Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

pysimm: A python package for simulation of molecular systems

Michael E. Fortunato [a], Coray M. Colina [a,b,*]

[a] Department of Chemistry, University of Florida, Gainesville, FL 32611, United States

[b] Department of Materials Science and Engineering and Nuclear Engineering, University of Florida, Gainesville, FL 32611, United States

ARTICLE INFO

Article history:
Received 15 August 2016
Received in revised form
1 December 2016
Accepted 5 December 2016

Keywords:
Amorphous polymers
Molecular simulation
Python

ABSTRACT

In this work, we present *pysimm*, a python package designed to facilitate structure generation, simulation, and modification of molecular systems. *pysimm* provides a collection of simulation tools and smooth integration with highly optimized third party software. Abstraction layers enable a standardized methodology to assign various force field models to molecular systems and perform simple simulations. These features have allowed *pysimm* to aid the rapid development of new applications specifically in the area of amorphous polymer simulations.

Code metadata

Current code version	v0.1
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-16-00070
Legal Code License	MIT
Code versioning system used	git
Software code languages, tools, and services used	python2.7
Compilation requirements, operating environments & dependencies	Linux
If available Link to developer documentation/manual	http://pysimm.org/documentation/
Support email for questions	support@pysimm.org

1. Motivation and significance

Molecular simulations have proven to be useful tools to explore time and length scales often inaccessible through experimental means and can offer invaluable insight to supplement or explain experimental data. As available computational power increases so does the practicality of simulating larger and more complex molecular systems. It is important that the development of new software keeps pace to take full advantage of this continual increase in computational power.

New software is often developed to take advantage of new hardware or hardware acceleration, but that computational power must also be accessible to researchers to solve scientific problems. The use of highly optimized software is sometimes limited to those that have a deep understanding of computer programming and high performance computing techniques. However, there is a potentially large user base of scientists that could employ computers for research if relatively easy to use applications were available to them.

One area where computational studies can have a great impact is in the amorphous polymeric community. It is nontrivial to predict amorphous polymer chain geometries due to a lack of well defined three dimensional structures. Once structures are obtained, however, there are numerous *in silico* characterizations that could be performed to interrogate material properties. While there are many tools that exist designed to simplify simulation setup for biomolecules fewer tools exist to simplify the structure preparation process for amorphous polymers. This may be due to the great diversity in polymer composition and morphology as compared to protein chains consisting of a small set of amino acids. These tools often take known three dimensional structures determined from experimental data, such as x-ray crystallography, and apply a given force field model. Predicting secondary, tertiary,

* Corresponding author at: Department of Chemistry, University of Florida, Gainesville, FL 32611, United States.
E-mail address: colina@chem.ufl.edu (C.M. Colina).

http://dx.doi.org/10.1016/j.softx.2016.12.002

and quaternary structures of proteins from their linear sequence of amino acids is still an area of active research, however. Additionally, it may be intractable to have a highly optimized but broadly applicable force field capable of capturing all polymer physics given the variety of functional groups used in polymer chemistry. The availability of software that simplifies the process to study a system with various force fields can yield insight into which models can capture a subset of polymer physics applicable to a given scientific inquiry as well as help guide force field development in the polymer simulation community.

pysimm is a python package designed specifically to address these issues by providing users simple but powerful tools to build structures and apply various force fields models to study amorphous polymer morphologies. The software is built to exist between other layers of software that can provide highly optimized execution of expensive calculations and user friendly (graphical) interfaces. Bridging this gap helps to open up doors to many more researchers to utilize computational power to study polymer science.

