

The high level of abstraction and the pythonic API, together with comprehensive Python doc strings, make MDAnalysis well suited for interactive and rapid prototyping work in IPython [PG07] and Jupyter notebooks. It works equally well as an interactive analysis tool, especially with Jupyter notebooks, which then contain an executable and well-documented analysis protocol that can be easily shared and even accessed remotely. Universes and AtomGroups can be visualized in Jupyter notebooks using nglview, which interacts natively with the MDAnalysis API (Figure 3).

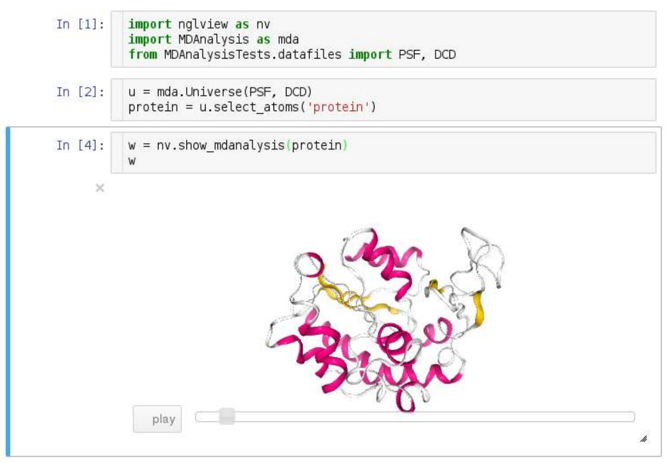


```python
import numpy as np
import MDAnalysis as mda

u = mda.Universe("topol.tpr", "trj.xtc")
ca = u.select_atoms("name CA")
means = np.zeros((len(ca), 3))
sumsq = np.zeros_like(means)
for k, ts in enumerate(u.trajectory):
    sumsq += (k/(k+1.0)) *
            (ca.positions - means)**2
    means[:] = (k*means + ca.positions)/(k+1.0)
rmsf = np.sqrt(sumsq.sum(axis=1)/(k+1.0))

matplotlib.pyplot.plot(ca.residues.resids, rmsf)
```

*Fig. 2: Example for how to calculate the root mean square fluctuation (RMSF) for each residue in a protein with MDAnalysis and NumPy. **A**: Based on the input simulation data (topology and trajectory in the Gromacs format (TPR and XTC), MDAnalysis makes coordinates of the selected* C_α *atoms available as NumPy arrays. From these coordinates, the RMSF is calculated by averaging over all frames in the trajectory. The RMSF is then plotted with matplotlib. The algorithm to calculate the variance in a single pass is due to Welford [Wel62]. **B**:* C_α *RMSF for each residue.*



```
In [1]: import nglview as nv
        import MDAnalysis as mda
        from MDAnalysisTests.datafiles import PSF, DCD

In [2]: u = mda.Universe(PSF, DCD)
        protein = u.select_atoms('protein')

In [4]: w = nv.show_mdanalysis(protein)
        w
```

Fig. 3: MDAnalysis can be used with nglview to directly visualize molecules and trajectories in Jupyter notebooks. The adenylate kinase (AdK) protein from one of the included test trajectories is shown. .

Other Python packages that have become extremely useful in notebook-based analysis work flows are pandas [McK10] for rapid analysis of time series analysis, distributed [Roc15] for simple parallelization, FireWorks [JOC+15] for complex work flows, and MDSynthesis [DGS+16] for organizing, bundling and querying many simulations.

