Deeptime: a Python library for machine learning dynamical models from time series data

MACHINE
LEARNING
Science and Technology

CrossMark

OPEN ACCESS

RECEIVED
5 November 2021

REVISED
24 November 2021

ACCEPTED FOR PUBLICATION
26 November 2021

PUBLISHED
10 December 2021

PAPER

Deeptime: a Python library for machine learning dynamical models from time series data

Moritz Hoffmann[1], Martin Scherer[1], Tim Hempel[1,2], Andreas Mardt[1], Brian de Silva[3,11], Brooke E Husic[1,4,5,6], Stefan Klus[7], Hao Wu[8], Nathan Kutz[3], Steven L Brunton[9] and Frank Noé[1,2,10,*]

[1] Fachbereich Mathematik und Informatik, Freie Universität Berlin, 14195 Berlin, Germany
[2] Fachbereich Physik, Freie Universität Berlin, 14195 Berlin, Germany
[3] Department of Applied Mathematics, University of Washington, Seattle, WA 98105, United States of America
[4] Lewis Sigler Institute for Integrative Genomics, Princeton University, Princeton, NJ 08540, United States of America
[5] Princeton Center for Theoretical Science, Princeton University, Princeton, NJ 08540, United States of America
[6] Center for the Physics of Biological Function, Princeton University, Princeton, NJ 08540, United States of America
[7] Department of Mathematics, University of Surrey, Guildford GU2 7XH, United Kingdom
[8] School of Mathematical Sciences, Tongji University, Shanghai 200092, People's Republic of China
[9] Department of Mechanical Engineering, University of Washington, Seattle, WA 98105, United States of America
[10] Rice University, Department of Chemistry, Houston, TX 77005, United States of America
[11] Work performed prior to employment at Amazon.
[*] Author to whom any correspondence should be addressed.

E-mail: frank.noe@fu-berlin.de

Keywords: machine-learning, time-series analysis, transfer operators, metastable and coherent sets, Markov state models, coarse graining, system identification

Abstract

Generation and analysis of time-series data is relevant to many quantitative fields ranging from economics to fluid mechanics. In the physical sciences, structures such as metastable and coherent sets, slow relaxation processes, collective variables, dominant transition pathways or manifolds and channels of probability flow can be of great importance for understanding and characterizing the kinetic, thermodynamic and mechanistic properties of the system. *Deeptime* is a general purpose Python library offering various tools to estimate dynamical models based on time-series data including conventional linear learning methods, such as Markov state models (MSMs), Hidden Markov Models and Koopman models, as well as kernel and deep learning approaches such as VAMPnets and deep MSMs. The library is largely compatible with scikit-learn, having a range of Estimator classes for these different models, but in contrast to scikit-learn also provides deep Model classes, e.g. in the case of an MSM, which provide a multitude of analysis methods to compute interesting thermodynamic, kinetic and dynamical quantities, such as free energies, relaxation times and transition paths. The library is designed for ease of use but also easily maintainable and extensible code. In this paper we introduce the main features and structure of the deeptime software. Deeptime can be found under https://deeptime-ml.github.io/.

1. Introduction

Deeptime is an open source Python library for the analysis of time-series data; i.e. the provided methods relate to finding relationships between instantaneous data $\mathbf{x}_t$ for some $t \in [0, \infty)$ and corresponding future data $\mathbf{x}_{t+\tau}$ for some so-called *lag-time* $\tau > 0$. Most of the implemented methods try to estimate the behavior of processes when going from $\mathbf{x}_t$ to $\mathbf{x}_{t+\tau}$ by predicting the latter based on the former. The API is similar to what is implemented in the well-known software package scikit-learn [1] and there is basic compatibility of the methods in the two packages via duck typing[12]. Deeptime has two main goals: (1) making methods which were developed in different communities (such as molecular dynamics and fluid dynamics) accessible to a

[12] Duck typing refers to the objects' behavior defining in which contexts it can be used, not so much its concrete type.

broad user base by implementing them in a general-purpose way, and (2) easy to extend and maintain due to modularity and very few hard dependencies.

Deeptime offers the following main groups of methods:

- *Dimension reduction of dynamical data.* One vitally important ingredient to understanding high-dimensional data is projecting them onto a low-dimensional manifold which preserves the 'interesting' parts of the signal. One prominent linear method which can perform this task is principle component analysis (PCA) [2, 3]. While PCA is a widely implemented method, it is not designed for extracting dynamically relevant features from time-series data. Time-lagged independent component analysis (TICA) [4, 5] and dynamic mode decomposition (DMD) [6, 7] provide dimension reduction along with a best-fit linear model. Deeptime offers a range of methods which are based on the mathematical framework of transfer operators [8–12], enabling users to study in particular kinetic properties of the data as well as find temporally metastable and coherent regions.
- *Nonlinear dimension reduction.* While linear methods are widely used due to their simplicity, the high-dimensional data manifold might not always be structured in a way in which important processes are linearly separable. For that reason, deeptime offers some featurizations, explicitly defined basis functions, and kernel methods.
- *Deep dimension reduction.* For nonlinear dimension reduction—especially in the high-data regime—there is also a weak dependency (i.e. no strict requirement for installation) to PyTorch [13], enabling the use of deep learning techniques for dimension reduction of time-series data. A variety of such methods has recently been developed, for example time-lagged Autoencoders [14], linearly-recurrent Autoencoder networks [15], VAMPNets [16], deep generative Markov state models (MSMs) [17, 18], deep Koopman networks [19], and variational dynamical encoders [20]. Some of the mentioned methods are implemented in this software.
- *MSMs.* MSMs [21–29] are stochastic models describing temporal transitions between states in chains of events where each event only depends on its predecessor and has no dependency on events further in the past (known as the Markov property). They also fit into the mathematical framework of transfer operators. Based on MSMs one can estimate in particular kinetic properties from data.
- *Hidden Markov models (HMMs).* HMMs [30, 31] are a type of model consisting of a hidden (i.e. not observable) Markov process emitting an observable output process depending on the hidden process. In comparison to MSMs, HMMs are more expressive and can produce good results where MSMs would not, but are harder to estimate[13].
- *Sparse identification of nonlinear dynamics (SINDy).* SINDy [33] identifies nonlinear governing equations with as few terms as possible from a library of candidate terms that best fit the data. In that way, it complements the dimension-reduction techniques. In particular, while most methods model and analyze the relationships of time-shifted pairs of data, SINDy predicts maps yielding the infinitesimal expected temporal change of the system's current state. On the other hand, SINDy can also predict discrete-time maps by directly relating the system's future state to the system's current state (see section 5 for details).

Deeptime currently focuses on the domain-agnostic estimation of dynamical models and their analysis in terms of physically relevant quantities describing equilibrium or nonequilibrium behavior. The aim of Deeptime is not to provide tools specific for a single domain, such as molecular dynamics, but it can be easily combined with python packages that, e.g. load and featurize domain-specific data files in order to prepare such data for analysis with Deeptime [34–39]. Alternatively there also exist time-series analysis packages that are more domain-specific [16, 40–43] or implement a subset of deeptime's methods but with a wider range of options and/or more flexibility [44–46]. As the dynamical model and its properties take the center stage in Deeptime, its aim is also *not* to perform time-series forecasting, e.g. for weather or financial data, or clustering, regression, and annotation directly on the dynamical data itself. For these types of tasks there is, e.g. the sktime project [47][14].

2. Design and implementation

Deeptime is mainly implemented in and available for Python 3.7+ (as of now[15]) and available for all major operating systems via the Python package index and conda-forge [48]. Some computationally expensive

[13] A more effective/efficient model for hidden Markov processes with discrete output probability distributions is the observable operator model MSM (OOM) [32] that can also be found within the deeptime package.

[14] sktime also provides a curated overview of various projects dealing with time-series data: www.sktime.org/en/latest/related_software.html.

[15] 24 November 2021.

calculations are implemented in C++ using pybind11 [49] or if appropriate using NumPy [50] and SciPy [51].

The API itself is inspired (and largely compatible with) the one used by scikit-learn [1]. In particular, deeptime offers `Estimator` classes, which can be fit on data. An important point at which deeptime's implementation is different to what is offered by scikit-learn is the following: a call to `fit` leads to the creation of a `Model` instance; in particular, estimators can be fit multiple times and each time produce an independent model instance (therefore are model factories). Regarding the structure of data they store, `Models` carry the estimation results and are rather simple classes, that are akin to Python dictionaries. If possible, estimators offer a `partial_fit` method that allows the user to continuously update a model with a stream of data. This is particularly useful if the dataset does not fit into the computer's main memory. Additionally, `Models` may also be `Transformers`, meaning they can `transform` data based on the state of the `Model` instance. In such cases the corresponding `Estimator` also implements the `Transformer` interface, dispatching the call to the latest estimated model.

In comparison, in scikit-learn an `Estimator` is also a `Model` and the estimation results are dynamically attached to the estimator instance. Given that our models come with a large variety of attached methods and properties, we deviate from this paradigm to ensure clarity and component separation and to avoid an overcrowded interface. Furthermore, as our `Models` are relatively lightweighted objects that are divorced from the data they have been trained on, it is straightforward to use the Python pickle module for serialization. This way, `Estimator` instances can be re-used on existing models without side effects, fostering deeptime's applicability to parameter studies.

The number of dependencies is kept as low as possible to reduce maintenance efforts. The base functionality of deeptime only depends on the established packages NumPy [50], Scipy [51], and scikit-learn [1]. Dependencies to plotting routines (matplotlib [52]) and deep learning components (PyTorch [13]) are optional.

The code is hosted on GitHub (https://github.com/deeptime-ml/deeptime) and licensed under LGPLv3, meaning it uses a license with weak copyleft so the library can be used also in proprietary codes. The repository is coupled to the continuous integration service Azure Pipelines, performing automated testing upon changes or proposed changes to the main branch. The project uses the pytest testing framework [53].

The documentation aims for maximal transparency with respect to the implemented methods and the implementation details. To that end, the main methods and their basic usage are explained in Jupyter notebooks [54] with some theoretical background, references, and illustrative examples. The detailed API documentation is generated directly from the Python source code, so that it can be referred to while using the software but also while developing new components or fixing bugs. Furthermore, there are short example scripts for the datasets and selected methods, compiled into example galleries. All this is rendered into HTML and transparently hosted on GitHub pages using Sphinx under https://deeptime-ml.github.io/.

The deeptime library is structured in such a way that the entire user interface is exposed at package-level. We structure the (sub-)packages as follows:

- `deeptime.base`: Contains all the basic classes of deeptime, in particular the interface definitions for `Estimators`, `Models`, and `Transformers`.
- `deeptime.basis`: A set of basis functions which can be used for SINDy and some of the dimension reduction algorithms as ansatz and/or featurization.
- `deeptime.kernels`: A set of predefined kernels which can be used in kernel methods. Some of these possess subclasses with a `Torch` prefix, indicating that they are PyTorch-ready and support batched evaluation as well as backpropagation.
- `deeptime.sindy`: Contains an implementation of the SINDy estimator (see section 5).
- `deeptime.covariance`: Methods for estimating covariance and autocorrelation matrices from time-series data in an online fashion. These are mainly used by some of the decomposition methods (see section 3.3).
- `deeptime.decomposition`: Decomposition methods for time-series data (see section 3 for a comprehensive list of implemented estimators).
- `deeptime.markov`: Analysis tools, validators, and estimators for MSMs and HMMs (see section 4).
- `deeptime.clustering`: A collection of clustering/discretization algorithms. These are mostly intended for assigning frames to discrete states (potentially after using one of the dimension reduction algorithms) and subsequently estimating MSMs or HMMs.
- `deeptime.numeric`: A collection of numerical utilities, most notably for eigenvalue problems and regularized inverses of symmetric positive semi-definite matrices.
- `deeptime.data`: A selection of example data on which the algorithms can be tested (see section 6).
- `deeptime.util.data`: Utilities which relate to data processing, e.g. time-series specific `DataSet` implementations which can be used in conjunction with PyTorch.

Some of the implementations are based on the molecular-dynamics analysis package PyEMMA 2 [41, 42] including its dependencies bhmm [55] and msmtools[16]—modified so that they are no longer dependent on any molecular-dynamics specific libraries and offer greater flexibility—and on the dynamical systems toolbox d3s[17]. The `deeptime.sindy` package is based on and compatible to PySINDy [46]. The `deeptime.decomposition` package contains an implementation of DMD [6, 7, 56, 57]. For a richer feature set and different variants and flavors of DMD we recommend the PyDMD package [44].

3. Dimension reduction and decomposition methods

Deeptime offers a range of methods that can be used to reduce the dimension of observed data by projecting it onto dominant processes. This relates to the mathematical framework of transfer operators [8, 9, 11, 58–60]. We regard all operators that describe the temporal evolution of, e.g. probability densities or observables of the system's state as transfer operators. The operators we consider here are all linear operators (although in general not finite-dimensional).

For an introduction to these operators we follow the presentation of [59]. We distinguish two different cases: time-homogeneous processes, which possess transition probabilities that do not depend on a particular point in time (this is the case for, e.g. autonomous differential equations) and the more general case of time-inhomogeneous processes.

3.1. Time-homogeneous processes

Let $\{\mathbf{x}_t\}_{t\geqslant 0}$ be a Markovian and time-homogeneous stochastic process in state space $\mathbf{x}_t \in \Omega \subset \mathbb{R}^d$ with transition density

$$p_{s,t} : \Omega \times \Omega \to \mathbb{R}_{\geqslant 0}, \quad \mathbb{P}[\mathbf{x}_t \in B \mid \mathbf{x}_s = x] = \int_B p_{s,t}(\mathbf{x}, \mathbf{y})\mathrm{d}\mathbf{y}, \tag{1}$$

which is the probability of finding state $\mathbf{x}_t$ in a measurable set $B \subset \Omega$ given state $\mathbf{x}$ at time s. Time-homogeneity means that $p_{s,t}$ only depends on a lag-time $\tau := t - s$ but not on specific start and end times s and t individually, i.e.

$$p_{s,t}(\mathbf{x}, \mathbf{y}) = p_\tau(\mathbf{x}, \mathbf{y}). \tag{2}$$

However, this does not mean that the law (or distribution) of the process

$$B \mapsto \mathrm{law}(\mathbf{x}_t)[B] := \mathbb{P}[\mathbf{x}_t \in B]$$

for sets $B \subset \Omega$ is time-independent. For example Brownian motion is a time-homogeneous process, however its law for a single particle at initial time is given by a delta peak in the initial position and converges to a uniform spatial distribution over time.

Generally speaking, transfer operators describe the effect of the underlying dynamics on functions of the state $\mathbf{x}_t$. A particularly important transfer operator, the Koopman operator (first introduced in [8]), is defined as

$$\mathcal{K}_\tau : L^\infty(\Omega) \to L^\infty(\Omega), \quad [\mathcal{K}_\tau g](\mathbf{x}) := \int g(\mathbf{y}) p_\tau(\mathbf{x}, \mathbf{y})\mathrm{d}\mathbf{y} = \mathbb{E}[g(\mathbf{x}_{t+\tau}) \mid \mathbf{x}_t = \mathbf{x}], \tag{3}$$

evolving the observable g for a lag-time $\tau > 0$. The function space[18] $g \in L^\infty(\Omega)$ is of the family of L^p spaces with

$$L^p(\Omega) := \left\{ f : \Omega \to \mathbb{C} \text{s.t.}\, f \,\text{measurable}\ \wedge \|f\|_p := \left(\int_\Omega |f|^p \right)^{1/p} < \infty \right\}$$

for $1 \leqslant p < \infty$ and $L^\infty(\Omega) := \{ f : \Omega \to \mathbb{C}\ \text{s.t.}\, f\,\text{measurable}\ \wedge \exists C \geqslant 0 : |f(x)| \leqslant C\ \text{a.e.}\}$. In case of deterministic dynamics $\mathbf{x}_{t+\tau} = \mathbf{\Psi}(\mathbf{x}_t)$, the transition density consists of delta peaks and the Koopman operator is simply the composition $\mathcal{K}_\tau g = g \circ \mathbf{\Psi}$.

[16] https://github.com/markovmodel/msmtools.

[17] https://github.com/sklus/d3s.

[18] Strictly speaking L^p consists of equivalence classes of measurable functions where the equivalence relation is defined by functions being equal 'almost everywhere', i.e. can differ on sets of measure zero.

Another commonly used transfer operator to describe Markovian dynamics is the Perron–Frobenius (PF) operator [61, 62]

$$\mathcal{P}_\tau : L^1(\Omega) \to L^1(\Omega), \quad [\mathcal{P}_\tau f](\mathbf{y}) = \int \mathbf{f}(\mathbf{x}) p_\tau(\mathbf{x}, \mathbf{y}) \mathrm{d}\mathbf{x}, \tag{4}$$

which evolves probability density functions $f \in L^1(\Omega)$. Since it is a Markov operator ($\mathcal{P}_\tau f \geqslant 0$ and $\|\mathcal{P}_\tau f\| = \|f\|$ for all $f \geqslant 0$), probability density functions are mapped to probability density functions [61].

