AutoKoopman: A Toolbox for Automated System Identification via Koopman Operator Linearization

ABSTRACT

While Koopman operator linearization has brought many advances for prediction, control, and verification of dynamical systems, its main disadvantage is that the quality of the resulting model heavily depends on the correct tuning of hyper-parameters such as the number of observables. Our `AutoKoopman` toolbox is a Python package that automates learning accurate models in a Koopman linearized representation at low effort, offering several tuning strategies to optimize the hyper-parameters associated with the Koopman operator techniques automatically. `AutoKoopman` supports discrete as well as continuous-time models and implements all major types of observables, which are polynomials, random Fourier features, and neural networks. As we demonstrate on several benchmarks, our toolbox is able to automatically identify very accurate dynamic models for symbolic, black-box, as well as real systems.

KEYWORDS

Koopman operator linearization, system identification, random Fourier features, deep Koopman.

ACM Reference Format:

. 2023. AutoKoopman: A Toolbox for Automated System Identification via Koopman Operator Linearization. In *26th ACM International Conference on Hybrid Systems: Computation and Control (HSCC '23), May 9–12, 2023, San Antonio, TX, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Identifying models that represent the dynamic behavior of systems is a central challenge in science and engineering. Koopman operator theory has emerged as a useful perspective of these systems because it equivalently represents nonlinear systems by higher-dimensional linear system [18], allowing one to leverage a robust body of linear system analysis methods. In particular, Koopman operator linearization transforms the states of the original system into a new space of observables where the dynamics are linear. Obtaining the nonlinear mapping from states to observables is a prime challenge and often inhibits the usage of the Koopman framework in practice. Our `AutoKoopman` toolbox addresses this issue by automating this process, which also enables non-experts to efficiently use the Koopman framework.

HSCC '23, May 9–12, 2023, San Antonio, TX, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1.1 State of the Art

Koopman theory has long been used for analysis [5, 27], control [19, 32], and verification [2, 3] of dynamical systems, and also several theoretical contributions motivate it as a well-suited representation for data-driven models [26, 28]. For identifying linear systems, dynamic mode decomposition (DMD) is the ubiquitous method for learning the system matrices from data trajectories [36, 38]. While DMD produces a best-fit linear model, advances have been made to extend it to highly nonlinear systems by fixing the observables in the Koopman framework to a specified function [43, 44]. Also, variants for DMD have been developed for many adjacent problems, such as control [34], noisy models [41], and multi-resolution components [20]. More recently, techniques leverage deep learning to discover arbitrary Koopman representations. These techniques often use autoencoders to learn an observables mapping [1, 42, 46]. A notable variation to the autoencoder architecture aims to reduce the dimensionality of the observable space by adding an auxiliary network that parameterizes continuous spectra of a system [23]. Finally, Koopman linearization is related to the more general problem of coordinate discovery. While Koopman linearization embeds states in a space where their dynamics become linear, coordinate discovery embeds in them a space where the dynamics become simpler but can remain weakly nonlinear. Recent work combines sparse symbolic regression and deep learning approaches to identify such spaces [8, 29].

Some libraries exist to identify dynamics: pySINDy [10] uses sparse regression to learn system equations. The packages pyDMD [12] and pyKoopman [33] implement system identification algorithms via variants of DMD, though they do not provide deep learning-based approaches. Moreover, the estimators in these packages depend on numerous hyper-parameters that need to be tuned either manually or with another framework. To the best of our knowledge, there do not exist any libraries that provide general implementations of Koopman approaches with neural network observables.

1.2 Contribution

In this paper, we present the `AutoKoopman` toolbox, which provides:

(1) A high-level tool that automatically optimizes all hyper-parameters to obtain an accurate Koopman linearized model.
(2) A software architecture with a modular design, making it easy to add custom modules.
(3) A collection of Koopman based algorithms centered on conventional dynamic mode decomposition and deep learning.
(4) All major types of static observables, which are polynomials, random Fourier features, and neural networks.
(5) Support for learning discrete-time as well as continuous-time Koopman linearized systems.

The resulting Koopman linearized models are well-suited for solving a variety of task for cyber-physical systems, such as:

(1) **Prediction:** Predict the evolution of a system over long time horizons
(2) **Control:** Synthesize control signals that achieve desired closed-loop behaviors and are optimal with respect to some objective.
(3) **Verification:** Prove or falsify the safety requirements of a system.

We demonstrate the performance of the Koopman linearized models identified by our toolbox for some of these tasks in Sec. 4.

2 KOOPMAN OPERATOR LINEARIZATION

We aim to identify the dynamics of a continuous-time system

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) \tag{1}$$

with state space $\mathbf{x}(t) \in \mathcal{X} \subseteq \mathbb{R}^n$ and input space $\mathbf{u}(t) \in \mathcal{U} \subseteq \mathbb{R}^m$ from a set of o trajectories

$$\mathcal{T} = \left\{ [(\mathbf{x}_i(t_0), \mathbf{u}_i(t_0)), \ldots, (\mathbf{x}_i(t_{s_i-1}), \mathbf{u}_i(t_{s_i-1}))] \right\}_{i=1}^{o},$$

where the i-th trajectory has s_i snapshots. For the ease of presentation we assume that the observations are uniformly sampled with time step size $\Delta t = t_{k+1} - t_k$, even though `AutoKoopman` also supports non-uniform trajectories. Since we only require the states at specific time points, we from now on consider the equivalent discrete-time system

$$\mathbf{x}_{k+1} = \mathbf{F}_{\Delta t}(\mathbf{x}_k, \mathbf{u}_k), \tag{2}$$

instead of (1), where the flow function $\mathbf{F}_t : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ is defined as

$$\mathbf{F}_t(\mathbf{x}_0, \mathbf{u}(t)) = \mathbf{x}_0 + \int_0^t \mathbf{f}(\mathbf{x}(\tau), \mathbf{u}(\tau)) d\tau, \tag{3}$$

and we use the shorthand notation $\mathbf{x}_k = \mathbf{x}(t_k)$. The only requirement we have on the system in (2) is that it is possible to generate system trajectories from it. Consequently, the system dynamics can be described by an arbitrary black-box function $\mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))$, which includes hybrid systems, discrete-time systems, and even real systems.

