Under consideration for publication in Formal Aspects of Computing

Inferring Switched Nonlinear Dynamical Systems

Xiangyu Jin[1,2], Jie An[3,4], Bohua Zhan[1,2], Naijun Zhan[1,2] and Miaomiao Zhang[3]

[1]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
[2]University of Chinese Academy of Sciences, Beijing, China
[3]School of Software Engineering, Tongji University, Shanghai, China
[4]Max Planck Institute for Software Systems, Kaiserslautern, Germany

Abstract. Identification of dynamical and hybrid systems using trajectory data is an important way to construct models for complex systems where derivation from first principles is too difficult. In this paper, we study the identification problem for switched dynamical systems with polynomial ODEs. This is a difficult problem as it combines estimating coefficients for nonlinear dynamics and determining boundaries between modes. We propose two different algorithms for this problem, depending on whether to perform prior segmentation of trajectories. For methods with prior segmentation, we present a heuristic segmentation algorithm and a way to classify the modes using clustering. For methods without prior segmentation, we extend identification techniques for piecewise affine models to our problem. To estimate derivatives along the given trajectories, we use Linear Multistep Methods. Finally, we propose a way to evaluate an identified model by computing a relative difference between the predicted and actual derivatives. Based on this evaluation method, we perform experiments on five switched dynamical systems with different parameters, for a total of twenty cases. We also compare with three baseline methods: clustering with DBSCAN, standard optimization methods in SciPy and identification of ARX models in Matlab, as well as with state-of-the-art identification method for piecewise affine models. The experiments show that our two methods perform better across a wide range of situations.

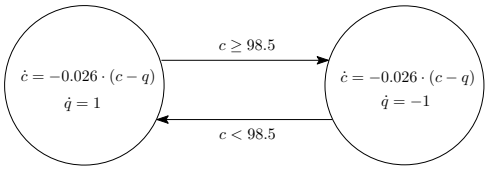
Keywords: System identification; Black-box inference; Grey-box inference; Switched dynamical systems; Linear multistep methods

1. Introduction

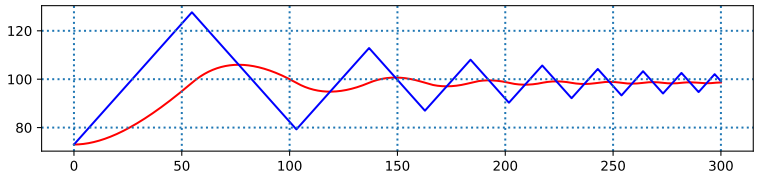
Recent decades witnessed a huge investment in Cyber-Physical Systems (CPS) which have become ubiquitous in our daily life, for example in autonomous vehicles, drones, industrial robots, etc. Along with this increasing adoption comes greater concern for the safety of such systems. Formal design and analysis of embedded control software rely on mathematical models. Thus, constructing formal models for dynamical and hybrid systems is an important yet challenging research problem in computer science and control theory. For many

Correspondence and offprint requests to: Bohua Zhan, e-mail: bzhan@ios.ac.cn

(a) The Isolette system



(b) The corresponding SNDS modelled as a hybrid automaton



(c) A simulated trajectory

Fig. 1. Part (a) shows a real-life Isolette system. Part (b) shows an automaton description of the SNDS. Part (c) shows a simulated trajectory of the system, where q is shown as a blue line and c is shown as a red line.

real-life complex systems, determining a model by derivation from first principles is very difficult. It is common in CPS applications to use many different kinds of sensors and monitors to gather data from systems. There are a lot of work in the machine learning community focusing on inferring features from time-series data. However, many of these methods suffer from a lack of interpretability. In particular, models from deep neural networks are well-known to be difficult to interpret. An alternative way is system identification [Lju15, Lju99] or model learning [Vaa17], i.e. learning a formal model from observable behaviours of the system regarded as a black-box or a grey-box. In black-box learning, the learner has no knowledge about the system. There are two basic types of black-box learning: active learning and passive learning. In the active learning setting, the learner can run the system and gather observable data, but not look inside the system. This is distinguished from passive learning, which involves generating a model from a given data set. For grey-box learning, the learner has some knowledge about the system. For example, we often have systems where we know its signal data matches a combination of basic patterns such as lines, polynomial curves, exponential curves, and sinusoids. However, we do not know how the basic patterns connect to each other and what are the values of parameters in the patterns. Then grey-box learning tries to infer such unknown information from system data. One goal of system identification and model learning is similar to that of machine learning: inferring information from data. However, a major difference is that system identification usually leads to an interpretable formal model.

In this paper, we focus on identifying Switched Nonlinear Dynamical Systems (SNDS) from given trajectories. A switched nonlinear dynamical system consists of a finite number of modes, with continuous behaviour of each mode described by an ordinary differential equation (ODE) with polynomial derivatives. The modes form a partition of the state space of the system, and are separated by predicates described by polynomial inequalities. Hence, a trajectory of an SNDS consists of a sequence of *segments*, where the behavior in each segment is given by the ODE of a mode.