The PF operator is the adjoint of the Koopman operator [61, 62], i.e.

$$\langle \mathcal{P}_\tau f, g \rangle = \langle f, \mathcal{K}_\tau g \rangle \quad \forall f \in L^1(\Omega), g \in L^\infty(\Omega), \tag{5}$$

where the bracket is defined as $\langle h_1, h_2 \rangle := \int_\Omega h_1(\mathbf{x}) h_2(\mathbf{x}) \mathrm{d}\mathbf{x}$. Although L^p spaces with $p \neq 2$ are not Hilbert spaces, the product of two functions $h_1 \in L^p(\Omega)$ and $h_2 \in L^q(\Omega)$ is integrable as long as $1/p + 1/q = 1$ for $1 \leqslant p, q \leqslant \infty$.

For the rest of this section we assume that there exists a stationary distribution $\mu \in L^1(\Omega)$ satisfying $\mathcal{P}_\tau \mu = \mu$. If such a stationary distribution μ exists and $\mu(x) > 0$ almost everywhere, then the time-homogeneous processes $\{\mathbf{x}_t\}_{t \geqslant 0}$ is ergodic and the stationary distribution is unique. Vice versa, if $\{\mathbf{x}_t\}_{t \geqslant 0}$ is ergodic, there exists at most one stationary distribution [61].

Given the stationary distribution we can define a PF operator with respect to μ (also simply called the transfer operator),

$$\mathcal{T}_\tau : L^1(\Omega) \to L^1(\Omega), \quad [\mathcal{T}_\tau u](\mathbf{y}) = \frac{1}{\mu(\mathbf{y})} \int \mu(\mathbf{x}) u(\mathbf{x}) p_\tau(\mathbf{x}, \mathbf{y}) \mathrm{d}\mathbf{x}. \tag{6}$$

Instead of evolving probability densities f, it evolves densities $u = f/\mu$ with respect to the stationary distribution. Due to this construction we obtain the normalization $\mathcal{T}_\tau \mathbb{1} = \mathbb{1}$, encoding that the stationary distribution is preserved under propagation in time.

Under some conditions [10, 11, 60, 63], the function spaces from and to which the operators map can be assumed to be reweighted L^2 spaces,

$$L^2_\rho(\Omega) := \left\{ h : \|h\|^2_\rho < \infty \text{ with } \langle f, g \rangle_\rho := \int_\Omega f(\mathbf{x}) \overline{g(\mathbf{x})} \rho(\mathbf{x}) \mathrm{d}\mathbf{x} \right\}, \tag{7}$$

where $\mathcal{P}_\tau : L^2_{\mu^{-1}}(\Omega) \to L^2_{\mu^{-1}}(\Omega)$, $\mathcal{T}_\tau : L^2_\mu(\Omega) \to L^2_\mu(\Omega)$, and $\mathcal{K}_\tau : L^2_\mu(\Omega) \to L^2_\mu(\Omega)$. In what follows we assume that this is the case.

Via a straightforward calculation using (5) one obtains that Koopman operator and transfer operator are also adjoint in the reweighted spaces, i.e.

$$\langle \mathcal{T}_\tau f, g \rangle_\mu = \langle f, \mathcal{K}_\tau g \rangle_\mu \quad \forall f, g \in L^2_\mu(\Omega). \tag{8}$$

3.2. Time-inhomogeneous processes

In the case of time-inhomogeneous processes, the transition density (1) depends directly on the initial and/or final time; i.e. equation (2) no longer holds. This also means that the operators (3)–(6) no longer depend on the lag-time τ but rather on specific start and end times s and t, respectively (equivalently: on start time s and with lag-time τ). For such systems there is in general no stationary distribution μ, so we consider the distribution μ_s at initial time s and μ_t at final time t, related by $\mu_t = \mathcal{P}_{s,t} \mu_s$. The transfer operator can be defined as

$$\mathcal{T}_{s,t} : L^2_{\mu_s}(\Omega) \to L^2_{\mu_t}(\Omega), \quad \mathcal{T}_{s,t} u = \frac{1}{\mu_t} \mathcal{P}_{s,t}(u \mu_s). \tag{9}$$

As in the time-homogeneous case, this operator is the adjoint of the time-inhomogeneous Koopman operator [64]

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

where $\mathcal{K}_{s,t} : L^2_{\mu_t}(\Omega) \to L^2_{\mu_s}(\Omega)$.

For the remainder of this section we will simplify the notation to $\mathcal{P}$, $\mathcal{T}$, and $\mathcal{K}$ for the PF, transfer, and Koopman operators, respectively. Also we often wish to consider/use vector-valued feature functions, in which case it is assumed that the transfer operators act component-wise.

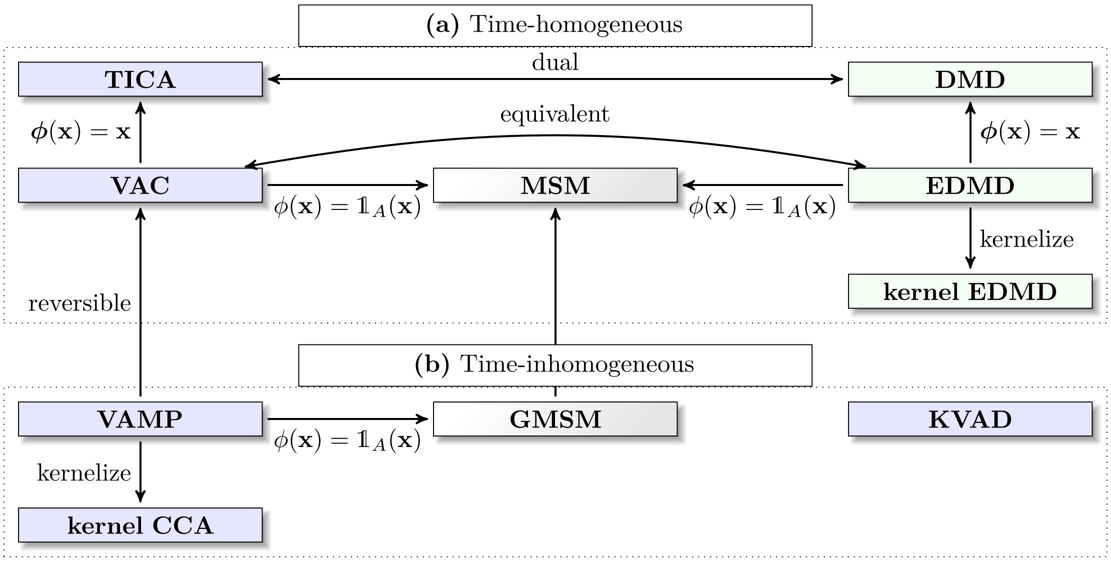

(a) Time-homogeneous
TICA
dual
DMD
$\phi(\mathbf{x}) = \mathbf{x}$
equivalent
$\phi(\mathbf{x}) = \mathbf{x}$
VAC
MSM
EDMD
$\phi(\mathbf{x}) = \mathbb{1}_A(\mathbf{x})$
$\phi(\mathbf{x}) = \mathbb{1}_A(\mathbf{x})$
kernelize
kernel EDMD
reversible
(b) Time-inhomogeneous
VAMP
GMSM
KVAD
$\phi(\mathbf{x})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

Figure 1. Relationships of conventional dimension reduction methods. Methods shaded in green are regression-based, while methods shaded in blue are based on a variational principle. (a) Methods assuming time-homogeneous dynamics. While the 'variational approach to conformational dynamics' (VAC) can only be applied under time-reversible dynamics, 'extended dynamic mode decomposition' (EDMD) makes no assumptions about the dynamics' reversibility. (b) Methods supporting time-inhomogeneous dynamics. While the 'kernel embedding based variational approach for dynamical systems' (KVAD) is based on a variational principle, the ansatz and in particular the estimated transfer operator is a PF operator in contrast to the 'variational approach for Markov processes' (VAMP), which estimates an approximation of the Koopman operator.

One particular advantage of considering any of the transfer operators over directly analyzing the (in general highly nonlinear) temporal evolution of processes' full states is their linearity. While the considered operator usually cannot be represented as a finite-dimensional matrix, one can seek projections and/or approximations. These approximations can be used to identify and project onto the slow processes as well as metastable and coherent sets [10, 65, 66]. There are different methods available for making the approximations which vary in their assumptions, approximation power, and interpretability, some of which are accompanied by variational theorems.

3.3. Conventional dimension reduction and decomposition

The conventional machine learning estimators for dimension reduction supported by deeptime are detailed below. For more thorough introductions to available methods and overviews of their relationships, we refer the reader to [11, 67, 68]. Most of the following methods seek a matrix $K \in \mathbb{R}^{m\times m}$, a finite-dimensional approximation of a transfer operator that should fulfill

$$\mathbb{E}[\mathbf{g}(\mathbf{x}_{t+\tau})] = K^\top \mathbb{E}[\mathbf{f}(\mathbf{x}_t)] \tag{10}$$

as closely as possible for time series data $\mathbf{x}_t$. The system's state $\mathbf{x}_t$ is transformed into feature space by $\mathbf{f}, \mathbf{g} \in \mathcal{F}^m$, where $\mathcal{F}$ is the space of scalar feature functions.

We give an overview of conventional dimension reduction methods in figure 1, all of which reside in the `deeptime.decomposition` subpackage. Roughly, the methods can be divided into groups of estimators that are restricted to data observed from time-homogeneous systems (figure 1(a)) and estimators that are also capable of working with data of time-inhomogeneous systems (figure 1(b)).

Another distinction can be made by considering the estimation approach of the respective methods. While some are regression-based (green shade in figure 1), others (purple shade) operate within the framework of an underlying variational principle.

In what follows, we have instantaneous data $\mathbf{x}_i \in \mathbb{R}^d$ and time-lagged data $\mathbf{y}_i \in \mathbb{R}^d$ organized into matrices $X = [\mathbf{x}_1, \ldots, \mathbf{x}_n] \in \mathbb{R}^{d\times n}$ and $Y = [\mathbf{y}_1, \ldots, \mathbf{y}_n] \in \mathbb{R}^{d\times n}$, respectively.

3.3.1. Dynamic mode decomposition (DMD)

DMD [6, 7, 56, 57] was introduced in the fluid dynamics community to extract spatiotemporal coherent structures from high-dimensional time series data. It is closely related to (10) in the sense that its objective is to solve the regression problem

$$\min_M \|Y - M_{\mathrm{DMD}} X\|_F \tag{11}$$

for a matrix $M_{\mathrm{DMD}} \in \mathbb{R}^{d\times d}$. A subsequent spectral analysis of M_{DMD} can reveal information about the dominant dynamics of the system.

There are a variety of extensions to DMD. For example, DMD algorithms have been developed that incorporate control [69], promote sparsity [70], are randomized [71], and act on time delay vectors [72] (the last has a relationship to Koopman operator analysis). Bagheri [73] demonstrated the sensitivity of the DMD algorithm to measurement noise, motivating several noise-robust variants: total least squares DMD [74], forward backward DMD [75], Bayesian DMD [76], optimized DMD [77], and variational DMD [78].

While deeptime offers a basic version of DMD, most of these extensions are currently not available. The PyDMD Python package [44] offers a broad range of DMD based methods.

3.3.2. Extended dynamic mode decomposition (EDMD)

EDMD [79] defines a basis set of functions or observables $B := \{\psi_1, \ldots, \psi_m\} \subset \mathcal{F}$ to construct the vector-valued function $\mathbf{\Psi}(x) = (\psi_1(x), \ldots, \psi_m(x))^\top \in \mathcal{F}^m$. The sought-after matrix K with respect to $\mathbf{f} = \mathbf{g} = \mathbf{\Psi}$ is the solution of the regression problem

$$\hat{K} = \mathrm{argmin}_K \|\mathbf{\Psi}(Y) - K\mathbf{\Psi}(X)\|_F \in \mathbb{R}^{m\times m}, \tag{12}$$

where the application of $\mathbf{\Psi}$ is column-wise.

A projection onto dominant processes can be found by applying the eigenfunctions of the Koopman operator deduced from $\hat{K}$ and $\mathbf{\Psi}$ corresponding to the largest eigenvalues to the transformed input data.

The solution of the regression problem (12) is an approximate version of the desired property (6) for specific choices of $\mathbf{f}$ and $\mathbf{g}$. In deeptime this is implemented by the model of the EDMD estimator being a `TransferOperatorModel` (see figure 2).

As its name suggests, DMD can be understood as a special case of EDMD in which the feature basis contains only the identity function, i.e. $\mathbf{\Psi}(\mathbf{x}) = \mathbf{x}$. If we define the set of basis functions to contain indicator functions for a given discretization of the state space, EDMD estimates MSMs (see figure 1 and section 4 for details on MSMs).

3.3.3. Time-lagged independent component analysis (TICA)

TICA [4] is a linear transformation method which was introduced for molecular dynamics in [5], was independently derived as a method for extracting the slow molecular order parameters by invoking the variational approach for conformation dynamics (see below) [80], and introduced as a method for constructing high-accuracy MSMs in [80, 81]. TICA is designed for time-homogeneous processes and also assumes that the process is reversible, although it may still perform well practically when applied outside these constraints. A process is defined to be reversible if it fulfills the detailed balance condition

$$\mu(\mathbf{x})p_\tau(\mathbf{x},\mathbf{y}) = \mu(\mathbf{y})p_\tau(\mathbf{y},\mathbf{x}) \quad \forall \mathbf{x},\mathbf{y} \in \Omega. \tag{13}$$

As a consequence, the transfer (6) and Koopman (3) operators are identical and therefore self-adjoint. Assuming $\mathcal{K}$ to be a Hilbert–Schmidt operator, this means that (using the Hilbert–Schmidt theorem) there is an eigenvalue decomposition

$$\mathcal{K} = \sum_{i=1}^{\infty} \lambda_i \langle \cdot, \varphi_i \rangle_\mu \varphi_i, \tag{14}$$

where φ_i are eigenfunctions with $\langle \varphi_i, \varphi_{i'} \rangle_\mu = \delta_{ii'}$ and λ_i are eigenvalues which are real and bounded by the eigenvalue $\max_i \lambda_i = 1$ with a multiplicity of one (see [82]).

The objective of TICA is to yield components which are uncorrelated and also maximize the time-autocorrelation under lag-time τ. To this end, one can solve the generalized eigenvalue problem

$$C_{0\tau}\hat{\varphi}_i = \hat{\lambda}_i C_{00}\hat{\varphi}_i, \tag{15}$$

where $C_{00} = \frac{1}{n-1}XX^\top$ is the instantaneous covariance matrix and $C_{0\tau} = \frac{1}{n-1}XY^\top$ is the time-lagged covariance matrix. The reversibility assumption leads to $C_{00} = C_{\tau\tau}$ and $C_{0\tau} = C_{\tau 0} = C_{0\tau}^\top$ and therefore eigenvalues $\hat{\lambda}_i \in \mathbb{R}$. Because real numbers possess a total order, we can assume that the eigenvalues $\hat{\lambda}_i$ are in a descending order and the transformation $\hat{\boldsymbol{\varphi}}(\cdot) = [\hat{\varphi}_1(\cdot), \ldots, \hat{\varphi}_k(\cdot)]$ is the TICA projection onto the first k dominant components. The corresponding eigenvalues can be related to relaxation timescales of the processes $\hat{\varphi}_i$ [80]. Therefore, if we know *a priori* that the system is time-homogeneous and reversible, TICA can be more data-efficient and yield more interpretable results compared to methods, which do not make these assumptions.

For a comparison with DMD it is useful to identify TICA with $M_{\text{TICA}} = C_{00}^{\dagger} C_{0\tau}$, where $C_{00}^{\dagger}$ denotes the Moore–Penrose pseudoinverse. It can be seen that $M_{\text{TICA}} = M_{\text{DMD}}^{\top}$ [11]. Therefore, the TICA transformation consists of Koopman eigenfunctions projected onto the basis spanned by $\psi_k(\mathbf{x}) = \mathbf{x}_k$ and the DMD modes are the corresponding Koopman modes—, i.e. the coefficients $\eta_k = (\eta_{k1}, \ldots, \eta_{kd})^{\top}$ required to write the kth component of the full-state observable in terms of eigenfunctions $g_k(\mathbf{x}) = \mathbf{x}_k = \sum_i \eta_{ki}\varphi_i(\mathbf{x})$ [11, 79].

This relationship is reflected in figure 1 by identifying DMD and TICA as 'dual'. This duality can also be found within the deeptime software: in contrast to DMD, TICA is a subclass of `TransferOperatorModel` (see figure 2).

3.3.4. Variational approach for conformational dynamics (VAC)

Like TICA, VAC [82, 83] assumes time-homogeneous and reversible dynamics. Similar to the generalization from DMD to EDMD, VAC generalizes TICA using a basis $B := \{\psi_1, \ldots, \psi_m\} \subset \mathcal{F}$ to construct a transformation $\boldsymbol{\Psi}(\mathbf{x}) = (\psi_1(\mathbf{x}), \ldots, \psi_m(\mathbf{x}))^{\top}$. Subsequently the instantaneous and time-lagged data is transformed to $\boldsymbol{\Psi}(X)$ and $\boldsymbol{\Psi}(Y)$, respectively, and used in the TICA problem instead of X and Y. From this it becomes clear that TICA can be understood as a special case of VAC with $\boldsymbol{\Psi}(\mathbf{x}) = \mathbf{x}$ (see figure 1). Because it is algorithmically identical to TICA under a prior featurization of data, there is no dedicated VAC estimator in deeptime. Under the particular choice of basis functions being indicator functions, VAC estimates MSMs (see figure 1 and section 4 for details on MSMs).

