HelixMC Documentation

Release 0.9

Fang-Chieh Chou

May 13, 2016

Contents

1 HelixMC Tutorial **3**
- 1.1 Install . . . 3
- 1.2 Run HelixMC . . . 4
- 1.3 Other Examples . . . 5
- 1.4 Base-pair Step Parameters Database . . . 6
- 1.5 References . . . 6

2 HelixMC Reference **7**
- 2.1 Code Organization . . . 7
- 2.2 Constant Random Seed . . . 7
- 2.3 Helix Pose . . . 8
- 2.4 Random Base-pair Steps Generators . . . 10
- 2.5 Score Functions . . . 14
- 2.6 Utility Functions . . . 17
- 2.7 Useful Fitting Functions . . . 25

3 About us **31**
- 3.1 Contacts . . . 31
- 3.2 Links . . . 31
- 3.3 Citing HelixMC . . . 31
- 3.4 Funding . . . 31

Bibliography **33**

HelixMC is a software package for Monte-Carlo (MC) simulations of DNA/RNA helices using the base-pair level model. It provides a powerful tool to understand the flexibility of DNA/RNA helices through numerical simualtions.

The base-pair level model, first developed by Olson and collegues, bridges between the simple elastic rod model and the full-atom representation, providing a reasonably sophiscated and yet computationally tractable way to model long DNA/RNA helices up to thousands of base-pairs and to evalute their mechanical properties. HelixMC has the utility of applying external stetching forces and torques to the helix, and measuring the helix extension and the rotation of the helix (known as the linking number) during the process. This properly emulate the setup of recent single-molecule tweezers experiments, making HelixMC a useful tool for direct simulations of these experiments.

HelixMC is coded in Python in an object-oriented fashion. Rate-limiting core computations are speeded up using Cython. Therefore HelixMC provides a framework that is easy to use and to extend, as well as being reasonably fast when it comes to large-scale computations.

CHAPTER 1

HelixMC Tutorial

Release 0.9

Date May 13, 2016

This tutorial demonstrates how to install and run simple calculations with HelixMC, and breifly summarizes available examples and bp-step paramerter database. For details on the classes and functions availble, please see *HelixMC Reference*.

1.1 Install

Currently HelixMC has only been tested on Linux. It should run on other Unix-like system (e.g. Mac OS X). The following packages are also required by HelixMC. The versions we used are given in parathesis.

- Python (2.7.3)
- Numpy (1.6.1)
- Matplotlib (1.1.0)
- A working C/C++ compiler. Here we used GCC (4.6.3)

The easiest way to setup the python environment is to use latest Enthought Python Distribution (http://www.enthought.com/).

The easiest way to install is to use pip install:

```
$ pip install helixmc
```

Alternatively, one can download the source code from the latest GitHub repository. Simply run:

```
$ git clone https://github.com/fcchou/HelixMC.git
```

Or you can go to https://github.com/fcchou/HelixMC/ and download the source code by clicking the "Download ZIP" button.

After this, you can instal HelixMC using *setup.py*:

```
$ python setup.py build
$ sudo python setup.py install
```

Instead of installing using setup.py, you can just add your HelixMC folder into the system's `$PATH` and `$PYTHONPATH`. In bash this can be done by adding the following lines to your `~/.bashrc`:

```
export PATH=$PATH:<HelixMC Path>
export PYTHONPATH=$PYTHONPATH:<HelixMC Path>
```

Then build the Cython extension. Under the `helixmc/` folder, run:

```
$ python _cython_build.py build_ext --inplace
```

Note that this requires you to have Cython installed. Otherwise you can choose to build the c source file, then you do not need Cython:

```
$ python _c_build.py build_ext --inplace
```

Now you should be all set. To test the install, simply run:

```
$ helixmc-run --help
```

This should output the help information for `helixmc-run` application.

1.2 Run HelixMC

The *helixmc-run* application wraps the classes and functions of HelixMC to allow simple MC job submissions from command line.

A detailed help for all options of `helixmc-run` can be obtained by running:

```
$ helixmc-run --help
```

Now we demonstrate a simple example for `helixmc-run`.

First, run the following command to kick out a MC run:

```
$ helixmc-run -params DNA_default.npz -n_bp 100 -n_step 10 -seq GCCG \
-force 5 -compute_fuller_link -out_frame test_run
```

Here, `-params` give the input database file that contains bp-step parameters curated from PDB (by default it searches the helixmc database folder if it does not find the input file). `-n_bp` is the total number of bp in the helix. `-n_step` is the number of MC steps. `-seq` gives the sequence of the nucleic acids (ATCG for DNA and AUCG for RNA). `-force` is the applied z-direction stretching force. `-compute_fuller_link` tells HelixMC to compute and store the linking number using Fuller's approximation *[R1]*. `-out_frame` option will make HelixMC save the final frame to disk as `test_run.npz` in this case.

Depending on your machine it will take a few seconds to run. After completion you should see something like:

```
Total time = 1.941772
Accept rate = 0.787879
```

It is advisable to check the *Accept rate* of each run, and make sure it is not too low. As a rule of thumb, if *Accept rate* < 0.1, this means most MC moves are rejected, and you will need use a higher-than-normal number of MC steps to acheive the same level of sampling.

Now we can analyze the output data. Open a Python session and enters the following:

```
>>> import numpy as np
>>> from helixmc.pose import HelixPose
```

The observables for each frame are stored in `MC_data.npz`. Normally the coordinates and reference frames of the last bp are recorded. If `-compute_fuller_link` or `-compute_exact_link` is used, the twist and writhe of the helix will also be stored (note that link = twist + writhe).

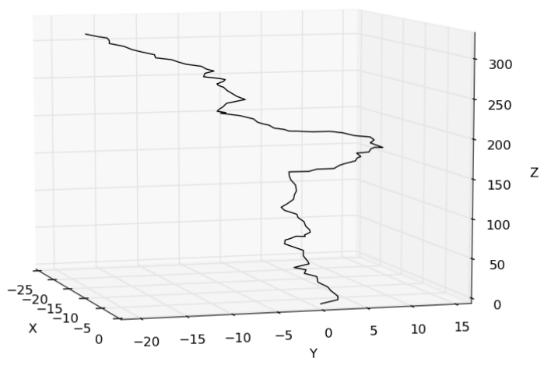
For example I can compute the average z-extension and the average link as follows:

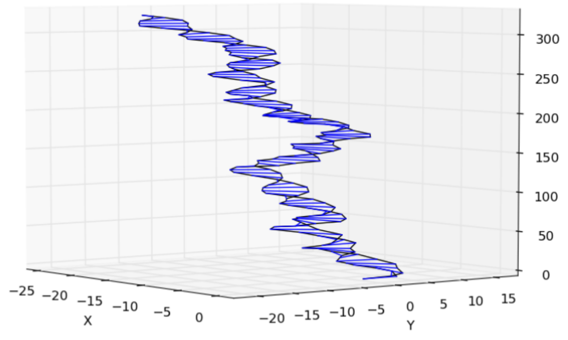
```
>>> data = np.load('MC_data.npz')
>>> data.files
['coord_terminal', 'twist', 'writhe', 'frame_terminal']
>>> data['coord_terminal'][:,2]  # 2 for the z-elements
array([ 309.06198311,  317.92717085,  320.17158221,  304.42561971,
        319.07461907,  306.94162915,  314.7566295 ,  319.04106375,
        322.42125203,  325.72718993])
>>> np.average(data['coord_terminal'][:,2])  # avg. z-extension in Å
315.95487393228649
>>> np.average(data['twist'] + data['writhe'])  # avg. link in radian
60.648749666780688
```

Remember we stored the final frame of the simulation to `test_run.npz`. We will now plot the helix using that:

```
>>> pose = HelixPose('test_run.npz')
>>> pose.plot_centerline()  # plot the centerline
>>> pose.plot_helix()  # plot the entire helix
```

You should see something similar to the following





This is the end of the example. For more examples, check the `examples/` folder in HelixMC, which is briefly summarized below.

