foldamers Documentation

Release 0.0

Shirts research group

Garrett A. Meek
Lenny T. Fobe
Michael R. Shirts

Dept. of Chemical and Biological Engineering
University of Colorado Boulder

May 12, 2019

CONTENTS

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

CHAPTER

ONE

COARSE GRAINED MODEL UTILITIES

This page details the functions and classes in src/cg_model/cgmodel.py

1.1 'CGModel' class for OpenMM simulation

Shown below is a detailed description of the 'cgmodel' class object, which contains all information about a coarse grained model.

1.2 Other coarse grained model utilities

class cg_model.cgmodel.**CGModel**(*positions=None, polymer_length=12, backbone_length=1, sidechain_length=1, sidechain_positions=[0], masses=Quantity(value=12.0, unit=dalton), sigma=Quantity(value=8.4, unit=angstrom), epsilon=Quantity(value=0.5, unit=kilocalorie/mole), bond_lengths=Quantity(value=1.0, unit=angstrom), bond_force_constants=990000.0, charges=Quantity(value=0.0, unit=elementary charge), constrain_bonds=False, include_bond_forces=True, include_nonbonded_forces=True, include_bond_angle_forces=True, include_torsion_forces=True, check_energy_conservation=True*)

Construct a coarse grained model.

positions: Positions for all of the particles, default = None

polymer_length: Number of monomer units (integer), default = 8

backbone_length: Number of beads in the backbone portion of each (individual) monomer (integer), default = 1

sidechain_length: Number of beads in the sidechain portion of each (individual) monomer (integer), default = 1

sidechain_positions: List of integers defining the backbone bead indices upon which we will place the sidechains, default = [0] (Place a sidechain on the backbone bead with index "0" (first backbone bead) in each (individual) monomer

masses: Masses of all particle types (List ([[Backbone masses], [Sidechain masses]])) default = [[12.0 * unit.amu], [12.0 * unit.amu]]

sigma: Non-bonded bead Lennard-Jones equilibrium interaction distance (float * simtk.unit.distance) default = 8.4 * unit.angstrom

epsilon: Non-bonded Lennard-Jones equilibrium interaction strength (float * simtk.unit.energy) default = 0.5 * unit.kilocalorie_per_mole

bond_lengths: Bond lengths for all bond types (float * simtk.unit.distance) default = 1.0 * unit.angstrom

bond_force_constants: Bond force constants for all bond types (float) default = 9.9e5 kJ/mol/A^2

charges: Charges for all beads (float * simtk.unit.charge) default = 0.0 * unit.elementary_charge (for all beads)

num_beads: Total number of particles in the coarse grained model (integer) default = polymer_length * (backbone_length + sidechain_length)

system: OpenMM system object, which stores forces, and can be used to check a model for energy conservation (OpenMM System() class object) default = None

topology: OpenMM topology object, which stores bonds, angles, and other structural attributes of the coarse grained model (OpenMM Topology() class object) default = None

constrain_bonds: Logical variable determining whether bond constraints are applied during a molecular dynamics simulation of the system. (Logical) default = False

bond_list: List of bonds in the coarse grained model (List([[int, int] for # bonds])) default = None

angle_list: List of bond angles that are defined for this coarse grained model (List([[int, int, int] for # bond angles]))

torsion_list: List of torsions that are defined for this coarse grained model List([[int, int, int, int] for # torsions]))

include_bond_forces: Include contributions from bond (harmonic) potentials when calculating the potential energy (Logical) default = True

include_nonbonded_forces: Include contributions from nonbonded interactions when calculating the potential energy (Logical) default = True

include_bond_angle_forces: Include contributions from bond angles when calculating the potential energy (Logical) default = False

include_torsion_forces: Include contributions from torsions when calculating the potential energy (Logical) default = False

polymer_length backbone_length sidechain_length sidechain_positions masses sigma epsilon bond_lengths bond_force_constants charges num_beads positions system topology constrain_bonds bond_list angle_list torsion_list include_bond_forces include_nonbonded_forces include_bond_angle_forces include_torsion_forces

charges = None
Get bond, angle, and torsion lists.

