MSMBuilder 2.8 Tutorial

Kyle Beauchamp TJ Lane Greg Bowman
Robert McGibbon Christian Schwantes Vijay Pande

January 5, 2014

1 Table of Contents

2 Installation

2.1 General Prerequisites

- CPU with SSE3 support
  - Standard on all x86 processors produced after 2005
- Working C compiler
  - GCC 4.2 or later, clang, or MSVC
- Python
- Some familiarity with the command line environment.

2.2 Python Prerequisites

MSMBuilder is written in the python programming language, and uses a variety of tools from the wider scientific python ecosystem, which may need to be installed separately. They include

- MDTraj
- Numpy
- Scipy
- PyTables
- numexpr
- fastcluster (for hierarchical clustering)
- matplotlib (optional for plotting)
- ipython (optional for interactive mode)
- pymol (optional for visualization)

Two companies, Enthought and Continuum Analytics, produce python distributions which bundle many of these packages in with the python interpreter into a single binary installer, available for all major operating systems. These are the Enthought Canopy python distribution and Continuum's Anaconda.

2.3 Install Python and Python Packages

Rather than individually install the many python dependencies, we recommend that you download the Python2.7 version of the Enthought Canopy or Continuum Anaconda, which contain almost all python dependencies required to run MSMBuilder. If you have a 64 bit platform, please use the 64 bit versions, as this will give higher performance.

Note for OSX users: Enthought represents the easiest way to obtain a working Python installation. The OSX system Python install is broken and cannot properly build Python extensions, which are required for MSMBuilder installation. Also, see FAQ question 11 for a known issue with OSX Lion and OpenMP.

Note: if you are unable to use Canopy or Anaconda, there are other pre-compiled Python distributions available, although they might not be as fast as Enthought. Options include Python(x,y) and the Scipy Superpack (OSX). Finally, most Linux users can install most prerequisites using their package manager. In Ubuntu, the following will install most of the prerequisites:

```
$ sudo apt-get install libhdf5-serial-dev python-dev python-numpy \
python-scipy python-setuptools python-nose python-tables \
python-matplotlib python-yaml swig ipython
```

Neither Canopy nor Anaconda include MDTraj, a library for loading and manipulating molecular dynamics trajectories. For details on MDTraj, visit its documentation at `http://rmcgibbo.github.io/mdtraj/`. Once you've installed python, the remaining dependencies can be installed using python's package manager, `pip`.

```
$ pip install -r requirements.txt
```

2.4 Download and Install MSMBuilder

Download MSMBuilder, unzip, move to the msmbuilder directory. Install using setup.py:

```
$ python setup.py install
```

You may need root privileges during the install step; alternatively, you can specify an alternative install path via `--prefix=XXX`. If you performed the install step with `--prefix=XXX`, you need to ensure that

1. XXX/bin is included in your PATH
2. XXX/lib/python2.7/site-packages/ is included in your PYTHONPATH

Step (1) ensures that you can run MSMBuilder scripts without specifying their location. Step (2) ensures that your Python can locate the MSMBuilder libraries.

3 MSMBuilder Design and Usage

MSMBuilder has been designed to provide both ease of use and versatility. To facilitate the workflows of both novices and experts, we have designed MSMBuilder with two modes of operations:

1. MSMBuilder is a set of python scripts.

2. MSMBuilder is a library.

Python scripts allow most users to work without writing a single line of Python code. Advanced users can write their own Python scripts using the MSMBuilder library. Using MSMBuilder as a library is further described in the MSMBuilder Scripting Guide; we recommend that users first master the script interface before jumping into the library.

4 Tutorial

4.1 MSM Construction

4.1.1 Overview of MSM Construction

Constructing a Markov State model involves several steps, which are summarized below:

1. Simulate the system of interest.
2. Convert trajectory data to MSMBuilder format.
3. Cluster and assign your data to determine microstates.
4. Construct a microstate MSM
5. Validate microstate MSM
6. Calculate macrostates using PCCA+

4.2 Alanine Dipeptide Tutorial

This section walks users through a complete Markov state model analysis of the Alanine Dipeptide data provided in MSMBuilder.

