PYBILAYERS: A NEW TOOL FOR THE STRUCTURAL EVALUATION OF COMPUTER SIMULATED LIPID BILAYERS

M. Silvina Moyano[a,b], Hugo E. Di Lorenzo[a], Michael Ferguson[b], Marcos A. Guerra[a], Ana P. Tapia[a], Agustín N. Quiroga[a], Santiago García[a], Ana M. Nuñez[a] and Matías A. Via[b,c]

[a]*Instituto de Bioingenería, Facultad de Ingenería, Universidad de Mendoza, Mendoza, Argentina, silvina.moyano@um.edu.ar*

[b]*Consejo Nacional de Investigaciones Científicas y Técnicas, Mendoza, Argentina*

[c]*Facultad de Ciencias de la Salud, Universidad de Mendoza, Mendoza Argentina*

Keywords: Molecular Dynamics, Python, Analysis, Membranes, Computer Simulation.

Abstract. From a biological point of view, the study of lipid bilayers is of great importance as they form part of the plasmatic membrane in each and every living cell. Due to the complexity of their basic functions, the mechanical properties of lipid bilayers have long been studied. These investigations have not been limited to experimental approaches and there are now many research groups tackling the problem from a computer simulation perspective. In general, these studies are performed using classical molecular dynamics simulations with coarse grained lipids (MARTINI force field). It is the analysis of these simulations that allows for a potentially greater understanding of bilayers mechanics. In this work, we present a new tool which has been specifically designed for lipid bilayer simulations. The tool focuses on two areas 1) the analysis of the thermodynamic quantities of the system, and 2) structural analysis based on the instantaneous configurations of the system. Designed with generality at its forefront, the tool can read configurations of a given membrane and, through the incorporation of the parallelisation libraries available in python, users can expect a high throughput of complex analyses on a modern desktop computer. The implementation of this tool in the wider biophysics of lipid membranes community will provide consistent, standarised results and, therefore, open a new pathway to a greater understanding of lipid bilayer mechanics.

1 INTRODUCTION

Cells are generally viewed as the building blocks of all living things. Cells themselves are actually comprised of a great number of individual components, each of which determines different mechanical properties. These properties are essential for life itself as a great many functions, both internal and external, of each cell are dependent on mechanical phenomena. Cellular mechanics is the generic name given to this group of processes and it includes, shape retention, cellular mobility, cellular adhesion, and the interaction between cells and their environment.(Kamm and Mofrad, 2006) Cells are separated from the environment by a thin lipid bilayer which is comprised of a mixture of cholesterol, glucolipids, phospholipids, and a range of transmembrane proteins. The phospholipids are the most abundant and as their length and polarity varies, so do the structural characteristics and mechanical properties of individual membranes. The composition of lipids differs and generally responds to changes in intensive thermodynamic variables such as temperature, pressure and solvent concentration.(Heimburg, 2007)

The mechanical properties of cells have been the subject of study and discussion for centuries due to the large number of mechanical processes carried out by each unit to fulfill its basic functions. Currently there are experimental methods that allow for the application of forces in the order of pN and the measurement of distances to the order of nanometers. This allows to study viscoelastic properties in macromolecules and fractions of a cell. The external forces are applied by optical tweezers, glass needles, magnetic particles and indenters, while the deformations in a scale of nanometers to microns are measured by optical detectors or by high resolution microscopy.(Kamm and Mofrad, 2006; Heimburg, 2007) However, experiments alone are not always sufficient to describe the mechanical properties of cellular membranes.

Computer simulation is a complementary technique which allows for the study of materials at the atomistic/molecular level. It is often employed alongside experimental investigations to aid in the description of small length- and time-scale processes. Molecular dynamics (MD) allows for the calculation of the movement of individual classical molecules in models of solids, liquids and gases.(Allen and Tildesley, 2017) It starts from the idea that the behavior of a system can be calculated if we have a group of initial conditions and the interaction forces between all of the components of the system. From here, a sequence of configurations, or trajectory, is generated by continually solving Newton's equations of motion for all of the particles in the system. The inter-particle interactions are described by empirical potentials taken from experiments and quantum mechanics calculations. Selecting the appropriate parameters for the system under study allows for the accurate calculation of values associated with experimental observables. MD simulations may contain hundreds of millions of atoms and configurations in the trajectory are saved frequently. This leads to large amounts of data that must be processed in order to determine the macroscopic properties of the system. Analysing the trajectories generated by MD simulations can often become tedious and tiresome tasks for individual scientists to perform, at times requiring days of discrete calculations in order to obtain the desired results. For this purpose we are developing a new analysis tool called PyBiLayers. And, while still in early stages of development, it is already proving to be a fast and accurate tool for the analysis of soft biomaterials, such as lipid bilayers, under mechanical stress.

