HPlattice Documentation

Release 1.0

Geoff Rollins

Mar 10, 2020

Contents

HPlattice is a Python library for the HP lattice model of Dill and Chan. It is ideally used as a teaching tool, or as a way to quickly prototype 2D lattice simulation ideas with easy-to-use extensible code. HPlattice can either 1) enumerate conformations, or 2) perform replica exchange monte carlo "dynamics" for 2-dimensional, square-lattice "bead-on-a-string" type chains.

The easiest way to get started with HPlattice is to download and install the Anaconda Python distribution. After installing Anaconda or another Python distribution of your choice, here are the steps for installing HPlattice:

1. Download Bento
2. Install Bento

```
unzip Bento-master.zip
cd Bento-master
python bootstrap.py
./bentomaker configure
./bentomaker build
./bentomaker install
```

3. Download HPlattice (or visit the GitHub Repo)
4. Install HPlattice

```
tar xzf hp-lattice-1.0.tar.gz
cd hp-lattice-1.0
tar xzf HP-sequences.tgz # this step may take 30min to complete
cd hplattice/util
python setup.py build_ext --inplace
cd ../..
bentomaker install
```

5. Run unit tests (optional)

```
conda install pytest # (or pip install pytest if Anaconda not installed)
py.test hplattice
```

6. Try examples

```
cd examples/enumerate
python enumerate.py
cd ../mcrex
python mcrex.py
```

CHAPTER 1

Contents

1.1 User Guide

Release 1.0

Date Mar 10, 2020

1.1.1 Configuration File Format

hplattice simulations require a configuration file. The file should have formatted rows consisting of two fields, separated by white-space (or any non-printing characters, like tabs):

```
HPSTRING                PHPPHPHPPHH
INITIALVEC              [1, 0, 1, 2, 1, 2, 1, 2, 3, 3]
EPS                     -5.0
RESTRAINED_STATE        [(1, 4), (6, 9)]
KSPRING                 0.0
NREPLICAS               9
REPLICATEMPS            [275.0, 300.0, 325.0, 350.0, 400.0, 450.0, 500.0, 600.0,
↪1000.0]
MCSTEPS                 1000
SWAPEVERY               50
SWAPMETHOD              random pair
MOVESET                 MS2
PRINTEVERY              1
NATIVEDIR               ../../HP-sequences/sequences/clist/hp11
STOPATNATIVE            False
```

Here is the full list of the parameters and what each one represents. If a parameter is not specified in the file, it will be set to a default value.

HPSTRING The *HPSTRING* specifies the chemical nature of each monomer. The only supported options are Hydrophobic (H) or Polar (P).

INITIALVEC The *INITIALVEC* is a list of integers that specifies the direction of the chain between two neighboring monomers: 0 (up), 1 (left), 2 (down) or 3 (right). For example, `[0,0,0,0]` would correspond to a 5-mer that points straight up from the origin (in the positive-y direction).

EPS The energy of a hydrophobic contact (two H's in adjacent lattice spaces that are not $i+1$ or $i+2$ neighbors along the chain).

NREPLICAS The number of replicas for replica exchange simulations. This is usually set to `1` for enumeration simulations because those simulations do not involve random moves in conformational space.

REPLICATEMPS A list of floats that specifies the temperature (K) of each replica. The length of the *REPLICATEMPS* list should be equal to *NREPLICAS*.

MCSTEPS The number of monte carlo steps to run. Each replica will run for this number of steps, and they will periodically attempt to swap temperatures.

SWAPEVERY The number of steps between replica swap attempts.

SWAPMETHOD How to swap replicas. `random pair` to randomly choose two replicas to swap; `neighbors` to randomly choose one replica `i` and swap it with its `i+1` neighbor.

MOVESET Select which type of monte carlo moves will be used to sample conformational space: `MS1` for three-bead flips and rigid rotations; `MS2` for three-bead flips, crankshaft moves, and rigid rotations; and `MS3` for rigid rotations only.

RESTRAINED_STATE A list of tuples that specifies contacts that should be harmonically restrained. Each tuple in the list should contain a pair of integers that correspond to the indices of the monomers that should be restrained. An example would be `[(1, 4), (6, 9)]` which would add restraints to the monomer1-monomer4 contact and the monomer6-monomer9 contact. Note that the indices are 0-indexed, so monomer0 is the first monomer in the chain.

KSPRING The force constant of the harmonic restraints specified in *RESTRAINED_STATE*.