As its name suggests, VAC involves a variational bound. It defines the score $s_{\text{VAC}} := \sum_i \hat{\lambda}_i$ which is bounded from above by the sum over the eigenvalues of the true Koopman operator and therefore expresses how much of the slow dynamics is captured in the projection [82, 83]. The score can be used to optimize the feature functions $\boldsymbol{\Psi}$. We will see in the following paragraph that under the assumption of reversible dynamics, the VAC score is equal to the Variational approach for Markov processes (VAMP)-1 score, which is why deeptime only offers a VAMP score implementation. Assuming reversible dynamics, VAC is equivalent to EDMD (see figure 1).

3.3.5. Variational approach for Markov processes (VAMP)

VAMP [60], sometimes also referred to as 'time-lagged canonical correlation analysis' (TCCA) [84], not only optimizes for K but also optimizes for $\mathbf{f}$ and $\mathbf{g}$. This cannot be achieved by merely solving the regression problem (12)—as, e.g. the trivial model $\mathbf{f} = \mathbf{g} \equiv (1, \ldots, 1)^{\top}$, $K = \text{Id}$ is not informative but yields zero error. Instead, VAMP minimizes the left-hand side of

$$\|\mathcal{K} - \hat{\mathcal{K}}\|_{\text{HS}}^2 = -\mathcal{R}(\mathbf{f}, \mathbf{g}) + \|\mathcal{K}\|_{\text{HS}}^2, \tag{16}$$

the Hilbert–Schmidt norm of the difference between true Koopman operator (3) and approximated Koopman operator $\hat{\mathcal{K}}$ deduced from K, $\mathbf{f}$, and $\mathbf{g}$. The minimization is achieved by maximizing $\mathcal{R}$, a variational score. The decomposition (16) of the modeling error assumes that $\mathcal{K}$ is indeed a Hilbert–Schmidt operator.

In [60] it was shown that the smallest approximation error (16) is achieved for

$$\hat{\mathcal{K}} = \sum_{i=1}^{m} \sigma_i \langle \cdot, \phi_i \rangle \psi_i, \tag{17}$$

where $K = \text{diag}(\sigma_1, \ldots, \sigma_m)$, $\mathbf{f} = (\psi_1, \ldots, \psi_m)$, $\mathbf{g} = (\phi_1, \ldots, \phi_m)$, and σ_i, ψ_i, ϕ_i are the square root of the ith eigenvalue, left eigenfunction, and right eigenfunction of the forward-backward operator $\mathcal{K}^*\mathcal{K}$, respectively.

During estimation (similar to TICA and VAC), covariance matrices are estimated and under regularization inverted to perform whitening operations to finally construct an approximation of the Koopman operator. One obtains coefficient matrices $U, V \in \mathbb{R}^{m\times k}$ and the matrix $K \in \mathbb{R}^{k\times k}$, so that

$$\mathbb{E}[V^{\top}\boldsymbol{\chi}_1(\mathbf{x}_{t+\tau})] \approx K^{\top}\mathbb{E}[U^{\top}\boldsymbol{\chi}_0(\mathbf{x}_t)], \tag{18}$$

where $\boldsymbol{\chi}_0$ and $\boldsymbol{\chi}_1$ are vectors of basis functions which optimally should contain ψ_i and ϕ_i in their span, respectively.

The family of VAMP-r scores,

$$\mathcal{R}_r := \sum_i \sigma_i^r, \tag{19}$$

as well as the VAMP-E score (see [60] for a definition) can be optimized to minimize the model error on the left-hand side of (16) and therefore can be used to select optimal features and/or observables by using cross-validation techniques (see, e.g. [85]). These scores give rise to the 'variational' aspect of VAMP as they are bounded from above and their maximization leads to better approximations.

VAMP therefore generalizes VAC to a time-inhomogeneous and nonreversible setting (recall figure 1). While VAMP is applicable in more situations, i.e. because it possesses greater generality and nonequilibrium dynamics are more common in nature, it also loses some of its interpretability—as, e.g. singular values can in general no longer by related to relaxation timescales of processes.

The deeptime library reflects the mathematics of the VAMP approach by the VAMP estimator producing a `CovarianceKoopmanModel`, an extension of the `TransferOperatorModel`, which in particular allows the evaluation of VAMP scores (see figure 2). The estimator can deal with large amounts of data, because the estimation procedure is based on the decomposition of covariance matrices, which can be constructed incrementally [86]. Furthermore, TICA is a subclass of VAMP, as the two methods are algorithmically closely related[19].

Analogously to VAC, the choice of indicator feature functions leads to generalized MSMs (GMSMs) [59], which are also applicable to time-inhomogeneous systems (see figure 1).

3.3.6. Kernel canonical correlation analysis (kernel CCA)

Kernel CCA [87] is a kernelized version of canonical correlation analysis [88] that seeks to maximize the correlation between two multidimensional random variables X and Y (pairs of instantaneous and time-lagged data, respectively). In kernel CCA, the standard inner products are replaced by a kernel function $\kappa(\cdot,\cdot)$ using the 'kernel trick'. Deeptime has a subpackage dedicated to kernel implementations (`deeptime.kernels`), containing (amongst others) vectorized versions of the popular Gaussian kernel

$$\kappa(\mathbf{x},\mathbf{x}') = \exp\left(-\frac{1}{2}\|\mathbf{x}-\mathbf{x}'\|_2^2/\sigma^2\right) \tag{20}$$

as well as the polynomial kernel $\kappa(\mathbf{x},\mathbf{x}') = (c+\mathbf{x}^\top\mathbf{x}')^p$.

It was shown in [67] that kernel CCA can be derived from optimizing the VAMP-1 score (19) within a kernel approach and thus can be understood as a kernelized version of VAMP (for this reason it is sometimes referred to as kernel VAMP).

In addition to the kernel parameters, the estimator also possesses a regularization parameter ε, as kernel CCA involves inverting covariance operators (which on their own are generally not invertible).

3.3.7. Kernel extended dynamic mode decomposition (Kernel EDMD)

Kernel EDMD [89, 90] is, analogously to kernel CCA, a kernelized version of EDMD. In contrast to kernel CCA, it assumes a time-homogeneous process. Furthermore, kernel EDMD requires a regularziation parameter ε in order to ensure invertibility of covariance operators.

3.3.8. Kernel embedding based variational approach for dynamical systems (KVAD)

KVAD [63] is an alternative to VAMP which can also be applied to systems in which the transfer operator $\mathcal{T}$ is not Hilbert–Schmidt as an operator from $L^2_{\mu_s}$ to $L^2_{\mu_t}$[20], which is (e.g.) the case for some deterministic systems. To this end, the similarity of functions of interest is not determined using norms of L^2 function spaces but rather using kernel embeddings of said functions. In particular, for a given kernel $\kappa(\mathbf{x},\mathbf{x}') = \langle\varphi(\mathbf{x}),\varphi(\mathbf{x}')\rangle$, functions q can be embedded via

$$\mathcal{E}q = \int \varphi(\mathbf{x})q(\mathbf{x})\mathrm{d}\mathbf{x}. \tag{21}$$

The similarity between functions q and $q^{'}$ can then be measured as

$$\|q-q'\|_{\mathcal{E}} = \langle\mathcal{E}(q-q'),\mathcal{E}(q-q')\rangle. \tag{22}$$

In [63] it was shown that for universal and bounded kernels κ, the Hilbert–Schmidt assumption is always fulfilled if the PF operator is considered as

$$\mathcal{P}_\tau : L^2_{\mu_s^{-1}} \to L^2_{\mathcal{E}}, \tag{23}$$

[19] While TICA and VAC are special cases of VAMP, in deeptime the estimators are not combined into one due to differences in how covariance matrices are estimated—in particular, TICA's stronger inductive bias is implemented by forced symmetrizations which are not applicable to VAMP—and differences in the decomposition (eigenvalue decomposition and singular value decomposition for TICA and VAMP, respectively).

[20] In case of time-homogeneous processes, we have $\mu_s = \mu_t = \mu$, which is the stationary distribution.

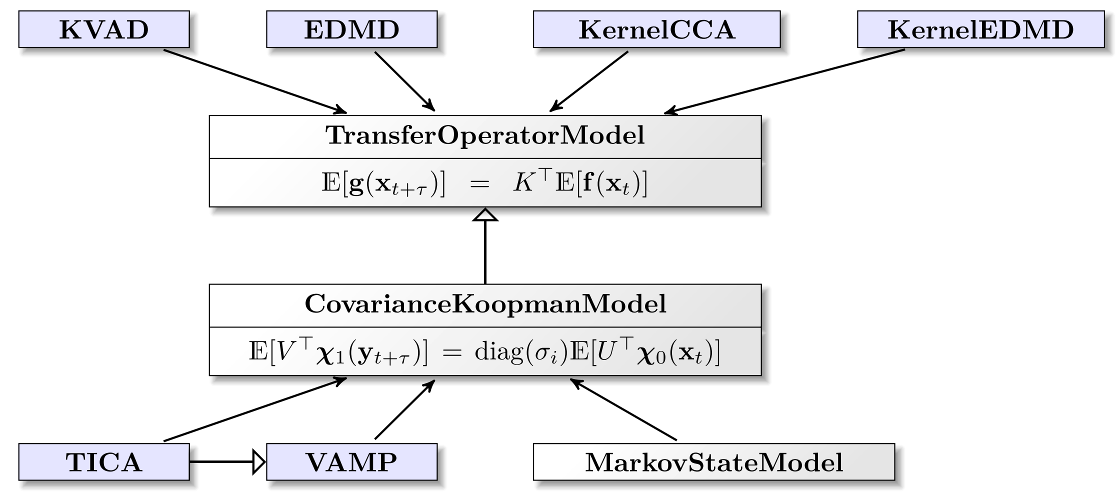

KVAD
EDMD
KernelCCA
KernelEDMD
TransferOperatorModel
$\mathbb{E}[\mathbf{g}(\mathbf{x}_{t+\tau})] = K^\top \mathbb{E}[\mathbf{f}(\mathbf{x}_t)]$
CovarianceKoopmanModel
$\mathbb{E}[V^\top \chi_1(\mathbf{y}_{t+\tau})] = \mathrm{diag}(\sigma_i)\mathbb{E}[U^\top \chi_0(\mathbf{x}_t)]$
TICA
VAMP
MarkovStateModel

Figure 2. Class diagram illustrating relationships between estimators producing approximations of transfer operators. Estimators have a blue background while models are shaded in gray. The `TransferOperatorModel` implements observable transforms $f(\cdot)$ and $g(\cdot)$ as well as propagation of $f(x_t)$ with the Koopman matrix K. It is produced (➔) by `EDMD`, `KernelEDMD`, `KernelCCA`, and `KVAD` estimators. The the `CovarianceKoopmanModel` extends (—▷) the `TransferOperatorModel`. It assumes the estimation to be based on covariance matrices and defines the Koopman matrix in a whitened space, where χ_0 and χ_1 are basis transformations of the state x_t and $x_{t+\tau}$, respectively, and U and V are basis transform matrices. The Koopman matrix is then a diagonal matrix. The `CovarianceKoopmanModel` can be produced by `TICA`, `VAMP`, and `MarkovStateModel`s and additionally possesses a `score()` function.

where $L^2_\mathcal{E} = \{f \in L^2 : \|f\|_\mathcal{E} < \infty\}$ is an L^2 space equipped with the kernel similarity measure (22). Note that in this case the PF operator as defined in (23) is in general no longer the adjoint of the Koopman operator.

Like VAMP, KVAD is based on the optimization of a (variational) score that is bounded from above and expresses the quality of the found approximation. A key difference is the ansatz: While VAMP yields approximations of the Koopman operator, KVAD estimates its adjoint[21], the PF operator. To this end, KVAD uses the transition density (1) and assumes that it can be represented as

$$\hat{p}_\tau(\mathbf{x}_t, \mathbf{x}_{t+\tau}) = \mathbf{f}(\mathbf{x}_t)^\top \mathbf{q}(\mathbf{x}_{t+\tau}), \tag{24}$$

where $\mathbf{q} = (q_1, \ldots, q_m)^\top$ are m density basis functions and $\mathbf{f}$ are, as in (10), feature functions of the system's state. This leads to the linear model (10) with $\mathbf{f} = \mathbf{g}$ and

$$K = \int \mathbf{q}(\mathbf{y})\mathbf{f}(\mathbf{y})^\top \mathrm{d}\mathbf{y}.$$

It has been shown [63] that $\mathbf{q}$ can be estimated directly from data in a nonparametric fashion, which means that all the model's parameters reside inside the definition of $\mathbf{f}$. With the help of estimated $\mathbf{f}$ and $\mathbf{q}$, also the transition matrix K can be constructed. This kind of ansatz—sans the modified codomain in (23)—is similar to what was used in [17].

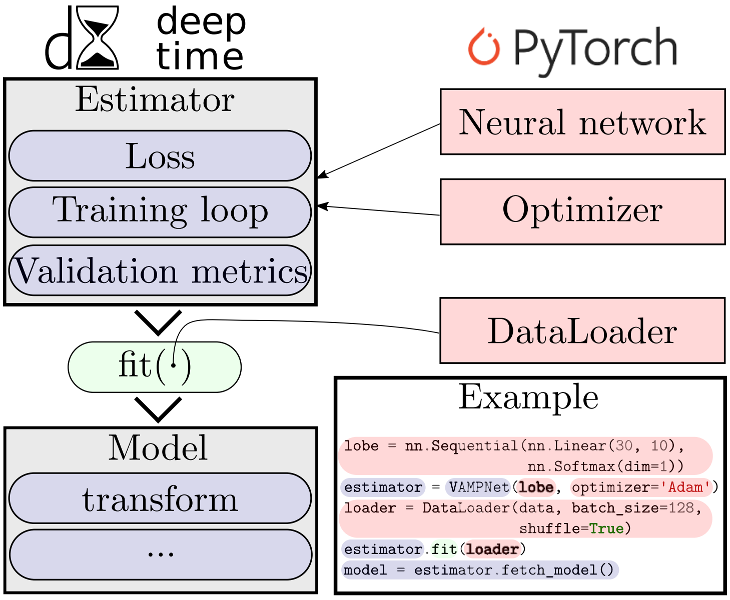
3.4. Deep dimension reduction and decomposition

In addition to the conventional learning methods introduced in section 3.3, deeptime also offers several deep learning methods for dimension reduction.

The deep learning components require PyTorch; however, PyTorch-dependent parts of the library are separate—i.e. a working installation of PyTorch is not required for the rest of the library. The estimators providing deep dimension reduction can be found in the `deeptime.decomposition.deep` subpackage.

Deep learning requires some additional flexibility and data handling compared to conventional learning. In particular, the user first must define a neural network architecture and an optimizer for adjusting the network's weights. There are some predefined architectures directly available in deeptime (such as multilayer perceptrons (MLPs)), but in principle these as well as the optimizer are defined with PyTorch (see figure 3(b)). Once defined, one can construct a deep estimator that contains losses, validation metrics, and training procedures (see figure 3(a)). Fitting deep learning components typically involves shuffling and

[21] Adjoint in the sense of equation (5).



```
lobe = nn.Sequential(nn.Linear(30, 10),
                     nn.Softmax(dim=1))
estimator = VAMPNet(lobe, optimizer='Adam')
loader = DataLoader(data, batch_size=128,
                    shuffle=True)
estimator.fit(loader)
model = estimator.fetch_model()
```

Figure 3. Schematic overview of the flexible integration of deep learning components in deeptime. (a) Deeptime interface: some `Estimator` instances in deeptime, for example VAMPNets and Time-lagged Autoencoders, take neural networks as configurational input. These estimators offer a `fit( )` method which takes PyTorch data loaders. The data loaders are configured by the users and manage, e.g. shuffling and batch sizes. After a `Model` has been fit, it can be used for further analysis of the input data. Models containing deep learning components are designed so that they can also seamlessly work with NumPy arrays and interact with the rest of the deeptime library. (b) PyTorch interface: neural networks can be defined using PyTorch and also be trained using optimizers which either are already included in PyTorch or at the very least are PyTorch compatible. Instances thereof can be used with certain estimators in deeptime. (c) Example: a typical workflow for a deep learning estimator, color-coded according to which library the classes and respective instances belong.

dividing the data into batches. Since the optimal batch size and also the shuffling method are problem-dependent, these choices must be made by the user. PyTorch offers `DataLoaders` for this exact purpose. Therefore `estimator.fit` is performed on a data loader instance rather than arrays (see figure 3(a)). Finally, deep learning estimators also produce models which encapsulate among other things a copy of the trained neural network. While PyTorch neural networks operate on `torch.Tensor` instances, deeptime models with deep learning components are designed so that they can also work with ordinary NumPy arrays, ensuring a seamless integration with other and in particular conventional models and methods. For more details about PyTorch, see the official documentation[22].