1.3 Other Examples

Here is a list of examples in the `examples/` folder.

force_ext This is just the example above.

link_cst This is for link-contrained simulation, similar to the torsioal-trap single-molecule experiment *[R2]*.

z-dna Simulation of Z-DNA using `helixmc-run`.

fuller_check Check the if the Fuller's approximation is correct in certain criteria.

data_fitting How to use `helixmc.fitfxn` to fit simulation or experiment data to simple analytical models.

helixplot More examples for plotting the helices.

lp_olson How to perform alternative evaluation of bending persistence length using the method suggested by Olson et al. *[R3]*.

bp_database Examples on curating bp-step parameters from PDB.

1.4 Base-pair Step Parameters Database

In the `helixmc/data/` folder, several different bp-step parameter sets are given. These datasets were all extracted from structures in Protein Data Bank (PDB, http://www.pdb.org/), with different selection and filtering. The list below summarizes these data.

DNA_default B-DNA data from structures with resolution (Rs) <= 2.8 Å, excluding protein-binding models.

DNA_2.8_all A-DNA + B-DNA, Rs <= 2.8 Å, including protein-binding models.

DNA_2.0_noprot B-DNA, Rs <= 2.0 Å, excluding protein-binding models.

RNA_default RNA, Rs <= 2.8 Å, excluding protein-binding models.

RNA_2.8_all RNA, Rs <= 2.8 Å, including protein-binding models.

RNA_2.0_noprot RNA, Rs <= 2.0 Å, excluding protein-binding models.

Z-DNA Z-DNA, Rs <= 2.8 Å, including protein-binding models.

***unfiltered** Unfiltered datasets (no filtering of histogram outliers).

DNA_gau Single 6D Gaussian built from DNA_default.

RNA_gau Single 6D Gaussian built from RNA_default.

DNA_gau_graft Chimera dataset with mean from DNA_gau and covariance from RNA_gau.

RNA_gau_graft Chimera dataset with mean from RNA_gau and covariance from DNA_gau.

***gau_refit** Manually refitted datasets to match experimental measurements.

***_2.8_all_?bp** Multi-bp datasets derived from the *2.8_all* pdb lists.

Note that Gaussian dataset (`*gau*.npy`) must be loaded with `-gaussian_params` tag in `helixmc-run` command line (instead of `-params`). Also Gaussian dataset does not support sequence specific simulations.

The corresponding lists of PDB models being used are given in the `helixmc/data/pdb_list/` folder.

These datasets are in npy/npz format (Numpy array/archive). For the npz files, the data for different bp-steps of different sequences were separated into different arrays in the file. For B-DNA and RNA, parameter sets with Rise >= 5.5 Å or Twist <= 5° were thrown away as outliers. Then, parameter sets with values beyond 4 standard deviations away from the mean for any of the 6 bp-step parameters were also removed. For B-DNA (except `DNA_2.8_all`, where the protein binding makes A-DNA and B-DNA unseparable), we further clustered the data using k-means algorithm to separate the A-DNA and B-DNA data. Note that these filtering steps are skipped in the unfiltered datasets.

For Z-DNA, we only considered two types of bp-steps: CG and GC. We used the following selection criteria: Twist <= -30° for GC, and -30° < Twist <= 5° for CG. For CG bp-steps, we further filtered the data by only keeping parameter sets with 4.5 Å <= Rise < 6.3 Å. Parameter sets with values beyond 4 standard deviation away from the mean were then removed, similar to the above cases.

See also `examples/bp_database/` for a detailed example for the curation of `DNA_2.0_noprot.npz`.

1.5 References

HelixMC Reference

Release 0.9

Date May 13, 2016

This reference document details functions, modules, and objects included in HelixMC, describing what they are and what they do. For learning how to use HelixMC, see also *HelixMC Tutorial*.

2.1 Code Organization

HelixMC is coded in Python in an object-oriented fashion, allowing easy usage and extension. Here we briefly summarizes the organization of the code.

First, *HelixPose* object stores all information of the current helix conformation. Various properties, e.g. coordinates, twist, writhe, etc. can be directly accessed from the object. The object also contains functions that allow one to update the conformation and to plot the helix.

Second, *RandomStep* are used to generate random samples of base-pair step parameters. *RandomStepSimple* takes in a list of database parameter sets, and can randomly emit one of input parameter set, or construct a multivariate Gaussian and emit samples from the distribution, upon the choice of the user. *RandomStepAgg* aggregates multiple *RandomStep* objects into one, allow easy handling of sequence-dependent sampling (by aggregating several *RandomStepSimple* for different sequences). The user can also create their own *RandomStep* objects by inheriting from *RandomStepBase*.

Third, *Score* objects take a *HelixPose* and scores it. The scoring can then be used to decide whether a Monte-Carlo move should be accepted. Currently we have 3 simple score terms *ScoreExt*, *ScoreTorsionTrap* and *ScoreXyTrap*, which scores a HelixPose under Z-extension, torsional trap and xy horizontal trap respectively. These 3 score terms are summarized into *ScoreTweezers*, which is a sub-class of *ScoreAgg*, an aggregator class that combines multiple score functions. *ScoreTweezers* is the workhorse score functions currently in used. The user can also define their own scoring by inheriting from *ScoreBase*.

Last, the *util* module contains useful functions for evaluating twist and writhe, for conversion between bp-step parameters and cartesian translation and rotation operation, and so on. The *fitfxn* module is a standalone module that contains a few widely used analytical fitting functions based on the elastic rod model.

2.2 Constant Random Seed

`constant_seed`([seed])	Set constant random seed.

2.2.1 helixmc.constant_seed

`helixmc.`**`constant_seed`**(*seed=24601*)

Set constant random seed.

Parameters **seed** : int, optional

Random seed.

2.3 Helix Pose

pose.HelixPose(params[, frame0, compute_tw_wr])	Pose object storing the state info of the helix.

2.3.1 helixmc.pose.HelixPose

class `helixmc.pose.`**`HelixPose`**(*params*, *frame0=None*, *compute_tw_wr=False*)

Pose object storing the state info of the helix.

