perses Documentation

Chodera lab // MSKCC

Feb 18, 2019

Contents:

Caution: This is module is undergoing heavy development. None of the API calls are final. This software is provided without any guarantees of correctness, you will likely encounter bugs.

If you are interested in this code, please wait for the official release to use it. In the mean time, to stay informed of development progress you are encouraged to:

- Follow this feed for updates on releases.
- Check out the github repository .

A Python framework for automated small molecule free energy driven design.

`perses` is a Python framework that uses OpenMM for GPU-accelerated molecular design driven by alchemical free energy calculations.

CHAPTER 1

Installation

1.1 Installing via *conda*

The simplest way to install `perses` is via the conda package manager. Packages are provided on the omnia Anaconda Cloud channel for Linux, OS X, and Win platforms. The perses Anaconda Cloud page has useful instructions and download statistics.

If you are using the anaconda scientific Python distribution, you already have the `conda` package manager installed. If not, the quickest way to get started is to install the miniconda distribution, a lightweight minimal installation of Anaconda Python.

On `linux`, you can install the Python 3 version into `$HOME/miniconda3` with (on `bash` systems):

```
$ wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
$ bash ./Miniconda3-latest-Linux-x86_64.sh -b -p $HOME/miniconda3
$ export PATH="$HOME/miniconda3/bin:$PATH"
```

On `osx`, you want to use the ``osx` binary

```
$ wget https://repo.continuum.io/miniconda/Miniconda2-latest-MacOSX-x86_64.sh
$ bash ./Miniconda3-latest-Linux-x86_64.sh -b -p $HOME/miniconda3
$ export PATH="$HOME/miniconda3/bin:$PATH"
```

You may want to add the new ``$PATH` extension to your `~/.bashrc` file to ensure Anaconda Python is used by default. Note that `perses` will be installed into this local Python installation, so that you will not need to worry about disrupting existing Python installations.

Note: `conda` installation is the preferred method since all dependencies are automatically fetched and installed for you.

1.1.1 Release build

You can install the latest stable release build of perses via the `conda` package with

```
$ conda config --add channels omnia
$ conda install perses
```

This version is recommended for all users not actively developing new algorithms for alchemical free energy calculations.

Note: `conda` will automatically dependencies from binary packages automatically, including difficult-to-install packages such as OpenMM, numpy, and scipy. This is really the easiest way to get started.

1.1.2 Development build

The bleeding-edge, absolute latest, very likely unstable development build of perses is pushed to binstar with each GitHub commit, and can be obtained by

```
$ conda config --add channels omnia
$ conda install perses-dev
```

Warning: Development builds may be unstable and are generally subjected to less testing than releases. Use at your own risk!

1.1.3 Upgrading your installation

To update an earlier `conda` installation of perses to the latest release version, you can use `conda update`:

```
$ conda update perses
```

CHAPTER 2

Changelog

CHAPTER 3

Modules

3.1 Alchemical transformations

Tools for nonequilibrium alchemical transformations

3.1.1 Nonequilibrium switching

`NCMCEngine`
`NCMCHybridEngine`
`NCMCAlchemicalIntegrator`
`NCMCGHMCAlchemicalIntegrator`

3.1.2 Relative alchemical transformations

`HybridTopologyFactory`

3.1.3 Principle of HybridTopologyFactory

HybridTopologyFactory is a class that automates the construction of so-called hybrid topologies and systems. In short, this amounts to using an atom map to merge two initial systems into a new system that contains a union of the former systems' degrees of freedom, along with a `lambda` parameter to control the degree to which the hybrid represents the old or new system. The process of creating this system happens in several steps, detailed below. Before proceeding, there are several important caveats:

- **Atom maps:**
  - Atoms whose constraints change between new and old system may not be mapped
  - Virtual sites are only permitted if their parameters are identical in the old and new systems
- **Systems:**

  - Only `HarmonicBondForce`, `HarmonicAngleForce`, `PeriodicTorsionForce`, `NonbondedForce`, and `MonteCarloBarostat` are supported. The presence of other forces will raise an exception.

Basic Terminology

Throughout the rest of this document, the following terms will be referenced:

- Hybrid topology: An object that "fuses" two systems based on an atom map, and whose identity is controlled by a set of global parameters called `lambda`. Mapped atoms are switched from the type in one system to the type in the other; unmapped atoms are switched on or off, depending on which system they belong to.
- Old system: We use this to refer to the endpoint represented by `lambda=0`.
- New system: We use this to refer to the endpoint represented by `lambda=1`.