2 TOOLS AND METHODS

Any automated tool must be programmed, and we have decided to write PyBiLayers in the Python language. Python(van Rossum, 1995) has become one of the favoured programming languages for scientific investigations around the world. This is due to various factors, from the fast learning curve, readability and power of the language, to the many functions and modules which are integrated through Python's free software licence, and its ability to incorporate other programming languages. This final characteristic is one which we will explore at a later stange in this work. Further to the technical advantages, Python has a large, active programming community which, thanks to their support, allows for quick and efficient code development.

The base programming of PyBiLayers is written in Python 3 due the aforementioned advantages and that it will continue to receive support and updates unlike its predecessor, Python 2. Beyond this, the employment of the NumPy(Walt et al., 2011) and SciPy(Jones et al., 2001) modules are essential to the efficiency of our calculations. NumPy provides support for application of mathematical functions to matrices and multi-dimensional arrays, while SciPy allows us to perform optimizations, linear algebra, integration, and interpolation. As we are dealing with large amounts of three-dimensional data, the inclusion of Big Data processing tools is key. Big Data, refers to data sets to big to be handled by traditional informatics programs. To work with such large amounts of data we are investigating the advances made in the area of process parallelisation though the OpenCL library(Stone et al., 2010) and its Python module PyOpenCl.(Klöckner et al., 2012) Parallelisation of the code is of great importance if we are to achieve operational efficiency when searching and storing data for analysis, and in the visualisation of complex results. To give the users visual results, we are employing the powerful, and very popular, MatPlotLib module(Hunter, 2007) which is designed to allow the user to produce high quality graphics with relatively few lines of code.

In its current form PyBiLayers, already has a wide range of functionalities available to the simulator. These include the geometric center of mass (COM) of the system, true COM of the lipid species, dipole moment of water, self-diffusion coefficient of each species, lipid density in the xy-plane, area of potential pores, average radial distribution function, and plotting of the system thermodynamic quantities. Unilateral density profiles of the number of molecules, molecular and atomic charges are also calculated.

The geometric COM of the system is calculated simply as follows,

$$COM = \frac{1}{M}\sum_{i=1}^{n} m_i \boldsymbol{r}_i \tag{1}$$

where M is the total mass of all n particles in the system, m_i is the mass of the given particle i and $\boldsymbol{r}_i$ are its coordinates. The dipole moment of the water particles in the system is calculated by,

$$\langle\vec{\mu}\rangle = \frac{1}{n_w}\sum_{i=1}^{n_w} q_i \vec{r}_i \tag{2}$$

where $\langle\vec{\mu}\rangle$ is the average molecular dipole moment for all waters in the system, q_i is the charge differential between the O and the two H atoms along the vector $\vec{r}_i$. The vector $\vec{r}_i$ runs from the midpoint between the two H atoms and the O atom of each water molecule.

The self-diffusion coefficient is commonly derived from an mean squared displacement (MSD) calculation of the atomic coordinates throughout a trajectory. In general, only the initial configuration of the system, i.e. at $time = 0$, is used as the reference point for the MSD calculation. With PyBiLayers we have implemented the more accurate MSD from multiple time origins(Allen and Tildesley, 2017) calculation, with minimal effect on the overall calculation time.

The lipid density in the xy-plane is calculated as a two-dimensional histogram in the following form;

$$\rho_{x,y} = \frac{1}{A_{x,y}} \sum_{x=1}^{h} \sum_{y=1}^{h} n_{x,y}^{lipid} \tag{3}$$

where ρ_{xy} is the lipid density in the histogram bin x, y, h is the number of histogram bins in each direction, $n_{x,y}^{lipid}$ is the number of lipids in histogram bin x, y and $A_{x,y}$ is the area enclosed by the denoted histogram bin.

