unrolr Documentation

Release 0.4.0.6

Jerome Eberhardt

Jul 29, 2021

User guide / tutorial

CHAPTER 1

Abstract

Molecular dynamics (MD) simulations are widely used to explore the conformational space of biological macromolecules. Advances in hardware, as well as in methods, make the generation of large and complex MD datasets much more common. Although different clustering and dimensionality reduction methods have been applied to MD simulations, there remains a need for improved strategies that handle nonlinear data and/or can be applied to very large datasets. We present an original implementation of the pivot-based version of the stochastic proximity embedding method aimed at large MD datasets using the dihedral distance as a metric. The advantages of the algorithm in terms of data storage and computational efficiency are presented, as well as the implementation realized.

Example

```
from __future__ import print_function

from unrolr import Unrolr
from unrolr.feature_extraction import Dihedral
from unrolr.utils import save_dataset


top_file = 'examples/inputs/villin.psf'
trj_file = 'examples/inputs/villin.dcd'

# Extract all calpha dihedral angles from trajectory and store them into a HDF5 file␣
↪(start/stop/step are optionals)
d = Dihedral(top_file, trj_file, selection='all', dihedral_type='calpha', start=0,␣
↪stop=None, step=1).run()
X = d.result
save_dataset('dihedral_angles.h5', "dihedral_angles", X)

# Fit X using Unrolr (pSPE + dihedral distance) and save the embedding into a csv file
U = Unrolr(r_neighbor=0.27, n_iter=50000, verbose=1)
U.fit_transform(X)
U.save(fname='embedding.csv')

print('%4.2f %4.2f' % (U.stress, U.correlation))
```

Todo list

- [ ] Compare SPE performance with UMAP
- [x] Compatibility with python 3
- [x] Compatibility with the latest version of MDAnalysis (==0.17)
- [ ] Unit tests
- [x] Accessible directly from pip
- [ ] Improve OpenCL performance (global/local memory)

Citation

Eberhardt, J., Stote, R. H., & Dejaegere, A. (2018). Unrolr: Structural analysis of protein conformations using stochastic proximity embedding. Journal of Computational Chemistry, 39(30), 2551-2557. https://doi.org/10.1002/jcc.25599

License

MIT

1.1 Installation

Conformational analysis of MD trajectories based on (pivot-based) Stochastic Proximity Embedding using dihedral distance as a metric.

1.1.1 Prerequisites

You need, at a minimum (requirements.txt):

- Python 2.7 or python 3
- NumPy
- H5py
- Pandas
- Matplotlib
- PyOpenCL
- MDAnalysis (>=0.17)

1.1.2 Installation on UNIX (Debian/Ubuntu)

I highly recommand you to install the Anaconda distribution (https://www.continuum.io/downloads) if you want a clean python environnment with nearly all the prerequisites already installed (NumPy, H5py, Pandas, Matplotlib).

1 . First, you have to install OpenCL:

- MacOS: Good news, you don't have to install OpenCL, it works out-of-the-box. (Update: bad news, OpenCL is now depreciated in macOS 10.14. Thanks Apple.)
- AMD: You have to install the AMDGPU graphics stack.
- Nvidia: You have to install the CUDA toolkit.
- Intel: And of course it's working also on CPU just by installing this runtime software package. Alternatively, the CPU-based OpenCL driver can be also installed through the package `pocl` (http://portablecl.org/) with the conda package manager.

For any other informations, the official installation guide of PyOpenCL is available here.

2 . As a final step, installation from PyPi server

```
pip install unrolr
```

Or from the source

```
# Get the package
wget https://github.com/jeeberhardt/unrolr/archive/master.zip
unzip unrolr-master.zip
rm unrolr-master.zip
cd unrolr-master

# Install the package
python setup.py install
```

And if somehow pip is having problem to install all the dependencies,

```
conda config --append channels conda-forge
conda install pyopencl mdanalysis

# Try again
python setup.py install
```

1.1.3 OpenCL context

Before running Unrolr, you need to define the OpenCL context. And it is a good way to see if everything is working correctly.

```
python -c 'import pyopencl as cl; cl.create_some_context()'
```

