EMMA 1.2 Markov Model Algorithms Tutorial and Documentation

Martin Senne* and Frank Noé

martin.senne@fu-berlin.de

frank.noe@fu-berlin.de

September 2, 2011

Markov models, often called Markov state models (MSMs) or kinetic transition networks, are concise and easy-to-analyze discrete models of continuous stochastic processes. They have been extensively used in the analysis of molecular dynamics (MD) [34, 35, 8, 17, 5, 44], see [33] for an introductory overview. The EMMA software provides a number of command line tools that provide the basics of construction, validation and analysis of Markov models. EMMA is Java-based and can be executed on Linux, Windows and Mac OS shells. It can be used in conjunction with a few common molecular dynamics trajectory file formats (Gromacs xtc, Charmm dcd, NAMD dcd, tabulated ASCII). EMMA stands for **E**MMA's **M**arkov **M**odel **A**lgorithms.

This document is organized in three parts:

Part-I tutorial with installation directions. The individual EMMA commands are then illustrated on a simple model of a diffusion in a metastable potential. Example input and output files are provided with the software.

Part-II command reference that explains the available EMMA commands and all arguments in detail.

Part-III (in Appendix) definition of file formats

The EMMA v1.2 framework provides the following commands:

mm_generate Generate sample trajectories for simple model potentials (this is just meant for testing)

mm_discretize Discretize the state space of continuous input trajectories (known formats dcd, xtc or ASCII) and construct discrete trajectories. Clustering is done in two steps: (1) choosing cluster centers with k-means, k-centers, equally spaced in time or equally spaced in space; (2) assigning all trajectory frames to the nearest center using either Euclidean or minimal RMS distance.

mm_connectivity Test the connection between microstates and determine the largest connected set of microstates.

mm_timescales Computation of implied timescales of Markov models constructed on the discrete trajectories using different lagtimes τ. This is useful to decide whether the clustering is sufficiently fine and to pick an appropriate lagtime τ.

mm_estimation Calculation of the transition probability matrix $\mathbf{T}(\tau)$, which *is* the Markov model for given lagtime τ using all data.

mm_transitionmatrixAnalysis Basic tools to analyze the Markov model $\mathbf{T}(\tau)$, especially including calculating eigenvalues, eigenvectors and the stationary distribution.

mm_pcca Tools to compute metastable states from a Markov model [34, 49, 43].

*Surname changed from Fischbach to Senne.

mm_tpt Transition path theory is useful for the computation of transition pathways and fluxes from Markov models, e.g. folding pathways [35].

mm_observables Analyze MSMs in a way that allows comparison to experimental measurements. Dynamical observables such as perturbation-relaxation and correlation curves can be calculated as they can also be measured in kinetic experiments. The interpretation of this curves in terms of *dynamical fingerprints* which can be dissected into dynamical features that are associated with individual relaxation timescales and structural rearrangement processes [36, 20].

Part I. Tutorial

1. Overview of the tutorial

This tutorial presents the essential steps to construct, validate and analyze a Markov model from molecular dynamics simulation data. For the sake of simplicity and execution speed, the tutorial uses 2d model trajectory as input data. The tutorial will enlighten the individual steps and commands to construct and validate a Markov model, as well as it will present possible analysis steps to extract essential features.

Having this general approach in mind, a typical Markov model analysis from molecular dynamics data may include the steps listed below. A schematic overview is given in Figure 1.

- Preparation of input data
  - Optionally, the input data is **reduced in dimensionality**, thus reduced to "interesting coordinates", which may be certain angles, distances or principal components (via PCA [1]). Such data reduction will speed up the modeling process and may help to reduce statistical problems. On the other hand, the construction of appropriate and valid Markov Models may be impossible when important degrees of freedom are neglected.
- I. Markov state model construction
  - **Discretization** of time-series data. Either
    - by simple partitioning such as binning, or
    - by geometrical clustering algorithms (e.g. k-means, k-centers, ...) with
      - determination of appropriate cluster centers and
      - subsequent assignment of time-series data to these cluster centers, which is usually done by Voronoi partitioning.
  - **Determination of** an **appropriate lagtime**: The computation of implied timescales for discrete time-series data reveals this information.
  - **Count matrix construction** from discretized data for appropriate lagtime.
  - **Transition probability matrix construction** from determined count matrix. This transition matrix together plus the structural definition of states is called Markov Model.
- II. **Determination of metastable sets** by means of the Perron-cluster-cluster-analysis (PCCA) method.
- III. Markov Model **validation**
  - Chapman-Kolmogorov test

- IV. Markov model **analysis**
  - Analysis of transition matrix: eigenvalues and -vectors, stationary distribution
  - **Computation of transition pathways** and fluxes from source to target states.

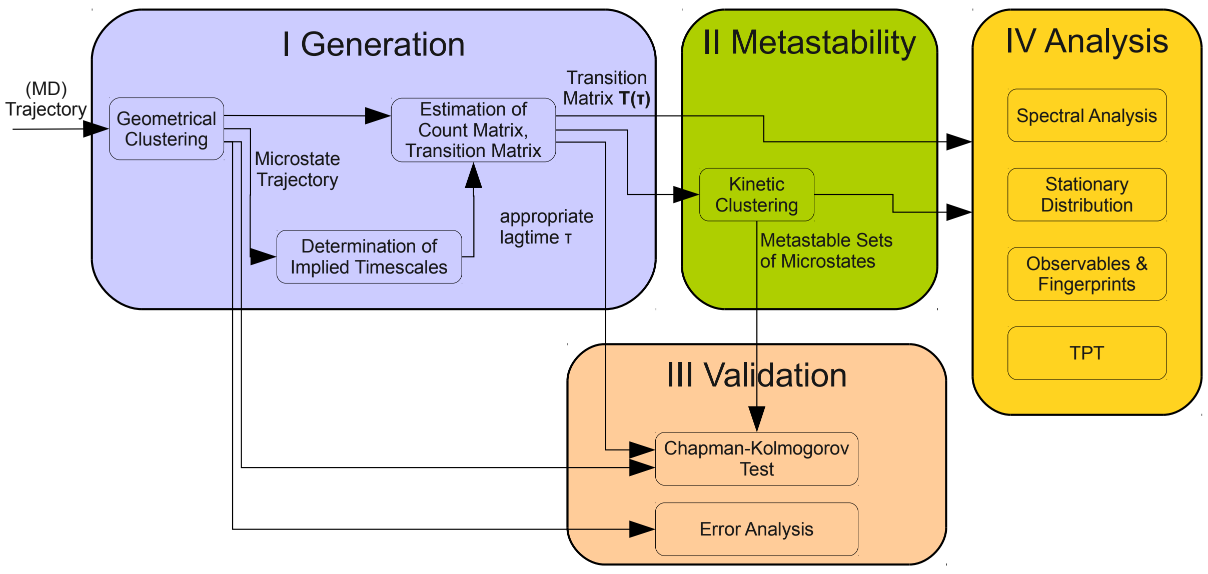

Figure 1: Schema of the steps involved in construction, validation and analysis of Markov state models.

2. Installation and Preparation

The following steps are required to use the tools for Markov Model construction:

- Extract provided zip *Emma_v1_2__DATE_TIME.zip*.
- Make sure, you have an appropriate Java environment setup.
- Only if running from Linux, please make sure you have the Bash-shell installed. The scripts use the Bash-shell internally, so it is required.

The scripts have been tested for Linux, MacOS and Windows. In case of any difficulties, problems or irregularities or potentials bugs please contact the authors.

2.1. Zip-File extraction

Extract the zip-file *MarkovStateModel.zip* to a folder of your choice. Either use *unzip* (Linux) or an archive extraction utility of your choice (file-roller in Linux, WinZip or Winrar in Windows).

2.2. Java Environment and prerequisites

The usage of the command line tools for Markov models requires that you have installed **Sun Java Runtime Environment (Version 5 (v1.5) or later)**. The command line tools have been developed and tested with Sun Java JDK. Other Java runtime environments implementations, such as Open-JDK may work as well, but have not been tested. (Feedback on those is welcome.)

The EMMA-package contains a collection of command-line tools. Each tool is command which is to be executed via a command line shell (on Windows/Linux / MacOS). Since the commands rely on the Java executable, it is essential that you have added the Java executable "java" to your path. Please test this by executing the following command in a console to validate, that your java version is sufficient and your path is set appropriately:

```
> java -version
```

The output should be similar to:

```
java version "1.6.0_20"
Java(TM) SE Runtime Environment (build 1.6.0_20-b02)
Java HotSpot(TM) Server VM (build 16.3-b01, mixed mode)
```

Here, the java version is reported to 1.6., but any version higher than 1.5 is sufficient.

If the command is unrecognized, you need to modify the path variable. For Linux this requires (see also: http://www.troubleshooters.com/linux/prepostpath.htm):

```
> PATH=$PATH:/path/to/java/executable
> export PATH
```

Under Windows the installation routine of Java (JRE or JDK) places the Java executable "java.exe" usually at the location "c:\windows\system32", where the executable is automatically included by the system path. If you can not execute "java" directly from the console, please modify your path environment accordingly. There are several how-to's available, when searching for "set path windows" under Google.

3. Model Trajectory Generation

The first step of the tutorial consists in the generation of two 2-dimensional model toy trajectories, which are used for the Markov model construction and for further analysis.

Open a shell and navigate to the directory (via the "cd" command), where the .zip file was extracted to.

Change path to the example directory via

```
Clustering$ cd example
```

Now execute the command below to generate the first model trajectory starting at (0.0, 0.0). For Linux, the command execution is

```
Clustering/example$ ./mm_example_generate_start0
```

For Windows (if running the cmd shell) please omit the "./" and add ".bat" at the end of each command. Thus please call

```
Clustering\example> mm_example_generate_start0.bat
```

In the following, the commands are only denoted for Linux, so please adapt the example commands accordingly if using Windows.

After the execution of the command, the output should be similar to

```
Command line parameters valid.
Generating trajectory 'modeltrajectories/traj1.traj' with 10000000 steps,
starting at ( 0.0, 0.0).
```

The command which is actually executed by the example script is

```
mm_generate
-start 0.0 0.0
-steps 10000000
-dt 0.1
-potdef potentials/toypotentialdef.ascii
-o modeltrajectory/traj1.traj
```

This command generates a trajectory on the potential defined in the file *potentials/toypotentialdef.ascii*. Details are given in Section 8.1.

The toy potential consists of a sum of two-dimensional Gaussian basins with the definition:

$$B(x,y) = -I \exp\left(-\frac{(x-\mu_x)^2}{2\sigma_x^2} - \frac{(y-\mu_y)^2}{2\sigma_y^2}\right),$$

using the parameters:

Basin #	I Intensity	μ_x	μ_y	σ_x	σ_y
1	2.0	15.0	15.0	20.0	15.0
2	1.2	9.0	9.0	5.0	5.0
3	0.8	21.0	9.0	5.0	5.0
4	1.0	13.0	21.0	5.0	5.0

By executing the following command, we generate a second model trajectory, this time starting at (10.0, 10.0).

```
Clustering/example$ ./mm_example_generate_start10
```

Since the created model trajectories are ascii-files an arbitrary editor can be used to view the content of the files.

A plot of a generated trajectory is given in Figure 2. The trajectory within the potential of four Gaussian basins is given in Figure 3.

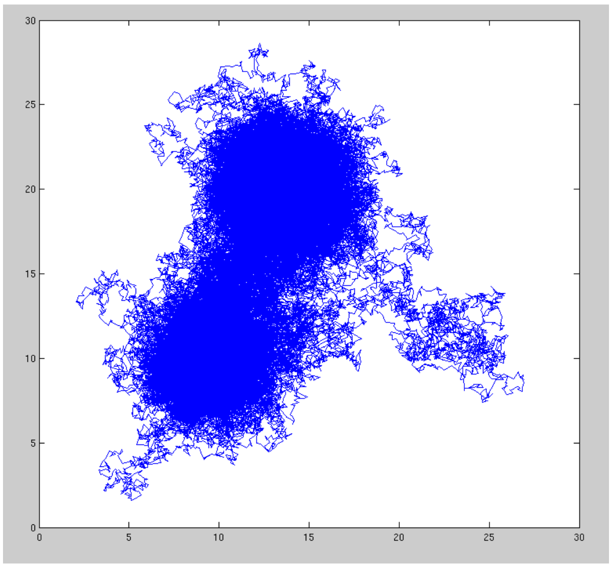

Figure 2: The generated trajectory.