Parameters **params** : ndarray of (N,6)

List of all bp-step parameters. A N bp helix has (N-1)*6 parameters.

frame0 : ndarray of (3,3), optional

The frame of the first base-pair, default is np.eye(3) (overlaps with global coordinate).

compute_tw_wr : bool, optional

Whether to compute twist and writhe (Fuller writhe) during system update. Should be set to True for link-constrained simulations.

Attributes

compute_tw_wr	(bool) See Parameters section above.
writhe_exact	(float) Exact writhe of the helix.
writhe_fuller	(float) Fuller's writhe of the helix.
twist	(float) Supercoiling twist of the helix.
coord_terminal	(1d array of (3)) Coordinate (x,y,z) of the center of last base-pair.
frame_terminal	(ndarray of (3,3)) Coordinate frame of the last base-pair. Each column represent the cooresponding axis (frame[:,0] is the x-axis etc.)
z_terminal	(float) Z-component of coord_terminal
link_fuller	(float) Link of the helix computed using Fuller's approximation.
link_exact	(float) Exact link of the helix.
n_bp	(int) Number of base-pairs in the helix.
coord	(ndarray of (N,3)) Coordinates of all base-pairs in the helix (n_bp entries).
dr	(ndarray of (N,3)) Delta-r vectors of the entire helix (n_bp-1 entries).
frames	(ndarray of (N,3,3)) Frames of all base-pairs in the helix (n_bp entries).
params	(ndarray of (N,6)) List of all bp-step parameters in the helix (n_bp-1 entries).
rb_vec	(ndarray of (N,3)) Ribbon vectors of all base-pairs in the helix (n_bp entries).

`__init__`(*params*, *frame0=None*, *compute_tw_wr=False*)

`accept_update``()`
Accept a trial update.

`copy``()`
Return a copy of the current pose.

Returns **pose_copy** : HelixPose

Copy of the current pose.

classmethod **`from_file`**`(`*input_file*, *compute_tw_wr=False*`)`
Load pose data from an input file.

Parameters **input_file** : str

Input file (.npz) that stores the helix pose data.

compute_tw_wr : bool, optional

Whether to compute twist and writhe (Fuller writhe) during system update. Should be set to True for link-constrained simulations.

Raises **ValueError**

If n_bp < 2 in the input file.

`guess_twist_center``()`
Attempt to guess the twist center value. Use circmean of the current twists.

`plot_centerline``(`*color='k'*, *show=True*, *fig_ax=None*`)`
Plot the helix center-line using matplotlib + mplot3d.

Parameters **color** : str, optional

Colors of the centerline (matplotlib color codes).

show : bool, optional

Whether to invoke pyplot.show() method at the end.

fig_ax : mplot3d.Axes3D, optional

Input mplot3d.Axes3D object. For ploting multiple helix on the same graph.

`plot_helix``(`*rb_width=5.0*, *color='kb'*, *show=True*, *fig_ax=None*`)`
Plot the helix using matplotlib + mplot3d.

Parameters **rb_width** : float, optional

Width of the helix ribbon in Å.

color : str, optional

Colors of the plot (matplotlib color codes), first letter for backbond and second letter for base-pair.

show : bool, optional

Whether to invoke pyplot.show() method at the end.

fig_ax : mplot3d.Axes3D, optional

Input mplot3d.Axes3D object. For ploting multiple helix on the same graph.

`reject_update``()`
Reject a trial update.

`set_params` (*params*, *frame0=None*)

Set the bp-step params of the HelixPose.

Parameters **params** : ndarray of (N,6), optional

List of all bp-step parameters. Number of params should equal to n_bp - 1.

frame0 : ndarray of (3,3)

The frame of the first base-pair, default is np.eye(3).

`update` (*i*, *params*, *o=None*, *R=None*)

Full update of the i-th bp-step.

Parameters **i** : int

The index of the bp-step to be updated.

params : ndarray

Input base-pair step parameter set.

o : ndarray, optional

Coordiantes for the bp-centers of the 2nd base-pair.

R : ndarray, optional

Rotational matrix for the transformation.

Raises **ValueError**

If i < 0 or i >= n_bp.

`update_trial` (*i*, *params*, *o=None*, *R=None*)

Trial update of the i-th bp-step. Following accept_update or reject_update is required.

Parameters **i** : int

The index of the bp-step to be updated.

params : ndarray

Input base-pair step parameter set.

o : ndarray, optional

Coordiantes for the bp-centers of the 2nd base-pair.

R : ndarray, optional

Rotational matrix for the transformation.

Raises **ValueError**

If i < 0 or i >= n_bp.

`write2disk` (*filename*)

Write the current pose to disk.

Parameters **filename** : str

Name of the output file.

2.4 Random Base-pair Steps Generators

random_step.RandomStepBase()	Base class for Random bp-step generator, for inheritence only.
random_step.RandomStepSimple([params, ...])	Simple random bp-step generator.
random_step.RandomStepAgg([data_file, ...])	Random bp-step generator by aggregating multiple independent simple bp-s

2.4.1 helixmc.random_step.RandomStepBase

class `helixmc.random_step.`**`RandomStepBase`**

Base class for Random bp-step generator, for inheritence only.

See also:

RandomStepSimple Simple random bp-step generator.

RandomStepAgg Random bp-step generator by aggregrating multiple independent bp-step generators.

`__call__`()

`__init__`()

2.4.2 helixmc.random_step.RandomStepSimple

class `helixmc.random_step.`**`RandomStepSimple`**(*params=None*, *params_cov=None*, *params_avg=None*, *gaussian_sampling=True*)

Simple random bp-step generator. Pick random datasets from database or sample from a multivariate Gaussian built from the database.

Parameters **params** : ndarray, shape (N,6), optional

Input base-pair step parameters. Usually obtained by analyzing the structures in PDB. Must be specified if params_cov and params_avg is not being input, or if gaussian_sampling is set to False. Order = [Shift, Slide, Rise, Tilt, Roll, Twist] Distance in unit of Å, angle in unit of radians.

params_cov : ndarray, shape (6,6), optional

Covariance matrix of the multivariate Gaussian for step parameters. Used for Gaussian sampling.

params_avg : ndarray, shape (6,6), optional

Averages of the multivariate Gaussian for step parameters. Used for Gaussian sampling.

gaussian_sampling : bool, optional

Whether to sample assuming a multivariate Gaussian. Default is True.

Raises **ValueError**

If params is not specified, and either params_avg and params_cov are not specified. If params is not specified, but gaussian_sampling is set to False.

See also:

RandomStepBase Base class for Random bp-step generator, for inheritence only.

RandomStepAgg Random bp-step generator by aggregrating multiple independent bp-step generators.

Attributes

gaussian_sampling	(bool) See the Parameters section.
params_avg	(ndarray) See the Parameters section.
params_cov	(ndarray) See the Parameters section.
params	(ndarray) See the Parameters section. Return None if not specified at init.

`__call__`()

Draw one random bp-step parameter set from the distribution.

Returns params: ndarray, shape (6)

Randomly generated bp-step parameter set.

o : ndarray, shape (3)

Corresponding bp-center for the 2nd bp of the bp-step (1st bp is at origin and oriented with the coordinate frame).

R : ndarray, shape (3,3)

Corresponding frame for the 2nd bp of the bp-step.