PRINTEVERY In a monte carlo simulation, save coordinates to trajectory after this number of steps.

NATIVEDIR The path to the file that specifies what the native contacts are for the chain specified by *HPSTRING*.

STOPATNATIVE If the monte carlo simulation finds the native conformation of the chain (as defined by the contacts in *NATIVEDIR*), then halt the simulation if *STOPATNATIVE* is `True`.

1.1.2 HP Simulations

Replica Exchange Monte Carlo

The basic steps involved in running a monte carlo simulation are:

1. Create a *LatticeFactory*.
2. Use the *LatticeFactory* to load a *Configuration*.
3. Create an *MCSampler*, passing the *LatticeFactory* and *Configuration* as arguments.
4. Call the `do_mc_sampling` method of the *MCSampler*.

```python
from hplattice import LatticeFactory
from hplattice.MCSampler import MCSampler

lattice_factory = LatticeFactory()
config = lattice_factory.make_configuration(filename='mcrex.conf')

```

(continues on next page)

(continued from previous page)

```
mc = MCSampler(lattice_factory, config)
mc.do_mc_sampling(save_trajectory=True, trajectory_filename='traj.xyz')
```

Conformational State Enumeration

The basic steps involved in running an enumeration simulation are:

1. Create a *LatticeFactory*.
2. Use the *LatticeFactory* to load a *Configuration*.
3. Create an *Enumerator*, passing the *LatticeFactory* and *Configuration* as arguments.
4. Call the `enumerate_states` method of the *Enumerator*.

```
from hplattice import LatticeFactory
from hplattice.Enumerator import Enumerator

lattice_factory = LatticeFactory()
config = lattice_factory.make_configuration(filename='enumerate.conf')

en = Enumerator(lattice_factory, config)
en.enumerate_states(save_trajectory=True, trajectory_filename='traj.xyz')
```

1.1.3 Visualizing HP Trajectories in VMD

Output from HPlattice simulations can be easily viewed using the *VMD molecular graphics program <http://www.ks.uiuc.edu/Research/vmd/>*. HPlattice trajectories are saved as xyz coordinate files, which can be loaded by VMD.

To facilitate visualization of the lattice models, HPlattice includes a .vmdrc file. If you copy this file to your home directory and save it as `.vmdrc`, then VMD should automatically apply the styles to any HPlattice .xyz trajectories that you load from the command line.

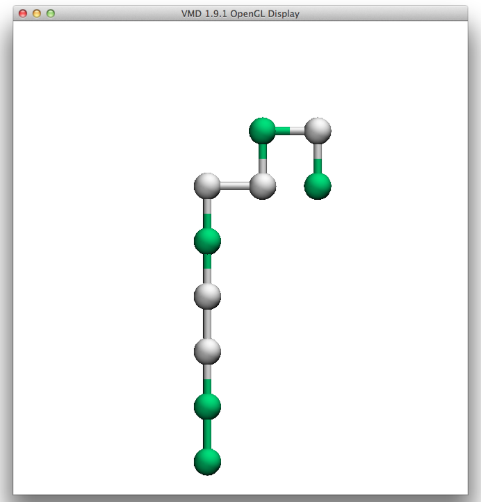
To call VMD from the command line, create a link to the VMD binary. On a mac, the command would be:

```
ln -s /Applications/VMD\ 1.9.1.app/Contents/vmd/vmd_MACOSXX86 /usr/local/bin/vmd
```

Then, if the .vmdrc file is in your home directory, this command would open an HPlattice trajectory in VMD and apply the HPlattice style by default:

```
vmd trajectory.xyz
```

Which should produce a chain that looks like this:



Sometimes VMD draws the topology of the chain incorrectly. In that case, the easiest way to correct it is to enter these commands on the Tk console:

```
topo getbondlist
topo setbondlist {{0 1} {1 2} {2 3} {3 4} {4 5} {5 6} {6 7} {7 8} {8 9}}
```

You'll need to adjust the second command to match the length of your chain. The last pair should be `{N-2 N-1}`, where N is the length of the chain.

1.2 API Reference

Release 1.0

Date Mar 10, 2020

1.2.1 hplattice

class `hplattice.`**`LatticeFactory`**(*chain_cls=<class 'hplattice.Chain.Chain'>, replica_cls=<class 'hplattice.Replica.Replica'>, monty_cls=<class 'hplattice.Monty.Monty'>, trajectory_cls=<class 'hplattice.Trajectory.Trajectory'>, conf_cls=<class 'hplattice.Config.Config'>*)

LatticeFactory objects orchestrate the creation of other objects in the HP model, namely chains, replicas, monte carlo samplers, trajectories, and configurations.

