**BFEE v2.1 Python API**

**Import BFEE2 module:**

import BFEE2.inputGenerator as inputGenerator

import BFEE2.postTreatment as postTreatment

**Use inputGenerator:**

# initialize inputGenerator object

iGenerator = inputGenerator.inputGenerator()

# generate all the input files for NAMD alchemical simulation

iGenerator.generateNAMDAlchemicalFiles(

path, topFile, coorFile, forceFieldType, forceFieldFiles,

temperature, selectionPro, selectionLig,

stratification = [1,1,1,1],

doubleWide = False,

minBeforeSample = False,

membraneProtein = False,

pinDownPro = True,

vmdPath = '')

# Inputs:

# path (string): the directory for generation of all the files

# topFile (string): the path of the topology (psf, parm) file

# coorFile (string): the path of the coordinate (pdb, rst7) file

# forceFieldType (string): 'charmm' or 'amber'

# forceFieldFiles (list of strings): list of CHARMM force field files

# temperature (float): temperature of the simulation

# selectionPro (string): MDAnalysis-style selection of the protein

# selectionLig (string): MDAnalysis-style selection of the ligand

# stratification (list of int, 8): number of windows for each simulation

# doubleWide (bool): whether double-wide simulations are carried out

# minBeforeSample (bool): minimization before sampling in each FEP window

# membraneProtein (bool): whether simulating a membrane protein

# pinDownPro (bool): whether pinning down the protien

# vmdPath (string): path to vmd '''

# generate all the input files for NAMD Geometric simulation

iGenerator.generateNAMDGeometricFiles(

path, topFile, coorFile, forceFieldType, forceFieldFiles,

temperature, selectionPro, selectionLig,

selectionRef = '',

userProvidedPullingTop = '',

userProvidedPullingCoor = '',

stratification = [1,1,1,1,1,1,1,1],

membraneProtein = False,

pinDownPro = True,

parallelRuns = 1,

vmdPath = '')

# Inputs:

# path (string): the directory for generation of all the files

# topFile (string): the path of the topology (psf, parm) file

# coorFile (string): the path of the coordinate (pdb, rst7) file

# forceFieldType (string): 'charmm' or 'amber'

# forceFieldFiles (list of strings): list of CHARMM force field files

# temperature (float): temperature of the simulation

# selectionPro (string): MDAnalysis-style selection of the protein

# selectionLig (string): MDAnalysis-style selection of the ligand

# selectionRef (string): MDAnalysis-style selection of the reference group for pulling simulation, by default, this is the protein

# userProvidedPullingTop (string): user-provided large solvation box for pulling simulation

# userProvidedPullingCoor (string): user-provided large solvation box for pulling simulation

# stratification (list of int, 8): number of windows for each simulation

# membraneProtein (bool): whether simulation a membrane protein

# pinDownPro (bool): whether pinning down the protien

# parallelRuns (int): generate files for duplicate runs

# vmdPath (string): path to vmd

**Use posttreatment:**

# initialize posttreatment object

pTreat = postTreatment.postTreatment(temperature, unit, jobType)

# Inputs:

# temperature (float): temperature of the simulation

# unit (string): unit convention used by MD engine, 'namd' or 'gromacs'

# jobType (string): 'geometric' or 'alchemical'

# calculate the binding free energy through the geometrical route

pTreat.geometricBindingFreeEnergy(filePathes, parameters)

# Inputs:

# filePathes (list of strings, 8): pathes of PMF files for step1 - step8

# parameters (np.array, floats, 8): (forceConstant1, FC2, FC3, FC4, FC5, FC6, r\*, FC8)

# Return:

# np.array, float, 10: (contributions for step1, 2, 3, 4 ... 8, bulk restraining contribution, binding free energy)

# calculate the binding free energy through the alchemical route

pTreat.alchemicalBindingFreeEnergy(filePathes, parameters):

# Inputs:

# filePathes (list of strings, 8): pathes of alchemical output files

# (step1-forward, step1-backward, step2-forward ...)

# parameters (np.array, floats, 9): (eulerTheta, polarTheta, r, forceConstant1, FC2, FC3, FC4, FC5, FC6)

# Return:

# np.array, float, 6: (contributions for step1, 2, 3, 4, bulk restraining contribution, free energy)

# np.array, float, 6: errors corresponding each contribution