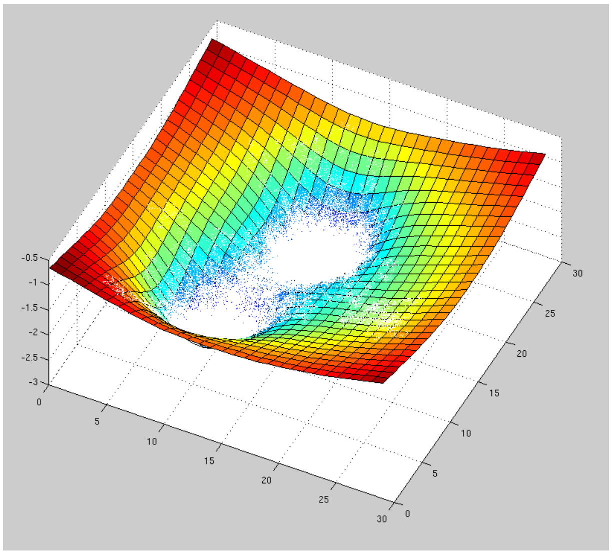

Figure 3: The generated trajectory within the potential of four Gaussian basins.

4. Markov Model Construction

4.1. Clustering and discretization

Two 2-dimensional model trajectories have been created in the last step. The construction of an appropriate Markov model is done within the next steps. First the trajectories need to be clustered and discretized (see 4.1). In order to do so the command

```
Clustering/example$ ./mm_example_discretize
```

needs to be executed. Internally this invocation calls the command:

```
mm_discretize
-i modeltrajectories/traj1.traj modeltrajectories/traj2.traj
-istepwidth 1000
-algorithm kmeans -clustercenters 50 -metric euclidian
-oclustercenters discretized/clusterCenters.ascii
-o ./discretized
```

which takes every 1000th step of the input model trajectories as input for a clustering algorithm. Here, the k-means clustering algorithm is selected for the computation of a a total number of 50 clusters, while using the Euclidean metric. After the cluster centers have been determined by the clustering algorithm each frame of the trajectory is assigned to the closest cluster according to the given metric (in the above case Euclidean metric). This assignment is called Voronoi-Partitioning. The resulting sequence of microstates obtained thereby is called microstate trajectory or discretized trajectory.

In addition, the clusters centers obtained by the clustering algorithm are written to a file (in our case: "./discretized/clusterCenters.ascii"). This file contains in the i-th line the i-th cluster center representing microstate i. A graphical illustration of the trajectory and the determined cluster centers is given in Figure 4.

The output of the example command is similar to:

```
Command line parameters valid.
Trajectories used for input
  modeltrajectories/traj1.traj
  modeltrajectories/traj2.traj
Using stepwidth of 1000 for cluster input.
```

```
312kb required, 828438kB available.  Copying trajectories to memory.
Performing clustering.
Iteration step: 1
Iteration step: 2
[...]
Iteration step: 69
Successful. 50 clusters found.
Writing cluster centers to ascii file './discretized/clusterCenters.ascii'.
Performing assignment to clusters.
Writing discrete trajectory  'discretized/traj1.disctraj'.
Writing discrete trajectory  'discretized/traj2.disctraj'.
```

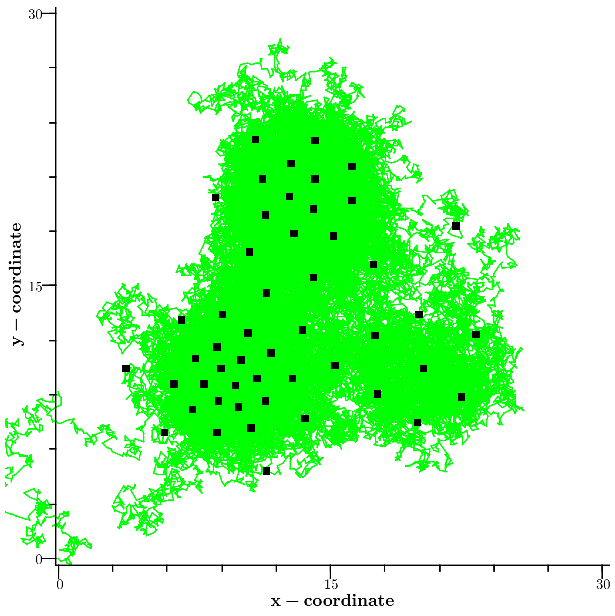

Figure 4: Display of 2d model trajectory and cluster centers (black squares) obtained from k-means clustering.

4.2. Connectivity

The connectivity `tool` tests which microstates are dynamically connected. It can output the largest connected subset of microstates, which is normally used to conduct the Markov model estimation on. Two microstates i and j are said to be connected, if trajectories exist that go from i to and j and from j to i. A set of states is said to be connected when all states in this set can be reached from all other states. It is important to have dynamical connection between the states used to build a Markov model. Not only are numerical burden existent, but more importantly: Only within a connected component one can calculate a well-defined stationary probability distribution, and this is a prerequisite for the correct functionality of many MSM algorithms.

If all microstates used are dynamically connected, the analysis can be continued without intervention.

For the discretized model trajectories of the tutorial, execution of the example command

```
Clustering/example$ ./mm_example_connectivity
```

which internally executes

```
mm_connectivity
-i discretized/traj1.disctraj discretized/traj2.disctraj
-o
```

The output (see below) reveals, that all microstates 0 to 49 are connected within one strong component / communicating class.

At this point we are fine and need not to consider the connection aspect any further. However, you are advised to check explicitly for connectivity of microstates when dealing with molecular dynamics simulation data. If more than one strong component occurs, the connectivity command detects the component with the highest number of microstates and writes this component out. Each of the EMMA commands, which uses microstates trajectories as input, especially the commands mm_timescales, mm_estimate and mm_chapman can be restricted to a subset of microstate by the option “-restrictToStates”.

```
Command line parameters valid.
Using 2 microstate trajectories for input:
discretized/traj1.disctraj
discretized/traj2.disctraj
Reading microstate trajectories.
Reading microstate trajectories finished.
Using sliding window sampling: (1 <-> 1+tau) (2 <-> 2+tau) ... (n <-> n+tau)
Strong components found in total: 1
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, ↩
24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, ↩
45, 46, 47, 48, 49}
```

4.3. Implied timescales

Once a dynamically connected set of microstates has been identified, it is possible to count the number of transitions occurring for any pair of microstates between times t and $t+\tau$. The resulting transition matrix $\mathbf{C}(\tau)$ is then converted into an estimate of the transition matrix $\mathbf{T}(\tau)$ which together with the microstate definition comprises the Markov model. However, until arriving at a $\mathbf{T}(\tau)$ which is useful for further analysis some tests must be conducted. The assumption that the jump process between microstates is Markovian is only approximately true. Its quality depends on two properties [40, 24]: Firstly, is the microstate definition fine enough? Secondly, is τ long enough?

In order to evaluate the test for implied timescales, to determine an appropriate τ and thus to test the quality of the Markov model, given the discretized trajectories, the following command needs to be executed

```
Clustering/example$ ./mm_example_timescales
```

Internally this command calls:

```
mm_timescales
-i discretized/traj1.disctraj discretized/traj2.disctraj
-sampling slidingwindow
-timestep 0.1
-lagtimes 1 2 3 5 10 20 50 100 200 500 1000
-neig 3
-o ./its/result.its
```

In the above command, the two discretized trajectories, which were created in the previous tutorial step 4.1, are taken as input. The “-sampling slidingwindow” option indicates that the discretized trajectory $z_\mu(t)$ is evaluated by counting transitions at $0 \rightarrow \tau,\ 1 \rightarrow \tau + 1, \ldots,\ n - \tau \rightarrow n$. Details on the counting modes are given in Section 8.5 and in [40][1], Section 8.5. The timestep option is only needed to present a proper, physical meaningful, output. Here, each time step of the trajectory corresponds to a time of 0.1 units, as specified during trajectory generation in Section 3. Furthermore, the lagtimes are set, for each of which the implied timescale is calculated.

The number of eigenvalues (-neig) sets how many (dominated) eigenvalues are considered. This number should be plus 1 the number of implied timescales desired, since the first eigenvalue, also called Perron eigenvector, corresponds to the stationary process with an infinitely large timescale. By invoking the

[1]online available at http://publications.mi.fu-berlin.de/944/

implied timescales command, an appropriate timescale has been determined, where the Markov model does not suffer from non-Markovianity. This lagtime τ, where the implied timescale is becoming nearly constant, is chosen to construct a Markov model, $T(\tau)$, in the next tutorial step.

The output after the execution of mm_timescales looks similar to the output below and is written to a file. A plot of the resulting implied timescales is shown in Figure 5).

```
Command line parameters valid.
1,2,3,5,10,20,50,100,200,500,1000
Writing out its result file './its/result.its'.
0.1 39.50737060984496 16.017638305505443
0.2 56.21277745404773 22.76737152466685
0.3 69.26151098799545 27.8235891807563
0.5 89.94370429637985 36.004840722332716
1.0 128.71891129899757 51.68936786031137
2.0 181.6785451346952 73.04776641968233
5.0 269.04011647320135 111.8660116985601
10.0 328.87568512113955 141.61474716730407
20.0 370.9098068045524 166.7593724367217
50.0 402.46733412661604 181.4473432890914
100.0 408.4191706409627 190.69735215906582
```

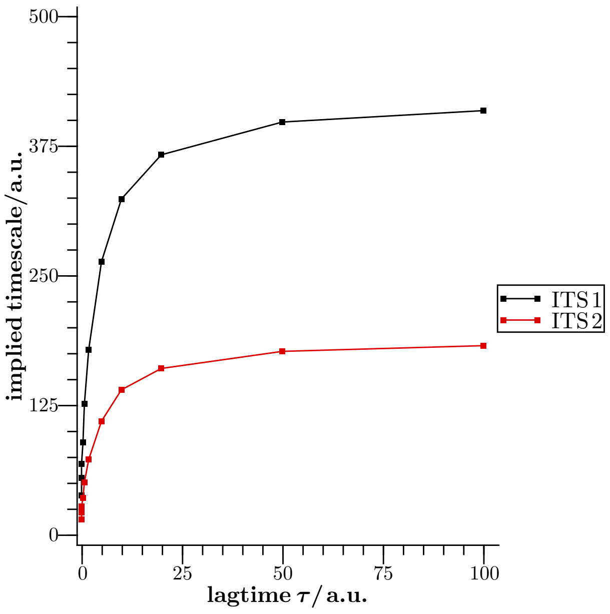

Figure 5: The implied timescales for different absolute lagtimes τ. Convergence of the implied timescales, meaning that the timescales become nearly constant when varying τ, is achieved for a time of 80.0.

4.4. Transition Matrix Estimation

The next step consists in the estimation of the transition matrix for an appropriate lagtime τ. Using the implied timescales test, an absolute lagtime of 80.0 (corresponds to 800 timesteps at a time of 0.1 between each frame) was estimated in the previous tutorial step to be large enough to construct a Markov model. Remember, that the Markovian property, which expresses that the system does not suffer from memory effects, must be fulfilled. Furthermore, note that the Markov model still needs to be validated later on using the Chapman-Kolmogorov test.

Next, we construct the Markov model for the lagtime $\tau = 80$. This consists of two steps

- construction of the count matrix and
- construction of the transition matrix.

These two steps are done by the command mm_estimate.[2]The transition matrix can be estimated, once the discrete trajectories are given and an appropriate lagtime has been determined. The next tutorial step is the execution of the command

```
Clustering/example$ ./mm_example_estimate
```

which internally calls:

```
mm_estimate
-i discretized/traj1.disctraj discretized/traj2.disctraj
-lagtime 800
-sampling slidingwindow
-outputtransitionmatrix matrix/transitionmatrixOf_Traj1_Traj2.ascii
```

The option “-i” specifies the discretized input trajectories. The option “-lagtime” sets the lagtime, which has been determined beforehand by the implied timescales calculation. The sampling method which determines the way, how the count matrix is constructed, has been selected as “slidingwindow”. You are referred to Section 8.5 for further details on the counting mode. The final transition matrix is written to the file “matrix/transitionmatrixOf_Traj1_Traj2.ascii”.

Within the steps of the tutorial and due to the good nature of our 2d model trajectory it is not essential to consider the following aspects of microstate connectivity, statistical problems and reversibility now. But even though none of these aspects is problematic for our model trajectory, which is sufficiently long to avoid several statistical issues and second, does not suffer from being in a non-equilibrium state, the above aspects become soon very evident when dealing with “real” molecular dynamics data. Here, only references to possible strategies to solving the upcoming problems are given: the non-connectivity of microstates can be circumvented by using the mm_connectivity command to discover connected sets of microstates which can be used in commands, which depend on microstates trajectories. The problems of insufficient statistics, e.g. avoid numerical problems in scenarios with little data, is a to use an appropriate prior to overcome this problems. Last, the burden of estimating reversible transition matrices, which are transition matrices fulfilling the detailed balance condition, is overcome by using the “-reversible” transition matrix option, which finds the maximum likelihood transition matrix **T** under the constraints of detailed balance. These options are just mentioned here, but are discuss in detail in Section 4.4.

