**A tutorial for using ScMile2 to run Milestoning simulations**

In this tutorial, we will go through the Milestoning simulation set-up of a solvated adenosine with ScMile2 step by step.

The structure of the adenosine molecule is shown in the figure below. We choose two coarse variables to characterize its conformational space: the glycosyl torsion  (O4’-C1’-N9-C4) and the pseudo-rotation angle

where the set  are torsions corresponding to C1’-C2’-C3’-C4’, C2’-C3’-C4’-O4’, C3’-C4’-O4’-C1’, C4’-O4’-C1’-C2’, O4’-C1’-C2’-C3’ respectively. We partition the conformational space into cells using Voronoi tessellation by randomly picking up 20 anchors in the 2D coarse space. The cell centered at anchor 15 is defined as the reactant (i.e. all interfaces (milestones) surrounding the anchor 15 are used as the reactant state) and the cell centered at anchor 6 is the product state. Note that indices used inside ScMile2 starts from 1.

A picture containing accessory, necklet

Description automatically generated 

To start the Milestoning calculation, we first need to prepare 20 pdb files, named as 1.pdb, 2.pdb…, for 20 anchors. These files are the configurations of each anchor, which will be later used as the starting configuration, e.g., for seeking milestones nearby each anchor. We used NAMD 2.13 as the Molecular Dynamics (MD) engine for the illustration. So, we also need to prepare the normal MD input files for our system, i.e., Adenosine\_sol.psf and Adenosine\_sol.pdb.

Now we are ready to start working on the ScMile2 script. In the folder my\_project\_input, we prepared the input.txt file, in which we setup all the input keywords for ScMile2. anchors.txt is where we enter the anchor positions. colvar.txt and custom.covlar are the colvar setup files.

The anchors.txt file contains two columns. Each line represents an anchor position , i.e. the first element is  and the second element is . The order of anchors in this file should match the name of the anchor pdb files, i.e., the first line associates to 1.pdb, the second line associates to 2.pdb.. etc.

The input.txt file contains all the keywords setup for ScMile2.

Starting from the first option, **outputname** is the output name option in NAMD. In this tutorial, we use Adenosine.

**method** determines how you want to do your Milestoning: 0 for classical Milestoning and 1 for exact Milestoning.

**max\_iteration** is the maximum iteration number for exact Milestoning. The script will stop if the maximum iteration number is reached, even the results is not converged yet.

**milestoneSearch** is how we define our milestones: 0 for Traverse and 1 for Seek. If Traverse is selected, it is assumed that there always exists a milestone between anchor i and anchor i+1, and milestones are placed along the anchor indices in one dimension. In Seek, we also need to have two additional setups, **initial\_traj** and **initial\_time**. We conduct **initial\_traj** number of unbiased trajectories starting from each of the anchor, let them run for up to **initial\_time** of time (in picosecond) and at the first time they hit a milestone we record the configuration. The discovery of this configuration activates the milestone between two anchors and provide an initial configuration of sampling at the milestone. If Traverse is selected, the initial configuration of sampling is the conformation form one of the anchors associated to the milestone (e.g. 1.pdb for milestone between anchor 1 anchor 2).

**pbc\_names** designates periodic coarse variables in Milestoning calculations. And another option **L** is used to set the range of those periodic variables. When a coarse variable is a complex of a set of more elementary coarse variables, like the p variable in the current tutorial, we need to activate another keyword **substitution**.The script will then automatically combine all the elementary coarse variables written in colvar.txt into a complex one according the template in custom.colvar.

**colvar\_names** is all the colvars used in the distance calculation in Milestoning.

**customColvars**: ‘on’ or ‘off’ to indicate if any customized colvar exists. If it is set to ‘on’, the script will read custom.colvar file, and copy and paste everything in this file to the colvar configuration. Such colvar can be additional colvar module or just an index file. In this example, we want to have both coarse vairables,  and p, output all the time, so we choose ‘on’. At the same time, it serves as the template to combine elementary colvars, , into the p variable.

**custom\_colvar** the number of custom colvars in custom.colvar. Use 0 if no custom colvar module. Because we have two additional coarse variables, we have 2 here.

**covlarsTrajFrequency**: the frequency of each colvar is saved.

**colvarsResatartFrequency**: the frequency of each colvar is written to colvar.traj file.

**anchorsNum** is the total number of anchors. It will read the first anchorsNum of lines in the anchor.txt and will only consider the same number of pdb files.

**reactant**: milestones associated to the reactant state. Based on the setup in this example, we have reactant at the milestone between anchor 15 and 20.

**product**: similar to reactant. Milestones associated to the product state. Here we have milestone between anchor 6 and 11.

**total\_trajs** is the total number of initial configurations from sampling at each milestone. If you run 1 nanosecond for sampling and save restart files every 1 picosecond, you will have 1000 sets of restart files, which can be used as initial configurations for unbiased trajectories.

**start\_traj** is the first configurations be used as the initial conformation for free trajectory. If Seek is used, even the very first configuration is close to the milestone, so you can use a small number for this option, say 1. However, if you use Traverse, it will take some time to reach the milestone because it started from anchor. Thus, you may want ot use a large number here.

