Coarse-grained OpenMM Documentation

Release 0.0.1

Garrett A. Meek

Research group of Michael R. Shirts

Dept. of Chemical and Biological Engineering
University of Colorado Boulder

Aug 27, 2019

CONTENTS

1 Building OpenMM objects for coarse grained modeling **2**
- 1.1 Building an OpenMM System() and Topology() 2
- 1.2 Other tools for building and verifying the OpenMM System() and Topology() . . . 3

2 OpenMM simulation tools for coarse grained modeling **6**
- 2.1 Building OpenMM simulation objects 6
- 2.2 Building and running Yank replica exchange simulations 7
- 2.3 Plotting tools 9
- 2.4 Other simulation tools 10

3 Utilities for coarse grained modeling in OpenMM **13**

4 Indices and tables **14**

Python Module Index **15**

Index **16**

This documentation is generated automatically using Sphinx, which reads all docstring-formatted comments from Python functions in the ‘cg_openmm’ repository. (See cg_openmm/doc for Sphinx source files.)

CHAPTER

ONE

BUILDING OPENMM OBJECTS FOR COARSE GRAINED MODELING

1.1 Building an OpenMM System() and Topology()

All OpenMM simulations require a System() and a Topology().

Listed below are functions and classes that aid the building of OpenMM System() and Topology() class objects for coarse grained models with user-defined properties:

`build.cg_build.`**`build_system`**(*cgmodel*)

Builds an OpenMM System() object, given a CGModel() as input.

Parameters **`cgmodel`** (*`class`*) – CGModel() class object

Returns OpenMM System() object

`build.cg_build.`**`build_topology`**(*cgmodel*, *use_pdbfile=False*, *pdbfile=None*)

Construct an OpenMM Topology() class object for our coarse grained model,

Parameters

- **`cgmodel`** (*`class`*) – CGModel() class object
- **`use_pdbfile`** (*`Logical`*) – Determines whether or not to use a PDB file in order to generate the Topology().

Warning: When 'use_pdbfile'=True, this function will use the PDBFile() class object from OpenMM to build the Topology(). In order for this approach to function correctly, the particle names in the PDB file must match the particle names in the coarse grained model.

1.2 Other tools for building and verifying the OpenMM System() and Topology()

Shown below are other functions/tools to build and verify the System/Topology:

`build.cg_build.`**`add_force`**(*cgmodel*, *force_type=None*)

Given a 'cgmodel' and 'force_type' as input, this function adds the OpenMM force corresponding to 'force_type' to 'cgmodel.system'.

Parameters

- **`cgmodel`** – CGModel() class object.
- **`type`** – class
- **`force_type`** (*`str`*) – Designates the kind of 'force' provided. (Valid options include: "Bond", "Nonbonded", "Angle", and "Torsion")

Returns CGModel() class object

Return type class

Returns An OpenMM Force() object.

Return type

`build.cg_build.`**`add_new_elements`**(*cgmodel*)

Add coarse grained particle types to OpenMM.

Parameters **`cgmodel`** (*`class`*) – CGModel object (contains all attributes for a coarse grained model).

Returns particle_list: a list of the particles that were added to OpenMM's 'Element' List.

Return type list

Example

```
>>> particle_types = add_new_elements(cgmodel)
```

Warning: If the particle names were user defined, and any of the names conflict with existing element names in OpenMM, OpenMM will issue an error exit.

`build.cg_build.`**`get_num_forces`**(*cgmodel*)

Given a CGModel() class object, this function determines how many forces we are including when evaluating the energy.

Parameters **`cgmodel`** (*`class`*) – CGModel() class object

Returns Number of forces

Return type int

`build.cg_build.`**`test_force`**(*cgmodel*, *force*, *force_type=None*)

Given an OpenMM Force(), this function determines if there are any problems with its configuration.

Parameters

- **`cgmodel`** (*`class`*) – CGModel() class object.
- **`force`** (*`class`*) – An OpenMM Force() object.
- **`force_type`** (*`str`*) – Designates the kind of 'force' provided. (Valid options include: "Nonbonded")

Returns 'success', a variable designating whether or not the force test passed.

Return type Logical

`build.cg_build.`**`test_forces`**(*cgmodel*)

Given a cgmodel that contains positions and an an OpenMM System() object, this function tests the forces for cgmodel.system.

