MoSDeF, a Python-Based Molecular Simulation and Design Framework

Peter T. Cummings,[1,2] Clare McCabe,[1,2] Christopher R. Iacovella,[1,2] Ákos Lédeczi[3]

[1] Department of Chemical and Biomolecular Engineering, Vanderbilt University, Nashville, TN

[2] Multiscale Modeling and Simulation Center, Vanderbilt University, Nashville, TN

[3] Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN

Motivation

Scientific research of all kinds, from experimental to computational, are plagued with issues regarding reproducible science. In computational chemistry related research, initialization of the system and application of force field parameters are not rigorously reported. This leads to discrepancies when trying to reproduce research, which leads to the question:

How can a computational study be performed in a way that is Transparent, Reproducible, Usable by others, and Extensible (TRUE)?

Sandve, *et al.*[1] provided ten "rules" for reproducible computational research proposing, among others, that *version control*, *scripting*, and *public access* to scripts, runs, and results, are key to reproducibility.

Challenge

Develop a set of computational tools to serve as a scriptable, open-source, self-documenting suite of software that assists in the initialization and parameterization of chemical systems of interest. These systems should be:

- **Non-specific**
- **Force field agnostic**
- **Engine agnostic**

With these design principals in mind, an extensible set of tools can be developed to ensure system initialization and parameterization are reproducible by the general community.

Approach

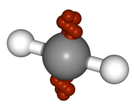
We describe MoSDeF[2], a suite of extensible, open-source, Python tools designed to facilitate the programmatic initialization, atom typing, and screening of chemical systems. MoSDeF is built around two key pieces of software: *mBuild,*[3,4] and *foyer.*[5,6] mBuild[3,4] is used for generating the initial starting configurations of the chemical systems of interest, while Foyer[5,6] is a generalized tool for parameterizing the system. These tools have been used successfully with molecular dynamics simulations and we are currently developing support for various Monte Carlo simulation engines.[7,8]

mBuild: a Molecule Builder

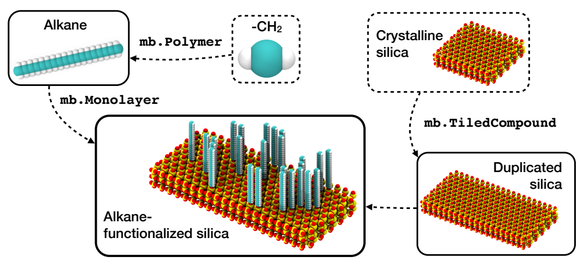
mBuild[3,4] allows for quickly developing molecular systems in a hierarchical manner, constructed programmatically from `Compounds`.

- `Compounds` are *mBuild*'s representation of matter which:
  - Can contain atoms or sets of atoms
  - Can contain data and data operators
  - Can contain other `Compounds`
  - Contain operations on `Compounds`
- `Compounds` can be joined to one another through `Ports`
  - `Ports` are assigned manually based on underlying chemistry
  - `Ports` behave like dangling bonds, can be attached to `Compounds`
  - `Ports`, connected through `force_overlap` method, allow for `Compounds` to be joined to one another

`Ports` attached to a `CH2 Compound`. `Ports` are a collection of four non-interacting particles, which provide enough unique points in space to allow for connections between two `Ports`.



Alkane Polymer Surface

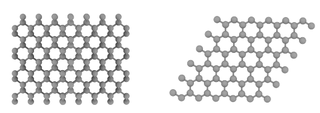



```python
import mbuild as mb

class CH2(mb.Compound):
    """An mBuild Compound representing a methylene bridge. """
    def __init__(self):
        super(CH2, self).__init__()

        # Load the CH2 coordinates and bonds from a structure file
        mb.load(filename='ch2.pdb', compound=self)

        # Locate the carbon atom
        carbon = list(self.particles_by_name('C'))[0]

        # Add Ports for the two dangling bonds
        self.add(mb.Port(anchor=self[0], orientation=[0, 1, 0],
            separation=0.07), label='up')
        self.add(mb.Port(anchor=self[0], orientation=[0, -1, 0],
            separation=0.07), label='down')

# Create a polymer of length 18
polymer = mb.Polymer(monomers=CH2(), n=18)

# Attach copies of the polymer to a surface
from mbuild.lib.surfaces import Betacristobalite
functionalized_surface = mb.Monolayer(surface=Betacristobalite(),
    chains=polymer, pattern=mb.Random2DPattern(25), tile_x=2, tile_y=1)
```

