OPSPiggybacker Documentation

Release 0.1-dev

David W.H. Swenson

September 27, 2016

Contents

OPSPiggybacker is a tool to assist importing other simulation records into OpenPathSampling for analysis. Because of the extensive metadata that OPS tracks for each move, this isn't completely trivial. The idea of OPSPiggybacker is to request a reasonable set of inputs that a user of another path sampling simulation tool can give, and then create from that a file that can use most of OPS's standard analysis tools.

The current version, 0.1, only aims to cover one-way shooting moves in a single ensemble, as with TPS simulations. See the roadmap for future plans.

CHAPTER 1

Installation

TODO

CHAPTER 2

Overview

The overall approach is very similar to the setup of an OPS simulation. You use the standard OPS volume, collective variable, network, and ensemble objects. The only things that change are the move scheme/path movers, and the simulation object.

Instead of an OPS move scheme or OPS path movers, you build what we call a mover stub. Currently, the `ShootingStub` is the only mover stub supported. The `ShootingStub` is analogous to an `OneWayShootingMover` in OpenPathSampling. It is initialized with an `ensemble`, a `selector` (only `UniformSelector` is currently supported) and optionally an `engine`.

Once you've defined the mover stub, you create a pseudo-simulator. Currently, the only supported pseudo-simulator is the `ShootingPseudoSimulator`. The pseudo-simulator plays the same role as a `PathSampling` object in OpenPathSampling. It is initialized with a `storage`, a set of `initial_conditions` (in the form of an `openpathsampling.SampleSet`), a `network`, and a mover stub called `mover`, which takes the place of the move scheme used in OPS.

Once this is done, you simply use the `run` method of the `ShootingPseudoSimulator` to generate your file. However, whereas the run method of the `PathSampling` object in OPS takes an integer with a number of steps, in OPSPiggybacker, you must provide the output of your previous simulation to the run method of the pseudo-simulator. The following subsection will describe the moves.

CHAPTER 3

Partial input trajectories

CHAPTER 4

Full input trajectories

One of the input options for shooting moves is to use full input trajectories (`pre_joined=True`). In this case, the input trajectory must be an OPS format trajectory for the full trial trajectory. In addition, *the frames which are shared with other trajectories must be identical in memory frames.* Since this is quite hard to do, it is usually easier to use the `pre_joined=False` version with partial input trajectories.

However, we full input trajectories, you don't need to specify whether a given trial was forward or backward: the OPSPiggybacker can figure that out for you.

This is in the format of a list of 4-tuples `(replica, trial_trajectory, shooting_point_index, accepted)`, where each 4-tuple represents a trial move. In detail, the elements of the tuple are:

- `replica`: the replica ID. Currently always the same (usually 0).
- `trial_trajectory`: the generated trial trajectory, as an `openpathsampling.engine.Trajectory` object. Note that there are two tricky things here. First, this must be the *entire* trial trajectory (not just the part generated during one-way shooting). Second, frames which are shared between two trajectories much actually be the same object in memory. This means that you have to rebuild the shooting process for your trajectories. (TODO: find ways to make this part easier on people)
- `shooting_point_index`: the frame number of the shooting point from the *previous* trajectory (counting from 0).
- `accepted`: boolean as to whether this trial move was accepted.

That's it! If you can make that tuple for each of your moves, you can import those moves into OPS for analysis.

CHAPTER 5

API Reference

5.1 Mover Stubs

5.2 Simulation Stubs