Beyond the VAMP

Moritz Hoffmann

February 24, 2022

Transfer operator methods

Ex: Time-homogeneous and deterministic Koopman operator

Deterministic setting

Under a relationship like

$$x_{t+\tau} = T(x_t), \quad v_t = g(x_t) \in \mathbb{C}$$

the Koopman operator is defined as

$$[\mathcal{K}_\tau g](x) := g(T(x)).$$

Linearity

$\mathcal{K}_\tau(\alpha g + \beta f) = (\alpha g + \beta f) \circ T = \alpha g \circ T + \beta f \circ T = \alpha \mathcal{K}_\tau g + \beta \mathcal{K}_\tau f$

Transfer operators (time-homogeneous)

Transition density

$$p_\tau : \Omega \times \Omega \to \mathbb{R}_{\geq 0}, \quad \mathbb{P}[x_{t+\tau} \in B \mid x_t = x] = \int_B p_\tau(x, y) dy,$$

Koopman operator (propagates observables):

$$[\mathcal{K}_\tau g](x) = \int g(y) p_\tau(x, y) dy = \mathbb{E}[g(x_{t+\tau}) \mid x_t = x],$$

Perron–Frobenius operator (propagates densities, Markov operator)

$$[\mathcal{P}_\tau f](y) = \int f(x) p_\tau(x, y) dx,$$

Transfer operators (cont')

Reweighted PF / transfer operator

propagates densities $u = f/\mu$ (ergodicity $\Rightarrow \exists \mu : \mathcal{P}\mu = \mu$):

$$[\mathcal{T}_\tau u](y) = \frac{1}{\mu(y)} \int \mu(x) u(x) p_\tau(x, y) dx.$$

then: $\mathcal{T}\mathbb{1} = \mathbb{1}$

Transfer operators (time-inhomogeneous)

Transition density:

$$p_{s,t} : \Omega \times \Omega \to \mathbb{R}_{\geq 0}, \quad \mathbb{P}[x_t \in B \mid x_s = x] = \int_B p_{s,t}(x, y)dy,$$

Initial distribution μ_s and final distribution μ_t related by $\mu_t = \mathcal{P}_{s,t}\mu_s$.

Transfer operator:

$$\mathcal{T}_{s,t} : L^2_{\mu_s} \to L^2_{\mu_t}, \quad \mathcal{T}_{s,t}u = \frac{1}{\mu_t}\mathcal{P}_{s,t}(u\mu_s).$$

Adjointness:

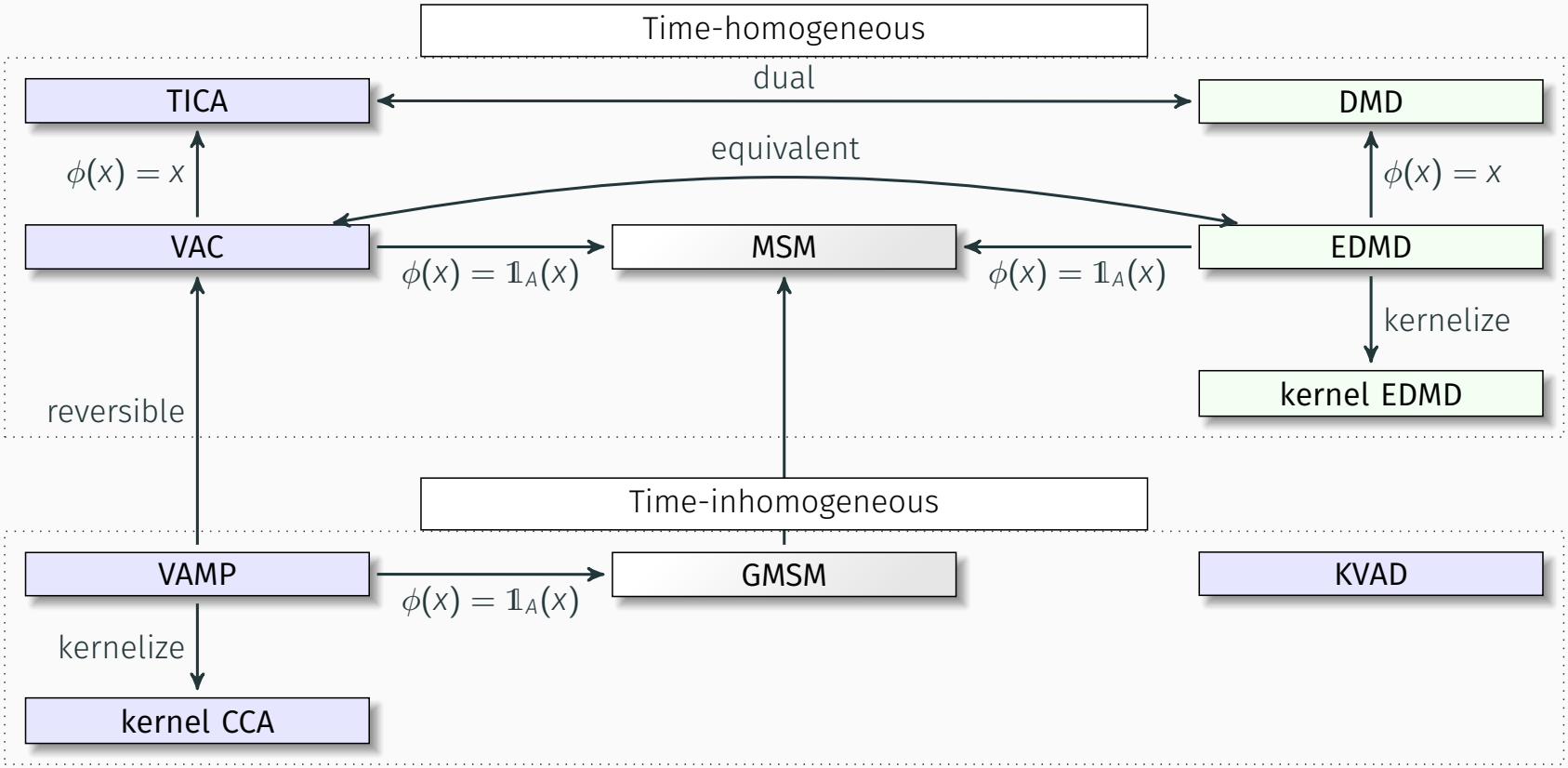
$$\langle \mathcal{T}_{s,t}f, g\rangle_{\mu_t} = \langle f, \mathcal{K}_{s,t}g\rangle_{\mu_s} \quad \forall f \in L^2_{\mu_s} \forall g \in L^2_{\mu_t}.$$