Assignment of Particles to Appropriate Groups

In order to properly assign parameters to the particles in the hybrid system, the particles are first assigned to one of four groups:

- **Environment: These atoms are mapped between the new and old systems, and additionally are not part of a residue that d**
  - Examples: solvent molecules, protein residues outside a changing residue
- **Core: These atoms are mapped between the new and old system, but are part of a residue that differs between new and ol**
  - Example: The carbon atoms in a benzene that is being transformed to a chlorobenzene.
- **Unique old: These atoms are not in the map, and are present only in the old system**
  - Example: an extraneous hydrogen atom in a cyclohexane being transformed into a benzene
- **Unique new: These atoms are not in the map, and are present only in the new system**
  - Example: The chlorine atom that appears when a benzene is transformed into a chlorobenzene.

Treatment of different interaction groups

For each supported force, we have to create at least one custom force, as well as an unmodified force. In general, the interactions are handled as follows:

- Environment-Environment: These interactions are handled by regular unmodified forces, as their parameters never change.
- Environment-{Core, Unique old, Unique new}: These interactions are handled by Custom*Forces. For interactions involving the core atoms, the interaction scales from the parameter in the old system to the parameter in the new system as `lambda` is switched from 0 to 1. For interactions involving the unique atoms, the interaction scales from full to zero as `lambda` switches from 0 to 1 in unique old atoms, and vice versa for unique new atoms.
- Core-Core interactions: These interactions scale from old system parameters to new system parameters as `lambda` is switched from 0 to 1.
- Core-{unique old, unique new} interactions: These interactions scale from the parameters in the old system to the parameters in the new system as `lambda` is switched from 0 to 1. This means that unique atoms always maintain intramolecular interactions.

- Unique-old and unique-new interactions: These are disabled.
- intra-Unique atoms: These interactions are always enabled, and as such are handled by an unmodified force.

Force terms

Given the above division of atoms into different groups, the assignment of interactions to different force classes is straightforward.

Bonds

Bonds are handled with a `HarmonicBondForce` and a `CustomBondForce`.

- `HarmonicBondForce`: This handles all bond interactions that do not change with lambda.
- `CustomBondForce`: This handles bond interactions that change with lambda. Bond terms never become zero; they are always on to ensure that molecules do not fly apart

Angles

Angles are also handled with a `HarmonicAngleForce` and a `CustomAngleForce`

- `HarmonicAngleForce`: This handles all angle interactions that do not change with lambda.
- `CustomAngleForce`: This handles angle interactions that change with lambda. Angle terms never become zero; they are always on to ensure overlap.

Torsions

Torsions are handled with a `PeriodicTorsionForce` and a `CustomTorsionForce`

- `PeriodicTorsionForce`: This handles all torsion interactions that do not change with lambda.
- `CustomTorsionForce`: This handles interactions that change with lambda. Although the torsion potential for a given group of atoms is never set to zero, the force assignments sometimes include zero as an endpoint. This is to handle the case of torsion multiplicities greater than one.

Nonbonded (both sterics and electrostatics)

- `NonbondedForce`: This handles sterics and electrostatics interactions that do not change. All interactions that are involved in custom nonbonded forces are excluded from this force.
- `CustomBondForce`: This handles the exceptions from the standard `NonbondedForce` above.

Nonbonded (Sterics)

- `CustomNonbondedForce`: This handles softcore sterics for interactions that change with lambda. This force uses interaction groups to exclude interactions that do not change with `lambda` (and are covered by the `NonbondedForce` above). This force supports no cutoff, reaction field, and PME. An important note relates to the dispersion correction; by default, it is not enabled to improve performance. However, it can be enabled as a constructor argument in the HybridTopologyFactory.

Nonbonded (Electrostatics)

- `CustomNonbondedForce`: This is a separate `CustomNonbondedForce` that handles electrostatic interactions that change with lambda. This force uses interaction groups to exclude interactions that do not change with `lambda` (and are covered by the `NonbondedForce` above). This force supports the options nocutoff, reaction field, and PME.

Specific unsupported forces

Any force that is not in one of the above-mentioned classes will cause an exception to be raised. Some important examples of this are:

- Amoeba forces
- GB forces
- CMMotionRemover

Additional Features

In addition to creating a hybrid topology dependent on a set of `lambda_force` functions, the HybridTopologyFactory can also take advantage of OpenMM's automatic differentiation to provide dU/dlambda. In order to use this functionality, you must provide alchemical functions to the constructor of HybridTopologyFactory. These functions are described below.

