MSMBuilder Documentation

Release 2.8

MSMBuilder Team

January 06, 2014

Contents

1 Documentation **3**
- 1.1 Installation . . . 3
- 1.2 Tutorial : Introduction and Alanine Dipeptide . . . 4
- 1.3 Tutorial : Additional Methods . . . 10
- 1.4 MSMBuilder Commands . . . 13
- 1.5 Frequently Asked Questions . . . 17
- 1.6 What's New . . . 19
- 1.7 Contributing . . . 20

2 MSM Theory Notes **25**
- 2.1 Time-Structure Based Independent Component Analysis (tICA) . . . 25
- 2.2 Calculation of Autocorrelation Functions . . . 28
- 2.3 Maximum-Likelihood Reversible Transition Matrix . . . 29

3 Library API Reference **31**
- 3.1 `msmbuilder.MSMLib` . . . 31
- 3.2 `msmbuilder.msm_analysis` . . . 31
- 3.3 `msmbuilder.assigning` . . . 31
- 3.4 `msmbuilder.clustering` . . . 32
- 3.5 `msmbuilder.drift` . . . 32
- 3.6 `msmbuilder.cfep` . . . 32
- 3.7 `msmbuilder.SCRE` . . . 32
- 3.8 `msmbuilder.reduce.tICA` . . . 32

4 Indices and tables **35**

MSMBuilder is a tools for constructing Markov state models of biomolecular conformational dynamics.

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.

CHAPTER 1

Documentation

1.1 Installation

MSMBuilder should run on most modern computers equiped with a scientific python installation. But, in the interest of being explicit, here the requirements

- A CPU with SSE3 support, which has been tandard on all x86 processors produce after 2006.
- A working C compiler, such as GCC 4.2 or later, clang, or MSVC.
- Python, with some scientific modules installed (see below)

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

1.1.1 Python Prerequisites

- 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.

1.1.2 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 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 nor fastcluster. They can be installed be installed using python's package manager, `pip`.

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

1.1.3 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.

1.2 Tutorial : Introduction and Alanine Dipeptide

1.2.1 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+

1.2.2 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.

Move to tutorial directory, prepare trajectories

Assuming you've just downloaded and installed the package, move to the Tutorial directory.

```
$ cd Tutorial
```

Create an MSMBuilder Project

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

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.

Alternative Clustering Protocols

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

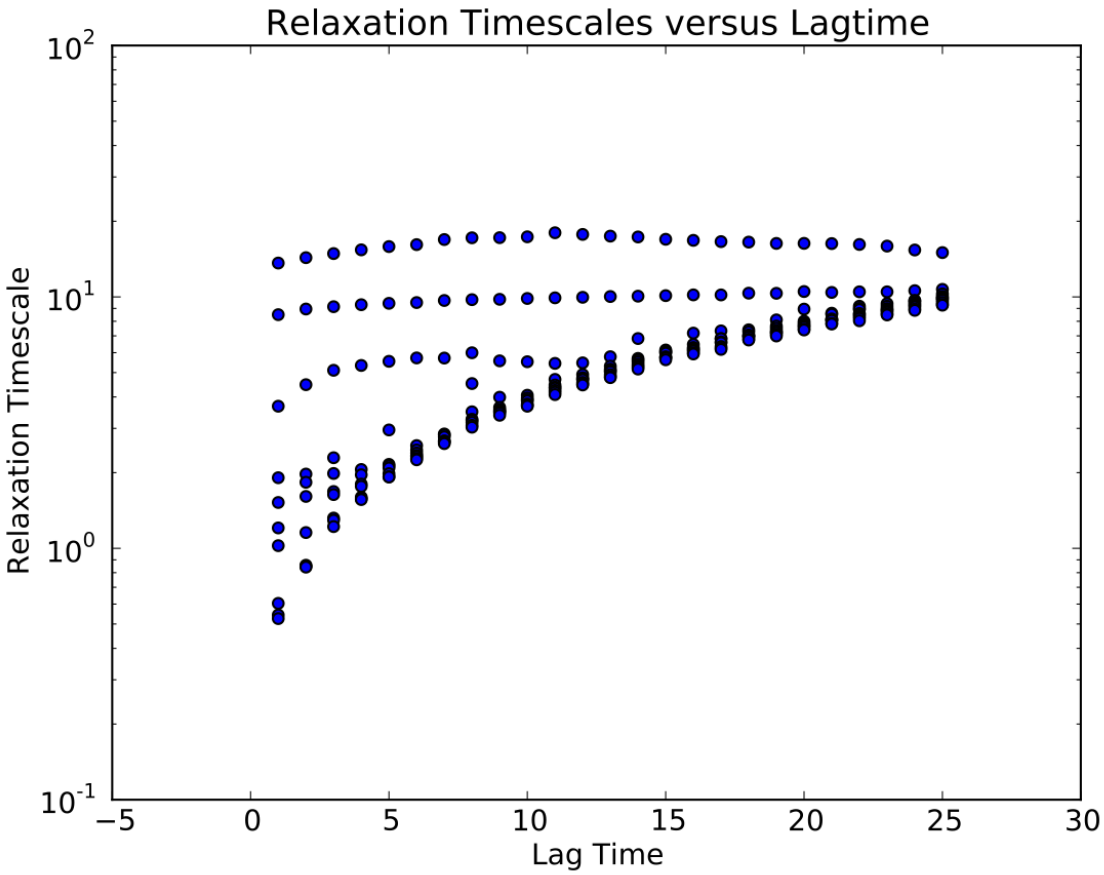
Validate microstate model with relaxation timescales.