There is a plethora of software that exists to do various tasks associated with molecular simulations from highly optimized simulation packages such as LAMMPS [1], AMBER [2], GROMACS [3], CHARMM [4], HOOMD [5], OpenMM [6] and Cassandra [7], to software designed to prepare systems for simulation such as AmberTools LEaP [2], PackMol [8], and mBuild [9], to computational chemistry packages such as RDKit [10] and OpenBabel [11], to post-processing software such as AmberTools CPPTRAJ [2], LAMMPS Pizza.py [1], PoreBlazer [12], VMD [13], PyMol [14] and MD-Traj [15], and software specifically designed to create polymer structures such as PolymerModeler [16], Polymatic [17] and Assemble! [18]. Despite the length of this list of software, it is far from exhaustive. It is improbable that any computational scientist would attempt to learn the intricate details of all of these software packages, but may want to use bits and pieces of each of them.

The goal of *pysimm* is not to replace any of the above software packages for their specific purpose, but to facilitate the collaboration between them by providing simple API behavior to integrate different features using the popular scripting language of Python. For example, *pysimm* was used in the development of the web-based application *nusimm*: nanoHUB User Simulation Interface for Molecular Modeling [19] which provides a user friendly interface for performing high performance molecular simulations. Integration with LAMMPS [1], PoreBlazer [12], PyMol [14], and Polymatic [17] allows users to generate a polymer structure with Polymatic, sample conformational space of polymer chains through simulation with LAMMPS, visualize and analyze 3D structures with PyMol, and characterize pore structure with PoreBlazer. The infrastructure built by nanoHUB.org [20] provided convenient access to computational resources while the *pysimm* platform provided reusable and modular tools which allowed the development of *nusimm* to focus mainly on designing the front end of the application to make an organized and intuitive guided experience through polymer structure generation, equilibration and characterization procedures.

2. Software description

pysimm scripts are written using standard python syntax. Parts of the *pysimm* package such as force field typing using the **forcefield** sub-package (see Section 2.1.2) can be used as standalone python code, however most of the power of using the *pysimm* platform incorporates other third-party open source software. In these cases, proper installation of external software is considered a prerequisite to using *pysimm*. Configuring the integration between *pysimm* and other software packages is handled through environment variables or can be modified in __init__.py. Very often the integration only requires knowledge of the path to an executable. For example, to configure the use of the LAMMPS software package through the **lmps** module (see Section 2.1.3), a user can either set an environment variable 'LAMMPS_EXEC' on their own or modify a line of code in __init__.py to search for their executable (see directions in the README included in the *pysimm* source code repository).

pysimm can be used interactively by invoking python at a Linux terminal and importing *pysimm*. This technique can be extremely useful for examining and preparing molecular systems for simulation. For example, it is often of interest to visually inspect a system under study in order to get an atom's tag, or its unique identifier. Using *pysimm* interactively can allow users to read system information from a file and visualize a system to help determine specific tags. Upon closing the visualization software the user is returned to the interactive python interpreter to continue to configure or modify the molecular system. Alternatively *pysimm* scripts can be run in batches by passing a script to the python interpreter.

2.1. Software architecture and functionality

The *pysimm* python package is broken into three core parts. The **system** module is capable of generating, modifying and interpreting molecular data. This includes the position of atoms and how they interact. The way in which the atoms interact in molecular systems usually refers to force fields or inter-atomic potentials, for which the **forcefield** package exists in *pysimm*. This package interprets force field data and can assign these parameters to a given molecular system. The last core module handles how molecular simulations are performed. This task is usually very computationally expensive and there are many highly optimized software packages that have been designed specifically to accomplish these tasks efficiently. For this reason, *pysimm* opts to employ third party simulation software. While currently *pysimm* has been developed to integrate seamlessly with parts of the LAMMPS simulation package through the **lmps** module, the future direction of the package involves expanding to integrate with numerous simulation packages, each which can provide unique functionality.

The philosophy behind creating a python interface for a molecular simulation package is that the level of abstraction allows users to take advantage of the inherent optimization without having to worry about the sometimes cryptic syntax required to set up a simulation. The file formatting, organization, and execution is handled behind the scenes, while users only have to worry about setting up higher level molecular simulation settings such as temperature or pressure. However, for advanced users who understand the syntax for a given simulation software, the option still exists to provide raw input to run a simulation.