In the following, we assume that you have properly installed MSMBuilder. We also assume that you unzipped the MSMBuilder source file into directory /msmbuilder/. If you unzipped the file elsewhere, you will need to change the paths accordingly.

Finally, in this tutorial we assume that you have installed pymol for viewing conformations.

4.2.1 Move to tutorial directory, prepare trajectories

```
$ cd Tutorial
```

4.2.2 Create an MSMBuilder Project

```
$ msmb ConvertDataToHDF  -s native.pdb -i XTC
```

4.2.3 Cluster your data

The following command clusters your data using the RMSD metric using a hybrid k-centers k-medoids approach. K-centers clustering continues until the intercluster distance (-d) is 0.045 nm. At that point, 50 iterations (-l) of hybrid k-medoids are performed to refine those clusters.

```
$ msmb Cluster rmsd hybrid -d 0.045 -l 50
```

After clustering, one must assign the data to the clusters. For the clustering settings used above, this happens automatically, so you will not need to run a separate assignment step. For other clustering protocols, you may need to run Assign.py or AssignHierarchical.py after the clustering phase.

The assignments of each conformation are stored as Data/Assignments.h5. The cluster centers are stored as Data/Gens.lh5.

Note that clustering with the RMSD metric requires a list of which atom indices to use during RMSD calculation. This file is typically called AtomIndices.dat and can typically be created using the script CreateAtomIndices.py. Because alanine dipeptide contains non-standard atom names, it cannot be generated automatically; a default AtomIndices.dat has already been placed in the Tutorial directory for your use. Note that AtomIndices.dat uses *zero* based indexing–the first atom in your system has index 0.

4.2.4 Alternative Clustering Protocols

Note that other clustering protocols (e.g. Ward's algorithm) often leads to improved models; see the AdvancedMethods tutorial for details.

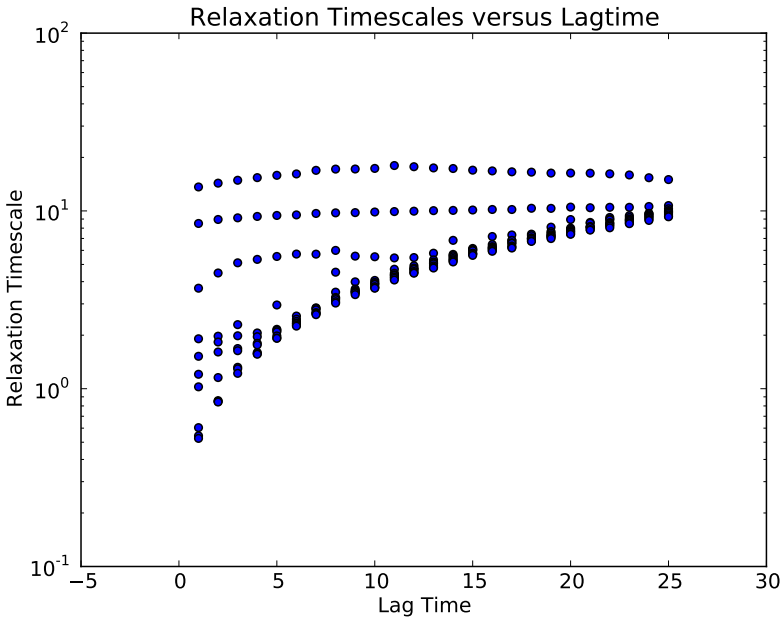
4.2.5 Validate microstate model with relaxation timescales.

We calculate the relaxation timescales for a sequence of lagtimes $\{1, 2, ..., 25\}$:

```
$ msmb CalculateImpliedTimescales -l 1,25 -i 1 -o Data/ImpliedTimescales.dat
```

Next, we use python to plot the results, specifying the lagtime between frames (1 ps):

```
$ msmb PlotImpliedTimescales -d 1. -i Data/ImpliedTimescales.dat
```



4.2.6 Construct MSM at appropriate lagtime

The plotted relaxation timescales suggest that the three slow timescales are reasonable flat at a lagtime of 3 timesteps [ps]. Thus, we construct an MSM using that lagtime:

```
$ msmb BuildMSM -l 3
```

At this point, MSMBuilder has written the following files into your ./Data/ directory:

```
Assignments.Fixed.h5
tCounts.mtx
tProb.mtx
Mapping.dat
Populations.dat
```

Assignments.Fixed.h5 contains a "fixed" version of your microstate assignments that has removed all data that is trimmed the maximal ergodic subgraph of your data.

tCounts.mtx contains the maximum likelihood estimated reversible count matrix. This is a symmetric matrix.

tProb.mtx contains the maximum likelihood estimated transition probability matrix.