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
```


Relaxation Timescales versus Lagtime
Relaxation Timescale
Lag Time
10^2
10^1
10^0
10^{-1}
−5
0
5
10
15
20
25
30

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.

Construct a Macrostate MSM

Spectral clustering methods such as PCCA+ 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 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+
```

Examining the macrostate decomposition

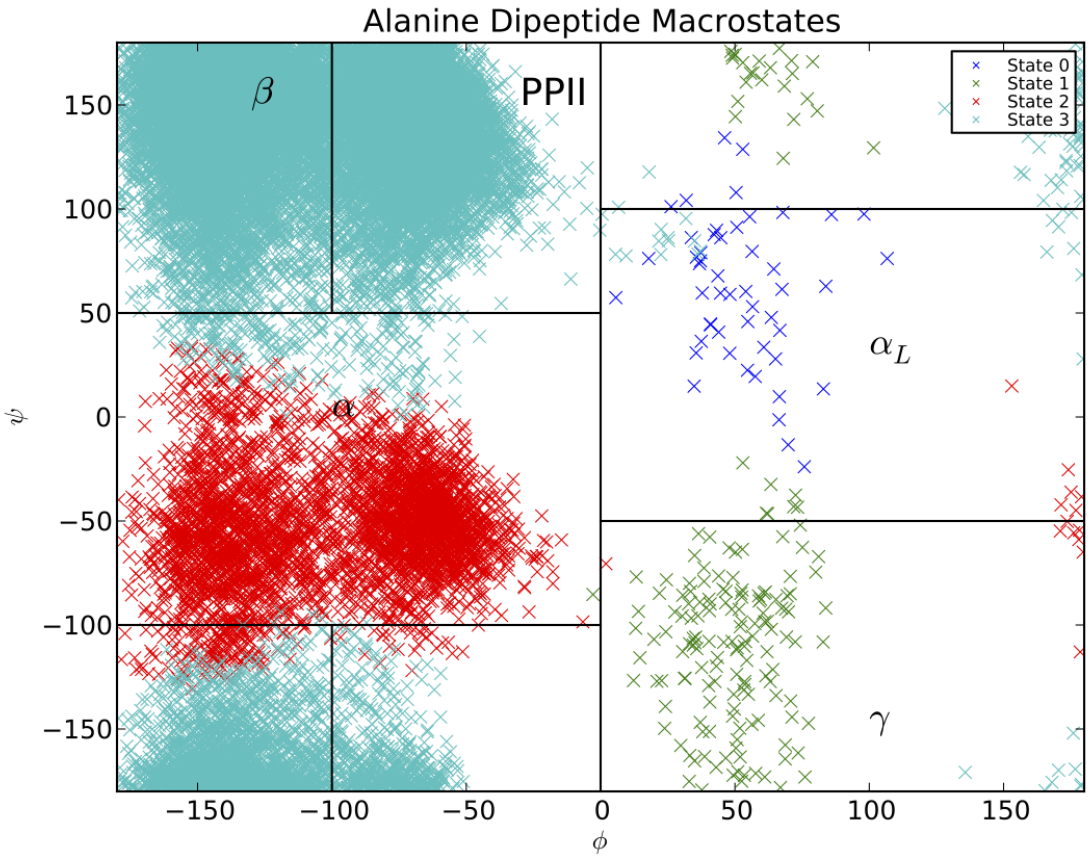
It is known that the relevant degrees of freedom for alanine dipeptide are the phi and psi backbone angles. Thus, it is useful to examine (phi,psi). This data has been pre-calculated and is stored in Dihedrals.h5, or you can compute it via

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

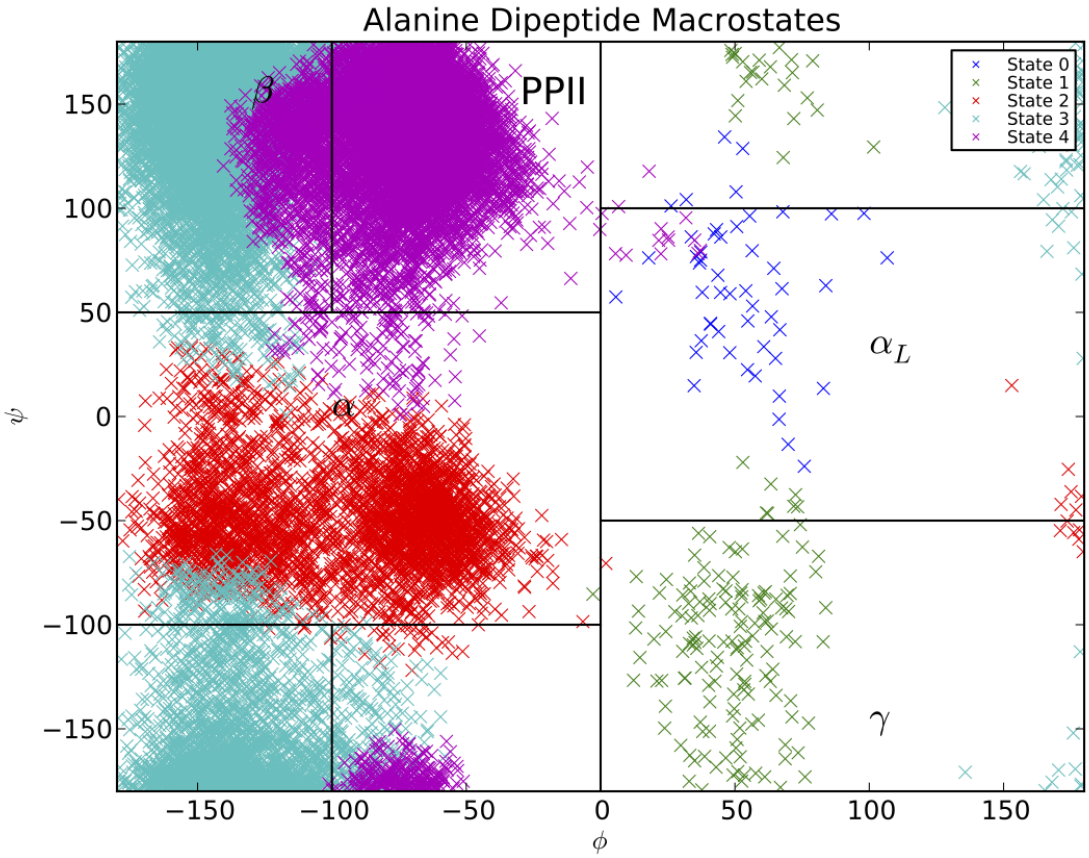
This will sample up to 1000 conformations from each macrostate. To sample all macrostates, use -n -1. We then visualize the data.

```
$ 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):


Alanine Dipeptide Macrostates
State 0
State 1
State 2
State 3
β
PPII
α
α_L
γ
150
100
50
0
−50
−100
−150
ψ
−150
−100
−50
0
50
100
150
ϕ

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 . 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.

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 mis-identified. 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.

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
```