Parameters

- **`chain_cls`** (*`callable`*) – optional, chain class
- **`replica_cls`** (*`callable`*) – optional, replica class
- **`monty_cls`** (*`callable`*) – optional, monte carlo sampler class
- **`trajectory_cls`** (*`callable`*) – optional, trajectory class
- **`conf_cls`** (*`callable`*) – optional, HP configuration class

make_chain (**args*, ***kwargs*)
Make a chain object.

make_configuration (**args*, ***kwargs*)
Make a configuration object.

make_monty (**args*, ***kwargs*)
Make a monte carlo sampler object.

make_replica (**args*, ***kwargs*)
Make a replica object.

make_trajectory (**args*, ***kwargs*)
Make a trajectory object.

1.2.2 hplattice.Chain

class `hplattice.Chain.`**Chain** (*hpstring*, *initial_vec*)
The 2D HP lattice chain. The chain is defined by monomers (beads), which can be either Hydrophobic (H) or Polar (P). By convention, the first monomer in the chain is fixed at the origin on a two-dimensional square lattice.

Adjacent monomers are connected by bonds. The directionality of the bonds is stored in `Vectors`, and each bond-vector is stored as an integer: 0 (up), 1 (left), 2 (down) or 3 (right). A chain with N monomers has $N - 1$ vectors.

In addition to the vector representation of the chain, `Chain` objects also store the 2D coordinates of each monomer. The set of coordinates is stored in a `Coords` object.

Parameters

- **hpstring** (*str*) – String that specifies H or P for each monomer. Example: `PHPPHP`
- **initial_vec** (*list*) – List that specifies the direction of each bond. Example: `[0,0,1,2,1]`, which would correspond to up, up, left, down, left.

class Coords (*num_monomers=0*)
`Coords` is the set of monomer coordinates for a chain.

Parameters num_monomers (*int*) – The number of monomers in the chain.

as_npy_array ()
Convert `Coords` object to `numpy.ndarray`
Returns array of coordinates
Return type `numpy.ndarray`

copy ()
Returns a copy of this object
Return type `Coords`

distance_between_pts (*idx1*, *idx2*)
Compute the cartesian distance between two monomers.
Parameters
- **idx1** (*int*) – the index of a monomer
- **idx2** (*int*) – the index of a monomer

get (*idx=None*)
Get the coordinates of a monomer.
Parameters idx (*int*) – the index of a monomer

grow()
: Append another monomer to C-terminus of the chain, one space above the current C-terminal monomer.

is_viable()
: Check for steric overlap and chain crossovers, neither of which are allowed.
  Returns `True` if no problems are found with the chain.
  Return type bool

pop()
: Remove the last monomer from the C-terminus of the chain.

rotate_down_to_left(*idx*)
: Decrements the x-coord and increments the y-coord of a monomer.
  Parameters **idx** (*int*) – the index of a monomer

rotate_right_to_down(*idx*)
: Decrements the x-coord and decrements the y-coord of a monomer.
  Parameters **idx** (*int*) – the index of a monomer

rotate_up_to_right(*idx*)
: Increments the x-coord and decrements the y-coord of a monomer.
  Parameters **idx** (*int*) – the index of a monomer

set(*coords*, *idx=None*)
: Change the coordinates for all monomers or one monomer if *idx* is defined.
  Parameters
  - **coords** (`numpy.ndarray`) – 2D array of coordinates
  - **idx** (*int*) – (optional), the index of a monomer

vec2coords(*vec*)
: Convert a `Vectors` object into a set of coordinates, and replace the currently stored coordinates with the result.
  Parameters **vec** (`Vectors`) – convert these vectors into a set of coordinates

class Vectors(*vec_list=None*)
: `Vectors` stores the directionality of the bonds between monomers in a *Chain*. Each bond-vector is stored as an integer: 0 (up), 1 (left), 2 (down) or 3 (right). A chain with N monomers has $N - 1$ vectors.

