ProDy

Protein Dynamics & Sequence Analysis

Conformational Sampling

Release

Ahmet Bakan, Cihan Kaya

June 07, 2024

CONTENTS

CHAPTER

ONE

INTRODUCTION

This tutorial describes sampling alternate protein conformations along Anisotropic Network Model (ANM)[1] modes, then optimizing them using a molecular dynamics program. Conformations obtained in this way can be useful in, for example, docking studies when the target binding site is flexible and can be affected by motions of protein along collective modes.

We will use a structure of mitogen-activated protein kinase 14 (**:wiki:'MAPK14'**), which is also known as p38 MAPK. The structure identifier is **:pdb:'5uoj'**. PDB and PSF files are provided in documentation files.

1.1 Required Programs

Latest version of **ProDy_**, **Matplotlib_**, VMD[2] and NAMD[3] are required.

1.2 Recommended Programs

IPython_ is highly recommended for interactive usage.

1.3 Getting Started

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

```
 471 Feb 29 20:20 min.conf
437K Feb 29 20:20 p38.pdb
1.3M Feb 29 20:20 p38.psf
```

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()
```

[1] http://prody.csb.pitt.edu/tutorials/enm_analysis/anm.html#anm
[2] http://www.ks.uiuc.edu/Research/vmd/
[3] http://www.ks.uiuc.edu/Research/namd/

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

ANM CALCULATIONS

Required imports:

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

In [2]: from pylab import *

In [3]: ion()
```

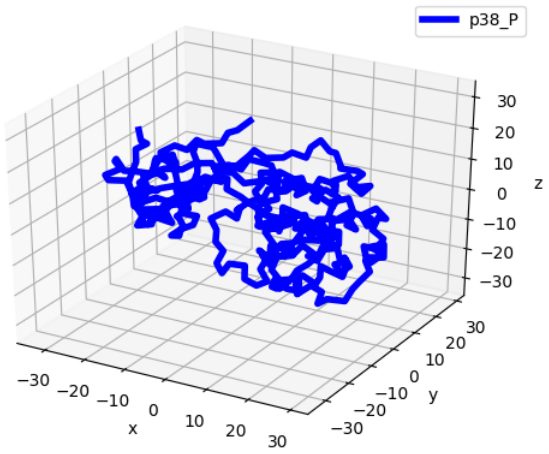
2.1 Atom Selection

First, we parse p38 structure `p38.pdb`:

```
In [4]: p38 = parsePDB('p38.pdb')

In [5]: p38
Out[5]: <AtomGroup: p38 (5658 atoms)>
```

Let's take a look at the structure:

```
In [6]: showProtein(p38)
Out[6]: <matplotlib.axes._subplots.Axes3DSubplot at 0x7fd1eb196190>

In [7]: legend()
Out[7]: <matplotlib.legend.Legend at 0x7fd1eb0c0350>
```



Note that this structure has hydrogen atoms which were added using PSFGEN that comes with NAMD:

```
In [8]: p38.numAtoms('hydrogen')
Out[8]: 2824
```

We will perform ANM calculations for 351 Cα atoms of the structure:

```
In [9]: p38_ca = p38.ca

In [10]: p38_ca
Out[10]: <Selection: 'ca' from p38 (351 atoms)>
```

2.2 ANM Calculation

First, let's instantiate an `ANM` object:

```
In [11]: p38_anm = ANM('p38 ca')

In [12]: p38_anm
Out[12]: <ANM: p38 ca (0 modes; 0 nodes)>
```

Now, we can build Hessian matrix, simply by calling `ANM.buildHessian()` method:

```
In [13]: p38_anm.buildHessian(p38_ca)

In [14]: p38_anm
Out[14]: <ANM: p38 ca (0 modes; 351 nodes)>
```

We see that `ANM` object contains 351 nodes, which correspond to the Cα atoms.