As an example, consider the *Isolette* system[1] shown in Fig. 1. The Isolette system is used to maintain the temperature of the Isolette box, a physical environment, within a desired range that is beneficial to the infant. The continuous evolution of temperature c depends on the current status of the actuator. If the heater is on, the temperature will increase, otherwise it will decrease. Thus there exist two modes and the condition for switching between two modes depends on the current temperature in the box. The equation

[1] Copyright belongs to Dräger Ltd. https://www.draeger.com/en_uk/Products/isolette-8000-plus.

describing the above system is given by:

$$\dot{c} = -0.026 \cdot (c - q)\,,$$
$$\dot{q} = \begin{cases} -1 & \text{if } c \geq 98.5 \\ 1 & \text{if } c < 98.5 \end{cases}$$

where c is the temperature of the Isolette box, and q is the temperature of the heater. Here, the boundary separating the two modes is specified by the inequality $c \geq 98.5$, and both modes are described by ODEs where the derivative is a linear function of c and q.

There are many other real-life systems that can be viewed as switched dynamical systems. For example, electrocardiogram records of electrical signals in a person's heart, as analyzed in [GAG+00, NQF+19, BDG+20], and robot motion patterns in [MKY11, BDG+20].

In general, the problem of inferring a switched nonlinear dynamical system is to determine, from a given set of trajectories, the coefficients of the ODE describing each mode of the system, as well as the inequalities separating the modes.

In this paper, we propose two different methods for solving this problem. We begin by describing some common ingredients. First, we use Linear Multistep Method (LMM) [BG08] to estimate the derivative at each point of a discrete trajectory. Next, we optionally employ a segmentation algorithm to divide the trajectory into segments that are likely to lie in a single mode. Under the assumption that each segment lies in a single mode, with behavior described by an ODE with polynomial derivatives of a fixed degree, we estimate the coefficients of the polynomial using linear regression. This will also be an ingredient in our method for dealing with trajectories containing multiple modes. Once all points in the trajectories are classified into different modes, we use support vector machine (SVM) with polynomial kernels to determine the boundary between modes, under the assumption that the boundary can be described by polynomial inequalities.

Our first method, named InferByMerge, begins by segmenting the trajectory, then using linear regression to find the coefficients of the ODE for each segment, and finally using a clustering technique where the basic idea is to combine two segments whenever the linear fit is still good for the merged data. A pruning method is used to reduce search space. For the second method InferByPWA, we do not perform prior segmentation, but extend identification methods for piecewise affine models to estimate coefficients and classify modes at the same time.

Both methods are implemented, and extensive experiments are performed to determine and compare their performance. We first describe a metric for evaluating an inferred model in comparison to the original model, based on comparing the actual and predicted derivatives along trajectories of the system. We then design five switched dynamical systems, and consider different choices of parameters, and evaluate the two methods on a total of twenty cases. The results show that both methods perform well across a wide range of situations. We also compare our methods with the original identification methods [AS14] for piecewise affine models. It demonstrates that by considering nonlinearity with polynomial equations directly, our method gives a significant improvement over learning of purely piecewise linear models.

In summary, our contributions in this paper are as follows.

- A heuristic method to segment trajectories of a switched nonlinear dynamical system, so that each segment is likely to lie in a single mode.
- A model inference procedure based on segmenting the trajectories followed by a special clustering method with a pruning search.
- A model inference procedure by extending identification methods for piecewise affine models.
- A method for evaluating an inferred model of a switched nonlinear dynamical system in comparison to the original model.

Related work Learning models of a system from observable data has a long history. In 1956, Moore showed some Gedanken experiments on sequential machines and defined automaton learning [Vaa17] as black-box model inference of systems [Moo58]. In her seminal work [Ang87], Angluin presented an online, active and exact learning framework named L^* which is a query-answering process between a learner and a teacher with two kinds of queries: membership query and equivalence query. After that, there are many works on improving the L^* framework, including applications to learning different formal models such as Mealy machines [SG09], nondeterministic finite automata [BHKL09], Büchi automata [FCC+08, LCZL17], symbolic automata [MM14, DD17], Markov decision processes [TAB+19], and timed automata [ACZ+20, AWZ+ss].

For passive learning from a given sample set, RPNI [OG92] is a popular method and there are some works [VdWW11, VdWW12] on inferring timed automata following this method. In control theory, the problem was initially studied as system identification [Lju15]. The research on identification of linear dynamical systems started in the late 1950s, and a series of books established the field [Eyk74, Lju99]. There exist a lot of work focusing on discrete-time switched systems and piecewise affine models [FMLM03, GPV12, BGPV05, HV05]. In [BGPV05], Bemporad et al. aimed to fit the data to a piecewise affine autoregressive exogenous (ARX) model with error at most ϵ, while using as few submodels as possible. Ferrari-Trecate et al. gave a clustering algorithm (based on K-means over the estimated coefficients of the modes) to identify piecewise affine systems [FMLM03]. In [AS14], Alur et al. proposed a precise identification method for piecewise affine models from input-output data, which outperformed the method in [FMLM03]. One of our methods can be thought of as an extension of this work to nonlinear systems. Another way to perform clustering over estimated coefficients of the modes is a split-and-merge method with validation by the silhouette index [BIB11]. For identification of a nonlinear ODE from trajectories, in [KD19], Keller et al. used Linear Multistep Method to estimate the derivative at each data point. In [BPK16], Brunton et al. showed sparse regression methods can be used to estimate the coefficients of a nonlinear ODE.