Mapping.dat contains a mapping of the original microstate numbering to the "fixed" microstate numbering. This is necessary because some states may have been discarded during the ergodic trimming step.

Populations.dat contains the maximum likelihood estimated reversible equilibrium populations.

4.2.7 Construct a Macrostate MSM

Spectral clustering methods such as PCCA+ [6, 7, 9] can be used to construct metastable models with a minimal number of states. You may also be interested in trying a Bayesian method called BACE [4] that appears to outperform existing spectral methods. However, for the purpose of this Tutorial, we will focus on the more standard spectral methods.

First, we need to construct a microstate model with a short lagtime. The short lagtime is necessary because PCCA+ tries to create macrostates that are long-lived, or metastable. At long lagtimes, states become less and less metastable.

```
$ msmb BuildMSM -l 1 -o L1
```

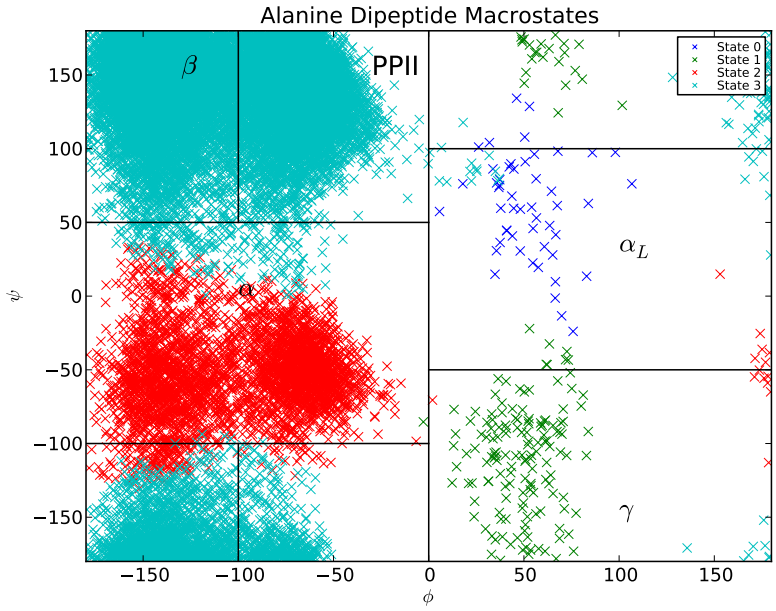
Our previous examination of the relaxation timescales suggested that there were 3 slow processes, so we choose to build a model with 4 macroscopic states.

```
$ msmb PCCA -n 4 -a L1/Assignments.Fixed.h5 -t L1/tProb.mtx -o Macro4/ -A PCCA+
```

4.2.8 Examining the macrostate decomposition

It is known that the relevant degrees of freedom for alanine dipeptide are the phi and psi backbone angles. Thus, it is useful to examine (phi,psi). This data has been pre-calculated and is stored in Dihedrals.h5, or you can compute it via

```
$ python GetDihedrals.py --pdb native.pdb -a Macro4/MacroAssignments.h5 -n 1000
```