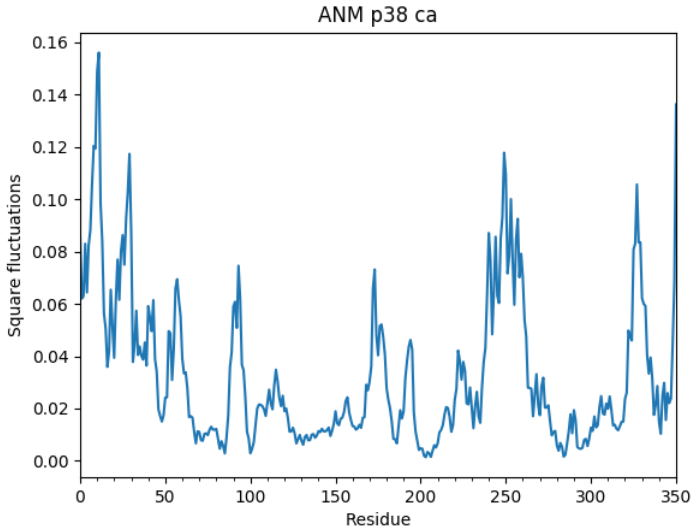
We will calculate only top ranking three ANM modes, since we are going to use only that many in sampling:

```
In [15]: p38_anm.calcModes(n_modes=3)

In [16]: p38_anm
Out[16]: <ANM: p38 ca (3 modes; 351 nodes)>
```

2.3 Analysis & Plotting

Let's plot mobility of residues along ANM modes:

```
In [17]: showSqFlucts(p38_anm);
```



We can also calculate collectivity of these modes as follows:

```
In [18]: for mode in p38_anm:
   ....:     print('{}\tcollectivity: {}'.format(str(mode), calcCollectivity(mode)))
   ....:
Mode 1 from ANM p38 ca        collectivity: 0.618084923455
Mode 2 from ANM p38 ca        collectivity: 0.585153336557
Mode 3 from ANM p38 ca        collectivity: 0.634434845453
```

2.4 Visualization

You can visualize ANM modes using Normal Mode Wizard[4]. You need to write an `.nmd` file using `writeNMD()` and open it using VMD:

```
In [19]: writeNMD('p38_anm.nmd', p38_anm, p38_ca)
Out[19]: 'p38_anm.nmd'
```

For visualization, you can use `viewNMDinVMD()`, i.e. `viewNMDinVMD('p38_anm.nmd')`

2.5 Extend Model

We want to use the ANM model to sample all atoms conformations of p38 MAPK, but we have a coarse-grained model. We will use `extendModel()` function for this purpose:

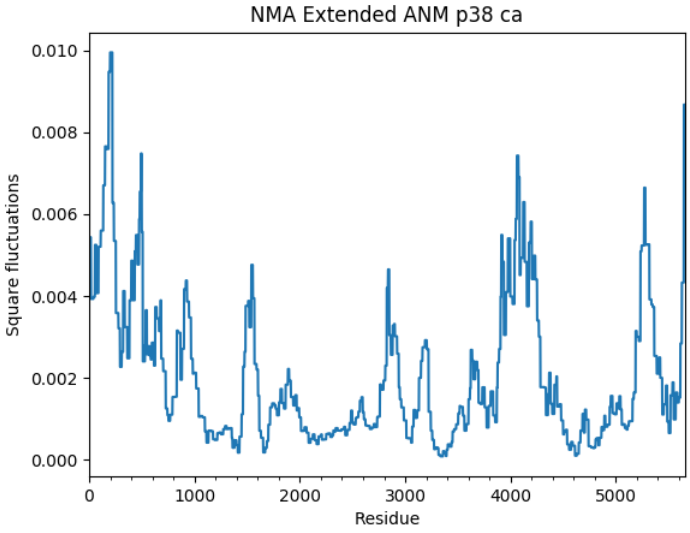
```
In [20]: p38_anm_ext, p38_all = extendModel(p38_anm, p38_ca, p38, norm=True)

In [21]: p38_anm_ext
Out[21]: <NMA: Extended ANM p38 ca (3 modes; 5658 atoms)>

In [22]: p38_all
Out[22]: <AtomMap: p38 from p38 (5658 atoms)>
```

Note `p38_anm_ext` is an `NMA` model, which has similar features to an `ANM` object. This extended model still has 3 modes, but 5668 atoms as opposed to 351 nodes in the original `ANM` model.