`__init__`(*params=None*, *params_cov=None*, *params_avg=None*, *gaussian_sampling=True*)

classmethod `load_gaussian_params`(*filename*)

Load a single Gaussian parameter file from disk.

Parameters filename : str

Name of the Gaussian parameter file, in .npy or plain text format. First row is the mean parameters and the following six rows represents the covariance matrix. The routine will look for helixmc/data/ if the file is not found in current folder. Order = [Shift, Slide, Rise, Tilt, Roll, Twist] Distance in unit of Å, angle in unit of radians.

Returns random_step : RandomStepSimple

Random step generator corresponds to the input file.

classmethod `load_params`(*filename*, *gaussian_sampling=True*)

Load a simple bp-step parameter file from disk.

Parameters filename : str

Name of the parameter file, in .npy or text format. It should be a (N * 6) numpy array. Each row contains the 6 step parameters. The routine will look for helixmc/data/ if the file is not found in current folder. Order = [Shift, Slide, Rise, Tilt, Roll, Twist] Distance in unit of Å, angle in unit of radians.

gaussian_sampling : bool, optional

Whether to sample assuming a multivariate Gaussian.

Returns random_step : RandomStepSimple

Random step generator corresponds to the input file.

2.4.3 helixmc.random_step.RandomStepAgg

class `helixmc.random_step.`**`RandomStepAgg`**(*data_file=None*, *gaussian_sampling=True*)

Random bp-step generator by aggregating multiple independent simple bp-step generators. Useful for sequence dependence simulations.

Parameters **data_file** : str, optional

Pre-curated database file with sequence dependence in .npz format.

gaussian_sampling : bool, optional

Whether to sample assuming a multivariate Gaussian. Default is True.

See also:

`RandomStepBase` Base class for Random bp-step generator, for inheritence only.

`RandomStepSimple` Simple random bp-step generator.

Attributes

gaussian_sampling	(bool) See the Parameters section.
params_avg	(ndarray) Average values for the step parameters distribution.
rand_list	(list) List of all RandomStep objects in the aggregation.
names	(list) List of all names of RandomStep in the aggregation.

`__call__`(*identifier=None*)

Draw one random bp-step parameter set from the distribution. Randomly select one generator in the aggregation and return its generated result if no input parameter is given.

Parameters **identifier** : int or str

Index or name of the requested random step generator.

Returns params: ndarray, shape (6)

Randomly generated bp-step parameter set.

o : ndarray, shape (3)

Corresponding bp-center for the 2nd bp of the bp-step (1st bp is at origin and oriented with the coordinate frame).

R : ndarray, shape (3,3)

Corresponding frame for the 2nd bp of the bp-step.

Raises **TypeError :**

If the input identifier is not int or str

`__init__`(*data_file=None*, *gaussian_sampling=True*)

`append_random_step`(*name*, *random_step*)

Append one additional random bp-step generator to the aggregation.

Parameters **name** : str

Name of the random bp-step generator.

random_step : subclass of RandomStepBase

The random bp-step generator to be added to the aggregation

Raises **TypeError**

If random_step does not belong to a subclass of *RandomStepBase*.

ValuError

If the input name already exists in self.names.

clear_all()
Clear all random bp-step generator in the aggregation.

get_rand_step(*identifier*)
Return one RandomStep object stored in the aggregation.

Parameters identifier : int or str

Index or name of the requested random step generator.

Returns rand_step : subclass of RandomStepBase

RandomStep object stored in the aggregation.

Raises TypeError :

If the input identifier is not int or str

load_from_file(*data_file*)

Load data file in .npz format and append to the current aggregation. The routine will look for helixmc/data/ if the file is not found in current folder.

Parameters data_file : str, optional

Pre-curated database file with sequence dependence in .npz format.

name2rand_idx(*name*)
Get the index of a RandomStep object in rand_list from its name.

Parameters name : str

Name of the RandomStep in the aggregation.

Returns idx : int

The corresponding index of the RandomStep object.

2.5 Score Functions

score.ScoreBase()	Base class for scoring fucntion, for inheritence only.
score.ScoreExt(force)	Score function for force-extension along Z-axis.
score.ScoreTorsionTrap(stiffness, target_link)	Score function for torsional trap.
score.ScoreXyTrap(stiffness)	Score function for xy trap.
score.ScoreAgg([score_list])	Score function aggregates of multiple score terms.
score.ScoreTweezers([force, ...])	Score function for tweezers experiments.

2.5.1 helixmc.score.ScoreBase

class helixmc.score.**ScoreBase**
Base class for scoring fucntion, for inheritence only.

__call__(*pose*)

__init__()

2.5.2 helixmc.score.ScoreExt

class `helixmc.score.`**`ScoreExt`**(*force*)

Score function for force-extension along Z-axis.

Parameters **force** : float

Applied z-direction force to the helix, in pN.

Attributes

force	(float) See Parameters section above.

`__call__`(*pose*)

Score the input pose.

Parameters **pose** : HelixPose

Input pose for scoring.

Returns **score** : float

Score of the pose.

`__init__`(*force*)

2.5.3 helixmc.score.ScoreTorsionTrap

class `helixmc.score.`**`ScoreTorsionTrap`**(*stiffness*, *target_link*)

Score function for torsional trap.

Parameters **stiffness** : float

The stiffness of the torsional trap, in pN.Å.

target_link : float

Center of the harmonic torsional trap (link), in radians.

Attributes

stiffness	(float)
target_link	(float) See Parameters section above.

`__call__`(*pose*)

Score the input pose.

Parameters **pose** : HelixPose

Input pose for scoring.

Returns **score** : float

Score of the pose.

`__init__`(*stiffness*, *target_link*)

2.5.4 helixmc.score.ScoreXyTrap

class `helixmc.score.`**`ScoreXyTrap`**(*stiffness*)

Score function for xy trap.

Parameters **stiffness** : float

The stiffness of the xy trap, in pN/Å.

Attributes

stiffness	(float) See Parameters section above.

`__call__`(*pose*)

Score the input pose.

Parameters **pose** : HelixPose

Input pose for scoring.

Returns **score** : float

Score of the pose.

`__init__`(*stiffness*)

2.5.5 helixmc.score.ScoreAgg

class `helixmc.score.`**`ScoreAgg`**(*score_list=[]*)

Score function aggregates of multiple score terms.

Parameters **score_list** : list, optional

List of score terms (subclass of ScoreBase) in this score.

Attributes

score_list	(list) See Parameters section above.
is_empty	(bool) If the score_list is empty.

`__call__`(*pose*)

Score the input pose.

Parameters **pose** : HelixPose

Input pose for scoring.

Returns **score** : float

Score of the pose.

`__init__`(*score_list=[]*)

`append`(*score_term*)

Append new score term.

Parameters **score_term** : subclass of ScoreBase

Score term to be appended.

clear()

Clear the score_list.

2.5.6 helixmc.score.ScoreTweezers

class helixmc.score.**ScoreTweezers**(*force=0*, *torsional_stiffness=0*, *target_link=None*, *xy_stiffness=0*)

Score function for tweezers experiments.