The estimated output after execution looks similar to this output:

```
Command line parameters valid.
All parameters fine.
Reading trajectories.
Reading trajectories finished.
Writing transition matrix in sparse format to file
  'matrix/transitionmatrixOf_Traj1_Traj2.ascii'.
```

The constructed transition matrix is written (in sparse matrix format) to a file, it has dimension $50x50$, according to the number of states. The entry (i, j) of the transition matrix denotes the probability of going from state i to state j.

5. Metastability - Lumping microstates

5.1. Perron-Cluster-Cluster-Analysis (PCCA)

Based on the transition matrix one can calculate the metastable sets by means of improved PCCA clustering [50, 37]. In our example PCCA is executed by the command

```
Clustering/example$ ./mm_example_pcca
```

which internally calls

[2]This behavior has changed from v1.1 to v1.2: Emma v1.1 offered the commands mm_countmatrixEstimation and mm_transitionmatrixEstimation. From Emma v1.2 on,. both commands are unified in the command mm_estimate.

```
mm_pcca
-inputtransitionmatrix matrix/transitionmatrixOf_Traj1_Traj2.ascii
-nclusters 3
-ofuzzy pcca/fuzzy.ms
-ocrisp pcca/crisp.ms
-osets pcca/sets.ascii
```

Here, the option "-nclusters" specifies the number of metastable states to determine. Since our potential landscape consists of 3 basins, we also expect 3 metastable states.

In general, there are different strategies of choosing the number of clusters. One is often interested in the processes that occur slower than a certain threshold (e.g. nanoseconds or microseconds timescale). Then, the number of clusters should be set to the number of implied timescales at the chosen lagtime τ which lie above this threshold. In some systems, the intrinsic structure of implied timescales may also suggest a value of the number of clusters ("-nclusters"): If a large gap exists between timescale n and timescale $n+1$, then n is a useful setting for nclusters. Note that PCCA is used here only as a tool for analyzing and illustrating the essential features of a Markov model, and not to do calculations. Therefore, there is no "wrong" or "right" setting for "-nclusters", but instead it is only determined by the interest of the user.

```
Command line parameters valid.
Sparse matrix with dimension ( 50 x 50 ) read ↩
successfully from file 'matrix/transitionmatrixOf_Traj1_Traj2.ascii'.
Fuzzy cluster assignments written to file 'pcca/fuzzy.ms'.
Crisp cluster assignments written to file 'pcca/fuzzy.ms'.
Metastable sets definition written to file 'pcca/sets.ascii'.
```

The result file "sets.ascii" contains three lines (one for each metastable state), telling us, which microstates belong to the appropriate metastable state in that line:

```
7 9 10 11 12 17 22 26 27 30 31 34 35 38 41 44 45
2 5 15 18 21 36 39 47
0 1 3 4 6 8 13 14 16 19 20 23 24 25 28 29 32 33 37 40 42 43 46 48 49
```

A graphical representation of that calculated PCCA cluster assignment is given in Figure 6.

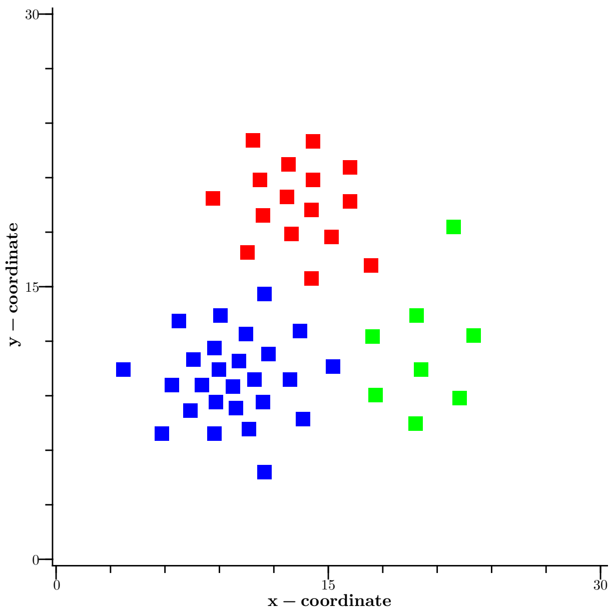

Figure 6: Cluster assignment to metastable states determined by kinetic clustering (PCCA). Three metastable sets (red, blue, green) are visible, each consisting out of approx. 10 microstates.

6. Markov Model Validation

6.1. Chapman-Kolmogorov Test

Using the Chapman-Kolmogorov test we can test the quality of the Markov model by monitoring the relaxation of probability out of a number of predefined sets [40]. This can be done for both the Markov model and the simulation data directly, and comparison of the results yields an indication whether the Markov model correctly predicts the long-time kinetics beyond the timescale τ used to parametrize the model.

Start the Chapman-Kolmogorov test with the following command:

```
Clustering/example$ ./mm_example_chapman
```

which calls:

```
mm_chapman
-i discretized/traj1.disctraj discretized/traj2.disctraj
-inputtransitionmatrix matrix/transitionmatrixOf_Traj1_Traj2.ascii
-dtTraj 0.1
-dtT 80.0
-sets chapman/sets.ascii
-kmax 10
-o chapman/result
```

In order to correctly relate the relaxation curves on the time axis,

- the time unit of the trajectory time step ("-dtTraj") and
- the time unit τ of the Markov model used for the test ("-dtT")

needs to be specified. The time between the frames is set to 0.1, given by "-dtT".

The test is done over a time range from 0 to $k_{max}\tau$ (here, $10 \cdot \tau = 800$ time units). The value of k_{max} is specified by the option "-kmax". The relaxation of population is monitored out of the sets defined by the "-sets" option. The user can define any sets of interest here (each set is defined by a single line, by a list of microstate indexes). For example, if one models protein folding or protein-ligand complex formation, one set might be the folded state or the complex state, respectively. For a general test that depends less on the subjectivity of the user we recommend using the sets identified by PCCA above. Since these sets are the most metastable sets of the system, this will be the hardest test - it may fail either due to a

- "bad" Markov model (discretization too coarse or lagtime τ too small) or due to
- insufficient statistics (the metastable sets are the ones with the fewest transitions between them).

Thus, this variant tests the weakest points of the Markov model and the Markov model is rather trustworthy if this test succeeds. The quantity of being tested is the following: for each set S_i chosen, one starts with an initial probability vector p_0 with a total probability of 1 on the set S_i and locally distributed according to the starting distribution chosen (in the example no option is given, so the stationary distribution of the input transition matrix is used). One then compares how much probability the two propagations of: $p_0 \cdot \hat{T}(\tau)^k$ (Markov model) and $p_0 \cdot \hat{T}(k\tau)$ (direct trajectory estimation) have on the sets for all times $k\tau$. Both probabilities should converge to the stationary probability of that set, i.e. both are expected to be approximately equal at long enough times $k\tau$. The Markov model is only a good model of the kinetics if the test succeeds at times larger than τ and smaller than this convergence time.

```
Command line parameters valid.
Sparse matrix with dimension ( 50 x 50 ) read ↩
successfully from file './matrix/transitionmatrixOf_Traj1_Traj2.ascii'.
Calculating stationary distribution.
Stationary distribution (calculated): ( 0.0202 ..... )
Loading discrete trajectories.
Loading discrete trajectories: done.
Reading sets from file './pcca/sets.ascii'.
Using trajectory at lag steps: 0 800 1600 2400 3200 4000 4800 5600 6400 7200 8000
```

```
Relaxing trajectory.
  Lagtime: 800
    Set: 0
    Set: 1
    Set: 2
    [ ... ]
Relaxing trajectory: done.
Relaxing transition matrix.
Relaxing transition matrix: done.

0.0 0.9999999999999999 1.0 0.0
80.0 0.860142591743427 0.8631694901239807 0.003528702196760236
160.0 0.770089343931751 0.778288221489421 0.006031913490155252
240.0 0.6969697577850095 0.7060631832996931 0.008101863060554474
320.0 0.6377816144085355 0.646027062671976 0.0098209689876617l
400.0 0.5898054630192 0.5941210102500002 0.011276660371134718
480.0 0.5508726082120141 0.5518770594990303 0.012511257167745999
560.0 0.5192499623572143 0.515578956227767 0.013581867743357704
640.0 0.493546625663306 0.48675981350332487 0.014523118864205422
720.0 0.4726428152831506 0.4655730202420918 0.015374549385603918
800.0 0.4556347577639292 0.4480369615332978 0.01615849167134321

[...]
```

A plot of the calculated relaxations is given in Figure 7.

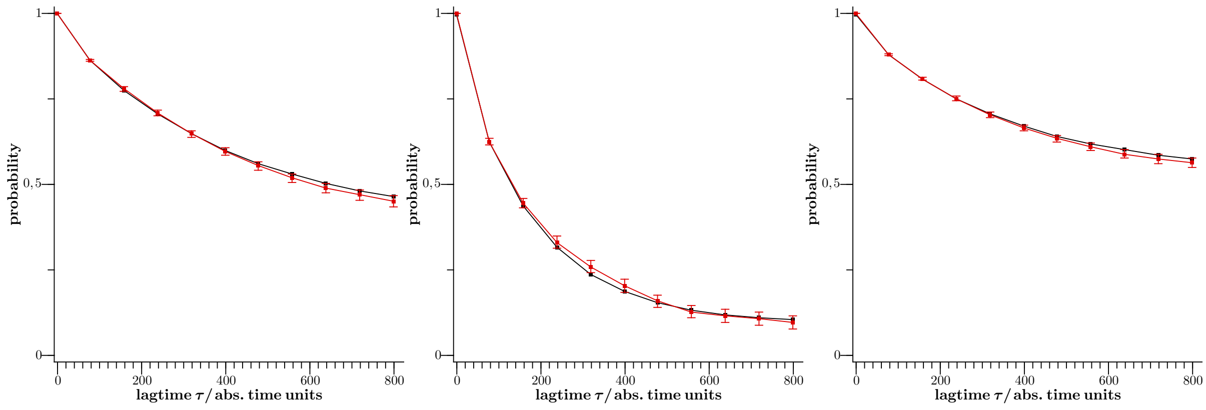

Figure 7: The propagation of the three different metastable determined by PCCA. Set $S_0 = \{7, 9, 10, 11, 12, 17, 22, 26, 27, 30, 31, 34, 35, 38, 41, 44, 45\}$, $S_1 = \{2, 5, 15, 18, 21, 36, 39, 47\}$ and $S_2 = \{0, 1, 3, 4, 6, 8, 13, 14, 16, 19, 20, 23, 24, 25, 28, 29, 32, 33, 37, 40, 42, 43, 46, 48, 49\}$. The red curve displays the relaxation of the transition according to $p_{S_i} \cdot \hat{T}(\tau)^k$. The blue curve displays the relaxation of the trajectory according to $p_{S_i} \cdot \hat{T}(k\tau)$.

7. Markov Model Analysis

7.1. Transition path theory (TPT)

Transition path theory allows to analyze the essential statistical features of the reactive transitions between two chosen subsets A and B. Especially when the dynamics are metastable, thus defining a natural partition into long-living sub-states, characterizing the set of transition pathways between two chosen subsets may provide a satisfactory picture of the process. Examples are protein folding where A may be the unfolded and B the native states ([38], [47]), as well as protein-ligand binding, where A is the dissociated set of states and B the bound complex ([23]).

The details of TPT are illustrated in Section 8.9.

Within this tutorial step we apply TPT to our 2-dimensional model system after having it characterized by a transition matrix between microstate clusters. This transition matrix has been shown to be a good model of the long-time kinetics of the system (see above).

The TPT algorithm expects the sets A and B to be chosen. The sets A and B are selected here for demonstration purposes to those microstates that belong to two of the metastable states obtained from PCCA clustering. The microstates assigned to metastable sets have been written to the file "pcca/sets.ascii" within the last step of this tutorial (see 5.1).

Please open that file containing the sets of PCCA with an editor. This file will have three lines, where the i-th line contains the microstates belonging to the i-th metastable state.

In order to identify, which line to copy and thus to determine, where the microstate are "located" in the energy landscape, you are referred to the file "discretized/clusterCenters.ascii", which was created in the 2nd step - Clustering of the tutorial.

Select the line, which contains the microstates for your set A, copy and place it into a new file "tpt/setA.ascii". The "tpt"-folder already contains a templateSetA.ascii, which shows, how your file should look like. The content of the templateSetA.ascii is

```
7 14 19 20 25 43 45 46
```

Select a second line from the "sets.ascii" representing your set B and copy that into a new file "tpt/setB.ascii" analogously.