[4] http://prody.csb.pitt.edu/tutorials/nmwiz_tutorial/intro.html#nmwiz

Let's plot mobility of residues again to help understand what extending a model does:

```
In [23]: showSqFlucts(p38_anm_ext);
```



As you see, the shape of the mobility plot is identical. In the extended model, each atom moves in the same direction as the Cα atoms of the residue to which they belong. The mobility profile is scaled down, however, due to renormalization of the mode vectors.

2.6 Save Results

Now let's save the original and extended model, and atoms:

```
In [24]: saveAtoms(p38)
Out[24]: 'p38.ag.npz'

In [25]: saveModel(p38_anm)
Out[25]: 'p38_ca.anm.npz'

In [26]: saveModel(p38_anm_ext, 'p38_ext')
Out[26]: 'p38_ext.nma.npz'
```

2.7 More Examples

We have performed a quick ANM calculation and extended the resulting model to all atoms of of the structure. You can see more examples on this in Elastic Network Models[5] tutorial.

[5] http://prody.csb.pitt.edu/tutorials/enm_analysis/index.html#enm-analysis

CHAPTER

THREE

SAMPLE CONFORMATIONS

In this part, we will sample conformations along ANM modes.

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

In [2]: from pylab import *

In [3]: ion()
```

3.1 Load results

First, we load results produced in the previous part. If you are in the same Python session, you don't need to do this.

```
In [4]: p38 = loadAtoms('p38.ag.npz')

In [5]: p38_anm = loadModel('p38_ca.anm.npz')

In [6]: p38_anm_ext = loadModel('p38_ext.nma.npz')
```

3.2 Sampling

We will use the `sampleModes()` function:

```
In [7]: ens = sampleModes(p38_anm_ext, atoms=p38.protein, n_confs=40, rmsd=1.0)

In [8]: ens
Out[8]: <Ensemble: Conformations along NMA Extended ANM p38 ca (40 conformations; 5658 atoms)>
```

This will produce 40 (`n_confs`) conformations with have an average RMSD of 1.0 Å from the input structure.

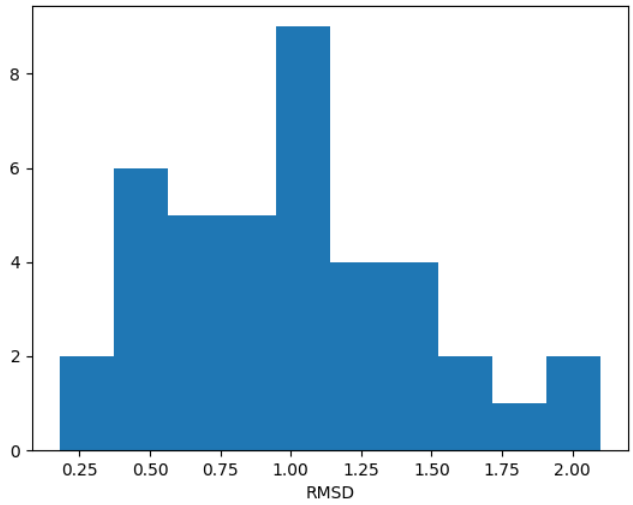
We can write this ensemble in a `.dcd` for visualization in VMD:

```
In [9]: writeDCD('p38all.dcd', ens)
```

3.3 Analysis

Let's analyze the `Ensemble` by plotting the RMSDs of all conformations relative to the input structure:

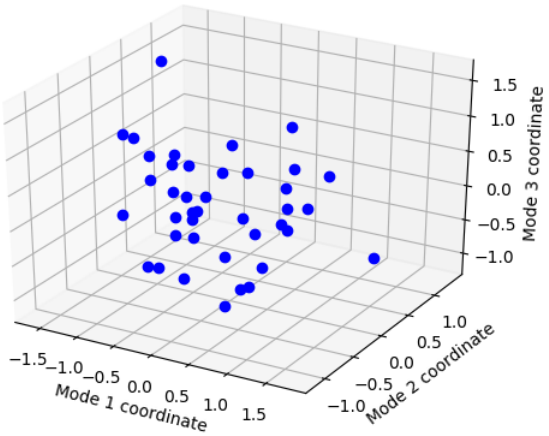
```
In [10]: rmsd = ens.getRMSDs()

In [11]: hist(rmsd, density=False);

In [12]: xlabel('RMSD');
```



This histogram might look like a flat distribution due to the small size of the ensemble. For larger numbers of conformations it will get closer to a normal distribution.

Let's see the projection of these conformations in the ANM slow mode space:

```
In [13]: showProjection(ens, p38_anm_ext[:3], rmsd=True);
```



3.4 Write conformations

We will write them in `p38_ensemble` folder:

```
In [14]: mkdir -p p38_ensemble
```

Let's add the conformations to the `AtomGroup` object and set beta[6] values of Cα atoms to 1 and of other atoms to 0:

[6] http://prody.csb.pitt.edu/manual/reference/atomic/fields.html#term-beta

```
In [15]: p38.addCoordset(ens.getCoordsets())

In [16]: p38
Out[16]: <AtomGroup: p38 (5658 atoms; active #0 of 41 coordsets)>

In [17]: p38.all.setBetas(0)

In [18]: p38.ca.setBetas(1)
```

In the next step, we will optimise the atom positions with a harmonic constraint on atoms with beta values of 1. The optimization aims to refine covalent geometry of atoms.We do not want the new Cα to change much to keep the refined ensemble diverse. We can easily verify that only Cα atoms have beta values set to 1:

```
In [19]: p38.ca == p38.beta_1
Out[19]: True
```

Now we write these conformations out:

```
In [20]: import os

In [21]: for i in range(1, p38.numCoordsets()):  # skipping 0th coordinate set
   ....:     fn = os.path.join('p38_ensemble', 'p38_' + str(i) + '.pdb')
   ....:     writePDB(fn, p38, csets=i)
   ....:
```

3.5 Visualization

You can visualize all of these conformations using VMD as follows:

```
$ vmd -m p38_ensemble/*pdb
```

CHAPTER

FOUR

OPTIMIZE CONFORMATIONS

In this part we will optimize the geometries of conformations generated in the previous step using NAMD.

4.1 Configuration

Let's find the location of NAMD executable:

```
In [1]: from prody.utilities import which

In [2]: namd2 = which('namd2')

In [3]: namd2
Out[3]: '/home/exx/NAMD_2.14_Linux-x86_64-multicore/namd2'
```

We will need a force field file for energy minimization. VMD ships with CHARMM force field files. We can write a tcl script to find and write their location as follows:

```
In [4]: tcl_cmd = '''package require readcharmmpar
   ...: package require readcharmmtop
   ...: global env
   ...: set outfile [open charmmdir.txt w]
   ...: puts $outfile $env(CHARMMPARDIR)
   ...: puts $outfile $env(CHARMMTOPDIR)
   ...: close $outfile
   ...: exit'''
   ...:

In [5]: with open('where_is_charmmpar.tcl', 'w') as inp:
   ...:     inp.write(tcl_cmd)
   ...:
```

This can be run in vmd from ipython as below:

```
In [6]: !vmd -dispdev text -e where_is_charmmpar.tcl
/home/exx/miniconda3/envs/py27/lib/vmd_LINUXAMD64: /lib/x86_64-linux-gnu/libGL.so.1: no version infor
Info) VMD for LINUXAMD64, version 1.9.3 (November 30, 2016)
Info) http://www.ks.uiuc.edu/Research/vmd/
Info) Email questions and bug reports to vmd@ks.uiuc.edu
Info) Please include this reference in published work using VMD:
Info)    Humphrey, W., Dalke, A. and Schulten, K., `VMD - Visual
Info)    Molecular Dynamics', J. Molec. Graphics 1996, 14.1, 33-38.
Info) ------------------------------------------------------------
Info) Multithreading available, 48 CPUs detected.
Info)   CPU features: SSE2 AVX AVX2 FMA KNL:AVX-512F+CD+ER+PF
Info) Free system memory: 122GB (97%)
```

```
Info) No CUDA accelerator devices available.
Info) Dynamically loaded 2 plugins in directory:

Info) /home/exx/miniconda3/envs/py27/lib/plugins/LINUXAMD64/molfile

1.3

1.2

file4

Info) VMD for LINUXAMD64, version 1.9.3 (November 30, 2016)
Info) Exiting normally.
vmd >
```

We then read the output file to get the parameter directory:

```
In [7]: inp = open('charmmdir.txt', 'r')

In [8]: lines = inp.readlines()

In [9]: inp.close()

In [10]: import os

In [11]: par = os.path.join(lines[0].strip(), 'par_all27_prot_lipid_na.inp')

In [12]: top = os.path.join(lines[1].strip(), 'top_all27_prot_lipid_na.inp')

In [13]: par
Out[13]: '/home/exx/miniconda3/envs/py27/lib/plugins/noarch/tcl/readcharmmpar1.3/par_all27_prot_lipi

In [14]: top
Out[14]: '/home/exx/miniconda3/envs/py27/lib/plugins/noarch/tcl/readcharmmtop1.2/top_all27_prot_lipi
```

To configure this computer

Let's make a folder for writing optimization input and output files:

```
In [15]: mkdir -p p38_optimize
```

We will write an NAMD configuration file for each conformation based on `min.conf`:

```
In [16]: import glob

In [17]: conf = open('conformational_sampling_files/min.conf').read()
---------------------------------------------------------------------------
IOError                                   Traceback (most recent call last)
<ipython-input-17-7c570bd80615> in <module>()
----> 1 conf = open('conformational_sampling_files/min.conf').read()

IOError: [Errno 2] No such file or directory: 'conformational_sampling_files/min.conf'

In [18]: for pdb in glob.glob(os.path.join('p38_ensemble', '*.pdb')):
   ....:     fn = os.path.splitext(os.path.split(pdb)[1])[0]
   ....:     pdb = os.path.join('..', pdb)
   ....:     out = open(os.path.join('p38_optimize', fn + '.conf'), 'w')
   ....:     out.write(conf.format(
   ....:         out=fn, pdb=pdb,
   ....:         par=par))
```

```
   ....:     out.close()
   ....:
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-18-3a8f7b33f646> in <module>()
      3     pdb = os.path.join('..', pdb)
      4     out = open(os.path.join('p38_optimize', fn + '.conf'), 'w')
----> 5     out.write(conf.format(
      6         out=fn, pdb=pdb,
      7         par=par))

NameError: name 'conf' is not defined
```

4.2 Optimization

Now we will run NAMD to optimize each of these conformations. We make a list of commands that we want to execute:

```
In [19]: os.chdir('p38_optimize')  # we will run commands in this folder

In [20]: cmds = []

In [21]: for conf in glob.glob('*.conf'):
   ....:     fn = os.path.splitext(conf)[0]
   ....:     cmds.append('namd2 ' + conf + ' > ' + fn + '.log')
   ....:

In [22]: cmds[:2]
Out[22]: ['namd2 p38_15.conf > p38_15.log']
```

We will run these commands using `multiprocessing`[7] module. We will allocate 3 processors for the job:

```
In [23]: from multiprocessing import Pool

In [24]: pool = Pool(3) # number of CPUs to use

In [25]: signals = pool.map(os.system, cmds)
```

`signals` will collect the output from execution of NAMD. If everything goes right, we should have only 0s.

