Coarse-grained OpenMM Documentation

Release 0.0.1

Garrett A. Meek

Michael R. Shirts

Dept. of Chemical and Biological Engineering
University of Colorado Boulder

Sep 04, 2019

CONTENTS

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:

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:

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*)

Build an OpenMM Simulation()

Parameters

- **topology** (Topology()) – OpenMM Topology()
- **system** (System()) – OpenMM System()
- **positions** (‘unit.Quantity() <https://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, default = 300.0 K
- **simulation_time_step** – Simulation integration time step

- **total_simulation_time** – Total run time for individual simulations
- **output_pdb** (*str*) – Output destination for PDB coordinates, Default = None
- **output_data** (*str*) – Output destination for non-coordinate simulation data, Default = None
- **print_frequency** – Number of simulation steps to skip when writing to output, Default = 100
- **test_time_step** (*Logical*) – Logical variable determining if a test of the time step will be performed, Default = False

Returns OpenMM Simulation() object

Return type

Simulation()

Example

```
>>> from simtk import unit
>>> from foldamers.cg_model.cgmodel import CGModel
>>> cgmodel = CGModel()
>>> topology = cgmodel.topology
>>> system = cgmodel.system
>>> positions = cgmodel.positions
>>> temperature = 300.0 * unit.kelvin
>>> simulation_time_step = 5.0 * unit.femtosecond
>>> total_simulation_time= 1.0 * unit.picosecond
>>> output_pdb = "output.pdb"
>>> output_data = "output.dat"
>>> print_frequency = 20
>>> openmm_simulation = build_mm_simulation(topology,system,
 ↪positions,temperature=temperature,simulation_time_
 ↪step=simulation_time_step,total_simulation_time=total_simulation_
 ↪time,output_pdb=output_pdb,output_data=output_data,print_
 ↪frequency=print_frequency,test_time_step=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

Parameters

- **cgmodel** (*class*) – CGModel() object
- **output_directory** (*str*) – Output directory for simulation data

- **total_simulation_time** – Total run time for individual simulations
- **simulation_time_step** – Simulation integration time step
- **temperature** – Simulation temperature, default = 300.0 K
- **print_frequency** – Number of simulation steps to skip when writing to output, Default = 100

Example

```
>>> import os
>>> from simtk import unit
>>> from foldamers.cg_model.cgmodel import CGModel
>>> cgmodel = CGModel()
>>> topology = cgmodel.topology
>>> system = cgmodel.system
>>> positions = cgmodel.positions
>>> temperature = 300.0 * unit.kelvin
>>> simulation_time_step = 5.0 * unit.femtosecond
>>> total_simulation_time= 1.0 * unit.picosecond
>>> output_directory = os.getcwd()
>>> output_pdb = "output.pdb"
>>> output_data = "output.dat"
>>> print_frequency = 20
>>> run_simulation(cgmodel,output_directory,total_simulation_time,
↪simulation_time_step,temperature,print_frequency,output_
↪pdb=output_pdb,output_data=output_data)
```

Warning: When run with default options this subroutine is capable of producing a large number of output files. For example, by default this subroutine will plot the simulation data that is written 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*)

Run a Yank replica exchange simulation using an OpenMM coarse grained model.

Parameters

- **topology** – OpenMM Topology
- **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_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** – Simulation integration time step

- **total_simulation_time** – Total run time for individual simulations
- **output_data** (*string*) – Name of NETCDF file where we will write simulation data
- **print_frequency** – Number of simulation steps to skip when writing to output, Default = 100
- **verbose_simulation** (*Logical*) – Determines how much output is printed during a simulation run. Default = False
- **exchange_attempts** (*int*) – Number of exchange attempts to make during a replica exchange simulation run, Default = None
- **test_time_step** (*Logical*) – Logical variable determining if a test of the time step will be performed, Default = False

Returns replica_energies: The potential energies for all replicas at all (printed) time steps

Return type replica_energies: ‘unit.Quantity() <http://docs.openmm.org/development/api-python/generated/simtk.unit.quantity.Quantity.html>‘_(np.float([number_replicas,number_simulation_steps]),simtk.unit)

Returns replica_positions: The positions for all replicas at all (printed) time steps

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

Returns replica_state_indices: The thermodynamic state assignments for all replicas at all (printed) time steps

Return type replica_state_indices: ‘unit.Quantity() <http://docs.openmm.org/development/api-python/generated/simtk.unit.quantity.Quantity.html>‘_(np.int64([number_replicas,number_simulation_steps]),simtk.unit)

Example