Parameters force : float, optional

Applied z-stretching force, in pN.

torsional_stiffness : float, optional

Stiffness of the torsional trap, in pN.Å.

target_link : float, optional

Center of the torsional trap (link), in radians.

xy_stiffness : float, optional

Stiffness of xy trap, in pN/Å.

__init__(*force=0*, *torsional_stiffness=0*, *target_link=None*, *xy_stiffness=0*)

2.6 Utility Functions

2.6.1 Rotation Matrices

`util.Rz`(theta)	Return z-rotation matrices with rotational angle theta.
`util.Rx`(theta)	Return x-rotation matrices with rotational angle theta.
`util.Ry`(theta)	Return y-rotation matrices with rotational angle theta.
`util.R_axis`(theta, axis)	Return rotation matrices with rotational angle theta along an arbitary rotation axis.

helixmc.util.Rz

helixmc.util.**Rz**(*theta*)

Return z-rotation matrices with rotational angle theta.

Parameters theta : array-like

Rotation angles of the matrix in radians.

Returns rot_matrix : ndarray

Corresponding z-rotation martrices for each input angle, align with the first index.

See also:

Ry Return y-rotation matrices with rotational angle theta.

Rx Return x-rotation matrices with rotational angle theta.

R_axis Return rotation matrices with rotational angle theta

`along`

helixmc.util.Rx

`helixmc.util.`**`Rx`**(*theta*)

Return x-rotation matrices with rotational angle theta.

Please refer to the documentation for *Rz* for further details.

See also:

`Rz`, *`Ry`*, *`R_axis`*

helixmc.util.Ry

`helixmc.util.`**`Ry`**(*theta*)

Return y-rotation matrices with rotational angle theta.

Please refer to the documentation for *Rz* for further details.

See also:

`Rz`, *`Rx`*, *`R_axis`*

helixmc.util.R_axis

`helixmc.util.`**`R_axis`**(*theta*, *axis*)

Return rotation matrices with rotational angle theta along an arbitary rotation axis.

Parameters **theta** : array-like

Rotation angles of the matrix in radians.

axis : ndarray, shape (N,3)

Rotational axis for the rotation being performed, align with first index (i.e. axis[3] is the rotational axis for angle theta[3]).

Returns **rot_matrix** : ndarray

Corresponding rotation martrices for each input angle, align with the first index.

See also:

`Rz` Return z-rotation matrices with rotational angle theta.

`Ry` Return y-rotation matrices with rotational angle theta.

`Rx` Return x-rotation matrices with rotational angle theta.

2.6.2 Twist and Writhe

`util.ribbon_twist`(dr, rb_vec[, ...])	Compute the ribbon-twist (supercoiling twist) of a helix.
`util.writhe_exact`(dr)	Compute the writhe of the helix using the exact Gauss double integral.
`util.writhe_fuller`(dr[, return_val_only])	Compute the writhe using the Fuller's approximated single integral.

helixmc.util.ribbon_twist

`helixmc.util.`**`ribbon_twist`**(*dr*, *rb_vec*, *return_val_only=True*, *twist_center=0.0*)

Compute the ribbon-twist (supercoiling twist) of a helix.

Parameters **dr** : ndarray, shape (N,3)

Input delta_r vectors (dr[i] = r[i+1] - r[i]).

rb_vec : ndarray, shape (N+1,3)

The ribbon vectors for the helix, equals to `frame[:,:,1]` in the current setting. Must be normalized vectors.

return_val_only : bool, optional

If True, return one float value for the overall twist of the system. Otherwise return the individual twists for each bp-step. Default set to True.

twist_center : float, optional

Fold the step twists to (center - pi, center + pi], default to 0.

Returns **twist** : float or ndarray of shape (N)

The overall twist or the individual twists of the helix. See the 'return_val_only' parameter above.

Raises **ValueError**

If dr.shape[0] != rb_vec.shape[0] + 1.

See also:

`writhe_fuller` Compute the writhe of the helix using the Fuller approximated single integral.

`writhe_exact` Compute the writhe of the helix using the exact Gauss double integral.

helixmc.util.writhe_exact

`helixmc.util.`**`writhe_exact`**`(`*dr*`)`

Compute the writhe of the helix using the exact Gauss double integral. O(N^2) complexity.

Parameters **dr** : ndarray, shape (N,3)

Input delta_r vectors (dr[i] = r[i+1] - r[i]).

Returns **writhe** : float

The overall writhe of the helix.

See also:

`writhe_fuller` Compute the writhe of the helix using the Fuller approximated single integral.

`ribbon_twist` Compute the ribbon-twist of a helix.

Notes

Details of the evaluation scheme is dicussed in reference [1] [2].

[1] Rossetto V, Maggs AC (2003) Writhing geometry of open DNA. J. Chem. Phys. 118: 9864-9874.

[2] Klenin K, Langowski J (2000) Computation of writhe in modeling of supercoiled DNA. Biopolymers 54: 307-317.

References

helixmc.util.writhe_fuller

helixmc.util.**writhe_fuller**(*dr*, *return_val_only=True*)

Compute the writhe using the Fuller's approximated single integral. Only guarantee to be correct modulo 4 pi. O(N) complexity.

Parameters **dr** : ndarray, shape = (N,3)

Input delta_r vectors (dr[i] = r[i+1] - r[i]).

return_val_only : bool, optional

If True, return one float value for the overall writhe of the system. Otherwise return the individual writhe contribution for each bp-step. Default set to True.

Returns **writhe** : float or ndarray of shape(N)

The overall writhe of the helix or the individual writhe contribution for each bp-step. See the 'return_val_only' parameter above.

See also:

writhe_exact Compute the writhe of the helix using the exact Gauss double integral.

ribbon_twist Compute the ribbon-twist of a helix.

Notes

See *writhe_exact* for references on details of the evaluation scheme.

2.6.3 Useful Conversions

util.params2coords(params)	Convert base-pair step parameters to bp-center coordinates and rotation matrix.
util.coords2params(o2, R2)	Convert bp-center coordinates and rotation matrix to base-pair step parameters.
util.dr2coords(dr)	Convert delta-r vectors to coordinates.
util.coords2dr(coord)	Convert xyz coordinates of axis curve to delta-r vectors.
util.params2data(params[, frame0])	Convert step parameters to delta-r vectors and frames.
util.data2params(dr, frames)	Convert delta-r vectors and frames to step parameters.
util.params_join(params)	Convert consequtive bp-params to the params between 1st and last bp.
util.frames2params(o1, o2, f1, f2)	Convert bp coordinates and frames to step parameters.
util.frames2params_3dna(o1, o2, f1, f2)	Convert bp coordinates and frames in 3DNA format to step parameters.

helixmc.util.params2coords

helixmc.util.**params2coords**(*params*)

Convert base-pair step parameters to bp-center coordinates and rotation matrix.

Parameters **params** : ndarray, shape (N,6)

Input base-pair step parameters. Distances in unit of Å, angles in unit of radians. Order = [Shift, Slide, Rise, Tilt, Roll, Twist]

Returns **o2** : ndarray, shape (N,3)

Coordiantes for the bp-centers of the 2nd base-pair.

