Clays and Clay Minerals, Vol. 67, No. 5:419–426, 2019

atom: A MATLAB PACKAGE FOR MANIPULATION OF MOLECULAR SYSTEMS

Michael Holmboe*
[1]Chemistry Department, Umeå University, SE-901 87 Umeå, Sweden

Abstract—This work presents Atomistic Topology Operations in MATLAB (*atom*), an open source library of modular MATLAB routines which comprise a general and flexible framework for manipulation of atomistic systems. The purpose of the *atom* library is simply to facilitate common operations performed for construction, manipulation, or structural analysis. Due to the data structure used, atoms and molecules can be operated upon based on different chemical names or attributes, such as *atom*- or *molecule*-ID, *name*, *residue name*, *charge*, *positions*, etc. Furthermore, the Bond Valence Method and a neighbor-distance analysis can be performed to assign many chemical properties of inorganic molecules. Apart from reading and writing common coordinate files (*.pdb, .xyz, .gro, .cif*) and trajectories (*.dcd, .trr, .xtc*; binary formats are parsed via third-party packages), the *atom* library can also be used to generate topology files with bonding and angle information taking the periodic boundary conditions into account, and supports basic Gromacs, NAMD, LAMMPS, and RASPA2 topology file formats. Focusing on clay-mineral systems, the library supports CLAYFF (Cygan, 2004) but can also generate topology files for the INTERFACE forcefield (Heinz, 2005, 2013) for Gromacs and NAMD.

Keywords—CLAYFF · Gromacs · INTERFACE force field · MATLAB · Molecular dynamics simulations · Monte Carlo

INTRODUCTION

Molecular modeling is becoming increasingly important in many different areas of fundamental and applied research (Cygan 2001; Lu et al. 2006; Medina, 2009). This is due not only to the development of high-performance computing in recent decades but also because of increasingly user-friendly molecular modeling software, allowing for high-throughput screening of large classes of molecules. This is especially true for life-science related research, which traditionally has spearheaded the development and design of molecular modeling software and forcefield tools. However, due to the wide use of molecular modeling in different scientific disciplines, custom-made setup or analysis tools specifically tailored to the user's system are often required. This is because of the need to perform specific tasks that topology tools aimed towards modeling of biomolecules normally do not offer. Examples of this include the ability to perform: (1) automatic atomtype assignment for less common or system-specific forcefields; or (2) the ability to find bonds and angles across the periodic boundary conditions (PBC) for periodic slabs or layered molecules, such as graphene and graphite oxides, zeolites, clay minerals, layered double hydroxides, etc.

In this context and especially for scientists lacking comprehensive programing skills, MATLAB, R, and similar software packages offer an attractive and versatile environment for scientific computing with simple yet robust scripting languages, plotting capabilities, and the possibility to display or copy data variables into spreadsheets. In fact, a handful of MATLAB and R packages for trajectory or statistical analysis focusing on biomolecules has been developed (Dien et al. 2014; Dombrowsky et al. 2018; Kapla & Lindén 2018; Matsunaga & Sugita 2018). This work presents the Atomistic Topology Operations in MATLAB (*atom*) library – a large collection of >100 modular MATLAB scripts and sub-routines (e.g. functions) under an open source license (simple BSD) which comprise a general framework for construction, manipulation, and analysis of atomistic systems, with the option of incorporating topological and forcefield information. In contrast to the above-mentioned MATLAB and R trajectory packages, the focus here lies not on the modeling of biomolecules but rather on the investigation of especially periodic inorganic structures, such as slabs and layers that have well defined unit cells (UC) stretching over the PBC.

Because the *atom* function calls can be invoked from the command line as well as from within custom-made MATLAB scripts and programs, the *atom* library is especially well suited to batch-mode operations. In particular, the *atom* library may be useful for scientists with an interest in modeling inorganic and geochemical systems. This is because apart from parsing the input or output of basic coordinate files (*.pdb*|*.xyz*|*.gro*|*.cif*), the *atom* library can also be used to output basic molecular topology files (*.lmp*|*.itp*|*.psf*) for the CLAYFF forcefield (Cygan et al. 2004) and INTERFACE (Heinz et al. 2005, 2013) for certain geochemical systems/software, which are used in molecular modeling software such as LAMMPS, Gromacs, NAMD and RASPA2 (Plimpton 1995; Phillips et al. 2005; Abraham et al. 2015; Dubbeldam et al. 2016).