**traj\_per\_launch** is the number of free trajectories to launch for each milestone.

**interval** is the interval between two configurations. For example, if interval is set to 10, free trajectories will use configurations number 1, 11, 21, … etc as initial conformations. If the restart files are written every picocsecond, the configurations used are 1000.coor/vel/xsc, 11000.coor/vel/xsc, 12000.coor/vel/xsc…

**tolerance** tolerance in the MFPT convergence check (0.001=0.1%). If the difference of MFPT between last and current iteration is smaller than this number, the result is considered as converged, and the script will stop.

**error\_sampling** is the number of resampling times for each transition probability and local lifetime.

**jobsubmission**: command for job submission on HPC. (sbatch or qsub)

**jobcheck**: command for job check on HPC. (squeue or qstat)

**username**: you r username on HPC

**seed**: random seed. Note that this is not the actual random seed used in NAMD. The seed used in NAMD will be a random number pulled based on the seed set in input.

**force\_const** is the force constant value used for restrained sampling on each milestone.

Text

Description automatically generated

We have finished the setup in the input.txt file. Now let’s look at the covlar setup. First, we open colvar.txt. In this file, we input all coarse variables in its elementary form, i.e. the form that is supported directly by NAMD, not combined as in customFunction. In this example, the  variable itself is already in the elementary form. The p variable is a customFunction, which need to write in its elementary form, i.e., .

Note that the colvars here in colvar.txt will not be output in the colvar.traj, and it is only used for constraining on the milestones. If you want to have them output, you can use customized colvar and turn on the custom\_colvar in the input.txt. Especially when a coarse variable is a customFunction, we should also have it in the custom.colvar, which will be used as the template for Scmile2 to combine its elementary colvars.

colvar.txt

A picture containing text, battery, plaque

Description automatically generated

custom.colvar

A screenshot of a computer

Description automatically generated with low confidence

Since ScMile2 interfaces with NAMD, next thing we will do is to setup NAMD configuration. We will need two NAMD configuration files, one for sampling process and one for unbiased trajectories. Rename the one for sampling with sample.namd and the one for free trajectories with free.namd, and move them to my\_project\_input. Such configurations may vary by cases. For most of the cases, these two setups are slightly different. In the example here, we use NVT for sampling and NVE for unbiased trajectories. For some systems, you may have same setup for both. However, you still need to have two copies in your input folder. Setup for NAMD is just like what you normally do for MD simulations. You specify the structure, parameters, force fields… etc. Here just do the same thing. However, a couple of things we need to keep in mind. 1) For all the parameters files, do use the absolute path instead of relevant path, so that the parameters can be accessed by all the jobs. 2) Have bincoordinates/ binvelocities/ extendedSystem in free.namd as a bookkeeper. ScMile2 will modify it and provide it with the correct file name. 3) save binary restart files. Remember that we use restart files as the initial configuration of unbiased trajectories, so the total number restarts should match what you have for total\_trajs in input.txt. For unbiased trajectories, the restart saving frequency can vary, but if you would like to use Exact Milestoning method, it is recommended to use a small number. Here we save restart files every 2 timesteps.

The last step before we can start the simulation is to tell ScMile2 how you would like to run on your cluster. Currently, a template bash script for job submission is necessary. It would be similar to what you use for other simulations. However, a couple of things need to be modified. First, use ‘name’ as a place keeper to replace the real job name. Second, add a ‘cd path’ command before call NAMD. This will help ScMile2 to locate the correct subfolder for each job. Thired, use ‘namd’ as a place keeper for the NAMD configuration file name. If you need to load additional module, you can do it as well.

Now we can call ScMiles/main.py to start ScMile.

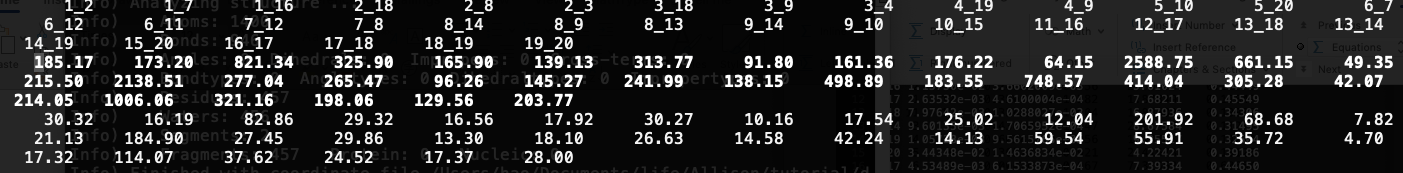
Once the simulation finished, the results can be found in my\_project\_output/, and label with iteration number and followed by time, i.e. my\_project\_output/1\_2021-01-01 00:00:00. You can also find the most recent result in my\_project\_output/current folder. Results.txt have most of the results. In this file, there are two sections. The first section has 6 columns showing the information for all the milestones. The first two columns represent the anchors associated withmilestone, i.e. (1,2) means the milstone between anchor 1 and anchor 2. The third column is the flux on that milestone. The fourth column shows the probability on the milestone. The last two columns show the free energy (in KbT) and the error bar for that milestone. The error calculation is based on 1000 sampled generated with Beta random sampling. The second section in this file shows the MFPT. Two different boundary conditions are used to do the calculations, and they show similar results.

Text

Description automatically generated

The kernel matrix, K, is in k.txt/k\_norm.txt.

life\_time.txt shows the lifetime information for each milestone and the standard deviation.



Committor.txt shows the committor function.

