ProDy

Protein Dynamics & Sequence Analysis

Trajectory Analysis

Release

Ahmet Bakan, Cihan Kaya

May 17, 2024

CHAPTER

ONE

INTRODUCTION

This tutorial shows how to analyze molecular dynamics trajectories, including essential dynamics analysis. This tutorial shows frame-by-frame analysis of trajectories, so it is particularly helpful for analysis of long trajectories that do not fit in your computers memory.

1.1 Required Programs

Latest versions of **ProDy_** and **Matplotlib_** are required.

1.2 Recommended Programs

IPython_ and **Scipy_** are strongly recommended.

1.3 Getting Started

To follow this tutorial, you will need the following files:

```
8.4M Feb 29 20:20 mdm2.dcd
112K Feb 29 20:20 mdm2.pdb
8.4M Feb 29 20:20 mdm2sim2.dcd
```

We recommend that you will follow this tutorial by typing commands in an IPython session, e.g.:

```
$ ipython
```

or with pylab environment:

```
$ ipython --pylab
```

First, we will make necessary imports from ProDy and Matplotlib packages.

```
In [1]: from prody import *

In [2]: from pylab import *

In [3]: ion()
```

We have included these imports in every part of the tutorial, so that code copied from the online pages is complete. You do not need to repeat imports in the same Python session.

CHAPTER

TWO

FRAMES AND ATOM GROUPS

This part shows how to use `AtomGroup` in place of `Frame`.

2.1 Trajectory Frames

`Frame` instances store only coordinate and some frame related data. For each frame data, you will a different frame instance:

```
In [1]: from prody import *

In [2]: dcd = Trajectory('mdm2.dcd')

In [3]: dcd
Out[3]: <Trajectory: mdm2 (next 0 of 500 frames; 1449 atoms)>

In [4]: frame0 = dcd.next()

In [5]: frame0
Out[5]: <Frame: 0 from mdm2 (1449 atoms)>

In [6]: frame1 = dcd.next()

In [7]: frame1
Out[7]: <Frame: 1 from mdm2 (1449 atoms)>
```

These `Frame` instances are different objects:

```
In [8]: frame0 is frame1
Out[8]: False
```

When you are not referring to any of these frames anymore in your code, Python garbage collector will free or reuse the memory space that was used by those frames.

2.2 Linking Atom Groups

When trajectory is not linked to an `AtomGroup` (using `link()`), `Frame` and `AtomGroup` objects share the same coordinate data. Let's see how this works:

When an `AtomGroup` is linked to the trajectory as follows, things work differently:

```
In [9]: pdb = parsePDB('mdm2.pdb')

In [10]: pdb
Out[10]: <AtomGroup: mdm2 (1449 atoms)>
```

```
In [11]: dcd.link(pdb)

In [12]: dcd.reset()
```

We get `Frame` instances in the same way:

```
In [13]: frame0 = dcd.next()

In [14]: frame0
Out[14]: <Frame: 0 from mdm2 (1449 atoms)>

In [15]: pdb.getACSLabel()
Out[15]: 'mdm2 frame 0'
```

Note that the active coordinate set of the `AtomGroup` and its label will change when we get the next frame:

```
In [16]: frame1 = dcd.next()

In [17]: frame1
Out[17]: <Frame: 1 from mdm2 (1449 atoms)>

In [18]: pdb.getACSLabel()
Out[18]: 'mdm2 frame 1'
```

Now the key difference is that the `Frame` instances are the same objects in this case:

```
In [19]: frame0 is frame1
Out[19]: False
```

As you see, a new frame was not instantiated. The same frame is reused and it always points to the coordinates stored in the `AtomGroup`. You can also make `Selection` instances that will point to the same coordinate set. This will allow making a more elaborate analysis of frames. For an example see *Trajectory Analysis II*.

CHAPTER

THREE

TRAJECTORY ANALYSIS

This example shows how to analyze a trajectory in DCD format. RMSD, RMSF, radius of gyration, and distance will be calculated from trajectory frames.