METHODS

The unifying theme among the different library functions is the *atom* (see Fig. 1), which is the default name of the variable typically containing the molecular information from a coordinate file such as a *.pdb* file. In order to call the different library

* E-mail address of corresponding author: michael.holmboe@umu.se
DOI: 10.1007/s42860-019-00043-y

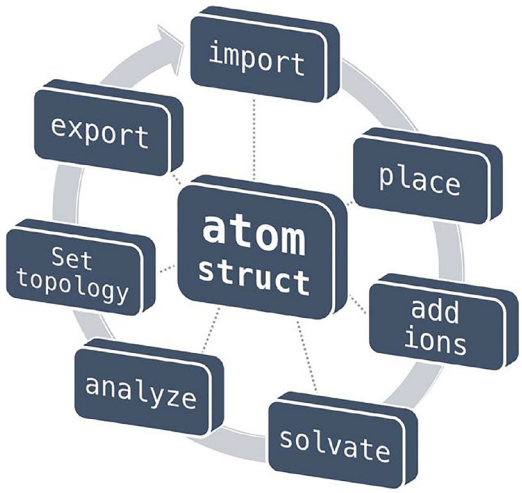

Fig. 1 Schematic of the typical workflow using the *atom* library, illustrating various operations performed on the *atom struct* variable

functions, i.e. to perform specific operations on the *atom* variable, the following syntax is used:

```
atom = {operation}_atom ({arguments})
```

where examples of common {operations} are: *composition*, *clayff, copy*, *create*, *dist_matrix*, *element*, *find_bonded*, *import*, *insert*, *interface, ionize*, *mass*, *radii*, *reorder*, *rotate*, *scale*, *slice*, *solvate*, *translate*, *update*, *wrap*, *write,* etc. The {arguments} represent required function input variables, which depend on the intended specific {operations} to be made. It is normally a coordinate filename, or the *atom* variable itself, followed by the UC or system size variable *Box_dim* (if such exists). Functions requiring additional arguments like *limits* (representing a volumetric region) are described in the documentation for each function.

The properties of each individual atomistic particle are accessible by its *atom*-ID index (ranging from 1 to the number of atomistic particles), because the *atom* variable itself is an indexed data type called structure array (denoted *struct*). The *struct* variable stores the individual atom and molecule attributes (i.e. chemical names and properties) in so-called fields, where the associated syntax is based on the so-called dot-notation as in [*struct.field*], or more generically [*atom.attribute*]. One advantage of the dot-notation is that it enables a flexible syntax for advanced selections because individual atoms and molecules can be filtered or selectively manipulated based on one or more specific attributes. This is done by using logical indexing (in the case of fields with numeral data) or string comparison (in case of fields containing text). A purposeful example is shown below, demonstrating how to select and delete (1) all atoms named Si that (2) belong to the molecule/molecule, (3) have the residue/molecule name MMT, and (4) have z coordinates of <10 Å (... only needed in case of line breaks).

```
atom(strcmp([atom.type],'Si') &...
     strcmp([atom.resname],'MMT') &...
    [atom.molid]==2 &...
    [atom.z] < 10) = [ ]
```

Similar combined selections can be made on all the different *struct* attributes. The default attributes include the *molecule*-ID, *atom*-ID, *atomtype names*, *residue names,* and the *coordinates*, and are denoted as *molid*, *index*, *type*, *resname*, *x*, *y*, *z*, respectively. If invoking other functions, the *atom struct* will expand to incorporate optional attributes, for instance storing information on velocities, forces, neighbors, bonds, and angles, or strictly chemical properties like the atomic mass, partial charges, bond valence values, and so on.

Another benefit of the *atom struct* variable type is that several *struct*s can easily be merged:

```
atom123 = [atom1 atom2 atom3]
```

Although this notation works well for concatenating (i.e. stacking) molecules with the same attributes, the *update_atom()* function is generally superior as it also updates the *atom*- and *molecule*-ID indexes accordingly:

```
atom123 = update_atom({atom1 atom2 atom3})
```

The *atom* library provides several functions to create, append, duplicate, solvate, translate, rotate, and slice parts of or whole molecules. Hence, in addition to the flexible selection syntax using the dot-notation described above, this library is ideal for building or manipulating the structure of multicomponent systems by adding entire crystal slabs, molecules, ions, and solvent molecules.

HIGHLIGHTS

Construction of Multi-component Systems

Building basic configurations of multicomponent and/or multilayered systems normally proceed by initially placing the primary solute molecules, e.g. a protein or a crystallographic mineral slab, into a pre-defined simulation box using functions such as *place_atom()*, *center_atom()*, *translate_atom()*, *wrap_atom(),* and so on.

https://doi.org/10.1007/s42860-019-00043-y Published online by Cambridge University Press

Ions can be added into a simulation box either randomly in specified regions (but without atomic overlap of existing molecules), on specified planes (in 3D), or with exponentially decaying concentrations from a surface with the function *ionize_atom()*. Other similar functions to add new or copy existing solutes also exist.