R2 : ndarray, shape (N,3,3)

Rotational matrix for the transformation.

See also:

`coords2params` Convert bp-center coordinates and rotation matrix to base-pair step parameters.

helixmc.util.coords2params

`helixmc.util.`**`coords2params`**(*o2*, *R2*)

Convert bp-center coordinates and rotation matrix to base-pair step parameters. The frame of bp1 is used as reference (o1 = [0 0 0] and R1 = np.eye(3))

Parameters **o2** : ndarray, shape (N,3)

Coordiantes for the bp-centers of the 2nd base-pair.

R2 : ndarray, shape (N,3,3)

Rotational matrix for the transformation.

Returns Base-pair step parameters.

Distance in unit of Å, angle in unit of radians. Order = [Shift, Slide, Rise, Tilt, Roll, Twist]

See also:

`params2coords` Convert base-pair step parameters to bp-center coordinates and rotation matrix.

helixmc.util.dr2coords

`helixmc.util.`**`dr2coords`**(*dr*)

Convert delta-r vectors to coordinates.

Parameters **dr** : ndarray, shape (N,3)

Input delta_r vectors (dr[i] = r[i+1] - r[i]).

Returns **coord** : ndarray, shape (N+1,3)

Coordinates for the base-pair-centers (r[i]).

See also:

`coords2dr` Convert xyz coordinates of axis curve to delta-r vectors.

helixmc.util.coords2dr

`helixmc.util.`**`coords2dr`**(*coord*)

Convert xyz coordinates of axis curve to delta-r vectors.

Parameters **coord** : ndarray, shape (N,3)

The axis curve xyz coordinates in unit of Å (r vectors).

Returns **dr** : ndarray, shape (N,3)

Delta-r vectors (dr[i] = r[i+1] - r[i]).

See also:

`dr2coords` Convert delta-r vectors to coordinates.

helixmc.util.params2data

`helixmc.util.`**`params2data`**(*params*, *frame0=None*)

Convert step parameters to delta-r vectors and frames.

Parameters **params** : ndarray, shape (N,6)

Input base-pair step parameters. Order = [Shift, Slide, Rise, Tilt, Roll, Twist] Distance in unit of Å, angle in unit of radians.

frame0 : ndarray, shape (3,3), optional

The frame for the 1st base-pair. Default set to np.eye(3).

Returns **dr** : ndarray, shape (N,3)

Delta-r vectors (dr[i] = r[i+1] - r[i]).

frames : ndarray, shape (N+1,3,3)

Frame of each base-pair.

See also:

`data2params` Convert delta-r vectors and frames to step parameters.

helixmc.util.data2params

`helixmc.util.`**`data2params`**(*dr*, *frames*)

Convert delta-r vectors and frames to step parameters.

Parameters **dr** : ndarray, shape (N,3)

Delta-r vectors (dr[i] = r[i+1] - r[i]).

frames : ndarray, shape (N+1,3,3)

Frame of each base-pair.

Returns **params** : ndarray, shape (N,6)

Base-pair step parameters. Distance in unit of Å, angle in unit of radians. Order = [Shift, Slide, Rise, Tilt, Roll, Twist]

See also:

`params2data` Convert step parameters to delta-r vectors and frames.

helixmc.util.params_join

`helixmc.util.`**`params_join`**(*params*)

Convert consequtive bp-params to the params between 1st and last bp.

Parameters **params** : ndarray, shape (N,6)

Input base-pair step parameters. Distances in unit of Å, angles in unit of radians. Order = [Shift, Slide, Rise, Tilt, Roll, Twist]

Returns **params** : ndarray, shape (6)

Base-pair step parameters between 1st and last.

helixmc.util.frames2params

`helixmc.util.`**`frames2params`**`(`*o1*, *o2*, *f1*, *f2*`)`

Convert bp coordinates and frames to step parameters.

Parameters **o1** : ndarray, shape (N,3)

Origins for the bp-centers of the 1st base-pair.

o2 : ndarray, shape (N,3)

Origins for the bp-centers of the 2nd base-pair.

f1 : ndarray, shape (N,3,3)

Frames for the 1st base-pair.

f2 : ndarray, shape (N,3,3)

Frames for the 2nd base-pair.

Returns **params** : ndarray, shape (N,6)

Base-pair step parameters. Distance in unit of Å, angle in unit of radians. Order = [Shift, Slide, Rise, Tilt, Roll, Twist]

See also:

`frames2params_3dna` Convert bp coordinates and frames in 3DNA format to step parameters.

helixmc.util.frames2params_3dna

`helixmc.util.`**`frames2params_3dna`**`(`*o1*, *o2*, *f1*, *f2*`)`

Convert bp coordinates and frames in 3DNA format to step parameters. Note that the 3DNA base-pair frames differ from HelixMC by a transpose operation.

See also:

`frames2params` Convert bp coordinates and frames to step parameters.

2.6.4 Other Functions

`util.unitarize`(R)	Enforce unitarity of the input matrix using Gram-Schmidt process.
`util.circmean`(arr[, axis])	Circular mean of angles.
`util.MC_acpt_rej`(score_old, score_new[, kT])	Decide whether to accept a Monte-Carlo move.
`util.read_seq_from_fasta`(fasta)	Read the sequence from a fasta file.
`util.locate_data_file`(data_file)	Search for the input data_file.

helixmc.util.unitarize

`helixmc.util.`**`unitarize`**(*R*)

Enforce unitarity of the input matrix using Gram-Schmidt process. This function modifies the input matrix inplace.

Parameters R : ndarray, shape (N,3,3)

(List of) input matrix to be unitarized.

Returns R : ndarray, shape (N,3,3)

Unitarized input matrix.

helixmc.util.circmean

`helixmc.util.`**`circmean`**(*arr*, *axis=None*)

Circular mean of angles. Results are in [-pi, pi]. Note that scipy.stats has a similar function, but we re-implement it because the scipy one depends on its version.

Parameters arr : ndarray

Input numpy array.

axis : int, optional

Axis on which the mean is computed.

Returns mean : ndarray

The circular mean.

helixmc.util.MC_acpt_rej

`helixmc.util.`**`MC_acpt_rej`**(*score_old*, *score_new*, *kT=41.164043971999995*)

Decide whether to accept a Monte-Carlo move.

Parameters score_old : float

score before the proposed update.

score_new : float

score after the proposed update.

kT : float

Temperature times Boltzmann constant, in pN.Å.

Returns out : bool

True for accept, False for reject.

helixmc.util.read_seq_from_fasta

`helixmc.util.`**`read_seq_from_fasta`**(*fasta*)

Read the sequence from a fasta file.

Parameters fasta : str

Name of the fasta file.

Returns **seq** : str

Sequence stored in fasta.

Raises **ValueError**

If the fasta file does not exist.

helixmc.util.locate_data_file

`helixmc.util.`**`locate_data_file`**(*data_file*)

Search for the input data_file. Will look in either current folder or HelixMC data/.

Parameters **data_file** : string

Input numpy array.

Returns **data_file_working** : string

Actual path of the found data file.

Raises **ValueError** : if the input file does not exist.