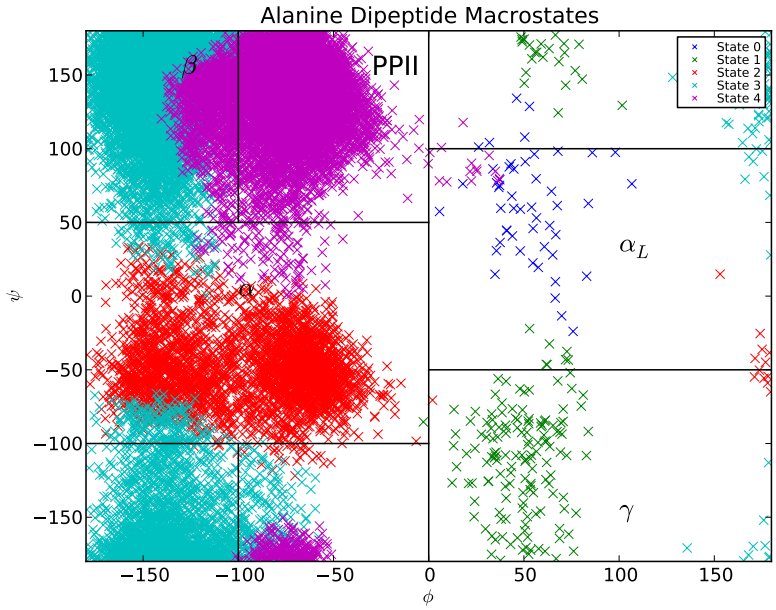
This will sample up to 1000 conformations from each macrostate. To sample all macrostates, use -n -1. We then visualize the data.

```
$ python PlotDihedrals.py Dihedrals.h5
```

You should see something like the following graph (our clustering and PCCA+ code both perform randomized searches, so your plot may appear slightly different):

Thus, the PCCA algorithm has automatically identified the key basins of alanine dipeptide. The black lines correspond to the $\beta, PP_{II}, \alpha_R, \alpha_L$ and γ conformational basins, as estimated previously [8]. If we want a model that is less coarse grained, we can build a macrostate MSM with more states. If, for example, we had used 5 states, we would produce a Ramachandran plot that also captures the barrier between the β and PP_{II} basins.

In general, PCCA and PCCA+ are best applied to capturing long-lived, metastable states. Thus, for this system, applying PCCA+ to construct models with more than 5 states may not produce useful models.



This is because alanine dipeptide only contains four eigenvalues that are significantly slower than the time resolution of 1 ps.

4.2.9 Calculate Macrostate Implied Timescales

```
$ msmb CalculateImpliedTimescales -l 1,25 -i 1 \
-o Macro4/ImpliedTimescales.dat -a Macro4/MacroAssignments.h5 -e 3

$ msmb PlotImpliedTimescales -i Macro4/ImpliedTimescales.dat -d 1
```

Occasionally, PCCA+ will lead to poor macrostates, so it is important to verify that:

1. The state decomposition makes physical sense
2. The macrostate implied timescales make sense
3. The macrostate implied timescales "follow" the microstate implied timescales

Furthermore, PCCA+ is best used to estimate metastable states. Here are some additional guidelines for achieving good success with PCCA+:

1. If your microstate model has too *long* of a lagtime, the model may not be metastable because significant dynamics occurs on the timescale of a single lagtime.

2. If your microstate model has too *short* of a lagtime, the microstate model may not be Markovian, leading to errors when estimating the eigenvalues and eigenvectors. Most importantly, significant non-Markovian dynamics can cause the slowest eigenvalues to be misidentified. If this occurs, your PCCA+ model will be worthless! To prevent this, a useful guide is to make sure that the slowest implied timescales do not cross one another (e.g. their rank ordering is constant).

3. If your microstate model has too *few* states, your microstate model may not be sufficiently Markovian. You may not have sufficient geometric resolution to accurately identify the primary kinetic barriers.

4. If your microstate model has too *many* states, your microstate model will have poor statistics, possibly leading to poor estimates of the slow eigenvectors.

Thus, success with PCCA+ may require some trial and error when selecting the appropriate lagtime and microstate clustering. Finally, note that our implementation of PCCA+ uses a simulated annealing minimization. This randomized search means that you may find multiple minima by repeating the PCCA+ calculation several times. You may find a better model by repeating the calculation several times.