Analysis module

In the `MDAnalysis.analysis` module we provide a large variety of standard analysis algorithms, like RMSD (root mean square distance) and RMSF (root mean square fluctuation) calculations, RMSD-optimized structural superposition [LAT10], native contacts [BHE13], [FKDD07], or analysis of hydrogen bonds as well as unique algorithms, such as the *LeafletFinder* in `MDAnalysis.analysis.leaflet` [MADWB11] and *Path Similarity Analysis* (`MDAnalysis.analysis.psa`) [SKTB15]. Historically these algorithms were contributed by various researchers as individual modules to satisfy their own needs but this lead to some fragmentation in the user interface. We have recently started to unify the interface to the different algorithms with an *AnalysisBase* class. Currently `PersistenceLength`, `InterRDF`, `LinearDensity` and `Contacts` analysis have been ported. `PersistenceLength` calculates the persistence length of a polymer, `InterRDF` calculates the pairwise radial distribution function inside of a molecule, `LinearDensity` generates a density along a given axis and `Contacts` analysis native contacts, as described in more detail below. The API to these different algorithms is being unified with a common `AnalysisBase` class, with an emphasis on keeping it as generic and universal as possible so that it becomes easy to, for instance, parallelize analysis. Most other tools hand the user analysis algorithms as black boxes. We want to avoid that and allow the user to adapt an analysis to their needs.

The new `Contacts` class is a good example of a generic API that allows straightforward implementation of algorithms while still offering an easy setup for standard analysis types. The `Contacts` class is calculating a contact map for atoms in a frame and compares it with a reference map using different metrics. The used metric then decides which quantity is measured. A common quantity is the fraction of native contacts, where native contacts are all atom pairs that are close to each other in a reference structure. The fraction of native contacts is often used in protein folding to determine when a protein is folded. For native contacts two major types of metrics are considered: ones based on differentiable functions [BHE13] and ones based on hard cut-offs [FKDD07] (which we set as the default implementation). We have designed the API to choose between the two metrics and pass user defined functions to develop new metrics or measure other quantities. This generic interface allowed us to implement a "$q_1 q_2$" analysis [FKDD07] on top of the `Contacts` class; q_1 and q_2 refer to the fractions of native contacts that are present in a protein structure relative to *two* reference states 1 and 2. Below is an incomplete code example that shows how to implement a $q_1 q_2$ analysis, the default value for the *method* keyword argument is overwritten with a user defined method *radius_cut_q*. A more detailed explanation can be found in the documentation.

```python
def radius_cut_q(r, r0, radius):
    y = r <= radius
    return y.sum() / r.size

contacts = Contacts(u, selection,
```

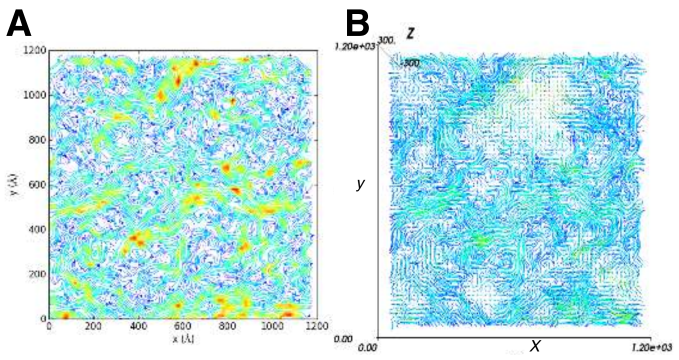


Fig. 4: *Visualization of the flow of lipids in a large bilayer membrane patch.* ***A:*** *2D stream plot (produced with* `MDAnalysis.visualization.streamlines` *and plotted with matplotlib [Hun07]).* ***B:*** *3D stream plot, viewed down the z axis onto the membrane (produced with* `MDAnalysis.visualization.streamlines_3D` *and plotted with MayaVi [RV11]).*

```
                                   (first_frame, last_frame),
                                   radius=radius,
                                   method=radius_cut_q,
                                   start=start, stop=stop,
                                   step=step,
                                   kwargs={'radius': radius})
```

This type of flexible analysis algorithm paired with a collection of base classes enables rapid and easy analysis of simulations as well as development of new ones.

Visualization module

The new `MDAnalysis.visualization` name space contains modules that primarily produce visualizations of molecular systems. Currently it contains functions that generate specialized streamline visualizations of lipid diffusion in membrane bilayers [CRG+14]. In short, the algorithm decomposes any given membrane into a grid and tracks the displacement of lipids between different grid elements, emphasizing collective lipid motions. Both 2D (`MDAnalysis.visualization.streamlines`) and 3D (`MDAnalysis.visualization.streamlines_3D`) implementations are available in MDAnalysis, with output shown in Figure 4. Sample input data files are available online from the Flows website along with the expected output visualizations.

