MSMBuilder Scripting and Developer Guide

Kyle Beauchamp TJ Lane Greg Bowman
Robert McGibbon Vijay Pande

July 27, 2012

1 Table of Contents

2 Docstrings

We are currently in the process of improving the docstrings throughout MSMBuilder. Every MSMbuilder function and class should have a built-in "help" function which can be explored in an interactive python session using the "help()" function. The docstring should indicate the proper usage of any function.

3 MSMBuilder Components

1. File IO and Serialization
2. MSMLib
3. Project
4. Distance Metric
5. Clustering
6. Lumping

3.1 File IO and Serialization

The following modules are useful for reading and writing files:

1. Serializer
2. Trajectory
3. Conformation

The Serializer is a dictionary that allows its keys to be saved to disk using the pytables HDF5 library. The resulting files will be saved using PyTables with the BloSC compression algorithm, leading to reasonable compression and performance. The Serializer can also be used to save arbitrary Numpy arrays to disk in a fast compressed format, as shown in the following example:

```
import numpy as np
from msmbuilder import Serializer

X = np.random.normal(size=(10,10))
Serializer.SaveData("./Data.h5",X)
X = Serializer.LoadData("./Data.h5")
```

The Trajectory object holds atom names, residue names, and coordinates for a sequence of snapshots. Trajectories can be loaded or saved in a number of formats, including XTC, PDB, DCD, and LH5. LH5 is the MSMBuilder default storage–it uses the Serializer to save the trajectory as a Pytables file. The following example loads an MSMBuilder trajectory, translates the coordinates by 1 nanometer, and saves it as an XTC for viewing in VMD:

```
from msmbuilder import Trajectory
R = Trajectory.Trajectory.LoadFromLHDF("SomeTrajectory.lh5")
R["XYZList"] += 1.0
R.SaveToXTC("./out.xtc")
```

The Conformation class provides access to conformation objects, as well as PDB input and output.

```
from msmbuilder import Conformation
C = Conformation.LoadFromPDB("./native.pdb")
```

3.2 MSMLib

MSMLib.py contain code necessary to construct and manipulate transition matrices and transition count matrices. Each function in MSMLib typically involves one of the following inputs:

1. An Assignments array
2. A (sparse or dense) Count Matrix

3. A (sparse or dense) Transition Matrix

4. A numpy array of populations

As an example, the following code calculates the 10 slowest eigenvectors of a transition matrix that that is loaded from disk:

```
from msmbuilder import MSMLib
import scipy.io
T = scipy.io.mmread("./Ward-2500/tProb.mtx").tocsr()
eigenvalues, eigenvectors = MSMLib.GetEigenvectors(T,10)
```

3.3 Project

The Project class contains the machinery necessary for interacting with a collection of MD trajectories. Project facilitates a number of operations, including:

1. Enumerating (and returning) the conformations in a dataset.

2. Randomly sampling conformations in a dataset.

3. Clustering a dataset

4. Assigning a dataset to states.

4 MSMBuilder Developer Guidelines

This document defines guidelines on how to contribute code to MSMBuilder.

4.0.1 Principle Developers

The principle developers for MSMBuilder (as of July 2012) are Kyle Beauchamp, Greg Bowman, TJ Lane, and Robert McGibbon. The PI of MSMBuilder is Vijay Pande.

4.1 Gaining SVN Access to the code

To gain read access to the MSMBuilder SVN, contact a Principle Developer or PI. In general, we are happy to provide read-only access to collaborators. The SVN may contain unpublished work, and we ask that you keep the SVN code confidential. Write access to the SVN is generally restricted to Pande group members, alumni, or close collaborators.

4.1.1 Bug Fixes

To contribute a bug fix, you should have approval of at least one principle developer.

4.1.2 How do I contribute new code to MSMBuilder?

In general, committed code must be useful. It must be, as far as possible, bug free. Finally, it must pass all unit tests. If your code satisfies these conditions, we are happy to include your code in one of the following ways.

If you wish to contribute substantial code changes to the "Core" of MSMBuilder, you need approval from a majority of the principle developers. We define the "Core" as any function or class that is called during the standard MSM pipeline, as documented in the Tutorial. For major changes to the "Core," we may make an SVN branch for your patch. This way, your code can take advantage of new additions, be checked out by others, and more easily integrated in, while the principle developers work with you to find a way to best resolve their concerns.

4.1.3 Unit Tests

Please ensure that contributed code contains unit tests to ensure its correctness.