3.1 Input files

Currently, ProDy supports only DCD format files. Two DCD trajectory files and corresponding PDB structure file are needed for this example:

- MDM2 files (ZIP)
- MDM2 files (TGZ)

3.2 Setup environment

We start by importing everything from ProDy:

```
In [1]: from prody import *

In [2]: from pylab import *

In [3]: ion()
```

3.3 Parse structure

The PDB file provided with this example contains an X-ray structure which will be useful in a number of places, so let's start with parsing this file first:

```
In [4]: structure = parsePDB('mdm2.pdb')

In [5]: repr(structure)
Out[5]: '<AtomGroup: mdm2 (1449 atoms)>'
```

This function returned a `AtomGroup` instance that stores all atomic data parsed from the PDB file.

3.4 Parse all frames

Using `parseDCD()` function all coordinate data in the DCD file can be parsed at once. This function returns an `Ensemble` instance:

```
In [6]: ensemble = parseDCD('mdm2.dcd')

In [7]: repr(ensemble)
Out[7]: '<Ensemble: mdm2 (0:500:1) (500 conformations; 1449 atoms)>'
```

Note: When parsing large DCD files at once, memory may become an issue. If the size of the DCD file is larger than half of the RAM in your machine, consider parsing DCD files frame-by-frame. See the following subsection for details.

Let's associate this ensemble with the *structure* we parsed from the PDB file:

```
In [8]: ensemble.setAtoms(structure)

In [9]: ensemble.setCoords(structure)
```

This operation set the coordinates of the *structure* as the reference coordinates of the *ensemble*. Now we can `Ensemble.superpose()` the *ensemble* onto the coordinates of the *structure*.

```
In [10]: ensemble.superpose()
```

Now, we can calculate RMSDs and RMSFs as follows:

```
In [11]: rmsd = ensemble.getRMSDs()

In [12]: rmsd[:10]
Out[12]:
array([0.95948706, 1.37571761, 1.86486719, 1.67050981, 1.81932183,
       1.99846088, 1.83607975, 1.85453853, 1.72450348, 1.99616932])

In [13]: rmsf = ensemble.getRMSFs()

In [14]: rmsf
Out[14]:
array([2.16987058, 2.50997848, 2.54671469, ..., 2.40085027, 2.35998639,
       2.36161442])
```

Preceding calculations used all atoms in the structure. When we are interested in a subset of atoms, let's say Cα atoms, we can make a selection before performing calculations:

```
In [15]: ensemble.setAtoms(structure.calpha)

In [16]: repr(ensemble)
Out[16]: '<Ensemble: mdm2 (0:500:1) (500 conformations; selected 85 of 1449 atoms)>'

In [17]: ensemble.superpose()
```

In this case, superposition was based on Cα atom coordinates.

```
In [18]: rmsd = ensemble.getRMSDs()

In [19]: rmsd[:10]
Out[19]:
array([0.57036271, 0.65563502, 1.07986844, 0.87149569, 1.00810079,
       1.08325961, 0.97407076, 0.97066068, 0.71017147, 0.98885846])

In [20]: rmsf = ensemble.getRMSFs()

In [21]: rmsf
```

```
Out[21]:
array([1.6330461 , 1.23173403, 0.80235834, 0.5958921 , 0.50945672,
       0.45622768, 0.4464998 , 0.56223584, 0.54737841, 0.43845661,
       0.49619213, 0.55505813, 0.5036435 , 0.56143506, 0.68243978,
       0.63595968, 0.65944352, 0.79793022, 0.616282  , 0.84929821,
       0.87003225, 1.36270963, 1.02894487, 0.58924426, 0.75705043,
       0.75241394, 0.80858253, 0.63143782, 0.50065952, 0.62382299,
       0.59612102, 0.4654029 , 0.55655523, 0.62571376, 0.56086838,
       0.50027906, 0.47802536, 0.62719066, 0.69342567, 0.6609691 ,
       0.63199592, 0.45820843, 0.55381   , 0.81272202, 1.429093  ,
       1.61484996, 1.3844535 , 1.449554  , 1.009925  , 0.56484553,
       0.44251167, 0.46495854, 0.51425355, 0.66156275, 0.78570958,
       0.51504623, 0.53755642, 0.48657396, 0.51736938, 0.59835881,
       0.6156337 , 0.67893898, 0.66840022, 0.70192971, 0.64153721,
       0.56550165, 0.60452546, 0.66527624, 0.85178562, 0.89024996,
       0.93976546, 0.99426097, 0.90499485, 0.7658395 , 0.64180648,
       0.60419211, 0.67995234, 0.68370779, 0.53503999, 0.64867237,
       0.79336428, 0.58609028, 0.54535261, 0.73394662, 1.55134084])
```

