lammpsio: Transparent and reproducible handling of LAMMPS particle data in Python

C. Levi Petix[1], **Mayukh Kundu**[1], **and Michael P. Howard**[1]

1 Department of Chemical Engineering, Auburn University, Auburn, AL 36849

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Rachel Kurchin

Reviewers:

- @mhlbapat
- @Ladme

Submitted: 19 July 2025
Published: unpublished

Summary

lammpsio provides a Python interface for reading and writing particle data in LAMMPS (A. P. Thompson et al., 2022) data and dump files. It aims to simplify the creation and parsing of these LAMMPS inputs and outputs, enabling LAMMPS users to more easily set up their simulations and to analyze their results with other Python tools. lammpsio also interconverts LAMMPS data and dump files with the GSD file used by HOOMD-blue (Anderson et al., 2020), another simulation package with an overlapping user base.

Statement of need

LAMMPS is a popular particle-based simulation package used across many scientific fields. Two of its important file formats are the data file, which defines the simulation state, and the dump file, which records particle trajectories. These files are text-based and can be tedious to parse or manipulate. Python tools such as Atomman and Pizza.py support reading and writing LAMMPS particle-data files, but their interfaces can be complex. Other established simulation analysis software such as MDAnalysis (Gowers et al., 2016; Michaud-Agrawal et al., 2011), MDTraj (McGibbon et al., 2015), and OVITO (Stukowski, 2009) may only provide a subset of these features. As a result, researchers may rely on their own codes, but private codes are vulnerable to unnoticed errors and hinder reproducibility.

lammpsio addresses this need by providing an object-oriented Python interface for reading and writing LAMMPS data and dump files while adhering to the TRUE (transparent, reproducible, usable by others, and extensible) principles of scientific software development (M. W. Thompson et al., 2020). It allows users to create a particle configuration (simulation input) and parse a trajectory (simulation output) in a consistent manner. lammpsio additionally supports interconversion with the GSD file used by HOOMD-blue, enabling sharing of particle configurations between simulation packages. This functionality is leveraged, for example, in our relative-entropy minimization software (Sreenivasan et al., 2024) to support runtime selection of LAMMPS or HOOMD-blue as a simulation engine. Taken together, the features of lammpsio help meet needs for creating reproducible, interoperable, and maintainable simulation workflows for LAMMPS.

Design and implementation

lammpsio is organized around a central Snapshot object representing a single configuration of particles. Two other objects, the DataFile and DumpFile, provide read–write functionality for Snapshots in the corresponding LAMMPS file formats.

The Snapshot stores information about a certain number of particles at a given time. The particles occupy a volume prescribed by a Box object, defined using the LAMMPS convention for restricted triclinic boxes. Each Snapshot stores per-particle data, such as position or velocity, in NumPy arrays. Topology data, defining the connectivity between subsets of particles (bonds, angles, dihedrals, and impropers) (Allen & Tildesley, 2017), is also stored in the Snapshot using correspondingly named objects that manage per-connection data as NumPy arrays. Importantly, lammpsio allocates its per-particle and per-connection arrays with the correct shape and type only when they are first accessed. This design prevents mismatches in array attributes and can reduce memory requirements. The use of NumPy arrays additionally allows easy integration with the scientific Python ecosystem. The currently supported particle and topology information accommodates many common use cases; however, additional data can be attached to the Snapshot by subclassing or patching.

The DataFile reads and writes LAMMPS data files, which are text-based files that store a single particle configuration. A LAMMPS data file is organized into a header section, which typically has high-level information about the simulation state, and a body containing multiple sections, many of which define per-particle and per-connection information for the configuration. Sections have standardized formats, but the Atoms section may have a different format depending on the atom style in the simulation. The DataFile's read method parses a data file into a Snapshot by tokenizing the headers and sections; the style of the Atoms section is inferred from a comment if present and may be manually specified otherwise. The DataFile's create method writes a data file containing only explicitly set information in the Snapshot and documented defaults for any data that is required but not explicitly set. Users can specify an atom style or allow lammpsio to infer the minimal style needed to represent their particles.

The DumpFile reads and writes LAMMPS dump files, which are text-based files that store a trajectory (sequence of particle configurations). A LAMMPS dump file has a significantly more flexible format than a data file. It is organized into a sequence of items, including timestep, number of atoms (particles), box bounds, and atoms. The atoms item, in particular, contains one or more columns representing per-particle properties and can be configured in a variety of ways in LAMMPS. Additionally, dump files may be optionally written in binary or compressed formats. The DumpFile's read method uses Python iteration to load one Snapshot at a time for each configuration in a dump file. It attempts to infer the schema for the atoms item from a comment, but the schema may be manually specified. Users can also provide a Snapshot from which to copy information that cannot be stored in the dump file (e.g., topology) or was not written in the dump file to reduce file size (e.g., static per-particle data). The DumpFile's create method generates a LAMMPS dump file from a sequence of Snapshots according to a user-defined schema for the atoms item. lammpsio currently supports gzip and Zstandard compression for both reading and writing.