2.7 Useful Fitting Functions

`fitfxn.wlc_odijk`(F, A, L, S[, kT])	Worm-like chain fitting formula described in Odijk 1995 paper.
`fitfxn.wlc_bouchiat`(A, L, z[, kT])	Worm-like chain fitting formula described in Bouchiat et al.
`fitfxn.wlc_bouchiat_impl`(A, L, S, z, F[, kT])	Implicit worm-like chain formula described in Bouchiat et al.
`fitfxn.f_wlc_bouchiat_impl`(A, L, S, z[, kT])	Approximately solve the force from implicit wlc model by grid search.
`fitfxn.moroz_3rd`(A, C, F[, kT])	3rd order Moroz-Nelson function for effective torsional persistence.
`fitfxn.moroz_1st`(A, C, F[, kT])	1st order Moroz-Nelson function for effective torsional persistence.

2.7.1 helixmc.fitfxn.wlc_odijk

`helixmc.fitfxn.`**`wlc_odijk`**(*F*, *A*, *L*, *S*, *kT=41.164043971999995*)

Worm-like chain fitting formula described in Odijk 1995 paper. It is an extensive WLC model.

Parameters **F** : float or 1D ndarray

Z-direction stretching force, in pN.

A : float

Bending persistence length, in Å.

L : float

Contour length, in Å.

S : float

Stretch modulus, in pN.

kT : float, optional

Temperature times Boltzmann constant, in pN.Å. Default is 298.15 K.

Returns **z** : float or 1D ndarray

Average Z-extension of the helix, in Å.

Notes

The fitting function [1] is

$$z = L\left(1 - \frac{1}{2}\sqrt{\frac{kT}{FA}} + \frac{F}{S}\right)$$

References

2.7.2 helixmc.fitfxn.wlc_bouchiat

`helixmc.fitfxn.`**`wlc_bouchiat`**(*A*, *L*, *z*, *kT=41.164043971999995*)

Worm-like chain fitting formula described in Bouchiat et al. 1999 paper. Assumes the chain has constant length (inextensive model). For fitting force-extension curve in low-to-medium force regime (< 10 pN).

Parameters **A** : float

Bending persistence length, in Å.

L : float

Contour length, in Å.

z : float or 1D ndarray

Average Z-extension of the helix, in Å.

kT : float, optional

Temperature times Boltzmann constant, in pN.Å. Default is 298.15 K.

Returns **F** : float or 1D ndarray

Z-direction stretching force, in pN.

See also:

`wlc_bouchiat_impl` Implict worm-like chain fitting formula described in Bouchiat et al.

`f_wlc_bouchiat_impl` Approximately solve the force from implicit wlc model by grid search.

Notes

The fitting function [1] is

$$F = \frac{kT}{A}\left[\frac{1}{4(1-z/L)^2} - \frac{1}{4} + \frac{z}{L} + \sum_{i=2}^{i\leq 7}\alpha_i\left(\frac{z}{L}\right)^i\right]$$

Where α = [-0.5164228, -2.737418, 16.07497, -38.87607, 39.49944, -14.17718] for i = 2~7.

[1] Odijk, T. (1995) Stiff Chains and Filaments under Tension. Macromolecules 28: 7016-7018.

[1] Bouchiat C, Wang MD, Allemand J, Strick T, Block SM, et al. (1999) Estimating the persistence length of a worm-like chain molecule from force-extension measurements. Biophys. J. 76: 409-413.

References

2.7.3 helixmc.fitfxn.wlc_bouchiat_impl

helixmc.fitfxn.**wlc_bouchiat_impl**(*A*, *L*, *S*, *z*, *F*, *kT=41.1640439719999995*)

Implicit worm-like chain formula described in Bouchiat et al. 1999 paper. Comparison is done in log10(F) space instead (more robust in practice). An extensive rod factor is added for fitting force-extension curve at high-force.

Parameters **A** : float

Bending persistence length, in Å.

L : float

Contour length, in Å.

S : float

Stretch modulus, in pN.

z : float or 1D ndarray

Average Z-extension of the helix, in Å.

F : float or 1D ndarray

Z-direction stretching force, in pN.

kT : float, optional

Temperature times Boltzmann constant, in pN.Å. Default is 298.15 K.

Returns **zero** : float or 1D ndarray

Difference value, should approach zero at perfect fit.

See also:

wlc_bouchiat Worm-like chain fitting formula described in Bouchiat et al.

f_wlc_bouchiat_impl Approximately solve the force from implicit wlc model by grid search.

Notes

The fitting function (see *wlc_bouchiat* for reference) is

$$\log_{10}\left[\frac{kT}{A}\left(\frac{1}{4(1-l)^2}-\frac{1}{4}+l+\sum_{i=2}^{i\leq 7}\alpha_i l^i\right)\right]-\log_{10}(F)=0$$

With $l = z/L - F/S$

2.7.4 helixmc.fitfxn.f_wlc_bouchiat_impl

helixmc.fitfxn.**f_wlc_bouchiat_impl**(*A*, *L*, *S*, *z*, *kT=41.1640439719999995*)

Approximately solve the force from implicit wlc model by grid search.

Parameters **A** : float

Bending persistence length, in Å.

L : float

Contour length, in Å.

S : float

Stretch modulus, in pN.

z : float or 1D ndarray

Average Z-extension of the helix, in Å.

kT : float, optional

Temperature times Boltzmann constant, in pN.Å. Default is 298.15 K.

Returns **F** : float or 1D ndarray

Z-direction stretching force, in pN.

See also:

`wlc_bouchiat` Worm-like chain fitting formula described in Bouchiat et al.

`wlc_bouchiat_impl` Implict worm-like chain fitting formula described in Bouchiat et al.

2.7.5 helixmc.fitfxn.moroz_3rd

`helixmc.fitfxn.moroz_3rd(A, C, F, kT=41.164043971999995)`

3rd order Moroz-Nelson function for effective torsional persistence.

Parameters **A** : float

Bending persistence length, in Å.

C : float

Torsional persistence length, in Å.

F : float or 1D ndarray

Z-direction stretching force, in pN.

kT : float, optional

Temperature times Boltzmann constant, in pN.Å. Default is 298.15 K.

Returns **Ceff** : float or 1D ndarray

Effective torsional persistence length in Å. Ceff = L / Var(Lk).

See also:

`moroz_1st` 1st order Moroz-Nelson function for effective torsional persistence.

Notes

The fitting function, orginally proposed in reference [1] [2], and summarized in [3], is

$$C_{eff} = C\left(1 - \frac{M}{4K_0} + \frac{M^2 - 2M}{16{K_0}^2} - \frac{4M^3 - 16M^2 + 21M}{256{K_0}^3}\right)$$

With

$$M = \frac{C}{A}, \quad K_0 = \sqrt{\frac{AF}{kT}}$$

References

2.7.6 helixmc.fitfxn.moroz_1st

`helixmc.fitfxn.`**`moroz_1st`**(*A*, *C*, *F*, *kT=41.164043971999995*)

1st order Moroz-Nelson function for effective torsional persistence. See *moroz_3rd* for detailed explanation.