The `Ensemble` instance can also be used in `PCA` calculations. See the examples in Ensemble Analysis[1] for more information.

3.5 Parse frames one-by-one

```
In [22]: dcd = DCDFile('mdm2.dcd')

In [23]: repr(dcd)
Out[23]: '<DCDFile: mdm2 (next 0 of 500 frames; 1449 atoms)>'
```

```
In [24]: structure = parsePDB('mdm2.pdb')

In [25]: dcd.setCoords(structure)

In [26]: dcd.link(structure)

In [27]: dcd.nextIndex()
Out[27]: 0

In [28]: frame = dcd.next()

In [29]: repr(frame)
Out[29]: '<Frame: 0 from mdm2 (1449 atoms)>'

In [30]: dcd.nextIndex()
Out[30]: 1
```

```
In [31]: frame.getRMSD()
Out[31]: 1.0965813503989288

In [32]: frame.superpose()

In [33]: frame.getRMSD()
Out[33]: 0.9594870434519088

In [34]: calcGyradius(frame)
Out[34]: 12.950192748991222
```

[1] http://prody.csb.pitt.edu/tutorials/ensemble_analysis/index.html#pca

We can perform these calculations for all frames in a for loop. Let's reset *dcd* to return to the 0th frame:

```
In [35]: dcd.reset()

In [36]: rgyr = zeros(len(dcd))

In [37]: rmsd = zeros(len(dcd))

In [38]: for i, frame in enumerate(dcd):
   ....:     rgyr[i] = calcGyradius(frame)
   ....:     frame.superpose()
   ....:     rmsd[i] = frame.getRMSD()
   ....:

In [39]: rmsd[:10]
Out[39]:
array([0.95948704, 1.37571759, 1.86486716, 1.6705098 , 1.81932183,
       1.99846087, 1.83607974, 1.85453852, 1.72450347, 1.99616931])

In [40]: rgyr[:10]
Out[40]:
array([12.95019275, 13.07770448, 12.93054754, 13.02506153, 12.95834748,
       13.01555473, 12.86652426, 12.93371279, 12.89667858, 12.86328841])
```

3.6 Handling multiple files

`Trajectory` is designed for handling multiple trajectory files:

```
In [41]: traj = Trajectory('mdm2.dcd')

In [42]: repr(traj)
Out[42]: '<Trajectory: mdm2 (next 0 of 500 frames; 1449 atoms)>'

In [43]: traj.addFile('mdm2sim2.dcd')

In [44]: repr(traj)
Out[44]: '<Trajectory: mdm2 (2 files; next 0 of 1000 frames; 1449 atoms)>'
```

Instances of this class are also suitable for previous calculations:

```
In [45]: structure = parsePDB('mdm2.pdb')

In [46]: traj.link(structure)

In [47]: traj.setCoords(structure)

In [48]: rgyr = zeros(len(traj))

In [49]: rmsd = zeros(len(traj))

In [50]: for i, frame in enumerate(traj):
   ....:     rgyr[i] = calcGyradius( frame )
   ....:     frame.superpose()
   ....:     rmsd[i] = frame.getRMSD()
   ....:

In [51]: rmsd[:10]
Out[51]:
```

```
array([0.95948704, 1.37571759, 1.86486716, 1.6705098 , 1.81932183,
       1.99846087, 1.83607974, 1.85453852, 1.72450347, 1.99616931])

In [52]: rgyr[:10]
Out[52]:
array([12.95019275, 13.07770448, 12.93054754, 13.02506153, 12.95834748,
       13.01555473, 12.86652426, 12.93371279, 12.89667858, 12.86328841])
```