Improvements in the internal topology data structures

Originally MDAnalysis followed a strict object-oriented approach with a separate instance of an Atom object for each particle in the simulation data. The AtomGroup then simply stored its contents as a list of these Atom instances. With simulation data now commonly exceeding 10^6 particles this solution did not scale well and so recently this design was overhauled to improve the scalability of MDAnalysis.

Because all Atoms have the same property fields (i.e. mass, position) it is possible to store this information as a single NumPy array for each property. Now an AtomGroup can keep track of its contents as a simple integer array, which can be used to slice these property arrays to yield the relevant data.

Overall this approach means that the same number of Python objects are created for each Universe, with the number of particles only changing the size of the arrays. This translates into a much smaller memory footprint (1.3 GB vs. 3.6 GB for a 10.1 M atom system), highlighting the memory cost of millions of simple Python objects.

# atoms	v0.15.0	v0.16.0	speed up
1.75 M	19 ms	0.45 ms	42
3.50 M	18 ms	0.54 ms	33
10.1 M	17 ms	0.45 ms	38

TABLE 1: *Performance comparison of subselecting an AtomGroup from an existing one using the new system (upcoming release v0.16.0) against the old (v0.15.0). Subselections were slices of the same size (82,056 atoms). Shorter processing times are better. The benchmarks systems were taken from the vesicle library [KB15] and are listed with their approximate number of particles ("# atoms"). Benchmarks were performed on a laptop with an Intel Core i5 2540M 2.6 GHz processor, 8 GB of RAM and a SSD drive.*

# atoms	v0.15.0	v0.16.0	speed up
1.75 M	250 ms	35 ms	7.1
3.50 M	490 ms	72 ms	6.8
10.1 M	1500 ms	300 ms	5.0

TABLE 2: *Performance comparison of accessing attributes with new AtomGroup data structures (upcoming release v0.16.0) compared with the old Atom classes (v0.15.0). Shorter access times are better. The same benchmark systems as in Table 1 were used.*

This transformation of the data structures from an Array of Structs to a Struct of Arrays also better suits the typical access patterns within MDAnalysis. It is quite common to compare a single property across many Atoms, but rarely are different properties within a single Atom compared. Additionally, it is possible to utilize NumPy's faster indexing capabilities rather than using a list comprehension. This new data structure has lead to performance improvements in our whole code base. The largest improvement is in accessing subsets of Atoms which is now over 40 times faster (Table 1), an operation that is used everywhere in MDAnalysis. Speed-ups of a factor of around five to seven were realized for accessing Atom attributes for whole AtomGroup instances (Table 2). The improved topology data structures are also much faster to initialize, which translates into speed-ups of about three for the task of loading a system from a file (for instance, in the Gromacs GRO format or the Protein Databank PDB format) into a *Universe* instance (Table 3). Given that for systems with 10 M atoms this process used to take over 100 s, the reduction in load time down to a third is a substantial improvement — and it came essentially "for free" as a by-product of improving the underlying topology data structures.

Analysis of large systems

MDAnalysis has been used extensively to study extremely large simulation systems for long simulation times. Marrink and coworkers [IME+14] used MDAnalysis to analyze a realistic model of the membrane of a mammalian cell with 63 different lipid species and over half a million particles for 40 µs. They discovered that transient domains with liquid-ordered character formed and disappeared on the microsecond time scale, with different lipid

# atoms	v0.15.0	v0.16.0	speed up
1.75 M	18 s	5 s	3.6
3.50 M	36 s	11 s	3.3
10.1 M	105 s	31 s	3.4

TABLE 3: *Performance comparison of loading a topology file with 1.75 to 10 million atoms with new AtomGroup data structures (upcoming release v0.16.0) compared with the old Atom classes (v0.15.0). Shorter loading times are better. The same benchmark systems as in Table 1 were used.*

