

# IOData: A Python Library for Reading, Writing, and Converting Computational Chemistry File Formats and Generating Input Files

Toon Verstraelen<sup>1\*</sup> | William Adams<sup>2</sup> | Leila Pujal<sup>3</sup> |  
Alireza Tehrani<sup>3</sup> | Braden D. Kelly<sup>2</sup> | Luis Macaya<sup>5</sup> |  
Fanwang Meng<sup>2</sup> | Michael Richer<sup>2</sup> | Raymundo  
Hernández-Esparza<sup>2</sup> | Xiaotian Derrick Yang<sup>2, 4</sup> |  
Matthew Chan<sup>2</sup> | Taewon David Kim<sup>2</sup> | Maarten  
Cools-Ceuppens<sup>1</sup> | Valerii Chuiko<sup>2,6</sup> | Esteban  
Vöhringer-Martinez<sup>5\*</sup> | Paul W. Ayers<sup>2\*</sup> | Farnaz  
Heidar-Zadeh<sup>1, 3\*</sup>

<sup>1</sup>Center for Molecular Modeling (CMM), Ghent University, Technologiepark-Zwijnaarde 46, B-9052, Zwijnaarde, Belgium

<sup>2</sup>Department of Chemistry and Chemical Biology, McMaster University, Hamilton, Ontario, L8S-4L8, Canada

<sup>3</sup>Department of Chemistry, Queen's University, Kingston, Ontario, K7L-3N6, Canada

<sup>4</sup>Laboratoire de Chimie Théorique, Sorbonne Université, Paris, 75052, France

<sup>5</sup>Departamento de Físico Química, Facultad de Ciencias Químicas, Universidad de Concepción, 4070371 Concepción, Chile

<sup>6</sup>Faculty of Physics; Taras Shevchenko National University of Kyiv, 01601 Kyiv, Ukraine

## Correspondence

Email: Toon.Verstraelen@ugent.be;

evohringer@udec.cl; ayers@mcmaster.ca;

farnaz.heidarzadeh@queensu.ca

IOData is a free and open-source Python library for parsing, storing, and converting various file formats commonly used by quantum chemistry, molecular dynamics, and plane-wave density-functional-theory software programs. In addition, IOData supports a flexible framework for generating input files for various software packages. While designed and released for stand-alone use, its original purpose was to facilitate the interoperability of various modules in the HORTON and ChemTools software packages with external (third-party) molecular quantum chemistry and solid-state density-functional-theory packages. IOData is designed to be easy to use, maintain, and extend; this is why we wrote IOData in Python and adopted many principles of modern software development, including comprehensive documentation, extensive

testing, continuous integration/delivery protocols, and package management. This article is the official release note of the IOData library.

#### KEYWORDS

quantum chemistry, molecular mechanics, computational chemistry, file format conversion, data parsing, basis set conversion, input file generation, JSON schema, chemistry software development, theoretical chemistry Python library

## 1 | WHAT IS IODATA?

IOData is a free and open-source Python 3 library for reading and writing formatted (non-binary) files generated by a broad array of molecular quantum chemistry, molecular dynamics, and plane-wave density-functional-theory programs. IOData is primarily intended as a utility for (a) parsing molecular/periodic geometry, wavefunction, and trajectory data from various file formats, (b) storing this data in a user-friendly object, so that it can be used in external Python modules, (c) dumping geometry, wavefunction, and trajectory data in a variety of standard file formats, hence (d) converting one standard file format to another, and (e) generating input files for various computational chemistry software packages. Section 5 details the file formats and input file generation currently supported by IOData; however, we designed the IOData framework to be not only easy to use but also easy to extend to new data formats and software.

## 2 | HISTORY

During the course of our scientific research, and especially our development of the HORTON[1] and ChemTools[2] software packages, we needed a utility that allowed us to parse the output files of various electronic structure theory programs and dump data from our software packages using standard file formats. Some of this functionality existed in HORTON 2.x [1], but in the process of writing HORTON 3, we decided to split IOData (as well as much of the other functionality of HORTON) into stand-alone libraries to better support modularity, maintenance, and ease of use.