```
>>> from foldamers.cg_model.cgmodel import CGModel
>>> from cg_openmm.simulation.rep_exch import *
>>> cgmodel = CGModel()
>>> replica_energies,replica_positions,replica_state_indices = run_
↪replica_exchange(cgmodel.topology,cgmodel.system,cgmodel.
↪positions)
```

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

Read replica exchange simulation data.

Parameters

- **system** – OpenMM system object, default = None
- **topology** – OpenMM topology object, default = None
- **temperature_list** – List of temperatures that will be used to define different replicas (thermodynamics states), default = None
- **output_data** (*str*) – Path to the output data for a Yank, NetCDF-formatted file containing replica exchange simulation data, default = None
- **print_frequency** (*int*) – Number of simulation steps to skip when writing data, default = None

Returns replica_energies: The potential energies for all replicas at all (printed) time steps

Return type replica_energies: 'unit.Quantity() <http://docs.openmm.org/development/api-python/generated/simtk.unit.quantity.Quantity.html>'_(np.float([number_replicas,number_simulation_steps]),simtk.unit)

Returns replica_positions: The positions for all replicas at all (printed) time steps

Return type replica_positions: 'unit.Quantity() <http://docs.openmm.org/development/api-python/generated/simtk.unit.quantity.Quantity.html>'_(np.float([number_replicas,number_simulation_steps,cgmodel.num_beads,3]),simtk.unit

Returns replica_state_indices: The thermodynamic state assignments for all replicas at all (printed) time steps

Return type replica_state_indices: 'unit.Quantity() <http://docs.openmm.org/development/api-python/generated/simtk.unit.quantity.Quantity.html>'_(np.int64([number_replicas,number_simulation_steps]),simtk.unit)

Example

```
>>> from foldamers.cg_model.cgmodel import CGModel
>>> from cg_openmm.simulation.rep_exch import *
>>> cgmodel = CGModel()
>>> replica_energies,replica_positions,replica_state_indices = run_
↪replica_exchange(cgmodel.topology,cgmodel.system,cgmodel.
↪positions)
```

(continues on next page)

(continued from previous page)

```
>>> replica_energies,replica_positions,replica_state_indices =␣
↪read_replica_exchange_data(system=cgmodel.system,
↪topology=cgmodel.topology,temperature_list=,output_data="output.
↪nc",print_frequency=None)
```

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

Make PDB files from replica exchange simulation trajectory data

Parameters

- **topology** – OpenMM Topology
- **replica_positions** (‘unit.Quantity() <http://docs.openmm.org/development/api-python/generated/simtk.unit.quantity.Quantity.html>‘_(np.array([n_replicas,cgmodel.num_beads,3]),simtk.unit)) – Positions array for the replica exchange data for which we will write PDB files

Returns file_list: A list of names for the files that were written.

Return type List(str)

Example

```
>>> from foldamers.cg_model.cgmodel import CGModel
>>> from cg_openmm.simulation.rep_exch import *
>>> cgmodel = CGModel()
>>> replica_energies,replica_positions,replica_state_indices = run_
↪replica_exchange(cgmodel.topology,cgmodel.system,cgmodel.
↪positions)
>>> pdb_file_list = make_replica_pdb_files(cgmodel.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)

Plot the potential energies for a batch of replica exchange trajectories

Parameters

- **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
- **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** – Simulation integration time step
- **steps_per_stage** (*int*) – The number of simulation steps for individual replica "stages" (period of time between state exchanges), default = 1
- **file_name** (*str*) – The pathname of the output file for plotting results, default = "replica_exchange_energies.png"
- **legend** (*Logical*) – Controls whether a legend is added to the plot

..warning:: If more than 10 replica exchange trajectories are provided as input data, by default, this function will only plot the first 10 thermodynamic states. These thermodynamic states are chosen based upon their indices, not their instantaneous temperature (ensemble) assignment.

