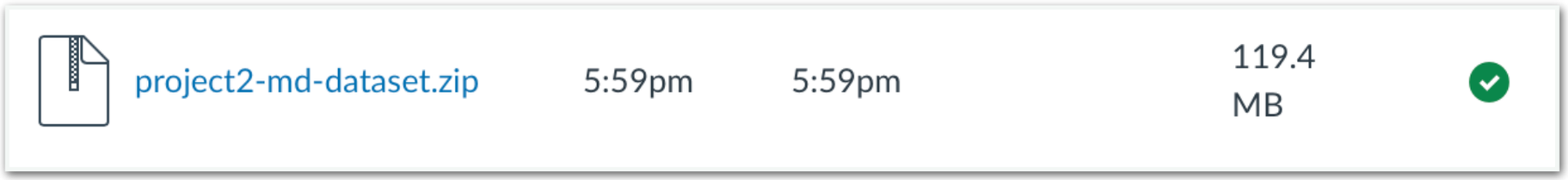
Project 2

Kinetic analysis of Alanine Dipeptide simulation datasets using VAMPnets and MSMs

Assignments

- Download the MD dataset



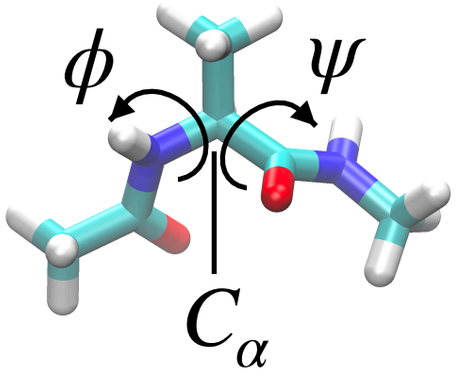
- Write your own code to do TICA (time-lagged independent component analysis)
- Write your own code to do PCCA (Perron cluster-cluster analysis)
- Build a Markov State Model based on your macro state model
- Use deeptime to build VAMPnets

Note: If your TICA / PCCA code doesn't work as expected, you can use MSMBuilder to continue; but bring your code to the evaluation even if it doesn't work.

Please refer to this tutorial if you don't know how to start: DOI: 10.26434/chemrxiv-2023-kvsvl

Step 1 of MSM and VAMPnets: generate features

- The MD dataset:



22 atoms of alanine dipeptide. (888 water not saved)

10 heavy atoms (6C, 2O, 2N), 12 hydrogens

45 pairwise distances of heavy atoms

100 MD trajectories (.xtc files)

saving interval: 1ps

length of each trajectories: 10ns, 10001 frames

100 text files of Ramachandran angles

- Generate features based on pairwise distances of heavy atoms

```
import mdtraj as md
from msmbuilder.featurizer import AtomPairsFeaturizer
import numpy as np

pairs_feat = AtomPairsFeaturizer(np.loadtxt("../ala2_atom_pairs"))

for i in range(0,10):
    traj = md.load("../ala2-xtc-1ps/ala2-1ps-0"+str(i)+".xtc", top="../ala2-xtc-1ps/ala2.pdb")
    np.save("features/ftraj_"+str(i), traj)
for i in range(10,100):
    traj = md.load("../ala2-xtc-1ps/ala2-1ps-"+str(i)+".xtc", top="../ala2-xtc-1ps/ala2.pdb")
    np.save("features/ftraj_"+str(i), traj)
```

Step 2 of MSM: generate TICA

- The Koopman operator

$$\mathbb{E}(\chi(\mathbf{x}_{t+\tau})) = K(\tau)^T \mathbb{E}(\chi(\mathbf{x}_t))$$

$$K(\tau) = C_{00}^{-1} C_{01}$$

$$C_{00} = \mathbb{E}_t \left[\mathbf{x}_t \mathbf{x}_t^T\right]$$
$$C_{11} = \mathbb{E}_{t+\tau} \left[\mathbf{x}_{t+\tau} \mathbf{x}_{t+\tau}^T\right]$$
$$C_{01} = \mathbb{E}_{t+\tau} \left[\mathbf{x}_t \mathbf{x}_{t+\tau}^T\right]$$

$$\chi(\mathbf{x}_t) = \Delta d_{12}(t) \oplus \Delta d_{13}(t) \oplus \ldots \oplus \Delta d_{23}(t) \oplus \ldots \qquad \Delta d_{ij}(t) = d_{ij}(t) - \bar{d}_{ij}$$

- The time-lagged independent component analysis

$$(\lambda, V) = \mathrm{eig}[K(\tau)]$$

$$\chi_k(t) = \sum_{(ij)} \Delta d_{(ij)}(t) \cdot V_{(ij),k}$$

Pick TICs $\chi_k(t)$ with largest λ_k

$\Rightarrow$ the TICA trajectories

PS: you can do TICA with $d_{ij}(t)$ instead of $\Delta d_{ij}(t)$. In this way, you will have an additional mode with $\lambda_1(t) \equiv 1$. Please ignore this eigenvector because it is stationary and not related to any dynamics.