```
In [26]: set(signals)
Out[26]: {256}
```

All NAMD output should be in `p38_optimize` folder. We go back to origional folder as follows:

```
In [27]: os.chdir('..')
```

[7] http://docs.python.org/library/multiprocessing.html#module-multiprocessing

CHAPTER

FIVE

ANALYZE CONFORMATIONS

First, necessary imports:

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

In [2]: from numpy import *

In [3]: from pylab import *

In [4]: ion()

In [5]: import os, glob
```

5.1 Parse conformations

Now, let's read initial and refined conformations:

```
In [6]: initial = AtomGroup('p38 initial')

In [7]: refined = AtomGroup('p38 refined')
```

```
In [8]: for pdb in glob.glob('p38_ensemble/*pdb'):
   ...:     fn = os.path.splitext(os.path.split(pdb)[1])[0]
   ...:     opt = os.path.join('p38_optimize', fn + '.coor')
   ...:     parsePDB(pdb, ag=initial)
   ...:     parsePDB(opt, ag=refined)
   ...:
---------------------------------------------------------------------------
MMCIFParseError                           Traceback (most recent call last)
<ipython-input-8-87036b051e71> in <module>()
      3     opt = os.path.join('p38_optimize', fn + '.coor')
      4     parsePDB(pdb, ag=initial)
----> 5     parsePDB(opt, ag=refined)
      6

/home/exx/ProDy-website/ProDy/prody/proteins/pdbfile.pyc in parsePDB(*pdb, **kwargs)
    125
    126     if n_pdb == 1:
--> 127         return _parsePDB(pdb[0], **kwargs)
    128     else:
    129         results = []

/home/exx/ProDy-website/ProDy/prody/proteins/pdbfile.pyc in _parsePDB(pdb, **kwargs)
    242         try:
    243             LOGGER.warn("Trying to parse as mmCIF file instead")
```

```
--> 244             return parseMMCIF(pdb, **kwargs)
    245         except KeyError:
    246             try:

/home/exx/ProDy-website/ProDy/prody/proteins/ciffile.pyc in parseMMCIF(pdb, **kwargs)
    130         kwargs['title'] = title
    131     cif = openFile(pdb, 'rt')
--> 132     result = parseMMCIFStream(cif, chain=chain, segment=segment, **kwargs)
    133     cif.close()
    134     if unite_chains:

/home/exx/ProDy-website/ProDy/prody/proteins/ciffile.pyc in parseMMCIFStream(stream, **kwargs)
    237
    238         _parseMMCIFLines(ag, lines, model, chain, subset, altloc,
--> 239                          segment, unite_chains, report)
    240
    241         if ag.numAtoms() > 0:

/home/exx/ProDy-website/ProDy/prody/proteins/ciffile.pyc in _parseMMCIFLines(atomgroup, lines, model,
    337                 stop = i
    338             else:
--> 339                 raise MMCIFParseError('mmCIF file contained no atoms.')
    340
    341         i += 1

MMCIFParseError: mmCIF file contained no atoms.
```

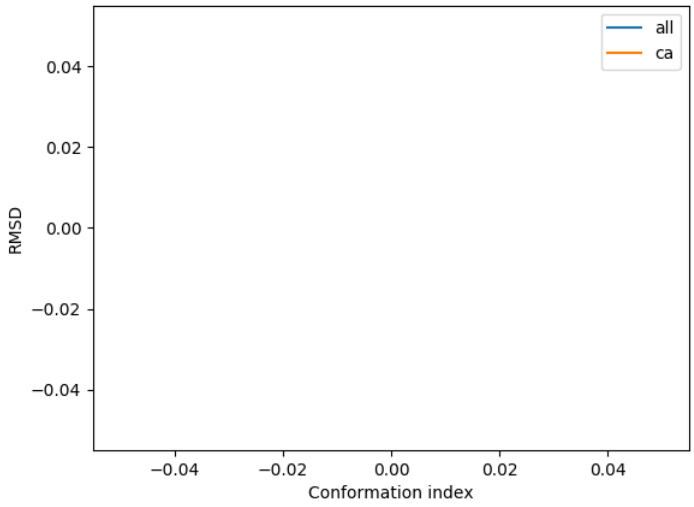
```
In [9]: initial
Out[9]: <AtomGroup: p38 initial (5658 atoms)>

In [10]: refined
Out[10]: <AtomGroup: p38 refined (0 atoms; no coordinates)>
```