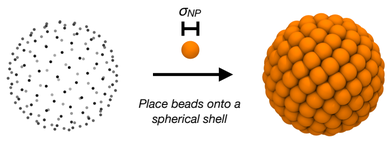
Shown above is a sample Python script that encapsulates the process for building variable size alkane chains, attached to a crystalline silica substrate. Note the use of `Ports` to connect the reusable `CH2 Compounds` to form the polymer. By wrapping up these repetitious parts of building complex chemical structures using object-oriented programming concepts, these traditionally non-trivial starting configurations are now trivial to produce.

mBuild Functionality

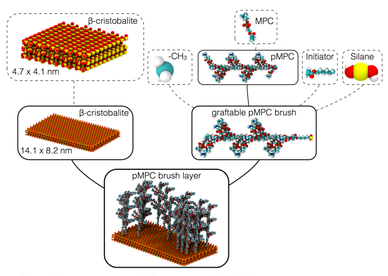
- **Lattice Building**



- **Nanoparticles**



- **Functionalized Monolayers**



- **Many more, including bilayers and stacked lamaelle, MXenes and nanodroples, that can be easily assembled from the base data structures and methods**

Foyer: a General Atom Typer

Foyer[5,6] provides a simulation engine agnostic tool to atom-type a molecular system using first-order logic. While many current atom-typing engines use a rigid rule hierarchy, *Foyer* is flexible enough where adding a new atom-typing rule will not require an overhaul of the underlying code base.

- The parameters and rules are both contained in the same file, separate from the engine itself
- Rules utilize the SMARTS[9] molecular structure definitions, which can be easily validate

General Data Format

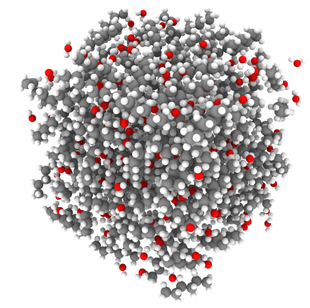
We have extended the OpenMM[10] XML-based forcefield data file, allowing rules (i.e., "`def`") to be encoded via SMARTS[9] and overrides to supersede other definitions. Additionally, we have supported the addition of DOI attributes, providing a link to the original source of the parameters. Furthermore, *Foyer* automatically logs associations between DOIs and atom types during the atom-typing process, providing a BibTeX file featuring the full citation for the sources of all parameters applied to a particular system, along with additional notes detailing precisely which atom types are contained within each source.

```
<ForceField>
  <AtomTypes>
    <Type name="opls_135" class="CT" element="C"
mass="12.01100" \\
          def="[C;D4]([C])([H])([H])([H])"
          desc="alkane CH3" \\
          doi="10.1021/ja9621760"/>
    <Type name="opls_148" class="CT" element="C"
mass="12.01100" \\
          def="[C;D4]([C;%opls_145])([H])([H])([H])"
          overrides="opls_135" \\
          desc="toluene CH3" doi="10.1021/ja9621760"/>
    ...
  </AtomTypes>
</ForceField>
```