In terms of solvation, the function *solvate_atom()* can solvate an entire simulation box or limited regions with generic three-, four-, or five-site water models, such as SPC/E or TIP3P/TIP4P/TIP5P, either as liquid water or hexagonal ice. Solvation with custom solvents using unwrapped solvent boxes is also supported, as is scaling the solvent density to optimize solvent molecule packing. Furthermore, one can solvate a solute by a certain solvation thickness (*Ex.* around a centered protein).

Visualization

For visualization one can make use of the function *vmd()* if the VMD software is installed separately (Humphrey et al. 1996), and the *PATH2VMD()* function is properly set. Alternatively, the basic *plot_atom()* or the *plot_density_atom()* functions rapidly plot a full simulation box with thousands of atoms (see Fig. 2), with or without the density profiles along the *x*,*y*,*z*-dimensions. Analogously, and although it is significantly slower the *show_atom()* and the *show_density_atom()* functions can be used to render glossy atoms with cylindrical bonds or VdW spheres.

Bonding and Structural Analysis

Most functions, such as *dist_matrix_atom()* and *bond_angle_atom()*, take PBC into account, which allows for generation of molecular topologies with bonds, angles and basic dihedrals across the PBC. Many functions also support triclinic boxes using the 'tilt vectors' *xy*, *xz*, *yz* as defined in Gromacs and LAMMPS. Several of the functions use two types of neighbor cutoffs to find interacting atoms, one shorter cutoff for bonded H's (default 1.25 Å) and one larger cutoff for all other interactions (default 2.25 Å). Furthermore, a general neighbor-distance analysis can be performed: (1) by comparison with the Revised Shannon radii (Shannon 1976; van Horn 2001); (2) by calculating the bond-valence values using the semi-empirical *Bond Valence Method* (Brown 2009, 2013, 2016) or; (3) by calculating basic XRD profiles. Here, analysis 2 is performed most easily by calling the *analyze_atom()* function, which first finds all atom bonds and angles to the nearest neighbors. Secondly, it makes educated guesses based on the chemical name and apparent coordination number, the oxidation state, electron configuration, and the ideal crystalline, ionic, and vdW-radii of the atoms. This function is, hence, useful for sanity-checks of either the initial input structures or to scrutinize one simulation result. As the Bond Valence Method also calculates the Global Instability Index (G_{II}), it can further be used in structural data-mining studies to check for structural stress in molecules. As for analysis 3 and XRD analysis, a basic XRD function called *XRD_atom()* for single-crystal UCs is provided, using 11-coefficient atomic scattering factors (Waasmaier & Kirfel 1995).

Trajectory Analysis

Although not the primary purpose of the *atom* library, support for trajectory reading (and writing) of binary trajectories utilizing the third-party MatDCD or the mxdrfile packages for the *.dcd* and *.xtc*|*.trr* trajectory formats, respectively, is provided (Gullingsrud 2000; Kapla & Lindén 2018). For reading *.xtc* trajectories, one could also use the Gro2mat package

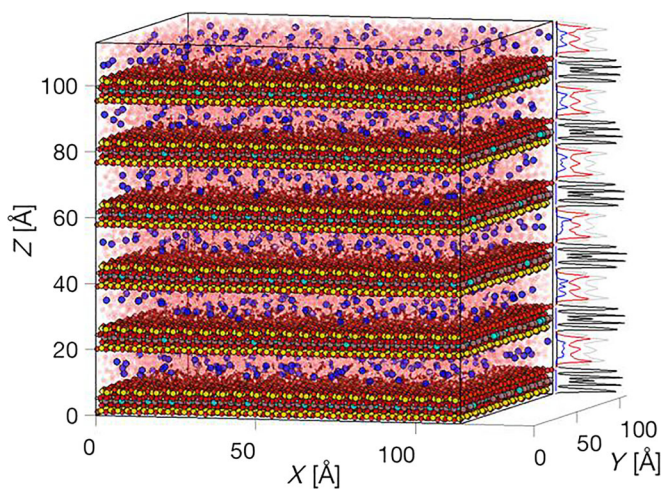

Fig. 2 Example image of a 136,000 particle system generated with the *plot_density_atom()* function, showing scaled density profiles of a multilayered montmorillonite particle (black profiles), Na^+ counter-ions (blue profiles) hydrated by three pseudo-layers of water (O_{water} in red, H_{water} in gray profiles)

https://doi.org/10.1007/s42860-019-00043-y Published online by Cambridge University Press

(Dien et al. 2014). In addition to these third-party binary trajectory packages, text-format trajectories in the *.pdb|.xyz|.gro* formats are also natively supported by the *atom* library, where one special version written specifically for the MC code RASPA2 named *import_mc_pdb_traj()* can handle trajectories with non-constant numbers of atoms.