Here in my example, I have the choice between 3 differents computing device (2 graphic cards and one CPU).

```
Choose platform:
[0] <pyopencl.Platform 'AMD Accelerated Parallel Processing' at 0x7f97e96a8430>
Choice [0]:0
Choose device(s):
[0] <pyopencl.Device 'Tahiti' on 'AMD Accelerated Parallel Processing' at 0x1e18a30>
[1] <pyopencl.Device 'Tahiti' on 'AMD Accelerated Parallel Processing' at 0x254a110>
[2] <pyopencl.Device 'Intel(R) Core(TM) i7-3820 CPU @ 3.60GHz' on 'AMD Accelerated␣
↪Parallel Processing' at 0x21d0300>
Choice, comma-separated [0]:1
Set the environment variable PYOPENCL_CTX='0:1' to avoid being asked again.
```

Now you can set the environment variable.

```
export PYOPENCL_CTX='0:1'
```

1.2 Tutorial

1.2.1 Import unrolr module

First load of the functions needed from the Unrolr package

```
import numpy as np

from unrolr import Unrolr
from unrolr.feature_extraction import Dihedral
from unrolr.sampling import neighborhood_radius_sampler, optimization_cycle_sampler
from unrolr.plotting import plot_sampling
from unrolr.utils import save_dataset
```

1.2.2 Extraction of dihedral angles

The first step will be the extraction of all the pseudo C-alpha dihedral angles (or all backbone dihedral angles) using the topology file (in psf format) and 200 ns aMD trajectory (with only 10000 frames) of the villin headpiece (in dcd format).

```
top_file = 'inputs/villin.psf'
trj_file = 'inputs/villin.dcd'

d = Dihedral(top_file, trj_file, selection='all', dihedral_type='calpha',
             start=0, stop=None, step=1).run()
X = d.result
save_dataset('dihedral_angles.h5', "dihedral_angles", X)
```

As output, you will have a HDF5 file, named `dihedral_angles.h5`, containing all the pseudo C-alpha dihedral angle (32 in total) from 10.000 frames of the villin headpiece. This HDF5 file can be opened and visualised easily using HDFView.

1.2.3 Search optimal pSPE parameters

Now the next big step will be the determination of the optimal neighborhood radius rc and optionally the optimal number of cycles needed to achieve a good convergence, generally between 10.000 and 50.000 cycles. However, concerning the optimal neighborhood radius rc there is no general rule, because it depends exclusively of the studied system. The choice of its value will influence significantly the representation in the low dimensional space (n = 2): **if rc is too small, only local distances will be faithfully represented in low dimension, and the final representation will appear as cloud of disconnected clusters. On the contrary, if rc is too large, we loose the magical power of rc and the method will revert to a linear dimensionality reduction method like multidimensional scaling.**

How to choose the optimal neighbourhood radius rc cutoff?