Recommended parameters:
number of TICs: 2~3
TICA lag time τ: 1~10ps, 20ps, etc

Step 3 of MSM: generate a micro state model

- Generate a micro state model with K-Centers

```
from msmbuilder.cluster import KCenters,MiniBatchKMeans
from msmbuilder.msm import MarkovStateModel
import numpy as np

ttrajs = ...
kcenter = KCenters(n_clusters=400)
kcenter.fit(ttrajs)
ktrajs = kcenter.transform(ttrajs)
save_to_files(ktrajs)
```

- Other methods: K-means, DBSCAN, etc

- But K-Centers is recommended for its high efficiency and ability to find states at transition regions.

Step 4 of MSM: build a macro state model

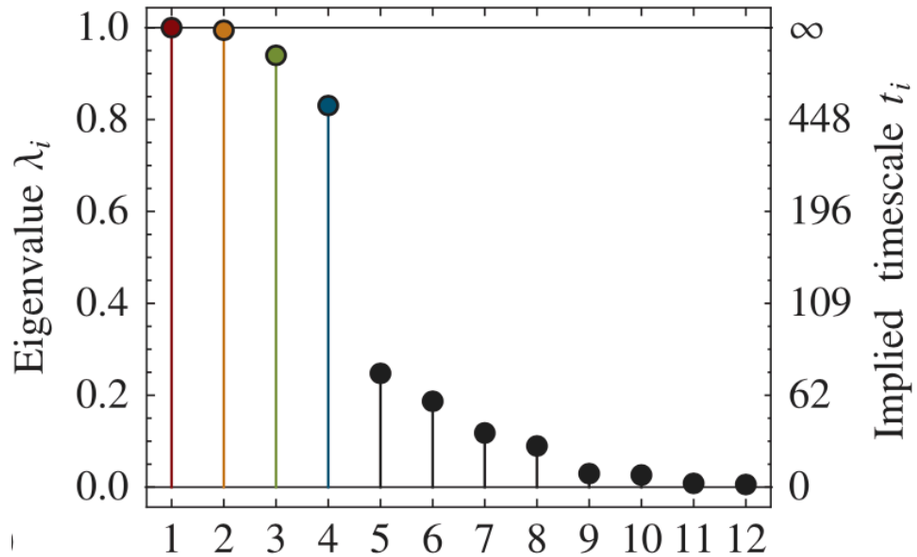
- The Transition Probability Matrix and the Master equation

$$T_{ij}(\tau) = P(\chi_{t+\tau} = j \;\; \chi_t = i) = \frac{C_{ij}(\tau)}{\sum_j C_{ij}(\tau)}$$

$$C(\tau) = \sum_{t_0} \vec{\chi}(t_0 + \tau) \cdot \vec{\chi}(t_0)$$

$$P_j(t_1 + \tau) = \sum_i P_i(t_1) T_{ij}(\tau)$$

$$P(t_1 + \tau) = P(t_1) T(\tau)$$



- Eigenvalues and eigenvectors

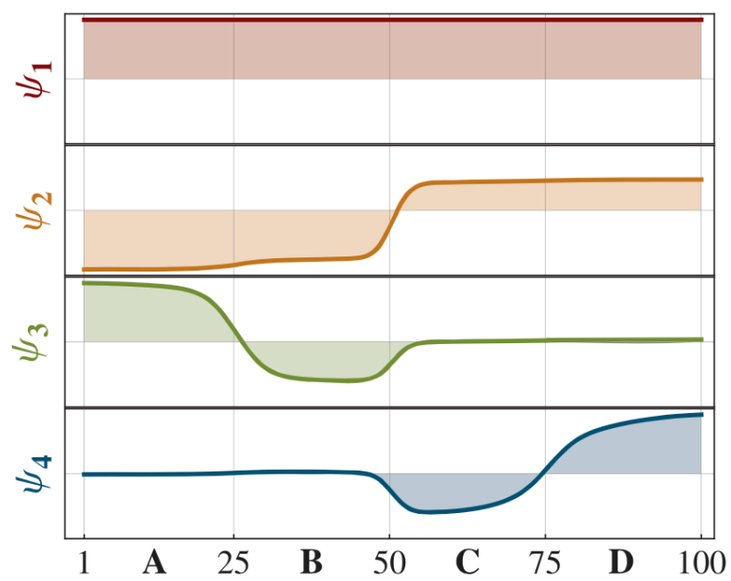
$$L^T T(\tau) R = \lambda \qquad (\lambda, R) = \mathrm{eig}[T(\tau)]$$