ARX model and its variants are often used in data-driven system identification. In [LB08], Lauer et al. considered both switched and piecewise ARX models and their nonlinear versions. They applied a combination of linear programming and support vector regression to compute both the discrete state and the submodels in a single step. In the succeeding work [LBV10], they extended their methods to nonparametric identification using the kernel method in support vector machines. In [POTN03], RBF-ARX model based on Gaussian radial basis function networks and ARX structure is built to characterize the system. In [XPTP20], deep belief networks (DBN) based SD-ARX model is used for nonlinear system modeling. Since the implementations of the methods in [POTN03, XPTP20] are unavailable for us, we conducted experiments using inference methods of ARX models provided in Matlab. The results show that ARX models do not perform well on the trajectories generated from SNDS in our case studies.

As a popular kind of models of hybrid systems, Hybrid Automata [Hen96] have powerful expressiveness. Designing such models is a time-intensive process and inferring a hybrid automaton from data is thus an interesting problem [PJFV07, LBG18, NSV+12]. In [MRBF15], Medhat et al. described an abstract framework, based on heuristics, to learn linear hybrid automata from input/output traces. They followed Angluin's framework to learn a finite automaton as discrete structure of the hybrid automaton and then extracted linear ODEs from data. Recently, Henzinger et al. presented a membership-based identification method [SHSZ19] for a kind of linear hybrid automata where the differential equations are in a special constant form. Our model can be viewed as a special kind of hybrid automata in which each mode contains a polynomial ODE, and the transitions have polynomial inequalities as guards, and no variable resets and assignments. Hence, the models described in this paper are more expressive than the models in [AS14, SHSZ19]. Our experiments also show that our methods performs better than the method in [AS14].

Dividing a trajectory into segments likely to lie in one mode is one of the main challenges, where the correctness of the results strongly affect the quality of the inferred models. For piecewise affine systems, Borges et al. proposed a switch detection method based on the detection of rank variations for projected subspaces that are computed from successive batches of data [BVVB05]. For ARX models, Ohlsson et al proposed a method based on optimization, using a sum-of-norms regularization to control the number of changepoints in the result [OLB10]. Ozay proposed a polynomial time segmentation algorithm based on dynamic programming [Oza16]. There are also several existing algorithms [AL89] for this problem that are accurate for specific types of signals. The basic idea is to guess every point as a changepoint in turn, then design a value function to decide which points are the best changepoints. In general, a correct segmentation method may be impossible since two trajectories generated from two modes can connect with each other smoothly. Compared with existing methods, our heuristic segmentation method is faster and scales better in our case studies.

Organization of the paper Section 2 introduces the concept of Switched Nonlinear Dynamical Systems that we consider in this paper. Section 3 shows how to use Linear Multistep Methods to estimate derivatives along a trajectory. In Section 4, we present the heuristic segmentation method and the inference methods. We describe experiments comparing the methods in Section 5. Finally, Section 6 concludes this paper.

2. Switched Nonlinear Dynamical Systems

Let $\mathbb{R}$ be the set of real numbers and $\mathbb{N}$ be the set of natural numbers. Let $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ represent a point in $\mathbb{R}^n$. In this paper, the position of a switched nonlinear dynamical system lies in $\mathbb{R}^n$. Here, we give the definition of Switched Nonlinear Dynamical Systems (SNDS) we consider in this paper.

Definition 1 (Switched Nonlinear Dynamical System). An SNDS has a finite number of modes. In its different N modes, position evolves according to the following differential equation

$$\dot{\boldsymbol{x}}(t) = \begin{cases} f_1(\boldsymbol{x}(t)) & \text{if } G_1(\boldsymbol{x}) \geq 0 \\ f_2(\boldsymbol{x}(t)) & \text{else if } G_2(\boldsymbol{x}) \geq 0 \\ \vdots & \\ f_{N-1}(\boldsymbol{x}(t)) & \text{else if } G_{N-1}(\boldsymbol{x}) \geq 0 \\ f_N(\boldsymbol{x}(t)) & \text{otherwise} \end{cases},$$

where $\forall i \in \{1, 2, \cdots, N-1\}$. $G_i(\boldsymbol{x}) \geq 0$ is a polynomial inequality, and each mode function $f \in \{f_1, f_2, \cdots, f_N\}$ is in the form

$$f(\boldsymbol{x}) = \begin{pmatrix} \sum_{d_1,d_2,\cdots,d_n} a_1^{(d_1,d_2,\cdots,d_n)} x_1^{d_1} x_2^{d_2} \cdots x_n^{d_n} \\ \sum_{d_1,d_2,\cdots,d_n} a_2^{(d_1,d_2,\cdots,d_n)} x_1^{d_1} x_2^{d_2} \cdots x_n^{d_n} \\ \vdots \\ \sum_{d_1,d_2,\cdots,d_n} a_n^{(d_1,d_2,\cdots,d_n)} x_1^{d_1} x_2^{d_2} \cdots x_n^{d_n} \end{pmatrix}^T . \tag{1}$$

where $a_j^{(d_1,d_2,\cdots,d_n)} \in \mathbb{R}$ for each $1 \leq j \leq n$ and $d_1, d_2, \cdots, d_n \in \mathbb{N}$. Usually we will impose a constraint on the maximum order d of the polynomials, e.g. $d_1 + d_2 + \cdots + d_n \leq d$.