Rather than learning $\mathbf{f}$ generically, we use Koopman operator linearization, which results in a globally linear representation of the system. Koopman linearization utilizes a new space $\mathcal{H}$, which is defined by the image of the states and inputs through an observables function $\mathbf{g} : \mathcal{X} \times \mathcal{U} \to \mathcal{H}$. The dynamics of the system are then governed by a linear operator $\mathcal{K}_t : \mathcal{H} \to \mathcal{H}$,

$$\mathcal{K}_t\big(\mathbf{g}(\mathbf{x}_0, \mathbf{u}_0)\big) = \mathbf{g}\big(\mathbf{F}_t(\mathbf{x}_0, \mathbf{u}_0), \mathbf{u}(t)\big), \tag{4}$$

which is the so-called Koopman operator. Note that since it is linear, the Koopman operator $\mathcal{K}_t(\mathbf{x}) = K_t \mathbf{x}$ can be represented by a matrix $K_t \in \mathbb{R}^{p \times p}$. While for each nonlinear system there exists a space $\mathcal{H}$ that enables the exact Koopman linearization in (4), this space is infinite dimensional in general. While techniques exist which can learn an infinite dimensional linear operator implicitly—e.g. kernel methods [44]—we utilize ones that learn the operator explicitly. Therefore, our goal is to find a finite space $\mathcal{H}$ where the dynamics can be approximated well by a linear operator. This corresponds to the following optimization problem for finding the observables $\mathbf{g}$ and the Koopman operator $K_{\Delta t}$ via minimization of the mean square error over the set of trajectories $\mathcal{T}$:

$$\underset{\mathbf{g}, K_{\Delta t}}{\operatorname{argmin}} \sum_{(\mathbf{x}_k, \mathbf{u}_k) \in \mathcal{T}} \left\| \mathbf{g}(\mathbf{x}_{k+1}, \mathbf{u}_{k+1}) - K_{\Delta t} \mathbf{g}(\mathbf{x}_k, \mathbf{u}_k) \right\|_{\text{MSE}}. \tag{5}$$

While (5) considers the error for the prediction over one time step, it is also possible to minimize the aggregated error for the prediction over multiple time steps. Since finding the optimal solution for the optimization problem (5) is infeasible in general, one instead aims to compute a close-to-optimal solution in a heuristic way, often by fixing the observables $\mathbf{g}$ and then learning $K_{\Delta t}$. Moreover, it is common practice to consider an observable mapping $\mathbf{g}(\mathbf{x}_k, \mathbf{u}_k) = [\mathbf{g}_x(\mathbf{x}_k), \mathbf{u}_k]$ with an observable function $\mathbf{g}_x$ that acts only on the states. In this case the dynamics of the Koopman linearized system (4) can be formulated as

$$\mathbf{g}_x(\mathbf{x}_{k+1}) = A\, \mathbf{g}_x(\mathbf{x}_k) + B\, \mathbf{u}_k, \quad A \in \mathbb{R}^{p \times p},\ B \in \mathbb{R}^{p \times m}, \tag{6}$$

where $K_{\Delta t} = [A, B]$. Note that it is also possible to directly identify a continuous-time instead of a discrete-time model, which we describe in detail in Sec. 3.3.1.

3 TOOLBOX FEATURES

We now present the features and implementation details of the `AutoKoopman` package. `AutoKoopman` is written in Python 3.9 and uses the third-party packages Pytorch for deep learning, pySindy for system identification, and pyDMD for dynamic mode composition. Fig. 1 outlines the package architecture. The toolbox uses a modular design, making it very easy to exchange single modules by custom implementations and to add new functionality. For example, one can easily add different optimization algorithms for performing the hyper-parameter tuning. Moreover, the toolbox provides a convenience function `auto_koopman` that allows the user to access the whole functionality of the toolbox with a single function call and without requiring any knowledge about the underlying class structure. Let us demonstrate this convenience function with the following code example:

```
# this is the convenience function
from autokoopman import auto_koopman

# learn model from data
experiment_results = auto_koopman(
   training_data,     # list of trajectories
   obs_type="rff",    # rand. Fourier feat. observables
   opt="bopt",        # auto-tuning via bayesian opt.
   n_obs=(50, 200),   # range for num. of observables
   max_opt_iter=200, # max. optimization iterations
   n_splits=5,        # splits for k-folds validation
   rank=(1, 200)      # range for rank in DMD
)
```

The results returned by this code example contain the identified Koopman model as well as the optimal parameter values determined by hyper-parameter optimization. The convenience function provides many settings to refine the training and optimization process, but they all have reasonable defaults to learn accurate models even if the user does not specify any settings.