4.2.10 Visualizing Structures with Pymol

Because macrostate models typically have a handful of states, it is easy for humans to compare the resulting structures visually. One way to do this is to save randomly selected conformations from each state, then view them in Pymol (or VMD):

```
$ msmb SaveStructures -s -1 -f pdb -S sep -a Macro4/MacroAssignments.h5
pymol PDBs/State0-0.pdb PDBs/State1-0.pdb PDBs/State2-0.pdb PDBs/State3-0.pdb
```

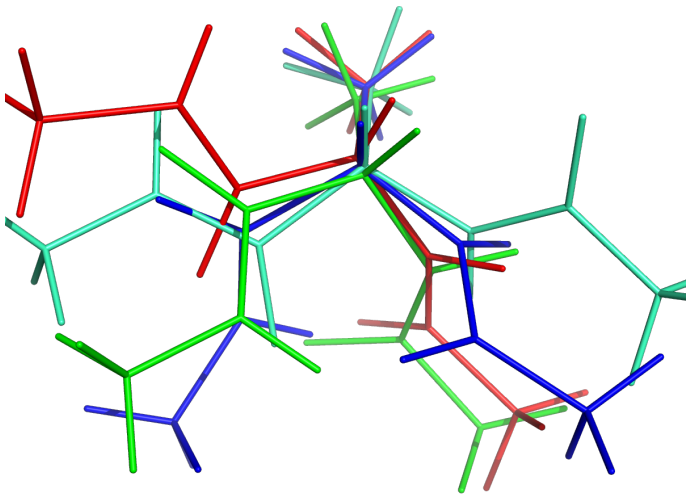


Figure 1: Randomly selected conformations from the four macrostate model. Colored by macrostate.

4.3 MSMBuilder Commands

The MSMBuilder commands are listed below. Each command corresponds to a single scrip that can be called either through `msmb` or on its own. Each command also provides instructions by running with the -h flag (e.g. `msmb Cluster -h` or `Cluster.py -h`). Note that the installer (setup.py) should have installed each script below to someplace in your PATH.

4.3.1 ConvertDataToHDF.py

Merges sequences of XTC or DCD files into HDF5 files that MSMB2 can read quickly. Takes data from a directory of trajectory directories or a FAH-style filesystem.

4.3.2 CreateAtomIndices.py

Selects atom indices you care about and dumps them into a flat text file. Can select all non-symmetric atoms, all heavy atoms, all alpha carbons, or all atoms.

4.3.3 Cluster.py

Cluster your data using your choice of clustering algorithm and distance metric. We have previously used several clustering protocols, which are summarized:

1. RMSD + k-centers clustering [3, 5] ("Cluster.py rmsd kcenters")
2. RMSD + hybrid k-centers / k-medoids clustering [1] ("Cluster.py rmsd hybrid")
3. RMSD + Ward clustering [2] ("Cluster.py rmsd hierarchical")

Note that Ward clustering calculates an $O(N^2)$ distance matrix, which may be prohibitive for datasets with many conformations.

Most of our experience has been in applying MSMBuilder to protein folding. Thus, non-folding applications may require a slightly different protocol.

4.3.4 Assign.py / AssignHierarchical.py

Assign.py assigns data to the cluster generators calculated using the k-centers or hybrid algorithms.

AssignHierarchical.py assigns data using the output of a hierarchical clustering algorithm such as Ward. The key difference is that a single hierarchical clustering allows construction of models with any number of states.

4.3.5 CalculateImpliedTimescales.py

Calculates the implied timescales for a python range of MSM lag times. This allows you to validate whether a given model is Markovian. Notes:

1. You might get a SparseEfficiencyWarning for every lag time. Ignore this.

2. Lagtimes are input in units of the time spacing between successive trajectory frames. If your trajectories are stored every 10 ns, then -l 1,4 estimates implied timescales with lagtimes 10, 20, 30, 40 ns.

4.3.6 PlotImpliedTimescales.py

A template for generating an implied timescales plot.

4.3.7 BuildMSM.py

Estimate a reversible transition and count matrix using a two step process:

1. Use Tarjan algorithm to find the maximal strongly-connected (ergodic) subgraph

2. Use likelihood maximization to estimate a reversible count matrix consistent with your data

This script also outputs the equilibrium populations of the resulting model, as well as a mapping from the original states to the final (ergodic) states.

4.3.8 GetRandomConfs.py

Selects random conformations from each state of your MSM. This is very useful for efficient calculation of observables.