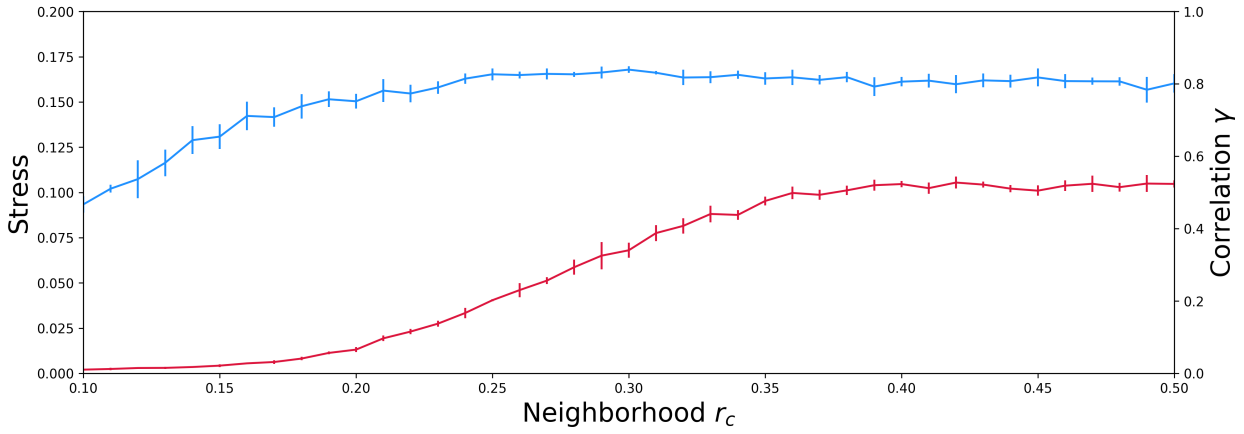
As the optimal value of rc depends of the studied system, we can quickly test multiple value, and choose one that will minimizes the stress and maximizes the correlation between the distances in high dimension space and 2D dimension space. For this, we will systematically test different values of rc from 0.01 to 0.5 by increments of 0.01. However, for the sake of effeciency, we won't use all the conformations (10.000 in our case), but just a reduced set with 5.000 conformations only. For this reduced set, 5 successive independent runs of pSPE are performed, with 5.000 steps of optimization cycles.

```
# We will sample neighorhood radius every 0.05
r_neighbors = np.linspace(0.1, 1.0, (1 / 0.05) - 1)

df = neighborhood_radius_sampler(X[::2,], r_neighbors, n_iter=5000, n_runs=5)
df.to_csv('r_neighbor_vs_stress_correlation.csv', index=False)
plot_sampling('r_neighbor_vs_stress_correlation.png', df, of='r_neighbor', show=False)
```

As output, you will find a file, named `r_neighbor_vs_stress-correlation.csv`, containing all the results, the stress and the correlation in function of the neighbourhood radius rc for each pSPE run, and the plot

corresponding, named `r_neighbor_vs_stress-correlation.png`.



The correlation between the actual and the projected distances increases as the neighbourhood radius rc is increased and converges to a plateau value of 0.80 (80%) for values of rc larger than 0.27. Further increase in rc does not improve the correlation but adversely affects the stress.

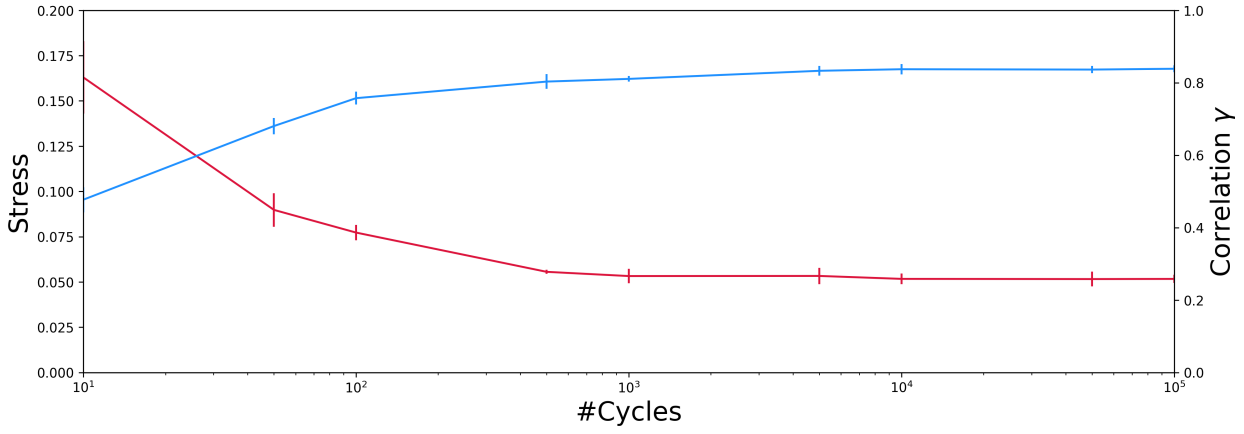
Optional: what is the minimal number of optimization cycles?

