MSMBuilder 2.5 Tutorial

Kyle Beauchamp TJ Lane Greg Bowman
Robert McGibbon Vijay Pande

April 27, 2012

1 Table of Contents

2 Installation

2.1 Prerequisites

- CPU with SSE3 support.
- GCC 4.2 or later (with OpenMP support)
- Python
- Numpy
- Scipy
- PyTables
- numexpr
- fastcluster (for hierarchical clustering)
- matplotlib (optional for plotting)
- ipython (optional for interactive mode)
- pymol (optional for visualization)

Note that the Enthought Python Distribution contains most of these prerequisites and provides an easy way to facilitate installation.

2.2 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 Python Distribution, which contains almost all (except fastcluster) python dependencies required to run MSMBuilder. Please use the 64 bit versions, as this will give higher performance in our RMSD code.

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.

Note: please use a 64 bit version of python (and thus of MSMBuilder). On certain systems (OSX Lion), 32 bit builds have experienced crashes in the RMSD / Clustering code.

2.3 Download and Install MSMBuilder

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

```
python setup.py build
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 your 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 Library 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+
7. Calculate a macrostate rate matrix using SCRE
8. Validate the resulting model.

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 $PREFIX/share/msmbuilder_tutorial
tar -xvf XTC.tar
```

Here, $PREFIX$ is the base directory of your msmbuilder installation.

4.2.2 Create an MSMBuilder Project

```
ConvertDataToHDF.py  -s native.pdb -i XTC
```

4.2.3 Cluster your data using Ward's algorithm and the RMSD metric

WARNING: this step requires a minimum of 3.0 GB of memory for storing a matrix of pairwise RMSD.

```
Cluster.py rmsd hierarchical
```

4.2.4 Assign data to states

```
AssignHierarchical.py -H Data/Zmatrix.h5 -n 200
```

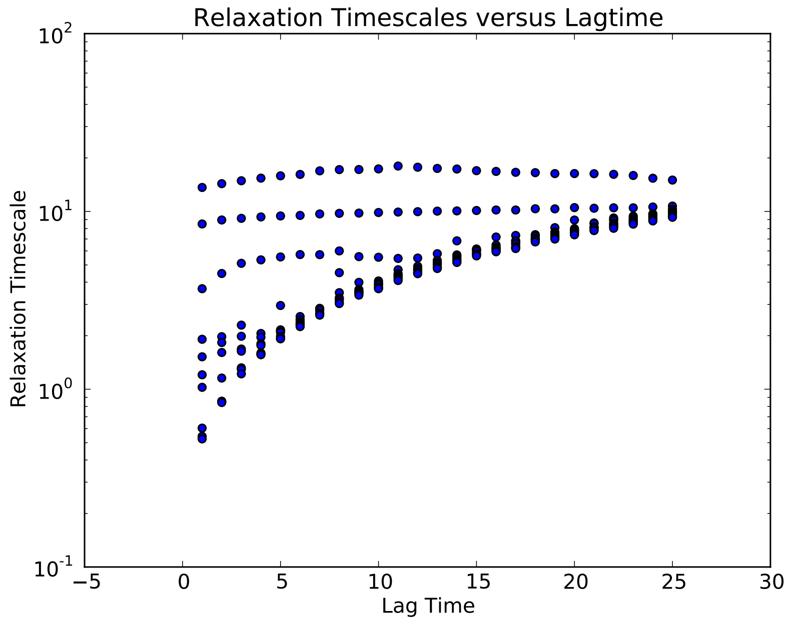
4.2.5 Validate microstate model with relaxation timescales.

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

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

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

```
PlotImpliedTimescales.py -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:

```
BuildMSM.py -l 3
```

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

```
Assignments.Fixed.h5
tCounts.UnSym.mtx
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.UnSym.mtx contains the raw counts of the ergodic data. These counts may not be reversible (thus, the count matrix may not be symmetric).