More specifically, this function confirms that the model does not have any "NaN" or unphysically large forces.

Parameters

- **`cgmodel`** – CGModel() class object.
- **`type`** – class

Returns success: Indicates if this cgmodel has unphysical forces.

Return type Logical

`build.cg_build.`**`verify_system`**(*cgmodel*)

Given a CGModel() class object, this function confirms that its OpenMM System() object is configured correctly.

Parameters **`cgmodel`** (*`class`*) – CGModel() class object

Warning: The function will force an error exit if the system is invalid, and will proceed as normal if the system is valid.

`build.cg_build.`**`verify_topology`**(*cgmodel*)

Given a coarse grained model that contains a Topology() (cgmodel.topology), this function verifies the validity of the topology.

Parameters **`cgmodel`** (*`class`*) – CGModel() class object.

> **Warning:** The function will force an error exit if the topology is invalid, and will proceed as normal if the topology is valid.

`build.cg_build.`**`write_xml_file`**(*cgmodel*, *xml_file_name*)

Write an XML-formatted forcefield file for a coarse grained model.

Parameters

- **`cgmodel`** (*`class`*) – CGModel() class object.
- **`xml_file_name`** (*`str`*) – Path to XML output file.

CHAPTER

TWO

OPENMM SIMULATION TOOLS FOR COARSE GRAINED MODELING

2.1 Building OpenMM simulation objects

OpenMM simulations are propagated using a Simulation() object.

Shown below are the main tools needed to build OpenMM Simulaton() objects for coarse grained modeling.

`simulation.tools.`**`build_mm_simulation`**(*topology*, *system*, *positions*, *temperature=Quantity(value=300.0, unit=kelvin)*, *simulation_time_step=None*, *total_simulation_time=Quantity(value=1.0, unit=picosecond)*, *output_pdb=None*, *output_data=None*, *print_frequency=100*, *test_time_step=False*)

Construct an OpenMM simulation object for our coarse grained model.

topology: OpenMM topology object

system: OpenMM system object

positions: Array containing the positions of all beads in the coarse grained model (np.array('num_beads' x 3 , (float * simtk.unit.distance))

temperature: Simulation temperature (float * simtk.unit.temperature)

simulation_time_step: Simulation integration time step (float * simtk.unit.time)

total_simulation_time: Total simulation time (float * simtk.unit.time)

output_pdb: Name of output file where we will write the coordinates for this simulation run (string)

output_data: Name of output file where we will write the data from this simulation (string) print_frequency: Number of simulation steps to skip when writing data to 'output_data' (integer)

test_time_step: Logical variable determining whether or not the user-provided time step will be tested prior to a full simulation run (Logical) Default = False

`simulation.tools.`**`run_simulation`**(*cgmodel*, *output_directory*, *total_simulation_time*, *simulation_time_step*, *temperature*, *print_frequency*, *output_pdb=None*, *output_data=None*)

Run OpenMM() simulation

cgmodel: CGModel() class object

output_directory: Output directory within which to place the output files from this simulation run.

total_simulation_time: The total amount of time for which we will run this simulation

simulation_time_step: The time step for the simulation run.

temperature: The temperature for the simulation run.

print_frequency: The number of steps to take when writing simulation data to an output file.

2.2 Building and running Yank replica exchange simulations

The Yank python package is used to perform replica exchange sampling with OpenMM simulations.

Shown below are the main functions and tools necessary to conduct Yank replica exchange simulations with a coarse grained model in OpenMM.

`simulation.rep_exch.`**`run_replica_exchange`**(*topology, system, positions, temperature_list=[Quantity(value=250.0, unit=kelvin), Quantity(value=260.0, unit=kelvin), Quantity(value=270.0, unit=kelvin), Quantity(value=280.0, unit=kelvin), Quantity(value=290.0, unit=kelvin), Quantity(value=300.0, unit=kelvin), Quantity(value=310.0, unit=kelvin), Quantity(value=320.0, unit=kelvin), Quantity(value=330.0, unit=kelvin), Quantity(value=340.0, unit=kelvin)], simulation_time_step=None, total_simulation_time=Quantity(value=1.0, unit=picosecond), output_data='output.nc', print_frequency=100, verbose_simulation=False, exchange_attempts=None, test_time_step=False*)

Construct an OpenMM simulation object for our coarse grained model.