The definition thus means that the regions for the modes of a SNDS form a partition of $\mathbb{R}^n$ and the boundary between regions are assumed to be described by polynomial inequalities.

For example, the Isolette system given in the introduction is an SNDS in $\mathbb{R}^2$ with two modes, with ODEs described by polynomials of degree one, and boundary described by a polynomial inequality of degree one. The Lorenz attractor [Lor63] is a dynamical system, and can be thought of as an SNDS in $\mathbb{R}^3$ with one mode, and with ODE described by polynomials of degree two. As one of the test cases, we will consider a variant of the Lorenz attractor with the same dimension and polynomial degree, but with two modes (see Section 5.2).

A continuous trajectory of an SNDS in $\mathbb{R}^n$ is a continuous function $\boldsymbol{x} : [0, T] \to \mathbb{R}^n$, where there exist times $0 = T_0 < T_1 < \cdots < T_n = T$ such that on each interval (T_i, T_{i+1}) for $0 \leq i < n$, the path $\boldsymbol{x}(t)$ for $t \in (T_i, T_{i+1})$ lies entirely in mode G_k for some $k \leq N$, and satisfies the corresponding ODE $\dot{\boldsymbol{x}}(t) = f_k(\boldsymbol{x}(t))$. In reality, we can only collect data at discrete time points. Hence, we define a *(discrete) trajectory* of an SNDS to be the values of a continuous trajectory at a finite set of time points $0 \leq t_0 < \cdots < t_m \leq T$. For simplicity, we will focus on the case where the sample points are taken periodically, i.e. $t_i = i \cdot t_{step}$, where t_{step} is a fixed *timestep size*, i.e., the period. Note that it is by no means guaranteed that we will obtain samples at the switching times T_i. Indeed we do not have explicit information about the switching times in the continuous or discrete trajectory. The *inference problem* for SNDS can now be described as follows.

Problem: Given a set of discrete trajectories of an SNDS in $\mathbb{R}^n$, with known upper bounds on degrees of polynomial describing the ODEs and the boundary, estimate the coefficients of the ODE in each mode as well as of the polynomial inequalities separating the modes.

In applications, we would like the inferred model to be useful not only along the known trajectories, but also along other potential trajectories of the system. Therefore, in evaluating an inferred model, we focus on its predictive power along both known trajectories and other potential trajectories. This will be evident in the experiments in Section 5.

3. Linear Multistep Methods

For all our methods, the first step is to estimate the derivative at each point of a discrete trajectory. For this, we apply Linear Multistep Methods (LMMs). In this section, we briefly review the original motivation of these methods and show how they are applied to our problem.

Linear multistep methods are originally designed for solving ordinary differential equations numerically, i.e. solving the initial value problems given the dynamics:

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t)), \quad a \leq t \leq b, \quad \boldsymbol{x}(t_0) = x_0 \tag{2}$$

An M-step multistep method uses states at the previous M time steps to approximate the next state. Here M is called the step number of the method. The common form is

$$\sum_{m=0}^{M} \alpha_m \boldsymbol{x}(t_{n-m}) \approx h \sum_{m=0}^{M} \beta_m f(\boldsymbol{x}(t_{n-m})). \tag{3}$$

Here the coefficients α_m and β_m are chosen depending on the different choice of LMM, with $\alpha_0 \neq 0$, and h is the step size. This expresses an equality between a linear combination of $\boldsymbol{x}(t_{n-M}), \ldots, \boldsymbol{x}(t_n)$ and a linear combination of $f(\boldsymbol{x}(t_{n-M})), \ldots, f(\boldsymbol{x}(t_n))$. If $\beta_0 = 0$ this gives an equation where $\boldsymbol{x}(t_n)$ appears only once, and hence can be solved immediately, corresponding to the explicit form of LMM. If $\beta_0 \neq 0$, then $\boldsymbol{x}(t_n)$ occurs multiple times, corresponding to the implicit form of LMM. In this case $\boldsymbol{x}(t_n)$ can be solved by an iterative approach.

The truncation error of LMM is defined as:

$$T_n = \sum_{m=0}^{M} \alpha_m \boldsymbol{x}(t_{n-m}) - h \sum_{m=0}^{M} \beta_m f(\boldsymbol{x}(t_{n-m})) \tag{4}$$

Assume $\boldsymbol{x}$ and f are smooth functions, after performing Taylor expansion at $\boldsymbol{x}(t_n)$, we have

$$T_n = \sum_{m=0}^{\infty} C_m h^m \nabla_{\boldsymbol{x}}^m \boldsymbol{x}(t_n) \tag{5}$$

where

$$C_0 = \sum_{m=1}^{M} \alpha_m, \tag{6}$$

$$C_m = (-1)^m \left[\frac{1}{m!} \sum_{k=0}^{M} k^m \alpha_k + \frac{1}{(m-1)!} \sum_{k=0}^{M} k^{m-1} \beta_k \right] \tag{7}$$

Hence, if the coefficients α_m and β_m are chosen so that $C_m = 0$ for all $m \leq p$, then

$$T_n = C_{p+1} h^{p+1} \nabla_{\boldsymbol{x}}^{p+1} \boldsymbol{x}(t_n) + O(h^{p+2}) \tag{8}$$

and we say the linear multistep method has error of order p.