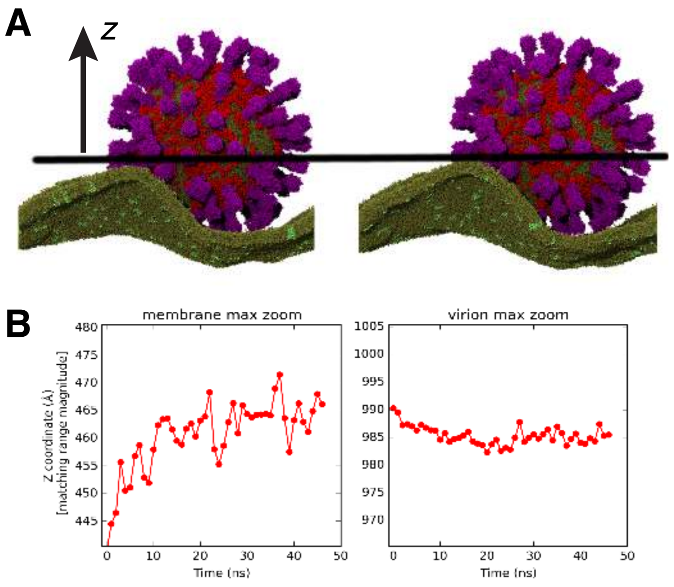


Fig. 5: *Simulation of a coarse-grained model of the influenza A virion membrane (purple/red) close to a model of the human plasma membrane (brown).* ***A****: Left: initial frame. Right: system after 40 ns . A horizontal black guide line is used to emphasize the rising plasma membrane position. The images were produced with VMD [HDS96].* ***B*** *Maximum Z (vertical) coordinate values for the influenza A virus envelope and the plasma membrane are tracked over the course of the simulation, indicating that the membrane rises to rapidly.*

species clustering in a lipid-specific manner. A coarse-grained model of the influenza A virion outer lipid envelope (5 M particles) was simulated for 5 microseconds and the resulting trajectory was analyzed using MDAnalysis [RSP+15] and the open source MDAnalysis-based lipid diffusion analysis code, which calculates the diffusion constants of lipids for spherical structures and planar bilayers [Red14]. The construction of the CG dengue virion envelope (1 M particles) was largely dependent on MDAnalysis [RS16]. The symmetry operators in the deposited dengue protein shell PDB file were applied to a simulated asymmetric unit in a bilayer, effectively tiling both proteins and lipids into the appropriate positions on the virion surface.

More recently, a 12.7 M CG particle system combining the influenza A envelope and a model of a plasma membrane [KS15] were simulated together (Figure 5 A). MDAnalysis was used to assess the stability of this enormous system by tracking, for example, the changes in *Z* coordinate values for different system components (Figure 5 B). In this case, the membrane appeared to rise too rapidly over the course of 50 ns, which suggests that the simulation system will likely have to be redesigned. Such large systems are challenging to work with, including their visualization, and analysis of quantities based on particle coordinates is essential to assess the correct behavior of the simulations.

Other packages that use MDAnalysis

The user interface and modular design work well in complex scripted work flows and for interactive work, as discussed in section Interactive Use and Visualization. MDAnalysis also serves as foundation for other packages. For example, ProtoMD [SMO16] is a toolkit that facilitates the development of algorithms for multiscale (MD) simulations and uses MDAnalysis for on-the-fly calculations of the collective variables that drive the coarse-grained degrees of freedom. The ENCORE package [TPB+15] enables users to compare conformational ensembles generated either from simulations alone or synergistically with experiments. MDAnalysis is also the back end for ST-analyzer [JJW+14], a standalone graphical user interface tool set to perform various trajectory analyses. MDSynthesis [DGS+16] (which is based on datreant (Dotson et al, this issue)) gives a Pythonic interface to molecular dynamics trajectories using MDAnalysis, giving the ability to work with the data from many simulations scattered throughout the file system with ease. It makes it possible to write analysis code that can work across many varieties of simulation, but even more importantly, MDSynthesis allows interactive work with the results from hundreds of simulations at once without much effort.