  Parameters **vec_list** (*list*) – List that specifies the direction of each bond. Example: `[0, 0,1,2,1]`, which would correspond to up, up, left, down, left.

as_npy_array()
: Convert `Vectors` object to `numpy.ndarray`
  Returns array of vector values
  Return type `numpy.ndarray`

copy()
: **Returns** copy of this object
  Return type `Vectors`

get(*idx*)
: Get one of the vectors.
  Parameters **idx** (*int*) – the index of a vector

grow()
: Append another vector to C-terminus of the chain, pointing up.

increment(*idx*)
: Rotate a vector clockwise.
  Parameters **idx** (*int*) – the index of a vector

pop()
Remove the last vector from the C-terminus of the chain.

set(*vec*)
Change all of the vectors.
Parameters **vec** (`numpy.ndarray`) – array of vector values

set_idx(*idx*, *value*)
Change one of the vectors.
Parameters
- **idx** (*int*) – the index of the vector to change
- **value** (*int*) – the value to change the vector to

contactstate()
Find all contacts between pairs of `H` monomers that are separated by at least three positions along the chain, i.e. will not include `(i, i+1)` and `(i, i+2)` pairs.

Returns list of `(idx1,idx2)` contacts (tuples)

Return type list

do_crankshaft(*vecindex*)
Swap the values of a pair of chain vectors that are next-nearest neighbors.

Parameters **vecindex** (*int*) – swap `vecindex` with `vecindex+2`

do_rigid_rot(*vecindex*, *direction*)
Rotate a chain vector clockwise or counterclockwise. The possible clockwise rotations are:

0 –> 1

1 –> 2

2 –> 3

3 –> 0

Parameters

- **vecindex** (*int*) – a valid vector index
- **direction** (*int*) – `1` for clockwise, `-1` for counterclockwise

do_three_bead_flip(*vecindex*)
Swap the values of a pair of neighboring chain vectors.

Parameters **vecindex** (*int*) – swap `vecindex` with `vecindex+1`

energy(*epsilon=0.0*)
Compute energy of chain, based on hydrophobic contacts.

Parameters **epsilon** (*float*) – the energy of one hydrophobic contact.

Returns the total energy of the chain.

Return type float

get_coord_array()

Returns the coordinates of the chain

Return type `numpy.ndarray`

get_hp_string()

Returns example `HPHPHPPPHHP`

Return type string

get_vec_length()

Returns The number of vectors in the chain, which should be one less than the number of monomers.

Return type int

grow()

Add a new monomer to C-terminus of the chain.

hpstr2bin()

Convert a string of type `HPHPHPPPHHP` to a list of 1s and 0s.

is_first_vec_one()

Returns `True` if the first vector points to the right

Return type bool

is_viable()

Check for steric overlap and chain crossovers, neither of which are allowed.

Returns `True` if no problems are found with the chain.

Return type bool

nextviable()

Check for steric overlap and chain crossovers in the not-yet-accepted next configuration of the chain. This is usually called before accepting a monte carlo move to make sure that the chain doesn't end up in a disallowed state.

Returns `True` if no problems are found with the chain.

Return type bool

nonsym()

Many of the conformations are related by rotations and reflections. We define a "non-symmetric" conformation to have the first direction `0` and the first turn be a `1` (right turn)

Returns `1` if the chain is non-symmetric, `0` otherwise.

Return type int

shift()

Shifts the chain vector to the 'next' list, according to an enumeration scheme where the most distal chain vector is incremented 0->1->2->3. After 3, the most distal vector element is removed, and the next most distal element is incremented. If there are multiple "3" vectors, this process is done recursively.

Example:

[0,0,0,0] –> [0,0,0,1]

[0,0,1,2] –> [0,0,1,3]

[0,1,0,3] –> [0,1,1]

[0,3,3,3] –> [1]

This operation is very useful for enumerating the full space of chain conformations.

Returns `1` if no more shifts are possible and `0` otherwise.

Return type int

update_chain()
Accept recent chain move. This is usually called after a trial monte carlo move to accept the chain perturbation.

vec2coords()
Convert a `Vectors` object into a set of coordinates, and replace the currently stored coordinates with the result.

Parameters **vec** (`Vectors`) – generate coordinates from current vectors

1.2.3 hplattice.Config

class hplattice.Config.**Config**(*filename=None*)
A data structure to hold all the configuration data for an HP model calculation.

Parameters **filename** (*str*) – optional, path to configuration file

print_config()
Output the values of the configuration variables.

read_configfile(*filename*)
Read in configuration parameters from file. The file should have formatted rows consisting of two fields, separated by white-space (or any non-printing characters, like tabs):

```
HPSTRING                PHPPHPPPHP
INITIALVEC              [0,0,0,0,0,0,0,0,0]
....
```

Parameters **filename** (*str*) – path to configuration file

1.2.4 hplattice.Enumerator

class hplattice.Enumerator.**Enumerator**(*lattice_factory*, *config*)
Enumerator objects are used to enumerate all conformations of an HP chain. The HP chain is defined in a configuration file, specified by the *config* parameter.