Goal

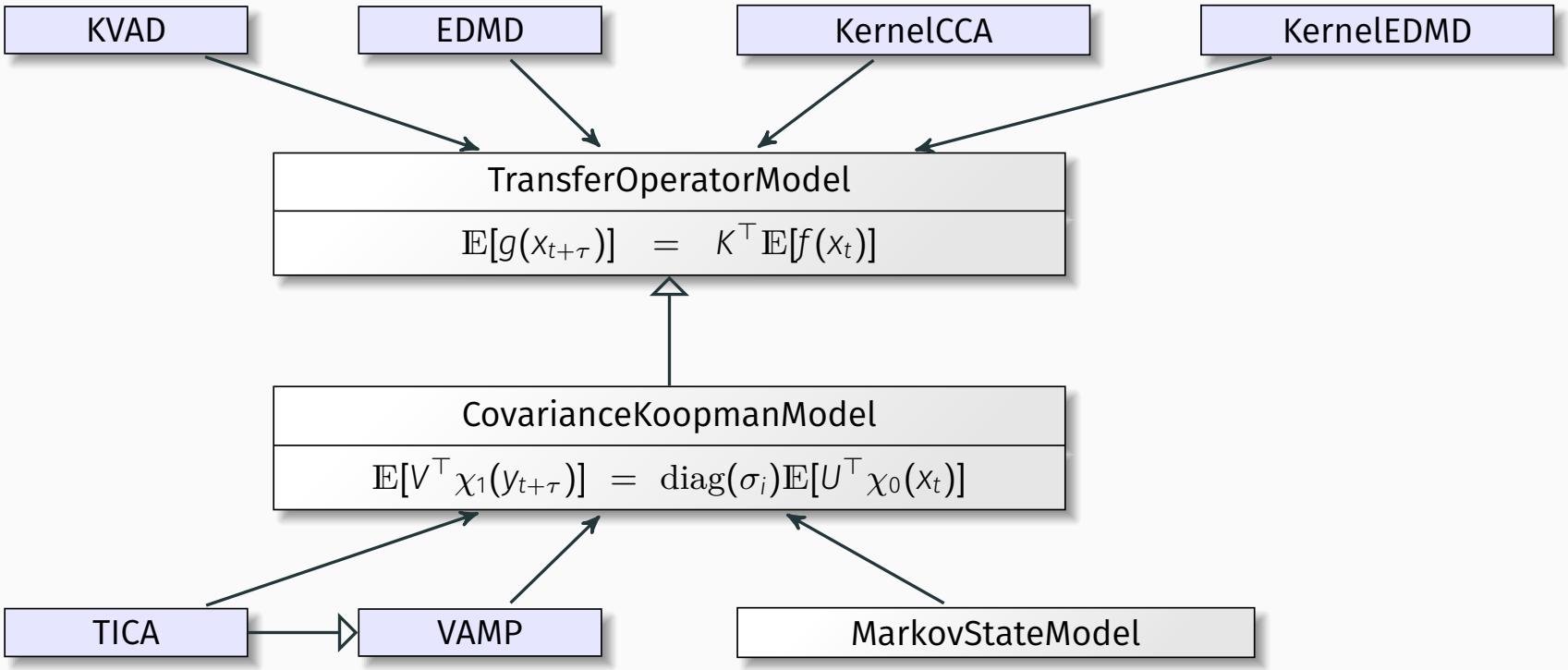
Find approximations to $\mathcal{K}$ or $\mathcal{T}$ so that

$$\mathbb{E}[\mathrm{g}(x_{t+\tau})] \approx K^\top \mathbb{E}[\mathrm{f}(x_t)]$$

Methods

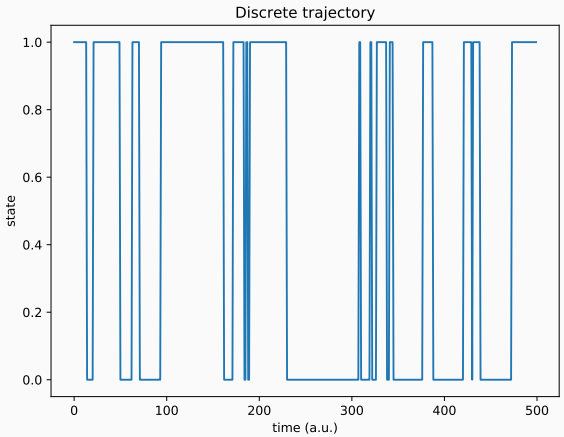

Time-homogeneous
dual
TICA
DMD
$\phi(x) = x$
equivalent
$\phi(x) = x$
VAC
MSM
EDMD
$\phi(x) = \mathbb{1}_A(x)$
$\phi(x) = \mathbb{1}_A(x)$
kernelize
kernel EDMD
reversible
Time-inhomogeneous
VAMP
GMSM
KVAD
$\phi(x) = \mathbb{1}_A(x)$
kernelize
kernel CCA

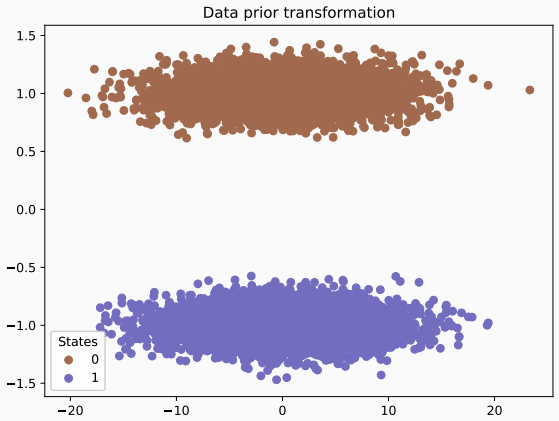
Methods class diagram

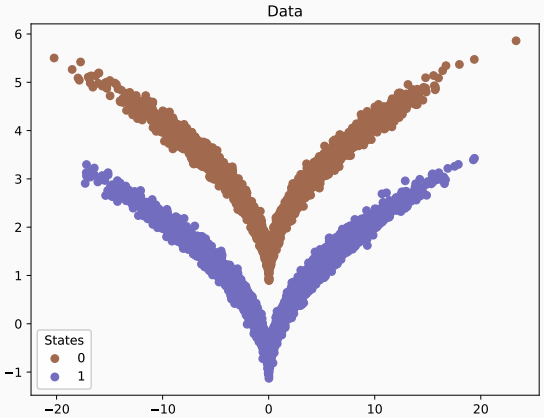


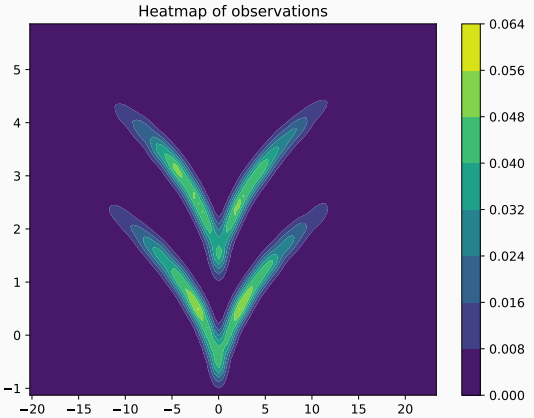
Double-wedge comparison

The dataset (time-homogeneous and reversible)