2.1.1. pysimm.system

The **system** module has class definitions for components of molecular systems such as **Particle**, **Bond**, **Angle**, **Dihedral**, and **Improper** to help define bonding geometry in molecular models. Each of these items reference a type object (**ParticleType**, **BondType**, etc.) that defines force field parameters. As an example, all **Bond** objects that behave the same way in a model reference the same **BondType** object that may define a bond stretching force constant and equilibrium bond length. All of this information is organized in a **System** object, which offers convenient programmatic access to all of the molecular system data.

While **System** objects can be built from the bottom-up by creating particles, connecting them with bonds, assigning angles, and so on (see example 1 in the source code repository), users often obtain data files that contain at least a subset of this

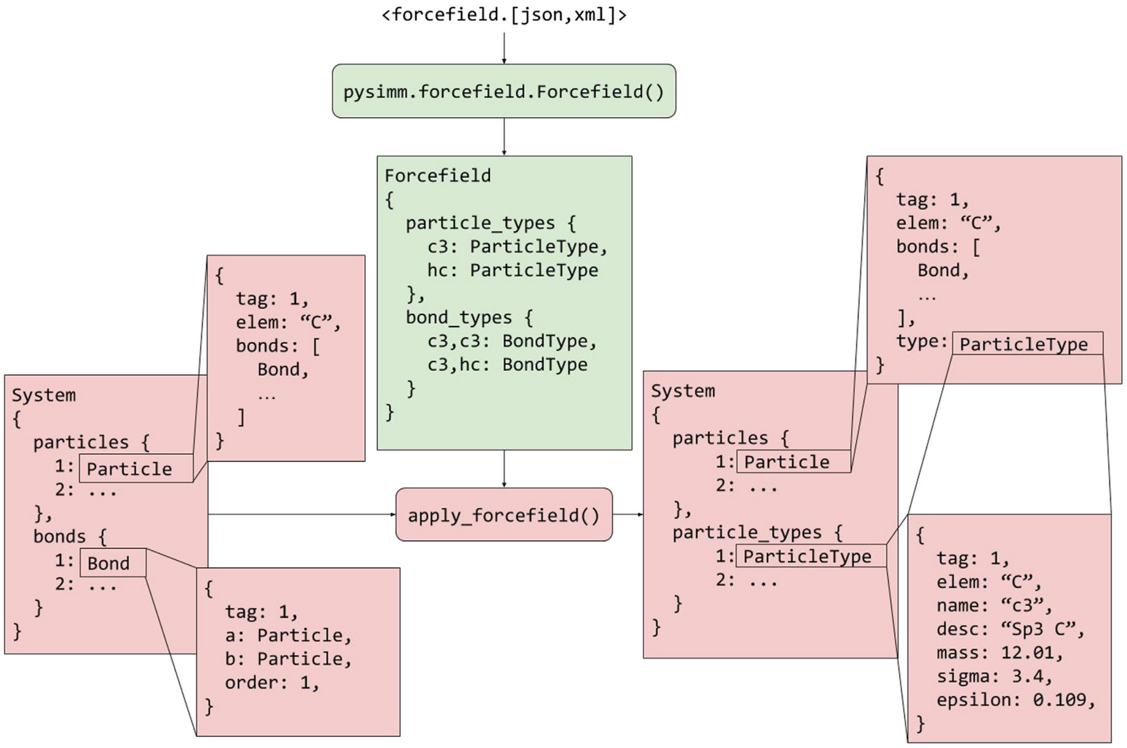
<forcefield.[json,xml]>
pysimm.forcefield.Forcefield()
Forcefield { particle_types { c3: ParticleType, hc: ParticleType }, bond_types { c3,c3: BondType, c3,hc: BondType } }
System { particles { 1: Particle 2: ... }, bonds { 1: Bond 2: ... } }
{ tag: 1, elem: "C", bonds: [Bond, …] }
{ tag: 1, a: Particle, b: Particle, order: 1, }
apply_forcefield()
System { particles { 1: Particle 2: ... }, particle_types { 1: ParticleType 2: ... } }
{ tag: 1, elem: "C", bonds: [Bond, …], type: ParticleType }
{ tag: 1, elem: "C", name: "c3", desc: "Sp3 C", mass: 12.01, sigma: 3.4, epsilon: 0.109, }