1.3 Tutorial : Additional Methods

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

MSMBuilder provides multiple clustering algorithms. In the 'basic' tutorial, we used a hybrid k-centers k-medoids algorithm. That algorithm is quite fast, but more advanced clustering algorithms can lead to better models. In particular, we have found that Ward clustering provides two key advantages over k-centers :

1. Improved statistics in each state–that is, few states will have "poor" statistics.
2. Ward clustering typically leads to slower, more converged implied timescales. This is due Ward better separating kinetically distinct regions of conformation space.

Note that the key disadvantage of Ward is that it requires large amounts of memory.

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

```
Cluster.py rmsd hierarchical
```

As example, the following figure shows that Ward clustering leads to improved implied timescales, as compared to Hybrid clustering:

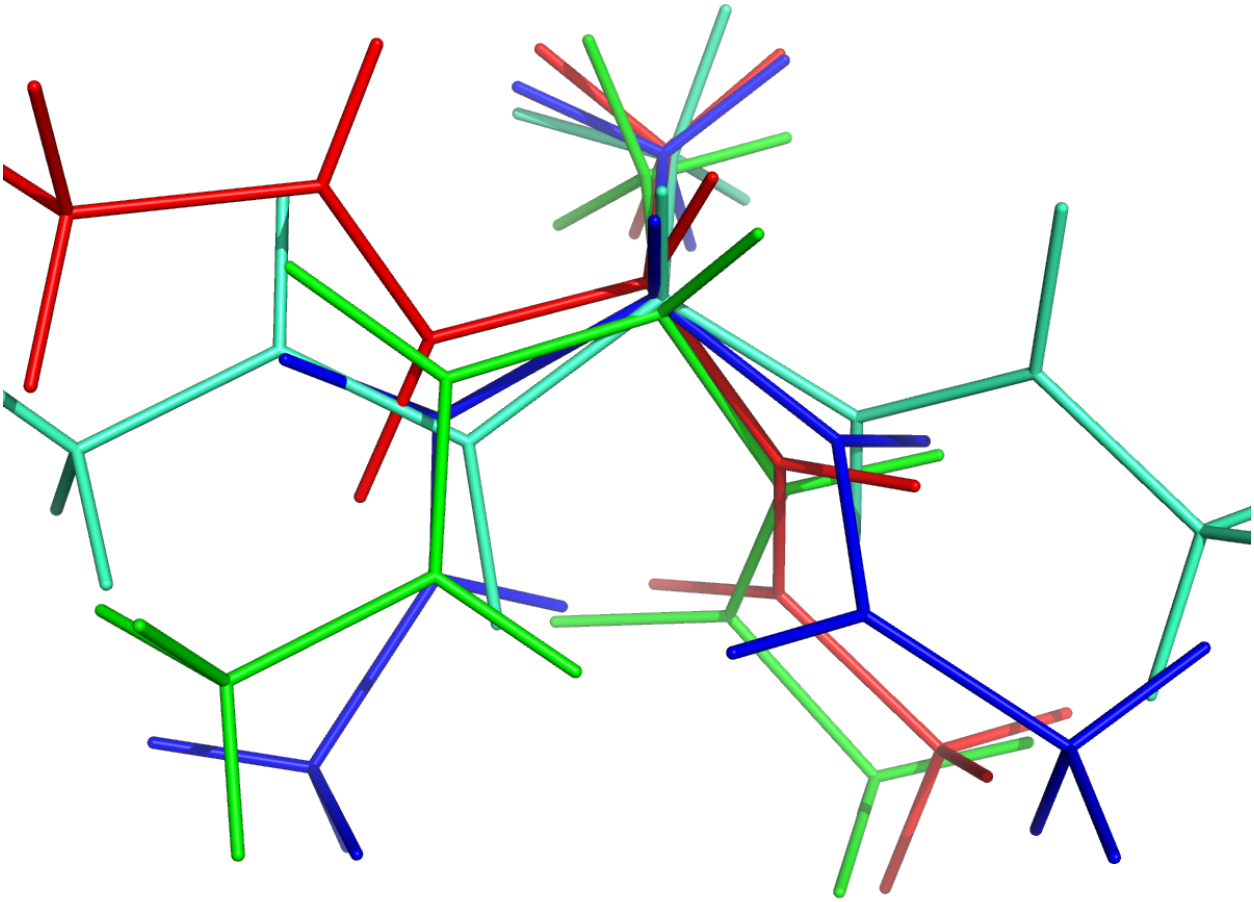
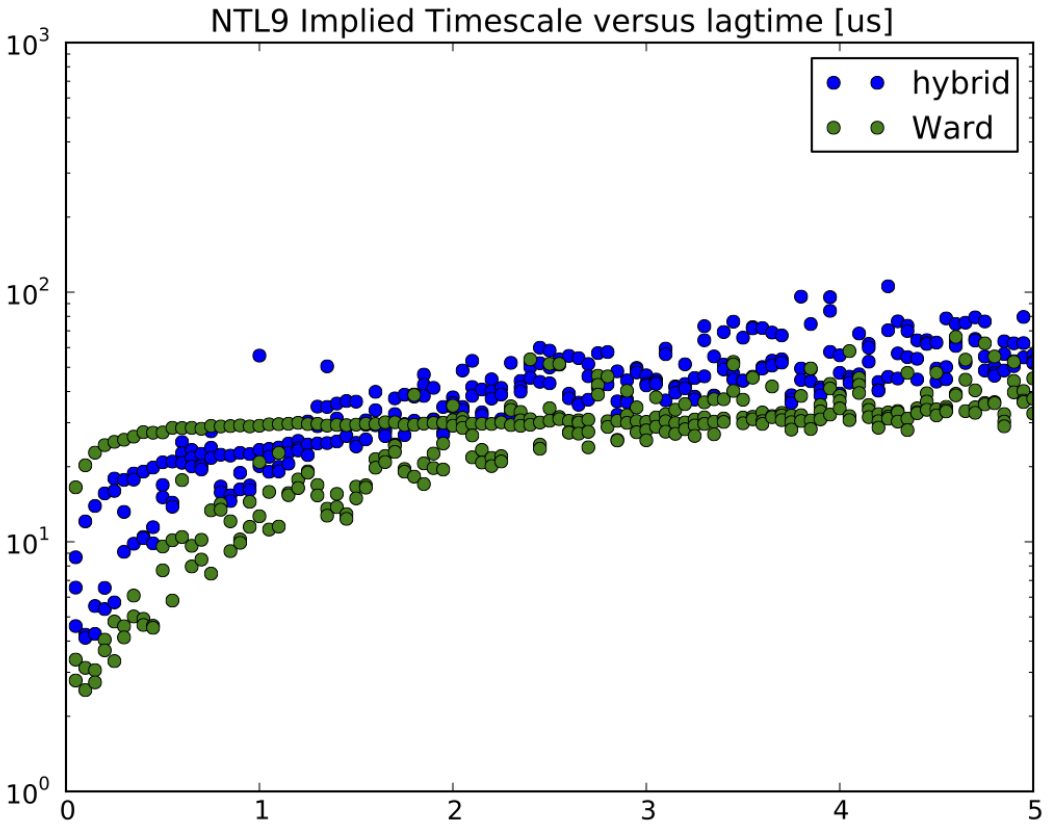