Generally, from my personal experience, the minimal number of optimization cycles needed is similar and independent of the nature and the size of the studied system, between 10.000 and 50.000 cycles. But still, we can test the influence of the number of pSPE optimization cycles on the correlation and the stress, while keeping the value of rc fixed at 0.27.

```
n_iters = [500, 1000, 5000, 10000, 50000, 100000]

df = optimization_cycle_sampler(X[::2,], n_iters, r_neighbor=0.27)
df.to_csv('n_iter_vs_stress_correlation.csv', index=False)
plot_sampling('n_iter_vs_stress_correlation.png', df, of='n_iter', show=False)
```

As output, you will find this time a file (named `n_iter_vs_stress-correlation.csv`) containing all the results, the stress and the correlation in function of the number of optimization cycles for each pSPE run, and the plot corresponding (named `n_iter_vs_stress-correlation.png`).



It can be seen that a minimum number of 10.000 cycles of optimization, at least, is needed to obtain converged values of the correlation and stress. Additional data (not shown here) shows that the size of the data set does not affect the convergence rate.

1.2.4 Fire off Unrolr!

As the final step, after determining the optimal neighbourhood radius rc cutoff, equal to 0.27 in this case, and the minimal number of optimization cycles, at least 10.000 cycles, the pSPE method can now be applied to the complete data set.

```
U = Unrolr(r_neighbor=0.27, n_iter=50000, verbose=1)
U.fit_transform(X)
U.save(fname='outputs/embedding.csv')

# Or you can add an extra column with frame ids (frames=(start, stop, skip))
# U.save(fname='outputs/embedding.csv', frames=np.arange(0, X.shape[0]))

print U.stress, U.correlation
```

The final pSPE optimization process takes approximately 13 seconds for 10.000 conformations with 32 pseudo C-alpha dihedral angles and 50.000 cycles on a single (and now old) AMD Radeon HD 7950 GPU. As output, you will find the final optimized configuration, named `embedding.csv`. Using the tool visualize, you can now explore easily the conformational space sampled during the MD simulation.

1.3 Unrolr

1.3.1 unrolr.Unrolr module

class unrolr.core.unrolr.**Unrolr**(*r_neighbor*, *metric='dihedral'*, *n_components=2*, *n_iter=10000*, *random_seed=None*, *init='random'*, *learning_rate=1.0*, *epsilon=0.0001*, *verbose=0*, *platform='OpenCL'*)

Bases: object

__init__(*r_neighbor*, *metric='dihedral'*, *n_components=2*, *n_iter=10000*, *random_seed=None*, *init='random'*, *learning_rate=1.0*, *epsilon=0.0001*, *verbose=0*, *platform='OpenCL'*)

Initialize Unrolr object.

Parameters

- **r_neighbor** (*float*) – neighbor radius cutoff
- **metric** (*str*) – distance metric (choices: dihedral or intramolecular) (default: dihedral)
- **n_component** (*int*) – number of component of the final embedding (default: 2)
- **n_iter** (*int*) – number of optimization iteration (default: 10000)
- **random_seed** (*int*) – random seed (default: None)
- **init** (*str*) – method to initialize the initial embedding (random or pca)(default: random)
- **learning_rate** (*float*) – learning rate, aka computational temperature (default: 1)
- **epsilon** (*float*) – convergence criteria when computing final stress and correlation (default: 1e-4)
- **verbose** (*int*) – turn on:off verbose (default: False)
- **platform** (*str*) – platform to use for spe (OpenCL or CPU) (default: OpenCL)

fit_transform(*r*)

Run the Unrolr (pSPE + didhedral distance) method.

Parameters **r** (*ndarray*) – n-dimensional dataset (rows: frame; columns: angle)

save (*fname='embedding.csv'*, *frames=None*)
Save all the data

Parameters

- **fname** (*str*) – pathname of the csv file containing the final embedding (default: embedding.csv)
- **frames** (*array-like*) – 1d-array containing frame numbers (Default: None)

unrolr.core.unrolr.**main**()
Main function, unrolr.py can be executed as a standalone script

Parameters

- **-f/--dihedral** (*filename*) – hdf5 file containing dihedral angles
- **-r/--rc** (*float*) – neighborhood radius cutoff (default: 1)
- **-n/-ndim** (*int*) – number of dimension of the final embedding (default: 2)
- **-c/--cycles** (*int*) – number of optimization iteration (default: 1000)
- **--start** (*int*) – index of the first frame to analyze (default: 1)
- **--stop** (*int*) – index of the last frame to analyze (default: -1)
- **--skip** (*int*) – number of frame to skip (default: 1)
- **-o/--output** (*filename*) – csv output file name (default: embedding.csv)
- **-s/--seed** – random seed (default: None)