Conclusions

MDAnalysis provides a uniform interface to simulation data, which comes in a bewildering array of formats. It enables users to rapidly write code that is portable and immediately usable in virtually all biomolecular simulation communities. It has an active international developer community with researchers that are expert developers and users of a wide range of simulation codes. MDAnalysis is widely used (the original paper [MADWB11] has been cited more than 195 times) and forms the foundation for more specialized biomolecular simulation tools. Ongoing and future developments will improve performance further, introduce transparent parallelization schemes to utilize multi-core and GPU systems efficiently, and interface with the SPIDAL library for high performance data analytics algorithms [QJLF14].

Acknowledgments

We thank the members of the MDAnalysis community for their contributions in the form of code contributions (see the file AUTHORS in the source distribution for the names of all 44 contributors), bug reports, and enhancement requests. RG was supported by BBSRC grant BB/J014478/1. ML was supported by the Max Planck Society. JB was supported by the TOP programme of Prof. Marrink, financed by the Netherlands Organisation for Scientific Research (NWO). TR was supported by the Canadian Institutes of Health Research, the Wellcome Trust, the Leverhulme Trust, and Somerville College; computational resources for TR's work were provided by PRACE, HPC-Europa2, CINES (France), and the SBCB unit (Oxford). MNM was supported by the NWO VENI grant 722.013.010. SLS was supported in part by a Wally Stoelzel Fellowship from the Department of Physics at Arizona State University. JD was in part supported by a Wellcome Trust grant 092970/Z/10/Z. DLD was in part supported by a Molecular

Imaging Fellowship from the Department of Physics at Arizona State University. IMK was supported by a REU supplement to grant ACI-1443054 from the National Science Foundation. OB was supported in part by grant ACI-1443054 from the National Science Foundation; computational resources for OB's work were in part provided by the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1053575 (allocation MCB130177 to OB). The MDAnalysis *Atom* logo was designed by Christian Beckstein.

References

[AMS+15] Mark James Abraham, Teemu Murtola, Roland Schulz, Szilárd Páll, Jeremy C. Smith, Berk Hess, and Erik Lindahl. GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX*, 1–2:19 – 25, 2015. URL: http://www.gromacs.org, doi:10.1016/j.softx.2015.06.001.

[BBIM+09] B R Brooks, C L Brooks III., A D Jr Mackerell, L Nilsson, R J Petrella, B Roux, Y Won, G Archontis, C Bartels, S Boresch, A Caflisch, L Caves, Q Cui, A R Dinner, M Feig, S Fischer, J Gao, M Hodoscek, W Im, K Kuczera, T Lazaridis, J Ma, V Ovchinnikov, E Paci, R W Pastor, C B Post, J Z Pu, M Schaefer, B Tidor, R M Venable, H L Woodcock, X Wu, W Yang, D M York, and M Karplus. CHARMM: the biomolecular simulation program. *J Comput Chem*, 30(10):1545–1614, Jul 2009. URL: https://www.charmm.org, doi:10.1002/jcc.21287.

[BHE13] Robert B Best, Gerhard Hummer, and William A Eaton. Native contacts determine protein folding mechanisms in atomistic simulations. *Proc Natl Acad Sci USA*, 110(44):17874–17879, 2013. doi:10.1073/pnas.1311599110.

[BWF+00] Helen M. Berman, John Westbrook, Zukang Feng, Gary Gilliland, T. N. Bhat, Helge Weissig, Ilya N. Shindyalov, and Philip E. Bourne. The Protein Data Bank. *Nucleic Acids Res*, 28(1):235–242, 2000. URL: http://www.rcsb.org/pdb/.

[CCD+05] David A Case, Thomas E Cheatham, 3rd, Tom Darden, Holger Gohlke, Ray Luo, Kenneth M Merz, Jr, Alexey Onufriev, Carlos Simmerling, Bing Wang, and Robert J Woods. The amber biomolecular simulation programs. *J Comput Chem*, 26(16):1668–1688, 2005. URL: http://ambermd.org/, doi:10.1002/jcc.20290.

[CRG+14] Matthieu Chavent, Tyler Reddy, Joseph Goose, Anna Caroline E. Dahl, John E. Stone, Bruno Jobard, and Mark S. P. Sansom. Methodologies for the analysis of instantaneous lipid diffusion in MD simulations of large membrane systems. *Faraday Discuss.*, 169:455–475, 2014. doi:10.1039/C3FD00145H.