In order to import a trajectory into MATLAB as shown below, both a coordinate file and a trajectory file is needed.

```
[atom,traj] = import_traj (...
                coordfile,trajfile)
```

Apart from the *atom struct* taken from the coordinate file, this command results in a data matrix called *traj*, of size 3*N*×frames, where *N* represents the number of atoms in the trajectory.

Case Study: A Hydrated Na^+-montmorillonite Nanopore System with CLAYFF atomtypes

This example demonstrates the main steps needed to setup a simulation box for MC or MD simulations of a system representing the water–mineral interface to a defect mineral particle called *MMT* (as in Montmorillonite). *MMT* will describe a negatively charged smectite layer, with an approximate thickness of 1 nm and being infinite in the lateral plane, due to the fact that PBC is used in MC or MD simulations and because the layer in this example will have no edges in order to avoid complexity.

Preparation of the Mineral Lattice

The negative charge of *MMT* mainly originates from isomorphic substitution of octahedrally coordinated Al(III) with Mg(II), which creates a charge defect which is smeared over the nearest coordinating O(-II) in the lattice. In most cases when performing molecular modeling of inorganic or mineral particles, initial UC data are available from either X-ray diffraction data or quantum mechanical calculations. For *MMT*, due to the isomorphic substitution, no precise UC data exist. Instead, one can use isostructural and defect-free UC data from pyrophyllite (Lee & Guggenheim 1981), and then replicate it to a mineral layer and subject it to isomorphic substitution. Hence, initially, a pyrophyllite UC structure must be imported:

```
Pyrox1 = import_atom('Pyrophyllite.pdb')
```

In many cases when UC information is taken from XRD data, the number and positions of H atoms are somewhat unreliable. Hence, in this example both initial removal and addition of H atoms to the clay lattice are demonstrated. First, all atoms with names starting with 'H' are removed.

```
Pyrox1(strncmpi([Pyrox1.type],'H',1)) = [ ]
```

Then the remaining mineral lattice is analyzed for the resulting structure using the function *analyze_atom()*.

```
properties = analyze_atom(Pyrox1,Box_dim)
```

This function will output a separate *properties struct* with bonding information, as well as reveal which atoms need healing by outputting a *heal_ind* variable. To protonate those individual atoms, one would issue:

```
H_atom = protonate_atom (...
            Pyrox1,Box_dim,healind)
```

To simplify this example one could convert the triclinic UC to an orthogonal UC:

```
Pyrox1 = orto_atom (...
            Pyrox1H_atom,Box_dim)
```

Next, the UC must be replicated into a layer, for instance by 6×4 times in the *X* and *Y*-directions, respectively:

```
Pyrox6x4 = replicate_atom (...
              Pyrox1,orto_Box_dim,[6 4 1])
```

From this pyrophyllite layer, a *MMT* layer can be generated by performing isomorphic substitution by replacing Al lattice atoms with Mg in two-thirds of all the UCs (which is typical for natural MMT clay minerals). The last numerical argument (in Ångström units) sets the nearest Mg–Mg neighbor distance, as charge defects are unlikely to be close to each other.

```
MMT = substitute_atom (Pyrox6x4,...
            Box_dim,6*4*2/3,'Al','Mg',5.5)
```

For consistency, one can at this point change the attribute *MMT.resname* from 'PYR' to 'MMT':

```
[MMT.resname] = deal({'MMT'})
```

Here, the *deal()* is a native MATLAB function. Next, the names of all atoms (i.e. the *atomtypes*) in MMT can be set to a specific forcefield, such as the original CLAYFF (Cygan et al. 2004).

```
MMT = clayff_2004_atom (MMT,Box_dim,'clayff')
```

This function will also output the layer net charge (–16 from the isomorphic substitution), which must be counterbalanced by counter-cations such as Na^+. In order to accommodate these ions and later water molecules, the dimensions of the definitive simulation box should be defined, from the lateral dimensions of the MMT layer along *X/Y* (already set in the existing *Box_dim* variable) and an arbitrary height in Z, here set to 40 Å:

https://doi.org/10.1007/s42860-019-00043-y Published online by Cambridge University Press

```
Box_dim(3) = 40; % in Angstrom
```

