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 31, 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 Basic 'CGModel' class to build/model coarse grained oligomers

Shown below is a basic 'cgmodel' class object, which requires only a minimal set of model characteristics, applying a set of default values for un-defined parameters.

`cg_model.cgmodel.`**`basic_cgmodel`**(*polymer_length=8, backbone_length=1, sidechain_length=1, sidechain_positions=[0], mass=Quantity(value=12.0, unit=dalton), charge=Quantity(value=0.0, unit=elementary charge), bond_length=Quantity(value=1.0, unit=angstrom), sigma=Quantity(value=2.5, unit=angstrom), epsilon=Quantity(value=0.5, unit=kilocalorie/mole), positions=None*)

Given a minimal set of model parameters, this function creates a cgmodel class object.

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

backbone_length: Integer defining the number of beads in the backbone default = 1

sidechain_length: Integer defining the number of beads in the sidechain default = 1

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

sidechain_positions: List of integers defining the backbone bead indices upon which we will place the sidechains, default = [0]

mass: Mass for all coarse grained beads. default = 12.0 * unit.amu

bond_length: Bond length for all bond types Default = 1.0 * unit.angstrom sigma: Non-bonded bead Lennard-Jones equilibrium interaction distance. default = 2.5 * bond_length

epsilon: Non-bonded Lennard-Jones equilibrium interaction energy default = 0.5 * unit.kilocalorie_per_mole

charge: Charge for all particles default = 0.0 * unit.elementary_charge

positions: Positions for coarse grained particles in the model. default = None

cgmodel: CGModel() class object

1.2 Full 'CGModel' class to build/model coarse grained oligomers

Shown below is a detailed description of the full 'cgmodel' class object.

1.3 Other coarse grained model utilities

class cg_model.cgmodel.**CGModel**(*positions=None, polymer_length=12, backbone_lengths=[1], sidechain_lengths=[1], sidechain_positions=[0], masses={'backbone_bead_masses': Quantity(value=1.0, unit=dalton), 'sidechain_bead_masses': Quantity(value=1.0, unit=dalton)}, sigmas={'bb_bb_sigma': Quantity(value=2.5, unit=angstrom), 'bb_sc_sigma': Quantity(value=2.5, unit=angstrom), 'sc_sc_sigma': Quantity(value=2.5, unit=angstrom)}, epsilons={'bb_bb_eps': Quantity(value=0.5, unit=kilocalorie/mole), 'bb_sc_eps': Quantity(value=0.5, unit=kilocalorie/mole), 'sc_sc_eps': Quantity(value=0.5, unit=kilocalorie/mole)}, bond_lengths={'bb_bb_bond_length': Quantity(value=1.0, unit=angstrom), 'bb_sc_bond_length': Quantity(value=1.0, unit=angstrom), 'sc_sc_bond_length': Quantity(value=1.0, unit=angstrom)}, bond_force_constants={'bb_bb_bond_k': 9900000000.0, 'bb_sc_bond_k': 9900000000.0, 'sc_sc_bond_k': 9900000000.0}, bond_angle_force_constants={'bb_bb_bb_angle_k': 200, 'bb_bb_sc_angle_k': 200, 'bb_sc_sc_angle_k': 200, 'sc_sc_sc_angle_k': 200}, torsion_force_constants={'bb_bb_bb_bb_torsion_k': 200, 'bb_bb_bb_sc_torsion_k': 200, 'bb_bb_sc_sc_torsion_k': 200, 'bb_sc_sc_bb_torsion_k': 200, 'bb_sc_sc_sc_torsion_k': 200, 'sc_bb_bb_sc_torsion_k': 200, 'sc_sc_sc_sc_torsion_k': 200}, equil_dihedral_angle=180, charges={'backbone_bead_charges': Quantity(value=0.0, unit=elementary charge), 'sidechain_bead_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, ho-*

Parameters

- **positions** (*np.array(float * unit (shape = num_beads x 3))*) – Positions for all of the particles, default = None
- **polymer_length** (*integer*) – Length of the polymer, default = 8
- **backbone_lengths** – List of integers defining the umber of beads in the backbone for each monomer type

portion of each (individual) monomer (integer), default = [1]

sidechain_lengths: List of integers defining the umber of beads in the sidechain for each monomer type 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]]

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

epsilons: Non-bonded Lennard-Jones equilibrium interaction strengths ([[float * simtk.unit.energy], [float * simtk.unit.energy], [float * simtk.unit.energy]]) default = [[0.5 * unit.kilocalorie_per_mole],[0.5 * unit.kilocalorie_per_mole],[0.5 * unit.kilocalorie_per_mole]]

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

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

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

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 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_lengths sidechain_lengths sidechain_positions masses sigmas epsilons bond_lengths nonbonded_interaction_list bond_list bond_angle_list torsion_list bond_force_constants bond_angle_force_constants torsion_force_constants equil_dihedral_angle charges num_beads positions system topology constrain_bonds include_bond_forces include_nonbonded_forces include_bond_angle_forces include_torsion_forces

check_energy_conservation = 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 indices for particles that define bond angles in our coarse grained model

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

get_monomer_types()
Get a list of monomer dictionary objects for each unique monomer type.