The radial distribution function $g(r)$ provides information about the local density of the system at a given radius r from a a specific point or particle.

$$g(r) = \frac{n(r \pm \frac{\Delta r}{2})}{\Omega(r \pm \frac{\Delta r}{2})} \frac{1}{\rho} \tag{4}$$

where $n(r \pm \frac{\Delta r}{2})$ is the number of particles within a shell of inner radius $r - \frac{\Delta r}{2}$ and outer radius $r + \frac{\Delta r}{2}$, Ω is the given shell's volume and ρ is the global density for the full system.

For a number of the functions in PyBiLayers it is vital to accurately determine the centre of mass of each molecule in the system. Generally, this is calculated by a weighted average of the molecule's individual particles. However, due to the employment of periodic boundary conditions in molecular dynamics simulations the weighted average method has potential to produce errors. For example, if half of a molecule crosses the maximum of the simulation cell that half will begin to appear in the opposite side of the cell. If the COM was taken from a weighted average at this point it would give a result in the center of the box, i.e. not where the molecule is. To avoid these errors we have opted to use a slightly more resource intensive method(Bai and Breen, 2008), where the coordinates of the molecule's particles are projected onto a circle and averaged to obtain the COM on the same circle. The projection is then reverted to give the cartesian coordinates of the COM. This allows us to analyse exactly where each particle is during the simulation. An added benefit to this calculation of course is the generation of a trajectory with unwrapped coordinates, which improves the visual inspection of the trajectory. As mentioned, we make great use of the NumPy module throughout the code. It has been especially advantageous in radial distribution, and uni- and bilateral density functions through the use of the 1D and 2D histogram functions. Calculation times for these functions are up to six times faster than were accomplished through more iterative techniques.

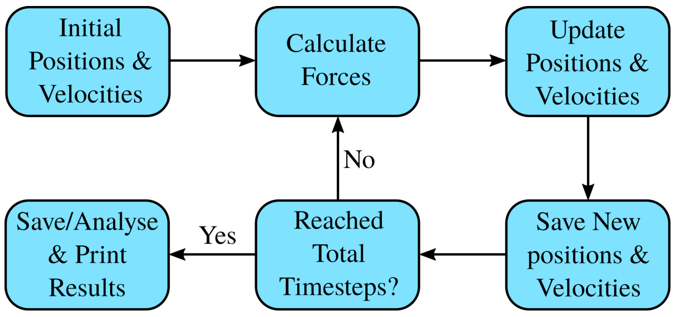
3 RESULTS

PyBiLayers is designed around molecular dynamics simulations performed using the LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) code.(Plimpton, 1995) Atoms are heavy enough to be considered as classical objects, therefore by solving Newton's second equation of motion, Equation 5, their movements can be determined.

$$m\boldsymbol{a} = m\frac{d^2\boldsymbol{r}}{dt^2} = -\frac{\partial V}{\partial \boldsymbol{r}} \tag{5}$$

where m is mass, $\boldsymbol{a}$ is acceleration V is the potential energy from all of the atomic positions which are contained in the vector $\boldsymbol{r}$ and t is time.(Jensen, 2007) For many-body systems the solutions of Newton's equation becomes extremely complex and must therefore be calculated numerically rather than analytically. There are multiple different algorithms available for this and LAMMPS takes advantage of the popular Verlet algorithm.(Jensen, 2007; Swope et al.,

1982) Continually determining the motion of all of the particles in a systems allows for the generation of a trajectory, following the process shown in Figure 1. We chose



Initial Positions & Velocities
Calculate Forces
Update Positions & Velocities
No
Save/Analyse & Print Results
Yes
Reached Total Timesteps?
Save New positions & Velocities

Figure 1: Process by which trajectories are generated in LAMMPS. Image adapted from Cai et al. (2012); Ferguson (2017).

LAMMPS because it is a very fast, flexible and powerful MD code however, there is a lack of soft matter analysis codes available for the post-processing of the trajectories that it produces. To test the code we performed initial simulations on a trial system consisting of a pure DOPC (1,2-dioleoyl-SN-glycero-3-phosphocholine) membrane immersed in water. We used the coarse grained MARTINI force field(Marrink et al., 2007) to describe all of the inter-particle interactions in the system, employing the polarizable description(Yesylevskyy et al., 2010) for water molecules. MARTINI was chosen as it is widely used in the biophysics simulation community as it allows for the simulations of realistic system sizes for long time scales.(Ingólfsson et al., 2014) During simulations LAMMPS typically generates two separate output files. One in which