[DDG+12] Ron O Dror, Robert M Dirks, J P Grossman, Huafeng Xu, and David E Shaw. Biomolecular simulation: a computational microscope for molecular biology. *Annu Rev Biophys*, 41:429–52, 2012. doi:10.1146/annurev-biophys-042910-155245.

[DGS+16] David Dotson, Richard Gowers, Sean Seyler, Max Linke, and Oliver Beckstein. MDSynthesis: release-0.6.1. (source code), May 2016. URL: https://github.com/datreant/MDSynthesis, doi:10.5281/zenodo.51506.

[FKDD07] Joel Franklin, Patrice Koehl, Sebastian Doniach, and Marc Delarue. MinActionPath: Maximum likelihood trajectory for large-scale structural transitions in a coarse-grained locally harmonic energy landscape. *Nucleic Acids Res*, 35(SUPPL.2):477–482, 2007. doi:10.1093/nar/gkm342.

[GNA+15] Jens Glaser, Trung Dac Nguyen, Joshua A. Anderson, Pak Lui, Filippo Spiga, Jaime A. Millan, David C. Morse, and Sharon C. Glotzer. Strong scaling of general-purpose molecular dynamics simulations on gpus. *Computer Physics Communications*, 192:97–107, 7 2015. URL: http://glotzerlab.engin.umich.edu/hoomd-blue/, doi:10.1016/j.cpc.2015.02.028.

[HDS96] W. Humphrey, A. Dalke, and K. Schulten. VMD – Visual Molecular Dynamics. *J Molec Graphics*, 14:33–38, 1996. URL: http://www.ks.uiuc.edu/Research/vmd/.

[Hin00] K. Hinsen. The molecular modeling toolkit: a new approach to molecular simulations. *J Comput Chem*, 21(2):79–85, 2000.

[HM03] Thomas Hamelryck and Bernard Manderick. PDB file parser and structure class implemented in python. *Bioinformatics*, 19(17):2308–2310, 2003. doi:10.1093/bioinformatics/btg299.

[Hun07] John D. Hunter. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3):90–95, May-Jun 2007. URL: http://matplotlib.org.

[IME+14] Helgi I Ingólfsson, Manuel N Melo, Floris J Van Eerden, Clement Arnarez, Cesar A López, Tsjerk A Wassenaar, Xavier Periole, Alex H De Vries, D Peter Tieleman, and Siewert J Marrink. Lipid organization of the plasma membrane. *J Am Chem Soc*, 136(41):14554–14559, 2014. doi:10.1021/ja507832e.

[JJW+14] Jong Cheol Jeong, Sunhwan Jo, Emilia L Wu, Yifei Qi, Viviana Monje-Galvan, Min Sun Yeom, Lev Gorenstein, Feng Chen, Jeffery B Klauda, and Wonpil Im. ST-analyzer: a web-based user interface for simulation trajectory analysis. *J Comput Chem*, 35(12):957–63, May 2014. doi:10.1002/jcc.23584.

[JOC+15] Anubhav Jain, Shyue Ping Ong, Wei Chen, Bharat Medasani, Xiaohui Qu, Michael Kocher, Miriam Brafman, Guido Petretto, Gian-Marco Rignanese, Geoffroy Hautier, Daniel Gunter, and Kristin A. Persson. Fireworks: a dynamic workflow system designed for high-throughput applications. *Concurrency and Computation: Practice and Experience*, 27(17):5037–5059, 2015. URL: https://github.com/materialsproject/fireworks, doi:10.1002/cpe.3505.

[KB15] Ian M. Kenney and Oliver Beckstein. SPIDAL Summer REU 2015: Biomolecular benchmark systems. Technical report, Arizona State University, Tempe, AZ, October 2015. doi:10.6084/m9.figshare.1588804.v1.

[KS15] Heidi Koldsø and Mark S. P. Sansom. Organization and dynamics of receptor proteins in a plasma membrane. *J Am Chem Soc*, 137(46):14694–14704, 2015. PMID: 26517394. doi:10.1021/jacs.5b08048.