Fig. 1. A **Forcefield** object is instantiated using force field data contained in a data file. The apply_forcefield() method is called on a **System** object containing atom elemental composition and bond orders using a given **Forcefield** object, which interprets local bonding environments and assigns force field parameters accordingly in the form of **ParticleType**s, **BondType**s, etc. It is important to note that the **System** object is modified in place, and therefore the two **System** objects represented on the left and the right are pointing to the same object in memory.

information, for example, from PubChem or other open access databases. The **system** module contains functions to read various file formats such as XYZ, PDB, CML, or SDF. These functions interpret data files and organize molecular system data into a consistent and extensible data structure regardless of initial file format. A representation of this **System** object can be written to various commonly used file formats or a YAML file format which supports inclusion of arbitrary data that may not have a place in standardized file formats. One example would be including data that identifies atoms in monomers that can be joined to make new polymer bonds. This usage of the YAML file format used in *pysimm* allows for serialization of the object oriented representation of a molecular system that can be transferred over a data connection and reconstructed on the other end.

2.1.2. pysimm.forcefield

The second core sub-package of *pysimm*, the **forcefield** package, is used to examine **System** objects and assign appropriate force field types to **Particle**s, **Bond**s, **Angle**s, etc. Force field parameters can be found in the literature or open source software packages. However, a universal standard for data formatting despite the functional similarity between different force fields is currently lacking. *pysimm* contains a collection of select force fields whose data has been formatted to a standard that can be interpreted by software code to instantiate **ParticleType**, **BondType**, **AngleType**, **DihedralType**, and **ImproperType** objects defined in the **system** module. Because each force field can have its own unique requirements for typing rules, each force field implemented in *pysimm* has its own class definition with typing rules defined in class methods. At the time of the first public release, a few atomistic force fields are supported (GAFF, DREIDING, PCFF), however the design of the **forcefield** module is extensible to coarse-grained models as well. Assignment of **ParticleType** objects to **Particle** objects is determined by the local bonding environment in a given **System** object. Bond, angle, dihedral and improper force field types are then identified using combinations of particle type names. This methodology follows the logic used when manually assigning force field parameters but benefits from standardization offered by the **forcefield** module. An illustration showing how force field typing works is shown in Fig. 1.

2.1.3. pysimm.lmps Module

The last core *pysimm* module is responsible for integration with the LAMMPS simulation package. In short, this module provides a level of abstraction between molecular system data and the software that will perform a given simulation. Data requirements for a simulation, such as a molecular dynamics ensemble or temperature at which a simulation should be performed, are separated from the syntactical requirements imposed by LAMMPS, effectively removing the burden of preparing input for the LAMMPS software from the end user. In theory, this abstraction can be applied to other simulation packages in which case a similar *pysimm* work flow could be easily transitioned from one simulation package to another.

The **lmps** module contains a class definition for a **Simulation** object, which accepts a **System** object and various keyword arguments to set up a simulation. As much as possible, settings for configuring a simulation are inferred from the **System** object. For example, rather than requiring a user to explicitly set a bond interaction style as harmonic, this information is obtained from the **System** object.

