ProDy

Protein Dynamics & Sequence Analysis

Cryo-EM Analysis

Release

Yan Zhang

June 07, 2024

CHAPTER

ONE

INTRODUCTION

This tutorial shows how to use cryo-EM electron density data to perform elastic network model analysis as in *[YZ20]*. Electron density maps can be provided by the user or retrieved from the EMDB[1].

Cryo-electron microscopy (cryo-EM) is a type of transmission electron microscopy where the studied sample, e.g. protein solution, is rapidly frozen. In structural biology, single particle cryo-EM is gaining more and more popularity due to its advantage of allowing the observation of protein under near-native conditions at ever-improving resolution, and as a result, the structures of numerous mega-Dalton biomolecules are captured and stored in the form of cryo-EM density maps, with resolutions ranging from 2 to 100 Å.

Fitting all-atom structures to a density map is time-consuming, and sometimes not feasible due to low resolution. In addition, for the purpose of studying the dynamics, MD simulations of supercomplexes with atomic details are extremely computationally expensive. It is thus desirable to apply coarse-grained methods, for example the Anisotropic Network Model (ANM), to such data to study the "big motions" of molecular machines beyond the atomic level.

ProDy can be used for constructing a bead-and-spring model from cryo-EM data using a previously published algorithm ([TM94]), which can be used for ANM.

1.1 Required Programs

Latest version of **ProDy_** is required.

1.2 Recommended Programs

We recommend a visualization program too, such as **VMD_**.

1.3 Getting Started

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

```
 31M Feb 29 20:20 2680.map
 31M Feb 29 20:20 2688.map
1.1M Feb 29 20:20 4a0v.pdb.gz
383K Feb 29 20:20 4uqj.pdb.gz
 12M Feb 29 20:20 EMD-1961.map
225K Feb 29 20:20 EMD-1961_mapped.pdb
231K Feb 29 20:20 EMD-1961.pdb
 55K Feb 29 20:20 EMD-2680_mapped.pdb
 78K Feb 29 20:20 EMD-2680.pdb
 57K Feb 29 20:20 EMD-2680_protein.pdb
```

[1] https://www.ebi.ac.uk/pdbe/emdb/

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

PROCESSING CRYO-EM ELECTRON DENSITY MAPS

Let's start with essential import statements:

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

In [2]: import numpy as np
```

2.1 Parse Density Map

The first step is to parse a .map file, which contains information about a density map as the electron density at points on a grid. This file format is a binary format also known as CCP4 or MRC2014.

We provide parameters to return pseudoatom beads from the electron density map using the Topology Representing Network algorithm.

Note: Depending on your hardware and the system size, this may take a while. To skip this step, you can load the structure file for the beads directly in PDB format (EMD-1961.pdb; see below).

```
In [3]: emd = parseEMD('1961', cutoff=1.2, n_nodes=200)