CHAPTER

FOUR

TRAJECTORY ANALYSIS II

This example shows how to perform a more elaborate calculations simultaneously. Radius of gyration, distance, psi angle calculated will be calculated using trajectory frames.

4.1 Input files

Two DCD trajectory files and a PDB structure file is provided for this example:

- MDM2 files (ZIP)
- MDM2 files (TGZ)

4.2 Setup environment

We start by importing everything from ProDy:

```
In [1]: from prody import *

In [2]: from pylab import *

In [3]: ion()
```

4.3 Parse structure

The PDB file provided with this example contains and X-ray structure which will be useful in a number of places, so let's start with parsing this file first:

```
In [4]: structure = parsePDB('mdm2.pdb')

In [5]: structure
Out[5]: <AtomGroup: mdm2 (1449 atoms)>
```

This function returned a `AtomGroup` instance that stores all atomic data parsed from the PDB file.

4.4 Handling multiple files

`Trajectory` is designed for handling multiple trajectory files:

```
In [6]: traj = Trajectory('mdm2.dcd')

In [7]: traj
Out[7]: <Trajectory: mdm2 (next 0 of 500 frames; 1449 atoms)>
```

```
In [8]: traj.addFile('mdm2sim2.dcd')

In [9]: traj
Out[9]: <Trajectory: mdm2 (2 files; next 0 of 1000 frames; 1449 atoms)>
```

4.5 Link trajectory to atoms

Atoms can be linked to the trajectory as follows:

```
In [10]: traj.link(structure)

In [11]: traj.setCoords(structure)
```

When an atom group is linked to a trajectory, frame coordinates parsed from trajectory files will overwrite coordinates of the atom group. By making atom selections, you can calculate and analyze different properties.

4.6 Setup for calculations

Let's make atom selections for different types of calculations:

4.6.1 End-to-end distance

We select atoms from terminal residues and make an empty array whose length equal to the number of frames:

```
In [12]: nter = structure.select('name CA and resnum 25')

In [13]: cter = structure.select('name CA and resnum 109')

In [14]: e2e = zeros(traj.numFrames())
```

4.6.2 Radius of gyration

We select atoms protein atoms this calculation and make an empty array:

```
In [15]: protein = structure.select('noh and protein')

In [16]: rgyr = zeros(traj.numFrames())
```

4.6.3 A psi angle

We select a residue an make an empty array:

```
In [17]: res30 = structure['PPP', 'P', 30]

In [18]: res30
Out[18]: <Residue: PRO 30 from Chain P from mdm2 (14 atoms)>

In [19]: res30psi = zeros(traj.numFrames())
```

4.7 Perform calculations

We perform all calculations simultaneously as follows:

```
In [20]: for i, frame in enumerate(traj):
   ....:     e2e[i] = calcDistance(nter, cter)
   ....:     res30psi[i] = calcPsi(res30)
   ....:     rgyr[i] = calcGyradius(protein)
   ....:
```