After having created the two files, we can now start TPT via

```
Clustering/example$ ./mm_example_tpt
```

This command calls:

```
mm_tpt
-inputtransitionmatrix matrix/transitionmatrixOf_Traj1_Traj2.ascii
-seta tpt/setA.ascii
-setb tpt/setB.ascii
-oforwardcommittor tpt/forwardcommittor.ascii
-obackwardcommittor tpt/backwardcommittor.ascii
-onetflux tpt/netflux.ascii

-coarsegrain pcca/sets.ascii
-ocoarseforwardcommittor tpt/coarsedforwardcommittor.ascii
-ocoarsebackwardcommittor tpt/coarsedbackwardcommittor.ascii
-ocoarseflux tpt/coarsedflux.ascii
-ocoarsenetflux tpt/coarsednetflux.ascii
```

Option "inputtransitionmatrix" sets the transition matrix, options "seta" and "setb" define the sets A and B for TPT. The computed forward- and and backward-committor are written to the appropriate files, as well as the netflux.

```
Command line parameters valid.
Sparse matrix with dimension ( 50 x 50 ) read ↩
successfully from file './matrix/transitionmatrixOf_Traj1_Traj2.ascii'.
Set A: 2 5 15 18 21 36 39 47
Set B: 7 9 10 11 12 17 22 26 27 30 31 34 35 38 41 44 45
Computing forward committor.
Computing backward committor.
Computing flux and netflux.
Total flux    : 0.02617799402685038
Reaction rate : 0.09250997102642466
```

Having performed the above calculating one yields the forward committor, the backward committor and the flux and netflux. The total flux and the reaction are written to the console only. A visual representation of the committor values is given in Figure 8.

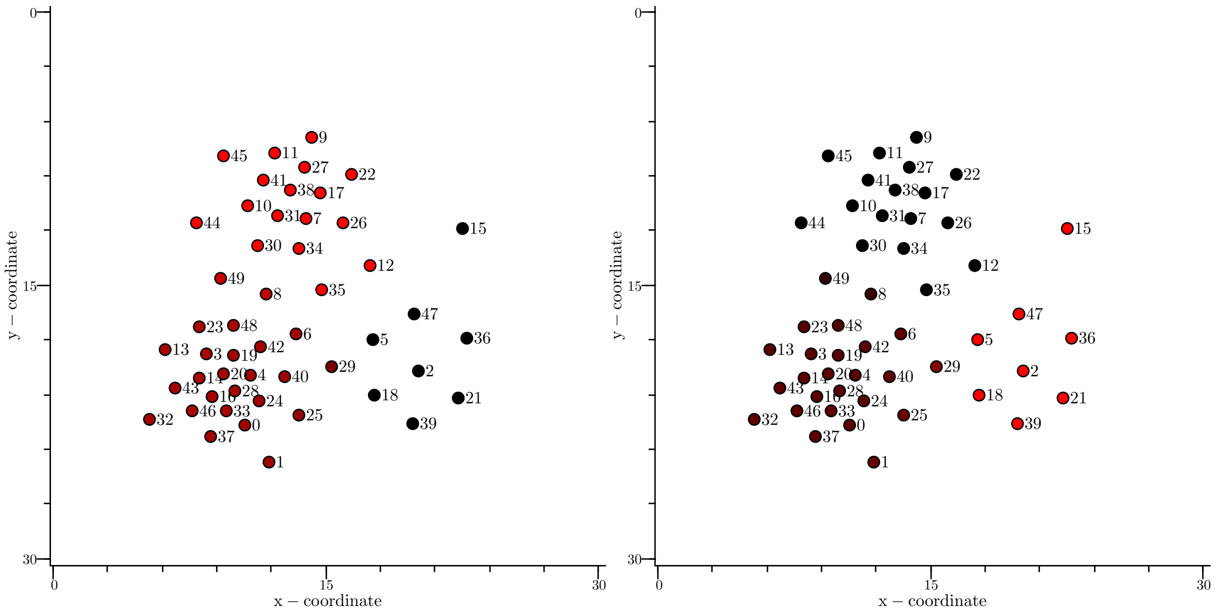
Figure 8: The forward q^+ and the backward committor q^- for each of the 50 clusters. The number right to the dots indicate the cluster number. Red dots indicate a value of 1.0, black dots a value of 0.0. Set $\mathbb{A}$ consists of the points in the bottom right, set $\mathbb{B}$ of the topmost clusters. .

In the second step of the TPT related example, the obtained forward- and backwardcommittor and the fluxes are coarse-grained, see Figure 9. This is done with the option "-coarsegrain" (see the command call above). Actually, the three sets obtained from PCCA are used in the example for coarse graining.

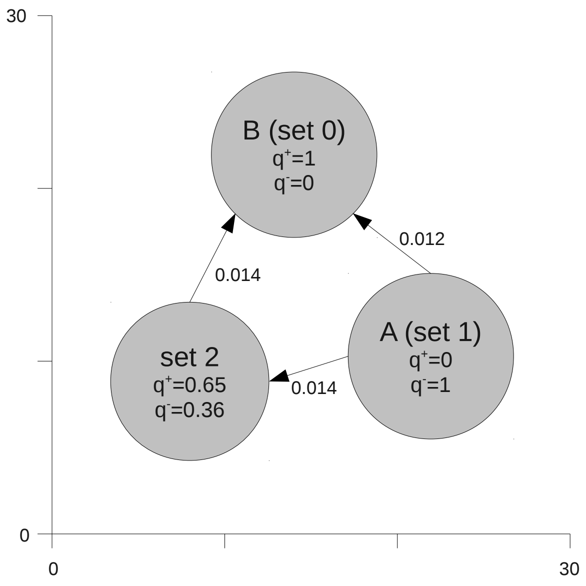

Figure 9: The coarse-grained sets 0 (set $\mathbb{B}$), 1 (set $\mathbb{A}$) and 2. For each node, the forward committor $q^+$and the backward committor q^- are given. Arrows indicate the flux between the nodes. .

7.2. Evaluating observables

An observation vector $\mathbf{o} \in \mathbb{R}^n$, where n is the number of microstates, tells which observation o_i is made, when the Markov chain is in microstate μ_i, that is $\mathbf{o} = (o_1, \ldots, o_i, ..., o_n)$. Here, for the sake of an illustrative example, we create two observation vectors $\mathbf{o}_1$ and $\mathbf{o}_2$. The vector $\mathbf{o}_1$ shows observation value 1, when the cluster center attributed to the microstate μ_i has an y-coordinate larger 15.0, and which shows an observation value of 0.0 for an y-coordinate below 15.0. Observation $\mathbf{o}_2$ does the same for the x-coordinate.

In order to create these artificial observations from trajectory / cluster center data, invoke the helper command

```
Clustering/example$ ./mm_example_createObservation
```

which creates the appropriate observation vectors in files "./observe/oObs1" and "./observe/oObs2".

After having created the observation within the tutorial two example commands are prepared:

```
Clustering/example$ ./mm_example_observables
```

which calls

```
mm_observables
-i matrix/transitionmatrixOf_Traj1_Traj2.ascii
-autocorrelation observe/obs1.ascii
-relax "1 2 5 10 50"
```

and computes the autocorrelation of the observation $\mathbf{o}_1$:

```
1.0      0.3324263002248801
2.0    0.2984933691943268
5.0      0.2301402516706576
10.0      0.17782130058131373
50.0      0.14700214159738803
100.0    0.14699069790336003
200.0    0.14699069731212533
```

Alternatively, the command

```
Clustering/example$ ./mm_example_observables2
```

calls

```
mm_observables
-i matrix/transitionmatrixOf_Traj1_Traj2.ascii
-crosscorrelation observe/obs1.ascii observe/obs2.ascii
-relax "1 2 5 10 50"
```

and computes the cross correlation of the observation $\mathbf{o}_1$ and $\mathbf{o}_2$:

```
1.0 0.06455918601233172
2.0 0.06677618291331067
5.0 0.06782919195267265
10.0 0.06552375520042573
50.0 0.06310203671080655
100.0 0.0631010317107933
200.0 0.06310103165887004
```

Part II. Command Line Tools for Markov Models

8. Command Overview

8.1. Trajectory Generation - mm_generate

Description

The command generates a two dimensional trajectory in a model toy potential according to Brownian / Smulokowski dynamics.

Brownian Dynamics in a potential is given by

$$\frac{d\mathbf{x}(t)}{dt} = -\frac{\nabla V(\mathbf{x}(t))}{\gamma m} + \sqrt{\frac{2k_BT}{\gamma m}}\frac{d\boldsymbol{\eta}(t)}{dt}. \tag{1}$$

with the Gaussian random force $\frac{d\boldsymbol{\eta}(t)}{dt}$, friction γ, mass m and thermal Energy kT. For simplicity, we set $\gamma m = 1$, and use the Euler-Maryama discretization methods, arriving at:

$$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) - \nabla V(\mathbf{x}(t)) + \sqrt{2k_BT}\frac{d\boldsymbol{\eta}(t)}{dt}. \tag{2}$$

In the long run, the dynamics samples from the stationary probability distribution $\exp(-V(\mathbf{x})/k_BT)$.

The command for the generation of model trajectories is restricted to the two-dimensional case. Thus positions $\mathbf{x}$ become (x, y). The potential $U(\mathbf{x}) = U(x, y)$ which is used is a sum of Gaussians:

$$U_i(x,y) = -I \cdot \exp\left(-\frac{(\mu_x - x)^2}{2\sigma_x} - \frac{(\mu_y - y)^2}{2\sigma_y}\right)$$

so the final potential is given by

$$U(x,y) = \sum_i U_i(x,y),$$

where I is the intensity of the Gaussian, μ_x and μ_y are the means, and σ_x and σ_y are the deviations according to the x and y axis. The appropriate covariances σ_{xy} or σ_{yx} are not used.

Synopsis

```
mm_generate
[-sigma <double>:{0.6}]
[-dt <double>:{0.1}]
[-randomseed <int>]
 -steps <int>
 -potdef <potential-filename>
 -start <double> <double>
 -o <trajectory-filename>
```

Options

-sigma <double>

Sets the noise intensity σ . Parameter is optional, default is 0.6
Example: -sigma 0.5

-dt <double>

Timestep dt. Parameter is optional. Default is 0.1.
Example: -dt 0.2

-randomseed <int>

Seed for initialization of random number generator. If omitted, uses system clock as random seed.
Example: -randomseed 241563

-steps <int>

Specify the number of steps, the model trajectory has.
Example: -steps 10000

-potdef <potential-filename>

Specify from which file the definition of the model potential is read.

Example: -potdef 3bassin.pot

-start <double> <double>

Sets, at which position the model trajectory is started.
Example: -start 0.25 0.5

-o <trajectory-filename>

Filename, where to store the generated model trajectory. Currently, output is limited to ascii file-format.
Example: -o modeltrajectory1.ascii

8.2. Clustering and discretization of trajectories - mm_discretize

Description

The discretization of data (e.g. trajectory frames in the above tutorial step) is performed by the two subsequent steps of clustering and partitioning:

- A clustering algorithm is used to determine a set of data points which serve as cluster centers. The input of the clustering is the set of data points (in our case trajectory frames), or a subset thereof.
- Assignment of data points (trajectory frames) to closest cluster centers, defining the so-called Voronoi partitioning.

When reasonably used, the algorithms provided here have a runtime of $O(nk)$ where n is the number of data points (trajectory snapshots) and k is the number of clusters. All the clustering algorithm provided here are geometrical clustering algorithms. The list of currently available clustering algorithms contains

- **K-Means**: $\underset{\mathbf{S}}{arg\ min} \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in S_i} \|\mathbf{x}_j - \boldsymbol{\mu}_i\|^2$
  - Assignment step: $S_i^{(t)} = \left\{ \mathbf{x}_j : \|\mathbf{x}_j - \mathbf{m}_i^{(t)}\| \leq \|\mathbf{x}_j - \mathbf{m}_{i^*}^{(t)}\| \text{ for all } i^* = 1, \ldots, k \right\}$
  - Update step: $\mathbf{m}_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{\mathbf{x}_j \in S_i^{(t)}} \mathbf{x}_j$

- **K-Centers**:
  Given a complete undirected graph $G = (V, E)$ with distances $d(vi, vj) \in N$ satisfying the triangle inequality, find a subset $S \subseteq V$ with $|S| = k$ while minimizing: $\max_{v\in V} \min_{s\in S} d(v, s)$.
  A simple greedy approximation algorithm that achieves an approximation factor of 2 builds S in k iterations. The first iteration chooses an arbitrary vertex and adds it to S. Each subsequent iteration chooses a vertex v for which $d(S, v)$ is maximized and adds v to S.
- **Regular spatial clustering**: A clustering method based on a minimum cut-off distance (wrt. to a given metric) between determined cluster centers:
  - The first data point $\mathbf{x}_0$ is selected as first cluster center μ_0. Variable k denotes the number of clusters found so far and is set to 1.
  - Index i goes from 1 to n.
    * If a data-point $\mathbf{x}_i$ is found, for which $d(\mathbf{x}_i, \mu_j) > d_{\min} \underset{j=0\ldots k}{\forall \mu_j}$ is fulfilled, then $\mathbf{x}_i$ becomes a new cluster center μ_j and k is incremented.
- **Regular temporal clustering**: Each i-th data point $\mathbf{x}_i$ of the trajectory is selected as cluster center. If the trajectory has length n, then $k = \lfloor \frac{n}{i} \rfloor$ cluster centers μ_j are selected.