In [4]: emd
Out[4]: <AtomGroup: 1961 (200 atoms)>
```

This function returns an `AtomGroup` from the electron density map. The **cutoff** parameter discards any electron density lower than the given number and we select this based on the suggestion on the **'EMDB'_** website. In this case, we raise it slightly to remove unassigned density inside the CCT rings.

The parameter **n_nodes** describes the total number of beads in the system, which we choose to be a few times smaller than the number of residues to increase efficiency and account for the low resolution.

This procedure is an iterative one and **num_iter** can be used to set the number of them (the default value is 20).

The resultant structure will look something like the following figure, although with much fewer nodes.

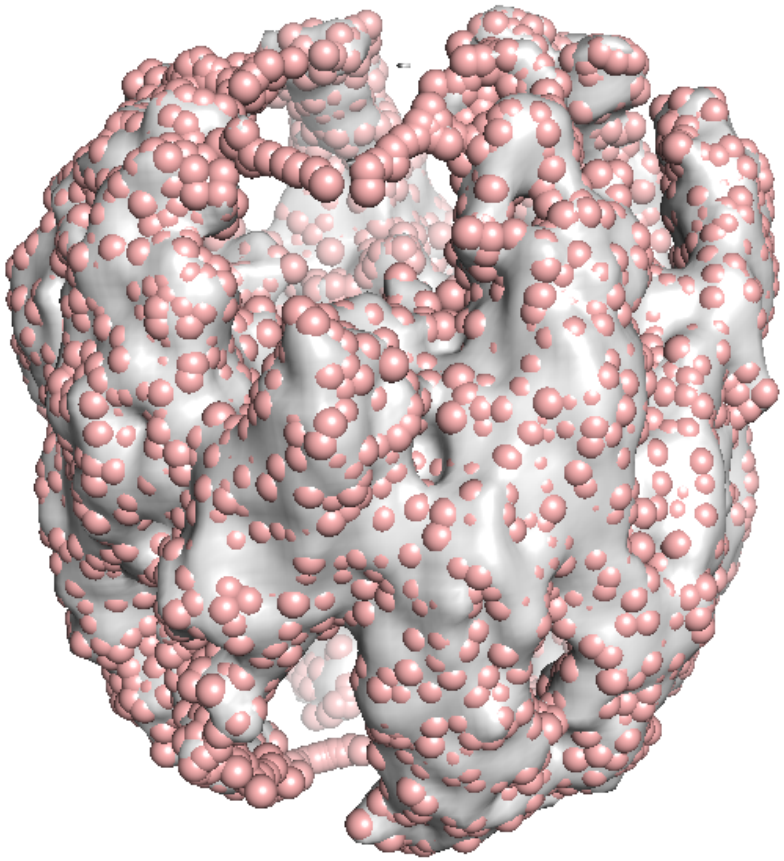
Now that the 144x144x144 density grid is converted into an `AtomGroup` object, elastic network model analysis can be applied to the constructed structure as usual.

First, let's save the structure so we can use it later:

```
In [5]: writePDB('EMD-1961.pdb', emd)
Out[5]: 'EMD-1961.pdb'
```

2.2 Map pseudo-atoms to PDB atomic model

To save time, you may parse pseudo-atoms from a PDB file generated as above. We also parse the PDB model for comparison later.



```
In [6]: emd = parsePDB('EMD-1961.pdb')

In [7]: pdb = parsePDB('4a0v', subset='ca')
```

The order of pseudo-atoms generated by TRN is random and does not follow a sequence like residues in an atomic model do. Also, they have no chain identifiers. For the purpose of visualization and later comparative analyses, we reorder the pseudo-atoms and assign the residue and chain identifers to the pseudo-atoms based on the PDB structure in the following section.

Unique one-to-one mapping of pseudo-atoms to an atomic model is nontrivial, since there is no correspondence between the beads and the residues of the protein. For simplicity, here we use the k-nearest neighbors algorithm to find 20 closest residues in the atomic model to a given pseudo-atom. Then we create a one-to-one mapping by assigning the closest residue, or the next closest if it is already assigned, to a pseudo-atom. Note that this is by no means the optimal one-to-one mapping, and for more complicated methods which guarantees the optimal mapping, see for example the "stable marriage problem".

```
In [8]: from sklearn.neighbors import NearestNeighbors

In [9]: mapping = []

In [10]: nbrs = NearestNeighbors(n_neighbors=20, algorithm='ball_tree').fit(pdb.getCoords())

In [11]: _, indices = nbrs.kneighbors(emd.getCoords())

In [12]: for neighbors in indices:
   ....:     for i in neighbors:
   ....:         if i not in mapping:
   ....:             mapping.append(i)
   ....:             break
   ....:

In [13]: indices = np.array(mapping)

In [14]: I = np.argsort(indices)
```

Note that `indices` returned from `NearestNeighbors` is a 2-D array with rows corresponding to pseudo-atoms and columns their k-neighbors. After being processed by the for-loop above, each element of `indices` is the index of the residue in the atomic model that should be assigned to the pseudo-atom. Then, `argsort()` is applied to obtain indices for reordering the pseudo-atoms following the order of the atoms (residues) in the atomic model.

We first create a `AtomMap` for the atomic model with only the residues that were mapped to a pseudo-atom.