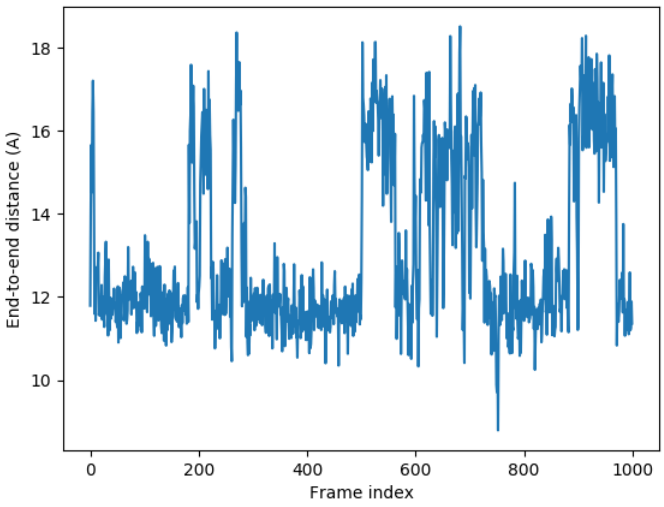
Let's print part of results:

```
In [21]: e2e[:10]
Out[21]:
array([11.78980637, 14.12566566, 15.6633606 , 14.52022934, 16.45702362,
       17.20821953, 16.45432854, 14.28651619, 11.59599113, 12.66241741])

In [22]: rgyr[:10]
Out[22]:
array([12.85520891, 12.98176002, 12.82791643, 12.91550011, 12.87289194,
       12.92314255, 12.76350405, 12.85890813, 12.81869193, 12.76178599])

In [23]: res30psi[:10]
Out[23]:
array([149.81183155, 170.65785032, 139.9378317 , 156.36605157,
       139.49376043, 151.11160105, 147.68076198, 151.81761523,
       143.4179355 , 155.13133287])
```

4.8 Plot results

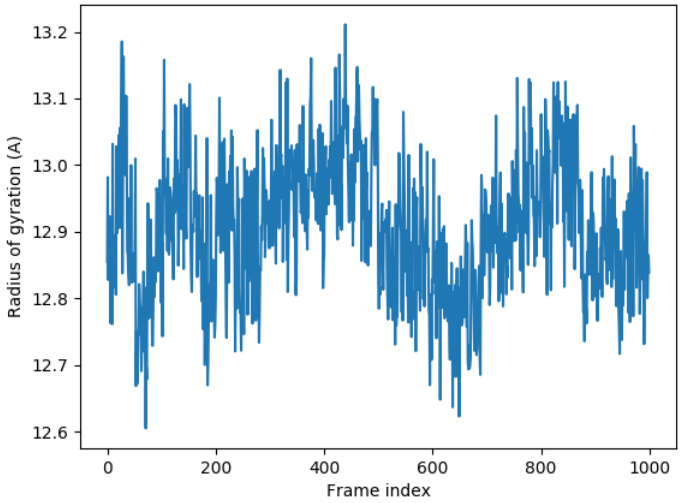
4.8.1 End-to-end distance

```
In [24]: plot(e2e);

In [25]: xlabel('Frame index');

In [26]: ylabel('End-to-end distance (A)');
```



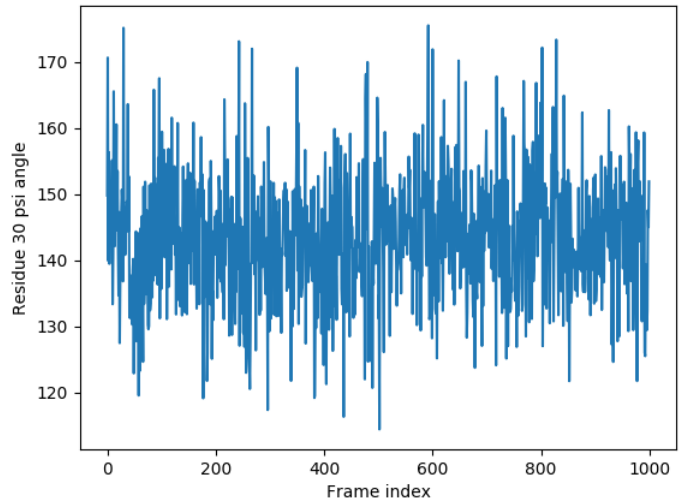
4.8.2 Radius of gyration

```
In [27]: plot(rgyr);

In [28]: xlabel('Frame index');

In [29]: ylabel('Radius of gyration (A)');
```



4.8.3 A psi angle

```
In [30]: plot(res30psi);

In [31]: xlabel('Frame index');

In [32]: ylabel('Residue 30 psi angle');
```



CHAPTER

FIVE

ESSENTIAL DYNAMICS ANALYSIS

5.1 Synopsis

This example shows how to perform essential dynamics analysis of molecular dynamics (MD) trajectories. A `EDA` instance that stores covariance matrix and principal modes that describes the essential dynamics of the system observed in the simulation will be built. `EDA` and principal modes (`Mode`) can be used as input to functions in `dynamics` module for further analysis.