Figure 1.1: Randomly selected conformations from the four macrostate model. Colored by macrostate.


NTL9 Implied Timescale versus lagtime [us]
hybrid
Ward
10^3
10^2
10^1
10^0
0
1
2
3
4
5

1.3.2 Estimate a 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 . Note that SCRE currently is practical only for models with few states ($n \leq 10$).

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
```

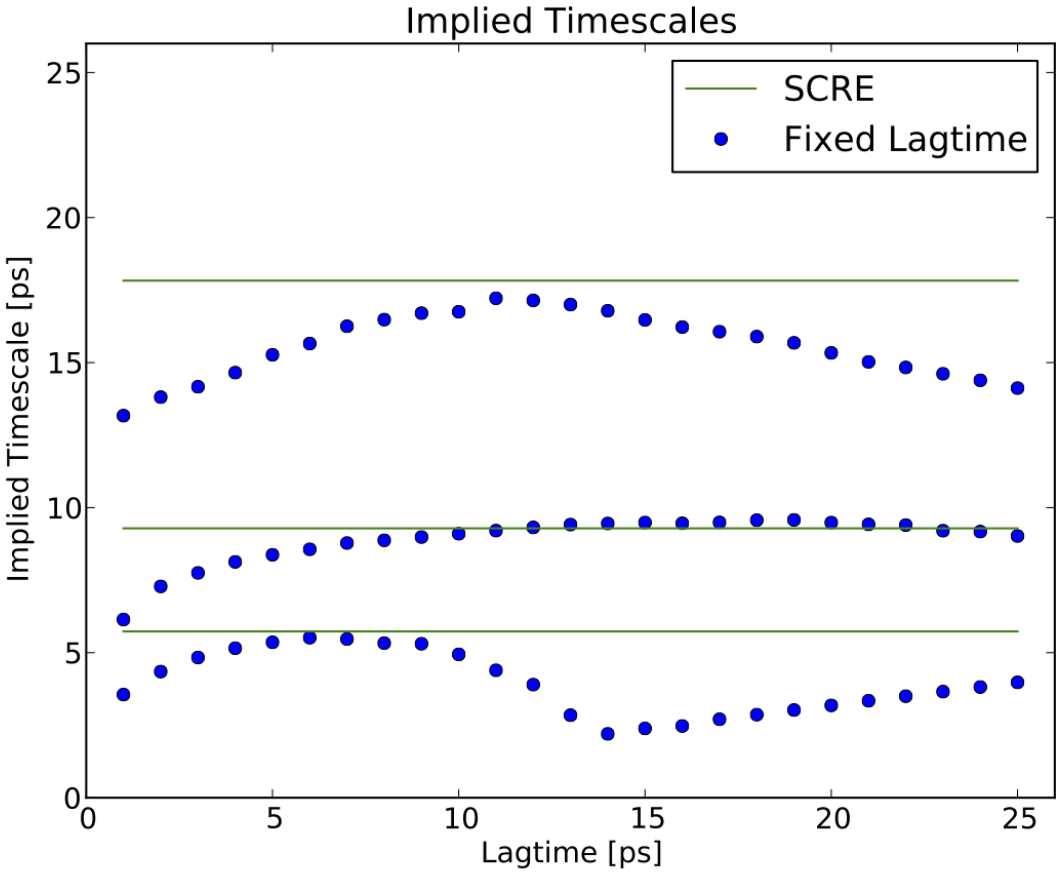
We find that the SCRE estimates capture the "converged" values of the fixed-lagtime timescales. Achieving convergence for each timescale is not possible with a fixed-lagtime MSM, because the 20 ps lagtime required for the second relaxation causes the third relaxation to alias.

1.4 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.

1.4.1 msmb

All of the individual msmbuilder commands can be accessed as subcommands from this script, i.e. `msmb ConvertDataToHDF`. Using `msmb -h`, you can get a list of all of the available msmbuilder commands.


Implied Timescales
SCRE
Fixed Lagtime
Implied Timescale [ps]
Lagtime [ps]
0
5
10
15
20
25

1.4.2 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.

1.4.3 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.

1.4.4 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-centersCluster.py rmsd)
2. RMSD + hybrid k-centers / k-medoidsCluster.py rmsd)
3. RMSD + WardCluster.py rmsd)

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.

1.4.5 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.

1.4.6 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.

1.4.7 PlotImpliedTimescales.py

A template for generating an implied timescales plot.

1.4.8 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.

1.4.9 GetRandomConfs.py

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

1.4.10 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.

1.4.11 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!)

1.4.12 CalculateProjectRMSD.py

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

1.4.13 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

1.4.14 SavePDBs.py

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

1.4.15 PCCA.py

Lumps microstates into macrostates using PCCA or PCCA+ . 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+).

1.4.16 BACE_Coarse_Graining.py

An alternative method for lumping microstates into macrostates using a Bayesian approach (Bayesian agglomerative clustering engine) . 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.

1.5 Frequently Asked Questions

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

   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
   ```