## 3 | ABOUT IODATA

IOData is, and always will be, a free and open-source library distributed under the GNU General Public License. The IOData source code is maintained on the GitHub platform; see <https://github.com/theochem/iodata>, and its documentation is hosted on Read the Docs; see <https://iodata.readthedocs.io/en/latest/index.html>. We strive to ensure that the IOData source code and website itself is comprehensively documented, including useful tests, scripts, and examples. As that documentation is maintained with the software, providing detailed (and eventually outdated) release notes here seems unwise. Instead, we will briefly list the distinguishing features and key capabilities of IOData in section 5 and demonstrate them in section 6.

## 4 | WHY IODATA?

Before starting to work on `IOData`, we considered using other free and open-source utilities for parsing and converting file formats like `OpenBabel`[3] (mostly geared toward cheminformatics), `RDKit`[4] (limited wavefunction capabilities), `cclib`[5] (limited capabilities for dumping output files), `ASE`[6] (limited features for molecular systems), `MultiWFN`[7] (difficult-to-extend Fortran 90 source code), `MDTraj`[8] and `MDAnalysis`[9, 10] (both geared towards trajectory formats and analysis). However, none of these achieved our criteria of being easy to use and modify, readable, stand-alone, modular, and written in Python. Most critically, most other software packages are limited in their ability to parse molecular wavefunction information and extremely limited in their ability to write popular quantum chemistry output file formats.

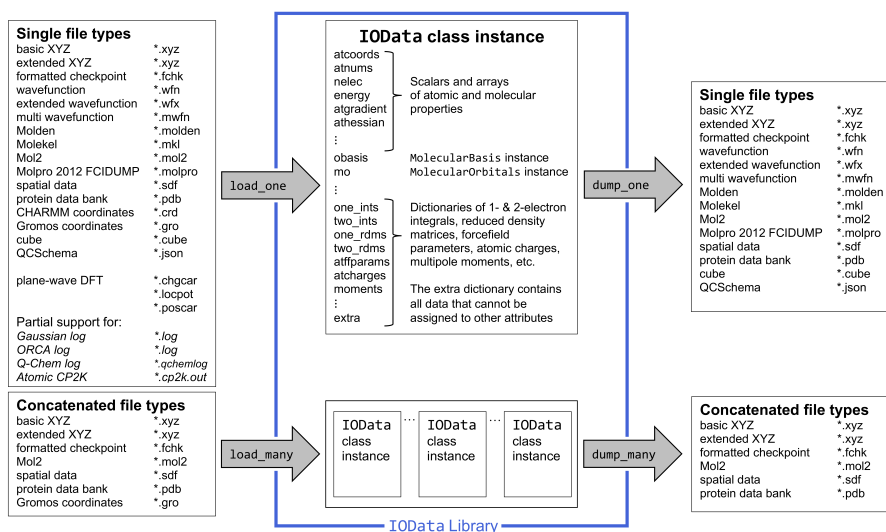
Anticipating that other researchers may have similar needs that can be met by directly using `IOData` or by building on top of it, we decided to release `IOData` as a free and open-source library. Nonetheless, we wish to acknowledge the utility of these other programs, which we also use in our research. `OpenBabel`[3] is extremely useful for cheminformatics tasks; `RDKit` is an excellent tool for generating molecular structures and conformations; `cclib` has refined capabilities for parsing an abundance of information from quantum chemistry output/log file formats; `ASE` supports an abundance of solid-state electronic structure programs; `MDTraj` and `MDAnalysis` support an impressive number of trajectory formats; `MultiWFN` has comprehensive post-processing capabilities and supports many of the most important formats supported by `IOData`, but does not produce an internal data structure (e.g., a dictionary or a class) that is easily used in other programs. While `IOData` overlaps with these tools in some respects, its emphasis on reading and writing molecular quantum chemistry data file formats, generating input files, and usability as a library within other programs is unique.