3.4.1. VAMPNets

VAMPNets [16] are a deep learning approach that seek to find parametrizations of neural networks χ_0 and χ_1 (referred to as 'lobes') so that the VAMP-E or one of the VAMP-r scores (19) under these transformations is maximized, leading to smaller model errors (recall paragraph about VAMP in section 3.3). This is possible because there is a variational upper bound to the scores and their computation is differentiable—therefore any of the VAMP scores can be used directly as an objective function in a deep learning context.

Other deep learning methods which are not currently included in deeptime but also approximate the Koopman operator are, e.g. those found in [15, 19, 91, 92].

3.4.2. KVADNets

Analogously to VAMPNets, KVADNets optimize the variational KVAD score to find an optimal parametrization of feature functions $\mathbf{f}$ (24). As in the case of VAMPNets, the KVAD score is differentiable [63].

3.4.3. Time-lagged (variational) autoencoders (T(V)AEs)

TAEs [14] are a type of neural network approach in which instantaneous data $\mathbf{x}_t \in \mathbb{R}^d$ is compressed/encoded through a parameterized function

$$E : \mathbb{R}^d \to \mathbb{R}^n, \mathbf{x}_t \mapsto E(\mathbf{x}_t) = \mathbf{z}_t$$

[22] https://pytorch.org/.

with $n \leqslant d$ and then reconstructed as time-lagged data $\mathbf{x}_{t+\tau}$, $\tau > 0$ via a decoder network

$$D : \mathbb{R}^n \to \mathbb{R}^d, \mathbf{z}_t \mapsto D(\mathbf{z}_t) \approx \mathbf{x}_{t+\tau}.$$

The optimization target is to reduce the mean-squared error between $\mathbf{x}_{t+\tau}$ and $(D \circ E)(\mathbf{x}_t)$, effectively training a latent and lower-dimensional representation $E(\mathbf{x}_t)$ of the process. In [14] it was shown that in the linear case TAEs perform TCCA, cf the paragraph about VAMP in section 3.3. An architecture that is akin to the one of TAEs was used in [93] to find collective variables in the context of molecular enhanced sampling.

A natural extension to TAEs is to exchange the neural network architecture of an autoencoder by the architecture of a variational autoencoder [94, 95], yielding the generative TVAE that can also be found in the deeptime library. In [20] these architectures (there called 'variational dynamics encoder') were used in conjunction with a loss term inspired by saliency maps [96] (known from computer vision) to produce interpretable dynamical models while still maintaining the high degree of nonlinearity that can be achieved by neural networks.

3.5. Numerical experiments

We compare some of the dimension reduction methods introduced in sections 3.3 and 3.4. The first example highlights differences in the approximation if used for dimension reduction in a time-homogeneous system. The second example uses data obtained from a time-inhomogeneous system with the objective to find coherent structures.

3.5.1. Dimension reduction

We consider a two-state HMM (see section 4) with transition matrix

$$P = \begin{pmatrix} 0.95 & 0.05 \\ 0.05 & 0.95 \end{pmatrix} \tag{25}$$

and anisotropic but linearly separable two-dimensional Gaussian emission distributions. In a subsequent step the data is transformed via

$$(x, y) \mapsto (x, y + \sqrt{|x|}). \tag{26}$$

This leads to two wedge-shaped output distributions which are no longer linearly separable (see figure 4). We simulate a trajectory of $T = 1000$ frames from this model and try to recover a separation into the two original hidden states using different dimension reduction methods by projecting onto the dominant slow process, which is the jump process between the two wedges.

Figures 4(a)–(c) and (g)–(i) shows the sampled and transformed time-series data $\mathbf{x}_t \in \mathbb{R}^2$ as a scatter plot. The estimated models yield two-dimensional decision landscapes $\chi : \mathbb{R}^2 \to \mathbb{R}$ which are shown in the background as a filled contour. Based on the decision landscape we obtain crisp assignments to one of the two states via k-means [97] clustering with $k = 2$ cluster centers in the projected space; these clusters determine the point colors in the scatter plot. In figures 4(a)–(c) and (g)–(i) we furthermore report the 10-fold cross-validated VAMP-2 score along with its standard deviation (see section 3.3 for the score and [98] for the cross-validation scheme), which enables a quantitative assessment of the quality of the projection χ.

Figures 4(d)–(f) and (j)–(l) shows the projection of the two-dimensional time series onto one dimension for the first 200 frames of the trajectory, where $\chi(x_t)$ is presented in transparent blue with corresponding crisp clustered assignments in opaque blue and the hidden reference state is presented in orange. We also report the assignment accuracy of the crisp state with respect to the hidden reference state over the entire dataset in the titles of subfigures (a)–(f).2. While the assignment accuracy can be used as another measure of the quality of the projection, it is only available if the ground truth is known. A VAMP score, on the other hand, can always be evaluated. Here, we chose the VAMP-2 score, because its maximization can be identified with the maximization of kinetic variance (see [99]). Maximizing kinetic variance achieves an optimal separation of metastable sets, which corresponds to the separation of the two wedges in this example.

In the limit of infinite data (i.e. when faithfully representing the original data distribution) and optimal featurization, the score should approach $s_{\text{lim}} = 1.81$. This can be found from using the ground truth hidden transition matrix (25) and applying it to the VAMP score assuming that the distribution of data is given by the stationary distribution[23].

Below, we discuss and further describe each of the panels in figure 4.

[23] In more detail, if $\boldsymbol{\mu} = (0.5, 0.5)^\top$ is the stationary distribution corresponding to (25), we assume that data distribution is given by the stationary distribution and set covariance matrices $C_{00} = C_{\tau\tau} = \text{diag}(\boldsymbol{\mu})$ and the cross-covariance matrix to $C_{0\tau} = P$ (in this example $\tau = 1$). From the covariance matrices we can obtain the Koopman matrix (cf section 3.3) which can be decomposed and used for scoring.

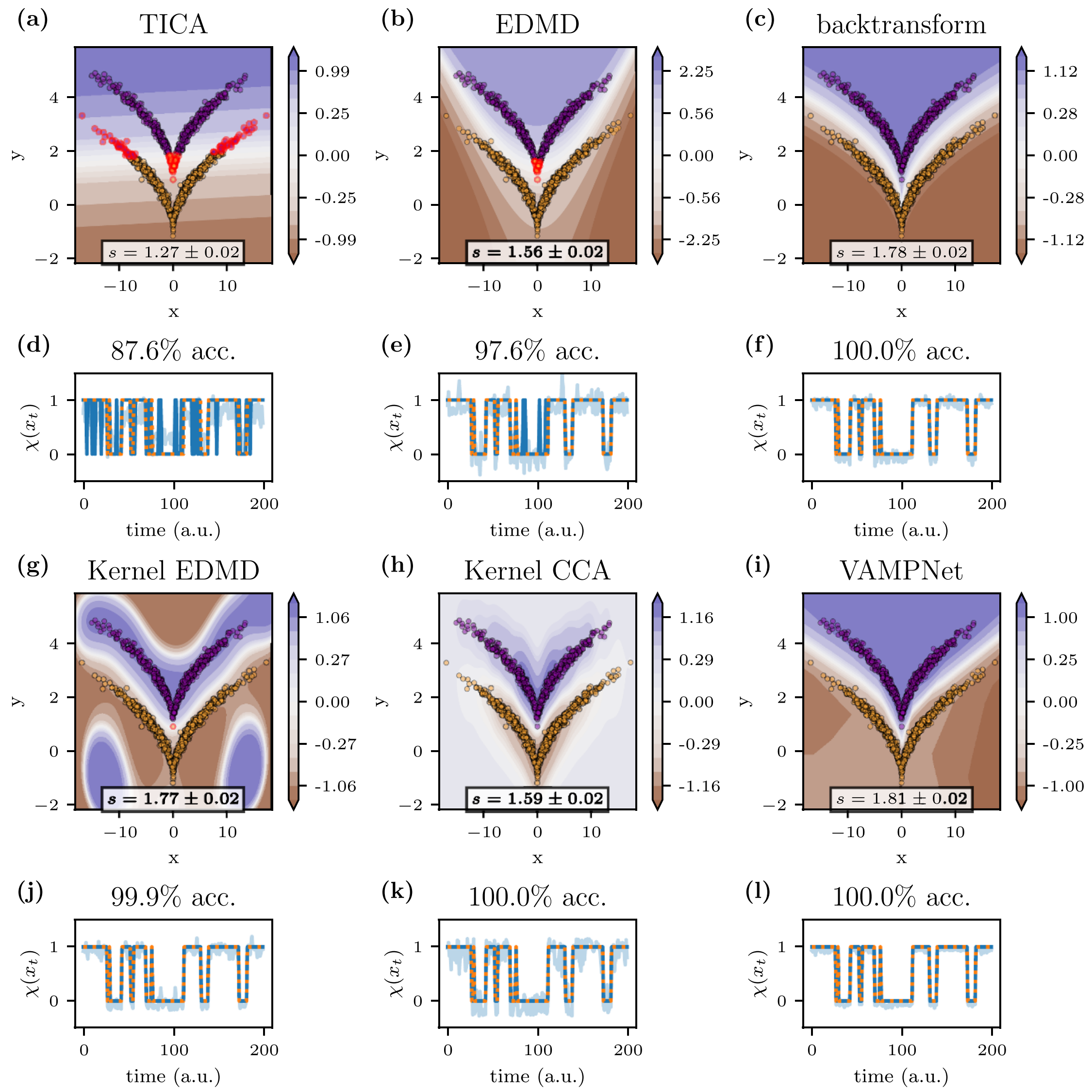


Figure 4. Comparison of different Koopman operator methods. The data is generated by a HMM with two hidden states and respective two-dimensional output probability distributions which are not linearly separable. The methods should approximately recover the underlying hidden process, where in (a)–(c), (g)–(i) the background contour is the decision landscape and the scatter colors denote sharp assignments obtained by a two-state clustering in the projection. Incorrectly assigned states have a red edge. The scores reported in the bottom parts of the plots are cross-validated VAMP-2 scores. Plots (d)–(f) and (j)–(l) show a projection of the two-dimensional time-series onto one dimension (transparent blue) with crisp assignments (opaque blue) and the ground truth (orange) as reference. The accuracy (acc.) refers to the amount of correctly assigned states after clustering.

(a) *TICA*. TICA is a linear method in that it can only draw linear decision boundaries and the dataset is deliberately not linearly separable. Therefore the tip of the upper wedge and the outer areas of the lower wedge are misclassified. This is also reflected in the comparably low VAMP-2 score and accuracy.

(b) *EDMD*. We choose EDMD with an ansatz basis of monomials up to degree two in two-dimensional space; i.e.

$$B = \left\{(x,y) \mapsto x^p y^q : p,q \in \mathbb{N}_{\geqslant 0},\ p+q \leqslant 2\right\}.$$

This leads to a decision landscape shaped like a rounded cone, able to separate most of the data into the two hidden states except for the tip of the upper wedge. Consequently, score and accuracy achieve a higher value than the one obtained from TICA.

(c) *Backtransform*. Here we use the hand-tailored transformation $(x,y) \mapsto (x, y - \sqrt{|x|})$, which makes the two states linearly separable again and apply VAMP. This featurization uses the ground truth as prior knowledge and therefore achieves perfect state separation. Consequently, the accuracy is at 100% and the VAMP-2 score reaches a high value. Due to finite data it does not quite reach the theoretical limit of $s_{\lim} = 1.81$.

(d) *Kernel EDMD*. We use kernel EDMD with a Gaussian kernel (20). The regularization parameter ε of the estimator as well as the bandwidth σ of the kernel are tuned to maximize the VAMP-2 score on a validation set using the SLSQP optimizer [100], yielding $\sigma \approx 1.42$ and $\varepsilon \approx 6.7 \times 10^{-4}$. The method finds a good separation between the two hidden states.

(e) *Kernel CCA*. As in the kernel EDMD case, we choose a Gaussian kernel (20) with regularization parameter and bandwidth tuned to maximize the VAMP-2 score on a validation set using the SLSQP optimizer [100]. This leads to $\sigma \approx 0.85$ and $\varepsilon \approx 0.36$. Compared to the other methods, the support of the estimated singular functions is smaller and in particular does not extend far beyond the area spanned by the sample data. This means that according to kernel CCA, there is large uncertainty as to which state a point in space belongs to as soon as it is outside the densely populated areas of the wedges. On the other hand, the score is lower compared to kernel EDMD or VAMPNets. This means that the metastable sets are separated less clearly, which can also be observed in the fuzziness of the transparent blue trajectory in figure 4(k) and therefore the slow dynamics of the system are not represented as well as they are represented with, e.g. kernel EDMD.

(f) *VAMPNets*. As an architecture for the lobe $\boldsymbol{\chi}$ we choose a MLP of depth $d = 5$ with a rectified linear unit nonlinearities and 15, 10, 10, 5, and 1 neurons, respectively. The network is trained using the Adam optimizer [101] with a learning rate of 10^{-3}. We obtain a decision landscape that resembles the one of the backtransform with a perfect state separation. Also the idealized VAMP-2 score s_{lim} based on the hidden transition matrix is within the standard deviation of the VAMPNet VAMP-2 score. The hyperparameters were chosen heuristically so that training was stable and yielded high scores.

For the optimization of the parameters of kernel EDMD we found it crucial to first whiten the data by removing the empirical mean $\boldsymbol{\mu}$ and transforming it into the PCA basis via

$$\mathbf{x}_t \mapsto C^{-\frac{1}{2}}(\mathbf{x}_t - \boldsymbol{\mu}), \tag{27}$$

where C denotes the covariance matrix over the trajectory. The other methods were numerically more stable and applicable directly to the raw data. Whether whitening is required does not only depend on the method but in particular also on the chosen ansatz.

3.5.2. Coherent set detection

Here, we illustrate how the introduced decomposition methods can be used to detect coherent sets; i.e. sets of particles which are geometrically consistent under a forward-backward dynamic and small perturbations [102, 103]. Following [103], one can quantitatively describe coherent sets $A \subset \Omega$ under the transfer operator $\mathcal{T}$ (see, e.g. (9)) as a set which is difficult to leave, i.e.

$$\left\langle \mathcal{T}^*\mathcal{T}\frac{1_A}{\mu_s(A)}, 1_A \right\rangle_{\mu_s} \approx 1, \tag{28}$$

the probability of staying within A under the forward-backward dynamic [103] should be close to 1. Here, $\mu_s(A)$ refers to the evaluation of the measure induced by the initial distribution.

While dominant eigenfunctions of methods assuming time-homogeneous dynamics (cf figure 1) can be related to metastable sets, methods that may also be applied to time-inhomogeneous dynamics yield coherent sets [59]. In particular, metastable sets can be understood as a special case of coherent sets.

Practically, these sets can be obtained by projecting Lagrangian data into the dominant (with respect to magnitude of singular values) left singular function space of an approximated PF or Koopman operator [67, 102], as these singular functions correspond to eigenfunctions of backward-forward dynamic $\mathcal{T}\mathcal{T}^*$ or forward-backward dynamic $\mathcal{T}^*\mathcal{T}$ [60, 67, 102] and therefore are an important ingredient for characterizing coherence (28). Spatial proximity in the singular function space indicates membership of the same coherent set.