5.2 Calculate RMSD change

We can plot RMSD change after refinement as follows:

```
In [11]: rmsd_ca = []

In [12]: rmsd_all = []

In [13]: initial_ca = initial.ca

In [14]: refined_ca = refined.ca
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-14-40de9febca08> in <module>()
----> 1 refined_ca = refined.ca

/home/exx/ProDy-website/ProDy/prody/atomic/atomic.pyc in __getattribute__(self, name)
     98                         dummies = self.numDummies()
     99                     except AttributeError:
--> 100                        indices = self._getSubset(name)
    101                        if len(indices):
    102                            return Selection(ag, indices, selstr,

/home/exx/ProDy-website/ProDy/prody/atomic/atomgroup.pyc in _getSubset(self, label)
```

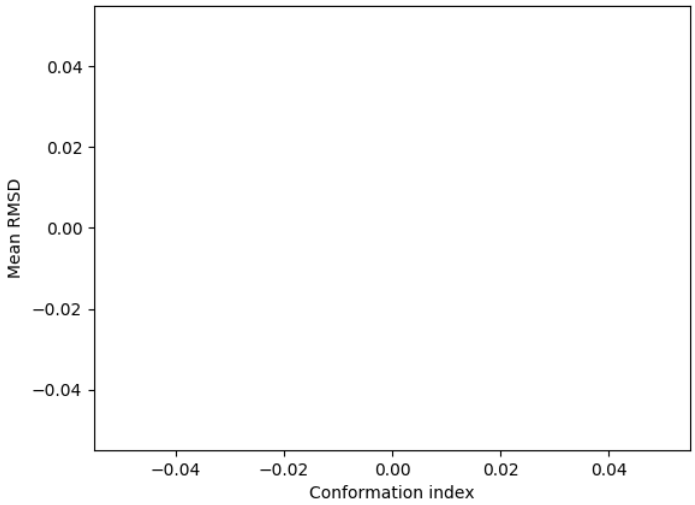
```
   1086             return self._subsets[label]
   1087         except KeyError:
-> 1088             flgs = self._getFlags(label)
   1089             try:
   1090                 return self._subsets[label]

/home/exx/ProDy-website/ProDy/prody/atomic/atomgroup.pyc in _getFlags(self, label)
   1028         except KeyError:
   1029             try:
-> 1030                 return FLAG_PLANTERS[label](self, label)
   1031             except KeyError:
   1032                 pass

/home/exx/ProDy-website/ProDy/prody/atomic/flags.pyc in setCalpha(ag, label)
    790
    791     flags = ag._getNames() == 'CA'
--> 792     indices = flags.nonzero()[0]
    793     if len(indices):
    794         torf = array([rn in AMINOACIDS for rn in ag._getResnames()[indices]])

AttributeError: 'bool' object has no attribute 'nonzero'

In [15]: for i in range(initial.numCoordsets()):
   ....:     initial.setACSIndex(i)
   ....:     refined.setACSIndex(i)
   ....:     initial_ca.setACSIndex(i)
   ....:     refined_ca.setACSIndex(i)
   ....:     rmsd_ca.append(calcRMSD(initial_ca, refined_ca))
   ....:     rmsd_all.append(calcRMSD(initial, refined))
   ....:
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-15-71aab0f1ba84> in <module>()
      1 for i in range(initial.numCoordsets()):
      2     initial.setACSIndex(i)
----> 3     refined.setACSIndex(i)
      4     initial_ca.setACSIndex(i)
      5     refined_ca.setACSIndex(i)

/home/exx/ProDy-website/ProDy/prody/atomic/atomgroup.pyc in setACSIndex(self, index)
    858             raise TypeError('index must be an integer')
    859         if n_csets <= index or n_csets < abs(index):
--> 860             raise IndexError('coordinate set index is out of range')
    861         if index < 0:
    862             index += n_csets

IndexError: coordinate set index is out of range

In [16]: plot(rmsd_all, label='all');

In [17]: plot(rmsd_ca, label='ca');

In [18]: xlabel('Conformation index');

In [19]: ylabel('RMSD');

In [20]: legend();
```