In our view, the distinguishing characteristics of `IOData` are its modularity, its flexibility, and its ease of use. We achieve this by designing a unifying framework in which data from various file formats can be loaded, stored in a versatile container class, and dumped in a desired file format. We provide comprehensive documentation and rigorously adhere to high coding standards, including comprehensive testing of code correctness, quality, and readability. While `IOData` is primarily intended to be used as a Python library and its API was designed to facilitate interoperability with other Python packages, many operations can be performed through its command line tools.

## 5 | FEATURES OF IODATA

We display various features of `IOData` by first discussing file formats that it can currently analyze. Specifically,

- The file formats (and their commonly used extensions) that can be both loaded and dumped by `IOData` include: basic XYZ Cartesian coordinate files (\*.xyz), extended XYZ files containing additional atomic properties (\*.extxyz), Gaussian[11] formatted checkpoint files (\*.fchk), wavefunction files (\*.wfn), extended wavefunction files (\*.wfx), MultiWFN files (\*.mwfn), Molden[12] files (\*.molden), Molekel files (\*.mkl), Mol2 files (\*.mol2), Molpro[13] 2012 FCIDUMP integral files (\*.molpro), spatial data files (\*.sdf), protein data bank files (\*.pdb), QCSchema[14] files (\*.json), and cube files (\*.cube). Most often, these file formats each contain information on a single chemical compound, and are commonly used in exchanging information between various software packages (e.g. for post-processing). Several of these file formats are often used to store information on multiple compounds in a single file (e.g. concatenation of a collection of molecules/frames in a single file with a single format). In this regard, `IOData` also supports load and dump functionality for concatenated \*.xyz, \*.extxyz, \*.fchk, \*.mol2, \*.sdf, and \*.pdb file formats.



**FIGURE 1** Schematic representation of IOData library.

- The file formats (and their commonly used extension) that can be only loaded by IOData include certain files from plane-wave density-functional theory codes including VASP (\*.chgcar, \*.locpot, \*.poscar), molecular dynamics packages like Gromacs (\*.gro) and CHARMM (\*.crd) coordinate files, and GAMESS[15] punch files (\*.dat). IOData can also partially parse and store data from log files of several quantum chemistry software packages including Gaussian[11] log files (\*.log), ORCA[16] log files (\*.out), Q-Chem[17] log files (\*.qchemlog), and atomic CP2K[18] log files (\*.cp2k.out). If the user is interested in a specific information printed in these log file that is not currently loaded by IOData, the existing parser can be easily extended.
- To support a broad range of external software, IOData supports, and interconverts between, arbitrary basis-set conventions, including basis-function order, sign (of the spherical harmonics), and normalization ( $L^1$ -normalized,  $L^2$ -normalized, and unnormalized). IOData also supports basis set conversion (e.g. Cartesian to/from spherical, primitive to/from contracted).
- IOData can be used to generate input files for various quantum chemistry and molecular dynamics software packages, including Gaussian[11] and ORCA[16]. Users have the option of providing an input template for specific use cases or using pre-defined basic templates. The input file writer can be easily extended for use with other software packages.

Considering the large number of file formats commonly used in computational chemistry, the IOData library is particularly useful as it provides a user-friendly platform to load the information contained in various file formats into an easy-to-use object and to convert one file format to the other. IOData's file-conversion capability is especially useful because software packages usually work with specific file formats. Obviously, the XYZ and cube file formats cannot be used to generate file formats containing wavefunction information, however, any of the above mentioned formats can be loaded by IOData (like fchk) and converted to any other format (like XYZ, wfn, molden, etc.).