Templates for molecular dynamics simulations or energy minimization simulations can be created using the **MolecularDynamics** and **Minimization** class definitions respectively. These

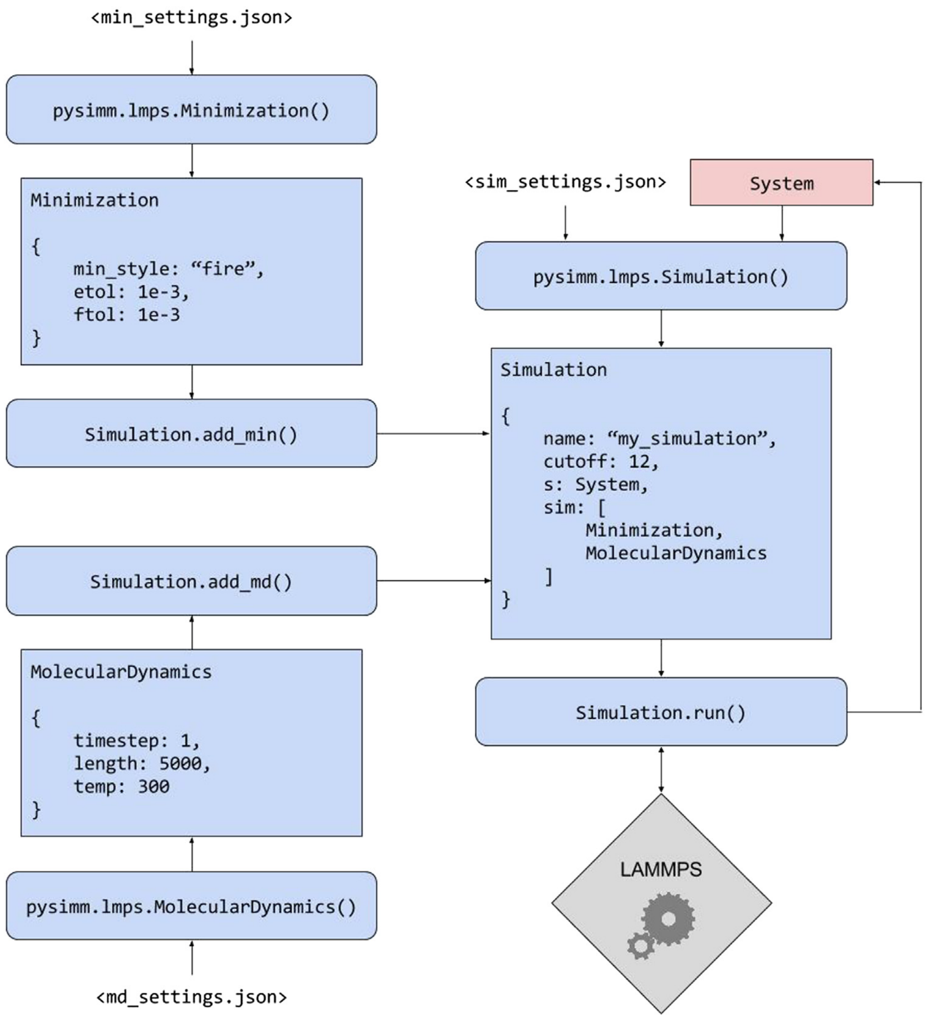

Fig. 2. Simulation settings are stored as key–value pairs and used in conjunction with an existing **System** object to instantiate a **Simulation** object. **Minimization** and/or **MolecularDynamics** objects are instantiated using key–value pair settings, and subsequently added to the **Simulation** object. Calling the **Simulation**.run() method invokes LAMMPS and, upon completion, the existing **System** object is updated in memory.

templates are added to a **Simulation** object, and the simulation can be executed by calling the **Simulation**.run() method. The run() method prepares the required input file(s) and calls the LAMMPS executable defined by the user during initial *pysimm* configuration. At completion of the simulation, the simulation box dimensions, particle positions and particle velocities are updated in the original System object. This complete abstraction from the simulation package only requires any new implementation of simulation packages to likewise update these values in the **System** object as well. To use multiple processors, passing an optional keyword argument, 'np', will specify how many processors should be used. A graphical representation of this templating behavior is shown in Fig. 2.