3.1.4 Basic Usage of HybridTopologyFactory

Basic Construction of Factory

In order to use the HybridTopologyFactory, several objects are necessary:

- `perses.rjmc.topology_proposal.TopologyProposal` is a container class which holds so-called new and old systems, topologies, and respective atom maps. It can be created manually using the constructor, or can be obtained through a subcalss of `perses.rjmc.topology_proposal.ProposalEngine`.
- You need to supply current positions and new positions for the old and new systems, respectively. It is assumed that the atoms which are mapped will have the same positions. If you only have positions for the old system, you can generate (decent) positions for the new system using `perses.rjmc.geometry.FFAllAngleGeometryEngine`. You should minimize the hybrid topology that results after this to avoid clashes, however.
- The dispersion correct is calculated for the sterics force if `use_dispersion_correction` is set to `True`. It is false by default; whether to enable this depends on what sort of calculation you are doing:
  - Your calculation requires frequent changes to `lambda` parameters, such as nonequilibrium switching: Set the dispersion correction to false.
  - Your calculation runs a simulation at a set of constant `lambda` values (that don't change during the simulation), such as thermodynamic integration: Set the dispersion correction to True.
- The alchemical functions argument can be supplied to the `HybridTopologyFactory`, in which case only `lambda` will be exposed. Otherwise, each individual energy term will have a `lambda` associated with it.

Alchemical functions

The functions argument allows you to specify the mathematical relationship between a main *lambda* value, and each individual *lambda*. Adding the functions here also automatically enables the computation of the derivative of the energy with respect to `lambda`, useful for various tasks.

Some important notes about specifying the alchemical functions:

- All `lambda` values must be specified. More specifically, the dictionary must have the following form:

```
functions_dictionary = {
'lambda_bonds' : 'f_bond(lambda)'
'lambda_angles' : 'f_angle(lambda)'
'lambda_torsions' : 'f_torsion(lambda)'
'lambda_sterics' : 'f_sterics(lambda)'
'lambda_electrostatics' : 'f_electrostatics(lambda)'
}
```

where `f_{energy_name}()` is an arbitrary function of `lambda` compliant with OpenMM Lepton syntax. See the OpenMM documentation for more information on what kinds of functions can be implemented there.

Example script

Below is a very simple example script that constructs a `HybridTopologyFactory` object that can transform a benzene into catechol in vacuum.

```
#import needed functionality
from topology_proposal import SmallMoleculeSetProposalEngine, TopologyProposal
from perses.annihilation.new_relative import HybridTopologyFactory
import simtk.openmm.app as app
from openmoltools import forcefield_generators

#these are utility functions to rapidly create test systems.
from perses.tests.utils import createOEMolFromIUPAC, createSystemFromIUPAC, get_data_
↪filename

#We'll generate the systems by IUPAC name
mol_name = "benzene"
ref_mol_name = "catechol"

#create the benzene molecule and system, as well as a molecule of catechol
m, unsolv_old_system, pos_old, top_old = createSystemFromIUPAC(mol_name)
refmol = createOEMolFromIUPAC(ref_mol_name)

#The SmallMoleculeSetProposalEngine class needs smiles strings
initial_smiles = oechem.OEMolToSmiles(m)
final_smiles = oechem.OEMolToSmiles(refmol)

#set up the SystemGenerator
gaff_xml_filename = get_data_filename("data/gaff.xml")
system_generator = SystemGenerator([gaff_filename])

#The GeometryEngine will be useful for generating new positions
geometry_engine = FFAllAngleGeometryEngine()

#Instantiate the proposal engine, which will generate the TopologyProposal.
proposal_engine = SmallMoleculeSetProposalEngine(
```

(continues on next page)

(continued from previous page)

```
    [initial_smiles, final_smiles], system_generator, residue_name=mol_name)

#generate a TopologyProposal with the SmallMoleculeSetProposalEngine
topology_proposal = proposal_engine.propose(solvated_system, solvated_topology)

#Generate new positions with the GeometryEngine--note that the second return value␣
↪(logp) is ignored for this purpose
new_positions, _ = geometry_engine.propose(topology_proposal, solvated_positions,␣
↪beta)

#instantiate the HybridTopologyFactory
factory = HybridTopologyFactory(topology_proposal, old_positions, new_positions)

#Now you have the objects you need to run a simulation, controlled by the `lambda`␣
↪parameters.
hybrid_system = factory.hybrid_system
hybrid_topology = factory.hybrid_topology
initial_hybrid_positions = factory.hybrid_positions
```

3.2 Bias engine

Bias engine for sampling chemical space

`BiasEngine`
`MinimizedPotentialBias`

3.3 Molecular geometry generation via RJMC

Reversible jump Monte Carlo (RJMC) engine for sampling molecular geometries in which atoms are created/destroyed.