Adding Counterions and Solvating with Water

Ions can be added to an existing simulation box in several ways using the *ionize_atom()* function. In this example, an *Ions struct* is created by adding ions close to the mineral surface (but >3 Å) of the wrapped *MMT*:

```
Ions = ionize_atom(...
        'Na','Na',Box_dim,16,3,...
        wrap_atom(MMT,Box_dim),'surface')
```

Note that wrapping the *MMT* layer into the box before adding the ions is important in order to avoid atomic overlap. Next, a *Water struct* is created in order to solvate the simulation box. This is accomplished by placing 1000 SPC/E water molecules with a relative density of 1.05 no nearer than 2 Å to any solute:

```
Water = solvate_atom(Box_dim,1.05,...
                2,1000,update_atom({MMT,Ions}))
```

Finally, all components are merged together using *update_atom()* into single *System struct*, which updates all *atom*- and *molecular*-ID indexes, and an ouput *.pdb* file is written. Note that because the different *System struct* components are stored in separate *struct* variables, the order of the components can be changed arbitrarily.

```
System = update_atom({MMT Water Ions})
write_atom_pdb(System,...
        Box_dim,'System.pdb')
```

The following lines show example commands used to plot the final *System struct* (see Fig. 3), where, for instance, the optional last numerical in the *plot_atom()* function denotes the VdW radii scaling factor, whereas the 'vdw' in the *show_atom()* function sets the display style:

```
plot_atom(System,Box_dim,0.2)
show_atom(System,Box_dim,'vdw')
vmd(System,Box_dim)
vmd('System.pdb')
```

Writing Molecular Topology Files

The *atom* library contains different flavors of both the CLAYFF and INTERFACE forcefields. The following examples show how to write basic *.psf*, *.lmp*, and *.itp* topology files according to the original CLAYFF to be used in the MD software like NAMD2, LAMMPS, and Gromacs.

```
write_atom_psf(System,Box_dim,'System')
write_atom_lmp(System,Box_dim,'System')
write_atom_itp(MMT,Box_dim,'MMT')
```

Note that when generating molecular topology files, one must, in general, decide which components to include (i.e. the *MMT*, *Ions,* and *Water*) in the molecular topology, as different MD and MC software may use different approaches on how to include molecular topologies in the total system topology. For instance, in Gromacs, separate molecular topology files are used commonly (*.itp* files) for different components; hence, the *Ions* and the *Water* in the current example would preferably not be included in the same molecular topology as MMT.

On a separate note, because the original CLAYFF (nor INTERFACE) does not contain all different oxygen atom types needed to model all minerals or, for instance, custom clay edge models, the *atom* library can also be used to derive partial charges and molecular topologies for new oxygen atom types not part of the original CLAYFF forcefield. This is accomplished by equation 1, which in fact can be used to calculate the partial charge of any new or original oxygen atom type in CLAYFF (Tournassat et al. 2016; Lammers et al. 2017).

$$Z_O = -2-\left(-\sum_n \frac{Z_n^f - Z_n^p}{CN_n}\right) \quad (1)$$

In this equation, the partial charge of a CLAYFF oxygen type (Z_O) is obtained from its formal charge (–2) minus the total charge needed to balance (hence the extra minus sign) the charge distributed over all the n nearest coordinating cationic centers (including H). This distributed charge represented by the second term on the right hand side is, in turn, calculated from the difference between the formal and partial charge Z_n^f and Z_n^p of each cation (hence the summation), divided by their respective number of coordinating oxygens, CN_n. For a more extensive explanation, see Lammers et al. (2017).

Analyzing the Lattice Structure

In order to study the structural integrity of molecules subjected to molecular simulations, computing the root mean square deviation of atomic positions (RMSD) from the positions in the input structure is an often-used method. This method is frequently used for large biomolecules; however, for crystalline inorganic lattice structures the semi-empirical Bond Valence Method can also be used. This method finds strained atoms in a lattice by relating the so-called bond valence values to the ideal semi-empirical bond distances (Brown 2016), as well as by comparing the total bond valence sum to the ideal atomic valence. In this example, this method was applied by invoking the *analyze_atom()* function on the average *MMT* lattice structure (Fig. 3) obtained from equilibrated simulations performed in Gromacs in the NVT ensemble over 1 ns. Six different forcefield implementations were tested in total, namely: (1–2) two implementations of the INTERFACE forcefield (Heinz et al. 2005, 2013), using the experimental or pre-defined bond distance (with the 5%

https://doi.org/10.1007/s42860-019-00043-y Published online by Cambridge University Press