In this paper, our goal is a bit different: we know all positions $\boldsymbol{x}(t_i)$ but not the ODE, and hence need to estimate the derivatives $\dot{\boldsymbol{x}}(t_i) = f(\boldsymbol{x}(t_i))$, then use them to estimate the coefficients in f. First, we state the problem more precisely.

Problem: Given a discrete trajectory (as defined in Section 2) as a list of pairs $(t_i, \boldsymbol{x}_i)$, taken from a continuous trajectory obeying a single ODE $\dot{\boldsymbol{x}} = f(x)$. Assume the time points are taken periodically, i.e., $t_i = i \cdot t_{step}$, where t_{step} is the period. Estimate the derivative $\dot{\boldsymbol{x}}(t_i) = f(\boldsymbol{x}_i)$ at every point from the pairs $(t_i, \boldsymbol{x}_i)$.

For this problem, we use a specific form of LMM called the Backwards Differentiation Formula (BDF). The corresponding coefficients α_m and β_m for BDF can be obtained as follows. First, obtain the Lagrange interpolating polynomial of the points $\boldsymbol{x}(t_{n-M}), \ldots, \boldsymbol{x}(t_n)$, this polynomial gives an approximation to the

continuous solution $\boldsymbol{x}(t)$:

$$\boldsymbol{x}(t) \approx \sum_{m=0}^{M} \boldsymbol{x}(t_{n-m}) \prod_{i \neq m} \frac{t - t_{n-m}}{t_{n-i} - t_{n-m}} \tag{9}$$

Then, take the derivative of both sides of Equation (9) with respect to t, we get:

$$f(\boldsymbol{x}(t_n)) \approx \sum_{m=0}^{M} \boldsymbol{x}(t_{n-m}) \cdot \frac{d}{dt} \prod_{i \neq m} \frac{t - t_{n-m}}{t_{n-i} - t_{n-m}} \Bigg|_{t_n} \tag{10}$$

The right side is a linear combination of $\boldsymbol{x}(t_{n-M}), \ldots, \boldsymbol{x}(t_n)$ with constant coefficients. Choose $M = 5$, we get:

$$f(\boldsymbol{x}(t_n)) \approx \frac{1}{h}\Big(\frac{137}{60}\boldsymbol{x}(t_n) - \frac{300}{60}\boldsymbol{x}(t_{n-1}) + \frac{300}{60}\boldsymbol{x}(t_{n-2}) - \frac{200}{60}\boldsymbol{x}(t_{n-3}) + \frac{75}{60}\boldsymbol{x}(t_{n-4}) - \frac{12}{60}\boldsymbol{x}(t_{n-5})\Big)$$

Define $\bar{f}(\boldsymbol{x}(t_n))$ to be the right side of this equation. We will use this as the estimated derivative in our paper.

The above method works under the assumption that the trajectory obeys a single ODE with smooth derivatives. In our case, the trajectory data may cross multiple modes. If there is a mode change between the points $\boldsymbol{x}(t_{n-5})$ to $\boldsymbol{x}(t_n)$, then the estimate using BDF will not be accurate. Hence, as an alternative and for comparison in the algorithm, we will also use a forward version of BDF, computing the estimates using the points $\boldsymbol{x}(t_n)$ to $\boldsymbol{x}(t_{n+5})$. The formula is:

$$f(\boldsymbol{x}(t_n)) \approx \frac{1}{h}\Big(-\frac{137}{60}\boldsymbol{x}(t_n) + \frac{300}{60}\boldsymbol{x}(t_{n+1}) - \frac{300}{60}\boldsymbol{x}(t_{n+2}) + \frac{200}{60}\boldsymbol{x}(t_{n+3}) - \frac{75}{60}\boldsymbol{x}(t_{n+4}) + \frac{12}{60}\boldsymbol{x}(t_{n+5})\Big)$$

then we have $C_0 = C_1 \cdots = C_5 = 0$ and $C_6 = -\frac{1}{6}$

Later in the algorithms, we will use LMM_b to denote the backward BDF, and LMM_f to denote the forward version.

4. Inferring SNDS from Trajectories

In this section, we describe the two proposed algorithms in detail. The algorithms involve many existing ideas in system identification, including the use of linear regression to infer coefficients, segmentation, clustering, and identification techniques for piecewise affine models. However, we also introduce new methods, in particular for the segmentation and clustering steps.

4.1. Inferring ODE for a single mode

We first describe the technique for estimating the coefficients of the ODE, based on a (discrete) trajectory which is assumed to lie in one mode. We assume the derivatives in the ODE are polynomials of order at most d.

First, we define a function Φ from $\mathbb{R}^n$ to $\mathbb{R}^{\binom{n+d}{d}}$, mapping each point $\boldsymbol{x} = (x_1, \ldots, x_n)$ to a vector whose coordinates are values of all monomials in x_i of order at most d, arranged in a pre-defined order.