As depicted in Figure 1, the IOData library stores the loaded data as attributes of the IOData class. These are mostly numerical data, arrays, or strings; if available, the information related to molecular basis set and orbitals are

stored as the 'obasis' and 'mo' attributes, which are instances of the `MolecularBasis` class and `MolecularOrbitals` class, respectively. Occasionally, file formats include information on 1- and 2-electron integrals, reduced density matrices, multipole moments, atomic charges, and/or force field parameters; these are each stored as dictionaries. Any other additional information that does not fit into a pre-existing field is stored in the 'extra' dictionary attribute of the `IOData` class; items in 'extra' may become standard attributes in future releases.

We aim to ensure that these input files can be parsed robustly, so that even when popular software packages fail to follow the technical specifications of a file format (usually because the file format was not strictly or publicly defined) the error is automatically diagnosed and corrected if possible. For example, many programs read and generate \*.wfn files that do not follow the original format statements from Richard Bader's AIMPAC program, and no formal file format specifications for \*.wfn seems ever to have been published. To remedy this, we looked at four \*.wfn formats from various open-source projects and wrote a \*.wfn format specification that was as flexible as possible, yet still supported all four versions. The file format specification is provided as part of the `IOData` documentation, and hopefully will prevent rogue \*.wfn file formats in the future. Another example is that different programs (PSI4[19], ORCA[16], Turbomole[20], Molden[12], Molekel, Q-Chem[17], etc.) generate \*.molden files with different basis-set normalization, sign, and ordering conventions. We automatically detect these conventions by evaluating the atomic-orbital overlap matrix, and then store the basis set in a flexible and universal way. When writing a \*.molden file, we always use the same normalization convention that the Molden program itself uses.

Among the recently proposed file formats, `IOData` supports loading and dumping `QCSchema`[14], a program-agnostic JSON schema that serves as both an input and output file format. Unfortunately, the implementation of the `QCSchema` in MolSSI projects such as `QCElemental`[21] has diverged from the original `QCSchema` specification[14]. The version implemented in `QCElemental` has transformed from a single, extended schema to four different sub-schema components: Molecule, Input, Output, and Basis. This reinvention of `QCSchema` is reasonable, but only exists as a Python-based partial implementation in `QCElemental`, not a true JSON schema specification usable by other potential adopters. `IOData` implements a backward-compatible and complete version of `QCSchema` by adopting many of the changes found in `QCElemental` and the development branch of `QCSchema` while also providing a genuine JSON schema and extensive documentation specifying each of the four sub-schema.

`IOData` uses an extensive automatic-testing framework to verify that the code behaves correctly, both when introducing code changes and when making new installations on various operating systems. Currently, `IOData` uses `pytest` framework and the unit test coverage feature of `coverage` to ensure a significant portion of our code is well tested; At present, 96% of the code base is tested. In addition, `IOData` uses an extensive set of quality-assurance tools (`pylint`, `autopep8`, `pycodestyle`, `pydocstyle`, etc.) to ensure that the code quality is not degraded by new contributions and the code remains readable and well-documented. The testing and quality-assurance workflows are automatically run by our continuous integration system every time a code change is proposed to our GitHub repository. Currently, we use `travis.io` for continuous integration, delivery, and deployment. If `travis.io` reports that the unit tests or quality-assurance tools is failed, the proposed changes are automatically rejected. When our code is ready for release, we Git tag a commit with a special keyword and `travis.io` builds and tests a release with the `Conda` and `Pip` package managers. This allows users to simply install `IOData` and all its binary dependencies in one command. Developers can also publish new releases to users in one command thanks to the automation provided by this workflow.

## 6 | EXAMPLES

For the most updated documentation and examples on how to use IOData, please refer to the IOData website. Here, we showcase several ways IOData can be used and incorporated into various workflows. Please note that these examples are based on version 1.0 of IOData, and the user might need to modify them if using future major releases of the IOData library. Within minor and bug-fix releases, backward compatibility is guaranteed.