Parameters

- **`topology`** (*`OpenMM Topology() class object`*) – An OpenMM object which contains information about the bonds and constraints in a molecular model
- **`system`** (*`OpenMM System() class object`*) – An OpenMM object which contains information about the forces and particle properties in a molecular model
- **`positions`** (*`np.array( 'num_beads' x 3 , ( float * simtk.unit.distance ) )`*) – Contains the positions for all particles in a model
- **`temperature_list`** – List of temperatures for which to perform

replica exchange simulations, default = [(300.0 * unit.kelvin).__add__(i * unit.kelvin) for i in range(-20,100,10)]

- **simulation_time_step** (*float * simtk.unit*) – Simulation integration time step, default = None
- **total_simulation_time** (*float * simtk.unit*) – Total simulation time
- **output_data** (*string*) – Name of NETCDF file where we will write data from replica exchange simulations

replica_energies: List(List(float * simtk.unit.energy for simulation_steps) for num_replicas)
List of dimension num_replicas X simulation_steps, which gives the energies for all replicas at all simulation steps

replica_positions: np.array((float * simtk.unit.positions for num_beads) for simulation_steps)
List of positions for all output frames for all replicas

replica_state_indices: np.array((float for num_replicas) for exchange_attempts)
List of thermodynamic state assignment labels for each replica during each stage of the replica exchange simulation run.

temperature_list: List(float * simtk.unit.kelvin for num_replicas) List of the temperatures for each replica.

simulation.rep_exch.**read_replica_exchange_data**(*system=None*, *topology=None*, *temperature_list=None*, *output_data='output.nc'*, *print_frequency=None*)

simulation.rep_exch.**make_replica_pdb_files**(*topology*, *replica_positions*)

2.3 Plotting tools

Shown below are functions which allow plotting of simulation results.

simulation.rep_exch.**plot_replica_exchange_energies**(*replica_energies*, *temperature_list*, *simulation_time_step*, *steps_per_stage=1*, *file_name='replica_exchange_energ*
legend=True)

`simulation.rep_exch.`**`plot_replica_exchange_summary`**(*replica_states*, *temperature_list*, *simulation_time_step*, *steps_per_stage=1*, *file_name='replica_exchange_state_t*, *legend=True*)

`simulation.tools.`**`plot_simulation_data`**(*simulation_times*, *y_data*, *plot_type=None*)

`simulation.tools.`**`plot_simulation_results`**(*simulation_data_file*, *plot_output_directory*, *simulation_time_step*)

2.4 Other simulation tools

Shown below are other tools which aid the building and verification of OpenMM simulation objects.

`simulation.tools.`**`get_mm_energy`**(*topology*, *system*, *positions*)

Get the OpenMM potential energy for a system, given a topology and set of positions.

topology: OpenMM topology object

system: OpenMM system object

positions: Array containing the positions of all beads in the coarse grained model (np.array('num_beads' x 3 , (float * simtk.unit.distance))

`simulation.tools.`**`get_simulation_time_step`**(*topology*, *system*, *positions*, *temperature*, *total_simulation_time*, *time_step_list=None*)

Determine a suitable simulation time step.

Parameters

- **`topology`** (Topology()) – OpenMM Topology
- **`system`** (System()) – OpenMM System()
- **`positions`** ('unit.Quantity() <http://docs.openmm.org/development/api-python/generated/simtk.unit.quantity.Quantity.html>'_(np.array([cgmodel.num_beads,3]),simtk.unit)) – Positions array for the model we would like to test
- **`temperature`** (SIMTK Unit()) – Simulation temperature

- **total_simulation_time** – Total run time for individual simulations
- **time_step_list** (*List, default = None*) – List of time steps for which to attempt a simulation in OpenMM.

Returns A successfully-tested time step

Return type

SIMTK Unit()

`simulation.tools.`**`minimize_structure`**(*topology, system, positions, temperature=Quantity(value=0.0, unit=kelvin), simulation_time_step=None, total_simulation_time=Quantity(value=1.0, unit=picosecond), output_pdb=None, output_data=None, print_frequency=1*)

Perform a minimization of the potential energy for our coarse grained model.