Synopsis

```
mm_discretize
 -i (<trajectory-filename | trajectory-filenamepattern>+)
[-iformat [ xtc | dcd | ascii | {auto} ] ]

[-istepwidth <int>:{1}]

[
 -algorithm kmeans -clustercenters <int>
    [-metric {euclidian}]
    [-maxiterations <int>]
|
 -algorithm kcenters -clustercenters <int>
     -metric [ minrmsd | euclidian ]
|
 -algorithm regularspatial -dmin <double>
     -metric [ minrmsd | euclidian ]
|
 -algorithm regulartemporal -spacing <int>
     -metric [ minrmsd | euclidian ]
]

[-o <trajectory-outputdirectory>]

[-oclustercenters <clustercenter-filename>]
```

Options

-i <trajectory-filename | trajectory-filenamepattern>+

Specify input files (trajectories) used for clustering. This can be a space separated list of filenames. Additionally, it is possible to use specific wildcards to select multiple trajectories. Two wildcards are supported: # and * . # stands for chain of at least one digit like 253, * for a chain of arbitrary signs.

Attention: Please use double-quotes to prevent the shell from substituting filename wildcards, e.g. * will be substituted by a Bash-Shell by a list of all the names in the current directory. "*" instead is treated as the character * itself.

Example: A directory contains three files: traj001.xtc, traj002.xtc and exp4.xtc.

- -i traj001. traj002.traj selects directly these two trajectories for input
- -i "traj#.xtc" selects all trajectories with a chain of digits after "traj", that is traj001.xtc and traj002.xtc .
- -i "*.xtc" selects all trajectories which are in .xtc format, that is traj001.xtc, traj002.xtc and exp4.xtc.
- -i *.xtc selects all trajectories which are in .xtc format, but this time *.xtc is substituted automatically by your bash shell to -i traj001.xtc traj002.xtc exp4.xtc. Note that there is a usually a limit of the number of filenames the shell can list in this way, so the quote-options above may be necessary. (Explanation: the bash-shell substitutes *.abc by a list of space-separated filenames, that have the extension abc.)

-iformat [xtc | dcd | ascii | auto]

Specifies the input format of the trajectories if it can not properly determined by the file extension:

- xtc: Gromacs xtc trajectory format
- dcd: CHARMM or NAMD dcd trajectory format
- ascii: Simple tabulated text file with each line being one data point and each column one dimension.

Default is auto, which means, the trajectory type is determined by file extension. This parameter is optional.

-istepwidth int:1

By this option the input, consisting of trajectory frames, to the clustering is restricted. Instead of taking every single frame of each supplied trajectory as input for the clustering algorithm, only a subset of frames is selected. Let s be the stepwidth selected by the option. Then of each trajectory only each s-th frame is taken.

Given are n input trajectories $traj_j$ with $j \in J = \{0, \ldots, n\}$ and $traj_j : \mathbb{A}_j \to \mathbb{R}^n$, where set $\mathbb{A}_j = \{0, \ldots, \text{length}(traj_j)\}$ is the set of all indices for trajectory $traj_j$. Thus the i-th trajectory frame of the j-th trajectory $traj_j$ is $f_{i,j} = traj_j(i)$. The input set F of selected frames for clustering is then given by

$$F = \{f_{i,j} | i = k \cdot s \ \wedge \ i \in \mathbb{A}_j \ \forall k \in \mathbb{N}\}.$$

This options is most likely useful, if you are experiencing a message like "Using direct disc access, not sufficient working memory available. Clustering may be slow." in the beginning of the clustering process. Then try this option to reduce the number of frames, so that all your frames, which are used for clustering, fit into the main memory.

-algorithm kmeans

-clustercenters <int> Sets the number of clusters to use for k-means.

-metric euclidian Sets the used metrics for k-means. K-Means allows no other metric than Euclidean metric. This parameter is optional.

-maxiterations <int> Sets the maximum number of iterations. Setting the number to 0 results in computation until cluster centers do not change any more, this is referred to as convergence. Setting this value to other than 0 runs the algorithm a maximum number of iterations, but the algorithm is stopped if convergence is achieved before that number of steps. This parameter is optional.

-algorithm kcenters

-clustercenters <int> Sets the number of clusters to use for k-centers.

-metric [minrmsd | euclidian] Sets the used metrics for k-centers. The minimal RMSD metric or the standard Euclidean metric can be used.

-algorithm regularspatial

-dmin <double> Sets the parameter $d_{\min}$, see above.

-metric [minrmsd | euclidian] Sets the used metrics for k-centers. The minimal RMSD metric or the standard Euclidean metric can be used.

-algorithm regulartemporal

-spacing <int> Sets the spacing in frames to use. A value of s selects $\mathbf{x}_0, \mathbf{x}_s, \mathbf{x}_{2s} \dots$ as cluster centers

-metric [minrmsd | euclidian] Sets the used metrics for k-centers. The minimal RMSD metric or the standard Euclidean metric can be used.

-o <trajectory-outputdirectory>

Directory, where to place discretized trajectories. Discretized trajectories automatically get their original name, but the extension becomes ".disctraj".

-oclustercenters <clustercenter-filename>

File, to which the cluster centers are written.

8.3. Determining connectivity between microstates - mm_connectivity

Description

The program `mm_connectivity` tests which microstates are dynamically connected, thus which microstates build a communicating class. Internally a graph is constructed. This graph has edges between those microstates, which show transitions in the given microstate trajectories. The algorithm of Tarjan is then used to extract the strong connected components of the graph. The largest connected subset of microstates, which is usually used to conduct the Markov model estimation on, can be written out.

Synopsis

```
mm_discretize

 -i (<disc-trajectory-filename | disc-trajectory-filenamepattern>)+

 -o [<largestset-filename>]
```

Options

-i <disc-trajectory-filename | disc-trajectory-filenamepattern>+

Specify discrete trajectory input files (microstates trajectories) used.

-o [<largestset-filename>]

This option specifies, how and where to output data. If option "-o" is present, but without an argument, then the number of strong components and the microstates belonging to each strong component will be written to the console output. Otherwise, if the argument is given, the largest component will be written to the file named via the value "largestSetFilename".

8.4. Calculation of Implied Timescales - mm_timescales

Description

This command is a combination of three commands, iteratively called for different lagtimes τ.

For each of a set of lagtimes τ_j, do

- Count matrix estimation for lagtime τ_j, this yields $C(\tau_j)$.
- Transition matrix estimation from count matrix: $T(\tau_j)$ is constructed from $C(\tau_j)$.
- Transition matrix analysis to determine eigenvalues $\lambda_i(\tau_j)$.

Output $t_i^*(\tau_j) = -\frac{\tau_i}{\ln \lambda_i(\tau)}$, where t_i^* is the characteristic timescale for mode i and lagtime τ_j.

Background

The eigenvalue / eigenvector pairs of a transition matrix indicate the elementary processes of the system. Here, an eigenvalue yields the implied rate or implied timescale of the corresponding process and the eigenvector bears the information between which states the corresponding process switches. The i^{th} implied timescale is related to the i^{th} eigenvalue via:

$$t_i^* = \frac{-\tau}{\ln[\lambda_i(\tau)]}.$$

If the process associated to eigenmode i is Markovian, then t_i^* is constant and thus independent of τ. For many processes, this is true only for some minimum timescale τ, after which the memory pertaining to the inter-state dynamics has disappeared. Monitoring the slowest timescales t_i^* as a function of τ allows an adequate lagtime τ to be chosen, indicating an approximately Markovian model $T(\tau)$.

The general estimation of the transition matrix T is on the one hand a basic building block for the calculation of the implied timescales. On the other hand, only the implied timescales allow to determine an appropriate τ to construct a "valid" Markov model via the transition matrix $T(\tau)$. The relevant aspects: "connected microstates", "choosing a suitable count-matrix prior", and "reversible transition matrix estimation" are considered in Section 8.5.

Synopsis

```
mm_timescales
 -i (<disc-trajectory-filename | disc-trajectory-filenamepattern>)+
[-restrictToStates <largestset-filename>]

[-prior <double:{0.01}>]
[-sampling [ {slidingwindow} | lag ] ]
[-reversible]
-lagtimes (<int>)+
-neig <int>
[-timestep <double:{1.0}>]
[-o <timescales-filename>]
```

Options

-i (<disc-trajectory-filename | disc-trajectory-filenamepattern>)+

Specify input files (discrete trajectories) obtained from clustering. This can be a space separated list of filenames. Additionally, it is possible to use specific wildcards to selected multiple trajectories. Two wildcards are supported: # and * . # stands for chain of at least one digit like 253, * for a chain of arbitrary signs.

Note: please see the description at 8.2 for further information.

-restrictToStates <largestset-filename>

Restrict states to the states given in largest connected set "largestset". A detailed explanation is given in 8.5

-prior <double:0.01>

Use prior. More information about the prior is given in 8.5

-sampling [slidingwindows | lag]

For details see countmatrix estimation 8.5.

-reversible

Estimate a reversible transition matrix. See 8.5

-lagtimes (<int>)+

Set of lagtimes (space separated), to calculate implied timescales for.
Example: lagtimes 1 2 5 10 50 100

-neig <int:neig>

Total number of eigenvalues (modes) to analyze. The number of timescales computed is $neig - 1$.

-timestep <double:1.0>

The length of one lag time unit τ, such it is the time from one trajectory frame to the next trajectory frame. If not set explicitly, it defaults to 1.0.

-o <timescales-filename>

File to write implied timescales to. The output file contains ascii and is column oriented. The first column contains the absolute lagtime, that is the productlagtime $\cdot$ timestep. Columns 2 to $2 + neig - 1$ contain the $neig - 1$ implied timescales.

8.5. Transition Matrix Estimation - mm_estimate

Description

Estimate a count matrix **C** from a given set of discrete trajectories, **C** has dimension $n \times n$ where n is the number of clusters used to discretize the data. From this count matrix **C** a transition matrix **T** is estimated, reversible of option "-reversible" is present. This transition matrix **T** sets up the foundation of the constructed Markov model.

Background

The countmatrix **C** is basically created by counting the number of transition from microstate i to state j, building the matrix entry c_{ij}. Different counting methods are available: slidingwindow, which creates a high number of counts, but neglects the fact of statistical independence, and the lag count mode, which count transition from $t(0)$ to $t(\tau)$, $t(\tau)$ to $t(2\tau)$ and so forth. That way, a count matrix **C** is determined.

The user can choose to add a prior matrix $\mathbf{C}^p$ to avoid numerical problems with states that were rarely visited or never left [32, 18]:

$$c_{ij} = c_{ij}^p + c_{ij}^o.$$

The purpose of the prior $\mathbf{C}^p$ is to avoid numerical problems in scenarios with little data. It adds a bias which vanishes when much data is accumulated as then the count matrix will be dominated by $\mathbf{C}^o$. However, the bias should be chosen as small as possible and as large as necessary to ensure numerical stability. As shown in [18, **?**], a useful prior is the neighbor prior: $c_{ij}^p = \alpha$ when $c_{ij}^o(1) + c_{ji}^o(1) > 0$, i.e. when ever two states have been seen adjacently in the trajectory, they are considered as neighbors and a small pseudo count is added to both in order to make sure that the stationary probability is nonzero in all states.

It is intuitively clear that in the limit of an infinitely long trajectory, the elements of the true transition matrix are given by the trivial estimator:

$$\hat{T}_{ij}(\tau) = \frac{c_{ij}}{\sum_k c_{ik}} = \frac{c_{ij}}{c_i}. \tag{3}$$

this is the maximum likelihood transition matrix using the count matrix $\mathbf{C}$.

But when unbiased molecular dynamics trajectories are used the system is assumed to be in equilibrium, and in this situation we expect detailed balance to hold:

$$\pi_i T_{ij} = \pi_j T_{ji}, \tag{4}$$

however the estimated (with the trivial estimator) matrix will generally not fulfill $\pi_i T_{ij} = \pi_j T_{ji}$, simply as a result of statistical deviations from detailed balance for finite amounts of data. Therefore, in cases where the dynamics are in equilibrium it is useful to *enforce* detailed balance on the matrix. A suitable approach is to use an estimator which finds the maximum likelihood $\hat{\mathbf{T}}$ under the constraints (4) [4, 40]. This optimal reversible estimator as described in [40] is available in EMMA via the option "-reversible". It is applicable for the command mm_estimation and mm_timescales.