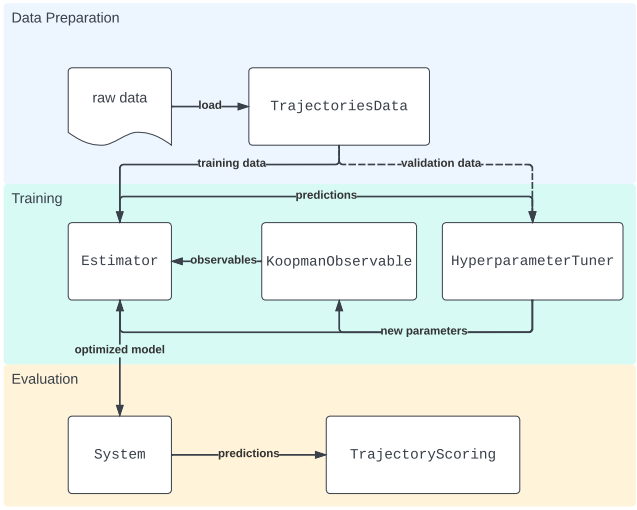

Figure 1: High-level structure of the `AutoKoopman` toolbox

3.1 Trajectories Data Preparation

The input to `AutoKoopman` is a set of trajectories $\mathcal{T}$, which is represented as an object of class `TrajectoriesData` class. This class provides many data preparation methods so that `AutoKoopman` can automatically pre-process the data before training. For example, if a discrete-time model should be identified and the provided data is not uniformly sampled, interpolation methods are used to convert the data to a uniform time series. Moreover, numerical differentiation methods for approximating the time-derivative are available, which is required for identifying continuous-time models. Overall, `AutoKoopman` therefore has no special requirements on the format of the data, making it very convenient to use the toolbox.

3.2 Types of Observables

The observable mapping is represented by the `KoopmanObservable` class. The main method of that class accepts a state and input and returns an element in the observables space $\mathbf{g} : \mathcal{X} \times \mathcal{U} \to \mathcal{H}$. One convenient feature is that the observable functions are composable: Any two observables functions $\mathbf{g}_1, \mathbf{g}_2$ can be augmented together to create larger sets of observables $\mathbf{g}(\mathbf{x}, \mathbf{u}) = [\mathbf{g}_1(\mathbf{x}, \mathbf{u}), \mathbf{g}_2(\mathbf{x}, \mathbf{u})]$. An exemplary use-case for this are applications where it is required to recover the original system state from the observable space, which can be achieved by adding identity observables $\mathbf{g}_1(\mathbf{x}, \mathbf{u}) = \mathbf{x}$.

3.2.1 Random Fourier Features. Random Fourier features [35] are used to approximate kernel DMD using extended DMD [11]. Kernel DMD is part of a class of algorithms that employ kernel functions, a symmetric, positive definite function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ that encodes the similarity of two observations. Kernels are especially advantageous for high-dimensional feature spaces, where instead of explicitly computing elements of that space, one can instead efficiently compute the similarity between data pairs via the kernel function. Commonly used kernel functions are radial basis functions, polynomials, and spline kernels. In our case we cannot use the kernel function directly since we require an explicit representation of the observable mapping $\mathbf{g}$. We therefore need a method to exploit the advantageous properties of kernels, without sacrificing the explicit observable mapping. Random Fourier features achieve this for stationary kernels by utilizing a connection of a kernel to its Fourier transform [35]. The function $\mathbf{g}_x(\mathbf{x}) = [g_1(\mathbf{x}), \ldots, g_p(\mathbf{x})]$ approximately preserves the kernel function $k(\mathbf{x}, \mathbf{x}') \approx \mathbf{g}_x(\mathbf{x})^T \mathbf{g}_x(\mathbf{x}')$ within some error bound, where the scalar observables take the form

$$g_i(\mathbf{x}) = \sqrt{2} \cos\left(\omega_i^T \mathbf{x} + b_i\right), \quad i = 1, \ldots, p.$$

Here, $b_i \in \mathbb{R}$ is selected uniformly from the interval $[0, 2\pi]$ and $\omega_i \in \mathbb{R}^n$ is drawn from a probability distribution $\mu(\omega)$ corresponding to the kernel function used. A normal distribution, for example, is used for the radial basis function kernel [35, Fig. 1].

3.2.2 Polynomials. Carleman linearization equivalently represents the dynamic behavior of a polynomial system by an infinite dimensional linear system [7]. This linearization can be viewed as Koopman linearization using a polynomial basis to span the function space. So, a natural choice for observables is a set of multi-variate monomials. For a specific polynomial system only a finite number of monomial terms are required if the vector space spanned by the observables is closed under the operation of Lie-derivatives [37]. However, the number of monomials that exist grows significantly with the dimensions and maximum order of the polynomial. For high polynomial orders, it is therefore often advantageous to represent the polynomial observables implicitly using kernel DMD.

3.2.3 Neural Network Observables. Since neural networks can be trained to represent arbitrary functions, a natural idea is to use them as observables. This approach has been shown to perform very well in many applications such as fluid control [30], object-centric physics [21], and autonomous driving [45]. To find both the mapping and linear dynamics simultaneously, deep learning has been applied successfully. For this, an autoencoder architecture consisting of a encoder/decoder pair $\mathbf{g}_x$, $\mathbf{g}_x^{-1}$ is used, where the encoder $\mathbf{g}_x : \mathcal{X} \to \mathcal{H}$ maps from the original state space to the observable space and the decoder $\mathbf{g}_x^{-1} : \mathcal{H} \to \mathcal{X}$ maps from the observable space back to the original space. Both, the encoder as well as the decoder are represented by neural networks. These networks depend on several hyper-parameters—e.g., the number of hidden layers, the dimensions of the hidden layers, and the type of activation function—which can either be set by the user or tuned automatically by `AutoKoopman`. Finally, the loss function for neural network training consists of several loss terms:

(1) Reconstruction loss: $\left\| \mathbf{x}_k - \mathbf{g}_x^{-1}\left(\mathbf{g}_x\left(\mathbf{x}_k\right)\right)\right\|_{\text{MSE}}$

(2) Prediction loss: $\left\| \mathbf{x}_{k+1} - \mathbf{g}_x^{-1}\left(K_{\Delta t}\, \mathbf{g}\left(\mathbf{x}_0, \mathbf{u}_0\right)\right)\right\|_{\text{MSE}}$

(3) Linearity loss: $\left\| \mathbf{g}\left(\mathbf{x}_{k+1}, \mathbf{u}_{k+1}\right) - K_{\Delta t}\, \mathbf{g}\left(\mathbf{x}_k, \mathbf{u}_k\right)\right\|_{\text{MSE}}$

(4) Metric loss: $\sum_{i,j} \left| \|\mathbf{g}_x(\mathbf{x}_i) - \mathbf{g}_x(\mathbf{x}_j)\| - \|\mathbf{x}_i - \mathbf{x}_j\| \right|$

Here, the reconstruction loss captures the reconstruction accuracy of the autoencoder, the prediction loss quantifies how well the network predicts future states, the linearity loss, which is identical to the general cost function in (5), evaluates how well the network predicts future states in the observable space, and the metric loss encourages that distances between states in the original and in the observable space are preserved, which aims to prevent obtaining

observables with very large values. Note that the prediction and linearity loss can also be computed over multiple time steps, which often improves the results.

3.3 Regression Estimators

The algorithms represented by the `Estimator` class implement different methods for solving the optimization problem (5).

3.3.1 Dynamic Mode Decomposition. While for neural network observables the Koopman operator is determined together with the observables via deep learning, the common practice for polynomial and random Fourier feature observables is to first determine suitable observables, and then find the Koopman operator that yields the best-fit linear model for these observables. In this case the optimal solution for the optimization problem (5) can be determined via extended dynamic mode decomposition (eDMD) which performs DMD in the observables space [43]. For eDMD one first constructs the matrices $X = [\mathbf{g}_x(\mathbf{x}_0), \dots, \mathbf{g}_x(\mathbf{x}_{s-1})]$, $X' = [\mathbf{g}_x(\mathbf{x}_1), \dots, \mathbf{g}_x(\mathbf{x}_s)]$ and $U = [\mathbf{u}_0, \dots, \mathbf{u}_{s-1}]$ from the set of trajectories $\mathcal{T}$. The dynamics of the Koopman linearized system (6) can then be expressed as $X' = AX + BU = K_{\Delta t}[X, U]$, so that the optimization problem (5) becomes

$$\underset{K_{\Delta t}}{\operatorname{argmin}} \|X' - K_{\Delta t}[X, U]\|_{MSE}.$$

It is well known that the solution that minimizes the mean square error is given as $K_{\Delta t} = X'[X, U]^+$, where $[X, U]^+$ denotes the Moore-Penrose inverse. Since we usually have a lot of data, the matrix $[X, U]$ is very large. For computational efficiently, the Moore-Penrose inverse is therefore estimated by a low-rank matrix approximation using singular value decomposition (SVD). The rank for SVD is a hyper-parameter, which if tuned properly helps to avoid over-fitting the model to high-frequency noise in the data. If the goal is to identify a continuous-time instead of a discrete-time linear system, we simply have to exchange the matrix X' by the corresponding time-derivatives $X' = [\partial \mathbf{g}_x(\mathbf{x}_0)/\partial t, \dots, \partial \mathbf{g}_x(\mathbf{x}_{s-1})/\partial t]$, which can be estimated using numerical differentiation

3.3.2 Sparse Regression. Though Koopman theory treats the dynamics in the observables space as linear, techniques exist that relax the observables to a change of coordinates yielding simpler but still nonlinear dynamics [8]. In particular, the goal is to obtain dynamics with only few nonlinear terms, where the nonlinear dynamics is represented as a symbolic closed-form expression. This method is called sparse identification of nonlinear dynamics (SINDy). SINDy treats system identification as a sparse regression problem, attempting to determine terms in the dynamics that are similar to terms in a provided library [6]. This approach comes with a collection of hyper-parameters, which are the library of terms to consider during regression as well as additional sparsity optimization parameters.

3.4 Hyper-parameter Tuning

As shown in the previous section, the overall Koopman linearization process introduces several hyper-parameters. The accuracy of an identified model is typically very sensitive to the choice of these parameters, making adequate manual tuning challenging. `AutoKoopman` therefore provides several strategies for tuning these hyper-parameters automatically, which are presented in this section. All of these strategies apply cross-validation, where the dataset $\mathcal{T}$ is split into a training set $\mathcal{T}_{train}$ and a validation set $\mathcal{T}_{val}$. `AutoKoopman` also supports k-folds cross-validation, which trains the model k times over disjoint validation datasets. This usually results in a better model, but also prolongs the computation time. As a metric for the accuracy of the identified model the loss function for the optimization problem in (5) is used.

3.4.1 Grid Search. Grid search exhaustively samples values for hyper-parameters by generating candidates from a grid. While suitable default values for the search ranges for all hyper-parameters are provided, `AutoKoopman` also allows the user to manually specify ranges for the hyper-parameters that should be optimized. This range is then discretized automatically for the grid search. Grid search is reliable but suffers from the curse of dimensionality; it works well in low-dimensional spaces, but the number of candidates grows exponentially as the hyper-parameter space dimension increases. Grid search is therefore well-suited for polynomial and random Fourier feature observables which only depend on few hyper-parameters, but computationally demanding for neural network observables that come with many parameters.