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)

Plot the thermodynamic state assignments for individual temperature replicas as a function of the simulation time, in order to obtain a visual summary of the replica exchanges from a Yank simulation.

Parameters

- **replica_states** (*List(List(float * simtk.unit. energy for simulation_steps) for num_replicas)*) – List of dimension num_replicas X simulation_steps, which gives the thermodynamic state indices for all replicas at all simulation steps
- **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** – Simulation integration time step
- **steps_per_stage** (*int*) – The number of simulation steps for individual replica "stages" (period of time between state exchanges), default = 1
- **file_name** (*str*) – The pathname of the output file for plotting results, default = "replica_exchange_state_transitions.png"
- **legend** (*Logical*) – Controls whether a legend is added to the plot

..warning:: If more than 10 replica exchange trajectories are provided as input data, by default, this function will only plot the first 10 thermodynamic states. These thermodynamic states are chosen based upon their indices, not their instantaneous temperature (ensemble) assignment.

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

Plot simulation data.

Parameters

- **simulation_times** (*List*) – List of simulation times (x data)
- **y_data** (*List*) – List of simulation data
- **plot_type** (*str*) – Form of data to plot, Default = None, Valid options include: "Potential Energy", "Kinetic Energy", "Total Energy", "Temperature"

Example

```
>>> import os
>>> from simtk import unit
>>> simulation_data_file = "output.pdb"
>>> simulation_time_step = 5.0 * unit.femtosecond
>>> simulation_data = read_simulation_data(simulation_data_file,
 ↪simulation_time_step)
>>> simulation_times = simulation_data["Simulation Time"]
>>> y_data = simulation_data["Potential Energy"]
>>> plot_type = "Potential Energy"
>>> plot_simulation_data(simulation_times,y_data,plot_type=
 ↪"Potential Energy")
```

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

Plot all data from an OpenMM output file

Parameters

- **simulation_data_file** (*str*) – Path to file containing simulation data
- **plot_output_directory** (*str*) – Path to folder where plotting results will be written.
- **simulation_time_step** – Simulation integration time step

Example

```
>>> import os
>>> from simtk import unit
>>> simulation_data_file = "output.pdb"
>>> plot_output_directory = os.getcwd()
>>> simulation_time_step = 5.0 * unit.femtosecond
>>> 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.

Parameters

- **topology** – OpenMM Topology()
- **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

Returns potential_energy: The potential energy for the model with the provided positions.

Return type ‘unit.Quantity() <http://docs.openmm.org/development/api-python/generated/simtk.unit.quantity.Quantity.html>‘_(float,simtk.unit)

Example

```
>>> from foldamers.cg_model.cgmodel import CGModel
>>> cgmodel = CGModel()
>>> topology = cgmodel.topology
```

(continues on next page)

(continued from previous page)

```
>>> system = cgmodel.system
>>> positions = cgmodel.positions
>>> openmm_potential_energy = get_mm_energy(topology,system,
↪positions)
```

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** – OpenMM Topology
- **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
- **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()

Example

```
>>> from simtk import unit
>>> from foldamers.cg_model.cgmodel import CGModel
>>> cgmodel = CGModel()
>>> topology = cgmodel.topology
>>> system = cgmodel.system
>>> positions = cgmodel.positions
>>> temperature = 300.0 * unit.kelvin
>>> total_simulation_time = 1.0 * unit.picosecond
>>> time_step_list = [1.0 * unit.femtosecond, 2.0 * unit.
↪femtosecond, 5.0 * unit.femtosecond]
>>> best_time_step,max_force_tolerance = get_simulation_time_
↪step(topology,system,positions,temperature,total_simulation_time,
↪time_step_list=time_step_list)
```

(continues on next page)

(continued from previous page)

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*)

Minimize the potential energy

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** – 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, default = 1

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)

Example