$$L^T = R^{-1} \quad R^T = L^{-1}$$

If $P(t_1) - P_{\mathrm{eq}} \propto L_i \qquad L_i^T T(\tau) = \lambda_i L_i^T$

$$P(t) - P_{\mathrm{eq}} = \lambda^{t/\tau}(P(0) - P_{\mathrm{eq}})$$

$$P(t) - P_{\mathrm{eq}} \propto e^{t \ln \lambda_i} \equiv e^{-t/\tau s_i}$$



Implied timescales: $s_i = -\tau / \ln \lambda_i$

Step 4 of MSM: build a macro state model

- The Perron cluster-cluster analysis (PCCA):
    - Using the sign structure of top right eigenvectors to find state boundary.
        - Right eigenvectors: $V_0 = [1,1,...1]$ for $\lambda_0 = 1$
    - For each transition mode V_i (i.e., eigenvector not corresponding to eigenvalue $\lambda = 1$), state j with $V_{ij} > 0$ and $V_{ij} \leq 0$ should be assigned to different states.
        - The first eigenvector, $R_0 = [1,1,...1]$ corresponding to $\lambda_0 = 1$ shouldn't be used here; it's stationary but not a transition mode.
    - A n-state model will have $n - 1$ transition modes, thus each V_i should only cut one previous state into two parts. Don't cut more than one states for each V_i.

Step 5 of MSM: build MSM

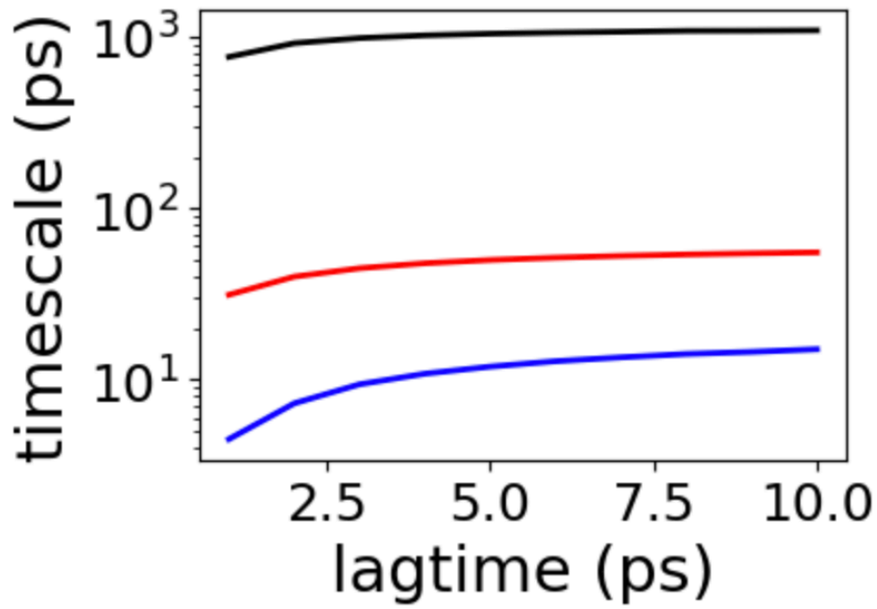
```
import numpy as np
macro4_trajs = []
for i in range(100):
    this_traj = np.loadtxt("macro4/macro_"+str(i), dtype=in
    macro4_trajs.append(this_traj)

ramas = []
for i in range(10):
    ramas.append(np.loadtxt("../ala2-xtc-1ps/rama-traj/ala2
for i in range(10,100):
    ramas.append(np.loadtxt("../ala2-xtc-1ps/rama-traj/ala2
```