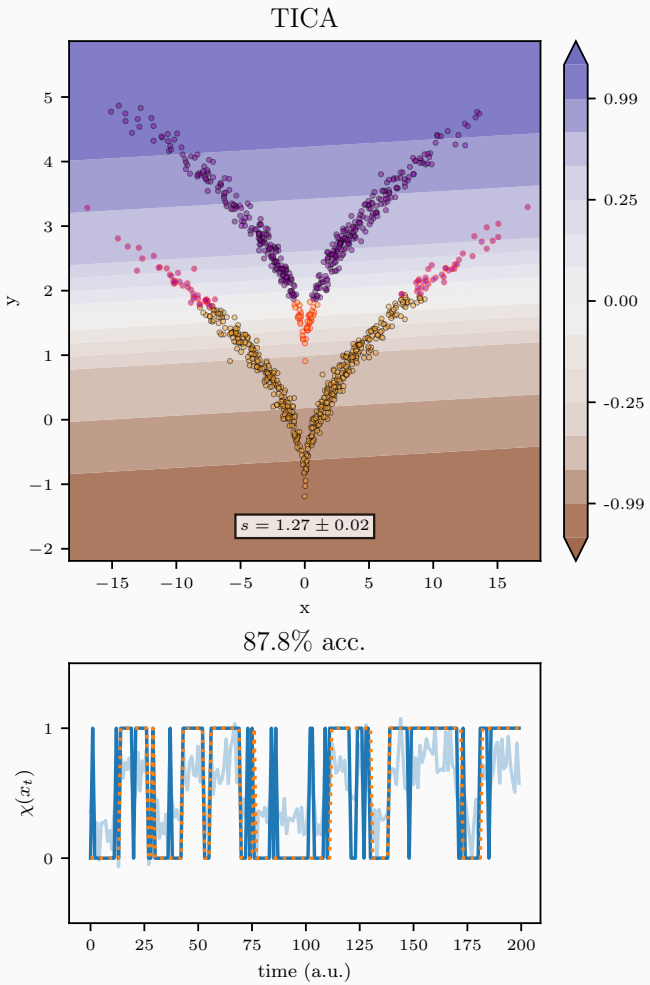


$$P = \begin{pmatrix} 0.95 & 0.05 \\ 0.05 & 0.95 \end{pmatrix}$$

$$(x, y) \mapsto (x, y + \sqrt{|x|})$$

```
from deeptime.data import sqrt_model
discrete_traj, traj = sqrt_model(n_samples=1000, seed=777)
```

TICA



Assumes detailed-balance.
If $\mathcal{K}$ is HS, it admits eigenvalue decomposition $\mathcal{K} = \sum_{i=1}^{\infty} \lambda_i \langle \cdot, \varphi_i \rangle_\mu \varphi_i$.
Approximate via

$$C_{0\tau} \hat{\varphi}_i = \hat{\lambda}_i C_{00} \hat{\varphi}_i$$

```
from deeptime.decomposition\
    import TICA, vamp_score_cv

est = TICA(dim=1)
proj = est.fit_transform(traj, lagtime=1)
scores = vamp_score_cv(
    est, traj, r=2,
    n=10, blocksize=100
)
```

(Molgedey and Schuster, 1994)

VAMP

Optimize for K, $\mathbf{f}$, and $\mathbf{g}$. Modelling error is decomposed

$$\|\hat{\mathcal{K}}_\tau - \mathcal{K}\|_{\mathrm{HS}}^2 = -\mathcal{R}(K, f, g) + \|\mathcal{K}_\tau\|_{\mathrm{HS}}^2,$$

then maximize $\mathcal{R}$.

Smallest error is achieved for

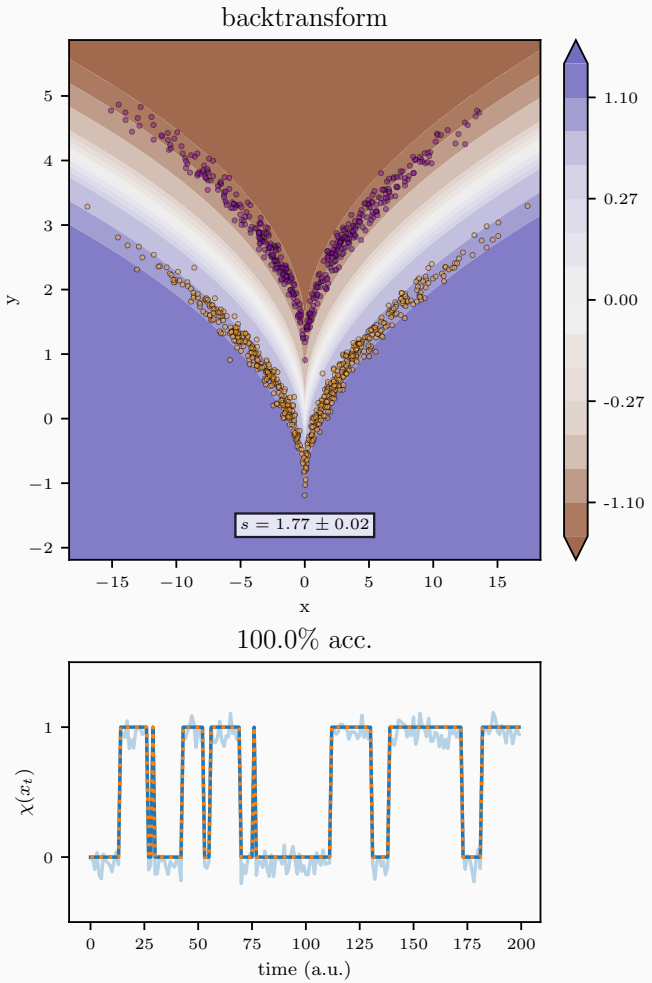
$$\hat{\mathcal{K}} = \sum_{i=1}^{m} \sigma_i \langle \cdot, \phi_i \rangle \psi_i.$$

The VAMP-r score $s_r = \sum_i \sigma_i^r$ makes the assumption that $\mathcal{K}_\tau$ is Hilbert–Schmidt ($\sum_i \|\mathcal{K}_\tau e_i\| < \infty$): $\lightning$ for most deterministic systems $\rightsquigarrow$ KVAD[1]

(Wu and Noe, 2020).