With the function Φ, the equation $\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t))$ can be rewritten as:

$$\dot{x_i}(t) = \Phi(\boldsymbol{x}(t)) \cdot F_i \tag{11}$$

where $x_i(t)$ is the ith component of $\boldsymbol{x}(t)$, and each F_i for $1 \leq i \leq n$ is a vector consisting of the coefficients of f in the ODE. The overall coefficient matrix F has F_i as columns. In our case, we know the values of $\Phi(\boldsymbol{x}(t))$ and estimates of $\dot{x_i}(t)$ from the LMM, and wish to estimate the coefficients F_i.

We will make the simplifying approximation that the error introduced by LMM is randomly and independently distributed around 0. Under this approximation, and given enough sample points $\boldsymbol{x}(t)$ and $\dot{x_i}(t)$, the coefficients in F_i can be estimated by linear regression. In [KD19], it is proved that as the timestep size approaches zero, the error introduced by BDF converges to zero for every choice of step number M. Hence,

we expect that the estimate of coefficients given by linear regression becomes better as the timestep size decreases. The method of inferring the coefficients of a nonlinear ODE using regression is studied in depth in [BPK16].

More precisely, the algorithm for linear regression is as follows. Given sample points $\boldsymbol{x}(t_1), \ldots, \boldsymbol{x}(t_m)$ along a trajectory y and estimated derivatives $\bar{f}(\boldsymbol{x}(t_1)), \ldots, \bar{f}(\boldsymbol{x}(t_m))$, let $A_y = [\cdots, \Phi(\boldsymbol{x}(t_j))^T, \cdots]^T$ be the matrix whose rows are the vectors $\Phi(\boldsymbol{x}(t_j))$, and $B_y = [\cdots, \bar{f}(\boldsymbol{x}(t_j))^T, \cdots]^T$ be the matrix whose rows are the estimates $\bar{f}(\boldsymbol{x}(t_j))$ obtained using BDF. Then, the linear regression problem is to find matrix F giving the best approximation to the equation $A \cdot F = B$. The solution using Moore-Penrose pseudoinverse [Pen55] is $F = (A^T A)^{-1} A^T B$. We call this procedure *LinearRegression*(A, B). And we record $A_y \cdot F = [\cdots, \tilde{f}(\boldsymbol{x}(t_j))^T, \cdots]^T$. We assume that on this closed trajectory $|\nabla^6_{\boldsymbol{x}} \boldsymbol{x}(t_j)|$ has a upper bound of D, then we have

$$|f(\boldsymbol{x}(t_j)) - \bar{f}(\boldsymbol{x}(t_j))| \leq \left| \frac{1}{6} h^5 \cdot D \right| + O(h^6) \tag{12}$$

Along a trajectory, the sum of squared error between $f(\boldsymbol{x})$ and $\bar{f}(\boldsymbol{x})$ is:

$$\sum_{\boldsymbol{x}(t_j) \in y} |f(\boldsymbol{x}(t_j)) - \bar{f}(\boldsymbol{x}(t_j))|^2 \leq \frac{T}{h} \cdot \left(\frac{1}{36} h^{10} \cdot |D|^2 + O(h^{11}) \right) = \frac{T}{36} h^9 \cdot |D|^2 + O(h^{10}) \tag{13}$$

Since the true values $f(\boldsymbol{x})$ of the derivative lie on a plane, there exists a linear fit with sum of squared error bounded above. Since linear regression minimizes the sum of squared error, the result of linear regression should also have error bounded above. This gives the sum of squared error between $\tilde{f}(\boldsymbol{x})$ and $\bar{f}(\boldsymbol{x})$ as follows:

$$\sum_{\boldsymbol{x}(t_j) \in y} |\tilde{f}(\boldsymbol{x}(t_j)) - \bar{f}(\boldsymbol{x}(t_j))|^2 \leq \frac{T}{36} h^9 \cdot |D|^2 + O(h^{10}) \tag{14}$$

and the overall sum of squared error between estimate from linear regression and true derivative is:

$$\sum_{\boldsymbol{x}(t_j) \in y} |\tilde{f}(\boldsymbol{x}(t_j)) - f(\boldsymbol{x}(t_j))|^2 \leq \frac{T}{18} h^9 \cdot |D|^2 + O(h^{10}) \tag{15}$$

Hence, we obtain a bound on the sum of squared error between the original derivatives and the derivatives we estimate.

Next, we consider how to estimate the upper bound D of $|\nabla^6_{\boldsymbol{x}} \boldsymbol{x}(t_j)|$ in terms of an estimate of the maximum coefficient in the SNDS. Let A_{max} be the maximum of coefficients $a_j^{(d_1, d_2, \cdots, d_n)}$ in Equation (1), and X_{max} be a bound on $|x_i|$ along the trajectory. For simplicity of calculation, we assume $X_{max} \geq 1$. This can be ensured by scaling the coordinates if necessary. Let d and n be the degree and dimension of the SNDS as before. Note that while X_{max} can be computed from the provided trajectory, A_{max} can only be estimated from a priori knowledge about the system.