### 6.1 | Using IOData as a Python library

**Loading File Formats:** The primary use case for IOData is to parse files of various formats and make the loaded data easily accessible to the user. This is mainly facilitated through the `load_one` and `load_many` functions, which return an IOData instance and a list of IOData instances, respectively, in which the loaded data is stored. For example,

```
1 # Loading Gaussian formatted checkpoint (fchk) file format
2 from iodata import load_one
3
4 data = load_one("water.fchk")      # instance of IOData class
5
6 # Examples of information that can be retrieved from the IOData instance
7 print(data.atnums)                # atomic numbers
8 print(data.atcoords)              # atomic coordinates
9 print(data.nelec)                 # number of electrons
10 print(data.charge)                # charge of molecule
11 print(data.obasis)                # instance of MolecularBasis class
12 print(data.obasis.nbasis)         # number of basis functions
13 print(data.mo)                   # instance of MolecularOrbitals class
14 print(data.mo.energies)           # molecular orbital energies
15 print(data.atcharges["mulliken"]) # Mulliken atomic charges
```

The format of the file specified on line 4 is derived from the file extension, however, the user can override the default file format detected using the optional `fmt` argument. The above example displays only a few IOData attributes; the data available from each file format is clearly tabulated on the IOData website. It is important to note that the wavefunction information is stored in an IOData instance in a manner designed to simplify calculations. For example, one can simply compute the spin-summed density matrix, for both restricted and unrestricted Slater determinants, using,

```
1 # Computing density matrix in the molecular orbital basis in 2 different ways
2 import numpy as np
3
4 dm = np.dot(data.mo.coeffs * data.mo.occs, data.mo.coeffs.T) # using dot product
```

```
5 dm = np.einsum("ij,i,ij", data.mo.coeffs, data.mo.occs, data.mo.coeffs) # using einsum
```

Similarly, the `load_many` function can be used to load concatenated file formats like databases (\*.sdf, \*.mol2, \*.xyz, or \*.extxyz) or successive conformations of molecules from molecular dynamics trajectories or reaction paths (\*.pdb, \*.fchk, \*.xyz, or \*.extxyz), stored in a single file. For example,

```
1 # Loading multiple molecules/frames contained in one XYZ file
2 from iodata import load_many
3
4 data = load_many("database.xyz") # list of instances of IOData class
5
6 # Examples of information that can be retrieved
7 print(len(data)) # number of molecules/frames in the database
8 print(data[0]) # IOData instance of first molecule/frame
9 print(data[0].atcoords) # atomic coordinates of first molecule/frame
10 print(data[-1].atnums) # atomic numbers of last molecule/frame
```

Because many file formats are not strictly defined, the `IOData` library must be extremely flexible. For example, for the XYZ file format, `IOData` accepts atomic numbers, atomic symbols, or mixtures of both as atom specifications in the first column of the Coordinates section. `IOData` also allows the user to provide their own format specifications. For example, the many versions of the extended XYZ file format store various atomic properties by appending columns to the basic XYZ format. `IOData` can be instructed to load this additional data through the optional `atom_columns` argument of the `load_one` or `load_many` functions.

**Writing File Formats:** The information loaded from one file format can be used to write output in a different file format, if the required data is available. This is made possible through the `dump_one` and `dump_many` functions, which take an `IOData` instance and sequence of `IOData` instances, respectively, and write out the specified file format. For example,

```
1 # Converting Gaussian formatted checkpoint (fchk) file format to Molden and XYZ file formats
2 from iodata import load_one, dump_one
3
4 data = load_one("water.fchk") # instance of IOData class
5 dump_one(data, "water.molden") # write out a Molden file
6 dump_one(data, "water.xyz", fmt="xyz") # write out an XYZ file
```

As mentioned previously, the format of a dumped file is derived from the filename extension; however, the user can override this default by specifying the optional `fmt` argument. Similarly, having a sequence of `IOData` instances, the `dump_many` function can be used to generate a single file containing the molecules/frames in the user-specified file format. Similarly, to generate an extended XYZ file, additional columns can be specified through the optional `atom_columns` argument of the `dump_one` or `dump_many` functions.