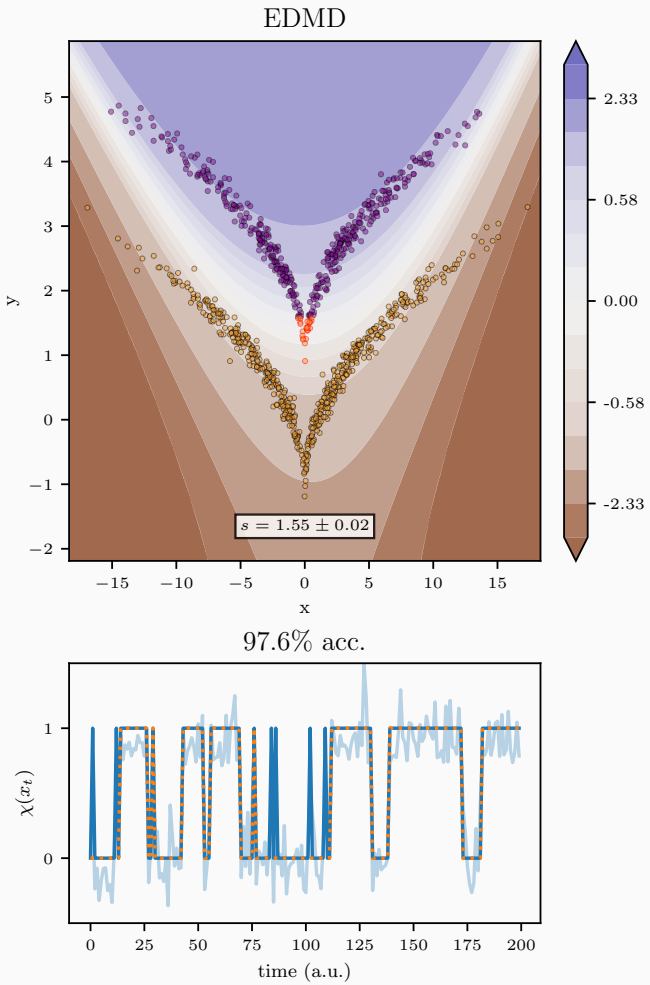
[1] (Tian and Wu, 2021).

Backtransform




```python
from deeptime.decomposition\
    import VAMP, vamp_score_cv

def backtransform(x):
    y = np.copy(x)
    y[:, 1] -= np.sqrt(np.abs(y[:, 0]))
    return y

est = VAMP(
    lagtime=1, dim=1,
    observable_transform=backtransform
)
scores = vamp_score_cv(
    est, traj, r=2,
    blocksize=100
)
```

EDMD

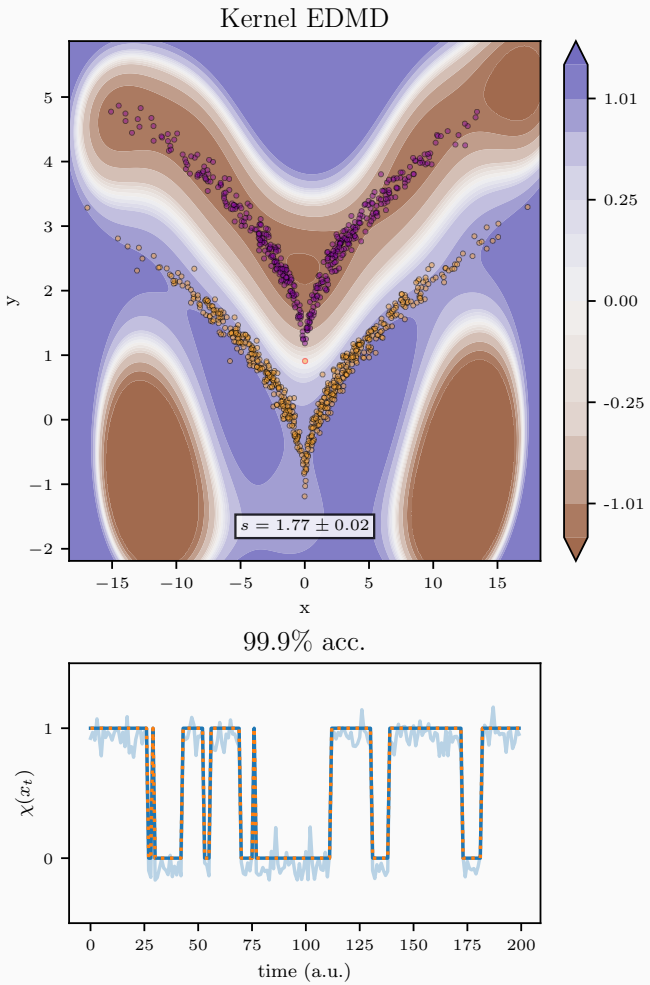


Given basis $\mathbf{\Psi}$: $\min_K \|\mathbf{\Psi}(Y) - K\mathbf{\Psi}(X)\|_F$

```
from deeptime.decomposition\
    import VAMP, EDMD, vamp_score_cv
from deeptime.basis\
    import Monomials

basis = Monomials(p=2, d=2)
edmd = EDMD(basis)\
    .fit_fetch(traj, lagtime=1)

vamp = VAMP(
    observable_transform=edmd
)
scores = vamp_score_cv(
    vamp, traj, r=2,
    blocksize=100
)
```

(Williams and Kevrekidis and Rowley, 2015)

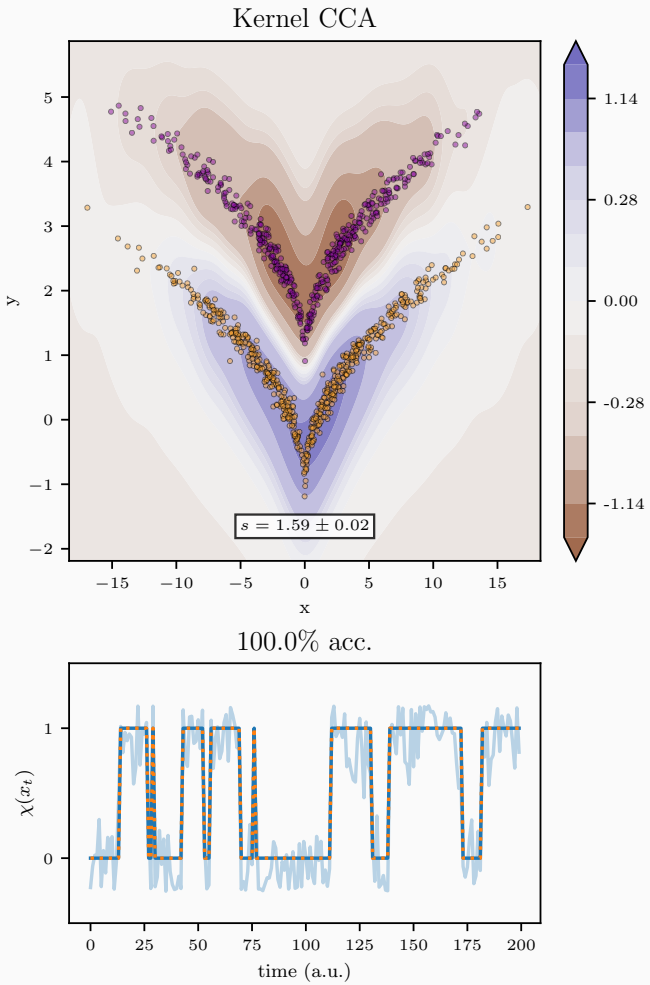
Kernel EDMD




```
from deeptime.decomposition
    import VAMP, vamp_score_cv, KernelEDMD
from deeptime.kernels\
    import GaussianKernel

kernel = GaussianKernel(1.42)
kedmd = KernelEDMD(
    kernel=kernel, epsilon=6e-4
).fit_fetch(traj, lagtime=1)

est = VAMP(
    lagtime=1, dim=1,
    observable_transform=kedmd
)
scores = vamp_score_cv(
    est, traj, r=2,
    blocksize=100
)
```