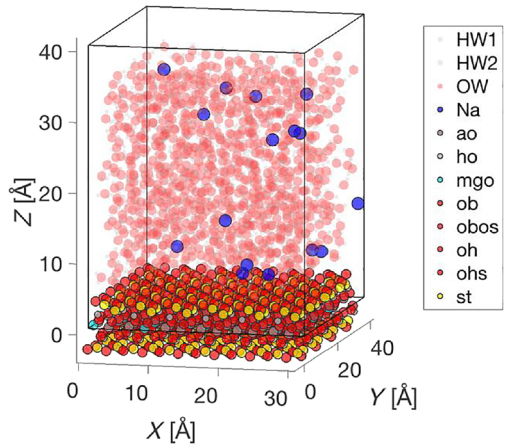

Fig. 3 The final hydrated and unwrapped Na^+-MMT System generated by the *plot_atom()*. Note the legend which indicates the CLAYFF

scaling factor) and angle values, respectively; (3) the original unconstrained CLAYFF (Cygan et al. 2004), having no defined metal–oxygen bonds or angles; (4) CLAYFF with no bonds but all M–O angle terms defined by the experimental values (as in 1); (5) a modified version of CLAYFF optimized for 1D-XRD modeling, using a larger basal O radius but smaller Si radius (Szczerba & Kalinichev 2016); and lastly (6) the same modified version of CLAYFF, but with all angle terms defined as in (4).

The resulting average bond distances (Fig. 4A) from the simulations generally agreed well among the various INTERFACE forcefields, apart from the Mg bond distance for CLAYFF, which was found to relax considerably compared to Al and the INTERFACE forcefields.

Average bond distances reveal little, however, about the strain in lattices. The extended Mg–O bond distances found with the unconstrained CLAYFF but not with the constrained INTERFACE forcefields is reasonable considering the bond valence sum values. This is because a relaxed coordinating octahedron of the neighboring O atoms is needed in order to maintain a physically reasonable atomic valence (+2 for Mg, see Fig. 4B). Nevertheless, the Global instability index, G_{II}, which is a measure of the structural strain for the six different forcefield implementations (1–6) were 0.257, 0.230, 0.366, 0.233, 0.294, and 0.237, respectively. Overall this indicates that the implementations of the forcefields used are not perfect, as G_{II} values >0.2 from experimentally determined structures are found rarely (Brown 2009), and hence indicate unstable lattices. Simulations with pyrophyllite (not shown) resulted in similar G_{II} values, which can be compared to the input pyrophyllite structure having a G_{II} value of 0.09. Although the INTERFACE 2013 implementation resulted in the lowest G_{II} value and, hence, the best representation of the internal atomic UC positions, the CLAYFF implementations with defined angles perform almost as well, and are better suited for 1D-XRD modeling due to the larger radius of the basal O atoms (Ferrage et al. 2011; Szczerba & Kalinichev 2016). Still, the radii of the atom types in CLAYFF could likely be further optimized to better represent the bond distances within the actual UC.

DOCUMENTATION

The entire atom library contains >100 unique functions that are summarized in a self-contained html-based documentation, containing explanations for the most common *variables* and functions and include multiple examples which can be executed interactively from within MATLAB, or be viewed in a browser (also available at moleculargeo.chem.umu.se/codes/atom-scripts). Furthermore, each *function* describes examples demonstrating all available function arguments.

INSTALLATION

The latest version of the *atom* library can be downloaded from the MATLAB File Exchange or the GitHub repository github.com/mholmboe/atom. No installation is needed, the library files must only be added to the MATLAB path. Optional use of the visualization program VMD or the MD