User needs to provide trajectory in DCD file format and PDB file of the system.

Example input:

- MDM2 files (ZIP)
- MDM2 files (TGZ)

5.2 Setup environment

We start by importing everything from ProDy:

```
In [1]: from prody import *

In [2]: from pylab import *

In [3]: ion()
```

5.3 Parse reference structure

The PDB file provided with this example contains and X-ray structure which will be useful in a number of places, so let's start with parsing this file first:

```
In [4]: structure = parsePDB('mdm2.pdb')

In [5]: structure
Out[5]: <AtomGroup: mdm2 (1449 atoms)>
```

This function returned a `AtomGroup` instance that stores all atomic data parsed from the PDB file.

5.4 EDA calculations

Essential dynamics analysis (EDA or PCA) of a trajectory can be performed in two ways.

5.4.1 Small files

If you are analyzing a small trajectory, you can use an `Ensemble` instance obtained by parsing the trajectory at once using `parseDCD()`:

```
In [6]: ensemble = parseDCD('mdm2.dcd')

In [7]: ensemble.setCoords(structure)

In [8]: ensemble.setAtoms(structure.calpha)

In [9]: ensemble
Out[9]: <Ensemble: mdm2 (0:500:1) (500 conformations; selected 85 of 1449 atoms)>

In [10]: ensemble.superpose()

In [11]: eda_ensemble = EDA('MDM2 Ensemble')

In [12]: eda_ensemble.buildCovariance( ensemble )

In [13]: eda_ensemble.calcModes()

In [14]: eda_ensemble
Out[14]: <EDA: MDM2 Ensemble (20 modes; 85 atoms)>
```

5.4.2 Large files

If you are analyzing a large trajectory, you can pass the trajectory instance to the `PCA.buildCovariance()` method as follows:

```
In [15]: dcd = DCDFile('mdm2.dcd')

In [16]: dcd.link(structure)

In [17]: dcd.setAtoms(structure.calpha)

In [18]: dcd
Out[18]: <DCDFile: mdm2 (linked to AtomGroup mdm2; next 0 of 500 frames; selected 85 of 1449 atoms)>

In [19]: eda_trajectory = EDA('MDM2 Trajectory')

In [20]: eda_trajectory.buildCovariance( dcd )

In [21]: eda_trajectory.calcModes()

In [22]: eda_trajectory
Out[22]: <EDA: MDM2 Trajectory (20 modes; 85 atoms)>
```

5.4.3 Comparison

```
In [23]: printOverlapTable(eda_ensemble[:3], eda_trajectory[:3])
Overlap Table
                         EDA MDM2 Trajectory
                            #1     #2     #3
EDA MDM2 Ensemble #1     +1.00   0.00   0.00
EDA MDM2 Ensemble #2      0.00  +1.00   0.00
EDA MDM2 Ensemble #3      0.00   0.00  +1.00
```

Overlap values of +1 along the diagonal of the table shows that top ranking 3 essential (principal) modes are the same.