Synopsis

```
mm_estimate
-i (<disc-trajectory-filename | disc-trajectory-filenamepattern>)+
[-restrictToStates <largestset-filename>]
[-prior <double:{0.01}>]
[-reversible]
[-lagtime <int>:{1}]
[-sampling [ {slidingwindow} | lag ]]
[-outputtransitionmatrix <transitionmatrix-filename>]
```

Options

-i (<disc-trajectory-filename | disc-trajectory-filenamepattern>)+

Specify input files (discrete trajectories) that were obtained from clustering. This can be a space separated list of filenames. Additionally, it is possible to use specific wildcards to selected multiple trajectories. Two wildcards are supported: # and * . # stands for chain of at least one digit like 253, * for a chain of arbitrary signs.

Note: fur further details, please see Section8.2 to get further information on input parameter wildcards.

-restrictToStates <largestset-filename>

Restrict states to the states given in largest connected set "largestset". A detailed explanation is given in 8.5

-prior <double:0.01>

Use a prior of the count matrix. See 8.5

-reversible

Estimate a reversible transition matrix. See

-lagtime <int>:1

Lagtime used for construction.

-sampling [slidingwindow | lag]

In order to estimate the number of transitions between states in the trajectory, several methods are available (see Figure 10 for graphical illustration). The discrete states of the trajectory are given wrt. $s(t)$, with t the time. A characteristic function $\chi_i(s)$ is zero, if $i \neq j$ and is 1 if $i = j$:
lag: $C_{ij} = \sum_{k=1}^{\frac{n-\tau}{\tau}} \chi_i(s(t)) \cdot \chi_j(s(k \cdot \tau))$.
slidingwindow: $C_{ij} = \sum_{t=1}^{n-\tau} \chi_i(s(t)) \cdot \chi_j(s(t+\tau))$.

-lagtime <int>:1

Lagtime used for construction.

-outputtransitionmatrix <transitionmatrix-filename>

File to write constructed transition matrix to.

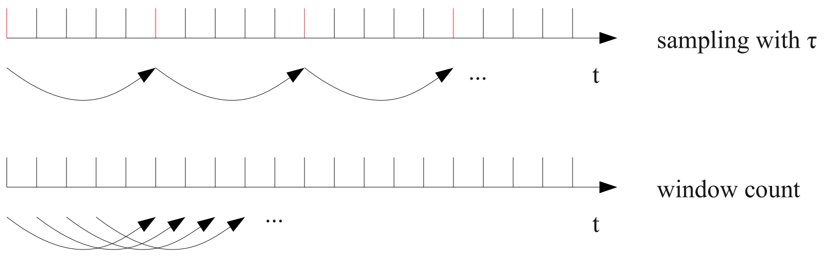

Figure 10: Illustration of different counting modes. Top view: counting mode “lag” . Bottom view: counting mode: “sliding window” .

8.6. Perron-Cluster-Cluster-Analysis (PCCA) - mm_pcca

PCCA is a method for the determination of metastable states based on a transition matrix ([10, 49, 43, 37]). The metastable sets are those sets of microstates, within which equilibration is most rapid and between which transitions are most rare. It is therefore a kinetic clustering method. By PCCA each of the microstates is assigned to one of 1,..,C clusters or metastable states. The assignment of macrostates to microstates is called the membership-assignment.

We are interested in finding m metastable sets from an $n \times n$ transition matrix. The membership matrix $\chi \in \mathbb{R}^{m \times n}$ indicates by its elements χ_{ij} to what degree each microstate j belongs to metastable set i, where

$$\sum_i \chi_{ij} = 1 \, \forall i.$$

The membership matrixχ is calculated with an approximate PCCA+ method (see [10, 49, 43, 37] for the mathematical details). Here, this is done, by calculating the set of $m-1$ right eigenvectors $\psi_2, ..., \psi_m$ of T (the stationary eigenvector ψ_1 is constant and irrelevant for this analysis). In this $m-1$-dimensional eigenvector space, each microstate has a coordinate $\psi^{(i)} = (\psi_{2,i}, ..., \psi_{m,i})$. The microstates lie in a simplex with m vertices $v_1, ..., v_m$ which represent the metastable centers of the metastable sets. These vertices are determined by the following (approximate) procedure:

1. Find v_1 and v_2 by searching the microstate pair with $|\psi^{(v_1)} - \psi^{(v_2)}| \rightarrow \max$.
2. For all $k = 3...m$: Find v_k by searching the microstate with $\sum_{i=1}^{k-1} |\psi^{(v_k)} - \psi^{(v_i)}|$

Subsequently, each microstate is assigned a membership vector $\chi_{\cdot,j}$ by calculating the convex coordinates to the vertices $v_1, ..., v_m$. In order to obtain a clear-cut assignment of microstates to metastable states, we find the metastable state assigned to microstate j, $m(j)$, by using simply the maximum membership:

$$m(j) = \arg\max_i \chi_{ij}.$$

Synopsis

```
mm_pcca
[
  -inputtransitionmatrix <transitionmatrix-filename>
|
  -inputeigenvectors <eigenvectors-filename>
]

 -nclusters <int>

[ -ofuzzy <fuzzyassignment-filename>]
[ -ocrisp <crispassignment-filename>]
[ -osets <microstateassignment-filename>]
```

Options

[-inputtransitionmatrix <transitionmatrix-filename> | -inputeigenvectors <eigenvectors-filename>] (alternative)

Specify input, which can be either a transition matrix or a set of eigenvectors in the form as they are generated by mm_transitionmatrixAnalysis. If the input is set to a transition matrix, the eigenvectors, required for PCCA, are computed internally.

-nclusters <int>

Number of clusters used for PCCA algorithm.

[-ofuzzy <fuzzyassignment-filename>] (optional)

Output the assignment of microstates to those states determined by PCCA. “Fuzzy” means, that one microstate is not strictly assigned to one PCCA state, but having a membership with each metastable state.

[-ocrisp <crispassignment-filename>] (optional)

Output the assignment of microstate to PCCA states. Each microstate is exactly assigned to one PCCA state.

[-osets <microstateassignment-filename>] (optional)

Output for every pcca state the microstates belonging to that pcca state. This contains the same assignment information as with -ocrisp, but differently formatted.

8.7. Chapman-Kolmogorov Test - mm_chapman

Description

Conducts a test of the Markov model $\hat{T}(\tau)$. This is done by comparing the long-time propagation kinetics of $\hat{T}(\tau)$ with the data available from trajectories and checking whether these two are consistent. When successful the Chapman-Kolmogorov Test is a validation of the quality of the Markov model.

Background

A strong test of the Markov model is obtained by comparing its prediction of the long-time kinetics obtained by propagating the transition matrix $\hat{T}(\tau)$, which had been obtained for a certain lagtime τ, $k = 1 \dots k_{max}$ times to the direct estimation of $\hat{T}(k\tau)$, where k_{max} is limited by the length of the trajectories available. For an exactly Markovian dynamics, this test would fulfill the Chapman-Kolmogorov equality which we here attempt to fulfill approximately:

$$[\hat{T}(\tau)]^k \approx \hat{T}(k\tau),$$

where $\hat{T}(\tau)$ is the transition matrix estimated for the time series at lag time τ and $\hat{T}(k\tau)$ is the transition matrix which is estimated for the time series for lagtimes $k\tau$. The matrix $\hat{T}(\tau)$ is taken to the power of k to simulate a propagation of the system for time $k \cdot \tau$.

But since it is difficult to compare the matrices $\hat{T}(\tau)^k$ and $\hat{T}(k\tau)$ directly, we instead choose to apply a distribution p_0 to $\hat{T}(\tau)^k$ while we are estimating the quantity $p_0\hat{T}(k\tau)$ directly from the trajectory, yielding:

$$p_0 \cdot [\hat{T}(\tau)]^k \approx p_0 \cdot \hat{T}(k\tau). \tag{5}$$

Here, p_0 is chosen such that it sums up to 1 on a set S_i of interest while being 0 otherwise. Within S_i, $p_0 \propto \rho$ where ρ is the a set of weights, which usually is taken to be the stationary distribution of T. The test is visualized by plotting the total probability on each set S_i tested over times $k\tau$, and checking whether Markov model and direct calculation agree within error (see [40] [3] for further details).

Synopsis

```
mm_chapman
 -i (<disc-trajectory-filename | disc-trajectory-filenamepattern>)+
[-restrictToStates <largestset-filename>]
 -inputtransitionmatrix <transitionmatrix-filename>
[-randomsets [<int>:{1}] | -sets <sets-filename> ]
 -dtT <double>
 -dtTraj <double>
 -kmax <int>
```

Options

-i (<disc-trajectory-filename | disc-trajectory-filenamepattern>)+

Specify input files (discrete trajectories) used for the Chapman-Kolmogorov test.
Note: please see the description at 8.2 to get further information.

-restrictToStates <largestset-filename>

Restrict states to the states given in largest connected set “largestset”. A detailed explanation is given in 8.5

-inputtransitionmatrix <transitionmatrix-filename>

Transition matrix $\hat{T}(\tau)$ for lagtime τ, which is propagated wrt. to $\hat{T}(\tau)^k$.

-randomsets [<int>:1]

Use i random subsets S_i of the discrete state space with its n discrete states which are propagated independently. For each subset S_i the probability distribution p

-sets <sets-filename>

Instead of using -randomsets the subsets S_i can be specified manually in a file. Here, each row of the file represents the discrete states of the subset S_i. These sets may e.g. come from PCCA (see below).

-dtT <double>

Specifies the time of one lag unit τ (in order to align the Markov model and the trajectory estimate in time).

-dtTraj <double>

Specifies the time of one discrete trajectory timestep (in order to align the Markov model and the trajectory estimate in time).

[3]available online at http://publications.mi.fu-berlin.de/944/

-kmax <int>

The test is performed on the time window 0 to $k_{max}\tau$.

8.8. Transition Matrix Analysis - mm_transitionmatrixAnalysis

Description

This tool allows to determine the stationary distribution, the first n eigenvalues plus the corresponding left and right eigenvectors of a given transition matrix. The stationary distribution, eigenvectors and eigenvalues are printed to the console or written to files.

Synopsis

```
mm_transitionmatrixAnalysis
 -inputtransitionmatrix <transitionmatrix-filename>
[-nev <int>]
[-stationarydistribution [<stationarydistribution-filename>]]
[-eigenvalues [<eigenvalues-filename>]]
[-lefteigenvectors [<lefteigenvectors-filename>]]
[-righteigenvectors [<righteigenvectors-filename>]]
```

Options

-inputtransitionmatrix <transitionmatrix-filename>

File, to read transition matrix from.

-nev <int>

Sets the number of eigenvalues / eigenvectors to calculate. This option The value has to be

-stationarydistribution [<stationarydistribution-filename>]

Output eigenvalues to a separate ascii file.

-eigenvalues [<eigenvalues-filename>]

Calculate and output eigenvalues. The number of eigenvalues computed depends on the value set by -nev. If the outputfilename is given, the eigenvalues are written to an ascii file, otherwise they are written to the screen.

-lefteigenvectors [<lefteigenvectors-filename>]

Calculate and output left eigenvectors. If the outputfilename is given, the eigenvectors are written to an ascii file, otherwise they are written to the screen.

-righteigenvectors [<righteigenvectors-filename>]

Calculate and output right eigenvectors. If the outputfilename is given, the eigenvectors are written to an ascii file, otherwise they are written to the screen.

8.9. Transition path theory (TPT) - mm_tpt

Transition path theory allows us to analyze the essential statistical features of the reactive transitions between two chosen subsets A and B [46, 27, 28, 38], especially the set of transition pathways between A and B, their relative probabilities, the total $A \to B$ flux and the $A \to B$ rate.

The essential object needed to calculate statistics pertaining to the $A \to B$ reaction is the committor probability, also called splitting probability or probability of folding [14, 9, 3, 26, 2, 12, 16, 25]. The committor $q(x)$ is a state function that provides the probability at any state $x \in \Omega$ of next moving to B rather than to A under the action of the system dynamics. By definition, $q(x) = 0$ for $x \in A$ and $q(x) = 1$ for $x \in A$, while $q(x) \in [0,1]$ for all other states. The committor probability is calculated first by the tpt command.

The committor is useful for calculating a number of quantities. Firstly, all sets of constant committor probability in the state space Ω

$$I(q^*) = \{x \in \Omega \mid q(x) = q^*\},\ \forall q^* \in [0,1] \tag{6}$$

are hypersurfaces that partition the state space into the two disjoint subsets $I_A(q^*) = \{x \in \Omega \mid q(x) < q^*\}$ with $A \subset I_A(q^*)$, $\forall q^* > 0$ and $I_B(q^*) = \{x \in \Omega \mid q(x) > q^*\}$ with $B \subset I_B(q^*)$, $\forall q^* < 1$. The committor is thus a measure for the progress of a process or reaction, i.e. it is the ideal reaction coordinate for the process $A \to B$ [14, 2, 25]. Of special interest in this context is the isocommittor surface $I(0.5)$, which can be interpreted as the transition state ensemble in protein folding theory [39].