constrain_bonds = None
Make a list of coarse grained particle masses:

get_bond_angle_list()
Construct a list of bond angles for our coarse grained model

get_bond_list()
Construct a bond list for the coarse grained model

get_nonbonded_interaction_list()
Construct a nonbonded interaction list for our coarse grained model

get_torsion_list()
Construct a torsion list for our coarse grained model

cg_model.cgmodel.**add_new_elements**(*cgmodel*, *list_of_masses*)
Adds new coarse grained particle types to OpenMM

cgmodel: CGModel() class object

list_of_masses: List of masses for the particles we want to add to OpenMM

cg_model.cgmodel.**build_system**(*cgmodel*)
Builds an OpenMM System() class object, given a CGModel() class object as input.

cgmodel: CGModel() class object

system: OpenMM System() class object

cg_model.cgmodel.**get_parent_bead**(*cgmodel*, *bead_index*, *backbone_bead_index=None*, *sidechain_bead=False*)
Determines the particle to which a given particle is bonded. (Used for coarse grained model construction.)

cgmodel: CGModel() class object

bead_index: Index of the particle for which we would like to determine the parent particle it is bonded to. (integer) Default = None

backbone_bead_index: If this bead is a backbone bead, this index tells us its index (within a monomer) along the backbone (integer) Default = None

sidechain_bead: Logical variable stating whether or not this bead is in the sidechain. (Logical) Default = False

parent_bead: Index for the particle that 'bead_index' is bonded to. (Integer)

cg_model.cgmodel.**get_particle_masses**(*cgmodel*)

Returns a list of unique particle masses

cgmodel: CGModel() class object

List(unique particle masses)

CHAPTER

TWO

THERMODYNAMIC ANALYSIS TOOLS FOR COARSE GRAINED MODELING

This page details the functions and classes in src/thermo

2.1 Tools to calculate the heat capacity with pymbar

Shown below are functions/tools used in order to calculate the heat capacity with pymbar.

CHAPTER

THREE

UTILITIES FOR THE 'FOLDAMERS' PACKAGE

This page details the functions and classes in src/util.

3.1 Input/Output options (src/utilities/iotools.py)

Shown below is a detailed description of the input/output options for the foldamers package.

utilities.iotools.**write_pdbfile_without_topology**(*CGModel*, *filename*)

Writes the positions in 'CGModel' to the file 'filename'.

CGModel: Coarse grained model class object

filename: Path to the file where we will write PDB coordinates.

3.2 Utilities and random functions (src/utilities/util.py)

utilities.util.**assign_position**(*positions*, *bond_length*, *sigma*, *bead_index*, *parent_index*)

Assign random position for a bead

positions: Positions for all beads in the coarse-grained model. (np.array(num_beads x 3))

bond_length: Bond length for all beads that are bonded, (float * simtk.unit.distance) default = 1.0 * unit.angstrom

positions: Positions for all beads in the coarse-grained model. (np.array(num_beads x 3))

utilities.util.**assign_position_lattice_style**(*cgmodel*, *positions*, *distance_cutoff*, *bead_index*, *parent_index*)

Assign random position for a bead

positions: Positions for all beads in the coarse-grained model. (np.array(num_beads x 3))

bond_length: Bond length for all beads that are bonded, (float * simtk.unit.distance) default = 1.0 * unit.angstrom

positions: Positions for all beads in the coarse-grained model. (np.array(num_beads x 3))

utilities.util.**attempt_lattice_move**(*parent_coordinates*, *bond_length*, *move_direction_list*)

Given a set of cartesian coordinates, assign a new particle a distance of 'bond_length' away in a random direction.

parent_coordinates: Positions for a single particle, away from which we will place a new particle a distance of 'bond_length' away. (np.array(float * unit.angstrom (length = 3)))

bond_length: Bond length for all beads that are bonded, (float * simtk.unit.distance) default = 1.0 * unit.angstrom

trial_coordinates: Positions for a new trial particle (np.array(float * unit.angstrom (length = 3)))

utilities.util.**attempt_move**(*parent_coordinates*, *bond_length*)

Given a set of cartesian coordinates, assign a new particle a distance of 'bond_length' away in a random direction.