As an application example we choose the Bickley jet, an idealized and periodically perturbed approximation of stratospheric flow which is described by a deterministic but non-autonomous system of ODEs [104, 105]. The ODEs act on particles $\mathbf{x} = (x, y) \in \Omega = [0, 20] \times [-4, 4]$ and are given by

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} -\dfrac{\partial \Psi}{\partial y} \\ \dfrac{\partial \Psi}{\partial x} \end{pmatrix} \tag{29}$$

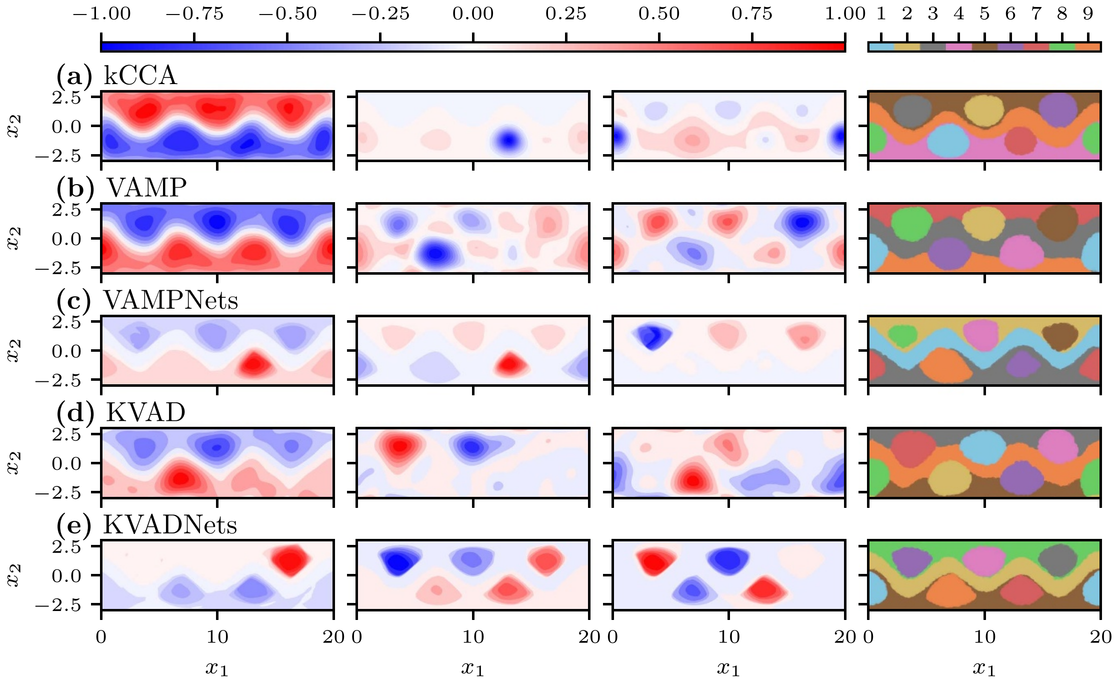


Figure 5. Comparison of different Koopman operator methods for coherent set detection. Columns 1–3 show the first, second, and third dominant singular function of the respective estimated Koopman operators. Column 4 shows a k-means clustering with $k = 9$ on the initial data after it has been transformed by the first nine respective singular functions. (a) Kernel CCA with a Gaussian kernel with bandwidth and regularization parameter optimized to maximize the VAMP-2 score. (b) VAMP-estimated model, where the ansatz featurization consists of a set of randomly shifted and distorted Gaussian functions $f(x) = \exp(-x^2)$. (c) VAMPNets with MLP lobes. (d) KVAD with the same ansatz as VAMP and a Gaussian kernel with bandwidth $\sigma = 1$. (e) KVADNets with a Gaussian kernel with bandwidth $\sigma = 0.5$ and a feature transformation given by MLPs.

with stream function

$$\begin{aligned}\Psi(x,y,t) &= c_3 y - U_0 L \tanh(y/L) + A_3 U_0 L \mathrm{sech}^2(y/L)\cos(k_1 x)\\ &\quad + A_2 U_0 L \mathrm{sech}^2(y/L)\cos(k_2 x - \sigma_2 t)\\ &\quad + A_1 U_0 L \mathrm{sech}^2(y/L)\cos(k_1 x - \sigma_1 t)\end{aligned}$$

and parameters chosen as in [103]. The domain Ω is quasi-periodic in x-direction. The Bickley jet is widely used as a benchmark problem in the coherent set literature, e.g. in [67, 102, 103, 106–108].

We expect to find a separation into nine coherent sets, where the domain Ω is separated into an upper Ω_{up} and lower Ω_{low} part with three circular coherent sets each, a coherent layer that is between Ω_{up} and Ω_{down} and the remainder of Ω_{up} and Ω_{low} sans the circular coherent sets, as illustrated in any of the panels of figure 5 column 4.

Because the ODE is not autonomous (meaning $\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x})$ depends on time t), we restrict ourselves to methods that support time-inhomogeneous dynamics, in particular kernel CCA, VAMP, VAMPNets, KVAD, and KVADNets. In order to fit the respective Koopman models, we first integrate $N = 3000$ particles whose positions are drawn uniformly in Ω from $t_0 = 0$ to $t_1 = 40$. From the resulting trajectories we use the initial time particle position matrix $X \in \Omega^N$ and final time particle position matrix $Y \in \Omega^N$ to find an embedding with corresponding Koopman or PF operator that describes transport from $\mathbf{x}_i$ to $\mathbf{y}_i$.

The visualization of the first three estimated dominant singular functions already reveals some of the coherent structure of the underlying process (see figure 5 columns 1–3). All of the methods yield similar results and, with different degrees of sharpness, each show the three vortices in the upper part and lower part of the domain.

To obtain crisp assignments to for a predefined number of coherent sets we perform k-means clustering using kmeans++ initialization with $k = 9$ cluster centers with one cluster center belonging to exactly one coherent set. The clustering is repeated 500 times and we select the cluster centers which yield the smallest cumulative squared distance between sample points and assigned cluster center (sometimes referred to as ‘inertia’). In the last column of figure 5, the particle positions at $t = 0$ are color-coded according to their cluster membership.

For both VAMP and KVAD we set up the feature functions in the following way: Weight matrices $W_1 \in \mathbb{R}^{100\times 3}$, $W_2 \in \mathbb{R}^{50,100}$ are generated by drawing i.i.d. samples from the normal distribution $\mathcal{N}(0,1)$ and bias vectors $b_1 \in \mathbb{R}^{100}$, $b_1 \in \mathbb{R}^{50}$ are generated by drawing i.i.d. samples from the uniform distribution $\mathcal{U}(-1,1)$. The vector-valued feature function is then given by

$$F : \mathbb{R}^2 \to \mathbb{R}^{50}, \quad \mathbf{x} \mapsto W_2\, \sigma(W_1 T(\mathbf{x}) + b_1) + b_2,$$

where $\sigma(x) = \exp(-x^2)$ acts component-wise and T is a transformation that embeds the two-dimensional data into three dimensions by mapping it onto a cylinder

$$T : \mathbb{R}^2 \to \mathbb{R}^3, \quad \begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} \cos(2\pi x/20) \\ \sin(2\pi x/20) \\ y/3 \end{pmatrix}, \tag{30}$$

accounting for the quasi-periodicity of the domain Ω. KVAD is equipped with a Gaussian kernel with bandwidth $\sigma = 1$.

For VAMPNets, the instantaneous and time-lagged lobes are MLPs and are using shared weights. The two-dimensional data is first transformed into three dimensions to account for quasi-periodicity in x direction via (30) and subsequently transformed through a batch normalization layer. The MLPs possess layers with 256, 512, 128, 128, and 9 neurons, respectively, separated using ELU nonlinearities and dropout $(p = 50\%)$.

In the case of KVADNets there is per definition just one lobe. Its architecture is the same as for VAMPNets.

All models project onto the dominant nine singular functions.

(a) *Kernel CCA*. We use a Gaussian kernel where the bandwidth and regularization parameter maximize the VAMP-2 score ($\sigma \approx 0.58$, $\varepsilon \approx 5.6 \times 10^{-3}$). Optimization was performed with SLSQP [100]. The first singular function shows a clear separation between upper and lower part of the domain as the dominant process. The vortices are circular in shape and can be observed in the evaluation of the singular functions as well as the clustering.
(b) *VAMP*. The results are qualitatively comparable to the ones of kernel CCA, however the clustering is less pronounced.
(c) *VAMPNets*. Some of the coherent structures are clearly visible in the first three singular functions. The clustering is pronounced; however, it yields vortices of varying sizes.
(d) *KVAD*. We use a Gaussian kernel with bandwidth $\sigma = 1$. The results are qualitatively comparable to VAMP.
(e) *KVADNets*. Here the vortices can easily be detected in the singular functions; however, the shape is less circular compared to the other methods. Furthermore, the first singular function has a less pronounced decision surface between upper and lower part of the domain; rather, it almost exclusively describes the exchange of mass between two individual vortices. The clustering, however, is sharp and is comparable to the other methods in terms of the detected sets.

While differences in estimated coherent sets can be evaluated qualitatively by visual inspection, we now seek to compare the methods in a quantitative fashion and try to determine a 'best' subdivision into coherent sets according to some criterium.

To this end, we define a 'coherence score'. Let $\mathbf{x}_t = \mathbf{\Phi}_{t_0,t}(\mathbf{x}_0)$ be the flow describing the solution of the governing equations given an initial position $\mathbf{x}_0$ at initial time t_0. Since in this example the ground truth dynamics are known, we can take our definition of a coherent set (28) as template. For a subdivision of Ω into disjoint coherent sets $\bigcup_i A_i = \Omega$, the score restricted to one set A_i is defined as

$$s_{\text{coh}}^{(i)} := \mathbb{P}\left[(\mathbf{\Phi}_{t_0,t_1}^{-1} \circ \mathbf{N}_\sigma \circ \mathbf{\Phi}_{t_0,t_1})(\mathbf{x}_{t_0}) \in A_i \mid \mathbf{x}_{t_0} \in A_i \right], \tag{31}$$

where $\mathbf{N}_\sigma(\mathbf{x}) := \mathbf{x} + \sigma\boldsymbol{\eta}$ distorts the forward-mapped $\mathbf{x}_0$ by white noise $\boldsymbol{\eta} = (\eta_1, \ldots, \eta_d)^\top$, $\eta_i \sim \mathcal{N}(0,1)$ with standard deviation σ. In this example, we chose $\sigma = 10^{-1}$. In other words, equation (31) describes the probability of a particle staying inside set $A_i \subset \Omega$ under propagation forward in time, addition of noise, and subsequent back-propagation to its initial time. This concept is illustrated in figure 6(a). Following the arrows in the figure, it depicts a subset A_i at $t_0 = 0$ in blue and red, the remainder is colored in light gray. The particles are then propagated according to the flow $\mathbf{\Phi}_{t_0,t_1}$ to the final time $t_1 = 40$ (upper right panel). The map N_σ is applied, yielding slightly different particle positions at t_1 (lower right panel). Subsequently, the distorted particles are mapped back to $t_0 = 0$. As one might expect, the particles leaving A_i aggregate at the

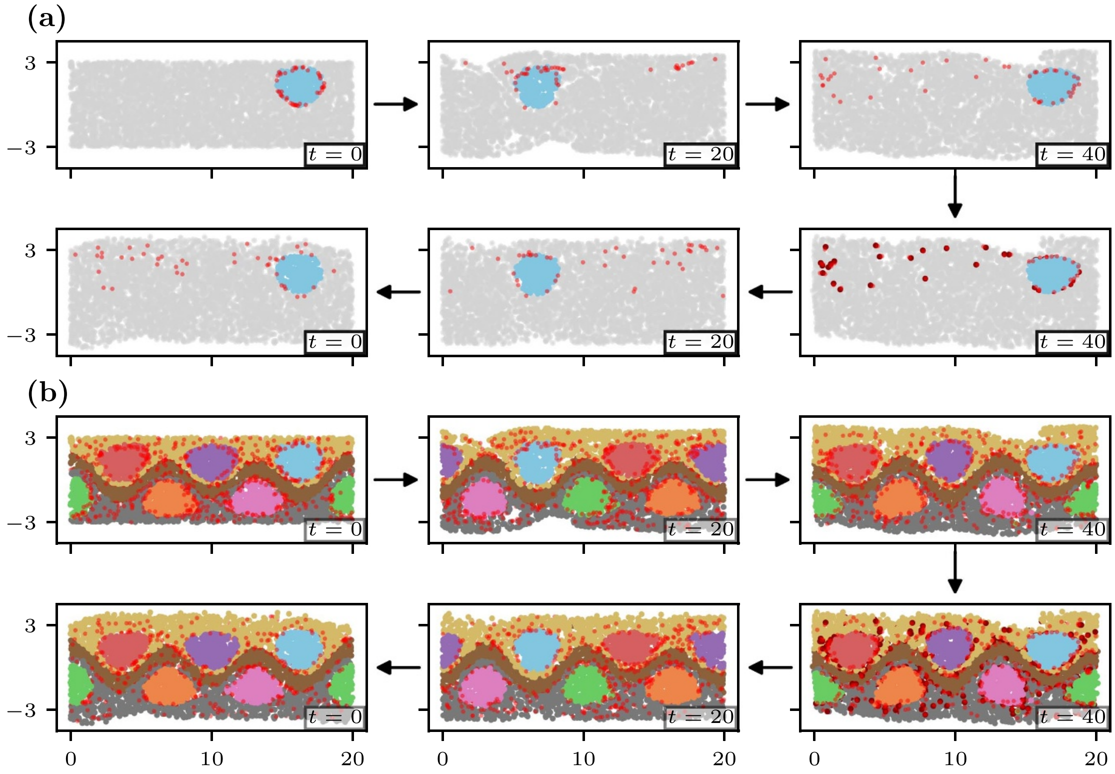


Figure 6. Coherence in the Bickley jet. This figure demonstrates the concept of 'leaked' particles used for defining a coherence score. Particles are colored according to their coherent set membership or marked as leaked (red dots). (a) Particles that stay assigned to a particular coherent set (blue dots) and particles that change their set membership under the forward-backward transformation with noise (red dots) as detected by KVADNets. The top row depicts application of forward dynamics, bottom row application of noise and backward dynamics, respectively. Particles initially not belonging to the considered coherent set are shown in grey. (b) Generalization of (a) depicting all coherent sets; red dots now depict particles leaking from *any* coherent set.

set's boundary. The figure uses the subdivision of the domain that is yielded by the KVADNets trained transfer operator (see figure 5(e)).

Finally, figure 6(b) shows the forward-backward mapping for each of the coherent sets. For clarity, we do not distinguish leaked particles from the respective sets but only show them as generally leaked from any of the sets. It can be observed that most of the interior area of the detected vortices remains vacant of leaked particles.

In order to arrive at one value for all estimated coherent sets, we consider the expectation

$$s_{\mathrm{coh}} := \mathbb{E}_{\mu_{t_0}}\left[s_{\mathrm{coh}}^{(i)}\right] = \sum_i \frac{\mu_{t_0}(A_i)}{\mu_{t_0}(\Omega)} s_{\mathrm{coh}}^{(i)}. \tag{32}$$

For practical evaluation of the score, we estimate a MSM without a reversibility constraint (see section 4) on $n = 2500$ discrete trajectories with nine states corresponding to the coherent sets, each trajectory corresponding to one individual particle and containing exactly two entries; namely, the coherent set before and after application of the forward-backward dynamic as given in (31). Then, the ith diagonal entry of the transition matrix is exactly the coherence score (31) corresponding to the ith coherent set A_i. The distributions $\mu_{t_0}(A_i)$ and $\mu_{t_0}(\Omega)$ are the empirical distributions; i.e. the number of particles initially inside set A_i and the total number of particles N, respectively.

A drawback of this score is that it does not indicate how faithfully the discovered coherent sets represent the system's dynamics. If, for example, the entire domain is its own coherent set, the score is maximized. Therefore, the score (32) should be considered in conjunction with other indicators such as the VAMP or KVAD score.

We compare these metrics in table 1. For all used methods of this example it shows the coherence score, the VAMP-2 score, and the KVAD score calculated with a Gaussian kernel with bandwidth $\sigma = 0.5$.

The table also reports standard deviations, which are obtained by repeating the scoring for fifteen rounds of $n = 2500$ independently and over the domain uniformly sampled initial particle positions. According to the coherence score, KVADNets deliver the best subdivision into coherent sets, with kernel CCA as a close second. This is also reflected in their respective VAMP-2 and KVAD scores. In contrast to KVAD, which yields

Table 1. Table showing different kinds of scores with standard deviations for different methods used for coherent set detection using the Bickley jet example (section 3.5.2). For the evaluation of the KVAD score, a Gaussian kernel with bandwidth $\sigma = 0.5$ was chosen. The methods are in ascending order from left to right according to their coherence score.

	KVAD	VAMP	VAMPNets	Kernel CCA	KVADNets
Coherence score	0.74 ± 0.01	0.77 ± 0.01	0.79 ± 0.01	0.85 ± 0.01	0.87 ± 0.01
VAMP-2 score	4.63 ± 0.06	5.18 ± 0.08	7.28 ± 0.06	5.77 ± 0.08	6.03 ± 0.09
KVAD score	$0.070 \pm 1.2 \times 10^{-3}$	$0.073 \pm 1.1 \times 10^{-3}$	$0.078 \pm 1.1 \times 10^{-3}$	$0.080 \pm 1.4 \times 10^{-3}$	$0.087 \pm 1.2 \times 10^{-3}$

the same sequence of methods (if ordered ascendingly) as the coherence score, the VAMP-2 score for VAMPNets is an outlier. The VAMP-2 score for VAMPNets is significantly higher compared to any of the other methods. The Bickley jet is a deterministic system and therefore the Koopman operator associated to it is not Hilbert–Schmidt—a violation of the assumptions that are made to define the VAMP scores. Up to noise effects caused by numerical integration, this might be the cause of the high score of VAMPNets.

It should be noted that the coherence score can still be approximated if the ground truth is not known or too expensive to compute by using a propagation model of the form (10). Assuming a good representation of the slow dynamics (which is indicated by a high VAMP or KVAD score), the error of integrating backwards in time with (10) is small.

4. Markov state models

MSMs are stochastic models describing the time evolution of a random process $\{\mathbf{x}_t\}_{t\geqslant 0}$, $\mathbf{x}_t \in \Omega$ (see [21–29]) and describe Markov chains with memory depth of 1. In other words, given a sequence $(\ldots, \mathbf{x}_{t-2\tau}, \mathbf{x}_{t-\tau}, \mathbf{x}_t)$ with a set of possible states Ω, the conditional probability of encountering a particular state $\mathbf{x}_{t+\tau} \in \Omega$ is only conditional on $\mathbf{x}_t \in S$; i.e. $\mathbb{P}(\mathbf{x}_{t+\tau}|\mathbf{x}_t, \mathbf{x}_{t-\tau}, \mathbf{x}_{t-2\tau}, \ldots) = \mathbb{P}(\mathbf{x}_{t+\tau}|\mathbf{x}_t)$. In contrast to the methods presented in section 3, we assume that we have a finite number of discrete states. Therefore we consider

$$S := \{1, \ldots, n\} \cong \Omega \tag{33}$$

as state space for the remainder of this section. Often we are presented with data that does not live in a countable or even finite state space. In these cases, the state space is tessellated using a finite amount of indicator functions. Typically, the tesselation is a Voronoi decomposition.