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

get_num_beads()
Calculate the number of beads in our coarse grained model(s)

get_particle_list()
Get a list of particles, where the indices correspond to those used in our system/topology

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

`cg_model.cgmodel.`**`basic_cgmodel`**(*polymer_length=8, backbone_length=1, sidechain_length=1, sidechain_positions=[0], mass=Quantity(value=12.0, unit=dalton), charge=Quantity(value=0.0, unit=elementary charge), bond_length=Quantity(value=1.0, unit=angstrom), sigma=Quantity(value=2.5, unit=angstrom), epsilon=Quantity(value=0.5, unit=kilocalorie/mole), positions=None*)

Given a minimal set of model parameters, this function creates a cgmodel class object.

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

backbone_length: Integer defining the number of beads in the backbone default = 1

sidechain_length: Integer defining the number of beads in the sidechain default = 1

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

sidechain_positions: List of integers defining the backbone bead indices upon which we will place the sidechains, default = [0]

mass: Mass for all coarse grained beads. default = 12.0 * unit.amu

bond_length: Bond length for all bond types Default = 1.0 * unit.angstrom

sigma: Non-bonded bead Lennard-Jones equilibrium interaction distance. default = 2.5 * bond_length

epsilon: Non-bonded Lennard-Jones equilibrium interaction energy default = 0.5 * unit.kilocalorie_per_mole

charge: Charge for all particles default = 0.0 * unit.elementary_charge

positions: Positions for coarse grained particles in the model. default = None

cgmodel: CGModel() class object

`cg_model.cgmodel.`**`get_all_particle_masses`**(*cgmodel*)

Returns a list of unique particle masses

cgmodel: CGModel() class object

List(unique particle masses)

`cg_model.cgmodel.`**`get_bond_force_constant`**(*cgmodel, particle_1_index, particle_2_index*)

Determines the correct bond force constant for two particles

cgmodel: CGModel() class object

particle_1_index: Index of the first particle in the bond (integer) Default = None particle_2_index: Index of the second particle in the bond (integer) Default = None

bond_force_constant: Bond force constant for the bond defined by these two particles. (Integer)

`cg_model.cgmodel.`**`get_bond_length`**(*cgmodel*, *particle_1_index*, *particle_2_index*)

Determines the correct bond force constant for two particles

cgmodel: CGModel() class object

particle_1_index: Index of the first particle in the bond (integer) Default = None

particle_2_index: Index of the second particle in the bond (integer) Default = None

bond_length: Bond length for the bond defined by these two particles. (simtk.unit.Quantity())

`cg_model.cgmodel.`**`get_bond_length_from_names`**(*cgmodel*, *particle_1_name*, *particle_2_name*)

Determines the correct bond length for two particles, given their symbols.

cgmodel: CGModel() class object

particle_1_name: Symbol for the first particle in the bond (string) Default = None

particle_2_name: Symbol for the second particle in the bond (string) Default = None

bond_length: Bond length for the bond defined by these two particles. (simtk.unit.Quantity())

`cg_model.cgmodel.`**`get_epsilon`**(*cgmodel*, *particle_index*, *particle_type=None*)

Returns the epsilon value for a particle, given its index.

cgmodel: CGModel() class object

Epsilon

`cg_model.cgmodel.`**`get_parent_bead`**(*cgmodel*, *monomer_index*, *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

monomer_index: Index of the monomer the child particle belongs to. (integer) Default = None

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_charge`**`(`*cgmodel*, *particle_index*`)`
Returns the charge for a particle, given its index.

cgmodel: CGModel() class object

Charge

`cg_model.cgmodel.`**`get_particle_mass`**`(`*cgmodel*, *particle_index*`)`
Returns the mass for a particle, given its index.

cgmodel: CGModel() class object

Mass

`cg_model.cgmodel.`**`get_particle_type`**`(`*cgmodel*, *particle_index*, *particle_name=None*`)`
Returns the name of a particle, given its index within the model

cgmodel: CGModel() class object

particle_index: Index of the particle for which we would like to determine the type Type: int()

particle_type: 'backbone' or 'sidechain' Type: str()

`cg_model.cgmodel.`**`get_sigma`**`(`*cgmodel*, *particle_index*, *particle_type=None*`)`
Returns the sigma value for a particle, given its index within the coarse grained model.

cgmodel: CGModel() class object

Sigma

`cg_model.cgmodel.`**`get_torsion_force_constant`**`(`*cgmodel*, *torsion*`)`
Determines the torsion force constant given a list of particle indices

cgmodel: CGModel() class object

torsion: Indices of the particles in the torsion (integer) Default = None

torsion_force_constant: Force constant for the torsion defined by the input particles. (Integer)

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*, *energy=None*`)`

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.`**`get_structure_from_library`**(*cgmodel*)

Given a coarse grained model class object, this function retrieves a set of positions for the model from the ensemble library, in: '../foldamers/ensembles/${backbone_length}_${sidechain_length}_${sidechain_positions}'

If this coarse grained model does not have an ensemble library, an error message will be returned and we will attempt to assign positions at random with 'random_positions()'.

cgmodel: CGModel() class object.

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

`utilities.util.`**`random_positions`**(*cgmodel*, *max_attempts=1000*, *use_library=True*)

Assign random positions for all beads in a coarse-grained polymer.

cgmodel: CGModel() class object.

max_attempts: The maximum number of times that we will attempt to build a coarse grained model with the settings in 'cgmodel'. default = 1000

use_library: A logical variable determining if we will generate a new random structure, or take a random structure from the library in the following path: '../foldamers/ensembles/${backbone_length}_${sidechain_length}_${sidechain_positions}' default = True (NOTE: By default, if use_library = False, new structures will be added to the

> ensemble library for the relevant coarse grained model. If that model does not have an ensemble library, one will be created.)

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

INDEX