2. I see the following error. What do I do?

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

   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)
   ```

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

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

4. In my implied timescales plot, I see unphysically slow timescales.

   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.

   (e) Use Ward clustering

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

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.

6. When building MSMBuilder, I see an LPRMSD error. What should I do?

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

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

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

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

8. Should I use FPCCA+ or PCCA+?

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
```

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.

10. How do I make an MSM movie?

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.

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

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.

1.6 What's New

1.6.1 2.8 Changelog

- Ported code to use MDTraj. This eliminates the need for MSMBuilder to directly handle the loading and saving of trajectories, and gives us much more cross-format support.
- New sphinx documentation

1.6.2 2.6 Changelog

- Migrated from SVN to Github for version control. The source code can now be found at https://github.com/SimTk/msmbuilder
- Renamed all the functions in MSMLib to be pep8 compliant. Aliases have been added for the old names. Note: Trajectory.py is still not pep8 compliant, but will be fixed by release 3.0.
- Split MSMLib into 2 files: MSMLib and msm_analysis. msm_analysis contains all the code for analyzing MSMs (eigenvector calculation, sampling, etc). MSMLib contains code for building models, while msm_analysis contains code for working with models.
- Removed AssignMPI.py
- Added more unit tests and Travis automated unit testing.
- Updated most of MSMBuilder to be PEP8 compliant.
- Rewrote code for estimating reversible maximum likelihood count matrix. The new code should be faster and easier to read.
- New code implementing min-cut/max-flow cut-based reaction coordinate learning from MSMs (cfep.py). Beta version.
- New code for the computation of MSM hub scores.
- Switched to YAML for storing project information.
- Added script RebuildProject.py to construct YAML ProjectInfo file.
- Deleted the `Serializer` class.
- New msmbuilder.io module for storing output.
- Added support for the BACE coarse graining algorithm.
- Better error checking for assignments arrays.
- Updates to tutorials.
- Rewrote lumping code for improved readability.

1.6.3 2.5.1 Changelog

- Updates to tutorials.
- Moved Ward clustering and SCRE to AdvancedMethods tutorial.

1.6.4 2.5 Changelog

- The libraries are distance metric agnostic, you can use your own new distance metric without having to change the clustering/assignment code.
- Dihedral, Contact (residues and/or atoms), hybrid distance metric code
- New clustering algorithms (CLARANS / subsampled CLARANS)
- Flux-based PCCA+.
- Support for Hierarchical Clustering (e.g. Ward).
- SCRE rate matrix estimation.

1.6.5 2.0.4 Changelog

- Improvements in PCCA, PCCA+ lead to better macrostate definition.

1.6.6 2.0.3 Changelog

- Bug fix: In ConvertDataToHDF, the Stride option was not being used for non-FAH style datasets.
- Fixed unicode strings were causing scipy issues on certain platforms.

1.6.7 2.0.2 Changelog

- Fixed issue with AtomIndices in UpdateProjectToHDF
- Using dtype='int' for Assignments, consistent throughout MSMBuilder.
- CalculateImpliedTimescales now allows users to exit through control+C
- Fixed some tabbing issues in MSMLib.py

1.6.8 2.0.1 Changelog

- Fixed a bug in ConvertProjectToHDF that prevented MSMBuilder from seeing multiple XTC files in a single directory.
- Fixed carriage return issues in UpdateProjectToHDF
- Fixed minor bugs in Project class.
- If Project cannot find its PDB, it also will look in the current directory. This helps when the absolute path of a Project changes.
- Unit tests.
- Removed unnecessary files, reducing the package size to 10MB.

1.7 Contributing

This is primarily an academic project, and everyone is welcome to contribute. The project is hosted on http://github.com/SimTk/msmbuilder

1.7.1 Submitting a bug report

If you experience issues using this package, do not hesitate to submit an issue to the bug tracker. You're also invited to post feature requests or links to pull requests.

Retrieving the latest code

We use Git for version control and GitHub for hosting our main repository.

You can check out the latest sources with the command:

```
git clone git://github.com/SimTk/msmbuilder.git
```

or if you have write privileges:

```
git clone git@github.com:SimTk/msmbuilder.git
```

If you run the development version, it is cumbersome to reinstall the package each time you update the sources. It is thus preferred that you add the msmbuilder directory to your `PYTHONPATH` and build the extension in place:

```
python setup.py build_ext --inplace
```

Contributing code

Note: To avoid duplicating work, it is highly advised that you contact the developers on the issue tracker before starting work on a non-trivial feature.