**Storing Data:** An instance of the `IOData` class can be constructed by the user, for example, in the process of performing a quantum chemistry calculation. This is helpful for both using the stored information internally (e.g. when using the HORTON 3 and ChemTools[2] software packages) or writing it out in a specific file format. For example,

```
1 # Storing information in IOData and dump file formats
2 import numpy as np
3 from iodata import IOData, dump_one
4
5 # Make an instance of IOData class with atomic numbers and coordinates (in Borh)
6 data = IOData(title="water")
7 data.atnums = np.array([8, 1, 1])
8 data.coordinates = np.array([[1.48123726, -0.93019123, 0.0], [0.0, 0.11720081, 0.0],
9                             [-1.48123726, -0.93019123, 0.0]])
10
11 # Write out an XYZ file (with atomic coordinates in Angstrom)
12 dump_one(data, "water.xyz")
```

**Writing Input Files:** Input files for common software packages can be generated using either built-in or user-defined templates. The basic input writing functionality requires only an `IOData` instance with sufficient data for the input file (i.e. level of theory, basis set name, etc.). For example,

```
1 # Writing Gaussian input files from IOData instance
2 import numpy as np
3 from iodata import IOData, write_input
4
5 # Get an instance of IOData class by loading an XYZ file
6 data = load_one("water.xyz")
7 # Change the template's default level of theory (HF) & basis set (STO-3G)
8 data.lot = "B3LYP"
9 data.obasis_name = "cc-pVTZ"
10
11 # Generate Gaussian & ORCA input files using the default template, charge (0), & multiplicity (1)
12 write_input(data, "water.com", fmt="gaussian")
13 write_input(data, "water.in", fmt="orca", template=None)
```

The `write_input` function accepts a user-defined input template through `template` argument which is a text file containing Python string substitution markers and `IOData` attribute names, like `${lot}` for level of theory, `${obasis_name}` for basis set name, etc..



## 6.2 | Using IOData as a command-line tool

`iodata-convert` is the command-line file-convesion interface accessible upon installing the IOData library. For example,

```
1 # Convert Gaussian formatted checkpoint to a Molden file
2 iodata-convert water.fchk water.molden
```

The input and output file formats are derived from the file extensions, however, they can be specified through optional command-line arguments. To obtain more information on how to customize this utility, please refer to its help message,

```
1 iodata-convert --help
```

## 7 | FREQUENTLY ASKED QUESTIONS

**Who is IOData for?** We intend IOData to be primarily used by computational chemists and researchers developing quantum chemistry software, especially post-processing and visualization tools. Our goal is for IOData to be easy to use for novice programmers and computational chemists, yet capable of supporting advanced quantum chemistry workflows. Achieving these goals requires adherence to software design best practices, so extending or contributing to IOData requires intermediate-level programming ability (i.e. knowledge of decorators, type hints, automatic testing, GitHub, continuous integration and automatic release, etc.). The command-line tools in IOData can be used by anyone who has passing familiarity with Linux shell commands.

**What is the mission of IOData?** We wish to serve the community of quantum chemistry users and developers by removing the barrier imposed by the plethora of (sometimes obscurely defined and/or documented) file formats. This is clearly a Sisyphean task, but we sincerely hope that others in the community will join our efforts and, if possible, eventually agree on a universal file format.

**What does IOData do?** As elaborated in sections 5 and 6, IOData currently parses various file formats from third-party software packages and makes the data available to Python programs or writes out the data into a different format, and thus supports file format conversion. The latter can also be achieved through command-line functionality of IOData. In addition, IOData generates input files for various quantum chemistry or molecular dynamics software packages using basic or user-defined input file templates.

**What is the future direction of IOData?** We aim to support parsing and writing additional file formats, and also to extend our support for writing input files and enable direct integration with QCEngine[21]. Our current to-do list for IOData can be found on its GitHub's issue page, and we anticipate that most items will be resolved by the end of 2020. Within our software development efforts, the IOData will eventually serve as the input/output module of the HORTON3 and ChemTools[2] software packages.