As shown at the example of the Prinz potential (figure 7(a)), the fineness of the chosen discretization affects MSM approximation quality [27]. The estimated transition matrix can approximate the dynamics with higher spatial resolution in a finer discretized space (figure 7(b)). Furthermore, the discretization has implications on the estimated eigenfunctions and, in particular, the estimated stationary distribution (figure 7(c)). Evaluating the eigenvalues for a given discretization yields a comprehensive picture of the model's quality (figure 7(d)) as the true eigenvalues present an upper bound to the estimated ones (variational principle [60]): the sum of eigenvalues reflects the VAMP-1 score.

MSMs fit into the framework of transfer operators as introduced in section 3 (see figure 1). In particular, an indicator function ansatz used with VAC and/or EDMD yields an MSM. When indicator functions are used with VAMP, one obtains GMSMs, which are capable of representing time-inhomogeneous dynamics. We suggest [27–29] for thorough reviews.

The conditional probabilities in the MSM framework are described by a transition matrix $P \in \mathbb{R}^{n\times n}$, where $n = |S|$ is the number of states. The transition matrix is given by

$$P_{ij} = \mathbb{P}(x_{t+\tau} = j | x_t = i) \qquad \forall t \geqslant 0, \tag{34}$$

i.e. the time-stationary probability of transitioning from state $i \in S$ to state $j \in S$ within time τ. This also means that P is a row-stochastic matrix. Note that the MSM transition matrix is a special case of a transfer operator approximation (see section 3), where the ansatz consists of indicator functions.

Dynamical quantities of interest can be computed from an MSM's transition matrix, e.g. mean first passage times and fluxes among (sets of) states [109], implied timescales [27], or metastable decompositions of Markov states [110].

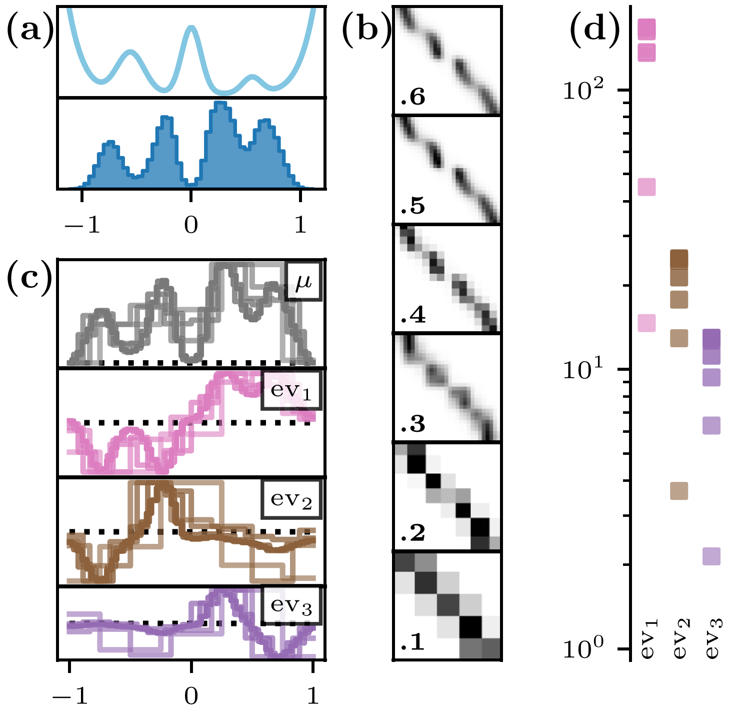


Figure 7. MSM spectral decomposition for a random walk in a asymmetric 1D 4-well potential; the corresponding potential function (upper part) with histogram of simulated data (lower part) is depicted in (a). (b) Shows transition matrix estimates for various discretizations, from very coarse (b).1 to very fine (b).6. The discrete states are sorted in ascending order with respect to their corresponding x-coordinate. (c) Depicts the four dominant left eigenvectors; discretizations are color coded from faint (coarse) to strong colors (fine discretization). The eigenvalues corresponding to the nonstationary eigenvectors are depicted in (d) in the same color.

4.1. MSM estimation with deeptime

The goal of the `deeptime.markov` module is to provide tools to estimate and analyze MSMs from discrete-state time-series data. If the data's domain is not discrete, classical discretization algorithms (such as the ones implemented in `deeptime.clustering`) can be employed to assign each frame to a state.

In what follows, we introduce the core object, the `MarkovStateModel`, as well as a variety of estimators. An overview of the main models contained in the `markov` module is depicted in figure 8.

Deeptime implements maximum-likelihood estimators for MSMs as well as Bayesian sampling routines [111], leading to `MarkovStateModel` and `BayesianPosterior` model instances, respectively. An integral component of MSM estimation and sampling based on time-series data is collecting statistics over the encountered state transitions (transition counting), which leads to a `TransitionCountModel`.

Bayesian sampling of MSMs leads to a `BayesianPosterior` that consists out of one `MarkovStateModel` instance representing the prior as well as the sampled `MarkovStateModels` instances (see figure 8(a)). Each `MarkovStateModel` possesses a transition matrix (34) and, if available, statistical information about the data in the form of a transition count matrix. Furthermore, deeptime provides augmented Markov models (AMMs) [112] which can be used when experimental data is available, as well as OOMs [32]. OOMs are unbiased estimators for the MSM transition matrix that correct for the effect of being presented with out of equilibrium data even when short lag-times are used. Both AMMs and OOMs inherit from the `MarkovStateModel` class definition.

While MSMs are a special case of the transfer operator model (cf section 3.3 and figures 1 and 2), they can be converted to `CovarianceKoopmanModels` of two different types. In one case, one can define the Koopman operator solely based on the transition matrix and corresponding stationary distribution; i.e. without respect to any statistical information. In the other case, when statistical information is present in the form of a `TransitionCountModel`, the statistics over transition counts may be used to estimate an empirical distribution according to which the Koopman operator is defined. The choice of Koopman model is up to the user; therefore, in deeptime MSMs do not inherit from `CovarianceKoopmanModel` but rather offer properties yielding respective instances of `CovarianceKoopmanModel`.

When estimating MSMs from data, deeptime assumes that the data is in the form of $k \geqslant 1$ trajectories $T_1, T_2, \ldots, T_k$ which comprise sequences of discrete states, i.e.

$$T_i = (s_1, s_2, \ldots, s_{n_i}),\ \forall j = 1, \ldots, n_i : s_j \in S, \tag{35}$$

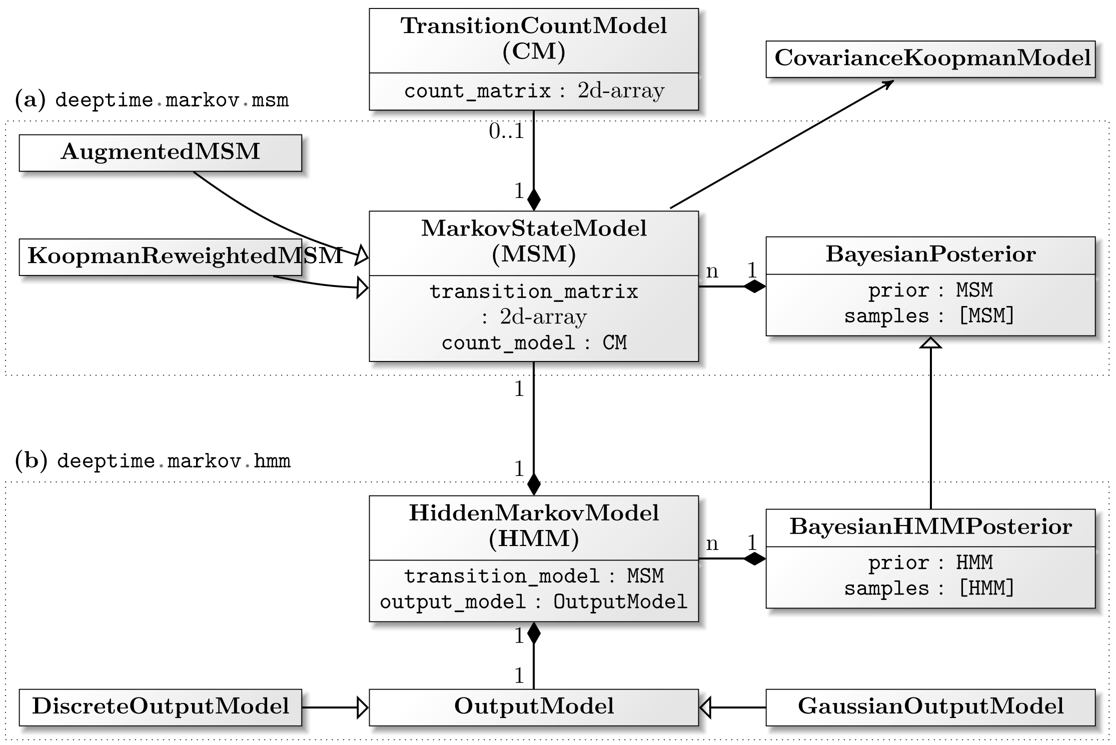


Figure 8. Class diagram showing relationships of main models contained in the `markov` package. We show two of the sub-packages of the `markov` module. Classes may be related via inheritance (—▷), composition (◆), or produce objects of another kind (➔). In case of composition, we further denote the cardinality of the relationship next to the arrow. (a) `msm`. The `markov.msm` package has the MSM at its core. It is completely determined by a transition matrix, but may also contain information about the statistics of data, in which case it possesses a `TransitionCountModel`. Furthermore there are `AugmentedMSM` and `KoopmanReweightedMSM` subclasses. An MSM is also a Koopman model using indicator basis functions; therefore, it can generate corresponding `CovarianceKoopmanModel` objects (see section 3 and figure 2). Bayesian sampling around an MSM leads to `BayesianPosteriors`, consisting of the prior and drawn samples. (b) `hmm`. This package contains estimators and models corresponding to HMM estimation. A `HiddenMarkovModel` consists out of a `transition_model` which describes evolution of a hidden state and an `OutputModel` which assigns a distribution over observable states to each hidden state. The discrete output model assigns each hidden state a discrete probability distribution over states, the Gaussian output model samples from one-dimensional Gaussians with means and variances conditioned on the hidden state. Same as MSMs, Bayesian sampling is available for HMMs, leading to a `BayesianHMMPosterior`.

where n_i is the length of the ith trajectory and $S = \{0, 1, \ldots, N_S - 1\}$ is the set of discrete states. In terms of further analysis it can be desirable to restrict the discrete state space onto a subset of $S' \subset S$, e.g. when certain state transitions are not populated and/or to select an ergodic subset. This task is best performed using a `TransitionCountModel` instance prior to estimating an MSM, as it possesses methods to produce new instances of the transition count model but restricted onto $S^{'}$.

4.2. Hidden Markov models

In many applications, the observed processes are only approximately Markovian in discrete state space; i.e. MSMs are only approximately valid [27]. The Markovianity assumption for the observed dynamics is discarded for HMMs which assume that the modeled stochastic process is hidden (not directly observable). Therefore, the central object of the HMM is the transition matrix $\tilde{P}$ among hidden states $s_i \in S$. The transition matrix $\tilde{P}$ can be estimated from the time series of observable states O with the Baum–Welch algorithm [30, 113–115]. Briefly: alongside the transition matrix $\tilde{P}$, for each hidden state $s_t \in S$ the algorithm estimates an emission probability for a given observable state $\mathbf{o}_t \in O$. HMMs therefore provide a (time-dependent) mapping between observable and hidden states along with the transition matrix $\tilde{P}$ [31]. This further allows us to estimate a maximum likelihood pathway of the trajectories in the hidden state space (Viterbi algorithm [116]).

Because the Baum–Welch algorithm converges to a local likelihood maximum [31], it is crucial to provide a reasonable initial guess of the emission probabilities and initial state distribution. Deeptime offers multiple possibilities to initialize the HMM estimation procedure (contained in the `deeptime.markov.hmm.init` package), with a fallback option to a classical MSM or MSM-derived (e.g. PCCA [110]) estimate of the metastable dynamics.

The initial guess is an object of type `HiddenMarkovModel` (see figure 8(b)). HMMs are composed of an MSM which describes the hidden state transitions and an output model. The output model is responsible for mapping a hidden state s_t to an observable state $\mathbf{o}_t = \mathbf{o}(s_t) \in O$. Deeptime offers `DiscreteOutputModels` which map each hidden state to a sample of a discrete probability distribution over observable states as well as `GaussianOutputModels` which map a hidden state to a sample of a one-dimensional Gaussian distribution with mean and variance depending on the hidden state.

As with MSMs, HMMs in deeptime also support Bayesian sampling following a Gibbs sampling scheme detailed in [55]. This produces a `BayesianHMMPosterior` which inherits from the `BayesianPosterior`, (cf figure 8), which allows samples of quantities of interest which can be derived from an HMM instance to be collected.

5. Sparse identification of nonlinear dynamics

The SINDy algorithm [33] is a data-driven method for discovering nonlinear dynamical systems models from measurement data using sparse regression. The method also fits into the Koopman operator framework presented in section 3 since it is related to an approximation of the Koopman generator, defined by $\mathcal{L}f := \lim_{\tau\to 0}(\mathcal{K}_\tau f - f)/\tau$, see [117] for details. The goal of SINDy is to approximate a nonlinear dynamical system

$$\frac{d}{dt}\mathbf{x} = \mathbf{f}(\mathbf{x}) \tag{36}$$

as a sparse linear combination of candidate functions $\theta_k(\mathbf{x})$:

$$\frac{d}{dt}x_j \approx \sum_{j=1}^{\ell} \xi_{jk}\theta_k(\mathbf{x}) \quad \Longrightarrow \quad \frac{d}{dt}\mathbf{x} \approx \Xi\boldsymbol{\Theta}(\mathbf{x}). \tag{37}$$

The matrix Ξ is assumed to be sparse, with the nonzero elements determining which terms in the library $\boldsymbol{\Theta}$ are active in the dynamics. In practice, the library $\boldsymbol{\Theta}$ is defined either to contain a generic set of terms, such as monomials, or terms guided by partial knowledge of the physical system. For example, metabolic regulatory networks often include rational function nonlinearities [118]. However, monomomials often suffice, either because the governing physics is polynomial (e.g. the Navier–Stokes equations for fluid dynamics), or because polynomials provide a reasonable Taylor expansion of the dynamics into a normal form [119].

The sparse matrix Ξ is typically identified via sparse regression based on a time-series of data $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m$ collected at times $t_1, t_2, \ldots, t_m$. This data is organized into a matrix $X \in \mathbb{R}^{n\times m}$, and a matrix of derivatives $\dot{X}$ is formed either by measuring the derivatives directly or numerically approximating them from the data in X. The library $\boldsymbol{\Theta}$ may now be evaluated on the data matrix X, resulting in the following matrix system of equations

$$\dot{X} \approx \Xi\boldsymbol{\Theta}(X). \tag{38}$$

The matrix Ξ is then solved for in the following optimization

$$\mathrm{argmin}_{\Xi}\|\dot{X} - \Xi\boldsymbol{\Theta}(X)\|_F + \lambda\|\Xi\|_0. \tag{39}$$

The first term measures the model error, while the $\|\cdot\|_0$ term counts the number of nonzero elements in Ξ, promoting sparsity. This zero norm is non-convex, and several relaxations are available that yield sparse solutions [33, 120].

There are several extensions to SINDy, e.g. incorporating the effect of actuation and control [121] and to enforce partially known physics, such as symmetries and conservation laws [122]. It is also possible to combine SINDy with deep autoencoders to identify a coordinate system in which the dynamics are approximately sparse [119]. Other extensions include the discovery partial differential equations [123, 124], the modeling of stochastic dynamics [125–127], and weak formulations of the problem [128–130], among others [124, 131–133]. SINDy has also been extended to accommodate tensor libraries, which dramatically increases its ability to handle systems with high state dimension [134]. This sparse modeling procedure has been applied to discover new physical models, for example in fluid dynamics [122, 135], including for turbulence closure modeling [136].