1.7.2 How to contribute

The preferred way to contribute to msmbuilder is to fork the main repository on GitHub:

1. Create an account on GitHub if you do not already have one.
2. Fork the project repository: click on the 'Fork' button near the top of the page. This creates a copy of the code under your account on the GitHub server.
3. Clone this copy to your local disk:

   ```
   $ git clone git@github.com:YourLogin/msmbuilder.git
   ```

4. Create a branch to hold your changes:

   ```
   $ git checkout -b my-feature
   ```

   and start making changes. Never work in the `master` branch!
5. Work on this copy, on your computer, using Git to do the version control. When you're done editing, do:

   ```
   $ git add modified_files
   $ git commit
   ```

   to record your changes in Git, then push them to GitHub with:

   ```
   $ git push -u origin my-feature
   ```

Finally, go to the web page of your fork of the msmbuilder repo, and click 'Pull request' to send your changes to the maintainers for review. request. This will send an email to the maintainers.

(If any of the above seems like magic to you, then look up the Git documentation on the web.)

It is recommended to check that your contribution complies with the following rules before submitting a pull request:

- Follow the coding-guidelines (see below).
- All public methods should have informative docstrings with sample usage presented as doctests when appropriate.
- All other tests pass when everything is rebuilt from scratch. On Unix-like systems, check with:

```
$ nosetests
```

You can also check for common programming errors with the following tools:

- No pyflakes warnings, check with:

```
$ pip install pyflakes
$ pyflakes path/to/module.py
```

- No PEP8 warnings, check with:

```
$ pip install pep8
$ pep8 path/to/module.py
```

- AutoPEP8 can help you fix some of the easy redundant errors:

```
$ pip install autopep8
$ autopep8 path/to/pep8.py
```

Note: The current state of the msmbuilder code base is not compliant with all of those guidelines, but we expect that enforcing those constraints on all new contributions will get the overall code base quality in the right direction.

Note: For two very well documented and more detailed guides on development workflow, please pay a visit to the Scipy Development Workflow and the Astropy Workflow for Developers pages.

Coding guidelines

The following are some guidelines on how new code should be written. Of course, there are special cases and there will be exceptions to these rules. However, following these rules when submitting new code makes the review easier so new code can be integrated in less time.

Uniformly formatted code makes it easier to share code ownership. The msmbuilder project tries to closely follow the official Python guidelines detailed in PEP8 that detail how code should be formatted and indented. Please read it and follow it.

In addition, we add the following guidelines:

- Use underscores to separate words in non class names: `n_samples` rather than `nsamples`.
- Avoid multiple statements on one line. Prefer a line return after a control flow statement (`if`/`for`).
- **Please don't use 'import *' in any case**. It is considered harmful by the official Python recommendations. It makes the code harder to read as the origin of symbols is no longer explicitly referenced, but most important, it prevents using a static analysis tool like pyflakes to automatically find bugs.
- Use the numpy docstring standard in all your docstrings.

A good example of code that we like can be found here.

Building the docs

To build the documentation on your local machine, you need to first ensure that the numpydoc package is installed. The documentation itself can be built with a `make` command from within the `docs/sphinx` directory:

```
$ easy_install numpydoc
$ make html
```

CHAPTER 2

MSM Theory Notes

2.1 Time-Structure Based Independent Component Analysis (tICA)

2.1.1 Introduction

The time-structure based Independent Component Analysis (tICA) method as applied to MSM construction is a new way to judge distances in the protein conformational landscape. The strategy will be to define a reduced dimensional representation of the protein conformations, and use distances in this space in the clustering step of the MSM construction process. Consider a single, long trajectory from a simulation, $\mathbf{X}(t) \in R^d$, which has d dimensions each corresponding to some structural degree of freedom (such as a single phi angle).

One natural way we could define the reduced space is to use Principal Component Analysis, or PCA (See Gerhard Stock's work for examples). In PCA, the goal is to find projection vectors that maximize their explained variance, subject to them being uncorrelated and having length one. In the end, these maximal variance projections correspond to the solutions of the following eigenvalue problem:

$$\Sigma v = \lambda v$$

where Σ is the covariance matrix given by:

$$\Sigma_{ij} = E\Big[X_i(t)X_j(t)\Big]$$

The problem with using PCA to define the reduced space, however, is that high-variance degrees of freedom need not be slow (for instance consider a floppy protein tail that varies wildly vs. a single dihedral angle that is required to rotate for a protein to fold). What we really want is to design projections that can best differentiate between slowly equilibrating populations, which is precisely where tICA comes in.

In tICA, the goal is to find projection vectors that maximize their autocorrelation function, subject to them being uncorrelated and having unit variance. It is easy to show (see Schwantes, CR and Pande, VS. *JCTC* **2013**, 2000-2009.) that the solution to the tICA problem are the solutions to this generalized eigenvalue problem (which is closely related to the PCA eigenvalue problem):

$$C^{(\Delta t)}v = \lambda \Sigma v$$

where $C^{(\Delta t)}$ is the time lag correlation matrix defined by:

$$C_{ij}^{(\Delta t)} = E\Big[X_i(t)X_j(t+\Delta t)\Big]$$

Given this solution, we can use the tICA method to define a reduced dimensionality representation of each $\mathbf{X}(t)$ by projecting the vector onto the slowest n tICs. Therefore, the strategy for using tICA to construct an MSM looks like:

1. Calculate $C^{(\Delta t)}$ and Σ and the solutions to the generalized eigenvalue problem given above
2. Choose the number of tICs to project onto
3. Use the reduced space to cluster and assign conformations to states
4. Build the MSM from these assignments and analyze as laid out in the MSMBuilder tutorial

In the next section we will go over how to do each of these steps within MSMBuilder.

2.1.2 Selection of tICA Parameters

There are two parameters introduced in the tICA method. The first is Δt, which is used in the calculation of the time-lag correlation matrix ($C^{(\Delta t)}$). The second is n, which is the number of tICs to project onto when calculating distances between conformations.