Returns csv file containing the final embedding (default: embedding.csv)

Return type output (file)

1.3.2 unrolr.feature_extraction.dihedrals module

class unrolr.feature_extraction.dihedrals.**Dihedral**(*top_file*, *trj_files*, *selection='backbone'*, *dihedral_type='calpha'*, ***kwargs*)
Bases: MDAnalysis.analysis.base.AnalysisBase

__init__(*top_file*, *trj_files*, *selection='backbone'*, *dihedral_type='calpha'*, ***kwargs*)
Create Dihedral analysis object.

Parameters

- **top_file** (*str*) – filename of the topology file
- **trj_files** (*str or array-like*) – one or a list of trajectory files
- **selection** (*str*) – protein selection (default: backbone)
- **dihedral_type** (*str*) – type of dihedral angles to extract (choices: dihedral or calpha) (default: backbone)

unrolr.feature_extraction.dihedrals.**main**()
Main function, dihedral.py can be executed as a standalone script

Parameters

- **-p/--top** (*filename*) – topology file used for simulation (pdb, psf)

- **-t/--trj** (*filename*) – one or list of trajectory files
- **-s/--selection** (*str*) – protein selection
- **-d/--dihedral** (*str*) – type of dihedral angles to extract (choices: dihedral or calpha) (default: backbone)
- **-o/--output** (*filename*) – hdf5 output file name (default: dihedral_angles.h5)

Returns hdf5 file containing the dihedral angles (default: dihedral_angles.h5)

Return type output (file)

1.3.3 unrolr.feature_extraction.intramolecular_distances module

class unrolr.feature_extraction.intramolecular_distances.**IntramolecularDistance**(*top_file*, *trj_files*, *selection='backbone'*, ***kwargs*)

Bases: MDAnalysis.analysis.base.AnalysisBase

__init__(*top_file*, *trj_files*, *selection='backbone'*, ***kwargs*)

Create IntramolecularDistance analysis object.

Parameters

- **top_file** (*str*) – filename of the topology file
- **trj_files** (*str or array-like*) – one or a list of trajectory files
- **selection** (*str*) – protein selection (default: backbone)

unrolr.feature_extraction.intramolecular_distances.**main**()

Main function, intramolecular_distances.py can be executed as a standalone script

Parameters

- **-p/--top** (*filename*) – topology file used for simulation (pdb, psf)
- **-t/--trj** (*filename*) – one or list of trajectory files
- **-s/--selection** (*str*) – protein selection
- **-o/--output** (*filename*) – hdf5 output file name (default: intramolecular_distances.h5)

Returns hdf5 file containing the intramolecular distances (default: intramolecular_distances.h5)

Return type output (file)

1.3.4 unrolr.sampling.sampling module

unrolr.sampling.sampling.**neighborhood_radius_sampler**(*X*, *r_neighbors*, *metric='dihedral'*, *n_components=2*, *n_iter=5000*, *n_runs=5*, *init='random'*, *platform='OpenCL'*)

Sample different neighborhood radius rc and compute the stress and correlation.

Parameters

- **X** (*ndarray*) – n-dimensional ndarray (rows: frames; columns: features/angles)
- **r_neighbors** (*array-like*) – list of the neighborhood raidus cutoff to try
- **metric** (*str*) – metric to use to compute distance between conformations (dihedral or intramolecular) (default: dihedral)
- **n_components** (*int*) – number of dimension of the embedding
- **n_iter** (*int*) – number of optimization cycles
- **n_runs** (*int*) – number of repetitions, in order to calculate standard deviation
- **init** (*str*) – method to initialize the initial embedding (random or pca)(default: random)
- **platform** (*str*) – platform to use for spe (OpenCL or CPU) (default: OpenCL)

Returns Pandas DataFrame containing columns ["run", "r_neighbor", "n_iter", "stress", "correlation"]

Return type results (DataFrame)

unrolr.sampling.sampling.**optimization_cycle_sampler**(*X*, *n_iters*, *r_neighbor*, *metric='dihedral'*, *n_components=2*, *n_runs=5*, *init='random'*, *platform='OpenCL'*)

Sample different number of optimization cycle with a certain neighborhood radius rc and compute the stress and correlation.