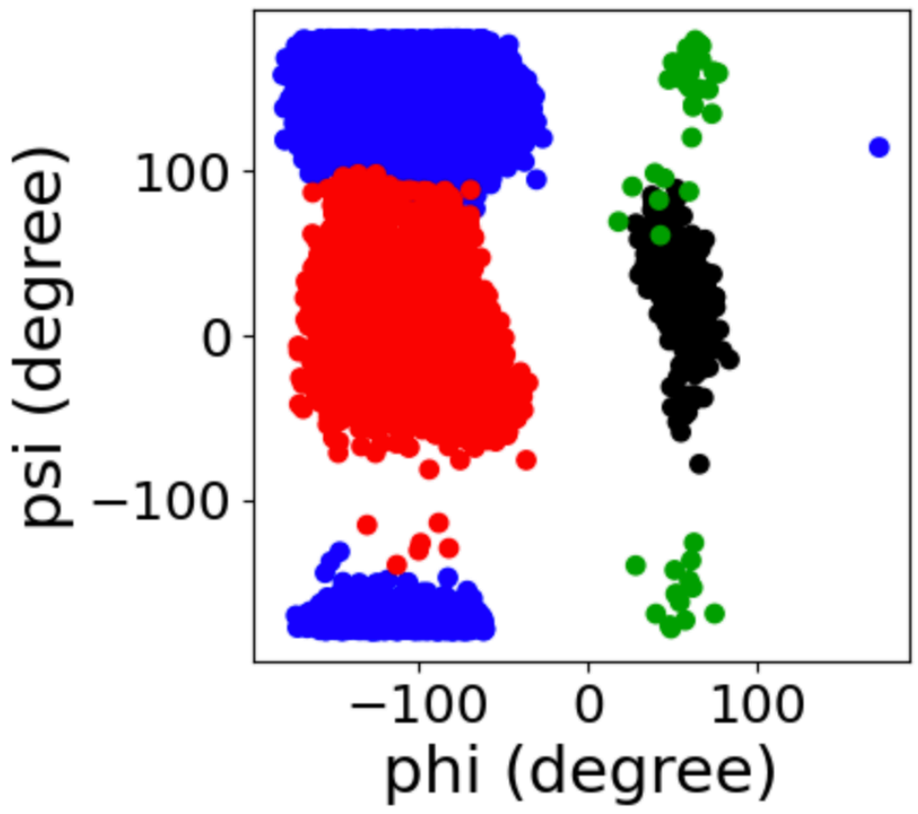
```
from msmbuilder.msm import MarkovStateModel
macro_lagtimes = range(1,11)
msm_timescales = []
for macro_lagtime in macro_lagtimes:
    msm = MarkovStateModel(n_timescales=3, lag_time=macro_l
    msm.fit(macro4_trajs)
    msm_timescales.append(msm.timescales_)
```

```
import matplotlib.pyplot as plt
figure = plt.plot(macro_lagtimes, msm_timescales[:,0], colo
figure = plt.plot(macro_lagtimes, msm_timescales[:,1], colo
figure = plt.plot(macro_lagtimes, msm_timescales[:,2], colo
```

```
state_probabilities = one_hot_of_all(macro4_trajs)
assignments = concatenate_all(macro4_trajs)
dihedrals = concatenate_all(ramas)
```

```
colors = ['black', 'blue', 'red', '#00A000']
for i in range(4):
    plt.scatter(*dihedrals[::50].T, c=colors[i], alpha=stat
```





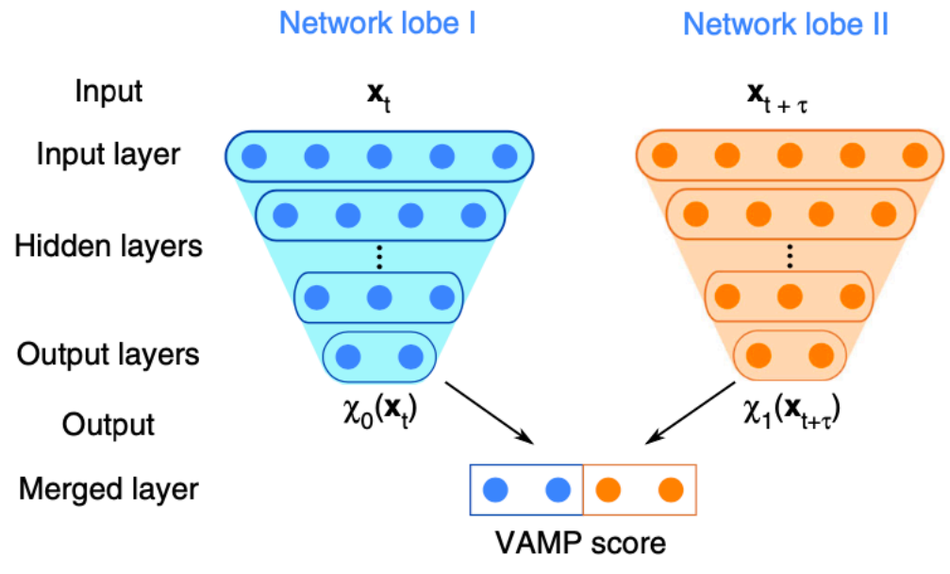
Step 2 of VAMPnets: build a VAMPnets model

$$\mathbb{E}(\chi_1(\mathbf{x}_{t+\tau})) = K(\tau)^T \mathbb{E}(\chi_0(\mathbf{x}_t))$$