The transport properties from A to B can be be computed *via* transition path theory (TPT) [46, 28]. In particular, the reactive flux f_{ij} between two states i and j is given by

$$f_{ij} = \begin{cases} \pi_i q_i^- k_{ij} q_j^+ & i \neq j \\ 0 & i = j \end{cases} \tag{7}$$

for rate matrices [28], or

$$f_{ij}(\tau) = \pi_i q_i^- T_{ij}(\tau) q_j^+ \tag{8}$$

if the transition probability matrix is used [38]. Here, q^- is the backward committor which is the probability that of the two states set $\mathbb{A}$ has been visited last and not $\mathbb{B}$ which is equal to $1 - q^+$ if the dynamics is reversible. The reactive flux f_{ij} is proportional to the probability that a reactive trajectory, that is, a trajectory directly connecting $\mathbb{A}$ and $\mathbb{B}$, passes through the transition $i \to j$. The net transport through $i \to j$ is given by

$$f_{ij}^+ = \max\{f_{ij} - f_{ji}, 0\}, \tag{9}$$

which defines a network flow out of A and into B that can be decomposed into a set of $A \to B$ reaction pathways along with their probabilities [28, 46, 38]. Finally the total flux of the $A \to B$ reaction is given by

$$F_{AB} = \sum_{i \in \mathbb{A}, j \notin \mathbb{A}} f_{ij} = \sum_{i \in \mathbb{A}, j \notin \mathbb{A}} f_{ij}^+. \tag{10}$$

and the $A \to B$ reaction rate is given by:

$$k_{AB} = \frac{F_{AB}}{\sum_i \pi_i q_i^-}. \tag{11}$$

Synopsis

```
mm_tpt
 -inputtransitionmatrix <transitionmatrix-filename>
[-statdist <stationarydistribution-filename> ]
 -seta <filename>
 -setb <filename>

[ -oforwardcommittor <forwardcommittor-filename>]
[ -obackwardcommittor <backwardcommittor-filename>]
[ -oflux <flux-filename>]
[ -onetflux <netflux-filename>]

[-coarse <string:filename>
  [-ocoarseforwardcommittor <string:filename>]
  [-ocoarsebackwardcommittor <string:filename>]
  [-ocoarseflux <string:filename>]
  [-ocoarsenetflux <string:filename>]
  [-ocoarseseta <string:filename>]
  [-ocoarsesetb <string:filename>]
]
```

Options

-inputtransitionmatrix <transitionmatrix-filename>

Specifies the input transition matrix.

[-statdist <stationarydistribution-filename>] (optional)

Input file which specifies the stationary distribution, which is required by TPT. If this option is not provided, the stationary distribution is computed from the transition matrix.

-seta <filename>

Specifies file containing the definitions of states belonging to set A.

-setb <filename>

Specifies file containing the definitions of states belonging to set B.

[-oforwardcommittor <forwardcommittor-filename>] (optional)

Output the forward committor q^+ (in vector form) to a file.

[-obackwardcommittor <backwardcommittor-filename>] (optional)

Output the backward committor q^-(in vector form) to a file.

[-oflux <flux-filename>] (optional)

Output the flux (in matrix form) to a file.

[-onetflux <netflux-filename>] (optional)

Output the net flux (in matrix form) to a file. This is the main result.

[-coarsegrain <string:filename>] (optional)

Specifies to coarse-grain the forward-committor, the backward-committor, the flux and the net flux onto the defined sets. The set definitions are read from given file. This option directly affects all output.

For example, the metastable sets obtained from PCCA may be used here, resulting in a coarse-graining for the committors and fluxes onto the metastable sets (see [38]).

Here, coarse-graining effectively means:

- Given the committor q the coarse-grained-committor is obtained by the summation of all microstate committor values weighted by the stationary distribution: $q_S = \sum_{i \in S} \pi_i q_i$.
- Given a flux matrix f_{ij}, the coarse-grained flux matrix F_{IJ} is obtained by summation of all microstate fluxes between sets: $F_{IJ} = \sum_{i \in I} \sum_{j \in J} f_{ij}$ for $I \neq J$.

The committors and fluxes for the coarse-grained case can also be output to files. The following output options can only be used if the option “-coarsegrain” is present. The options differ only by the part “coarse” from the output options for the non coarse-grained case above.

[-ocoarsedforwardcommittor <string:filename>] (optional output, only available if “-coarsegrain” present)

Output the coarse-grained forward committor to a file.

[-ocoarsedbackwardcommittor <string:filename>] (optional output, only available if “-coarsegrain” present)

Output the coarse-grained backward committor to a file.

[-ocoarsedflux <string:filename>] (optional output, only available if "-coarsegrain" present)

Output the coarse-grained flux to a file.

[-ocoarsednetflux <string:filename>] (optional output, only available if "-coarsegrain" present)

Output the coarse-grained net flux to a file.

[-ocoarsedseta <string:filename>] (optional output, only available if "-coarsegrain" present)

Output the coarsed set A to a file.

[-ocoarsedsetb <string:filename>] (optional output, only available if "-coarsegrain" present)

Output the coarsed set B to a file.

8.10. Expectation values, Correlation functions, Fingerprints - mm_observables

Description

The mm_observables command allows to analyze MSMs in a way that allow comparison to experimental measurements. We here assume that a scalar observable a_i is defined for each microstate, resulting in a vector $\mathbf{a}$. This vector is stored in a file that containing the vector elements in the rows. Such observables may be any function of the molecular state, such as a fluorescence, a FRET efficiency, a distance, etc.

Background

Many experiments measure ensemble averages. Given a transition matrix $\mathbf{T}(\tau)$ with associated stationary distribution $\boldsymbol{\pi}$ and observable vector $\mathbf{a}$, the ensemble average can be calculated with the `-expectation` command. It is simply estimated by:

$$\mathbb{E}[a] = \sum_{i=1}^{n} \pi_i a_i. \tag{12}$$

The most interesting features of `mm_observables` however allow dynamical observables to be calculated such as perturbation-relaxation and correlation curves as they can also be measured in kinetic experiments. Importantly, `mm_observables` can help to interpret these curves in terms of *dynamical fingerprints* which can be dissected into dynamical features that are associated with individual relaxation timescales and structural rearrangement processes [36, 20]. Here, we differentiate between two types of kinetic experiments: perturbation and correlation experiments.

In perturbation experiments, the ensemble average of an observable is tracked over time while the ensemble relaxes from some perturbed or triggered initial state at time $t = 0$ towards its stationary distribution. The initial trigger may consist of e.g. a jump in temperature [19, 41], pressure [13], a change in the chemical environment [7] or a photoflash [48, 42, 6]. Such a time-dependent ensemble averages can be calculated with the commands `-perturbation` and `-relax` *via*:

$$\mathbb{E}[a(k\tau)]_{\mathbf{p}_0} = \sum_{i=1}^{n}\sum_{j=1}^{n} p_{0,i} T_{ij}(k\tau) a_j \tag{13}$$

A special perturbation experiment is the temperature-jump experiment where an ensemble is prepared at temperature T_1 at $t < 0$ and is then suddenly changed to temperature T_2 at $t = 0$. The system is kept at T_2 and relaxes from its old stationary distribution $\mathbf{p}_0 = \boldsymbol{\pi}(T_1)$ to its new stationary distribution $\boldsymbol{\pi}(T_2)$. In cases where simulations have been conducted at both temperatures T_1 and T_2, the stationary distribution at T_1 can be straightforwardly plugged in as initial distribution to a perturbation experiment using the transition matrix at T_2 and the result can be calculated with the `-perturbation` command. If only a single-temperature simulation has been made at temperature T_2 we can still estimate the temperature-jump relaxation curve with the command `-Tjump`. This makes the assumption that the microstates clustering is fine enough such that

$$\begin{aligned}\pi_i(T_1) &\approx& Z_1^{-1}\exp(-E_i/k_BT_1)\\ \pi_i(T_2) &\approx& Z_1^{-1}\exp(-E_i/k_BT_1)\end{aligned}$$

with some common energy E_i. Based on this assumption, the initial distribution $\mathbf{p}_0$ for an initial temperature T_1 that is close to the target temperature T_2 is approximated, and the temperature-jump relaxation curve is calculated via Eq. 13.

A very common type of kinetic experiments are correlation experiments. Correlation experiments may either be realized through scattering techniques such as inelastic neutron scattering [11], or *via* low concentration or single molecule experiments accumulating auto- or cross-correlations of fluctuations, e.g. correlation spectroscopy of the fluorescence intensity [22, 31, 29, 45, 15] or Förster resonance energy transfer efficiency [21, 30]. The `-autocorrelation` command calculates equilibrium autocorrelations of observables $\mathbf{a}$ via:

$$\mathbb{E}[a(0)\,a(k\tau)]_{\boldsymbol{\pi}} = \sum_{i=1}^{n}\sum_{j=1}^{n} a_i\pi_i T_{ij}(k\tau)a_j \tag{14}$$

and the `-crosscorrelation` command calculates the cross-correlation between two observables $\mathbf{a}$ and $\mathbf{b}$ via:

$$\mathbb{E}[a(0)\,b(k\tau)]_{\boldsymbol{\pi}} = \sum_{i=1}^{n}\sum_{j=1}^{n} a_i\pi_i T_{ij}(k\tau)b_j \tag{15}$$

Instead of directly printing the perturbation-relaxation or correlation curve via `-relax`, one can output the dynamical fingerprint of a perturbation or correlation experiment via the `-fingerprint` command. As explained in detail in [36, 20], the long-timescale part of Eqs. (12), (13) and (14) can each be written in the form

$$\mathbb{E}(...,\, k\tau) = \sum_{i=1}^{m}\gamma_i \exp\left(-\frac{k\tau}{t_i^*}\right)$$

where γ_i^* is the i-th implied timescale and γ_i is an amplitude that depends on the specific experiment conducted. The amplitudes γ_i are derived in [36, 20] and can be calculated from scalar products of initial or stationary probability distributions, properly normalized left or right eigenvectors $\mathbf{l}_i$, $\mathbf{r}_i$ and observable vectors $\mathbf{a}$, $\mathbf{b}$:

$$\begin{aligned}\gamma_i^{\text{perturbation}} &=& \langle\mathbf{p}_0,\mathbf{r}_i\rangle\langle\mathbf{a},\mathbf{l}_i\rangle\\ \gamma_i^{\text{autocorrelation}} &=& \langle\mathbf{a},\mathbf{l}_i\rangle^2\\ \gamma_i^{\text{crosscorrelation}} &=& \langle\mathbf{a},\mathbf{l}_i\rangle\langle\mathbf{b},\mathbf{l}_i\rangle.\end{aligned}$$

The command `-fingerprint` outputs the amplitudes γ_i and timescales t_i^* for all spectral components with positive eigenvalues.

Synopsis

```
mm_observables
 -i <string:transitionmatrixfilename> [command] [output-mode]

 -expectation <obs>
 -perturbation <p0> <obs>
 -Tjump <temp1> <temp2> <obs>
 -autocorrelation <obs>
 -crosscorrelation <obs1> <obs2>
```

```
-relax [<time-list> | <tmin> <tmax> <dt>]
-relaxspectral [<time-list> | <tmin> <tmax> <dt>]
-fingerprint [<time-list> | <tmin> <tmax> <dt>]
```

Commands

-expectation <obs>

Stationary expectation value of the observable. Output-mode is disabled.

-perturbation <p0> <obs>

Time-dependent expectation value after letting the system relax from initial distribution p0.

-Tjump <temp1> <temp2> <obs>

1st order approximation to temperature jump experiment from temp1 to temp2. The MSM is assumed to be parametrized for temp2.

-autocorrelation <obs>

Stationary autocorrelation-function of the observable obs

-crosscorrelation <obs1> <obs2>

Stationary cross correlation-function of the observables obs1 and obs2.

Options

-relax [<time-list> | <tmin> <tmax> <dt>]

Direct relaxation curve for time-dependent expectations

-relaxspectral [<time-list> | <tmin> <tmax> <dt>]

relaxation curve calculated from spectral decomposition. Should be equivalent to -relax

-fingerprint [<time-list> | <tmin> <tmax> <dt>]

Dynamical fingerprint for time-dependent expectations

Part III. Appendix

A. File formats

Except from trajectory data, which can also be binary data (Gromacs: xtc, Charmm / NAMD: dcd) all files used for input and output are ascii-files. This allows a high transparency, since all files can be inspected visually and second, modifications are easily possible where appropriate. Nevertheless, it is essential to give a precise definition of how the formats of the different files are defined.