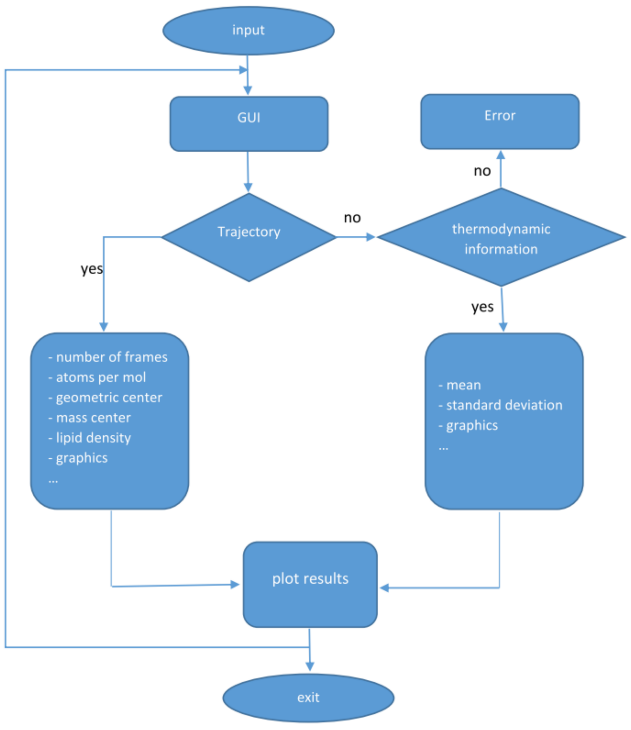


Figure 2: Flow diagram demonstrating the processing of LAMMPS thermodynamic and trajectory data by the PyBiLayers code.

the trajectory is printed and another which contains thermodynamic information. The current version of PyBiLayers is capable of reading both of these files independently and perform the analyses mentioned in the previous section. Once complete, the analysis is output as required by the user; as a new tabulated data file, a publication quality graphic, or both simultaneously. The file containing the thermodynamic output from LAMMPS also contains the initialisation, finalisation and computational efficiency data. PyBiLayers extracts the thermodynamic data for analysis and graphing purposes following the process as described in Figure 2.

Figure 3 shows one of the most basic functions of the PyBiLayers code, the translation of tabulated energy data into a graph which allows for a simpler interpretation of system behaviour. In this case the data is taken from a simulation where the DOPC membrane was annealed from $0\,\mathrm{K}$ to $300\,\mathrm{K}$ during a period of $0.5\,\mathrm{ns}$.

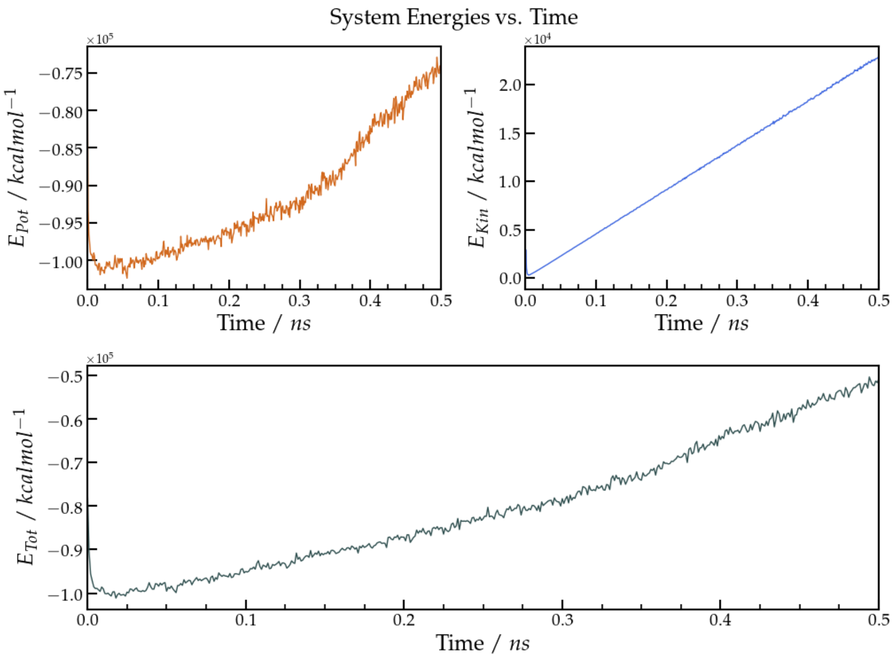


Figure 3: Output of PyBiLayers showing the evolution of the potential (E_{Pot}), kinetic (E_{Kin}) and total (E_{Tot}) energies of a DOPC membrane system during an annealing process.