5.5 Multiple files

It is also possible to analyze multiple trajectory files without concatenating them. In this case we will use data from two independent simulations

```
In [24]: trajectory = Trajectory('mdm2.dcd')

In [25]: trajectory.addFile('mdm2sim2.dcd')

In [26]: trajectory
Out[26]: <Trajectory: mdm2 (2 files; next 0 of 1000 frames; 1449 atoms)>

In [27]: trajectory.link(structure)

In [28]: trajectory.setCoords(structure)

In [29]: trajectory.setAtoms(structure.calpha)

In [30]: trajectory
Out[30]: <Trajectory: mdm2 (linked to AtomGroup mdm2; 2 files; next 0 of 1000 frames; selected 85 of

In [31]: eda = EDA('mdm2')

In [32]: eda.buildCovariance( trajectory )

In [33]: eda.calcModes()

In [34]: eda
Out[34]: <EDA: mdm2 (20 modes; 85 atoms)>
```

5.5.1 Save your work

You can save your work using ProDy function `saveModel()`. This will allow you to avoid repeating calculations when you return to your work later:

```
In [35]: saveModel(eda)
Out[35]: 'mdm2.eda.npz'
```

`loadModel()` function can be used to load this object without any loss.

5.6 Analysis

Let's print fraction of variance for top raking 4 essential modes:

```
In [36]: for mode in eda_trajectory[:4]:
   ....:     print(calcFractVariance(mode).round(2))
   ....:
0.26
0.11
0.08
0.06
```

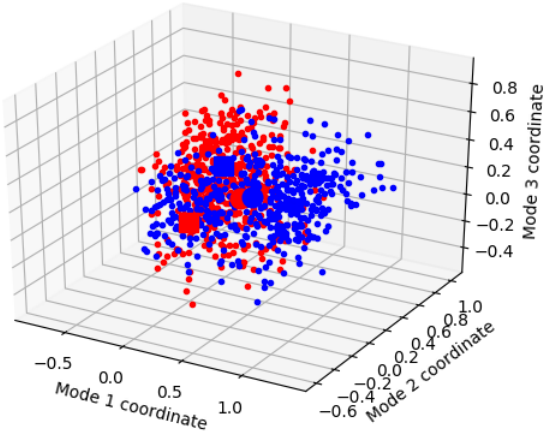
You can find more analysis functions in Dynamics Analysis[2].

[2] http://prody.csb.pitt.edu/manual/reference/dynamics/index.html#dynamics

5.7 Plotting

Now, let's project the trajectories onto top three essential modes:

```
In [37]: mdm2ca_sim1 = trajectory[:500]

In [38]: mdm2ca_sim1.superpose()

In [39]: mdm2ca_sim2 = trajectory[500:]

In [40]: mdm2ca_sim2.superpose()

# We project independent trajectories in different color
In [41]: showProjection(mdm2ca_sim1, eda[:3], color='red', marker='.');

In [42]: showProjection(mdm2ca_sim2, eda[:3], color='blue', marker='.');

# Now let's mark the beginning of the trajectory with a circle
In [43]: showProjection(mdm2ca_sim1[0], eda[:3], color='red', marker='o', ms=12);

In [44]: showProjection(mdm2ca_sim2[0], eda[:3], color='blue', marker='o', ms=12);

# Now let's mark the end of the trajectory with a square
In [45]: showProjection(mdm2ca_sim1[-1], eda[:3], color='red', marker='s', ms=12);

In [46]: showProjection(mdm2ca_sim2[-1], eda[:3], color='blue', marker='s', ms=12);
```



You can find more plotting functions in Dynamics Analysis[3] and Measurement Tools[4] modules.

5.8 Visualization

The above projection is shown for illustration. Interpreting the essential modes and projection of snapshots onto them is case dependent. One should know what kind of motion the top essential modes describe. You can use Normal Mode Wizard[5] for visualizing essential mode shapes and fluctuations along these modes.

[3] http://prody.csb.pitt.edu/manual/reference/dynamics/index.html#dynamics
[4] http://prody.csb.pitt.edu/manual/reference/measure/index.html#measure
[5] http://prody.csb.pitt.edu/tutorials/nmwiz_tutorial/intro.html#nmwiz

We can write essential modes in NMD Format[6] for NMWiz as follows:

```
In [47]: writeNMD('mdm2_eda.nmd', eda[:3], structure.select('calpha'))
Out[47]: 'mdm2_eda.nmd'
```

[6] http://prody.csb.pitt.edu/manual/reference/dynamics/nmdfile.html#nmd-format

CHAPTER

SIX

TRAJECTORY OUTPUT

This example shows how to output processed trajectories.