There is currently no optimal method for choosing these parameters, however, the method has been fairly robust to different choices.

In previous analyses of Fip35, villin, NTL9(1-39), and NuG2, Δt between 50 and 1000 nanoseconds produced MSMs with largely the same timescale distribution, indicating that something in this range should be appropriate for most protein systems.

MSMs are more sensitive, however, to the selection of the number of tICs to project onto. In the previous analyses listed above, we found that using a surprisingly small number (in the range of 5-20) of tICs worked well. The power behind tICA is to ignore the degrees of freedom that quickly decorrelate and only add noise in the distance calculation. However, as n gets smaller (and we throw out more degrees of freedom), the resolution of the MSM becomes limited, and can only discern between conformations along the slowest coordinate (which is often the folding process).

For example, in our analysis of NTL9, we found that increasing n from three up to seven kept the folding timescale largely unchanged but added new microsecond timescales to the resulting MSMs, while adding in too many (> 10) produced a folding timescale that was too fast.

2.1.3 Understanding the tICs

The top tICs represent linear combinations of the input degrees of freedom that decorrelate slowly. These vectors are not necessarily easy to visualize, however.

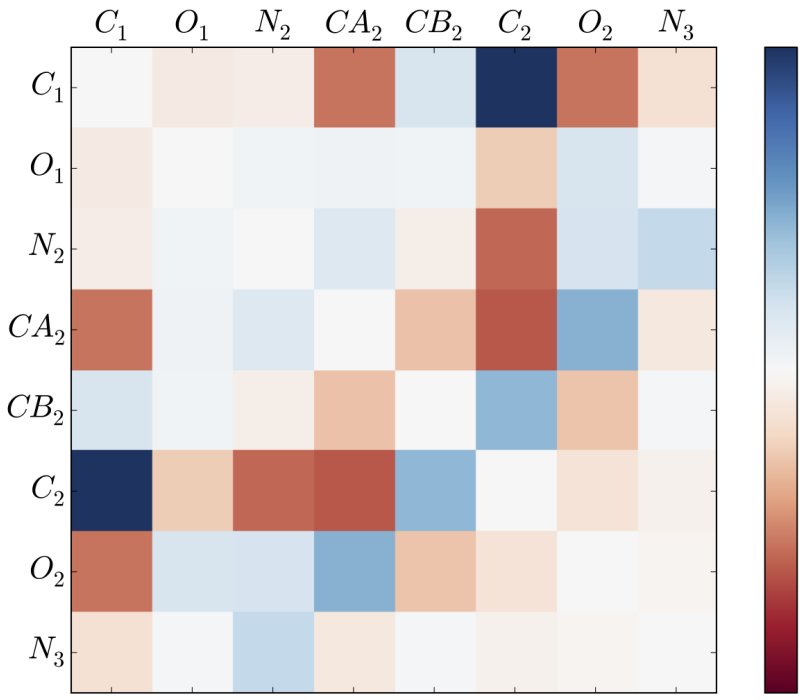
For instance, the slowest tIC from the above analysis can be visualized as a matrix, where each entry corresponds to the tIC entry corresponding to a pair of atoms' distances.

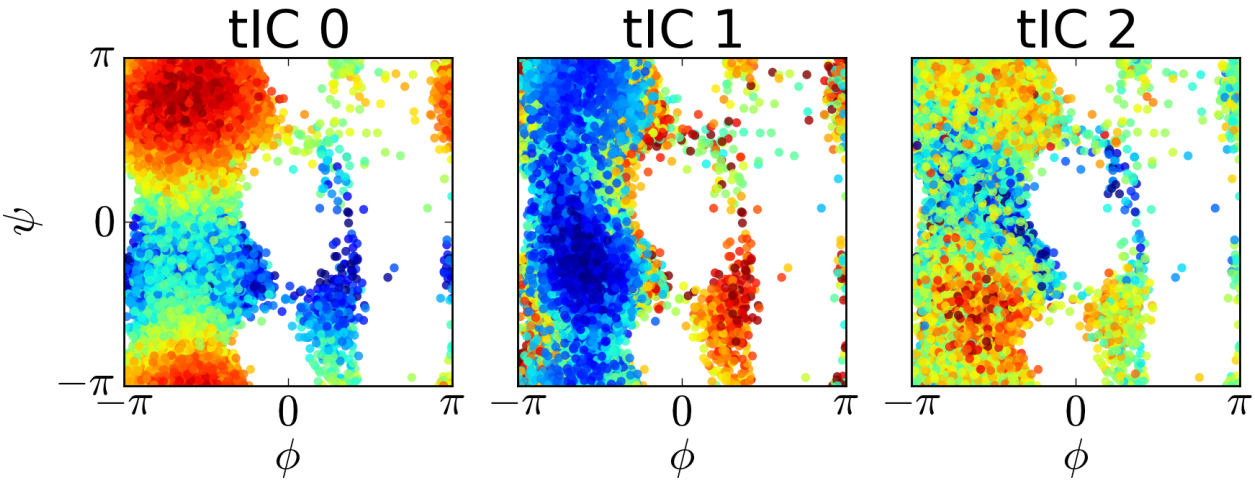
Here, the dark blue and dark red portions correspond to atom pairs that best distinguish between far regions along the first tIC. As is clear, this is not all that helpful to look at (though an area that could be greatly improved is providing a visualization tool for these degrees of freedom).

We can also attempt to visualize the tICs by comparing the projections onto each tIC to order parameters. For instance, for each conformation we sampled in the reference simulations, we can calculate the phi and psi angles along with the projection of the conformation onto each tIC. In this way, we can visualize what each tIC corresponds to. Below, we have plotted the phi and psi angles colored by that conformation's projection onto each tIC.