Graphics similar to that shown in Figure 3 can be produced for any of the possible thermodynamic data outputs from LAMMPS.(Plimptop, 1995) Analysis based on the thermodynamic data, such as the surface tension, also have their own dedicated plotting functions.

Performing analyses based on the trajectory data is, in general, a more complex and time consuming process. As mentioned in the methods section, we took advantage of the histogram functions found in the NumPy library to perform uni- and bi-lateral density analyses. Here, in Figure 4 we show the graphic output of the PyBiLayers codes for the unnormalised lipid density analysis in the xy-plane. This data is generated from the trajectory information by creating a 2D histogram in x and y with 20 bins in each direction, thus leading 400 bins in total for the plane. These bins are populated by using the x and y coordinates of the particles which represent the membrane in the system. We found that the most effective method to visualise this data was to generate a 2D heat-map of the data, Figure 4, an option available in the MatPlotLib library.

A grid of 400 histogram bins in some cases may be small and lead to observable discretisation of data in the plot. To avoid this we called the 2D interpolation functions from the SciPy library,

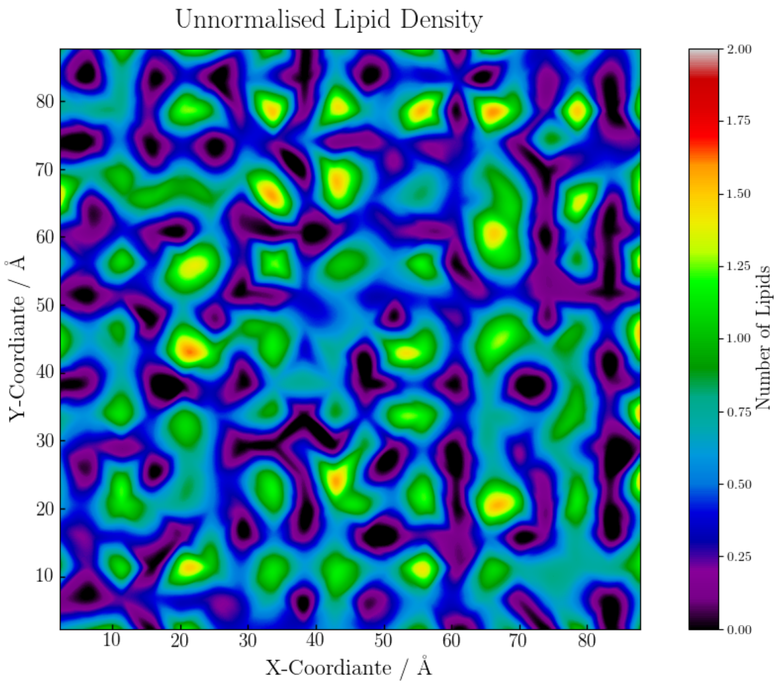


Figure 4: Graphical representation of the unnormalised lipid density in the xy-plane of the simulation cell.

specifically employing the cubic spline interpolation. This generates a smoother, more relatable, view of the lipid density and it allows for a more accurate calculation of the areas where the is a specific density of lipids. This analysis is performed for every frame in the trajectory, allowing the simulator to directly monitor the evolution of the lipid density in the membrane when a stress is applied.

4 CONCLUSIONS

In this work we have shown the potential applications of a new generic analysis tool for simulations in the area of cellular mechanics. While the code is still in the early stages of development it is already showing potential in the automation of membrane simulation analysis and the production of high quality output.

We are in continuous development of the code and soon plan to bring features including, animated graphic outputs, lipid orientation analysis, multi-method line tensions determination, and surface construction leading to free and occupied volume calculations. In tandem to the expansion of functionality, we plan to implement the parallelisation of the code for both multi-core processors and graphical processing units (GPUs) were possible. While a considerable amount of development remains, we are confident that, once released, PyBiLayers will be adopted by many in the biophysics of lipid membranes community.