(Williams and Rowley and Kevrekidis, 2015)

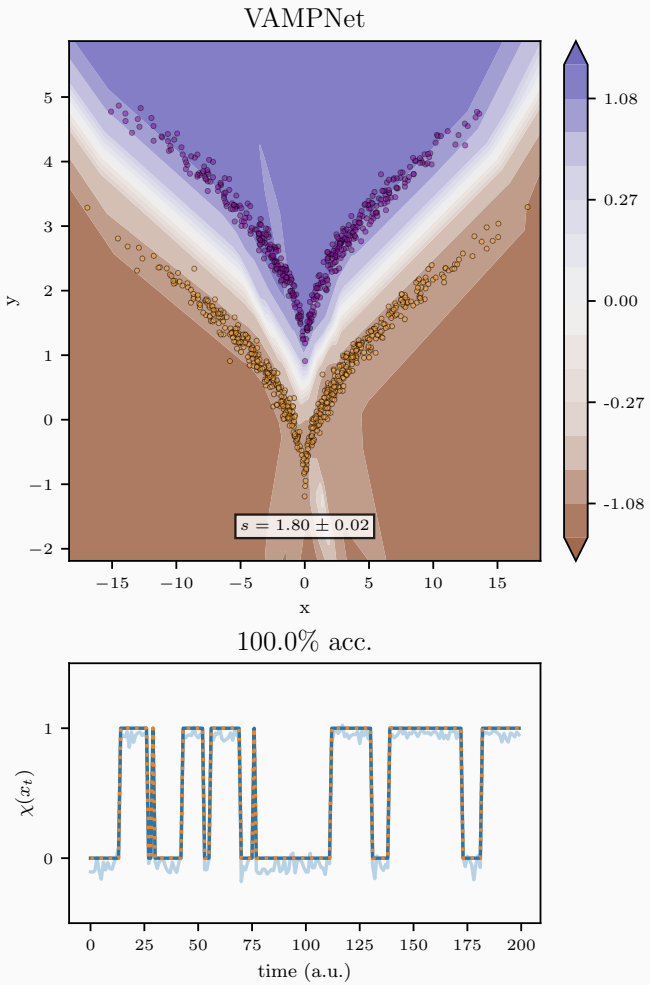
Kernel CCA




```
from deeptime.decomposition\
    import VAMP, vamp_score_cv, KernelCCA
from deeptime.kernels\
    import GaussianKernel

kernel = GaussianKernel(0.85)
kcca = KernelCCA(
    kernel=kernel, epsilon=0.35
).fit_fetch(traj, lagtime=1)

est = VAMP(
    lagtime=1, dim=1,
    observable_transform=kcca
)
scores = vamp_score_cv(
    est, traj, r=2,
    blocksize=100
)
```

(Bach and Jordan, 2002)

VAMPNets




```
from torch.utils.data import DataLoader
from deeptime.util.torch import MLP
from deeptime.util.data\
  import TrajectoryDataset
from deeptime.decomposition\
  import VAMP, vamp_score_cv
from deeptime.decomposition.deep\
  import VAMPNet

dataset = TrajectoryDataset(
  lagtime=1, traj.astype(np.float32)
)
loader_train = DataLoader(
  dataset, batch_size=128, shuffle=True
)
estimator = VAMPNet(lobe=MLP(...),
          learning_rate=1e-3)
vampnet = estimator.fit_fetch(
  loader_train, n_epochs=100, progress=tqdm
)
est = VAMP(lagtime=1, dim=1,
       observable_transform=vampnet)
scores = vamp_score_cv(est, traj, r=2,
             blocksize=100)
```

(Mardt and Pasquali and Wu and Noe, 2018)

Markov state models

Projection operator

$$\mathcal{Q}_h : L^1(\Omega) \to \Delta_h := \mathrm{span}(\mathbb{1}_1, \ldots, \mathbb{1}_n), \quad f \mapsto \sum_{i=1}^{n} \int_{S_i} f(x) dx \mathbb{1}_i$$

with $\mathbb{1}_i := 1/\mu(S_i)\chi_i$. Then:

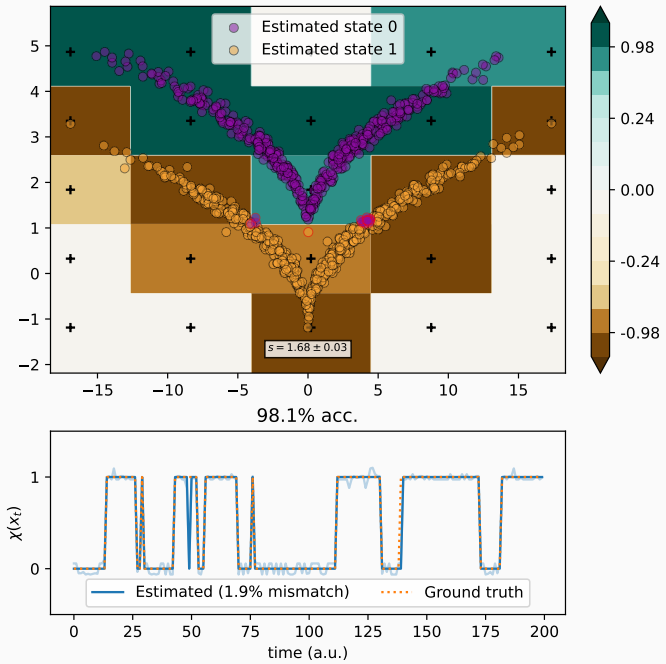
$$\langle \mathcal{Q}_h f - f, \mathbb{1}_i \rangle = 0$$

and consequently with $\mathcal{P}_h := \mathcal{Q}_h \mathcal{P}$

$$\langle \mathbb{1}_i, \mathcal{P}\mathbb{1}_j \rangle = \langle \mathbb{1}_i, \mathcal{P}_h \mathbb{1}_j \rangle$$

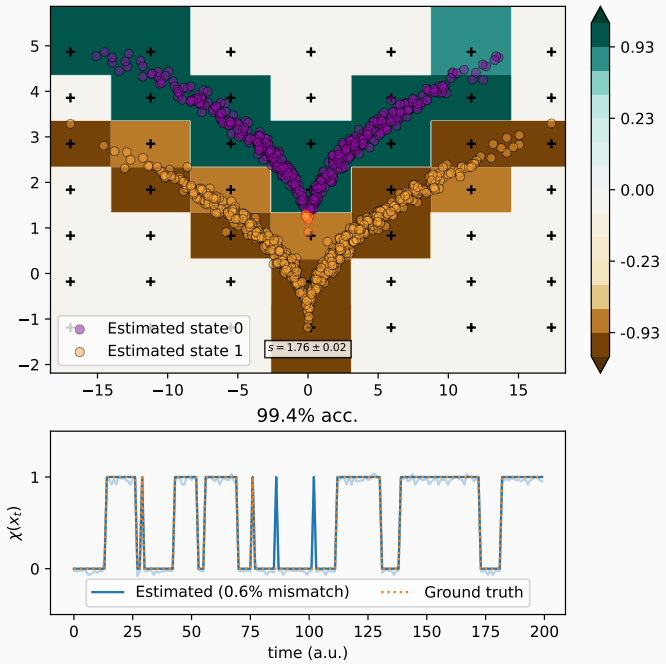
We obtain a Markov state model transition matrix!