It is important to note that SINDy also applies equally well to discrete time systems

$$\mathbf{x}_{k+1} = \mathbf{F}(\mathbf{x}_k) \tag{40}$$

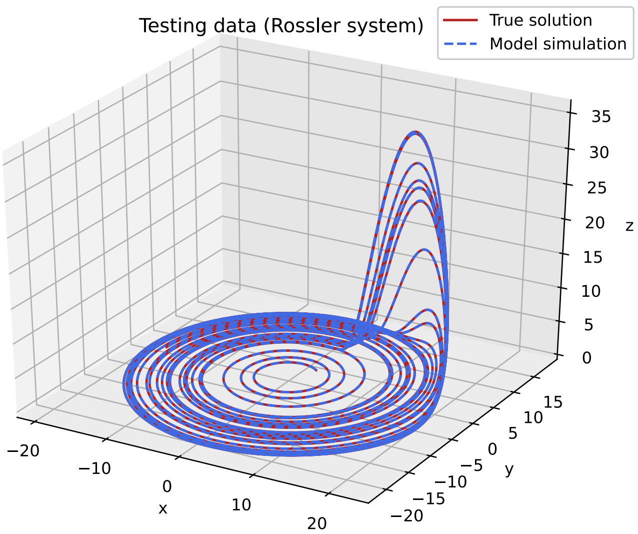


Figure 9. Reconstruction of the Rössler attractor using SINDy.

in which case derivatives need not be estimated. If SINDy is formulated in discrete time with no sparsity promoting term (i.e. $\lambda = 0$) and with a library $\boldsymbol{\Theta}(\mathbf{x}) = \mathbf{x}$, then the DMD approximation is recovered.

To demonstrate SINDy, we consider the Rössler attractor [137], a system of ODEs exhibiting chaotic behavior. Figure 9 shows the reconstruction of the dynamic attractor for the Rössler system of equations:

$$\begin{aligned}\dot{x}_1 &= -x_2 - x_3 \\ \dot{x}_2 &= x_1 + ax_2 \\ \dot{x}_3 &= b + x_3(x_1 - c)\end{aligned}$$

with constants $a = 0.1$, $b = 0.1$, and $c = 14$.

Deeptime has two SINDy objects. The `SINDy` estimator is used to solve the optimization problem (38) given $\boldsymbol{\Theta}$, X, and optionally $\dot{X}$. By default the sequentially-thresholded least-squares algorithm [33] is used to solve the optimization problem. If $\dot{X}$ is not user-provided, it is estimated from X with a first order finite difference method.

The estimator produces a `SINDyModel`, representing the learned dynamical system. The model can be used to predict derivatives given state variables, to simulate forward in time from novel initial conditions, and to score itself against ground truth data.

The implementation is API-compatible to the Python package PySINDy [46], which in particular enables users to make use of a wider range of optimizers defined in PySINDy.

6. Datasets

Deeptime offers a range of datasets to which its methods can be applied. The datasets and methods were purposefully designed to be non-domain-specific and to deliver data generators rather than fixed datasets. As a result, the repository as well as package size are remain small and generation parameters can be varied to study their effects on the algorithms. The data simulators are structured so that performance-critical parts are implemented in C++ and the generation procedure is not very time consuming.

In particular, a range of example SDEs of the form

$$\mathrm{d}\mathbf{x}_t = \mathbf{F}(t, \mathbf{x}_t)\mathrm{d}t + \sigma \mathrm{d}W_t,$$

where $\mathbf{F} : \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}^d$, W_t a d-dimensional Wiener process, and $\sigma \in \mathbb{R}^{d \times d}$, are implemented. All these SDEs are integrated using an Euler–Maruyama integrator. While the definition of these examples happens in C++, it is set up in such a way that also C++-inexperienced users can natively define their own.

For example, the definition of a double well system

$$\mathrm{d}\mathbf{x}_t = -\nabla V(\mathbf{x}_t)\mathrm{d}t + \sigma \mathrm{d}W_t, \quad V(\mathbf{x}) = (\mathbf{x}_1^2 - 1)^2 + \mathbf{x}_2^2,$$

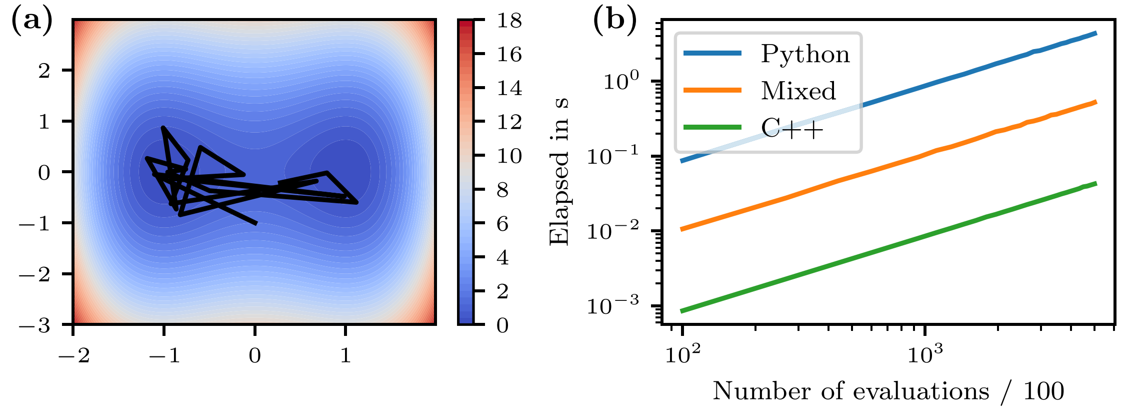


Figure 10. Two-dimensional double-well example system. We show a performance comparison between three different implementations of a two-dimensional double well system. (a) Potential landscape $V(\mathbf{x}) = V(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1^2 - 1)^2 + \mathbf{x}_2^2$ with example trajectory under diffusion matrix $\sigma = \mathrm{diag}(0.7, 0.7)$ and state snapshots taken every 10^4 steps. (b) Time elapsed over number of evaluations, while one evaluation corresponds to evolving the state by 100 steps under a integration step size of $h = 10^{-3}$. 'Python' refers to a native Python implementation, 'C++' refers to a native C++ implementation, and 'mixed' refers to a C++ implementation, where the diffusion matrix as well as the gradient of the potential are defined in Python.

with $\mathbf{x}_t \in \mathbb{R}^2$ and $\sigma = \mathrm{diag}(0.7, 0.7)$ can be achieved by a struct definition detailing the evaluation of the right-hand side. Many of the parameters of the system can be made available at compile-time, enabling further optimizations by the compiler. An example trajectory as well as a contour plot of the potential landscape can be found in figure 10(a). By making information such as the data type (e.g. `float` or `double`), the dimension of the state space, the integrator, and σ available at compile time, the compiler can perform further optimizations and potentially vectorizations that it otherwise could not, reducing the time it needs for evaluation.

Users also have the option to define the right-hand side $\mathbf{F}(\mathbf{x}_t)$ as well as the diffusion matrix σ in Python at some performance penalty (see figure 10(b)). Three different implementations are compared: one native C++ implementation, one implementation where just σ and the right-hand side are defined in Python, and one native Python implementation. One can see that roughly one order of magnitude in terms of evaluation performance is gained from native Python to a mixed Python/C++ implementation and from the mixed implementation to a native C++ implementation.

A drawback of making this information known at compile time is that for the mixed Python/C++ implementations, the dimension needs to be predefined; i.e. it must be explicitly exported when generating the Python bindings. On the other hand it improves performance and one can first prototype a system using the Python-defined diffusion matrix and right-hand side, and then eventually move the implementation to native C++ with relative ease.

7. Discussion and outlook

We have outlined the key components of deeptime's API and discussed the corresponding theory and methods as well as their relationships, in particular transfer operator based methods which can be used for dimension reduction, coherent set detection, analysis of kinetic quantities, and discovery of governing dynamics. These applications were each demonstrated with respective examples.

For future development we are actively looking for contributors and want to extend the currently available library of methods and datasets. For example there is a version of VAMPNets which allows the inclusion of experimental data. The SINDy module can be extended to include neural network based estimation of dynamics. Also the HMM module can be extended to support a richer set of output models. Furthermore the inclusion of more example datasets is desirable as this enables users to test and analyze existing or new methods and draw comparisons.

Finally we are planning to integrate time-series specific chunking and streaming capabilities so that methods which support online learning can more easily be used with data streams.

Data availability statement

The data that support the findings of this study are openly available at the following URL/DOI: https://github.com/deeptime-ml/deeptime.

Acknowledgments

We acknowledge financial support from Deutsche Forschungsgemeinschaft DFG (SFB/TRR 186, Project A12 and SFB 1114, Projects A04, B06, and C03), the European Commission (ERC CoG 772230 'ScaleCell'), and the Berlin Mathematics center MATH+ (AA1-6 and AA1-10). Part of this research was performed while M H, A M, B E H, S K, H W, N K, S L B, and F N were visiting the Institute for Pure and Applied Mathematics (IPAM), which is supported by the National Science Foundation (Grant No. DMS-1440415). F N acknowledges the German Ministry for Education and Research (Project BIFOLD—Berlin Institute for the Foundations of Learning and Data). S L B and N K acknowledge support from the National Science Foundation AI Institute in Dynamic Systems (Grant No. 2112085). B E H acknowledges the Lews-Sigler Institute at Princeton University, the Princeton Center for the Physics of Biological Function, and the Princeton Center for Theoretical Science. H W acknowledges the NSF of China (Grant No. 12171367), the Shanghai Municipal Science and Technology Commission (No. 20JC1413500), the Shanghai Municipal Science and Technology Major Project (Grant No. 2021SHZDZX0100) and the fundamental research funds for the central universities of China (Grant No. 22120210133).

ORCID iDs

Moritz Hoffmann https://orcid.org/0000-0002-9533-8586
Martin Scherer https://orcid.org/0000-0002-7983-4387
Tim Hempel https://orcid.org/0000-0002-0073-9353
Andreas Mardt https://orcid.org/0000-0002-7353-6063
Brian de Silva https://orcid.org/0000-0003-0944-900X
Brooke E Husic https://orcid.org/0000-0002-8020-3750
Stefan Klus https://orcid.org/0000-0002-9672-3806
Hao Wu https://orcid.org/0000-0002-2170-0618
Nathan Kutz https://orcid.org/0000-0002-6004-2275
Steven L Brunton https://orcid.org/0000-0002-6565-5118
Frank Noé https://orcid.org/0000-0003-4169-9324

References

[1] Buitinck L *et al* 2013 API design for machine learning software: experiences from the scikit-learn project *European conf ECML PKDD Workshop: Languages for Data Mining and Machine Learning* (*Prague, Czech Republic, 2013*) pp 108–22

[2] Pearson K 1901 LIII. On lines and planes of closest fit to systems of points in space *London, Edinburgh Dublin Phil. Mag. J. Sci.* **2** 559–72

[3] Hotelling H 1933 Analysis of a complex of statistical variables into principal components *J. Educ. Psychol.* **24** 417

[4] Molgedey L and Schuster H G 1994 Separation of a mixture of independent signals using time delayed correlations *Phys. Rev. Lett.* **72** 3634

[5] Naritomi Y and Fuchigami S 2011 Slow dynamics in protein fluctuations revealed by time-structure based independent component analysis: the case of domain motions *J. Chem. Phys.* **134** 02B617

[6] Schmid P J 2010 Dynamic mode decomposition of numerical and experimental data *J. Fluid Mech.* **656** 5–28

[7] Tu J H, Rowley C W, Luchtenburg D M, Brunton S L and Kutz J N 2014 On dynamic mode decomposition: theory and applications *J. Comput. Dyn.* **1** 391–421

[8] Koopman B O 1931 Hamiltonian systems and transformation in Hilbert space *Proc. Natl Acad. Sci. USA* **17** 315

[9] Gaspard P 1998 *Chaos, Scattering and Statistical Mechanics* Cambridge Nonlinear Science Series (Cambridge: Cambridge University Press) (https://doi.org/10.1017/cbo9780511628856)

[10] Klus S, Koltai P and Schütte C 2016 On the numerical approximation of the Perron–Frobenius and Koopman operator *J. Comput. Dyn.* **3** 51–79

[11] Klus S, Nüske F, Koltai P, Wu H, Kevrekidis I, Schütte C and Noé F 2018 Data-driven model reduction and transfer operator approximation *J. Nonlinear Sci.* **28** 985–1010

[12] Brunton S L, Budišić M, Kaiser E and Kutz J N 2021 Modern Koopman theory for dynamical systems (arXiv:2102.12086)

[13] Paszke A *et al* 2019 PyTorch: an imperative style, high-performance deep learning library *Advances in Neural Information Processing Systems 32* ed H Wallach, H Larochelle, A Beygelzimer, F d'Alché-Buc, E Fox and R Garnett (Curran Associates, Inc.) pp 8024–35

[14] Wehmeyer C and Noé F 2018 Time-lagged autoencoders: deep learning of slow collective variables for molecular kinetics *J. Chem. Phys.* **148** 241703

[15] Otto S E and Rowley C W 2019 Linearly-recurrent autoencoder networks for learning dynamics *SIAM J. Appl. Dyn. Syst.* **18** 558–93

[16] Mardt A, Pasquali L, Wu H and Noé F 2018 VAMPnets for deep learning of molecular kinetics *Nat. Commun.* **9** 1–11

[17] Wu H, Mardt A, Pasquali L and Noe F 2018 Deep generative Markov state models *Advances in Neural Information Processing Systems* vol 31 pp 3975–84

[18] Mardt A and Noé F 2021 Progress in deep Markov state modeling: coarse graining and experimental data restraints (arXiv:2108.01927)

[19] Lusch B, Kutz J N and Brunton S L 2018 Deep learning for universal linear embeddings of nonlinear dynamics *Nat. Commun.* **9** 4950

[20] Hernández C X, Wayment-Steele H K, Sultan M M, Husic B E and Pande V S 2018 Variational encoding of complex dynamics *Phys. Rev.* E **97** 062412

[21] Schütte C, Fischer A, Huisinga W and Deuflhard P 1999 A direct approach to conformational dynamics based on hybrid Monte Carlo *J. Comput. Phys.* **151** 146–68

[22] Swope W C, Pitera J W and Suits F 2004 Describing protein folding kinetics by molecular dynamics simulations. 1. Theory *J. Phys. Chem.* B **108** 6571–81

[23] Singhal N, Snow C D and Pande V S 2004 Using path sampling to build better Markovian state models: predicting the folding rate and mechanism of a tryptophan zipper beta hairpin *J. Chem. Phys.* **121** 415–25

[24] Noé F, Horenko I, Schütte C and Smith J C 2007 Hierarchical analysis of conformational dynamics in biomolecules: transition networks of metastable states *J. Chem. Phys.* **126** 155102

[25] Noé F 2008 Probability distributions of molecular observables computed from Markov models *J. Chem. Phys.* **128** 244103

[26] Noé F, Schütte C, Vanden-Eijnden E, Reich L and Weikl T R 2009 Constructing the equilibrium ensemble of folding pathways from short off-equilibrium simulations *Proc. Natl Acad. Sci.* **106** 19011–6

[27] Prinz J-H, Wu H, Sarich M, Keller B, Senne M, Held M, Chodera J D, Schütte C and Noé F 2011 Markov models of molecular kinetics: generation and validation *J. Chem. Phys.* **134** 174105

[28] Chodera J D and Noé F 2014 Markov state models of biomolecular conformational dynamics *Curr. Opin. Struct. Biol.* **25** 135–44

[29] Husic B E and Pande V S 2018 Markov state models: from an art to a science *J. Am. Chem. Soc.* **140** 2386–96

[30] Baum L E and Petrie T 1966 Statistical inference for probabilistic functions of finite state Markov chains *Ann. Math. Stat.* **37** 1554–63

[31] Rabiner L R 1989 A tutorial on hidden Markov models and selected applications in speech recognition *Proc. IEEE* **77** 257–86

[32] Nüske F, Wu H, Prinz J-H, Wehmeyer C, Clementi C and Noé F 2017 Markov state models from short non-equilibrium simulations—analysis and correction of estimation bias *J. Chem. Phys.* **146** 094104

[33] Brunton S L, Proctor J L and Kutz J N 2016 Discovering governing equations from data by sparse identification of nonlinear dynamical systems *Proc. Natl Acad. Sci.* **113** 3932–7

[34] Roe D R and Cheatham T E III 2013 PTRAJ and CPPTRAJ: software for processing and analysis of molecular dynamics trajectory data *J. Chem. Theory Comput.* **9** 3084–95

[35] Romo T D, Leioatts N and Grossfield A 2014 Lightweight object oriented structure analysis: tools for building tools to analyze molecular dynamics simulations *J. Comput. Chem.* **35** 2305–18

[36] McGibbon R T *et al* 2015 MDTraj: a modern open library for the analysis of molecular dynamics trajectories *Biophys. J.* **109** 1528–32

[37] Michaud-Agrawal N, Denning E J, Woolf T B and Beckstein O 2011 MDAnalysis: a toolkit for the analysis of molecular dynamics simulations *J. Comput. Chem.* **32** 2319–27

[38] Nguyen H, Roe D R, Swails J and Case D A 2016 Pytraj: Interactive Data Analysis for Molecular Dynamics Simulations (available at: http://amber-md.github.io/pytraj/)

[39] Gowers R J *et al* 2019 MDAnalysis: a python package for the rapid analysis of molecular dynamics simulations *Technical Report* (Los Alamos, NM: Los Alamos National Lab. (LANL))

[40] Beauchamp K A, Bowman G R, Lane T J, Maibaum L, Haque I S and Pande V S 2011 MSMBuilder2: modeling conformational dynamics on the picosecond to millisecond scale *J. Chem. Theory Comput.* **7** 3412–9