See also:

`moroz_3rd` 3rd order Moroz-Nelson function for effective torsional persistence.

Notes

The fitting function is

$$C_{eff} = C\left(1 - \frac{M}{4K_0}\right)$$

With

$$M = \frac{C}{A}, \quad K_0 = \sqrt{\frac{AF}{kT}}$$

[1] Moroz JD, Nelson P (1997) Torsional directed walks, entropic elasticity, and DNA twist stiffness. PNAS 94: 14418-14422.

[2] Moroz JD, Nelson P (1998) Entropic Elasticity of Twist-Storing Polymers. Macromolecules 31: 6333-6347.

[3] Lipfert J, Wiggin M, Kerssemakers JWJ, Pedaci F, Dekker NH (2011) Freely orbiting magnetic tweezers to directly monitor changes in the twist of nucleic acids. Nat. Comm. 2: 439.

CHAPTER 3

About us

The HelixMC project is authored by Fang-Chieh Chou in 2013, under the supervison of Dr. Rhiju Das, at the Biochemistry Department of Stanford Unviersity.

3.1 Contacts

- Fang-Chieh Chou <fcchou@stanford.edu>
- Rhiju Das <rhiju@stanford.edu>

3.2 Links

- Source code: https://github.com/fcchou/HelixMC
- HTML documentation: http://helixmc.readthedocs.org/
- Das Lab @ Stanford: http://daslab.stanford.edu

3.3 Citing HelixMC

If you use HelixMC in scientific publications, we would appreciate citations to the following paper:

- Chou F-C, Lipfert J, Das R (2014) Blind Predictions of DNA and RNA Tweezers Experiments with Force and Torque. PLoS Comput Biol 10(8): e1003756. Link

3.4 Funding

Fang-Chieh Chou is supported by an HHMI International Student Research Fellowship, and a Stanford Bio-X Graduate Student Research Fellowship.

Rhiju Das is supported by the NIH (R21-GM102716) and a Burroughs-Wellcome Career Award at Scientific Interface.

Bibliography

[R1] Fuller FB (1978) Decomposition of the linking number of a closed ribbon: A problem from molecular biology. PNAS 75: 3557-3561.

[R2] Lipfert J, Kerssemakers JWJ, Jager T, Dekker NH (2010) Magnetic torque tweezers: measuring torsional stiffness in DNA and RecA-DNA filaments. Nature Methods 7: 977–980.

[R3] Olson WK, Colasanti AV, Czapla L, Zheng G (2008) Insights into the Sequence-Dependent Macromolecular Properties of DNA from Base-Pair Level Modeling. In: Voth GA, editor. Coarse-Graining of Condensed Phase and Biomolecular Systems: CRC Press. pp. 205-223.

Symbols

__call__() (helixmc.random_step.RandomStepAgg method), 13
__call__() (helixmc.random_step.RandomStepBase method), 11
__call__() (helixmc.random_step.RandomStepSimple method), 12
__call__() (helixmc.score.ScoreAgg method), 16
__call__() (helixmc.score.ScoreBase method), 14
__call__() (helixmc.score.ScoreExt method), 15
__call__() (helixmc.score.ScoreTorsionTrap method), 15
__call__() (helixmc.score.ScoreXyTrap method), 16
__init__() (helixmc.pose.HelixPose method), 8
__init__() (helixmc.random_step.RandomStepAgg method), 13
__init__() (helixmc.random_step.RandomStepBase method), 11
__init__() (helixmc.random_step.RandomStepSimple method), 12
__init__() (helixmc.score.ScoreAgg method), 16
__init__() (helixmc.score.ScoreBase method), 14
__init__() (helixmc.score.ScoreExt method), 15
__init__() (helixmc.score.ScoreTorsionTrap method), 15
__init__() (helixmc.score.ScoreTweezers method), 17
__init__() (helixmc.score.ScoreXyTrap method), 16

A

accept_update() (helixmc.pose.HelixPose method), 8
append() (helixmc.score.ScoreAgg method), 16
append_random_step() (helixmc.random_step.RandomStepAgg method), 13

C

circmean() (in module helixmc.util), 24
clear() (helixmc.score.ScoreAgg method), 16
clear_all() (helixmc.random_step.RandomStepAgg method), 13
constant_seed() (in module helixmc), 8
coords2dr() (in module helixmc.util), 21
coords2params() (in module helixmc.util), 21
copy() (helixmc.pose.HelixPose method), 9

D

data2params() (in module helixmc.util), 22
dr2coords() (in module helixmc.util), 21

F

f_wlc_bouchiat_impl() (in module helixmc.fitfxn), 27
frames2params() (in module helixmc.util), 23
frames2params_3dna() (in module helixmc.util), 23
from_file() (helixmc.pose.HelixPose class method), 9

G

get_rand_step() (helixmc.random_step.RandomStepAgg method), 14
guess_twist_center() (helixmc.pose.HelixPose method), 9

H

HelixPose (class in helixmc.pose), 8

L

load_from_file() (helixmc.random_step.RandomStepAgg method), 14
load_gaussian_params() (helixmc.random_step.RandomStepSimple class method), 12
load_params() (helixmc.random_step.RandomStepSimple class method), 12
locate_data_file() (in module helixmc.util), 25

M

MC_acpt_rej() (in module helixmc.util), 24
moroz_1st() (in module helixmc.fitfxn), 29
moroz_3rd() (in module helixmc.fitfxn), 28

N

name2rand_idx() (helixmc.random_step.RandomStepAgg method), 14

P

params2coords() (in module helixmc.util), 20
params2data() (in module helixmc.util), 22
params_join() (in module helixmc.util), 22
plot_centerline() (helixmc.pose.HelixPose method), 9
plot_helix() (helixmc.pose.HelixPose method), 9

R

R_axis() (in module helixmc.util), 18
RandomStepAgg (class in helixmc.random_step), 12
RandomStepBase (class in helixmc.random_step), 11
RandomStepSimple (class in helixmc.random_step), 11
read_seq_from_fasta() (in module helixmc.util), 24
reject_update() (helixmc.pose.HelixPose method), 9
ribbon_twist() (in module helixmc.util), 18
Rx() (in module helixmc.util), 18
Ry() (in module helixmc.util), 18
Rz() (in module helixmc.util), 17

S

ScoreAgg (class in helixmc.score), 16
ScoreBase (class in helixmc.score), 14
ScoreExt (class in helixmc.score), 15
ScoreTorsionTrap (class in helixmc.score), 15
ScoreTweezers (class in helixmc.score), 17
ScoreXyTrap (class in helixmc.score), 16
set_params() (helixmc.pose.HelixPose method), 9

U

unitarize() (in module helixmc.util), 24
update() (helixmc.pose.HelixPose method), 10
update_trial() (helixmc.pose.HelixPose method), 10

W

wlc_bouchiat() (in module helixmc.fitfxn), 26
wlc_bouchiat_impl() (in module helixmc.fitfxn), 27
wlc_odijk() (in module helixmc.fitfxn), 25
write2disk() (helixmc.pose.HelixPose method), 10
writhe_exact() (in module helixmc.util), 19
writhe_fuller() (in module helixmc.util), 20