```
>>> from simtk import unit
>>> from foldamers.cg_model.cgmodel import CGModel
>>> cgmodel = CGModel()
>>> topology = cgmodel.topology
>>> system = cgmodel.system
>>> positions = cgmodel.positions
>>> temperature = 300.0 * unit.kelvin
>>> total_simulation_time = 1.0 * unit.picosecond
>>> simulation_time_step = 1.0 * unit.femtosecond
>>> output_pdb = "output.pdb"
>>> output_data = "output.dat"
>>> print_frequency = 20
>>> minimum_energy_structure,potential_energy,openmm_simulation_
↪object = minimize_structure(topology,system,positions,
↪temperature=temperature,simulation_time_step=simulation_time_
↪step,total_simulation_time=total_simulation_time,output_
↪pdb=output_pdb,output_data=output_data,print_frequency=print_
↪frequency)
```

simulation.tools.**read_simulation_data**(*simulation_data_file*, *simulation_time_step*)

Read OpenMM simulation data

Parameters

- **simulation_data_file** (*str*) – Path to file that will be read
- **simulation_time_step** – Time step to apply for the simulation data

Example

```
>>> from simtk import unit
>>> simulation_data_file = "output.dat"
>>> simulation_time_step = 5.0 * unit.femtosecond
>>> data = read_simulation_data(simulation_data_file,simulation_
↪time_step)
```

simulation.rep_exch.**get_minimum_energy_ensemble**(*topology*, *replica_energies*, *replica_positions*, *ensemble_size=5*, *file_name=None*)

Get an ensemble of low (potential) energy poses, and write the lowest energy structure to a PDB file if a file_name is provided.

Parameters

- **topology** – OpenMM Topology()

- **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
- **file_name** – Output destination for PDB coordinates of minimum energy pose, Default = None

Returns ensemble: A list of poses that are in the minimum energy ensemble.

Return type List (np.array((float * simtk.unit.positions for num_beads) for simulation_steps))

Example

```
>>> from foldamers.cg_model.cgmodel import CGModel
>>> from cg_openmm.simulation.rep_exch import *
>>> cgmodel = CGModel()
>>> replica_energies,replica_positions,replica_state_indices = run_
↪replica_exchange(cgmodel.topology,cgmodel.system,cgmodel.
↪positions)
>>> ensemble_size = 5
>>> file_name = "minimum.pdb"
>>> minimum_energy_ensemble = get_minimum_energy_ensemble(cgmodel.
↪topology,replica_energies,replica_positions,ensemble_
↪size=ensemble_size,file_name=file_name)
```

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*)

Calculate the distance between two particles, given their positions.

Parameters

- **positions_1** (‘unit.Quantity() <http://docs.openmm.org/development/api-python/generated/simtk.unit.quantity.Quantity.html>‘_(np.array([3]),simtk.unit)) – Positions for the first particle
- **positions_2** (‘unit.Quantity() <http://docs.openmm.org/development/api-python/generated/simtk.unit.quantity.Quantity.html>‘_(np.array([3]),simtk.unit)) – Positions for the first particle

Returns distance: Distance between two particles

Return type ‘unit.Quantity() <http://docs.openmm.org/development/api-python/generated/simtk.unit.quantity.Quantity.html>‘_(float,simtk.unit)

Example

```
>>> from foldamers.cg_model.cgmodel import CGModel
>>> cgmodel = CGModel()
>>> particle_1_coordinates = cgmodel.positions[0]
>>> particle_2_coordinates = cgmodel.positions[1]
>>> particle_distance = distance(particle_1_coordinates,particle_2_
↪coordinates)
```

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

Given a simulation box length, construct a vector.

Parameters **box_size** (‘unit.Quantity() <http://docs.openmm.org/development/api-python/generated/simtk.unit.quantity.Quantity.html>‘_(float,simtk.unit)) – Length of individual sides of a simulation box

Returns box_vectors: Vectors to use when defining an OpenMM simulation box.

Return type box_vectors: List(‘unit.Quantity() <http://docs.openmm.org/development/api-python/generated/simtk.unit.quantity.Quantity.html>‘_(float,simtk.unit))

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

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

Impose a set of simulation box vectors on an OpenMM simulation object.

Parameters

- **system** (System()) – OpenMM System()
- **box_size** (‘unit.Quantity() <http://docs.openmm.org/development/api-python/generated/simtk.unit.quantity.Quantity.html>‘_(float,simtk.unit)) – Length of individual sides of a simulation box

Returns system: OpenMM system object

Return type

System()

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

U

INDEX