3.4.2 Random Search. For random search all hyper-parameters are treated as independent random variables with either uniform or log-uniform distribution. The optimization algorithm then samples a fixed number of times or until a given compute budget is exceeded to determine good hyper-parameters. While by default `AutoKoopman` automatically decides for which parameters to use a uniform and for which a log-uniform distribution, this can also be explicitly specified by the user if desired. Random search has many practical advantages compared to grid search—simplicity and parallelism—and performs better in high-dimensional spaces [4].

3.4.3 Bayesian Optimization. Bayesian optimization can quickly find the global minimum of a multi-dimensional function by incorporating information learned from previous hyper-parameter evaluations [39]. This is achieved by constructing the posterior predictive distribution for the loss function. While Bayesian optimization is known for being a more efficient hyper-parameter tuning method than random or grid search, it comes with its own set of hyper-parameters: a covariance function to model the posterior distribution and an acquisition function to select the next batch of points. For the Bayesian optimizer implemented in `AutoKoopman` we choose the commonly used Matern52 covariance function and use a heuristic to determine its lengthscale.

4 NUMERICAL EXPERIMENTS

In this section we evaluate the performance of `AutoKoopman` on several benchmark systems. These benchmarks can be categorised into three groups: symbolic models, black-box models, and real data. The results for different types of observables are summarized in Tab. 1. For comparison, we also added the results for identity observables, which corresponds to identifying a linear model without using the Koopman framework. As a metric for the accuracy of the identified models we use the relative error based on the Euclidean

Table 1: Comparison of the constructed Koopman linearized models for different types of observables with respect to accuracy and the computation time of system identification in seconds, where n is the dimension of the benchmark.

	Benchmark		Identity		Polynomial		Rand. Fourier Feat.		Neural Network	
		n	error	time	error	time	error	time	error	time
symbolic	Pendulum	2	21.4%	0.14	1.37%	6.47	**1.36%**	23.8	97.1%	164
	FitzHugh-Nagumo	2	49.6%	0.11	35.6%	3.09	**0.53%**	23.5	67.8%	130
	Robertson Chemical Reaction	2	**6.83%**	0.11	**6.83%**	1.37	26.7%	23.7	24.8%	296
	Production Destruction	2	27.9%	0.11	27.9%	1.37	**0.53%**	24.1	75.4%	164
	Spring Pendulum	4	89.7%	0.11	89.7%	1.73	**0.03%**	23.9	22.8%	135
	Laub-Loomis	7	5.47%	0.11	0.21%	4.95	**0.04%**	24.7	19.9%	282
	Biological Model	9	0.06%	0.12	0.06%	1.95	**0.01%**	25.2	10.2%	91.9
	Transcriptional Regulator Network	48	27.5%	0.61	27.5%	2.78	**4.60%**	61.2	6.51%	88.9
black-box	Engine Control	2	13.5%	1.17	13.5%	18.8	**2.54%**	122	22.9%	270
	Longitudinal Control	7	3.24%	0.09	3.24%	0.90	**0.00%**	47.6	3.13%	280
	Ground-Collision Avoidance	16	2.61%	6.84	2.61%	126	**2.39%**	567	58.0%	1326
real data	Electric Circuit	3	**14.4%**	13.9	**14.4%**	1123	**14.4%**	862	15.1%	468
	F1tenth Racecar	4	64.1%	0.57	64.1%	26.9	60.1%	48.1	**53.3%**	209
	Robot Arm	12	66.4%	5.74	66.4%	78.8	33.1%	326	**17.1%**	240

norm, which is defined as

$$\epsilon = \frac{1}{s}\sum_{i=1}^{s} \frac{\left\| \mathbf{x}(t_i) - \mathbf{x}_{pred}(t_i) \right\|_2}{\left\| \mathbf{x}(t_i) \right\|_2},$$

where $\mathbf{x}(t)$ is the ground truth data and $\mathbf{x}_{pred}(t)$ is the prediction of the identified model. The error is computed on a validation dataset that is different from the training dataset. For a fair comparison, we use 200 observables for all experiments. Moreover, we apply grid search as the auto-tuning method for identity, polynomial, and random Fourier feature observables, and Bayesian optimization for neural network observables. This choice is motivated by the observation that Bayesian optimization usually performs much better than grid search if many hyper-parameters have to be tuned, as it the case for neural network observables. All computations are carried out on a 3.40 GHz AMD Ryzen 9 5950X 16-core processor with 64GB memory and an RTX 3090 GPU.

Symbolic Models. First, we examine models for which the dynamics is given by a symbolic nonlinear first-order differential equation. In particular, we consider a pendulum (see [40, Fig. 8.11]), a spring pendulum (see [25, Fig. 1.4]), the FitzHugh-Nagumo model [13], the Robertson chemical reaction system [15, Sec. 3.1], the production destruction and Laub-Loomis benchmarks from [14], the biological model in [9, Example 5.2.4], and a transcriptional regulator network from [24, Sec. VIII.D]. We use a randomly generated training and validation dataset consisting of 10 trajectories each as well as a sampling period of 0.1s and a final time of 10s for all models. The results in Tab. 1 demonstrate that the Koopman linearized models computed with `AutoKoopman` are on average much more accurate compared to identifying a linear model, where random Fourier feature observables achieve especially good results. This can also be seen in Fig. 2, where the predicted trajectories for different types of observables are exemplary visualized for the spring pendulum system. Moreover, training neural network observables via deep learning takes significantly longer than DMD used for the other observables. Finally, the comparison of the different hyper-parameter optimization strategies shown in Fig. 3 demonstrates that more sophisticated optimization strategies such as Bayesian optimization often perform better than simple strategies such as random search.