mBuild + Foyer: Towards TRUE Simulations

```python
import mbuild as mb

class MixtureBox(mb.Compound):
    """An box of linear alkane chains and water."""
    def __init__(self, chain_length, n_chains, n_water, density):
        """Initialize an AlkaneBox Compound.
        Parameters
        ----------
        chain_length : int
            Length of the alkane chains (in number of carbons)
        n_chains : int
            Number of chains to place in the box
        n_water : int
            Number of water molecules to place in the box
        density : float
            Density (in kg/m^3) at which the system should be created
        """
        super(MixtureBox, self).__init__()
        chain = Alkane(chain_length=chain_length)
        chain.energy_minimization()
        h2o = H2O()
        mixture_box = mb.fill_box(compound=[chain, h2o],
                                  n_compounds=[n_chains, n_water],
                                  density=density)
        for child in mixture_box.children:
            if child.n_particles > 3:
                child.name = 'Alkane'
            else:
                child.name = 'Water'
        self.add(mixture_box)

# build the alkane, water box mixture
chain_length = 6
n_chains = 250
n_water = 200
density = 734.5
mixture_box = MixtureBox(chain_length=chain_length,
                         n_chains=n_chains, n_water=n_water,
                         density=density)

# save out to gromacs compatible data formats
# with opls-aa parameters and SPC/E water parameters
mixture_box.save('mixture.gro', overwrite=True, residues=['Alkane', 'Water'])
mixture_box.save('mixture.top', forcefield_files='utils/spce-alkane.xml', overwrite=True,
                 residues=['Alkane', 'Water'])
```

In 21 lines of code, a TRUE system for generating solvated boxes of alkanes in water according to length of the alkane, number of alkanes, number of water molecules, and target density has been generated. This structure is then atom typed by Foyer, ready to be simulated under whichever engine of choice. In the above example, the engine of choice is GROMACS[11].



Additional Information

We are continually adding new functionality to the MoSDeF toolkit and encourage collaboration with new researchers and developers. MoSDeF is hosted on GitHub under an MIT open source license. Issues, feature requests, and contributions are always welcome.

MoSDeF can also be tied together with workflow managers to improve the capacity at which screening simulations can be executed. Large scale screening studies[12,13] have been successfully completed using the MoSDeF tools and the *signac*[14]and *signac-flow*[15] workflow managers.

References

1. G. Sandve, A. Nekrutenko, J. Taylor, E. Hovig, "Ten Simple Rules for Reproducible Computational Research." PLoS Comput. Biol. 2013, 9 (10), e1003285.
2. https://github.com/mosdef-hub
3. C. Klein, J. Sallai, T. J. Jones, C. R. Iacovella, C. McCabe, and P. T. Cummings, "A Hierarchical, Component Based Approach to Screening Properties of Soft Matter," in Foundations of Molecular Modeling and Simulation, 2016, pp. 79–92.
4. https://github.com/mosdef-hub/mbuild
5. C. Klein, A. Z. Summers, M. W. Thompson, J. B. Gilmer, C. McCabe, and P. T. Cummings, "Formalizing Atom-typing and the Dissemination of Force Fields with Foyer," in Computational Materials Science, 2019, pp. 215-227.
6. https://github.com/mosdef-hub/foyer
7. https://github.com/GOMC-WSU/GOMC
8. https://cassandra.nd.edu
9. http://www.daylight.com/dayhtml/doc/theory/theory.smarts.html
10. Eastman, P., Friedrichs, M. S., Chodera, J. D., Radmer, R. J., Bruns, C. M., Ku, J. P., … Pande, V. S. (2013). OpenMM 4: A reusable, extensible, hardware independent library for high performance molecular simulation. Journal of Chemical Theory and Computation, 9(1), 461–469.
11. Abraham, M. J., Murtola, T., Schulz, R., Páll, 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
12. M. W. Thompson, R. A. Matsumoto, R. L. Sacci, N. C. Sanders, and P. T. Cummings, "Scalable Screening of Soft Matter: A Case Study of Ionic Liquids and Organic Solvents" J. Phys. Chem. B. 2019, 123, pp. 1340-1347.
13. A. Z. Summers, J. B. Gilmer, C. R. Iacovella, P. T. Cummings, and C. McCabe, "MoSDeF, a Python Framework Enabling Large-Scale Computational Screening of Soft Matter: Application to Chemistry-Property Relationships in Lubricating Monolayer Films." J. Chem. Theory Comput., 2020.
14. Adorf, C. S., Dodd, P. M., Ramasubramani, V., & Glotzer, S. C. (2018). Simple data and workflow management with the signac framework. Computational Materials Science, 146, 220–229. https://doi.org/10.1016/j.commatsci.2018.01.035
15. C. S. Adorf and P. Dodd, "Signac-Flow." [Online]. Available: https://bitbucket.org/glotzer/signac-flow.

NSF

Award # 1835874

mosdef.org