Parameters

- **topology** (*Topology()*) – OpenMM topology for the coarse grained model
- **system** (*System()*) – OpenMM system
- **positions** (‘unit.Quantity() <http://docs.openmm.org/development/api-python/generated/simtk.unit.quantity.Quantity.html>‘_(np.array([cgmodel.num_beads,3]),simtk.unit)) – Positions array for the model we would like to test
- **temperature** – Simulation temperature
- **total_simulation_time** – Total run time for individual simulations
- **output_pdb** (*str*) – Output destinaton for PDB-formatted coordinates during the simulation
- **output_data** (*str*) – Output destination for simulation data
- **print_frequency** (*int*) – Number of simulation steps to skip when writing data

Returns positions: Minimized positions

Return type positions: ‘unit.Quantity() <http://docs.openmm.org/development/api-python/generated/simtk.unit.quantity.Quantity.html>‘_(np.array([cgmodel.num_beads,3]),simtk.unit)

Returns potential_energy: Potential energy for the minimized structure.

Return type potential_energy: ‘unit.Quantity() <http://docs.openmm.org/development/api-python/generated/simtk.unit.quantity.Quantity.html>‘_(np.array([cgmodel.num_beads,3]),simtk.unit)

`simulation.rep_exch.`**`get_minimum_energy_pose`**(*topology*, *replica_energies*, *replica_positions*, *file_name=None*)

CHAPTER

THREE

UTILITIES FOR COARSE GRAINED MODELING IN OPENMM

This page details the functionality of utilities in cg_openmm/src/utilities/util.py.

`utilities.util.`**`distance`**(*positions_1*, *positions_2*)

Construct a matrix of the distances between all particles.

positions_1: Positions for a particle (np.array(length = 3))

positions_2: Positions for a particle (np.array(length = 3))

distance (float * unit)

`utilities.util.`**`get_box_vectors`**(*box_size*)

Assign all side lengths for simulation box.

box_size: Simulation box length (float * simtk.unit.length)

`utilities.util.`**`lj_v`**(*positions_1*, *positions_2*, *sigma*, *epsilon*)

Given two sets of input coordinates, this function computes their Lennard-Jones interaction potential energy.

positions_1

`utilities.util.`**`set_box_vectors`**(*system*, *box_size*)

Build a simulation box.

system: OpenMM system object

box_size: Simulation box length (float * simtk.unit.length)

CHAPTER FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

b

build.cg_build, 3

s

simulation.rep_exch, 12
simulation.tools, 10

u

utilities.util, 13

INDEX

A

add_force() (*in module build.cg_build*), 3
add_new_elements() (*in module build.cg_build*), 3

B

build.cg_build (*module*), 2, 3
build_mm_simulation() (*in module simulation.tools*), 6
build_system() (*in module build.cg_build*), 2
build_topology() (*in module build.cg_build*), 2

D

distance() (*in module utilities.util*), 13

G

get_box_vectors() (*in module utilities.util*), 13
get_minimum_energy_pose() (*in module simulation.rep_exch*), 12
get_mm_energy() (*in module simulation.tools*), 10
get_num_forces() (*in module build.cg_build*), 3
get_simulation_time_step() (*in module simulation.tools*), 10

L

lj_v() (*in module utilities.util*), 13

M

make_replica_pdb_files() (*in module simulation.rep_exch*), 9
minimize_structure() (*in module simulation.tools*), 11

P

plot_replica_exchange_energies() (*in module simulation.rep_exch*), 9
plot_replica_exchange_summary() (*in module simulation.rep_exch*), 9
plot_simulation_data() (*in module simulation.tools*), 10
plot_simulation_results() (*in module simulation.tools*), 10

R

read_replica_exchange_data() (*in module simulation.rep_exch*), 9
run_replica_exchange() (*in module simulation.rep_exch*), 7
run_simulation() (*in module simulation.tools*), 7

S

set_box_vectors() (*in module utilities.util*), 13
simulation.rep_exch (*module*), 7, 9, 12
simulation.tools (*module*), 6, 10

T

test_force() (*in module build.cg_build*), 4
test_forces() (*in module build.cg_build*), 4

U

utilities.util (*module*), 13

V

verify_system() (*in module build.cg_build*), 4

verify_topology() (*in module build.cg_build*), 4

W

write_xml_file() (*in module build.cg_build*), 5