Markov state models (5x5)



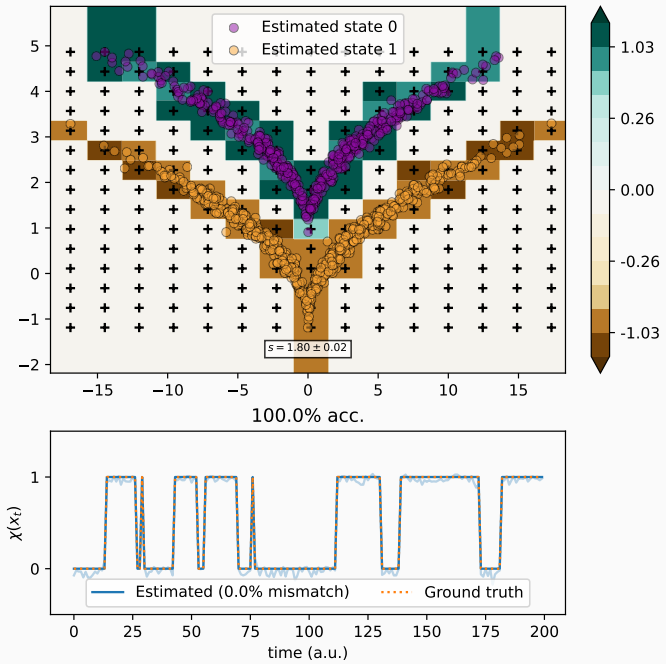

```
clustering = BoxDiscretization(dim=2, n_boxes=5)\
  .fit_fetch(traj)
msm = MaximumLikelihoodMSM(lagtime=1).fit_fetch(
  clustering.transform(traj)
)

def transform(xy):
  dtr = clustering.transform(xy)
  evr = msm.eigenvectors_right(2)
  return evr[dtr]

est = VAMP(lagtime=1, dim=1, \
  observable_transform=transform).fit(traj)
scores = vamp_score_cv(est, traj, \
  r=2, blocksize=100)
```

Markov state models (7x7)




```
clustering = BoxDiscretization(dim=2, n_boxes=7)\
  .fit_fetch(traj)
msm = MaximumLikelihoodMSM(lagtime=1).fit_fetch(
  clustering.transform(traj)
)

def transform(xy):
  dtr = clustering.transform(xy)
  evr = msm.eigenvectors_right(2)
  return evr[dtr]

est = VAMP(lagtime=1, dim=1, \
  observable_transform=transform).fit(traj)
scores = vamp_score_cv(est, traj, \
  r=2, blocksize=100)
```

Markov state models (15x15)

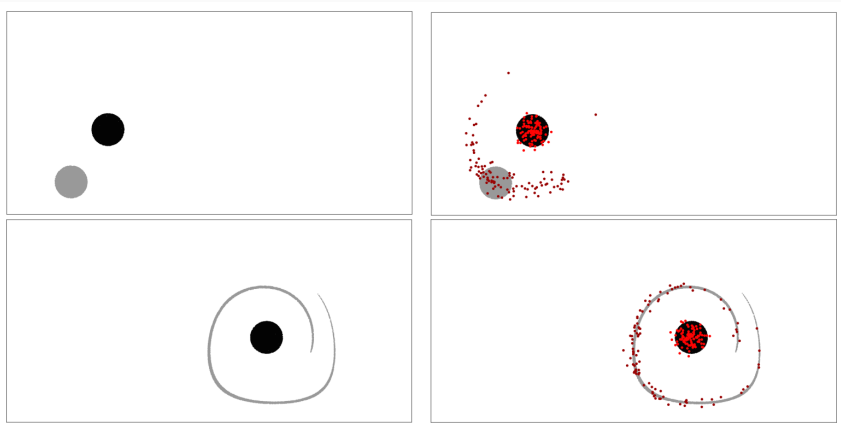



```
clustering = BoxDiscretization(dim=2, n_boxes=15)\
  .fit_fetch(traj)
msm = MaximumLikelihoodMSM(lagtime=1).fit_fetch(
  clustering.transform(traj)
)

def transform(xy):
  dtr = clustering.transform(xy)
  evr = msm.eigenvectors_right(2)
  return evr[dtr]

est = VAMP(lagtime=1, dim=1, \
  observable_transform=transform).fit(traj)
scores = vamp_score_cv(est, traj, \
  r=2, blocksize=100)
```

Coherent set estimation

Coherence

Coherent (geometry-preserving) and metastable sets can be detected by studying the forward-backward dynamics:



(Banisch and Koltai, 2016)

Study forward-backward dynamic

$$\mathcal{K}_\tau^* \mathcal{K}_\tau .$$

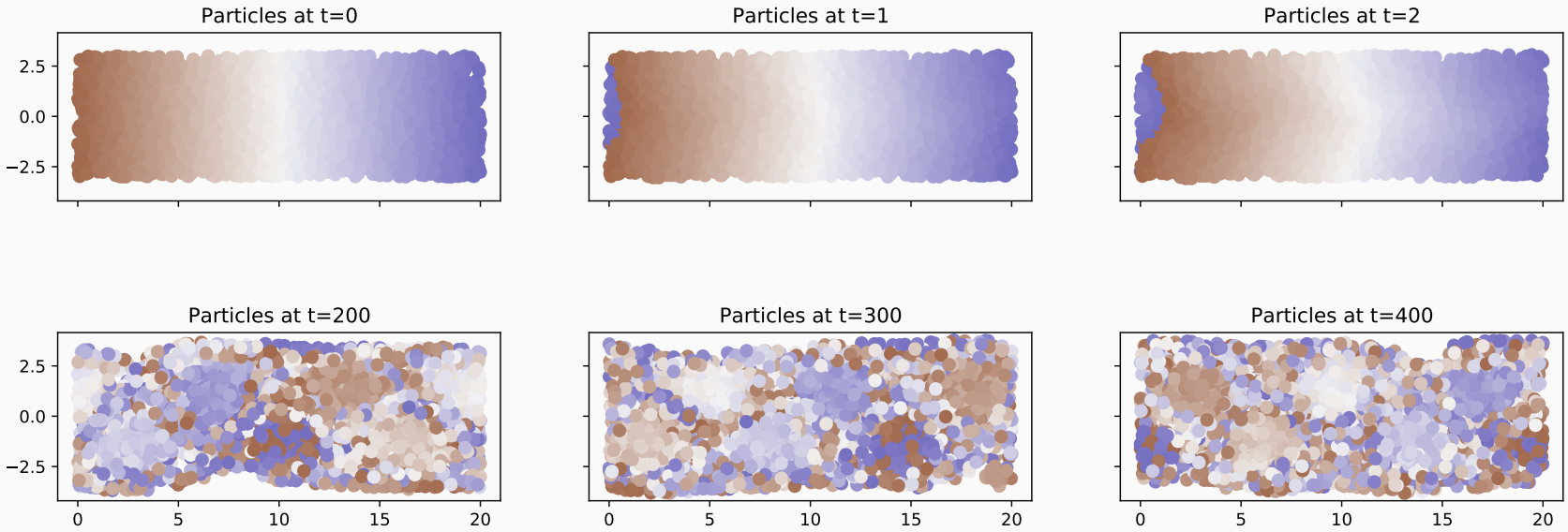
Eigenfunctions of this dynamic with eigenvalue close to 1 remain nearly unchanged.