[LAT10] Pu Liu, Dimitris K Agrafiotis, and Douglas L. Theobald. Fast Determination of the Optimal Rotational Matrix for Macromolecular Superpositions. *J Comput Chem*, 31(7):1561–1563, 2010. doi:10.1002/jcc.21439.

[MADWB11] Naveen Michaud-Agrawal, Elizabeth Jane Denning, Thomas B. Woolf, and Oliver Beckstein. MDAnalysis: A toolkit for the analysis of molecular dynamics simulations. *J Comput Chem*, 32:2319–2327, 2011. URL: http://mdanalysis.org, doi:10.1002/jcc.21787.

[MBH+15] Robert T. McGibbon, Kyle A. Beauchamp, Matthew P. Harrigan, Christoph Klein, Jason M. Swails, Carlos X. Hernández, Christian R. Schwantes, Lee-Ping Wang, Thomas J. Lane, and Vijay S. Pande. MDTraj: A modern open library for the analysis of molecular dynamics trajectories. *Biophysical J*, 109(8):1528 – 1532, 2015. URL: http://mdtraj.org, doi:10.1016/j.bpj.2015.08.015.

[McK10] Wes McKinney. Data Structures for Statistical Computing in Python. *Proceedings of the 9th Python Science Conference*, 1697900(Scipy):51–56, 2010. URL: http://conference.scipy.org/proceedings/scipy2010/mckinney.html.

[NRSC16] Hai Nguyen, Daniel R. Roe, Jason Swails, and David A. Case. PYTRAJ: Interactive data analysis for molecular dynamics simulations. (source code), 2016. URL: https://github.com/Amber-MD/pytraj.

[Oro14] Modesto Orozco. A theoretical view of protein dynamics. *Chem. Soc. Rev.*, 43:5051–5066, 2014. doi:10.1039/C3CS60474H.

[PBW+05] JC Phillips, R Braun, W Wang, J Gumbart, E Tajkhorshid, E Villa, C Chipot, RD Skeel, L Kale, and K Schulten. Scalable molecular dynamics with NAMD. *J Comput Chem*, 26:1781–1802, 2005. URL: http://www.ks.uiuc.edu/Research/namd/, doi:10.1002/jcc.20289.

[PG07] Fernando Pérez and Brian E. Granger. IPython: A system for interactive scientific computing. *Comput Sci Eng*, 9(3):21–29, 2007. URL: https://ipython.org/, doi:10.1109/MCSE.2007.53.

[Pli95] Steve Plimpton. Fast parallel algorithms for short-range molecular dynamics. *J Comput Phys*, 117(1):1–19, 1995. URL: http://lammps.sandia.gov/index.html, doi:10.1006/jcph.1995.1039.

[QJLF14] Judy Qiu, Shantenu Jha, Andre Luckow, and Geoffrey C. Fox. Towards HPC-ABDS: An initial high-performance big data stack. In *Building Robust Big Data Ecosystem*, San Diego Supercomputer Center, San Diego, CA, 2014. ISO/IEC JTC 1 Study Group on Big Data. URL: http://spidal.org.

[RCI13] Daniel R. Roe and Thomas E. Cheatham III. PTRAJ and CPPTRAJ: Software for processing and analysis of molecular dynamics trajectory data. *J Chemical Theory Computation*, 9(7):3084–3095, 2013. URL: https://github.com/Amber-MD/cpptraj, doi:10.1021/ct400341p.

[Red14] Tyler Reddy. diffusion_analysis_MD_simulations: Initial release. (source code), September 2014. URL: https://github.com/tylerjereddy/diffusion_analysis_MD_simulations, doi:10.5281/zenodo.11827.

[RG09] Tod D. Romo and Alan Grossfield. LOOS: An extensible platform for the structural analysis of simulations. In *31st Annual International Conference of the IEEE EMBS*, pages 2332–2335, Minneapolis, Minnesota, USA, 2009. IEEE. URL: http://loos.sourceforge.net/.

[RLG14] Tod D. Romo, Nicholas Leioatts, and Alan Grossfield. Lightweight object oriented structure analysis: Tools for building tools to analyze molecular dynamics simulations. *J Comput Chem*, 35(32):2305–2318, 2014. URL: http://loos.sourceforge.net/, doi:10.1002/jcc.23753.