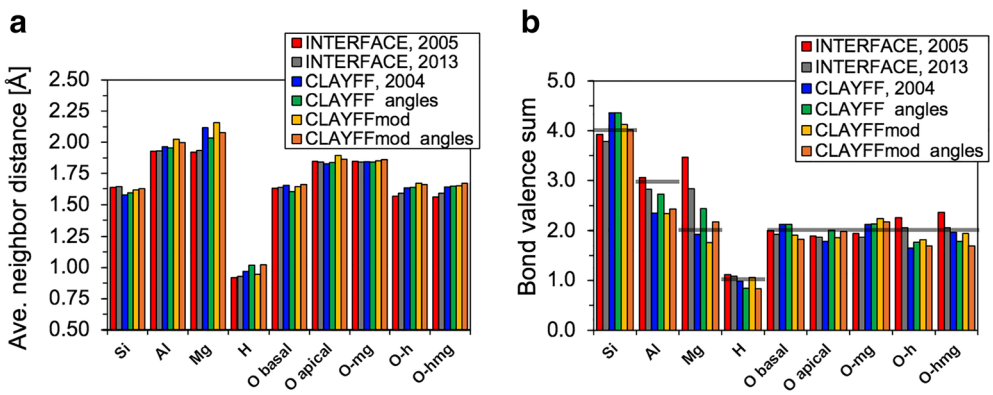

Fig. 4 Results from the structural analysis comparing six different implementations of the INTERFACE and CLAYFF forcefields (as explained in the text). (**a**) Average bond distances for the generic clay lattice atomtypes. (**b**) The corresponding bond valence sum, compared to the ideal atomic valence, shown as gray lines

https://doi.org/10.1007/s42860-019-00043-y Published online by Cambridge University Press

package Gromacs (which must be installed separately) from within the *atom* library is possible if the environmental variables in the functions called *PATH2VMD()* and *PATH2GMX()* are set. Binary trajectories can be imported with the *import_traj* function, generating a trajectory matrix and separate *Box_dim* variable. Supported formats are the *.dcd* format using the MatDCD script, or the Gromacs formats *.trr* and *.xtc* by using mxdrfile (recommended for Gromacs trajectories) or Gro2mat (Gullingsrud 2000; Dien et al. 2014; Kapla & Lindén 2018).

CONCLUSIONS

A flexible MATLAB library for construction and manipulation of molecular models and systems is presented. This *atom* library contains many useful and time-saving functions that can be used by most researchers with even modest programming experience, in order to setup and perform basic analysis of molecular simulation systems. The main use of the library is to construct custom-made molecular systems and generate topological bonding and angle information across the PBC from the command line, for modeling of geochemical systems such as hydrated clay modeled with the CLAYFF or INTERFACE forcefields in different molecular dynamics or Monte-Carlo simulation packages.

ACKNOWLEDGEMENTS

Open access funding provided by Umea University. The author acknowledges useful comments by A. Ohlin, and funding from the Faculty of Science and Technology at Umeå University and the Kempe foundation, as well as HPC resources provided by the Swedish National Infrastructure for Computing (SNIC) at High Performance Computing Center North (HPC2N), Umeå University.

Compliance with Ethical Standards

Conflict of Interest

The authors declare that they have no conflict of interest.

REFERENCES

Abraham, M. J., Murtola, T., Schulz, R., Pall, S., Smith, J. C., Hess, B., & Lindah, E. (2015). Gromacs: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX, 1–2*, 19–25. https://doi.org/10.1016/j.softx.2015.06.001.

Brown, I. D. (2009). Recent developments in the methods and applications of the bond valence model. *Chemical Reviews, 109*(12), 6858–6919. https://doi.org/10.1021/cr900053k.

Brown, I.D. (2013). Bond Valences in Education. In D. M. P. Mingos (Ed.), *Structure and Bonding* (Vol. 158, pp. 233–250). Springer. doi: https://doi.org/10.1007/430_2012_83.

Brown, I.D. (2016). Bond valence parameters. Retrieved November 3, 2016, from www.iucr.org/resources/data/datasets/bond-valence-parameters

Cygan, R. T. (2001). Molecular modeling in mineralogy and geochemistry. *Reviews in Mineralogy and Geochemistry, 42*(1), 1–35. https://doi.org/10.2138/rmg.2001.42.1.

Cygan, R. T., Liang, J. J., & Kalinichev, A. G. (2004). Molecular models of hydroxide, oxyhydroxide, and clay phases and the development of a general force field. *Journal of Physical Chemistry B, 108*(4), 1255–1266.

Dien, H., Deane, C. M., & Knapp, B. (2014). Gro2mat: A package to efficiently read gromacs output in MATLAB. *Journal of Computational Chemistry, 35*(20), 1528–1531. https://doi.org/10.1002/jcc.23650.

Dombrowsky, M., Jager, S., Schiller, B., Mayer, B. E., Stammler, S., & Hamacher, K. (2018). StreaMD : advanced analysis of molecular dynamics using R. *Journal of Computational Chemistry, 39*, 1666–1674. https://doi.org/10.1002/jcc.25197.

Dubbeldam, D., Calero, S., Ellis, D. E., & Snurr, R. Q. (2016). RASPA: molecular simulation software for adsorption and diffusion in flexible nanoporous materials. *Molecular Simulation, 42*(2), 81–101. https://doi.org/10.1080/08927022.2015.1010082.