Black-Box Systems. To evaluate the performance of `AutoKoopman` for black-box systems, we consider the model of a F-16 fighter jet ground collision avoidance system [16]. In particular, we examine the following three scenarios: 2-dimensional engine control, 7-dimensional longitude control, and the full 16-dimensional model. We use 400 trajectories for training the 16-dimensional system, and 20 trajectories for the lower-dimensional cases. Moreover, the final time is 50s for the 2-dimensional model and 15s for the two other systems. All other settings are identical to the ones for the symbolic models. The results in Tab. 1 demonstrate that even for very complex black-box systems `AutoKoopman` is able to identify very accurate models in reasonable time. Note that the computation time for the 16-dimensional model is higher since we used a larger training set.

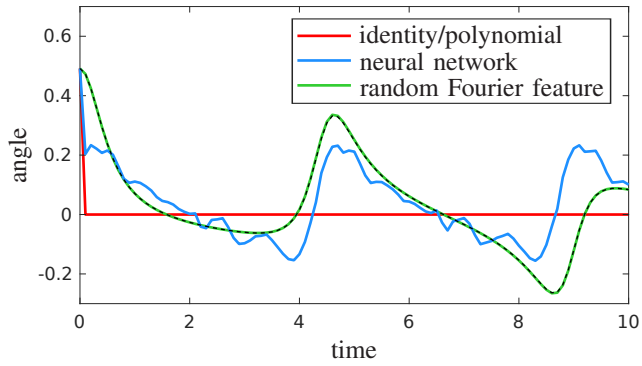

Figure 2: Comparison of predictions for the spring pendulum using different types of observables, where the ground truth is shown by the dashed black line. Identity and polynomial observables yield exactly the same result for this system.

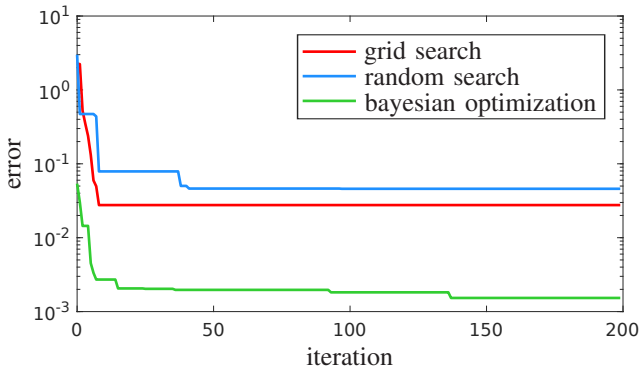

Figure 3: Comparison of different hyper-parameter optimization strategies for identifying a Koopman linearized model with random Fourier feature obervables for the Laub-Loomis benchmark.

Real Measurements. Finally, we can also apply `AutoKoopman` directly to data measured from the real system, which completely eliminates the requirement for a system model. For this we consider a dataset from an electric circuit that represents a LMC6484 lowpass filter [17] consisting of 7 trajectories, a dataset from a 6 degree-of-freedom Schunk LWA 4P robot arm [22] consisting of 100 trajectories, and a dataset from a F1tenth racecar [31] consisting of 41 trajectories. The randomly generated validation dataset consists of 2 trajectories for the electric circuit and 10 trajectories for the two other benchmarks, and we train on all remaining trajectories. The results in Tab. 1 demonstrate that `AutoKoopman` robustly generates accurate models, even though the measured data traces are pertubed by high-frequent measurement errors. Moreover, as shown in Fig. 4, identifying a continuous-time instead of a discrete-time model can often further improve the results since it might prevent over-fitting to noisy data.

Application to System Control. One of the main advantages of the Koopman framework is that the dynamics of the resulting models is linear, which makes them easier to analyse, verify, and control. We demonstrate this on the example of the F1tenth car, for which we design a model predictive controller that solves a

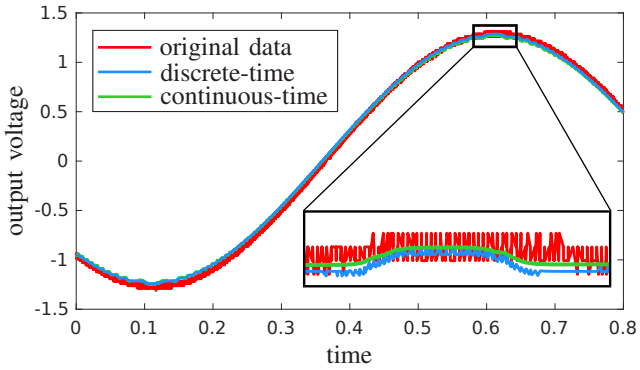

Figure 4: Comparison of predictions from a discrete-time and a continuous-time model identified with `AutoKoopman` on noisy data measured from an electric circuit.