3. Illustrative example

3.1. Random walk polymerization

The *pysimm* functionality outlined above was used to create an application that uses monomers as molecular building blocks to make linear self-avoiding random walk polymer chains with control over chain length and monomer pattern/composition.

Monomers are added to the polymer chain end one at a time with force field terms and parameters introduced by new polymer bonds automatically retrieved from a user supplied **Forcefield** object and added to the system containing the growing polymer chain. After each monomer addition, structural relaxation and molecular dynamics simulations are performed (a methodology adapted from the Polymatic algorithm [21]) to relax the geometry in the vicinity of the new polymer bond and generate a new direction of growth for the polymer chain to avoid other growing chains. This series of simulation is carried out using LAMMPS by default, however the random walk application accepts a generalized Simulation object that only requires correct implementation of the run class method explained in Section 2.1.3. A series of snapshots from an example polymerization of polymethyl methacrylate are shown in Fig. 3. Renderings are created using ChemDoodle Web Components [22]. See Examples 4–6 in the *pysimm* source code repository for demonstrations of the **apps.random_walk** module.

4. Impact and conclusions

pysimm is designed to impact two separate but connected groups in the scientific community. The first of which consists of experienced computer programmers with intimate knowledge

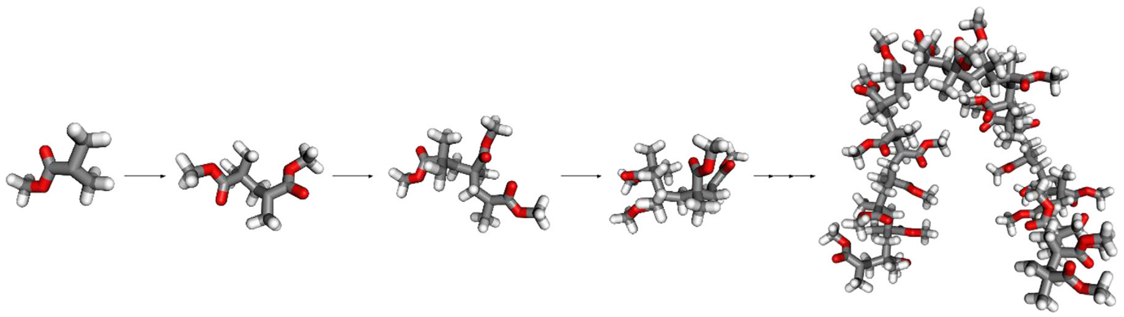

Fig. 3. Polymethyl methacrylate random walk polymer growth illustration. At each iteration a monomer is added to the end of the growing polymer chain until the desired chain length is obtained. A LAMMPS simulation is performed in between monomer additions. Renderings are created using ChemDoodle Web Components [22].

of one or more molecular software packages. This group can be considered *pysimm* developers who use the modular tools defined at the core of *pysimm* to build application work flows to solve scientific problems. These developers have the computer programming capabilities to fill in the blanks where specific functionality for their application is missing from *pysimm*. The second group makes use of these applications to answer scientific inquiries. They do not often develop new applications (but instead create *pysimm* scripts), but may provide feedback to guide application development. This group may use a given application, for example the random walk polymerization application, to study different polymer morphologies, or alter functional groups in monomer repeat units to observe changes in the structure or dynamics of amorphous polymers.

Section 3.1 described how *pysimm* was employed to develop a polymerization algorithm capable of controlling molecular weight and monomer composition. Polymer properties can depend greatly on these characteristics, and is important to have the ability to predictively generate realistic three dimensional structures with control of these variables. However, molecular weight and monomer composition are just two of many different ways in which polymer morphology can vary. *pysimm* provides tools for developers to rapidly create applications capable of generating new morphologies, for example, where a user can input reactivity probabilities to generate a random copolymer or where a user can input a number average molecular weight and a dispersity to generate a random polymer system with the given molecular weight distribution.

pysimm provides a standardized methodology to apply force field parameters to a molecular system based on local bonding environment. The extension of the set of force fields supported in *pysimm* will enable users to easily compare the capabilities of different force fields for their specific needs and guide development of new force fields.