AUTHOR CONTRIBUTIONS

Authors M. S. Moyano, H. E. Di Lorenzo, and M. Ferguson all contributed equally to the design, preparation, and writing of the presented work.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge funding from la Dirección de Investigaciones de la Universidad de Mendoza (DIUM), Argentina. Resolución: HCS N^{o} 81/2016

REFERENCES

Allen M.P. and Tildesley D.J. *Computer simulation of liquids*. Oxford University Press, Oxford, 2017. ISBN 9780198803201.

Bai L. and Breen D. Calculating center of mass in an unbounded 2d environment. *J. Graph. Tools*, 13(4):53–60, 2008. doi:10.1080/2151237X.2008.10129266.

Cai W., Li J., and Yip S. Molecular dynamics. In R.J. Konings, editor, *Comprehensive Nuclear Materials*, volume 1, chapter 9, pages 249 – 265. Elsevier, Oxford, 2012. ISBN 978-0-08-056033-5. doi:http://dx.doi.org/10.1016/B978-0-08-056033-5.00128-2.

Ferguson M. *Understanding the Mechanochemical Reaction Between Aspirin and Meloxicam*. Ph.D. thesis, Queen's University Belfast, 2017.

Heimburg T. *Thermal Biophysics of Membranes*. Wiley-VCH Verlag GmbH and Co. KGaA, 2007. ISBN 9783527611591.

Hunter J.D. Matplotlib: A 2d graphics environment. *Comput. Sci. Eng.*, 9(3):90–95, 2007. doi:10.1109/MCSE.2007.55.

Ingólfsson H.I., Lopez C.A., Uusitalo J.J., de Jong D.H., Gopal S.M., Periol e.X., and Marrink S.J. The power of coarse graining in biomolecular simulations. *Wiley Interdiscip. Rev.: Comput. Mol. Sci.*, 4(3):225–248, 2014. doi:10.1002/wcms.1169.

Jensen F. *Introduction to Computational Chemistry*. John Wiley & Sons, Chichester, 2nd edition, 2007.

Jones E., Oliphant T., Peterson P., et al. SciPy: Open source scientific tools for Python. http://www.scipy.org/, 2001. [Online; accessed 17-Jul-2018].

Kamm R.D. and Mofrad M.R.K. *Cytoskeletal Mechanics: Models and Measurements in Cell Mechanics*. Cambridge Texts in Biomedical Engineering. Cambridge University Press, 2006. doi:10.1017/CBO9780511607318.

Klöckner A., Pinto N., Lee Y., Catanzaro B., Ivanov P., and Fasih A. PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation. *Parallel Computing*, 38(3):157–174, 2012. doi:10.1016/j.parco.2011.09.001.

Marrink S.J., Risselada H.J., Yefimov S., Tieleman D.P., and de Vries A.H. The martini force field: Coarse grained model for biomolecular simulations. *J. Phys. Chem. B*, 111(27):7812–7824, 2007. doi:10.1021/jp071097f.

Plimpton S. Fast Parallel Algorithms for Short-Range Molecular Dynamics. *J. Comput. Phys.*, 117(1):1–19, 1995. doi:10.1006/jcph.1995.1039.

Plimptop S. thermo_style command. https://lammps.sandia.gov/doc/thermo_style.html, 1995. [Online; accessed 17-Jul-2018].

Stone J.E., Gohara D., and Shi G. Opencl: A parallel programming standard for heterogeneous computing systems. *IEEE Des. Test*, 12(3):66–73, 2010. ISSN 0740-7475. doi:10.1109/MCSE.2010.69.

Swope W.C., Andersen H.C., Berens P.H., and Wilson K.R. A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters. *J. Chem. Phys.*, 76(1):637–649, 1982. doi:http://dx.doi.org/10.1063/1.442716.

van Rossum G. Python reference manual. Technical Report, Amsterdam, The Netherlands, 1995.

Walt S.v.d., Colbert S.C., and Varoquaux G. The numpy array: A structure for efficient numerical computation. *Comput. Sci. Eng.*, 13(2):22–30, 2011. doi:10.1109/MCSE.2011.37.

Yesylevskyy S.O., Sch afer L.V., Sengupta D., and Marrink S.J. Polarizable water model for the coarse-grained martini force field. *PLoS Comput. Biol.*, 6(6):1–17, 2010. doi:10.1371/journal.pcbi.1000810.