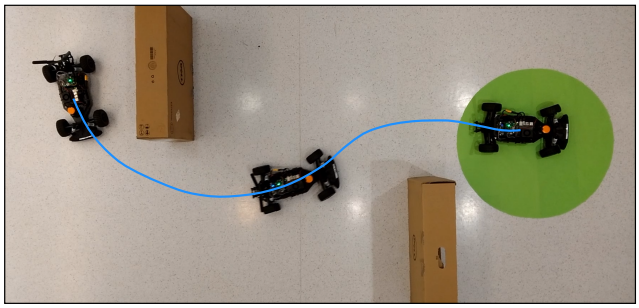

Figure 5: Trajectory driven by the F1tenth car during application of model predictive control, where the goal set is shown in green.

reach-avoid task. In particular, we use the Koopman model that we obtained by applying `AutoKoopman` together with random Fourier feature observables to traces measured from the F1tenth car. Since the dynamics of the system is linear, the optimization problem for model predictive control can be solved very efficiently, which enables us to perform the computations online on the real F1tenth car with a control frequency of 10Hz. Moreover, the results of the experiments visualized in Fig. 5 demonstrate that the model obtained with `AutoKoopman` is accurate enough to steer the car to the desired goal set while avoiding obstacles.

5 CONCLUSION

In this paper, we present `AutoKoopman`, a toolbox for fully automated system identification using Koopman operator linearization. The toolbox implements polynomial observables, random Fourier feature observables, and neural network observables, and enables identifying discrete-time as well as continuous time models. Moreover, `AutoKoopman` provides multiple strategies for optimizing hyper-parameters, and therefore fully automates the system identification process. The numerical results demonstrate that `AutoKoopman` is able to identify accurate Koopman linearized models for symbolic system models, black-box systems as well as data measured from real systems. Moreover, the obtained models are well suited for system control, as we showcase on the example of a F1tenth car.

REFERENCES

[1] D. J. Alford-Lago, C. W. Curtis, A. T. Ihler, and O. Issan. 2022. Deep Learning Enhanced Dynamic Mode Decomposition. *Chaos: An Interdisciplinary Journal of Nonlinear Science* 32, 3, Article 033116 (2022).

[2] S. Bak and et al. 2021. Reachability of Black-Box Nonlinear Systems after Koopman Operator Linearization. In *Proc. of the International Conference on Analysis and Design of Hybrid Systems*. 253–258.

[3] S. Bak and et al. 2022. Reachability of Koopman Linearized Systems Using Random Fourier Feature Observables and Polynomial Zonotope Refinement. In *Proc. of the International Conference on Computer Aided Verification*. 490–510.

[4] J. Bergstra and Y. Bengio. 2012. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research* 13, 2 (2012), 281–305.

[5] P. Bevanda, S. Sosnowski, and S. Hirche. 2021. Koopman Operator Dynamical Models: Learning, Analysis and Control. *Annual Reviews in Control* 52 (2021), 197–212.

[6] S. L. Brunton, J. L. Proctor, and J. N. Kutz. 2016. Discovering Governing Equations from Data by Sparse Identification of Nonlinear Dynamical Systems. *Proceedings of the National Academy of Sciences* 113, 15 (2016), 3932–3937.

[7] T. Carleman. 1932. Application de la Théorie des Équations Intégrales Linéaires aux Systèmes d'Équations Différentielles non Linéaires. *Acta Mathematica* 59 (1932), 63–87.

[8] K. Champion, B. Lusch, J. N. Kutz, and S. L. Brunton. 2019. Data-Driven Discovery of Coordinates and Governing Equations. *Proceedings of the National Academy of Sciences* 116, 45 (2019), 22445–22451.

[9] X. Chen. 2015. *Reachability Analysis of Non-Linear Hybrid Systems Using Taylor Models*. Ph.D. Dissertation. RWTH Aachen University.

[10] B. de Silva and et al. 2020. PySINDy: A Python Package for the Sparse Identification of Nonlinear Dynamical Systems from Data. *Journal of Open Source Software* 5, 49, Article 2104 (2020).

[11] A. M. DeGennaro and N. M. Urban. 2019. Scalable Extended Dynamic Mode Decomposition Using Random Kernel Approximation. *SIAM Journal on Scientific Computing* 41, 3 (2019), 1482–1499.

[12] N. Demo, M. Tezzele, and G. Rozza. 2018. PyDMD: Python Dynamic Mode Decomposition. *Journal of Open Source Software* 3, 22, Article 530 (2018).

[13] R. FitzHugh. 1961. Impulses and Physiological States in Theoretical Models of Nerve Membrane. *Biophysical Journal* 1, 6 (1961), 445–466.

[14] L. Geretti and et al. 2020. ARCH-COMP20 Category Report: Continuous and Hybrid Systems with Nonlinear Dynamics. In *Proc. of the International Workshop on Applied Verification of Continuous and Hybrid Systems*. 49–75.

[15] L. Geretti and et al. 2021. ARCH-COMP21 Category Report: Continuous and Hybrid Systems with Nonlinear Dynamics.. In *Proc. of the International Workshop on Applied Verification of Continuous and Hybrid Systems*. 32–54.

[16] P. Heidlauf, A. Collins, M. Bolender, and S. Bak. 2018. Verification Challenges in F-16 Ground Collision Avoidance and Other Automated Maneuvers. In *Proc. of the International Workshop on Applied Verification of Continuous and Hybrid Systems*. 208–217.

[17] N. Kochdumper and et al. 2020. Establishing Reachset Conformance for the Formal Analysis of Analog Circuits. In *Proc. of the Asia and South Pacific Design Automation Conference*. 199–204.

[18] B. O. Koopman. 1931. Hamiltonian Systems and Transformation in Hilbert Space. *Proceedings of the National Academy of Sciences* 17, 5 (1931), 315–318.

[19] M. Korda and I. Mezić. 2018. Linear Predictors for Nonlinear Dynamical Systems: Koopman Operator meets Model Predictive Control. *Automatica* 93 (2018), 149–160.