3.3.1 Topology proposal engines

`TopologyProposal`
`ProposalEngine`
`SmallMoleculeSetProposalEngine`
`PolymerProposalEngine`
`PointMutationEngine`
`PeptideLibraryEngine`
`NullProposalEngine`
`NaphthaleneProposalEngine`
`ButaneProposalEngine`
`PropaneProposalEngine`

3.3.2 OpenMM System generation utilities

`SystemGenerator`

3.3.3 Geometry proposal engines

`GeometryEngine`
`FFAllAngleGeometryEngine`
`OmegaGeometryEngine`

3.3.4 Geometry utility classes

`PredAtomTopologyIndex`
`BootstrapParticleFilter`
`GeometrySystemGenerator`
`GeometrySystemGeneratorFast`
`PredHBond`
`ProposalOrderTools`

3.4 Sampler stack

Samplers for driving sampling over conformations or chemical states.

`ThermodynamicState`
`SamplerState`
`GHMCIntegrator`
`MCMCSampler`
`ExpandedEnsembleSampler`
`SAMSSampler`
`MultiTargetDesign`
`ProtonationStateSampler`

3.5 Storage handling

Storange handling driver

`NetCDFStorage`	NetCDF storage layer.

3.5.1 perses.storage.NetCDFStorage

class `perses.storage.`**`NetCDFStorage`** (*filename*, *mode='w'*)

NetCDF storage layer.

Methods

`close`()	Close the storage layer.
`get_object`(envname, modname, varname[, . . .])	Get the serialized Python object.
`sync`()	Flush write buffer.
`write_array`(varname, array[, iteration])	Write a numpy array as a native NetCDF array
`write_configuration`(varname, positions, topology)	Write a configuration (or one of a sequence of configurations) to be stored as a native NetCDF array
`write_object`(varname, obj[, iteration])	Serialize a Python object, encoding as pickle when storing as string in NetCDF.
`write_quantity`(varname, value[, iteration])	Write a floating-point number

__init__(*filename*, *mode='w'*)
Create NetCDF storage layer, creating or appending to an existing file.

Parameters

filename [str] Name of storage file to bind to.

mode [str, optional, default='w'] File open mode, 'w' for (over)write, 'a' for append.

Methods

`__init__`(filename[, mode])	Create NetCDF storage layer, creating or appending to an existing file.
`close`()	Close the storage layer.
`get_object`(envname, modname, varname[, . . .])	Get the serialized Python object.
`sync`()	Flush write buffer.
`write_array`(varname, array[, iteration])	Write a numpy array as a native NetCDF array
`write_configuration`(varname, positions, topology)	Write a configuration (or one of a sequence of configurations) to be stored as a native NetCDF array
`write_object`(varname, obj[, iteration])	Serialize a Python object, encoding as pickle when storing as string in NetCDF.
`write_quantity`(varname, value[, iteration])	Write a floating-point number

close()
Close the storage layer.

get_object(*envname*, *modname*, *varname*, *iteration=None*)
Get the serialized Python object.

Parameters

envname [str] The name of the environment for the variable

modname [str] The name of the module for the variable

varname [str] The variable name to be stored

iteration [int, optional, default=None] The local iteration for the module, or *None* if this is a singleton

Returns

obj [object] The retrieved object

sync()
Flush write buffer.

write_array (*varname*, *array*, *iteration=None*)
Write a numpy array as a native NetCDF array

Parameters

varname [str] The variable name to be stored

array [numpy.array of arbitrary dimension] The numpy array to be written

iteration [int, optional, default=None] The local iteration for the module, or *None* if this is a singleton

write_configuration (*varname*, *positions*, *topology*, *iteration=None*, *frame=None*, *nframes=None*)
Write a configuration (or one of a sequence of configurations) to be stored as a native NetCDF array

Parameters

varname [str] The variable name to be stored

positions [simtk.unit.Quantity of size [natoms,3] with units compatible with angstroms] The positions to be written

topology [md.Topology object] The corresponding Topology object

iteration [int, optional, default=None] The local iteration for the module, or *None* if this is a singleton

frame [int, optional, default=None] If these coordinates are part of multiple frames in a sequence, the frame number

nframes [int, optional, default=None] If these coordinates are part of multiple frames in a sequence, the total number of frames in the sequence

write_object (*varname*, *obj*, *iteration=None*)
Serialize a Python object, encoding as pickle when storing as string in NetCDF.

Parameters

varname [str] The variable name to be stored

obj [object] The object to be serialized

iteration [int, optional, default=None] The local iteration for the module, or *None* if this is a singleton

write_quantity (*varname*, *value*, *iteration=None*)
Write a floating-point number

Parameters

varname [str] The variable name to be stored

value [float] The floating-point value to be written

iteration [int, optional, default=None] The local iteration for the module, or *None* if this is a singleton

3.6 Analysis

Analysis methods for perses simulations.

3.6.1 Cache objects

`Analysis`

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index