5.3 Select a diverse set

To select a diverse set of refined conformations, let's calculate average RMSD for each conformation to all others:

```
In [21]: rmsd_mean = []

In [22]: for i in range(refined.numCoordsets()):
   ....:     refined.setACSIndex(i)
   ....:     alignCoordsets(refined)
   ....:     rmsd = calcRMSD(refined)
   ....:     rmsd_mean.append(rmsd.sum() / (len(rmsd) - 1))
   ....:

In [23]: bar(arange(1, len(rmsd_mean) + 1), rmsd_mean);

In [24]: xlabel('Conformation index');

In [25]: ylabel('Mean RMSD');
```

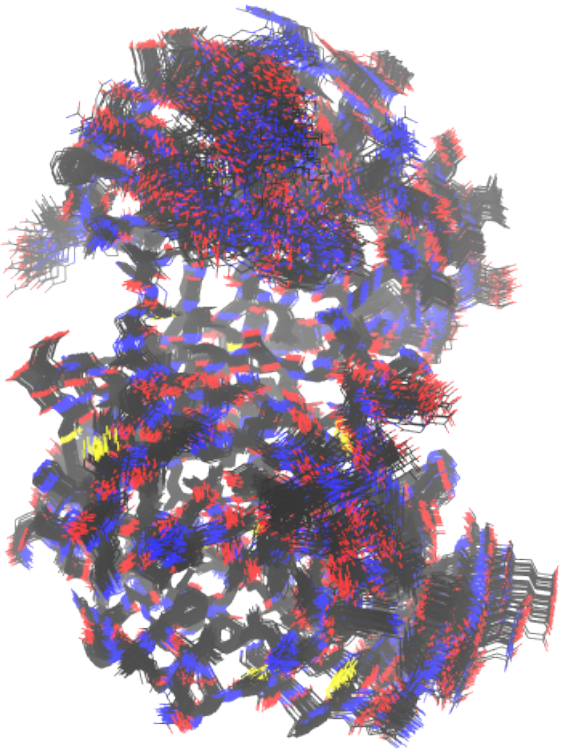


Let's select conformations that are 1.2 Å away from other on average:

```
In [26]: selected = (array(rmsd_mean) >= 1.2).nonzero()[0]

In [27]: selected
Out[27]: array([], dtype=int64)

In [28]: selection = refined[selected]
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-28-691a562240aa> in <module>()
----> 1 selection = refined[selected]

/home/exx/ProDy-website/ProDy/prody/atomic/atomgroup.pyc in __getitem__(self, index)
    208         elif isinstance(index, (list, np.ndarray)):
    209             unique = np.unique(index)
--> 210             if unique[0] < 0 or unique[-1] >= self._n_atoms:
    211                 raise IndexError('index out of range')
    212             return Selection(self, unique, 'index ' + rangeString(index),

IndexError: index 0 is out of bounds for axis 0 with size 0

In [29]: selection
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-29-ae784f1d4d09> in <module>()
----> 1 selection

NameError: name 'selection' is not defined
```

5.4 Visualization

When you visualize the refined ensemble, you should see something similar to this:



Acknowledgments

Continued development of Protein Dynamics Software *ProDy* and associated programs is partially supported by the NIH[8]-funded Biomedical Technology and Research Center (BTRC) on *High Performance Computing for Multiscale Modeling of Biological Systems* (MMBios[9]) (P41 GM103712).

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