(Froyland and Santitissadeekorn and Monahan, 2010)

The Bickley jet

Models atmospheric flow, propagates (noninteracting) particles according to an ordinary differential equation.

```
from deeptime.data import BickleyJet
simulator = BickleyJet(h=1e-3, n_steps=100)
traj = simulator(t0=0, x0=Xinit, length=401)
plot(traj)
```



(Hadjighasem and Karrasch and Teramoto and Haller, 2016)

$$\|\hat{\mathcal{P}} - \mathcal{P}\|_{\mathrm{HS}}^2 = -\mathcal{R}(K, f, g) + \|\mathcal{P}\|_{\mathrm{HS}}^2$$

- Let $\kappa(x, x') = \langle \varphi(x), \varphi(x') \rangle$ be a given kernel
- Functions can be embedded as $\mathcal{E}q = \int \varphi(x) q(x) dx$
- Similarity measure:

$$\|q - q'\|_{\mathcal{E}} = \langle \mathcal{E}(q - q'), \mathcal{E}(q - q') \rangle$$

- Consider Perron–Frobenius operator as

$$\mathcal{P}_{s,t} : L^2_{\rho_s^{-1}} \to L^2_{\mathcal{E}}.$$

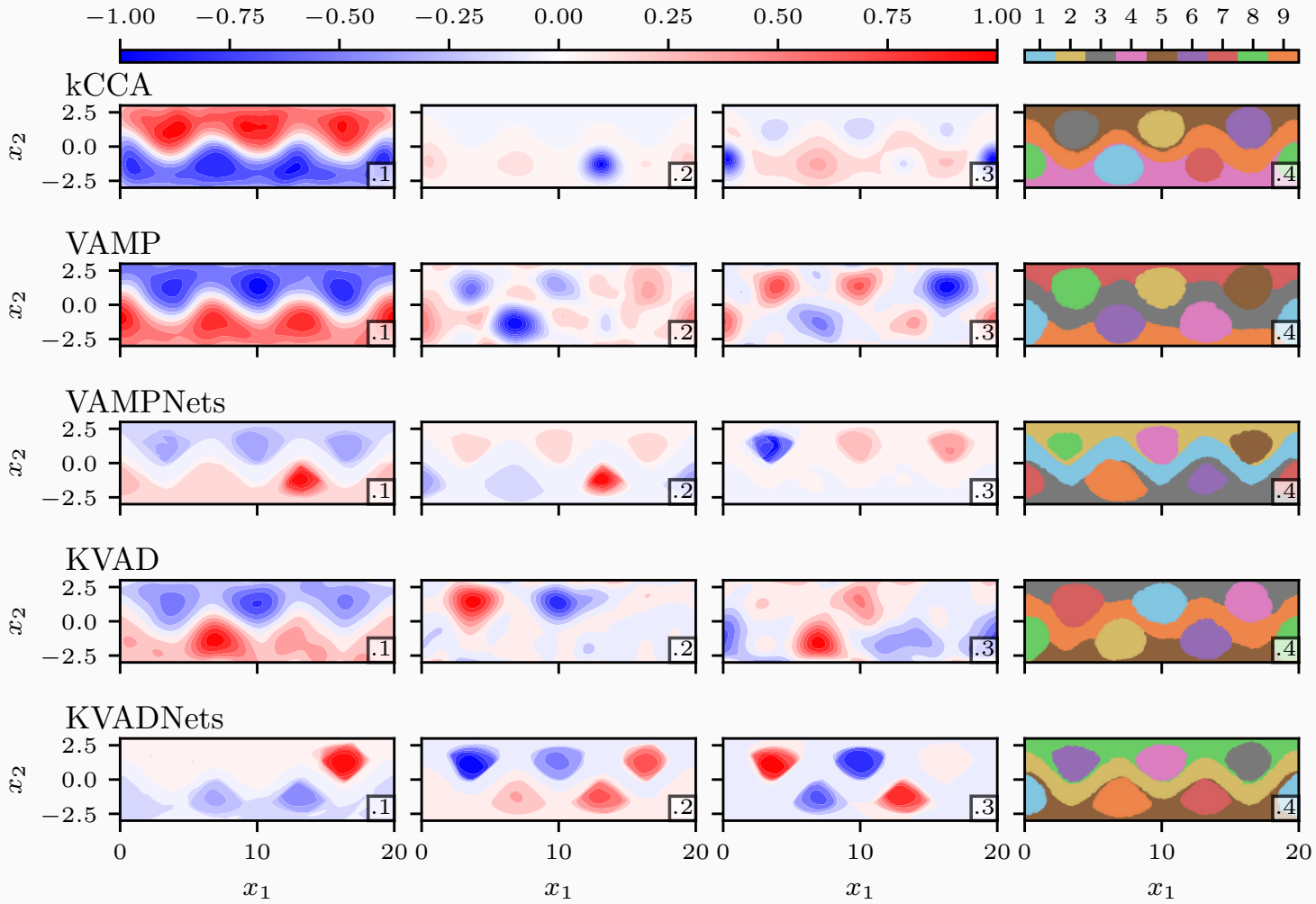
- Ansatz: Represent transition density $\hat{p}_\tau(x_s, x_t) = \mathbf{f}(x_s)^\top \mathbf{q}(x_t)$, where $\mathbf{q}$ density basis functions and $\mathbf{f}$ feature functions.