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 cluster methods such as PCCA+ [4, 5, 7] can be used to construct metastable models with a minimal number of states. 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.

```
 BuildMSM.py -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.

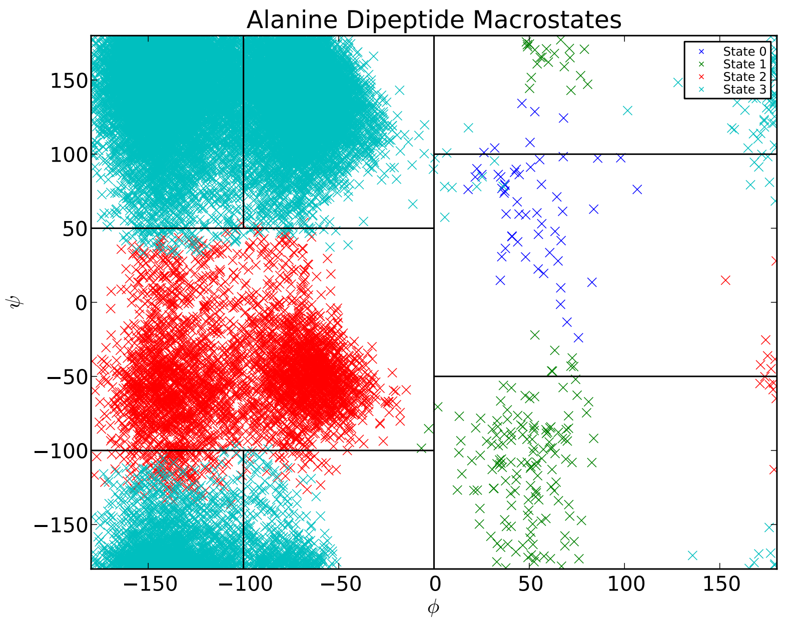
```
PCCA.py -n 4 -a L1/Assignments.Fixed.h5 -t L1/tProb.mtx -o Macro4/ -A PCCA+
```

4.2.8 Examining the macrostate decomposition

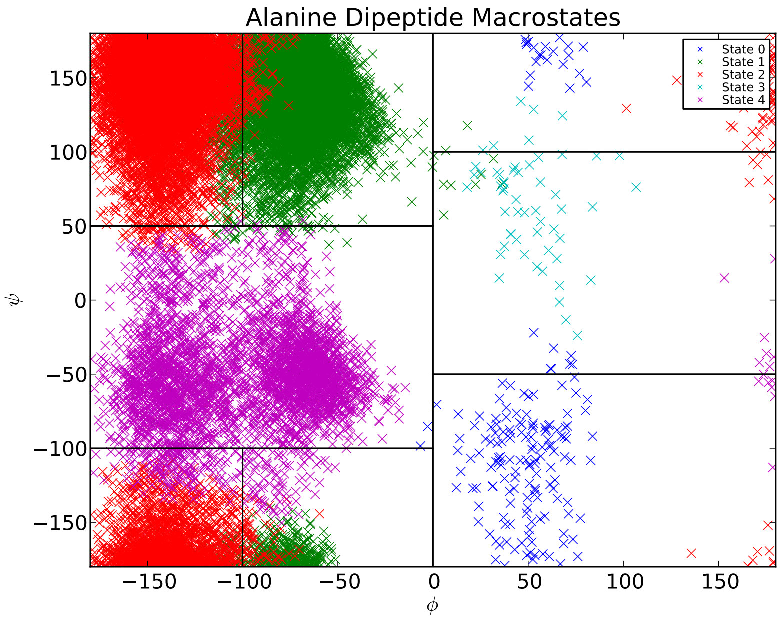
It is known that the relevant degrees of freedom for alanine dipeptide are the phi and psi backbone angles. Thus, it is useful to examine (phi,psi), which we have pre-calculated for you.

```
python PlotDihedrals.py Macro4/MacroAssignments.h5
```

You should see the following graph:



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 [6]. 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

```
CalculateImpliedTimescales.py -l 1,25 -i 1 -o Macro4/ImpliedTimescales.dat -a Macr
```

```
PlotImpliedTimescales.py -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 Estimate a converged rate matrix using SCRE

In the macrostate implied timescales, the slowest implied timescale converges at approximately 10 ps, while the third timescale converges as 5 ps. Furthermore, the third timescale is aliased at lagtimes starting with 14 ps; lagtimes longer than 14 ps will not accurately capture the dynamics of this relaxation. This suggests that this system might benefit from estimating rates using SCRE and multiple lagtimes.