The abstraction from simulation code that *pysimm* provides opens up opportunity to those who have interest in utilizing computer simulations but do not have extensive knowledge with high performance computing. Once configured, *pysimm* allows users to run complex simulations with python code, which is notoriously easy to read and write. Additionally, the abstraction creates a common methodology for running simulations, regardless of which underlying software is being used. In this way, a user that may be proficient at running a simulation using one software can utilize a new technique that may be uniquely available in a different software, using the same centralized syntax via *pysimm*. This can enable, for example, collaboration between groups who may be performing similar types of simulations using different software.

Ultimately, as an open source software package, the future of the *pysimm* package is in the hands of whatever community adopts its usage. The underlying data structures defined at this time of publication were intentionally designed to be vague and flexible, but extensible to accommodate different types of data and accessible to minimize the complexity of resulting scripts and applications. The philosophy of abstracting away intricate details of force field typing and simulation software execution should enable the evolution of *pysimm* in a way end users encounter limited changes in usage from the initial release.

Acknowledgments

The authors would like to acknowledge Alejandro Strachan, Ben Haley, and Chunyu Li at Purdue University for helpful discussions, and the National Science Foundation (NSF) for funding (grant ACI-1613155).

References

[1] Plimpton S. Fast parallel algorithms for short-range molecular dynamics. J Comput Phys 1995;117:1–19.
[2] Case DA, Betz RM, Botello-Smith W, Cerutti DS, Cheatham, III TE, Darden TA, Duke RE, Giese TJ, Gohlke H, Goetz AW, Homeyer N, Izadi S, Janowski P, Kaus J, Kovalenko A, Lee TS, LeGrand S, Li P, Lin C, Luchko T, Luo R, Madej B, Mermelstein D, Merz KM, Monard G, Nguyen H, Nguyen HT, Omelyan I, Onufriev A, Roe DR, Roitberg A, Sagui C, Simmerling CL, Swails J, Walker RC, Wang J, Wolf RM, Wu X, Xiao L, York DM, Kollman PA. Amber 2016. San Francisco: University of California; 2016.
[3] Abraham MJ, Murtola T, Schulz R, Páll S, Smith JC, Hess B, Lindahl E. Gromacs: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. SoftwareX 1–2 2015;19–25. http://dx.doi.org/10.1016/j.softx.2015.06.001.
[4] Brooks BR, Brooks III CL, MacKerell Jr AD, Nilsson L, Petrella RJ, Roux B, Won Y, Archontis G, Bartels C, Boresch S, Caflisch A, Caves L, Cui Q, Dinner AR, Feig M, Fischer S, Gao J, Hodoscek M, Im W, Kuczera K, Lazaridis T, Ma J, Ovchinnikov V, Paci E, Pastor RW, Post CB, Pu JZ, Schaefer M, Tidor B, Venable RM, Woodcock HL, Wu X, Yang W, York D, Karplus M. Charmm: The biomolecular simulation program. J Comput Chem 2009;30(10):1545–614. http://dx.doi.org/10.1002/jcc.21287.
[5] Anderson JA, Lorenz CD, Travesset A. General purpose molecular dynamics simulations fully implemented on graphics processing units. J Comput Phys 2008;227(10):5342–59. http://dx.doi.org/10.1016/j.jcp.2008.01.047.
[6] Eastman P, Friedrichs MS, Chodera JD, Radmer RJ, Bruns CM, Ku JP, Beauchamp KA, Lane TJ, Wang LP, Shukla D, Tye T, Houston M, Stich T, Klein C, Shirts MR, Pande VS. Openmm 4: A reusable, extensible, hardware independent library for high performance molecular simulation. J Chem Theory Comput 2013;9(1):461–9. http://dx.doi.org/10.1021/ct300857j.
[7] Shah JK, Maginn EJ. A general and efficient Monte Carlo method for sampling intramolecular degrees of freedom of branched and cyclic molecules. J Chem Phys 2011;135:134121. http://dx.doi.org/10.1063/1.3644939.
[8] Martínez L, Andrade R, Birgin EG, Martínez JM. PACKMOL: A package for building initial configurations for molecular dynamics simulations. J Comput Chem 2009;30(13):2157–64. http://dx.doi.org/10.1002/jcc.21224.
[9] Klein C, Sallai J, Jones TJ, Iacovella CR, McCabe C, Cummings PT. A Hierarchical, Component Based Approach to Screening Properties of Soft Matter. In: Foundations of molecular modeling and simulation: Select papers from FOMMS 2015. Singapore: Springer; 2016. p. 79–92. http://dx.doi.org/10.1007/978-981-10-1128-3_5.
[10] Landrum G. Rdkit: Open-source cheminformatics. Accessed 8 August 2016.
[11] O'Boyle NM, Banck M, James CA, Morley C, Vandermeersch T, Hutchison GR. Open Babel: An open chemical toolbox. J Chem Inf 2011;3(33):1–14. http://dx.doi.org/10.1186/1758-2946-3-33.