```
In [15]: pmap = AtomMap(pdb, indices[I])
```

Then we create a new `AtomGroup` for the pseudo-atoms based on the mapping, such that they are ordered according to the sequence of residues they are assigned to:

```
In [16]: emd2 = AtomMap(emd, I).toAtomGroup()

In [17]: resnums = pmap.getResnums()

In [18]: emd2.setResnums(resnums)

In [19]: chids = pmap.getChids()

In [20]: emd2.setChids(chids)
```

Now we can calculate the RMSD between the pseudo-atoms and their mapped residues in the atomic model:

```
In [21]: calcRMSD(emd2, pmap)
Out[21]: 3.1359088897798038
```

Finally, we save the ordered pseudo-atom model to a PDB file for visualization and other downstream analyses:

```
In [22]: writePDB('EMD-1961_mapped.pdb', emd2)
Out[22]: 'EMD-1961_mapped.pdb'
```

CHAPTER

THREE

ELASTIC NETWORK MODEL ANALYSIS

Elastic network model analysis can be applied to the pseudo-atomic model as usual. We use `cutoff=60` to account for the level of coarse-graining (see *[PD02]*).

```
In [23]: anm_emd = ANM('TRiC EMDMAP ANM Analysis')

In [24]: anm_emd.buildHessian(emd2, cutoff=60)

In [25]: anm_emd.calcModes(n_modes=5)

In [26]: writeNMD('tric_anm_3_modes_200nodes.nmd', anm_emd[:3], emd2)
Out[26]: 'tric_anm_3_modes_200nodes.nmd'
```

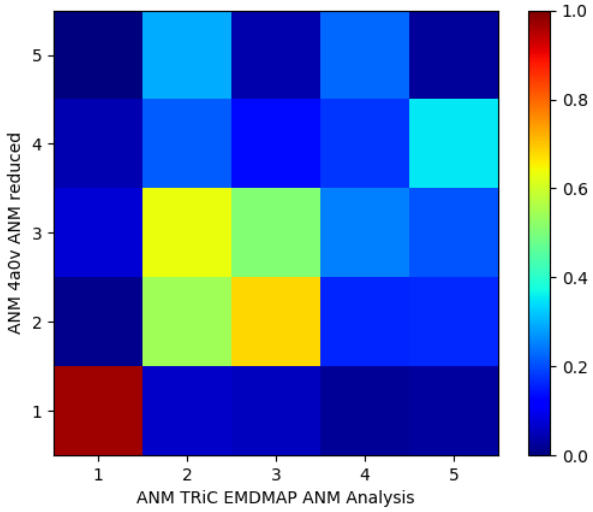
3.1 Compare results with atomic models

For comparison, let's perform ENM analysis for the atomic model (i.e. `pmap` we created earlier) as well, and apply the reduced model to it to treat residues that are not assigned to a pseudo-atom as the environment.

```
In [27]: anm_pdb = ANM('4a0v ANM')

In [28]: anm_pdb.buildHessian(pdb)

In [29]: anm_pdb_reduced, _ = reduceModel(anm_pdb, pdb, pmap)

In [30]: anm_pdb_reduced.calcModes(n_modes=5)
```

Now we compare modes of the pseudo-atomic model to the atomic model:

```
In [31]: showOverlapTable(anm_emd, anm_pdb_reduced)
Out[31]:
(<matplotlib.image.AxesImage at 0x7f85bbdd6490>,
 [],
 <matplotlib.colorbar.Colorbar at 0x7f85bbd94a10>)
```


ANM 4a0v ANM reduced
ANM TRiC EMDMAP ANM Analysis
1
2
3
4
5
1.0
0.8
0.6
0.4
0.2
0.0

CHAPTER

FOUR

DYNAMICAL DOMAIN DECOMPOSITION

Let's start with essential import statements:

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

In [2]: import numpy as np
```

We'll start by parsing the pseudo-atoms from a PDB file generated in the previous step.

```
In [3]: emd = parsePDB('EMD-1961.pdb')
```

The order of pseudo-atoms generated by TRN is random and does not follow a sequence like residues in an atomic model do. Nor do they have no chain identifiers.