parent_coordinates: Positions for a single particle, away from which we will place a new particle a distance of 'bond_length' away. (np.array(float * unit.angstrom (length = 3)))

bond_length: Bond length for all beads that are bonded, (float * simtk.unit.distance) default = 1.0 * unit.angstrom

trial_coordinates: Positions for a new trial particle (np.array(float * unit.angstrom (length = 3)))

utilities.util.**collisions**(*distance_list*, *distance_cutoff*)

Determine whether there are any collisions between non-bonded particles, where a "collision" is defined as a distance shorter than the user-provided 'bond_length'.

distances: List of the distances between all nonbonded particles. (list (float * simtk.unit.distance (length = # nonbonded_interactions)))

bond_length: Bond length for all beads that are bonded, (float * simtk.unit.distance) default = 1.0 * unit.angstrom

collision: Logical variable stating whether or not the model has bead collisions. default = False

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.**distance_matrix**(*positions*)
Construct a matrix of the distances between all particles.

positions: Positions for an array of particles. (np.array(num_particles x 3))

distance_matrix: Matrix containing the distances between all beads. (np.array(num_particles x 3))

utilities.util.**distances**(*interaction_list*, *positions*)
Calculate the distances between a trial particle ('new_coordinates') and all existing particles ('existing_coordinates').

new_coordinates: Positions for a single trial particle (np.array(float * unit.angstrom (length = 3)))

existing_coordinates: Positions for a single trial particle (np.array(float * unit.angstrom (shape = num_particles x 3)))

distances: List of the distances between all nonbonded particles. (list (float * simtk.unit.distance (length = # nonbonded_interactions)))

utilities.util.**first_bead**(*positions*)
Determine if we have any particles in 'positions' Parameters ————- positions: Positions for all beads in the coarse-grained model. (np.array(float * unit (shape = num_beads x 3))) Returns ——- first_bead: Logical variable stating if this is the first particle.

utilities.util.**get_move**(*trial_coordinates*, *move_direction*, *distance*, *bond_length*, *finish_bond=False*)
Given a 'move_direction', a current distance, and a target 'bond_length' (Index denoting x,y,z Cartesian direction), update the coordinates for the particle.

trial_coordinates: positions for a particle (np.array(float * unit.angstrom (length = 3)))

move_direction: Cartesian direction in which we will attempt a particle placement, where: x=0, y=1, z=2. (integer)

distance: Current distance from parent particle (float * simtk.unit.distance)

bond_length: Target bond_length for particle placement. (float * simtk.unit.distance)

finish_bond: Logical variable determining how we will update the coordinates for this particle.

trial_coordinates: Updated positions for the particle (np.array(float * unit.angstrom (length = 3)))

utilities.util.**random_positions**(*cgmodel*, *max_attempts=100*)
Assign random positions for all beads in a coarse-grained polymer.

polymer_length: Number of monomer units (integer), default = 8

backbone_length: Number of beads in the backbone portion of each (individual) monomer (integer), default = 1

sidechain_length: Number of beads in the sidechain portion of each (individual) monomer (integer), default = 1

sidechain_positions: List of integers defining the backbone bead indices upon which we will place the sidechains, default = [0] (Place a sidechain on the backbone bead with index "0" (first backbone bead) in each (individual) monomer

bond_length: Bond length for all beads that are bonded, (float * simtk.unit.distance) default = 1.0 * unit.angstrom

sigma: Non-bonded bead Lennard-Jones interaction distances, (float * simtk.unit.distance) default = 8.4 * unit.angstrom

bb_bond_length: Bond length for all bonded backbone beads, (float * simtk.unit.distance) default = 1.0 * unit.angstrom

bs_bond_length: Bond length for all backbone-sidechain bonds, (float * simtk.unit.distance) default = 1.0 * unit.angstrom

ss_bond_length: Bond length for all beads within a sidechain, (float * simtk.unit.distance) default = 1.0 * unit.angstrom

positions: Positions for all beads in the coarse-grained model. (np.array(num_beads x 3))

utilities.util.**random_sign**(*number*)

Returns 'number' with a random sign.

number: float

number

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

U

INDEX