Let D_k be a bound on $|\nabla^k_{\boldsymbol{x}}(\boldsymbol{x}(t_j))|$, then we have the recurrence relation:

$$D_k \leq A_{max} \cdot \binom{d}{n} \cdot (d \cdot X_{max}^{d-1} \cdot D_{k-1}) + \cdots \tag{16}$$

The leading term corresponds to the terms of $f(\boldsymbol{x})$ with degree d, and the remaining terms correspond to terms in $f(\boldsymbol{x})$ with degree less than d. The base value is $D_0 \leq X_{max}$. Solving the recurrence relation, we get the estimate

$$D_k \leq A_{max}^k \cdot \binom{d}{n}^k \cdot d^k \cdot X_{max}^{k(d-1)+1} + \cdots \tag{17}$$

In particular, suppose we scale the coordinates so that $X_{max} \approx 1$, then the bound on D is in the order of $(A_{max} \cdot \binom{d}{n} \cdot d)^6$. This bound is a conservative one: if only few of the $\binom{d}{n}$ terms of degree d in $f(\boldsymbol{x})$ are nonzero, D_n would be much less than this estimate.

In addition to giving an idea about the error in the estimated derivative, this bound on D also provides guidance on setting the parameter δ on the allowable absolute error in the following methods (if one is able

Algorithm 1: Segmenting

```
// Heuristic segmentation procedure
```

Input: a single trajectory $y = (\boldsymbol{x}(t_1), \ldots, \boldsymbol{x}(t_n))$, and relative error tolerance ϵ.
Output: a list of trajectory segments.

$CP \leftarrow \emptyset$;
foreach $\boldsymbol{x}(t_i) \in y$ **do**
  if $i < M$ *or* $i > n - M$ **then**
    continue;
  $b_1 \leftarrow LMM_b(\boldsymbol{x}(t_{i-M}), \ldots, \boldsymbol{x}(t_i))$;
  $b_2 \leftarrow LMM_f(\boldsymbol{x}(t_i), \ldots, \boldsymbol{x}(t_{i+M}))$;
  if $d(b_1, b_2) > \epsilon$ **then**
    add t_i to CP;
segs $\leftarrow$ consecutive intervals of $y \backslash CP$;
return *segs*;

to obtain an estimate on the size of coefficients of the SNDS). In particular, we conclude that in the noise-free case, LMM is expected to provide good estimates of the derivative if $h \cdot A_{max} \ll 1$.

4.2. Segmenting the trajectory

The above method works under the assumption that the given trajectory lies in a single mode of the SNDS. However, we do not have explicit information about where the mode changes in the trajectory. The first method that we propose begins by segmenting the given trajectory. Unlike most of the existing methods, it has linear complexity in the length of the given trajectory, and hence is applicable to large data sets.

Our method is based on the intuitive idea of detecting sudden changes in the derivative along a trajectory. First, we define a way to measure the relative difference between two vectors, that will be used throughout the paper.

Definition 2 (Relative difference). Given two vectors $\boldsymbol{v}$ and $\boldsymbol{w}$, their relative difference is defined as:

$$d(\boldsymbol{v}, \boldsymbol{w}) = \frac{\|\boldsymbol{v} - \boldsymbol{w}\|}{\|\boldsymbol{v}\| + \|\boldsymbol{w}\|} \tag{18}$$

where $\|\boldsymbol{v}\|$ is the norm of $\boldsymbol{v}$.

This measure has the advantage that it is rotationally invariant, and does not change on the simultaneous scaling of the input vectors.

Let M be the step number of BDF as mentioned in Section 3. For each point on the trajectory, except the $M - 1$ points at the beginning and at the end, we estimate the derivative twice: the one using the M preceding points and the other using the M subsequent points. Denote the two estimated derivatives by b_1 and b_2, respectively. If a point is within M steps of a change point, then the computation of at least one of b_1 and b_2 will involve points in a different mode. This means b_1 and b_2 are likely to have a bigger relative difference between them compared to at other points. Hence, we compute the relative difference between b_1 and b_2 using Equation (18), and consider the point as near a boundary if the difference is larger than some threshold ϵ (for our experiments, we set ϵ to 0.01). This naturally divides the trajectory into disjoint segments, formed by those points where the computed b_1 and b_2 are close enough with the threshold ϵ. Note that a boundary usually contains several points in a row. A larger threshold ϵ may miss some changepoints and a smaller threshold may cause redundant changepoints. Thus different classes of systems may require different choices of the threshold, and we may need to tune it to adapt to the specific case at hand. The full procedure is given in Algorithm 1.

Algorithm 2: InferByMerge

// INFERBYMERGE: Clustering by merging

Input: collection of sets of points $S = \{S_1, \ldots, S_k\}$, δ is the absolute error tolerance for each point, $nmode$ is the number of modes.

Output: list of classes and coefficient matrices.