6.1 Input files

Currently, ProDy supports only DCD format files. Two DCD trajectory files and corresponding PDB structure file is needed for this example:

- MDM2 files (ZIP)
- MDM2 files (TGZ)

6.2 Setup environment

We start by importing everything from ProDy:

```
In [1]: from prody import *

In [2]: from pylab import *

In [3]: ion()
```

6.3 Load structure

The PDB file provided with this example contains an X-ray structure:

```
In [4]: mdm2 = parsePDB('mdm2.pdb')

In [5]: repr(mdm2)
Out[5]: '<AtomGroup: mdm2 (1449 atoms)>'
```

This function returned a `AtomGroup` instance that stores all atomic data parsed from the PDB file.

6.4 Open trajectories

`Trajectory` is designed for handling multiple trajectory files:

```
In [6]: traj = Trajectory('mdm2.dcd')

In [7]: traj
Out[7]: <Trajectory: mdm2 (next 0 of 500 frames; 1449 atoms)>
```

```
In [8]: traj.addFile('mdm2sim2.dcd')

In [9]: traj
Out[9]: <Trajectory: mdm2 (2 files; next 0 of 1000 frames; 1449 atoms)>
```

Now we link the trajectory (*traj*) with the atom group (*mdm2*):

```
In [10]: traj.link(mdm2)
```

Note: Note that when a frame (coordinate set) is parsed from the trajectory file, coordinates of the atom group will be updated.

6.5 Output selected atoms

You can write a trajectory in DCD format using `writeDCD()` function. Let's select non-hydrogen protein atoms and write a merged trajectory for MDM2:

```
In [11]: traj.setAtoms(mdm2.noh)

In [12]: traj
Out[12]: <Trajectory: mdm2 (linked to AtomGroup mdm2; 2 files; next 0 of 1000 frames; selected 706 of

In [13]: writeDCD('mdm2_merged_noh.dcd', traj)
Out[13]: 'mdm2_merged_noh.dcd'
```

Parsing this file returns:

```
In [14]: DCDFile('mdm2_merged_noh.dcd')
Out[14]: <DCDFile: mdm2_merged_noh (next 0 of 1000 frames; 706 atoms)>
```

6.6 Output aligned frames

You can write a trajectory in DCD format after aligning the frames. Let's return to the first frame by resetting the trajectory:

```
In [15]: traj.reset()

In [16]: traj
Out[16]: <Trajectory: mdm2 (linked to AtomGroup mdm2; 2 files; next 0 of 1000 frames; selected 706 of
```

It is possible to write multiple DCD files at the same time. We open two DCD files in write mode, one for all atoms, and another for backbone atoms:

```
In [17]: out = DCDFile('mdm2_aligned.dcd', 'w')

In [18]: out_bb = DCDFile('mdm2_bb_aligned.dcd', 'w')

In [19]: mdm2_bb = mdm2.backbone
```

Let's align and write frames one by one:

```
In [20]: for frame in traj:
   ....:     frame.superpose()
   ....:     out.write(mdm2)
```

```
....:     out_bb.write(mdm2_bb)
....:
```

Let's open these files to show number of atoms in each:

```
In [21]: DCDFile('mdm2_aligned.dcd')
Out[21]: <DCDFile: mdm2_aligned (next 0 of 1000 frames; 1449 atoms)>

In [22]: DCDFile('mdm2_bb_aligned.dcd')
Out[22]: <DCDFile: mdm2_bb_aligned (next 0 of 1000 frames; 339 atoms)>
```

Acknowledgments

Continued development of Protein Dynamics Software *ProDy* and associated programs is partially supported by the NIH[7]-funded Biomedical Technology and Research Center (BTRC) on *High Performance Computing for Multiscale Modeling of Biological Systems* (MMBios[8]) (P41 GM103712).

[7] http://www.nih.gov/
[8] http://mmbios.org/