[12] Sarkisov L, Harrison A. Computational structure characterisation tools in application to ordered and disordered porous materials. Mol Simul 2011; 37(15):1248–57. http://dx.doi.org/10.1080/08927022.2011.592832.

[13] Humphrey W, Dalke A, Schulten K. VMD: Visual Molecular Dynamics. J Mol Graph 1996;14:33–8. http://dx.doi.org/10.1016/0263-7855(96)00018-5.

[14] Schrödinger, LLC, The PyMOL molecular graphics system, version 1.8, 2015.

[15] McGibbon RT, Beauchamp KA, Harrigan MP, Klein C, Swails JM, Hernández CX, Schwantes CR, Wang LP, Lane TJ, Pande VS. Mdtraj: A modern open library for the analysis of molecular dynamics trajectories. Biophys J 2015;109(8): 1528–32. http://dx.doi.org/10.1016/j.bpj.2015.08.015.

[16] Haley BP, Wilson N, Li C, Arguelles A, Jaramillo E, Strachan A. Polymer modeler, Aug 2016. http://dx.doi.org/10.4231/D3X921K69, URL https://nanohub.org/resources/9230.

[17] Abbott L, Colina C. Polymatic: A simulated polymerization algorithm, Mar 2016. URL https://nanohub.org/resources/17278.

[18] Degiacomi MT, Erastova V, Wilson MR. Easy creation of polymeric systems for molecular dynamics with assemble!. Comput Phys Comm 2016;202:304–9. http://dx.doi.org/10.1016/j.cpc.2015.12.026.

[19] Fortunato M, Abbott L, Hart KE, Colina C. nusimm: nanohub user simulation interface for molecular modeling (2016), http://dx.doi.org/10.4231/D3ZP3W18X, URL https://nanohub.org/resources/22366.

[20] Madhavan K, Zentner L, Farnsworth V, Shivarajapura S, Zentner M, Denny N, Klimeck G. nanohub.org: Cloud-based services for nanoscale modeling, simulation, and education. Nanotechnology Rev 2013;2(1):107–17. http://dx.doi.org/10.1515/ntrev-2012-0043.

[21] Abbott LJ, Hart KE, Colina CM. Polymatic: a generalized simulated polymerization algorithm for amorphous polymers. Theoret Chem Acc 2013;132(3):1–19. http://dx.doi.org/10.1007/s00214-013-1334-z.

[22] iChemLabs LLC. Chemdoodle web components. Accessed 8 August 2016.