Parameters

- **X** (*ndarray*) – n-dimensional ndarray (rows: frames; columns: features/angles)
- **n_iters** (*array-like*) – list of the iteration numbers to try
- **r_neighbor** (*float*) – neighborhood raidus cutoff
- **metric** (*str*) – metric to use to compute distance between conformations (dihedral or intramolecular) (default: dihedral)
- **n_components** (*int*) – number of dimension of the embedding
- **n_runs** (*int*) – number of repetitions, in order to calculate standard deviation
- **init** (*str*) – method to initialize the initial embedding (random or pca)(default: random)
- **platform** (*str*) – platform to use for spe (OpenCL or CPU) (default: OpenCL)

Returns Pandas DataFrame containing columns ["run", "r_neighbor", "n_iter", "stress", "correlation"]

Return type results (DataFrame)

1.3.5 unrolr.plotting.plot_embedding module

unrolr.plotting.plot_embedding.**plot_embedding**(*fname*, *embedding*, *label='Dihedral distance'*, *clim=None*, *bin_size=None*, *cmap='viridis'*, *show=True*)

Plot 2D histogram of the embedding. The color code refers to the number of conformations in each bin of the histogram.

Parameters

- **fname** (*str*) – filename of the embedding plot
- **embedding** (*ndarray*) – n-dimensional embedding array (rows: frames, columns: dimensions)
- **clim** (*array-like*) – list of two element: minimum and maximum bin number (default: None)
- **bin_size** (*float*) – size of the bin. if None, use interquartile range (IQR) to define the bin size (default: None)
- **cmap** (*str*) – color map (default: viridis)
- **show** (*bool*) – show the plot (default: True)

1.3.6 unrolr.plotting.plot_sampling module

unrolr.plotting.plot_sampling.**plot_sampling**(*fname*, *df*, *of='r_neighbor'*, *show=True*)
Helper function to plot results from sampling (neighborhood radii or iterations)

Parameters

- **fname** (*str*) – filename of the figure
- **df** (*DataFrame*) – Pandas DataFrame obtained from functions neighborhood_radius_sampler or optimization_cycle_sampler
- **of** (*str*) – Show the evolution of stress and correlation in function of "r_neighbor" or "n_iter" (choices: r_neighbor or n_iter) (default: r_neighbor)
- **show** (*bool*) – show the plot (default: True)

1.3.7 unrolr.utils module

unrolr.utils.utils.**read_dataset**(*fname*, *dname*, *start=0*, *stop=-1*, *skip=1*)
Read dataset from HDF5 file.

unrolr.utils.utils.**save_dataset**(*fname*, *dname*, *data*)
Save dataset to HDF5 file.

unrolr.utils.utils.**transform_dihedral_to_metric**(*dihedral_timeseries*)
Convert angles in radians to sine/cosine transformed coordinates.

The output will be used as the PCA input for dihedral PCA (dPCA)

Parameters **dhedral_timeseries** (*ndarray*) – array containing dihedral angles, shape (n_samples, n_features)

Returns sine/cosine transformed coordinates

Return type ndarray

unrolr.utils.utils.**transform_dihedral_to_circular_mean**(*dihedral_timeseries*)
Convert angles in radians to circular mean transformed angles.

The output will be used as the PCA input for dihedral PCA+ (dPCA+)

Parameters **dhedral_timeseries** (*ndarray*) – array containing dihedral angles, shape (n_samples, n_features)

Returns circular mean transformed angles

Return type ndarray

unrolr.utils.utils.**is_opencl_env_defined**()
Check if OpenCL env. variable is defined.

unrolr.utils.utils.**path_module**(*module_name*)

unrolr.utils.utils.**max_conformations_from_dataset**(*fname*, *dname*)
Get maximum number of conformations that can fit into the memory of the selected OpenCL device and also the step/interval

Python Module Index

u

Index

Symbols

D

F

I

M

N

O

P

R

S

T

U