2.1.4 Drawbacks of tICA

Since part of the process of using tICA is a dimensionality reduction, there is always the opportunity to throw out important pieces of information. In particular, by throwing out the faster degrees of freedom, we can better estimate


C_1 O_1 N_2 CA_2 CB_2 C_2 O_2 N_3
C_1
O_1
N_2
CA_2
CB_2
C_2
O_2
N_3


tIC 0
tIC 1
tIC 2
ψ
ϕ
π
0
$-\pi$

the slowest timescales; but this comes with the trade-off of not representing the fast timescales correctly. The result is illustrated when trying to sample a trajectory from the MSM built on tICA. The result is a trajectory that represents the folding/unfolding transition well, but when in the unfolded state jumps around more than would be seen in a typical MD simulation.

2.2 Calculation of Autocorrelation Functions

First, suppose we have calculate the left eigenvectors and eigenvalues:

$$T^T \phi_i = \lambda \phi_i$$

Suppose that π is the equilibrium population. Then, we can normalize the eigenvectors such that:

$$\phi_i^T \pi^{-1} \phi_j = \delta_{ij}$$

Above, we denote π^{-1} to be a diagonal matrix with elements π_i^{-1}.

The autocorrelation function of the observable f_i can be denoted:

$$E(f(z_t)f(z_0)) = \sum_{i,j} f_i P(z_0 = i) f_j P(z_t = j | z_0 = i) = \sum_{i,j} f_i f_j \pi_i T_{ij} =$$

We know that

$$T_{ab}(t) = \sum_k \lambda_k(t)(\psi_k)_a(\phi_k)_b = \sum_k \lambda_k(t)(\pi_a)^{-1}(\phi_k)_a(\phi_k)_b$$

Thus,

$$E(f(z_t)f(z_0)) = \sum_{i,j,k} f_i f_j \lambda_k(t)(\phi_k)_i(\phi_k)_j = \sum_k \lambda_k(t) s_k^2$$

Where

$$s_k = \sum_i f_i(\phi_k)_i$$

Finally, note that $\lambda_i(\infty) = \delta_{i0}$, so the long-timescale behavior is simply:

$$E(f(z_\infty)f(z_0)) = s_0^2$$

For most applications, one is interested in the zero-centered ACF, so we simply skip the $k = 0$ term in the summation.

2.3 Maximum-Likelihood Reversible Transition Matrix

Here, we sketch out the objective function and gradient used to find the maximum likelihood reversible count matrix.

Let C_{ij} be the matrix of observed counts. C must be strongly connected for this approach to work! Below, f is the log likelihood of the observed counts.

$$f = \sum_{ij} C_{ij} \log T_{ij}$$

Let $T_{ij} = \frac{X_{ij}}{\sum_j X_{ij}}$, $X_{ij} = \exp(u_{ij})$, $q_i = \sum_j \exp(u_{ij})$

Here, u_{ij} is the log-space representation of X_{ij}. It follows that $T_{ij} = \exp(u_{ij})\frac{1}{q_i}$, so $\log(T_{ij}) = u_{ij} - \log(q_i)$

$$f = \sum_{ij} C_{ij} u_{ij} - \sum_{ij} C_{ij} \log q_i$$

Let $N_i = \sum_j C_{ij}$

$$f = \sum_{ij} C_{ij} u_{ij} - \sum_{i} N_i \log q_i$$

Let $u_{ij} = u_{ji}$ for $i > j$, eliminating terms with $i > j$.

Let $S_{ij} = C_{ij} + C_{ji}$

$$f = \sum_{i \leq j} S_{ij} u_{ij} - \frac{1}{2} \sum_i S_{ii} u_{ii} - \sum_i N_i \log q_i$$

$$\frac{df}{du_{ab}} = S_{ab} - \frac{1}{2} S_{ab} \delta_{ab} - \sum_i \frac{N_i}{q_i} \frac{dq_i}{du_{ab}}$$

$$\frac{dq_i}{du_{ab}} = \exp(u_{ab})[\delta_{ai} + \delta_{bi} - \delta_{ab}\delta_{ia}]$$

Let $v_i = \frac{N_i}{q_i}$

$$\sum_i V_i \frac{dq_i}{du_{ab}} = \exp(u_{ab})(v_a + v_b - v_a \delta_{ab})$$

Thus,

$$\frac{df}{du_{ab}} = S_{ab} - \frac{1}{2} S_{ab} \delta_{ab} - \exp(u_{ab})(v_a + v_b - v_a \delta_{ab})$$

CHAPTER 3

Library API Reference

3.1 msmbuilder.MSMLib

apply_mapping_to_assignments
apply_mapping_to_vector
build_msm
ergodic_trim
ergodic_trim_indices
estimate_rate_matrix
estimate_transition_matrix
get_count_matrix_from_assignments
get_counts_from_traj
invert_assignments
log_likelihood
mle_reversible_count_matrix
permute_mat
renumber_states
tarjan
trim_states

3.2 msmbuilder.msm_analysis

calc_expectation_timeseries
get_eigenvectors
get_implied_timescales
is_transition_matrix
msm_acf
project_observable_onto_transition_matrix
propagate_model
sample

3.3 msmbuilder.assigning

Continued on next page

Table 3.3 – continued from previous page

assign_in_memory
assign_with_checkpoint

3.4 msmbuilder.clustering

BaseFlatClusterer
Clarans
Hierarchical
HybridKMedoids
KCenters
concatenate_prep_trajectories
unconcatenate_trajectory
concatenate_trajectories
deterministic_subsample
stochastic_subsample
p_norm

3.5 msmbuilder.drift

get_epsilon_neighborhoods
hitting_time
square_drift

3.6 msmbuilder.cfep

contact_reaction_coordinate
VariableCoordinate
CutCoordinate

3.7 msmbuilder.SCRE

ConstructRateFromParams
ConvertTIntoK
FixEntry
GetFormat
LogLikelihood
MaximizeRateLikelihood
PlotRates

3.8 msmbuilder.reduce.tICA

tICA
tICA.initialize
tICA.project
tICA.save
tICA.solve
tICA.train

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*