Discussion

By providing a lightweight and intuitive object-oriented interface, lammpsio streamlines workflows on LAMMPS input and output particle data. With only a few lines of code, users are able to feed their simulation data to tools that do not natively support LAMMPS files but do use NumPy arrays. For example, lammpsio enables users to directly interface their data with freud (Ramasubramani et al., 2020), a simulation analysis tool that makes minimal assumptions about its input format, as demonstrated in an example in our documentation. In a similar way, lammpsio data can also be passed to popular machine-learning toolkits with Python interfaces. Additionally, lammpsio can be used as a file converter for LAMMPS dump files, both from other formats such as HOOMD-blue's GSD file and between LAMMPS dump files with different schemas. This functionality can be particularly useful when working with legacy analysis tools that assume a LAMMPS dump file with a specific schema. Future development of lammpsio is planned to add support for type labels, general triclinic boxes, and processing other information that can be stored in data and dump files.

Availability

lammpsio is freely available under the BSD 3-Clause License on GitHub and is publicly distributed on PyPI and conda-forge. lammpsio has continuous integration testing using GitHub Actions for all currently supported combinations of Python and stable LAMMPS releases. We test both the core Python functionality of lammpsio and round-trip file compatibility with LAMMPS. For installation instructions and Python API documentation, please visit the publicly hosted documentation. For examples of how to use lammpsio, please visit the tutorials

Acknowledgements

We thank Dr. Jing Chen from the Molecular Sciences Software Institute for her valuable feedback and helpful discussions. C.L.P. was supported by a fellowship from The Molecular Sciences Software Institute under National Science Foundation Award No. 2136142. This material is based upon work supported by the National Science Foundation under Award No. 2310724.

Conflict of interest statement

The authors declare the absence of any conflicts of interest: No author has any financial, personal, professional, or other relationship that affect our objectivity toward this work.

References

Allen, M. P., & Tildesley, D. J. (2017). *Computer simulation of liquids* (2nd ed.). Oxford University Press.

Anderson, J. A., Glaser, J., & Glotzer, S. C. (2020). HOOMD-blue: A python package for high-performance molecular dynamics and hard particle monte carlo simulations. *Computational Materials Science*, *173*, 109363. https://doi.org/10.1016/j.commatsci.2019.109363

Gowers, R. J., Linke, M., Barnoud, J., Reddy, T. J. E., Melo, M. N., Seyler, S. L., Domański, J., Dotson, D. L., Buchoux, S., Kenney, I. M., & Beckstein, O. (2016). MDAnalysis: A python package for the rapid analysis of molecular dynamics simulations. *Proceedings of the 15th Python in Science Conference*, 98–105. https://doi.org/10.25080/Majora-629e541a-00e

McGibbon, R. T., Beauchamp, K. A., Harrigan, M. P., Klein, C., Swails, J. M., Hernández, C. X., Schwantes, C. R., Wang, L.-P., Lane, T. J., & Pande, V. S. (2015). MDTraj: A modern open library for the analysis of molecular dynamics trajectories. *Biophysical Journal*, *109*(8), 1528–1532. https://doi.org/10.1016/j.bpj.2015.08.015

Michaud-Agrawal, N., Denning, E. J., Woolf, T. B., & Beckstein, O. (2011). MDAnalysis: A toolkit for the analysis of molecular dynamics simulations. *Journal of Computational Chemistry*, *32*(10), 2319–2327. https://doi.org/10.1002/jcc.21787

Ramasubramani, V., Dice, B. D., Harper, E. S., Spellings, M. P., Anderson, J. A., & Glotzer, S. C. (2020). Freud: A software suite for high throughput analysis of particle simulation data. *Computer Physics Communications*, *254*, 107275. https://doi.org/10.1016/j.cpc.2020.107275

Sreenivasan, A. N., Petix, C. L., Sherman, Z. M., & Howard, M. P. (2024). Relentless: Transparent, reproducible molecular dynamics simulations for optimization. *The Journal of Chemical Physics*, *161*(21), 212502. https://doi.org/10.1063/5.0233683

Stukowski, A. (2009). Visualization and analysis of atomistic simulation data with OVITO–the open visualization tool. *Modelling and Simulation in Materials Science and Engineering*, *18*(1), 015012. https://doi.org/10.1088/0965-0393/18/1/015012

Thompson, A. P., Aktulga, H. M., Berger, R., Bolintineanu, D. S., Brown, W. M., Crozier, P. S., Veld, P. J. in 't, Kohlmeyer, A., Moore, S. G., Nguyen, T. D., Shan, R., Stevens, M. J., Tranchida, J., Trott, C., & Plimpton, S. J. (2022). LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. *Computer Physics Communications*, *271*, 108171. https://doi.org/10.1016/j.cpc.2021.108171

Thompson, M. W., Gilmer, J. B., Matsumoto, R. A., Quach, C. D., Shamaprasad, P., Yang, A. H., Iacovella, C. R., Cabe, C. M., & Cummings, P. T. (2020). Towards molecular simulations that are transparent, reproducible, usable by others, and extensible (TRUE). *Molecular Physics*, *118*(9–10), e1742938. https://doi.org/10.1080/00268976.2020.1742938