4.3.9 CalculateClusterRadii.py

Calculates the mean RMSD of all assigned snapshots to their cluster generator for each cluster. Gives an indication of how structurally diverse clusters are.

4.3.10 CalculateRMSD.py

Calculate the RMSD between a PDB and a trajectory (or set of cluster centers). Useful for deciding which clusters belong to the folded, unfolded, or transition state ensembles (or any other grouping!)

4.3.11 CalculateProjectRMSD.py

Calculates the RMSD of all conformations in a project to a given conformation.

4.3.12 CalculateTPT.py

Performs Transition Path Theory (TPT) calculations. You will need to define good starting (reactants/U) and ending (products/F) ensembles for this script. Writes the forward and backward committors and the net flux matrix

4.3.13 SavePDBs.py

Allows you to sample random PDBs from particular states and save them to disk.

4.3.14 PCCA.py

Lumps microstates into macrostates using PCCA [6] or PCCA+ [7, 9]. This script generates a macrostate assignments file from a microstate model.

Notes:

1. We recommend PCCA+ for most applications
2. PCCA+ requires a reversible MSM as input
3. You can discard eigenvectors based on their equilibrium flux (fPCCA+).

4.3.15 BACE_Coarse_Graining.py

An alternative method for lumping microstates into macrostates using a Bayesian approach (Bayesian agglomerative clustering engine) [4]. This is an attractive option as it appears to outperform existing spectral methods. To learn how to use BACE, run the script with the -h option.

5 Frequently Asked Questions

Q1. How do I decrease / increase the number of threads used during clustering, assignment, and rmsd calculation?

A1. Set the

```
OMP_NUM_THREADS
```

environment variable to the desired number of threads. In linux, you would type (or add to your bashrc):

```
export OMP_NUM_THREADS=6
```

Q2: I see the following error. What do I do?

```
#004: H5Z.c line 1095 in H5Z_pipeline(): required filter is not registered
```

A2: You are trying to read an HDF5 file that was written using a different PyTables installation. Your current PyTables installation is likely missing the compression algorithm (filter) required to read the file. The solution is to find a version of Pytables that has the old compression algorithm (filter) and use MSMBuilder to read and then re-write the trajectories (by default, MSMBuilder uses the PyTables BLOSC compression). To do this (for a single File), you would so something like:

```
from msmbuilder import Trajectory
R1 = Trajectory.LoadFromLHDF(Filename)
R1.SaveToLHDF(NewFilename)
```

Q3: I received an "Illegal instruction" error. What does this mean?

A3: MSMBuilder2 requires an SSE3 compatible processor when Clustering and calculating RMSDs. Any processor built after 2006 should have the necessary instructions.

Q4: In my implied timescales plot, I see unphysically slow timescales.

A4: The current estimators for transition matrices are somewhat sensitive to poor statistics. The hybrid k-centers / k-kmedoids clustering focuses on providing the best possible clustering–without regard to the quality of the resulting statistics. Thus, to get more precise timescales, you may have to find a way to achieve better statistics. Here are a few ideas:

1. Collect longer trajectories.

2. Use fewer states. Also, by increasing the number of local and global k-medoid updates, you can often increase the accuracy of your clustering while simultaneously lowering the number of states.

3. Subsample your data when clustering.

4. Skip the initial k-centers step of clustering, instead using randomly selected conformations. This generally leads to poorer clustering quality, but considerably better statistics in each state. (Thus, the clusters will be much more localized to regions of high population density.) This can be achieved by setting "-r 0" when clustering.

5. Use Ward clustering

Q5. Why are there -1s in my Assignments matrix?

A5. We use -1 as a "padding" element in Assignment matrices. Suppose your project has maximum trajectory length of 100. If trajectory 0 has length 50, then A[0,50:] should be a vector of -1. Furthermore, when you perform trimming to ensure (strong) ergodicity, futher -1s could be introduced at the start or finish of the trajectory. Finally, if Ergodic trimming was performed with count matrices estimated using a sliding window, you could even see something like: -1 -1 -1 x -1 -1 y z ... This is because sliding window essentially splits your trajectory into independent subtrajectories–one for each possible window starting position. "x" then marks the start of one of these subtrajectories.

