foldamers Documentation

Release 0.0

Garrett A. Meek
Theodore L. Fobe
Connor M. Vogel
Research group of Professor Michael R. Shirts

Dept. of Chemical and Biological Engineering
University of Colorado Boulder

Jul 22, 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 The 'basic_cgmodel' function to build coarse grained oligomers

Shown below is the 'basic_cgmodel' function, which requires only a minimal set of input arguments to build a coarse grained model. Given a set of input arguments this function creates a CGModel() class object, 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=100.0, unit=dalton), bond_length=Quantity(value=7.5, unit=angstrom), sigma=Quantity(value=18.5, unit=angstrom), epsilon=Quantity(value=0.5, unit=kilocalorie/mole), positions=None*)

Parameters

- **polymer_length** (*integer*) – Number of monomer units, default = 8
- **backbone_length** (*integer*) – Defines the number of beads in the backbone (assumes all monomers have the same backbone length), default = 1
- **sidechain_length** (*integer*) – Defines the number of beads in the sidechain (assumes all monomers have the same sidechain length), default = 1

- **sidechain_positions** (*List(integer)*) – Defines the backbone bead indices upon which we will place the sidechains, default = [0]
- **mass** (*float * simtk.unit*) – Mass for all coarse grained beads, default = 100.0 * unit.amu
- **bond_length** (*float * simtk.unit*) – Defines the length for all bond types, default = 7.5 * unit.angstrom
- **sigma** (*float * simtk.unit*) – Non-bonded bead Lennard-Jones equilibrium interaction distance, default = 18.5 * bond_length (for all particle interactions)
- **epsilon** – Non-bonded Lennard-Jones equilibrium interaction energy, default = 0.5 * unit.kilocalorie_per_mole
- **positions** (*np.array(float * simtk.unit (shape = num_beads x 3))*) – Positions for coarse grained particles in the model, default = None

Returns 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.

class cg_model.cgmodel.**CGModel**(*positions=None, polymer_length=8, backbone_lengths=[1], sidechain_lengths=[1], sidechain_positions=[0], masses={'backbone_bead_masses': Quantity(value=100.0, unit=dalton), 'sidechain_bead_masses': Quantity(value=100.0, unit=dalton)}, sigmas={'bb_bb_sigma': Quantity(value=18.5, unit=angstrom), 'bb_sc_sigma': Quantity(value=18.5, unit=angstrom), 'sc_sc_sigma': Quantity(value=18.5, 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=7.5, unit=angstrom), 'bb_sc_bond_length': Quantity(value=7.5, unit=angstrom), 'sc_sc_bond_length': Quantity(value=7.5, unit=angstrom)}, bond_force_constants=None, bond_angle_force_constants=None, torsion_force_constants=None, equil_bond_angles=None, equil_torsion_angles=None, charges=None, constrain_bonds=True, include_bond_forces=True, include_nonbonded_forces=True, include_bond_angle_forces=True, include_torsion_forces=True, check_energy_conservation=True, homopolymer=True*)

Parameters

- **polymer_length** (*integer*) – Number of monomer units, default = 8

Parameters

- **positions** (*np.array(float * simtk.unit (shape = num_beads x 3))*) – Positions for all of the particles, default = None

param backbone_lengths Defines the number of beads in the backbone for each monomer type, default = [1]

type backbone_lengths List(integer)

param sidechain_lengths Defines the number of beads in the sidechain for each monomer type, default = [1]

type sidechain_lengths List(integer)

param sidechain_positions Defines the backbone bead indices where sidechains are positioned, default = [0] (Place a sidechain on the first backbone bead in each monomer.)

type sidechain_positions List(integer)

param masses Masses of all particle types, default = 100.0 * unit.amu (for all particles)

type masses dict('backbone_bead_masses': float * simtk.unit, 'sidechain_bead_masses': float * simtk.unit)

param sigmas Non-bonded bead Lennard-Jones equilibrium interaction distances, default = 18.5 unit.angstrom (for all particles)

type sigmas dict('bb_bb_sigma': float * simtk.unit,'bb_sc_sigma': float * simtk.unit,'sc_sc_sigma': float * simtk.unit}

param epsilons Non-bonded Lennard-Jones equilibrium interaction strengths, default = 0.5 * unit.kilocalorie_per_mole (for all particle interactions types)

type epsilons dict('bb_bb_eps': float * simtk.unit,'bb_sc_eps': float * simtk.unit,'sc_sc_eps': float * simtk.unit)

param bond_lengths Bond lengths for all bonds, default = 7.5 unit.angstrom

type bond_lengths dict('bb_bb_bond_length': float * simtk.unit,'bb_sc_bond_length': float * simtk.unit,'sc_sc_bond_length': float * simtk.unit)

param bond_angle_force_constants Bond angle force constants

for all bond types, default = 200 * kJ/mol/rad^2

type bond_angle_force_constants dict('bb_bb__bb_angle_k': float,'bb_bb_sc_angle_k': float, 'bb_sc_sc_angle_k': float, 'sc_sc_sc_angle_k': float, 'sc_bb_sc_angle_k': float, 'sc_sc_bb_angle_k': float)

param bond_force_constants Bond force constants for all bond types, default = 1250 kJ/mol/nm^2

type bond_force_constants dict('bb_bb_bond_k': float,'bb_sc_bond_k': float, 'sc_sc_bond_k': float)

param equil_bond_angles Equilibrium bond angle for all bond angle types, default = 120

type equil_bond_angles dict('bb_bb_bb_angle_0': float,'bb_bb_sc_angle_0': float,'bb_sc_sc_angle_0': float,'sc_sc_sc_angle_0': float, 'sc_bb_sc_angle_0': float,'sc_sc_bb_angle_0': float)