Parameters

- **lattice_factory** (*hplattice.LatticeFactory*) – factory object that knows how to create chains and trajectories
- **config** (*str*) – path to configuration file

enumerate_states(*save_trajectory=False*, *trajectory_filename='traj.xyz'*)
Enumerate all conformations of an HP chain. Prints density of contact states to stdout.

Parameters

- **save_trajectory** (*bool*) – Generate an xyz coordinate trajectory when `True`.
- **trajectory_filename** (*str*) – optional, save trajectory to this path

1.2.5 hplattice.MCSampler

class hplattice.MCSampler.**MCSampler**(*lattice_factory*, *config*)
MCSampler objects are used to run replica exchange monte carlo simulations of an HP chain. The HP chain is

defined in a configuration file, specified by the *config* parameter. The configuration file also specifies the replica exchange parameters, such as the temperature of each replica.

Parameters

- **`lattice_factory`** (*`hplattice.LatticeFactory`*) – factory object that knows how to create replicas and trajectories.
- **`config`** (*`str`*) – path to configuration file

`do_mc_sampling`(*save_trajectory=False*, *trajectory_filename='traj.xyz'*)
Run replica exchange monte carlo of the HP chain.

Parameters

- **`save_trajectory`** (*`bool`*) – Generate xyz coordinate trajectories when `True`. There will be separate trajectory for each replica.
- **`trajectory_filename`** (*`str`*) – optional, save trajectory to this path. Replica numbers will be prepended to the name specified here.

1.2.6 hplattice.Monty

class `hplattice.Monty.`**`DistRestraint`**(*contacts*, *kspring*)
A harmonic constraint based on squared distance $D = d^2$, where $D = \sum_{i,j} d_{ij}^2$ over all contacts.

Parameters

- **`contacts`** (*`list`*) – list of tuples, example `[(0, 4), (1, 6)]`
- **`kspring`** (*`float`*) – spring constant for restraint

`D`(*chain*)
Compute the sum of squared-distances of a given chain.

Parameters **`chain`** (*`hplattice.Chain.Chain`*) – Compute the sum of squared-distances of this chain.

Returns the sum of squared-distances over the selected contacts.

Return type float

`energy`(*chain*)
Compute the restraint energy of a given chain.

Parameters **`chain`** (*`hplattice.Chain.Chain`*) – Compute the restraint energy of this chain.

Returns energy of the distance restraint

Return type float

class `hplattice.Monty.`**`Monty`**(*config*, *temp*, *chain*)
A collection of functions to perform Monte Carlo move-set operations on an HP chain.

Parameters

- **`config`** (*`hplattice.Config.Config`*) – configuration parameters for chain and simulation
- **`temp`** (*`float`*) – temperature (K)
- **`chain`** (*`hplattice.Chain.Chain`*) – do monte carlo on this chain

`kT`()

Returns $k_b * T$

Return type float

metropolis(*replica*)

Judge the next conformation of the chain according to Metropolis criterion: $e^{-\Delta E/kT}$.

Parameters **replica** (*hplattice.Replica.Replica*) – The replica containing the chain that should be judged.

Returns `True` if next conformation should be accepted.

Return type bool

move1(*chain*, *vecindex=None*, *direction=None*)

Apply moveset MC1 (Dill and Chan, 1994, 1996) to the chain:

1. three-bead flips
2. rigid rotations

Parameters

- **chain** (*hplattice.Chain.Chain*) – apply move to this chain
- **vecindex** (*int*) – optional, vector to move. will be chosen randomly if no value is specified.
- **direction** (*int*) – optional, `1` for clockwise, `-1` for counterclockwise. will be chosen randomly if no value is specified.

move2(*chain*, *vecindex=None*, *direction=None*, *moveseed=None*)

Apply moveset MC2 (Dill and Chan, 1994, 1996) to the chain:

1. three-bead flips
2. crankshaft moves
3. rigid rotations

Parameters

- **chain** (*hplattice.Chain.Chain*) – apply move to this chain
- **vecindex** (*int*) – optional, vector to move. will be chosen randomly if no value is specified.
- **direction** (*int*) – optional, `1` for clockwise, `-1` for counterclockwise. will be chosen randomly if no value is specified.
- **moveseed** (*float*) – optional, *moveseed* $< 1/3$ for three-bead flip; $1/3 <=$ *moveseed* $< 2/3$ for crankshaft; $2/3 <=$ *moveseed* for rigid rotation.

move3(*chain*, *vecindex=None*, *direction=None*)

Apply rigid rotations to the chain:

1. rigid rotations

Parameters

- **chain** (*hplattice.Chain.Chain*) – apply move to this chain
- **vecindex** (*int*) – optional, vector to move. will be chosen randomly if no value is specified.