**How do I install IOData?** The IOData library can be installed from its source code available on GitHub or through *pip* and *conda* package-management systems. Maintaining the source code on GitHub allows users to access our latest development, even before it is officially released. For the most updated instructions on how to install IOData, please refer to the IOData website.

**Can I contribute to IOData?** Yes! We welcome and support new contributions in accordance with our Code of Conduct and Contributing Guidelines. For the most up-to-date instructions on how to contribute, please refer to the IOData website.

## 8 | SUMMARY

The purpose of this brief paper is to present the IOData module of HORTON 3, which parses, stores, and dumps a variety of file formats. The most unique capabilities of IOData are its robustness (e.g., its ability to read \*.molden files with nonstandard basis-set conventions; its ability to read \*.wfn files with strange fixed-formatting issues; the ability to read QCSchema with incompatible formats), its utilities to read and parse files for single and multiple structures (e.g. trajectories, reaction pathways, potential energy surface scans), its scope (including file formats typically used in Gaussian-basis-set-based molecular chemistry software, plane-wave density-functional-theory codes, and molecular dynamics calculations), its ability to output a variety of useful file formats, its meticulous documentation and testing protocols, its ease-of-use and extensibility, and its command-line tools. While we will continually improve IOData, we believe its present functionality already has significant benefits to the broader community of theoretical and computational chemists and welcome their comments and contributions.

## 9 | RESEARCH DATA STATEMENT

IOData is free and open-source software; the examples used herein (or embellishments thereof) are disseminated with the package, which is available on GitHub; <https://github.com/theochem/iodata>.

## Acknowledgements

We wish to acknowledge various refinements to the IOData library from Steven Vandenbrande, Jennifer Garner, Thomas Pigeon, Stijn Fias, Ali Malek, and the HORTON development team. T.V. acknowledges the Foundation of Scientific Research-Flanders (FWO) and the Research Board of Ghent University (BOF) for their financial support. L.M.Z. and E.V.M. acknowledge financial support by CONICYT/FONDECYT/REGULAR/FOLIO 1200369, PCI CONICYT INSTITUTO MAX PLANCK FOR TERRESTRIAL MICROBIOLOGY MARBURG MPG190003 and the Max-Planck Society. P.W.A. acknowledges Natural Sciences and Engineering Research Council (NSERC) of Canada, the Canada Research Chairs, Compute Canada, and CANARIE for financial and computational support. F.H.Z. acknowledges financial support from FWO, NSERC and Queen's University Research Initiation Grant.

## Conflict of Interest

There is no conflict of interest.