[41] Scherer M K, Trendelkamp-Schroer B, Paul F, Pérez-Hernández G, Hoffmann M, Plattner N, Wehmeyer C, Prinz J-H and Noé F 2015 PyEMMA 2: a software package for estimation, validation and analysis of Markov models *J. Chem. Theory Comput.* **11** 5525–42

[42] Wehmeyer C, Scherer M K, Hempel T, Husic B E, Olsson S and Noé F 2019 Introduction to Markov state modeling with the PyEMMA software [Article v1.0] *Living J. Comput. Mol. Sci.* **1** 5965

[43] De Sancho D and Aguirre A 2019 MasterMSM: a package for constructing master equation models of molecular dynamics *J. Chem. Inf. Model.* **59** 3625–9

[44] Demo N, Tezzele M and Rozza G 2018 PyDMD: python dynamic mode decomposition *J. Open Source Softw.* **3** 530

[45] Weiss R *et al* 2021 hmmlearn (Version 0.2.5) (available at: https://hmmlearn.readthedocs.io/) (Accessed 3 February 2021)

[46] de Silva B, Champion K, Quade M, Loiseau J-C, Kutz J and Brunton S 2020 PySINDy: a python package for the sparse identification of nonlinear dynamical systems from data *J. Open Source Softw.* **5** 2104

[47] Löning M, Bagnall A, Ganesh S, Kazakov V, Lines J and Király F J 2019 sktime: a unified interface for machine learning with time series (arXiv:1909.07872)

[48] conda-forge community 2015 The conda-forge project: community-based software distribution built on the conda package format and ecosystem *Zenodo* (available at: https://doi.org/10.5281/zenodo.4774216)

[49] Jakob W, Rhinelander J and Moldovan D 2017 pybind11—seamless operability between C++11 and python (available at: https://github.com/pybind/pybind11)

[50] Harris C R *et al* 2020 Array programming with NumPy *Nature* **585** 357–62

[51] Virtanen P *et al* 2020 SciPy 1.0: fundamental algorithms for scientific computing in python *Nat. Methods* **17** 261–72

[52] Hunter J D 2007 Matplotlib: a 2D graphics environment *Comput. Sci. Eng.* **9** 90–95

[53] Krekel H, Oliveira B, Pfannschmidt R, Bruynooghe F, Laugher B and Bruhin F 2004 pytest 6.2 (available at: https://github.com/pytest-dev/pytest)

[54] Kluyver T *et al* 2016 Jupyter notebooks—a publishing format for reproducible computational workflows *Positioning and Power in Academic Publishing: Players, Agents and Agendas* ed F Loizides and B Scmidt (Amsterdam: IOS Press) pp 87–90

[55] Chodera J D, Elms P, Noé F, Keller B, Kaiser C M, Ewall-Wice A, Marqusee S, Bustamante C, and Hinrichs N S 2011 Bayesian hidden Markov model analysis of single-molecule force spectroscopy: characterizing kinetics under measurement uncertainty (arXiv:1108.1430)

[56] Rowley C W, Mezic I, Bagheri S, Schlatter P and Henningson D S 2009 Spectral analysis of nonlinear flows *J. Fluid Mech.* **645** 115–27

[57] Kutz J N, Brunton S L, Brunton B W and Proctor J L 2016 *Dynamic Mode Decomposition: Data-Driven Modeling of Complex Systems* (Philadelphia, PA: SIAM) (https://doi.org/10.1137/1.9781611974508)

[58] Mezić I 2005 Spectral properties of dynamical systems, model reduction and decompositions *Nonlinear Dyn.* **41** 309–25

[59] Koltai P, Wu H, Noé F and Schütte C 2018 Optimal data-driven estimation of generalized Markov state models for non-equilibrium dynamics *Computation* **6** 22

[60] Wu H and Noé F 2020 Variational approach for learning Markov processes from time series data *J. Nonlinear Sci.* **30** 23–66

[61] Lasota A and Mackey M C 1994 *Chaos, Fractals and Noise: Stochastic Aspects of Dynamics* vol 97 (New York: Springer Science & Business Media) (https://doi.org/10.1007/978-1-4612-4286-4)

[62] Boyarsky A and Góra P 1997 *Laws of Chaos: Invariant Measures and Dynamical Systems in One Dimension* (Boston, MA: Birkhäuser Boston) (https://doi.org/10.1007/978-1-4612-2024-4)

[63] Tian W and Wu H 2021 Kernel embedding based variational approach for low-dimensional approximation of dynamical systems *Comput. Methods Appl. Math.* **21** 635–60

[64] Denner A 2017 Coherent structures and transfer operators *Dissertation* Technische Universität München, München, Germany

[65] Meyn S P and Tweedie R L 2012 *Markov Chains and Stochastic Stability* (Berlin: Springer Science & Business Media) (https://doi.org/10.1007/978-1-4471-3267-7)

[66] Schütte C and Sarich M 2013 *Metastability and Markov State Models in Molecular Dynamics* vol 24 (Providence, RI: American Mathematical Society) (https://doi.org/10.1090/cln/024)

[67] Klus S, Husic B E, Mollenhauer M and Noé F 2019 Kernel methods for detecting coherent structures in dynamical data *Chaos* **29** 123112

[68] Glielmo A, Husic B E, Rodriguez A, Clementi C, Noé F and Laio A 2021 Unsupervised learning methods for molecular simulation data *Chem. Rev.* **121** 9722–58

[69] Proctor J L, Brunton S L and Kutz J N 2016 Dynamic mode decomposition with control *SIAM J. Appl. Dyn. Syst.* **15** 142–61

[70] Jovanović M R, Schmid P J and Nichols J W 2014 Sparsity-promoting dynamic mode decomposition *Phys. Fluids* **26** 024103

[71] Benjamin Erichson N, Mathelin L, Kutz J N and Brunton S L 2019 Randomized dynamic mode decomposition *SIAM J. Appl. Dyn. Syst.* **18** 1867–91

[72] Brunton S L, Brunton B W, Proctor J L, Kaiser E and Kutz J N 2017 Chaos as an intermittently forced linear system *Nat. Commun.* **8** 1–9

[73] Bagheri S 2014 Effects of weak noise on oscillating flows: linking quality factor, Floquet modes and Koopman spectrum *Phys. Fluids* **26** 094104

[74] Hemati M S, Rowley C W, Deem E A and Cattafesta L N 2017 De-biasing the dynamic mode decomposition for applied Koopman spectral analysis *Theor. Comput. Fluid Dyn.* **31** 349–68

[75] Dawson S T M, Hemati M S, Williams M O and Rowley C W 2016 Characterizing and correcting for the effect of sensor noise in the dynamic mode decomposition *Exp. Fluids* **57** 1–19

[76] Takeishi N, Kawahara Y, Tabei Y and Yairi T 2017 Bayesian dynamic mode decomposition *26th Int. Joint Conf. on Artificial Intelligence* (https://doi.org/10.24963/ijcai.2017/392)

[77] Askham T and Kutz J N 2018 Variable projection methods for an optimized dynamic mode decomposition *SIAM J. Appl. Dyn. Syst.* **17** 380–416

[78] Azencot O, Yin W and Bertozzi A 2019 Consistent dynamic mode decomposition *SIAM J. Appl. Dyn. Syst.* **18** 1565–85

[79] Williams M O, Kevrekidis I G and Rowley C W 2015 A data-driven approximation of the Koopman operator: extending dynamic mode decomposition *J. Nonlinear Sci.* **25** 1307–46

[80] Pérez-Hernández G, Paul F, Giorgino T, De Fabritiis G and Noé F 2013 Identification of slow molecular order parameters for Markov model construction *J. Chem. Phys.* **139** 015102

[81] Schwantes C R and Pande V S 2013 Improvements in Markov state model construction reveal many non-native interactions in the folding of NTL9 *J. Chem. Theory Comput.* **9** 2000–9

[82] Nüske F, Keller B G, Pérez-Hernández G, Mey A S J S and Noé F 2014 Variational approach to molecular kinetics *J. Chem. Theory Comput.* **10** 1739–52

[83] Noé F and Nüske F 2013 A variational approach to modeling slow processes in stochastic dynamical systems *Multiscale Model. Simul.* **11** 635–55

[84] Husic B E and Noé F 2019 Deflation reveals dynamical structure in nondominant reaction coordinates *J. Chem. Phys.* **151** 054103

[85] Scherer M K, Husic B E, Hoffmann M, Paul F, Wu H and Noé F 2019 Variational selection of features for molecular kinetics *J. Chem. Phys.* **150** 194108

[86] Chan T F, Golub G H and LeVeque R J 1982 Updating formulae and a pairwise algorithm for computing sample variances *COMPSTAT 1982 5th Symp. (Toulouse)* (Springer) pp 30–41

[87] Bach F R and Jordan M I 2002 Kernel independent component analysis *J. Mach. Learn. Res.* **3** 1–48

[88] Hotelling H 1936 Relations between two sets of variates *Biometrika* **28** 321–77

[89] Williams M O, Rowley C W and Kevrekidis I G 2015 A kernel-based method for data-driven Koopman spectral analysis *J. Comput. Dyn.* **2** 247–65

[90] Klus S, Schuster I and Muandet K 2020 Eigendecompositions of transfer operators in reproducing kernel Hilbert spaces *J. Nonlinear Sci.* **30** 283–315

[91] Takeishi N, Kawahara Y and Yairi T 2017 Learning Koopman invariant subspaces for dynamic mode decomposition *Advances in Neural Information Processing Systems* pp 1130–40

[92] Yeung E, Kundu S and Hodas N 2019 Learning deep neural network representations for Koopman operators of nonlinear dynamical systems *2019 American Control Conf. (ACC)* pp 4832–9

[93] Chen W and Ferguson A L 2018 Molecular enhanced sampling with autoencoders: on-the-fly collective variable discovery and accelerated free energy landscape exploration *J. Comput. Chem.* **39** 2079–102

[94] Kingma D P and Welling M 2013 Auto-encoding variational Bayes (arXiv:1312.6114)

[95] Rezende D J, Mohamed S and Wierstra D 2014 Stochastic backpropagation and approximate inference in deep generative models *Int. Conf. on Machine Learning* (PMLR) pp 1278–86

[96] Kadir T and Brady M 2001 Saliency, scale and image description *Int. J. Comput. Vis.* **45** 83–105

[97] Lloyd S 1982 Least squares quantization in PCM *IEEE Trans. Inf. Theory* **28** 129–37

[98] McGibbon R T and Pande V S 2015 Variational cross-validation of slow dynamical modes in molecular kinetics *J. Chem. Phys.* **142** 03B621_1

[99] Noé F and Clementi C 2015 Kinetic distance and kinetic maps from molecular dynamics simulation *J. Chem. Theory Comput.* **11** 5002–11

[100] Kraft D *et al* 1988 A software package for sequential quadratic programming *Technical Report* (Germany: DFVLR Obersfaffeuhofen)

[101] Kingma D P and Ba J 2014 Adam: a method for stochastic optimization (arXiv:1412.6980)

[102] Froyland G, Santitissadeekorn N and Monahan A 2010 Transport in time-dependent dynamical systems: finite-time coherent sets *Chaos* **20** 043116

[103] Banisch R and Koltai P 2017 Understanding the geometry of transport: diffusion maps for Lagrangian trajectory data unravel coherent sets *Chaos* **27** 035804
[104] Bickley W G 1937 LXXIII. The plane jet *London, Edinburgh Dublin Phil. Mag. J. Sci.* **23** 727–31
[105] Rypina I I, Brown M G, Beron-Vera F J, Koçak H, Olascoaga M J and Udovydchenkov I A 2007 On the Lagrangian dynamics of atmospheric zonal jets and the permeability of the stratospheric polar vortex *J. Atmos. Sci.* **64** 3595–610
[106] Froyland G and Padberg-Gehle K 2012 Finite-time entropy: a probabilistic approach for measuring nonlinear stretching *Physica* D **241** 1612–28
[107] Hadjighasem A, Karrasch D, Teramoto H and Haller G 2016 Spectral-clustering approach to Lagrangian vortex detection *Phys. Rev.* E **93** 063107
[108] Husic B E, Schlueter-Kuck K L and Dabiri J O 2019 Simultaneous coherent structure coloring facilitates interpretable clustering of scientific data by amplifying dissimilarity *PLoS One* **14** e0212442
[109] Metzner P, Schütte C and Vanden-Eijnden E 2009 Transition path theory for Markov jump processes *Multiscale Model. Simul.* **7** 1192–219
[110] Röblitz S and Weber M 2013 Fuzzy spectral clustering by PCCA+: application to Markov state models and data classification *Adv. Data Anal. Classif.* **7** 147–79
[111] Trendelkamp-Schroer B, Wu H, Paul F and Noé F 2015 Estimation and uncertainty of reversible Markov models *J. Chem. Phys.* **143** 174101
[112] Olsson S, Wu H, Paul F, Clementi C and Noé F 2017 Combining experimental and simulation data of molecular processes via augmented Markov models *Proc. Natl Acad. Sci.* **114** 8265–70
[113] Baum L E and Eagon J A 1967 An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology *Bull. Am. Math. Soc.* **73** 360–3
[114] Baum L E, Petrie T, Soules G and Weiss N 1970 A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains *Ann. Math. Stat.* **41** 164–71
[115] Baum L E *et al* 1972 An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes *Inequalities* **3** 1–8
[116] Viterbi A 1967 Error bounds for convolutional codes and an asymptotically optimum decoding algorithm *IEEE Trans. Inf. Theory* **13** 260–9
[117] Klus S, Nüske F, Peitz S, Niemann J-H, Clementi C and Schütte C 2020 Data-driven approximation of the Koopman generator: model reduction, system identification and control *Physica* D **406** 132416
[118] Mangan N M, Brunton S L, Proctor J L and Kutz J N 2016 Inferring biological networks by sparse identification of nonlinear dynamics *IEEE Trans. Mol. Biol. Multi-Scale Commun.* **2** 52–63
[119] Champion K, Lusch B, Kutz J N and Brunton S L 2019 Data-driven discovery of coordinates and governing equations *Proc. Natl Acad. Sci.* **116** 22445–51
[120] Champion K, Zheng P, Aravkin A Y, Brunton S L and Kutz J N 2020 A unified sparse optimization framework to learn parsimonious physics-informed models from data *IEEE Access* **8** 169259–71
[121] Kaiser E, Kutz J N and Brunton S L 2018 Sparse identification of nonlinear dynamics for model predictive control in the low-data limit *Proc. R. Soc.* A **474** 20180335
[122] Loiseau J-C and Brunton S L 2018 Constrained sparse Galerkin regression *J. Fluid Mech.* **838** 42–67
[123] Rudy S H, Brunton S L, Proctor J L and Kutz J N 2017 Data-driven discovery of partial differential equations *Sci. Adv.* **3** e1602614
[124] Schaeffer H 2017 Learning partial differential equations via data discovery and sparse optimization *Proc. R. Soc.* A **473** 20160446
[125] Boninsegna L, Nüske F and Clementi C 2018 Sparse learning of stochastic dynamical equations *J. Chem. Phys.* **148** 241723
[126] Klus S, Nüske F, Peitz S, Niemann J-H, Clementi C and Schütte C 2020 Data-driven approximation of the Koopman generator: model reduction, system identification and control *Physica* D **406** 132416
[127] Callaham J L, Loiseau J-C, Rigas G and Brunton S L 2021 Nonlinear stochastic modelling with Langevin regression *Proc. R. Soc.* A **477** 20210092
[128] Schaeffer H and McCalla S G 2017 Sparse model selection via integral terms *Phys. Rev.* E **96** 023302
[129] Gurevich D R, Reinbold P A K and Grigoriev R O 2019 Robust and optimal sparse regression for nonlinear PDE models *Chaos* **29** 103113
[130] Reinbold P A K, Gurevich D R and Grigoriev R O 2020 Using noisy or incomplete data to discover models of spatiotemporal dynamics *Phys. Rev.* E **101** 010203
[131] Tran G and Ward R 2017 Exact recovery of chaotic systems from highly corrupted data *Multiscale Model. Simul.* **15** 1108–29
[132] Schaeffer H, Tran G, and Ward R 2017 Learning dynamical systems and bifurcation via group sparsity (arXiv:1709.01558)
[133] Zhang L and Schaeffer H 2019 On the convergence of the SINDy algorithm *Multiscale Model. Simul.* **17** 948–72
[134] Gelß P, Klus S, Eisert J and Schütte C 2019 Multidimensional approximation of nonlinear dynamical systems *J. Comput. Nonlinear Dyn.* **14** 061006
[135] Deng N, Noack B R, Morzynski M and Pastur L R 2020 Low-order model for successive bifurcations of the fluidic pinball *J. Fluid Mech.* **884** A37
[136] Beetham S, Fox R O and Capecelatro J 2021 Sparse identification of multiphase turbulence closures for coupled fluid–particle flows *J. Fluid Mech.* **914** A11
[137] Rössler O E 1976 An equation for continuous chaos *Phys. Lett.* A **57** 397–8