Trajectory format

A trajectory with n frames and with each frame $f_i \in \mathbb{R}^{dim}$ of dimension dim and $i \in 0, \dots, n-1$is stored as follows:

```
<time_of_frame_0:double> <x1_of_frame_0:double>   ... <x_dim_of_frame_0:double>
<time_of_frame_1:double> <x1_of_frame_1:double>   ... <x_dim_of_frame_1:double>
 ...
<time_of_frame_2:double> <x1_of_frame_n-1:double> ... <x_dim_of_frame_n-1:double>
```

Dense matrix format

A matrix $M^{rows \times columns}$ is stored in a dense format as defined below. The header line is required.

```
DENSE <rows:int> <columns:int>
<entry_0_0:double> <entry_0_1:double> ... <entry_0_c:double>
<entry_1_0:double> <entry_1_1:double> ... <entry_1_c:double>
  ...
<entry_rows_0:double> <entry_rows_1:double> ... <entry_rows_c:double>
```

Sparse matrix format

A matrix $M^{rows \times columns}$ is stored in a dense format as defined below. The header line is required.

```
SPARSE <rows:int> <columns:int>
<row_of_entry1_at_i_j> <column_of_entry1_at_i_j> <entry1:double>
<row_of_entry2_at_i_j> <column_of_entry2_at_i_j> <entry2:double>
... lines of the above format for every entry not zero ...
```

Vector format

All vectors are written column-wise e.g. stationary distribution. A vector $x \in \mathbb{R}^{dim}$ is stored as defined below. Currently there is no header. The i-line represents the i-th entry x_i of x.

```
<entry_0:double>
<entry_1:double>
    ...
<entry_dim:double>
```

Set definition format

Sets of microstates, defining a metastable state (like in "pcca/sets.ascii") have the format as defined below. The i-th line contains the microstates (arbitrarily many) that belong to the i-th metastable set.

```
<microstate_0_of_set_0:int> <microstate_1_of_set_0:int> <microstate_2_of_set_0:int>
<microstate_0_of_set_1:int> <microstate_1_of_set_1:int>
<microstate_0_of_set_2:int> <microstate_1_of_set_2:int> <microstate_2_of_set_2:int>
```

References

[1] A. Amadei, A. B. Linssen, and H. J. C. Berendsen. Essential dynamics of proteins. *Proteins*, 17:412–225, 1993.

[2] Robert B Best and Gerhard Hummer. Reaction coordinates and rates from transition paths. *Proc. Nat. Acad. Sci. USA*, 102(19):6732–6737, Jan 2005.

[3] Peter G Bolhuis, David Chandler, Christoph Dellago, and Phillip L Geissler. Transition path sampling: Throwing ropes over rough mountain passes, in the dark. *Annu. Rev. Phys. Chem.*, 53:291, Jan 2002.

[4] Gregory R Bowman, Daniel L Ensign, and Vijay S Pande. Enhanced modeling via network theory: Adaptive sampling of markov state models. *J. Chem. Theo. Comp.*, 6(3):787–794, Jan 2010.

[5] Nicaolae V. Buchete and Gerhard Hummer. Coarse master equations for peptide folding dynamics. *J. Phys. Chem. B*, 112:6057–6069, 2008.

[6] Janina Buck, Boris Fürtig, Jonas Noeske, Jens Wöhnert, and Harald Schwalbe. Time-resolved nmr methods resolving ligand-induced rna folding at atomic resolution. *Proc. Natl. Acad. Sci. USA*, 104(40):15699–15704, October 2007.

[7] Chi-Kin Chan, Yi Hu, Satoshi Takahashi, Denis L. Rousseau, William A. Eaton, and James Hofrichter. Submillisecond protein folding kinetics studied by ultrarapid mixing. *Proc. Natl. Acad. Sci. USA*, 94(5):1779–1784, March 1997.

[8] J. D. Chodera, K. A. Dill, N. Singhal, V. S. Pande, W. C. Swope, and J. W. Pitera. Automatic discovery of metastable states for the construction of Markov models of macromolecular conformational dynamics. *J. Chem. Phys.*, 126:155101, 2007.

[9] Christoph Dellago, Peter G Bolhuis, and Phillip L Geissler. Transition path sampling. *Adv. Chem. Phys.*, 123:1, Jan 2002.

[10] P. Deuflhard and M. Weber. Robust perron cluster analysis in conformation dynamics. *ZIB Report*, 03-09, 2003.

[11] Wolfgang Doster, Stephen Cusack, and Winfried Petry. Dynamical transition of myoglobin revealed by inelastic neutron scattering. *Nature*, 337(6209):754–756, February 1989.

[12] Rose Du, Vijay S Pande, Alexander Yu Grosberg, Toyoichi Tanaka, and Eugene I Shakhnovich. On the transition coordinate for protein folding. *J. Chem. Phys*, 108:334, 1998.

[13] Charles Dumont, Tryggvi Emilsson, and Martin Gruebele. Reaching the protein folding speed limit with large, sub-microsecond pressure jumps. *Nature Methods*, 6(7):515–519, May 2009.

[14] Weinan E, Weiqing Ren, and Eric Vanden-Eijnden. Transition pathways in complex systems: Reaction coordinates, isocommittor surfaces, and transition tubes. *Chem. Phys. Lett.*, 413(1-3):242–247, 2005.

[15] Robert R. Hudgins, Fang Huang, Gabriela Gramlich, and Werner M. Nau. A fluorescence-based method for direct measurement of submicrosecond intramolecular contact formation in biopolymers: An exploratory study with polypeptides. *J. Am. Chem. Soc.*, 124(4):556–564, January 2002.

[16] Gerhard Hummer. From transition paths to transition states and rate coefficients. *J. Chem. Phys*, 120(2):516–523, Jan 2004.

[17] J. Chodera and F. Noé. Probability distributions of molecular observables computed from markov models. ii: Uncertainties in observables and their time-evolution. *submitted to J. Chem. Phys. on 23 Oct 2009. In revision*, 2009.

[18] J.-H. Prinz and M. Held and J. C. Smith and F. Noé. Efficient computation of committor probabilities and transition state ensembles. *submitted to SIAM Multiscale Model. Simul. on*, 2009.

[19] M. Jäger, Y. Zhang, J. Bieschke, H. Nguyen, M. Dendle, M. E. Bowman, J. P. Noel, M. Gruebele, and J. W. Kelly. Structure-function-folding relationship in a ww domain. *Proc. Natl. Acad. Sci. USA*, 103:10648–10653, 2006.

[20] Bettina Keller, Jan-Hendrik Prinz, and Frank Noé. Markov models and dynamical fingerprints: Unraveling the complexity of molecular kinetics. *Chem. Phys. (in revision)*, 2011.

[21] Harold D. Kim, G. Ulrich Nienhaus, Taekjip Ha, Jeffrey W. Orr, James R. Williamson, and Steven Chu. Mg2+-dependent conformational change of rna studied by fluorescence correlation and fret on immobilized single molecules. *Procl. Natl. Acad. Sci. USA*, 99(7):4284–4289, April 2002.

[22] Lisa J. Lapidus, William A. Eaton, and James Hofrichter. Measuring the rate of intramolecular contact formation in polypeptides. *Proc. Natl. Acad. Sci. USA*, 97(13):7220–7225, June 2000.

[23] J.-H. Prinz M. Held, P. Metzner and F. Noé. Mechanisms of protein-ligand association and its modulation by protein mutations. *Biophys. J. (in press)*, 2011.

[24] M. Sarich and F. Noé and C. Schütte. On the approximation error of markov state models. *accepted for SIAM Multiscale Model. Simul. on 15 Mar 2010. Preprint at: http://proteomics-berlin.de/771/*, 2010.

[25] A Ma and Aaron R Dinner. Automatic method for identifying reaction coordinates in complex systems. *J. Phys. Chem. B*, 109:6769–6779, Jan 2005.

[26] Luca Maragliano, Alexander Fischer, Eric Vanden-Eijnden, and Giovanni Ciccotti. String method in collective variables: minimum free energy paths and isocommittor surfaces. *J. Chem. Phys*, 125(2):24106, Jul 2006.

[27] Philipp Metzner, Christof Schütte, and Eric Vanden-Eijnden. Illustration of transition path theory on a collection of simple examples. *J. Chem. Phys*, 125(8):084110, Jan 2006.

[28] Philipp Metzner, Christof Schütte, and Eric Vanden-Eijnden. Transition path theory for markov jump processes. *Multiscale Model. Sim.*, 7(3):1192–1219, 2009.

[29] Xavier Michalet, Shimon Weiss, and Marcus Jäger. Single-molecule fluorescence studies of protein folding and conformational dynamics. *Chem. Rev.*, 106:1785–1813, 2006.

[30] Daniel Nettels, Armin Hoffmann, and Benjamin Schuler. Unfolded protein and peptide dynamics investigated with single-molecule fret and correlation spectroscopy from picoseconds to seconds†. *J. Phys. Chem. B*, 112(19):6137–6146, May 2008.

[31] Hannes Neuweiler, Marc Löllmann, Sören Doose, and M. Sauer. Dynamics of unfolded polypeptide chains in crowded environment studied by fluorescence correlation spectroscopy. *J. Mol. Biol.*, 365:856–869, 2007.

[32] F. Noé. Probability Distributions of Molecular Observables computed from Markov Models. *J. Chem. Phys.*, 128:244103, 2008.

[33] F. Noé and S. Fischer. Transition networks for modeling the kinetics of conformational transitions in macromolecules. *Curr. Opin. Struc. Biol.*, 18:154–162, 2008.

[34] F. Noé, I. Horenko, C. Schütte, and J. C. Smith. Hierarchical Analysis of Conformational Dynamics in Biomolecules: Transition Networks of Metastable States. *J. Chem. Phys.*, 126:155102, 2007.

[35] F. Noé, C. Schütte, E. Vanden-Eijnden, L. Reich, and T. R. Weikl. Constructing the full ensemble of folding pathways from short off-equilibrium simulations. *Proc. Natl. Acad. Sci. USA*, 106:19011–19016, 2009.

[36] Frank Noé, Sören Doose, Isabella Daidone, Marc Löllmann, John D. ChoderaJ, Markus Sauer, and Jeremy C. Smith. Dynamical fingerprints: Understanding biomolecular processes in microscopic detail by combination of spectroscopy, simulation and theory. *Proc. Natl. Acad. Sci. USA, in press*, 2011.

[37] Frank Noé, Illia Horenko, Christof Schütte, and Jeremy C Smith. Hierarchical analysis of conformational dynamics in biomolecules: Transition networks of metastable states. *J. Chem. Phys*, 126:155102, 2007.

[38] Frank Noé, Christof Schütte, Eric Vanden-Eijnden, Lothar Reich, and Thomas R Weikl. Constructing the equilibrium ensemble of folding pathways from short off-equilibrium simulations. *Proc. Nat. Acad. Sci. USA*, 106(45):19011–6, Nov 2009.

[39] V. Pande. Pathways for protein folding: is a new view needed? *Current Opinion in Structural Biology*, 8(1):68–79, February 1998.

[40] J.-H. Prinz, H. Wu, M. Sarich, B. Keller, M. Fischbach, M. Held, J.D. Chodera, C. Schütte, J.D. Chodera, and F. Noé. Markov models of molecular kinetics: Generation and validation. *J. Chem. Phys., submitted*, 2011.

[41] Mourad Sadqi, Lisa J. Lapidus, and Victor Munoz. How fast is protein hydrophobic collapse? *Proc. Natl. Acad. Sci. USA*, 100(21):12117–12122, October 2003.

[42] Ilme Schlichting, Steven C. Almo, Gert Rapp, Keith Wilson, Kyriakos Petratos, Arno Lentfer, Alfred Wittinghofer, Wolfgang Kabsch, Emil F. Pai, Gregory A. Petsko, and Roger S. Goody. Time-resolved x-ray crystallographic study of the conformational change in ha-ras p21 protein on gtp hydrolysis. *Nature*, 345(6273):309–315, May 1990.

[43] C. Schütte, A. Fischer, W. Huisinga, and P. Deuflhard. A direct approach to conformational dynamics based on hybrid monte carlo. *J. Comput. Phys.*, 151:146–168, 1999.

[44] W. C. Swope, J. W. Pitera, and F. Suits. Describing protein folding kinetics by molecular dynamics simulations: 1. theory. *J. Phys. Chem. B*, 108:6571–6581, 2004.

[45] Philip Tinnefeld and Markus Sauer. Branching out of single-molecule fluorescence spectroscopy: Challen ges for chemistry and influence on biology. *Angewandte Chemie Intl. Ed.*, 44:2642–2671, 2005.

[46] E. Vanden-Eijnden. Transition path theory. pages 453–493. 2006.

[47] Vincent A. Voelz, Gregory R. Bowman, Kyle Beauchamp, and Vijay S. Pande. Molecular Simulation of ab Initio Protein Folding for a Millisecond Folder NTL9. *J. Am. Chem. Soc.*, 132(5):1526–1528, February 2010.

[48] A. Volkmer. One- and two-photon excited fluorescence lifetimes and anisotropy decays of green fluorescent proteins. *Biophys. J.*, 78(3):1589–1598, March 2000.

[49] M. Weber. Improved perron cluster analysis. *ZIB Report*, 03-04, 2003.

[50] Marcus Weber. Improved perron cluster analysis. *ZIB Report*, 04, Jan 2003.