## references

- [1] Verstraelen T, Tecmer P, Heidar-Zadeh F, González-Espinoza CE, Chan M, Kim TD, et al. HORTON 2.1.1; 2017. <https://theochem.github.com/horton/>.
- [2] Heidar-Zadeh F, Richer M, Fias S, Miranda-Quintana RA, Chan M, Franco-Perez M, et al. An explicit approach to conceptual density functional theory descriptors of arbitrary order. *Chemical Physics Letters* 2016;660:307–312.
- [3] O'Boyle NM, Banck M, James CA, Morley C, Vandermeersch T, Hutchison GR. Open Babel: An open chemical toolbox. *Journal of Cheminformatics* 2011;3(1):33.
- [4] Landrum G, RDKit: Open-source cheminformatics; <https://www.rdkit.org/>.
- [5] O'boyle NM, Tenderholt AL, Langner KM. cclib: A library for package-independent computational chemistry algorithms. *Journal of Computational Chemistry* 2008;29(5):839–845. <https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.20823>.
- [6] Larsen AH, Mortensen JJ, Blomqvist J, Castelli IE, Christensen R, Duřak M, et al. The atomic simulation environment—a Python library for working with atoms. *Journal of Physics: Condensed Matter* 2017;29(27):273002. <http://stacks.iop.org/0953-8984/29/i=27/a=273002>.
- [7] Lu T, Chen F. Multiwfn: A multifunctional wavefunction analyzer. *Journal of Computational Chemistry* 2012;33(5):580–592. <https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.22885>.
- [8] McGibbon RT, Beauchamp KA, Harrigan MP, Klein C, Swails JM, Hernández CX, et al. MDTraj: A Modern Open Library for the Analysis of Molecular Dynamics Trajectories. *Biophysical Journal* 2015;109(8):1528 – 1532.
- [9] Michaud-Agrawal N, Denning EJ, Woolf TB, Beckstein O. MDAAnalysis: A toolkit for the analysis of molecular dynamics simulations. *J Comput Chem* 2011;32(10):2319–2327.
- [10] Richard J Gowers, Max Linke, Jonathan Barnoud, Tyler J E Reddy, Manuel N Melo, Sean L Seyler, et al. MDAAnalysis: A Python Package for the Rapid Analysis of Molecular Dynamics Simulations. In: Sebastian Benthall, Scott Rostrup, editors. *Proceedings of the 15th Python in Science Conference*; 2016. p. 98 – 105.
- [11] Frisch MJ, Trucks GW, Schlegel HB, Scuseria GE, Robb MA, Cheeseman JR, et al., Gaussian~16 Revision C.01; 2016. Gaussian Inc. Wallingford CT.
- [12] Schaftenaar G, Vlieg E, Vriend G. Molden 2.0: quantum chemistry meets proteins. *Journal of Computer-Aided Molecular Design* 2017;31(9):789–800. <https://doi.org/10.1007/s10822-017-0042-5>.
- [13] Werner HJ, Knowles PJ, Knizia G, Manby FR, Schütz M. Molpro: a general-purpose quantum chemistry program package. *WIREs Comput Mol Sci* 2012;2:242–253.
- [14] To learn about QCSchema, please refer to its website; <https://molssi-qc-schema.readthedocs.io/en/latest/index.html>.
- [15] Barca GMJ, Bertoni C, Carrington L, Datta D, De Silva N, Deustua JE, et al. Recent developments in the general atomic and molecular electronic structure system. *The Journal of Chemical Physics* 2020;152(15):154102. <https://doi.org/10.1063/5.0005188>.
- [16] Neese F. Software update: the ORCA program system, version 4.0. *WIREs Computational Molecular Science* 2018;8(1):e1327.
- [17] Shao Y, Gan Z, Epifanovsky E, Gilbert ATB, Wormit M, Kussmann J, et al. Advances in molecular quantum chemistry contained in the Q-Chem 4 program package. *Mol Phys* 2015;113:184–215.
- [18] Kühne TD, Iannuzzi M, Del Ben M, Rybkin VV, Seewald P, Stein F, et al. CP2K: An electronic structure and molecular dynamics software package - Quickstep: Efficient and accurate electronic structure calculations. *The Journal of Chemical Physics* 2020;152(19):194103. <https://doi.org/10.1063/5.0007045>.

- [19] Parrish RM, Burns LA, Smith DGA, Simonett AC, DePrince AE, Hohenstein EG, et al. Psi4 1.1: An Open-Source Electronic Structure Program Emphasizing Automation, Advanced Libraries, and Interoperability. *Journal of Chemical Theory and Computation* 2017;13(7):3185–3197. <https://doi.org/10.1021/acs.jctc.7b00174>.
- [20] TURBOMOLE V7.4 2019, a development of University of Karlsruhe and Forschungszentrum Karlsruhe GmbH, 1989-2007, TURBOMOLE GmbH, since 2007; available from [http://www.turbomole.com](http://www.turbomole.com;);
- [21] Smith DGA, Altarawy D, Burns LA, Welborn M, Naden LN, Ward L, et al. The MolSSI QCArchive project: An open-source platform to compute, organize, and share quantum chemistry data. *WIREs Computational Molecular Science*;n/a(n/a):e1491. <https://onlinelibrary.wiley.com/doi/abs/10.1002/wcms.1491>.

## GRAPHICAL ABSTRACT