- **direction** (*int*) – optional, `1` for clockwise, `-1` for counterclockwise. will be chosen randomly if no value is specified.

`hplattice.Monty.`**`random`**`()` → x in the interval [0, 1).

1.2.7 hplattice.Trajectory

class `hplattice.Trajectory.`**`Trajectory`**`(`*save_trajectory*, *trajectory_filename*, *open_stream_fcn=<function open_file_stream>*`)`

Trajectory objects write chain coordinates to output streams. They used to record the progress of an enumeration or monte carlo simulation.

Parameters

- **save_trajectory** (*bool*) – `True` if trajectory should be saved to output stream.
- **trajectory_filename** (*str*) – write trajectory to this path
- **open_stream_fcn** (*callable*) – optional, call this function to open an output stream

`finalize``()`

Close any open output streams.

`snapshot``(`*chain*`)`

Record coordinates of chain to output stream.

Parameters **chain** (*hplattice.Chain.Chain*) – Save coords of this chain.

`hplattice.Trajectory.`**`open_file_stream`**`(`*filename*`)`

Helper function to open a file for writing.

Parameters **filename** (*str*) – open this path for writing

1.2.8 hplattice.Replica

class `hplattice.Replica.`**`Replica`**`(`*lattice_factory*, *config*, *repnum*, *nativeclist=None*`)`

Each *Replica* is a container for a chain and a monte carlo sampler. During replica exchange simulations, the replicas attempt to swap temperatures at regular intervals. The success of a swap depends on the energies and temperatures of the replicas.

Parameters

- **lattice_factory** (*hplattice.LatticeFactory*) – factory object that knows how to create chains and monty samplers.
- **config** (*str*) – path to configuration file
- **repnum** (*int*) – replica number
- **nativeclist** (*list*) – optional, native contacts as a list of tuples, example `[(0, 4), (1, 6)]`

`compute_mc_acceptance``()`

Compute the fraction of mc moves that have been viable and the fraction of viable moves that have been accepted.

`contactstate``()`

Get contacts of current chain conformation.

Returns list of tuples, example `[(0, 4), (1, 6)]`

Return type list

energy()
Compute energy of current chain conformation.

Returns energy

Return type float

get_T()
Get current temperature.

Returns temperature

Return type float

get_vec()
Get chain vectors.

Returns chain vectors

Return type *hplattice.Chain.Chain.Vectors*

init_mc_stats()
Initialize stats about viability and acceptance of monte carlo moves.

is_native()
Check if current contacts match contacts of native state.

Returns `True` if current contacts match native contacts.

Return type bool

kT()

Returns $k_b * T$

Return type float

metropolis_accept_move()
Apply metropolis criterion to determine whether new conformation of chain should replace current conformation.

Returns `True` if new conformation should be accepted.

Return type bool

propose_move()
Do a monte carlo move to produce a new conformation of the chain.

Returns `True` if new conformation is viable.

Return type bool

record_stats(*move_is_viable*, *move_is_accepted*)
Update mc move stats based on outcome of most recent move.

Parameters

- **move_is_viable** (*bool*) – `True` if the move was a viable, non-overlapping chain conformation.
- **move_is_accepted** (*bool*) – `True` if the move was accepted as the new conformation of the chain.

hplattice.Replica.**attemptswap**(*swap_method*, *replicas*)
Attempt swap of replicas.

Parameters

- **`swap_method`** (*`str`*) – `'random pair'` to randomly choose two replicas to swap; `'neighbors'` to randomly choose one replica `i` and swap it with its `i+1` neighbor.
- **`replicas`** (*`list`*) – list of *`Replica`* objects

Returns the indices of the two replicas and the success or failure of the swap.

Return type (int, int, bool)

`hplattice.Replica.`**`random`**`()` → x in the interval [0, 1).

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index