Ferrage, E., Sakharov, B. A., Michot, L. J., Delville, A., Bauer, A., & Lanson, B. (2011). Hydration Properties and Interlayer Organization of Water and Ions in Synthetic Na-Smectite with Tetrahedral Layer Charge. Part 2. Toward a Precise Coupling between Molecular Simulations and Diffraction Data. *The Journal of Physical Chemistry C, 115*(5), 1867–1881. https://doi.org/10.1021/jp105128r.

Gullingsrud, J. (2000). MatDCD - MATLAB package DCD reading/writing. Retrieved from www.ks.uiuc.edu/Development/MDTools/matdcd/.

Heinz, H., Koerner, H., Anderson, K. L., Vaia, R. A., & Farmer, B. L. (2005). Force field for mica-type silicates and dynamics of octadecylammonium chains grafted to montmorillonite. *Chemistry of Materials, 5*, 5658–5669.

Heinz, H., Lin, T. J., Kishore Mishra, R., & Emami, F. S. (2013). Thermodynamically consistent force fields for the assembly of inorganic, organic, and biological nanostructures: The INTERFACE force field. *Langmuir, 29*, 1754–1765. https://doi.org/10.1021/la3038846.

Humphrey, W., Dalke, A., & Schulten, K. (1996). VMD – Visual molecular dynamics. *Journal of Molecular Graphics, 14*, 33–38.

Kapla, J., & Lindén, M. (2018). Mxdrfile: read and write Gromacs trajectories with MATLAB. *ArXiv:1811.03012 [Physics.Comp-Ph]*, 1–3. Retrieved from http://arxiv.org/abs/1811.03012.

Lammers, L. N., Bourg, I. C., Okumura, M., Kolluri, K., Sposito, G., & Machida, M. (2017). Molecular dynamics simulations of cesium adsorption on illite nanoparticles. *Journal of Colloid and Interface Science, 490*, 608–620. https://doi.org/10.1016/j.jcis.2016.11.084.

Lee, J. H., & Guggenheim, S. (1981). Single crystal X-ray refinement of pyrophyllite-1Tc. *American Mineralogist, 66*, 350–357.

Lu, D., Aksimentiev, A., Shih, A. Y., Cruz-Chu, E., Freddolino, P. L., Arkhipov, A., & Schulten, K. (2006). The role of molecular modeling in bionanotechnology. *Physical Biology, 3*, S40–S53. https://doi.org/10.1088/1478-3975/3/1/S05.

Matsunaga, Y., & Sugita, Y. (2018). Refining Markov state models for conformational dynamics using ensemble-averaged data and time-series trajectories. *The Journal of Chemical Physics, 148*(24), 241731. https://doi.org/10.1063/1.5019750.

https://doi.org/10.1007/s42860-019-00043-y Published online by Cambridge University Press

Medina, G. M. (2009). Molecular and multiscale modeling: Review on the theories and applications in chemical engineering. *Ciencia Tecnologia y Futuro, 3*, 205–224.

Phillips, J. C., Braun, R., Wang, W. E. I., Gumbart, J., Tajkhorshid, E., & Villa, E. (2005). Scalable Molecular Dynamics with NAMD. *Journal of Computational Chemistry, 26*, 1781–1802. https://doi.org/10.1002/jcc.20289.

Plimpton, S. (1995). Fast parallel algorithms for short-range molecular dynamics. *Journal of Computational Physics, 117*, 1–42.

Shannon, R. D. (1976). Revised effective ionic radii and systematic studies of interatomic distances in halides and chalcogenides. *Acta Crystallographica Section A Foundations of Crystallography, A32*, 751–767.

Szczerba, M., & Kalinichev, A. G. (2016). Intercalation of ethylene glycol in smectites: Several molecular simulation models verified by X-ray diffraction data. *Clays and Clay Minerals, 64*, 488–502. https://doi.org/10.1346/CCMN.2016.0640411.

Tournassat, C., Bourg, I., Holmboe, M., Sposito, G., & Steefel, C. (2016). Molecular dynamics simulations of anion exclusion in clay interlayer nanopores. *Clays and Clay Minerals, 64*, 374–388. https://doi.org/10.1346/CCMN.2016.0640403.

van Horn, D. (2001). Electronic Table of Shannon Ionic Radii. Retrieved from http://v.web.umkc.edu/vanhornj/shannonradii.htm.

Waasmaier, D., & Kirfel, A. (1995). New analytical scattering-factor functions for free atoms and ions. *Acta Crystallographica Section A Foundations of Crystallography, 51*, 416–431. https://doi.org/10.1107/S0108767394013292.