As an alternative to reordering the pseudo-atoms and assigning residue and chain identifers to the pseudo-atoms based on the PDB structure in the previous section, we demonstrate here the use of dynamical domain decomposition using GNM modes with the function `calcGNMDomains()`. The number of dynamical domains obtained is approximately equal to the number of modes used so we use 9 modes to get a similar set of domains to our paper (*[YZ20]*) including the zero mode.

```
In [4]: gnm, _ = calcGNM(emd, selstr='all', zeros=True)

In [5]: domains = calcGNMDomains(gnm[:9])

In [6]: print(np.unique(domains))
[0 1 2 3 4 5 6 7 8]
```

We actually get 9 in this case as you will see in the figure at the end.

We can assign this data to the `AtomGroup` as follows:

```
In [7]: emd.setData('domain', domains)
```

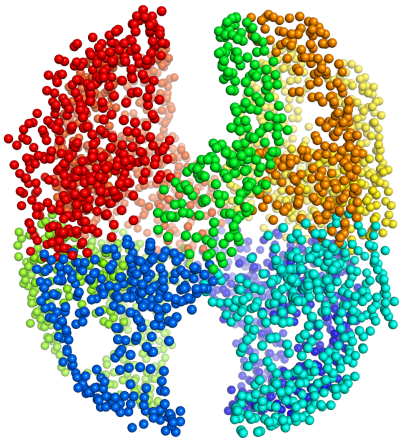
This allows the domains to be used for selection:

```
In [8]: domain1 = emd.select('domain 1')

In [9]: print(domain1.numAtoms())
207
```

Finally, we save a domain-labelled pseudo-atom model to a PDB file for visualization and other downstream analyses, using the B-factor field for writing the domains:

```
In [10]: writePDB('EMD-1961_domains.pdb', emd, beta=domains)
Out[10]: 'EMD-1961_domains.pdb'
```

Visualisation of the resultant structure will look something like the following figure.



We see 9 domains, including 7 pairs of domains and 2 individual domains.

Another round of decomposition with each of the resulting domains can be used to divide them further. For example, we can decompose the first domain (index 0) into 4 dynamical domains using the first 4 modes, which can be combined to create subunits or used as they are for comparison with dynamical domains for atomic models.

```
In [11]: gnm0, domain0 = calcGNM(emd, selstr='domain 0', zeros=True)

In [12]: domains0 = calcGNMDomains(gnm0[:4])

In [13]: writePDB('EMD-1961_domain0_subdomains4.pdb', domain0, beta=domains0)
Out[13]: 'EMD-1961_domain0_subdomains4.pdb'
```

Acknowledgments

Continued development of Protein Dynamics Software *ProDy* and associated programs is partially supported by the NIH[2]-funded Biomedical Technology and Research Center (BTRC) on *High Performance Computing for Multiscale Modeling of Biological Systems* (MMBios[3]) (P41 GM103712).

[2] http://www.nih.gov/
[3] http://mmbios.org/

[YZ20] Zhang Y, Krieger J, Mikulska-Ruminska K, Kaynak B, Sorzano COS, Carazo JM, Xing J, Bahar I State-dependent sequential allostery exhibited by chaperonin TRiC/CCT revealed by network analysis of Cryo-EM maps, *Prog. Biophys. Mol. Biol.* **In Press** 10.1016/j.pbiomolbio.2020.08.006.

[PD02] P. Doruker, R.L. Jernigan, I. Bahar, Dynamics of large proteins through hierarchical levels of coarse-grained structures, *J. Comput. Chem.* **2002** 23:119-127.