[20] J. N. Kutz, X. Fu, and S. L. Brunton. 2016. Multiresolution Dynamic Mode Decomposition. *SIAM Journal on Applied Dynamical Systems* 15, 2 (2016), 713–735.

[21] Y. Li and et al. 2020. Learning Compositional Koopman Operators for Model-Based Control. In *Proc. of the International Conference on Learning Representations*.

[22] S. B. Liu and M. Althoff. 2018. Reachset Conformance of Forward Dynamic Models for the Formal Analysis of Robots. In *Proc. of the International Conference on Intelligent Robots and Systems*. 370–376.

[23] B. Lusch, J. N. Kutz, and S. L. Brunton. 2018. Deep Learning for Universal Linear Embeddings of Nonlinear Dynamics. *Nature Communications* 9, Article 4950 (2018).

[24] M. Maïga, N. Ramdani, L. Travé-Massuyè, and C. Combastel. 2015. A Comprehensive Method for Reachability Analysis of Uncertain Nonlinear Hybrid Systems. *Transactions on Automatic Control* 61, 9 (2015), 2341–2356.

[25] J. D. Meiss. 2007. *Differential Dynamical Systems*. SIAM.

[26] I. Mezić. 2005. Spectral Properties of Dynamical Systems, Model Reduction and Decompositions. *Nonlinear Dynamics* 41, 1 (2005), 309–325.

[27] I. Mezić. 2013. Analysis of Fluid Flows via Spectral Properties of the Koopman Operator. *Annual Review of Fluid Mechanics* 45 (2013), 357–378.

[28] I. Mezić and A. Banaszuk. 2004. Comparison of Systems with Complex Behavior. *Physica D: Nonlinear Phenomena* 197, 1-2 (2004), 101–133.

[29] C. Michoski, M. Milosavljević, T. Oliver, and D. R. Hatch. 2020. Solving Differential Equations Using Deep Neural Networks. *Neurocomputing* 399 (2020), 193–212.

[30] J. Morton, A. Jameson, M. J. Kochenderfer, and F. Witherden. 2018. Deep Dynamical Modeling and Control of Unsteady Fluid Flows. *Advances in Neural Information Processing Systems* 31 (2018).

[31] M. O'Kelly, H. Zheng, D. Karthik, and R. Mangharam. 2020. F1tenth: An Open-Source Evaluation Environment for Continuous Control and Reinforcement Learning. *Proceedings of Machine Learning Research* 123 (2020), 77–89.

[32] S. E. Otto and C. W. Rowley. 2021. Koopman Operators for Estimation and Control of Dynamical Systems. *Annual Review of Control, Robotics, and Autonomous Systems* 4 (2021), 59–87.

[33] S. Pan, E. Kaiser, N. Kutz, and S. Brunton. 2022. PyKoopman: A Python Package for Data-Driven Approximation of the Koopman Operator. *Bulletin of the American Physical Society* (2022).

[34] J. L. Proctor, S. L. Brunton, and J. N. Kutz. 2016. Dynamic Mode Decomposition with Control. *SIAM Journal on Applied Dynamical Systems* 15, 1 (2016), 142–161.

[35] A. Rahimi and B. Recht. 2007. Random Features for Large-Scale Kernel Machines. In *Proc. of the International Conference on Neural Information Processing Systems*. 1177–1184.

[36] C. W. Rowley and et al. 2009. Spectral Analysis of Nonlinear Flows. *Journal of Fluid Mechanics* 641 (2009), 115–127.

[37] S. Sankaranarayanan. 2011. Automatic Abstraction of Non-Linear Systems using Change of Bases Transformations. In *Proc. of the International Conference on Hybrid Systems: Computation and Control*. 143–152.

[38] M. R. Schmid, M. Maehlisch, J. Dickmann, and H.-J. Wuensche. 2010. Dynamic Level of Detail 3D Occupancy Grids for Automotive Use. In *Proc. of the IEEE Intelligent Vehicles Symposium*. 269 – 274.

[39] B. Shahriari and et al. 2015. Taking the Human out of the Loop: A Review of Bayesian Optimization. *Proceedings of the IEEE* 104, 1 (2015), 148–175.

[40] A. L. Stanford and J. M. Tanner. 2014. *Physics for Students of Science and Engineering*. Academic Press.

[41] N. Takeishi, Y. Kawahara, Y. Tabei, and T. Yairi. 2017. Bayesian Dynamic Mode Decomposition. In *Proc. of the AAAI Conference on Artificial Intelligence*. 2814–2821.

[42] N. Takeishi, Y. Kawahara, and T. Yairi. 2017. Learning Koopman Invariant Subspaces for Dynamic Mode Decomposition. *Proc. of the International Conference on Advances in Neural Information Processing Systems*.

[43] M. O. Williams, I. G. Kevrekidis, and C. W. Rowley. 2015. A Data–Driven Approximation of the Koopman Operator: Extending Dynamic Mode Decomposition. *Journal of Nonlinear Science* 25, 6 (2015), 1307–1346.

[44] M. O. Williams, C. W. Rowley, and I. G. Kevrekidis. 2014. A Kernel-Based Approach to Data-Driven Koopman Spectral Analysis. *Journal of Computational Dynamics* 2, 2 (2014), 247–265.

[45] Y. Xiao and et al. 2022. Deep Neural Networks with Koopman Operators for Modeling and Control of Autonomous Vehicles. *Transactions on Intelligent Vehicles* (2022). IEEE Early Access.

[46] E. Yeung, S. Kundu, and N. Hodas. 2019. Learning Deep Neural Network Representations for Koopman Operators of Nonlinear Dynamical Systems. In *Proc. of the American Control Conference*. 4832–4839.