Q6. When building MSMBuilder, I see the following:

```
 ********************************************************************************
WARNING: The C extension 'LPRMSD' could not be compiled.
This may be due to a failure to find the BLAS libraries
********************************************************************************
```

What should I do?

A6. Don't worry. This module is not used by any of the standard MSMBuilder features.

Q7. What is the difference between PCCA+ and FPCCA+?

A7. FPCCA+ is PCCA+ with a different choice of eigenvectors to model. In particular, FPCCA+ uses a criterion based on both timescale *and* eigenvector flux.

Q8. Should I use FPCCA+ or PCCA+?

A8. First, note that FPCCA+ is more "lossy" or "coarse-grained" than PCCA+. By discarding slow but high-flux eigenvectors, you are losing some information from your microstate model. Essentially, the choice between FPCCA+ and PCCA+ depends on how much you weight model accuracy versus model simplicity.

Q9. I see warnings when using PCCA+:

```
ComplexWarning: Casting complex values to real discards the imaginary part
RuntimeWarning: invalid value encountered in cdouble_scalars
Warning: constraint violation detected.
f = nan
```

A9. This is probably due to PCCA+ finding a "degenerate" state decomposition, where one of your macrostates is empty. Usually, the minimization procedure should eventually find a feasible point with the correct number of states. Be sure to check that your resulting state decomposition makes sense.

Q10. How do I make an MSM movie?

A10. To build a movie, you just Sample states from the model (MSMLib.Sample). Then you sample conformations from each state (Project . GetRandomConfsFromState). Then you append each frame to a PDB file (Conformation.SavePDB or Trajectory.SavePDB). After you have the PDBs, you can use either VMD or pymol for movie making.

Q11. On an OSX Lion Machine, clustering fails with an "Abort Trap" error message. What do I do?

A11. This is due to a known bug in OSX Lion's support for OpenMP (see `https://discussions.apple.com/thread/3786045?start=0&tstart=0`). As a workaround, you can simply

```
export OMP_NUM_THREADS=1
```

to disable OpenMP support during clustering. This should eliminate the problem, but it limits you to single core clustering.

References

[1] K.A. Beauchamp, G.R. Bowman, T.J. Lane, L. Maibaum, I.S. Haque, and V.S. Pande. Msmbuilder2: Modeling conformational dynamics at the picosecond to millisecond scale. *J. Chem. Theory Comput.*, 7(10):3412–3419, 2011.

[2] K.A. Beauchamp, R. McGibbon, Y.S. Lin, and V.S. Pande. Simple few-state models reveal hidden complexity in protein folding. *Proceedings of the National Academy of Sciences*, 2012.

[3] G. R. Bowman, K. A. Beauchamp, G. Boxer, and V. S. Pande. Progress and challenges in the automated construction of Markov state models for full protein systems. *J. Chem. Phys.*, 131:124101, Sep 2009.

[4] G.R. Bowman. Improved coarse-graining of markov state models via explicit consideration of statistical uncertainty. *J. Chem. Phys.*, 137:134111, 2012.

[5] G.R. Bowman, X. Huang, and V.S. Pande. Using generalized ensemble simulations and markov state models to identify conformational states. *Methods*, 49(2):197–201, 2009.

[6] P. Deuflhard, W. Huisinga, A. Fischer, and C. Schütte. Identification of almost invariant aggregates in reversible nearly uncoupled markov chains. *Linear Algebra Appl.*, 315(1-3):39–59, 2000.

[7] P. Deuflhard and M. Weber. Robust perron cluster analysis in conformation dynamics. *Linear Algebra Appl.*, 398:161–184, 2005.

[8] A.K. Jha, A. Colubri, M.H. Zaman, S. Koide, T.R. Sosnick, and K.F. Freed. Helix, sheet, and polyproline ii frequencies and strong nearest neighbor effects in a restricted coil library. *Biochemistry*, 44(28):9691–9702, 2005.

[9] S. Kube and M. Weber. A coarse graining method for the identification of transition rates between molecular conformations. *J. Chem. Phys.*, 126:024103, 2007.