(Tian and Wu, 2021)

Example estimation

```
from deeptime.data import BickleyJet
from deeptime.clustering import KMeans
from deeptime.decomposition import VAMP

simulator = BickleyJet(h=1e-3, n_steps=int(1. / 1e-3))

# shape (n_particles, 41, 2)
traj = simulator.trajectory(0, Xinit, 41)

initial_pos = traj[:, 0]
final_pos = traj[:, -1]

vamp = VAMP().fit_fetch((initial_pos, final_pos))

clustering = KMeans(k=9).fit_fetch(vamp.transform(initial_pos))
assignments = clustering.transform(vamp.transform(initial_pos))
```

Results


−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00
1 2 3 4 5 6 7 8 9
kCCA
VAMP
VAMPNets
KVAD
KVADNets
x_2
2.5
0.0
−2.5
.1
.2
.3
.4
0 10 20
x_1

Making things more quantitative

Coherence can be described[2] as

$$\left\langle \mathcal{T}^* \mathcal{T} \frac{1_A}{\mu_S(A)}, 1_A \right\rangle_{\mu_S} \approx 1.$$

For a given subdivision of the domain to disjoint coherent sets $\Omega = \bigcup_i A_i$:

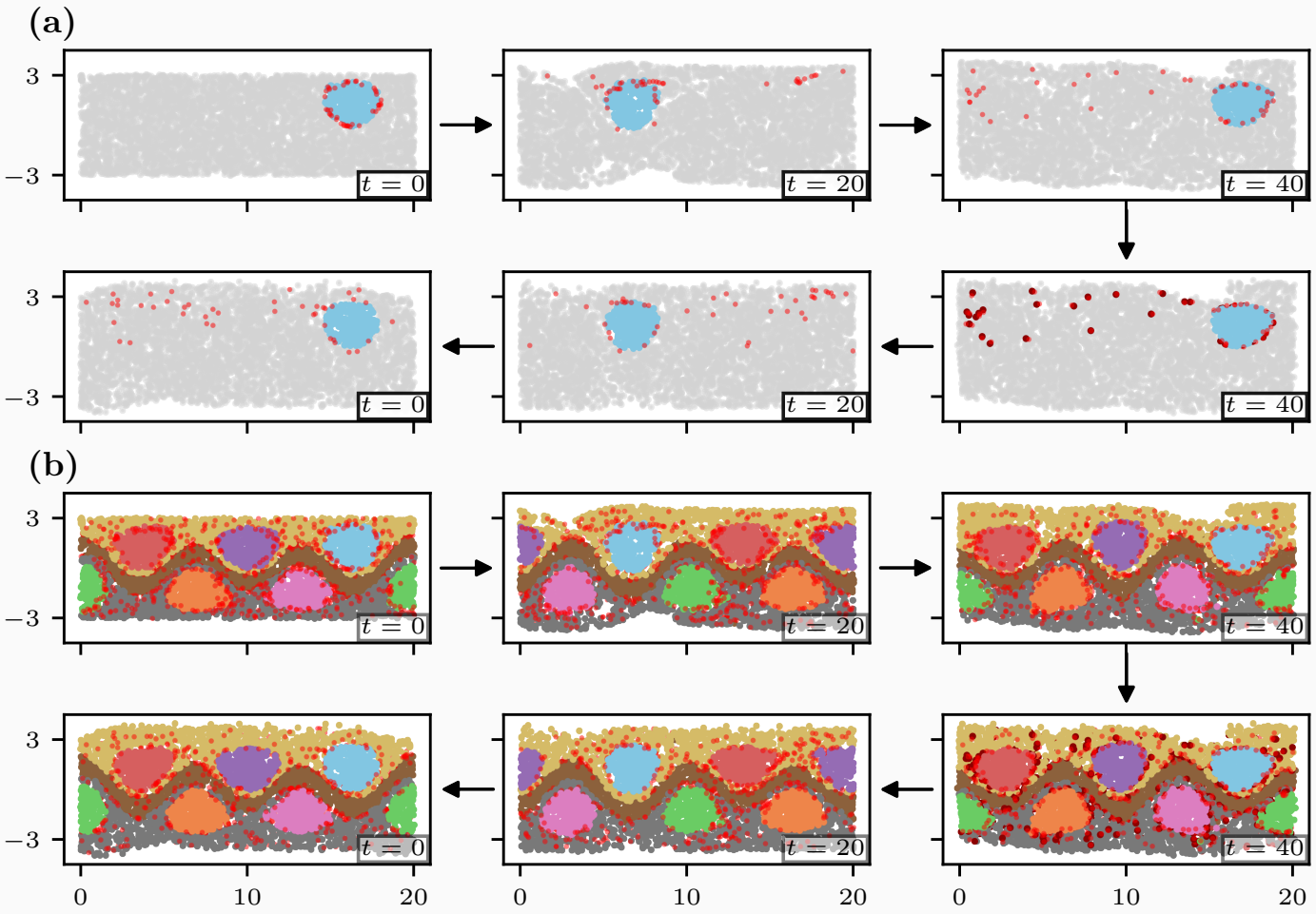
$$s_{\text{leak}}^{(i)} := \mathbb{P}\left[(\Phi_{t_1}^{-1} \circ N_\sigma \circ \Phi_{t_1})(x_{t_0}) \in A_i \mid x_{t_0} \in A_i \right],$$

where Φ is the flow and N_σ the addition of noise. Then:

$$s_{\text{leak}} := \mathbb{E}_{\mu_{t_0}}\left[s_{\text{leak}}^{(i)} \right] = \sum_i \frac{\mu_{t_0}(A_i)}{\mu_{t_0}(\Omega)} s_{\text{leak}}^{(i)}.$$

[2] (Banisch and Koltai, 2017)

Leakage



Leakage score vs VAMP vs KVAD

	KVAD ($\sigma = 1$)	VAMP	VAMPNets	Kernel CCA ($\sigma = 0.58$)	KVADNets ($\sigma = 0.5$)
Leakage	0.74 ± 0.01	0.77 ± 0.01	0.79 ± 0.01	0.85 ± 0.01	0.87 ± 0.01
VAMP-2	4.63 ± 0.06	5.18 ± 0.08	$\mathbf{7.28 \pm 0.06}$	5.77 ± 0.08	6.03 ± 0.09
KVAD ($\sigma = 0.5$)	$0.070 \pm 1.2 \cdot 10^{-3}$	$0.073 \pm 1.1 \cdot 10^{-3}$	$0.078 \pm 1.1 \cdot 10^{-3}$	$0.080 \pm 1.4 \cdot 10^{-3}$	$0.087 \pm 1.2 \cdot 10^{-3}$
KVAD ($\sigma = 0.7$)	$0.115 \pm 1.9 \cdot 10^{-3}$	$0.118 \pm 1.7 \cdot 10^{-3}$	$0.124 \pm 1.6 \cdot 10^{-3}$	$0.125 \pm 1.9 \cdot 10^{-3}$	$0.139 \pm 1.7 \cdot 10^{-3}$
KVAD ($\sigma = 1.0$)	$0.180 \pm 2.7 \cdot 10^{-3}$	$0.184 \pm 2.4 \cdot 10^{-3}$	$\mathbf{0.190 \pm 2.1 \cdot 10^{-3}}$	$0.188 \pm 2.4 \cdot 10^{-3}$	$0.209 \pm 2.3 \cdot 10^{-3}$

For $\sigma < 0.7$, the KVAD score gives the same order as the leakage score.