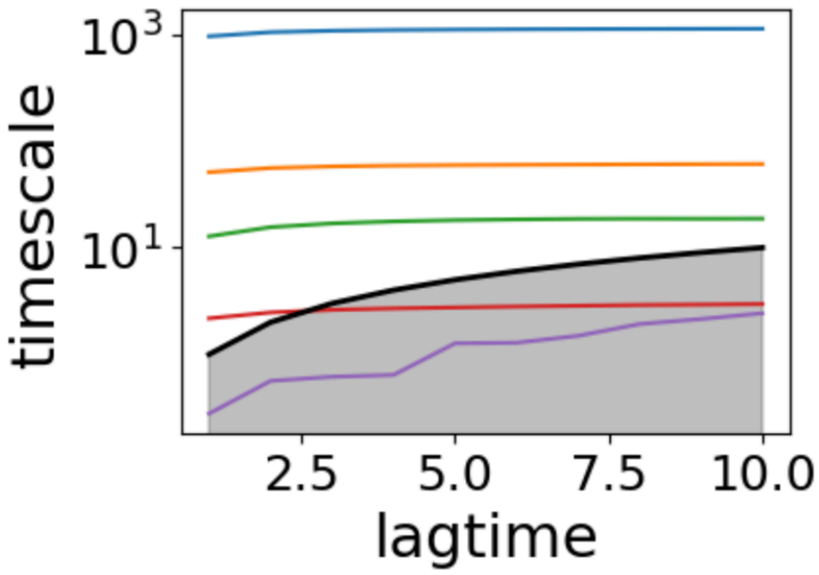
$$K(\tau) = C_{00}^{-1} C_{01}$$

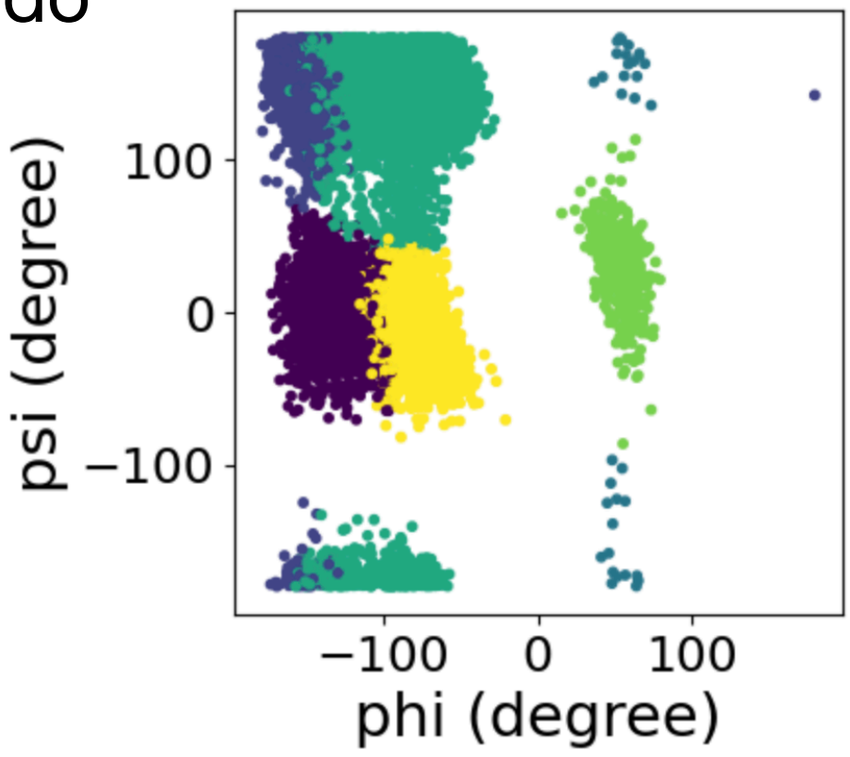
$$L = \| C_{00}^{-0.5} C_{01} C_{11}^{-0.5} \|_F^2$$

$$C_{00} = \overline{\mathbf{X}}\overline{\mathbf{X}}^T \quad C_{11} = \overline{\mathbf{Y}}\overline{\mathbf{Y}}^T \quad C_{01} = \overline{\mathbf{X}}\overline{\mathbf{Y}}^T$$



- Please follow the [tutorial of deeptime] to do VAMPnets





Tip: keep trying if your model fails. It doesn't always work.