To build an SCRE model, we first construct a macrostate transition matrix with lagtime 1.

```
BuildMSM.py -l 1 -a Macro4/MacroAssignments.h5 -o Macro4/
```

The key idea in SCRE is to estimate rate matrix $K_{ij}(\tau)$ for a variety of lagtimes. Then, each rate matrix element is fixed when $K_{ij}(\tau)$ becomes approximately constant with respect to the lagtime τ. The final rate matrix provides estimates of Markovian rates for each of the rate elements K_{ij}.

Next, we run an interactive script to estimate rates using SCRE.

```
Interactive-SCRE.py -a Macro4/Assignments.Fixed.h5 -o Macro4
```

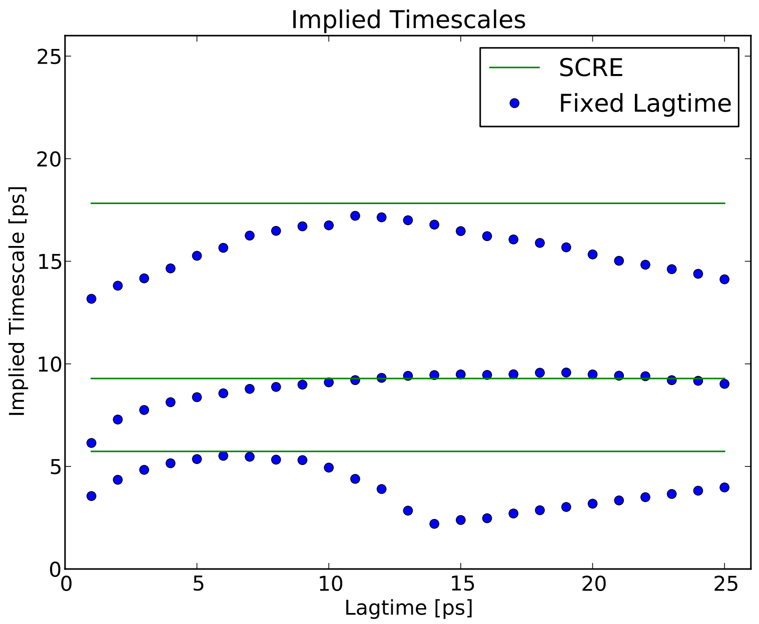
In practice, the script will prompt the user to select a maximum lagtime. The script then estimates and plots rate matrices for each lagtime up to the maximum value. The user is then asked to identify converged rate matrix elements. In general, one should try to estimate the quickly-convergign rates first. After those rates are constrained, one can then try to estimate the slower and / or less Markovian rates. For concreteness, I used the following inputs:

25 1,0,6 25 3,2,19 25 3,1,6 25 3,0,9 25 14

Note that on the last iteration, the user is asked only to select a lagtime. Also note that your choices may be different because of differences in the macrostate definitions; the PCCA+ minimization uses a non-deterministic simulated annealing. Also, if your state decomposition is insufficiently Markovian, you may find that the plotted rate estimates show a "periodic" behavior. This likely indicates that a given macrostate contains multiple slowly interconverting substates; as you increase the lagtime, the model switches from the faster process to the slower process. In such situations, your best bet is to increase the number of macrostates.

Finally, we compare the fixed-lagtime and SCRE estimates of implied timescales:

```
python PlotMacrostateImpliedTimescales.py
```



We find that the SCRE estimates capture the “converged” values of the fixed-lagtime timescales. Achieving convergence for each timescale is not possible with a fixed-lagtime MSM, because the 20 ps lagtime required for the second relaxation causes the third relaxation to alias.

4.3 MSMBuilder Scripts

The MSMBuilder scripts are listed below. Each script below provides instructions by running with the -h flag. 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.

Note: if you wish to convert a pre-existing MSMBuilder1 project, use UpdateProjectToHDF5.py

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

We recommend Ward clustering if it is computationally feasible; otherwise, use the hybrid k-centers / k-medoids clustering. Ward clustering calculations 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 estimtes 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 DoTPT.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 [4] or PCCA+ [5, 7]. 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+).

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.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: A. Collect longer trajectories. B. 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. C. Subsample your data when clustering. D. 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.

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.

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] 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.

[3] 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.

[4] 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.

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

[6] 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.

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