[Roc15] Matthew Rocklin. Dask: Parallel computation with blocked algorithms and task scheduling. In *Proceedings of the 14th Python in Science Conference*, number 130–136, 2015. URL: https://github.com/dask/dask.

[RS16] T. Reddy and M. S. Sansom. The role of the membrane in the structure and biophysical robustness of the Dengue virion envelope. *Structure*, 24(3):375–382, Mar 2016. doi:10.1016/j.str.2015.12.011.

[RSP+15] T. Reddy, D. Shorthouse, D. L. Parton, E. Jefferys, P. W. Fowler, M. Chavent, M. Baaden, and M. S. Sansom. Nothing to sneeze at: a dynamic and integrative computational model of an influenza A virion. *Structure*, 23(3):584–597, Mar 2015. doi:10.1016/j.str.2014.12.019.

[RV11] P. Ramachandran and G. Varoquaux. Mayavi: 3D visualization of scientific data. *Computing in Science & Engineering*, 13(2):40–51, 2011. URL: http://code.enthought.com/projects/mayavi/.

[SKTB15] Sean L. Seyler, Avishek Kumar, M. F. Thorpe, and Oliver Beckstein. Path similarity analysis: A method for quantifying macromolecular pathways. *PLoS Comput Biol*, 11(10):e1004568, 10 2015. doi:10.1371/journal.pcbi.1004568.

[SM14] Victoria Stodden and Sheila Miguez. Best practices for computational science: Software infrastructure and environments for reproducible and extensible research. *J Open Research Software*, 2(1):e21, July 2014. doi:10.5334/jors.ay.

[SMO16] Endre Somogyi, Andrew Abi Mansour, and Peter J. Ortoleva. ProtoMD: A prototyping toolkit for multiscale molecular dynamics. *Computer Physics Communications*, 202:337 – 350, 2016. URL: https://github.com/CTCNano/proto_md, doi:10.1016/j.cpc.2016.01.014.

[TPB+15] Matteo Tiberti, Elena Papaleo, Tone Bengtsen, Wouter Boomsma, and Kresten Lindorff-Larsen. ENCORE: Software for quantitative ensemble comparison. *PLoS Comput Biol*, 11(10):e1004415, 10 2015. doi:10.1371/journal.pcbi.1004415.

[TRB+08] T. Tu, C.A. Rendleman, D.W. Borhani, R.O. Dror, J. Gullingsrud, MO Jensen, J. L. Klepeis, P. Maragakis, P. Miller, K.A. Stafford, and David E. Shaw. A scalable parallel framework for analyzing terascale molecular dynamics simulation trajectories. In *International Conference for High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008.*, pages 1–12, Austin, TX, 2008. IEEE. doi:10.1109/SC.2008.5214715.

[TSTD06] Ilian T Todorov, William Smith, Kostya Trachenko, and Martin T Dove. DL_POLY_3: new dimensions in molecular dynamics simulations via massive parallelism. *Journal of Materials Chemistry*, 16(20):1911–1918, 2006. URL: http://www.ccp5.ac.uk/DL_POLY_CLASSIC/.

[VCV11] Stefan Van Der Walt, S. Chris Colbert, and Gael Varoquaux. The NumPy array: A structure for efficient numerical computation. *Comput Sci Eng*, 13(2):22–30, 2011. URL: http://www.numpy.org/, arXiv:1102.1523, doi:10.1109/MCSE.2011.37.

[WAB+14] Greg Wilson, D A Aruliah, C Titus Brown, Neil P Chue Hong, Matt Davis, Richard T Guy, Steven H D Haddock, Kathryn D Huff, Ian M Mitchell, Mark D Plumbley, Ben Waugh, Ethan P White, and Paul Wilson. Best practices for scientific computing. *PLoS Biol*, 12(1):e1001745, Jan 2014. doi:10.1371/journal.pbio.1001745.

[Wel62] B. P. Welford. Note on a method for calculating corrected sums of squares and products. *Technometrics*, 4(3):419–420, 1962. doi:10.1080/00401706.1962.10490022.