param torsion_force_constants Torsion force constants for all unique torsion definitions, default = 200

type torsion_force_constants dict('bb_bb_bb_bb_torsion_k': float,'bb_bb_bb_sc_torsion_k': float,'bb_bb_sc_sc_torsion_k': float, 'bb_sc_sc_sc_torsion_k': float, 'sc_bb_bb_sc_torsion_k': float, 'bb_sc_sc_bb_torsion_k': float, 'sc_sc_sc_sc_torsion_k': float, 'sc_bb_bb_bb_torsion_k': float)

param equil_torsion_angles Equilibrium torsion angle for all unique torsion angle definitions, default = 0

type equil_torsion_angles dict('bb_bb_bb_bb_torsion_0': float,'bb_bb_bb_sc_torsion_0': float,'bb_bb_sc_sc_torsion_0': float, 'bb_sc_sc_sc_torsion_0': float, 'sc_bb_bb_sc_torsion_0': float, 'bb_sc_sc_bb_torsion_0': float, 'sc_sc_sc_sc_torsion_0': float, 'sc_bb_bb_bb_torsion_0': float)

param charges Charges for all particles, default = 0.0 (for all particles)

type charges dict('backbone_bead_charges': float * simtk.unit,'sidechain_bead_charges': float * simtk.unit)

param num_beads Total number of particles in the coarse grained model, default = 16 (The total number of particles in a length=8 1-1 coarse-grained model)

type num_beads integer

param system OpenMM System() object, which stores the forces for the coarse grained model, default = None

type system OpenMM System() class object

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

type topology OpenMM Topology() class object

param constrain_bonds Logical variable determining whether bond constraints are applied during a simulation of the energy for the system, default = False

type constrain_bonds Logical

param include_bond_forces Include contributions from bond potentials when calculating the potential energy, default = True

type include_bond_forces Logical

param include_nonbonded_forces Include contributions from nonbonded interactions when calculating the potential energy, default = True

type include_nonbonded_forces Logical

param include_bond_angle_forces Include contributions from bond angle forces when calculating the potential energy, default = True

type include_bond_angle_forces Logical

param include_torsion_forces Include contributions from torsions when calculating the potential energy, default = True

type include_torsion_forces Logical

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

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

get_all_particle_masses()
Returns a list of unique particle masses

self: CGModel() class object

List(unique particle masses)

get_bond_angle_force_constant(*particle_1_index*, *particle_2_index*, *particle_3_index*)
Determines the correct equilibrium bond angle between three particles

self: CGModel() class object

particle_1_index: Index of the first particle in the bond, default = None

particle_2_index: Index of the second particle in the bond angle, default = None

particle_3_index: Index of the third particle in the bond angle, default = None

bond_angle: Bond angle for the two bonds defined by these three particles.

get_bond_angle_list()
Construct a list of particle indices for the bond angle definitions in our coarse grained model

get_bond_force_constant(*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, default = None

particle_2_index: Index of the second particle in the bond, default = None

bond_force_constant: Bond force constant for the bond defined by these two particles

get_bond_length(*particle_1_index*, *particle_2_index*)
Determines the correct bond force constant for two particles

self: 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())

get_bond_length_from_names(*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())

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

get_epsilon(*particle_index*, *particle_type=None*)
Returns the epsilon value for a particle, given its index.

self: CGModel() class object

Epsilon

get_equil_bond_angle(*particle_1_index*, *particle_2_index*, *particle_3_index*)
Determines the correct equilibrium bond angle between three particles

self: CGModel() class object

particle_1_index: Index of the first particle in the bond, default = None

particle_2_index: Index of the second particle in the bond angle, default = None

particle_3_index: Index of the third particle in the bond angle, default = None

bond_angle: Bond angle for the two bonds defined by these three particles.

get_equil_torsion_angle(*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)

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

self: CGModel() class object

Charge

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

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

self: CGModel() class object

Mass

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

self: 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()

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

self: CGModel() class object

Sigma

get_torsion_force_constant(*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)

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

1.3 Other coarse grained model utilities

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)

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_bonds**(*CGModel*, *pdb_object*)
Writes the bonds from an input CGModel class object to the file object 'pdb_object', using PDB 'CONECT' syntax.

CGModel: Coarse grained model class object

pdb_object: File object to which we will write the bond list

utilities.iotools.**write_cg_pdb**(*cgmodel*, *file_name*)
Writes the positions from an input CGModel class object to the file 'filename'. Used to test the compatibility of coarse grained model parameters with the OpenMM PDBFile() functions, which are needed to write coordinates to a PDB file during MD simulations.

CGModel: Coarse grained model class object

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

utilities.iotools.**write_pdbfile_without_topology**(*CGModel*, *filename*, *energy=None*)
Writes the positions from an input CGModel class object to the file 'filename'.

CGModel: Coarse grained model class object

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

energy: Energy to write to the PDB file, default = None

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*, *high_energy=False*, *low_energy=False*)

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.

Parameters

- **high_energy** (*Logical*) – If set to 'True', this function will generate an ensemble of high-energy structures, default = False
- **low_energy** (*Logical*) – If set to 'True', this function will generate an ensemble of low-energy structures, default = False

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=False*, *high_energy=False*, *low_energy=False*, *generate_library=False*)

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.)

Parameters

- **high_energy** (*Logical*) – If set to 'True', this function will generate an ensemble of high-energy structures, default = False
- **low_energy** (*Logical*) – If set to 'True', this function will generate an ensemble of low-energy structures, default = False

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

R

U

W