MSMBuilder 2.5 Tutorial

April 9, 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
- 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 all 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 Using MSMBuilder

MSMBuilder has been designed to provide both ease of use and versatility. To achieve both these goals, we have pursued a 2-pronged approach.

1. MSMBuilder is a library.
2. MSMBuilder is a set of python scripts.

Using MSMBuilder as a library allows for fast but powerful customization. Using MSMBuilder as a library is recommended only for advanced users and is described in the MSMBuilder Scripting Guide.

MSMBuilder contains a suite of python scripts that automate the most common tasks in model construction. For most situations, conformational dynamics can be modeled with the following protocol. Each script below provides instructions by running with the -h flag. Note that setup.py should have installed each script below to someplace in your PATH.

3.1 MSMBuilder Scripts

3.1.1 ConvertDataToHDF.py

Merges XTC files into continuous 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

3.1.2 CreateAtomIndices.py

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

3.1.3 Cluster.py

Clusters your data based on RMSD using a novel hybrid k-centers / k-medoids algorithm. It is recommend that you subsample data at a timestep similar to (10X) the desired lagtime of your eventual model (over-subsampling

can lead to kinetic barriers within states). Choose a clustering diameter (-D) suitable for your system. We find that for folding simulations of villin, a diameter of 0.175 nm led to a high resolution model. We recommend approximately 10 iterations of the (local) hybrid k-medoid clustering. Additional iterations of local and global hybrid clustering may lead to slightly reduced discretization error (up to 25NOTES: -The hybrid k-medoids clustering works by simultaneously minimizing both the average-case and worst-case clustering errors. Email kyle for details / MSMBuilder2 draft. - Stride is in units of frames to subsample. It will subsample whatever is in your Trajectories/ directory, so if you subsampled in the ConvertToHDF.py script then this will subsample the subsample.

3.1.4 Assign.py

Assigns data to the cluster generators. Use AssignOnPBS.py and MergeAssignOnPBS.py to parallelize this process on a PBS-based cluster (highly recommended!).

3.1.5 CalculateImpliedTimescales.py

Calculates the implied timescales for a python range of MSM lag times. This is equivalent to the old BuildMSMsAsVaryLagTime.py. Can be used on Micro or Macro scale models. NOTES: - You might get a SparseEfficiencyWarning for every lag time. Ignore this. - Lagtimes are input as frames that have been assigned, i.e. Those you have converted to HDF5

3.1.6 PlotImpliedTimescales.py

A template for generating an implied timescales plot.

3.1.7 BuildMSM.py

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

1. Merge states with low counts to improve statistics 2. Use Tarjan algorithm to find the maximal strongly-connected (ergodic) subgraph 3. Use MLE (e.g. Boxer iteration) to estimate a reversible count matrix consistent with your data.

This script also outputs the equilibrium populations of the resulting model.

3.1.8 GetRandomConfs.py

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

3.1.9 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.

3.1.10 CalculateRMSD.py

Uses MSMb2's rapid RMSD calculator to calculate the RMSD of your trajectories/generators/assignments. Very useful for deciding which clusters belong to the folded, unfolded, or transition state ensembles (or any other grouping!)

3.1.11 CalculateProjectRMSD.py

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

3.1.12 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

3.1.13 SavePDBs.py

Allows you to rip random PDBs from your project and save them to disk.

3.1.14 PCCA.py

Perron Cluster-Cluster Analysis (PCCA) clusters Microstates into Macrostates. This script generates a new, macro-level assignments file from a microstate model. (Use Assignments.Fixed.h5) The simplex version can be specified. This version is more robust, but more resource-intensive. See the main discussion of macrostate models in XXX.

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. Calculate macrostates using (f)PCCA+
6. Calculate a macrostate rate matrix using SCRE
7. 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 ./dipeptide.pdb -I ~/msmbuilder2.0/Tutorial/XTC/
```

4. Cluster your data: Note: we are using a 0.3 Å clustering diameter, which is appropriate for alanine dipeptide. The follow protocol uses the default scheme of clustering using k-centers with desired clustering diameter, followed by 10 iterations of a local hybrid k-medoids. Other protocols are possible; see Cluster.py -h for details.

Cluster.py -r 0.03

5. Assign data to states:

Assign.py

6. Validate model with relaxation timescales. First, we calculate the relaxation timescales for a sequence of lagtimes:

CalculateImpliedTimescales.py -l 1,50 -X 1

Next, we use python to plot the results.

PlotImpliedTimescales.py -I ImpliedTimescales.dat -t 1.

Note that in this example, relaxation timescales and lagtimes are plotted in units of [ps]; your own data may have different units.

7. 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.

8. Examine structural, equilibrium, and kinetic properties of MSM

First, let's use a tiny Python script to determine the 10 most populated states in our model:

python -c "from numpy import *;Pops=loadtxt('./Data/Populations.dat');Ind=argsort(Pops)[-5:];print(Ind);print(Pops[Ind])"

My model gives the following states and populations. Note that your results may differ due to the random search in k-medoids clustering.

[52 153 102 104 135] [0.02064257 0.02078313 0.02212851 0.02295181 0.02321285]

Let's extract 3 random conformations from each of these states:

SavePDBs.py -c 3 -H 52 153 102 104 135 -a Data/Assignments.Fixed.h5

Now, let's view them in Pymol:

pymol PDBs/State*.pdb

Because of its small size, structures of alanine dipeptide look quite similar in pymol. For larger proteins, however, visualization in Pymol often shows distinctive structural features of each MSM state.

9. Construct and validate a macroscopic MSM Spectral cluster methods such as PCCA and PCCA+ can be used to construct models with a minimal number of states. This can be useful for building intuition on the underlying equilibrium and kinetic properties. Note: macroscopic MSMs tend to have reduced kinetic fidelity, so they may be less appropriate for quantitative kinetic predictions.

In general, we find that PCCA works best for shorter lagtimes; this is likely due to the fact that the amount of available statistics decreases as lagtime increases. Thus, we construct an MSM with the shortest possible lagtime, 1 (here in ps).

BuildMSM.py -l 1 -w Model1

Our 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 -M 4 -a Model1/Assignments.Fixed.h5 -T Model1/tProb.mtx -w Model1/

As before, we calculate and plot the relaxation timescales:

CalculateImpliedTimescales.py -l 1,50 -X 1 -a Model1/MacroAssignments.h5 -o MacrostateRelaxationTimescales.dat -e 3

PlotImpliedTimescales.py -I MacrostateRelaxationTimescales.dat -t 1.

Note that the slowest timescales are not flat until the lagtime reaches 5 ps; this highlights the reduction in kinetic accuracy inherent in model reduction.

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.

cp /msmbuilder/Tutorial/Phi.h5 ./ cp /msmbuilder/Tutorial/Psi.h5 ./ cp /msmbuilder/Tutorial/PlotDihedrals.py ./

python PlotDihedrals.py ./Model1/MacroAssignments.h5

You should see the following graphs:

Thus, the PCCA algorithm has automatically identified the key basins of alanine dipeptide. 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 8 states, we would produce a Ramachandran plot like the following:

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.