$Pruned \leftarrow \emptyset$;
$error_{max} = (|S_1| + |S_2| + \cdots + |S_k|) \cdot \delta$; // The maximal sum of square error tolerated in every cluster.
foreach $S_i = \{\boldsymbol{x}_{i1}, \ldots, \boldsymbol{x}_{im}\}, S_j = \{\boldsymbol{x}_{j1}, \ldots, \boldsymbol{x}_{jn}\}$ *with* $1 \leq i < j \leq k$ **do**
  $A \leftarrow \left[\Phi(\boldsymbol{x}_{i1})^T, \ldots, \Phi(\boldsymbol{x}_{im})^T, \Phi(\boldsymbol{x}_{j1})^T, \ldots, \Phi(\boldsymbol{x}_{jn})^T\right]^T$;
  $B \leftarrow \left[\bar{f}(\boldsymbol{x}_{i1})^T, \ldots, \bar{f}(\boldsymbol{x}_{im})^T, \bar{f}(\boldsymbol{x}_{j1})^T, \ldots, \bar{f}(\boldsymbol{x}_{jn})^T\right]^T$;
  $F \leftarrow LinearRegression(A, B)$;
  $error_{sos} \leftarrow \|A \cdot F - B\|^2_{Frob}$; // $\|\cdot\|_{Frob}$: the Frobenius Norm of matrix.
  if $error_{sos} > error_{max}$ **then**
    $Pruned \leftarrow Pruned \cup \{S_i \cup S_j\}$;
$\mathcal{S}set \leftarrow PruningSearch(S, nmode, Pruned, error_{max})$;
foreach $\mathcal{S} \in \mathcal{S}set$ **do**
  $error_{\mathcal{S}} \leftarrow 0$;
  foreach $\mathcal{S}_j = \{\boldsymbol{x}_{j1}, \ldots, \boldsymbol{x}_{jn}\} \in \mathcal{S}$ **do**
    $A_j \leftarrow \left[\Phi(\boldsymbol{x}_{j1})^T, \ldots, \Phi(\boldsymbol{x}_{jn})^T\right]^T$;
    $B_j \leftarrow \left[\bar{f}(\boldsymbol{x}_{j1})^T, \ldots, \bar{f}(\boldsymbol{x}_{jn})^T\right]^T$;
    $F_j \leftarrow LinearRegression(A_j, B_j)$;
    $error_{\mathcal{S}} \leftarrow error_{\mathcal{S}} + \|A \cdot F - B\|^2_{Frob}$;
choose $\mathcal{S} = \{\mathcal{S}_1, \cdots, \mathcal{S}_{nmode}\} \in \mathcal{S}set$ with least $error_{\mathcal{S}}$;
return $\{\mathcal{S}_1, \cdots, \mathcal{S}_{nmode}\}$, $\{F_1, \cdots, F_{nmode}\}$;

4.3. Clustering by merging

After segmenting the trajectory, we can assume that each segment lies in one mode, so we can estimate the coefficients in that mode using linear regression as in Section 4.1. The next problem is to determine which segments lie in the same mode. This can be considered as a clustering problem. One possible approach is to use one of the existing clustering algorithms such as DBSCAN and k-means, where each segment is represented by the vector consisting of the estimated coefficients. For example, k-means is used in [BDG+20] as the clustering method on the coefficients of the learned basic shape patterns. Their experimental results show that the basic clustering method does not always perform well but they do not analyze the reasons.

In our work, we observed that a significant weakness of traditional clustering methods is that ODEs with very different coefficients may describe similar behavior within a localized region. Hence, linear regression based on points within a small region (such as within a segment of the trajectory) may yield very different coefficients, even if they actually lie in the same mode. The difference between coefficients of segments in the same mode may therefore be bigger than the difference between coefficients in different modes, and may lead traditional clustering methods to produce incorrect clusters. Hence, directly clustering the vectors of coefficients is not suitable. In our experimental results, we also show that DBSCAN performs poorly in some situations. In the following, we propose a different clustering method that is better suited for our problem.

We now describe a clustering method where segments are merged if their combined data fits well in a single linear regression of Φ and $\bar{f}$ (rather than if they have similar coefficients). The basic idea is to find a way to cluster all segments into N modes such that the total sum of squared error of all linear regressions are minimized. A naive search over all combinations of clusters will yield an exponential number of cases. In order to reduce the number of cases and accelerate the approach, we propose a pruning method during the search process. The full method named INFERBYMERGE is given in Algorithm 2 and the pruning method is presented in Algorithm 3.

We consider each segment as a set of points (to make it also useful for the method INFERBYPWA in Sec-

Algorithm 3: PruningSearch

// The pruning search approach used in InferByMerge.

Input: collection of sets of points $S = \{S_1, \ldots, S_k\}$, *Pruned* is the set to be pruned, *nmode* is the number of cluster to classify, $error_{max}$ is the maximal sum of square error tolerated in every cluster.

Output: list of classes

1 **if** $|S| = nmode$ **then**
2 **return** S;
3 **if** $nmode = 1$ **then**
4 **if** $\exists pr \in Pruned.\ pr \subseteq \bigcup_{S_i \in S} S_i$ **then**
5 **return** $\emptyset$;
6 assume $\bigcup_{S_i \in S} S_i = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$;
7 $A \leftarrow \left[\Phi(\boldsymbol{x}_1)^T, \ldots, \Phi(\boldsymbol{x}_n)^T\right]^T$;
8 $B \leftarrow \left[\bar{f}(\boldsymbol{x}_1)^T, \ldots, \bar{f}(\boldsymbol{x}_n)^T\right]^T$;
9 $F \leftarrow LinearRegression(A, B)$;
10 $error_{\bigcup_{S_i \in S} S_i} \leftarrow \|A \cdot F - B\|^2_{Frob}$;
11 **if** $error_{\bigcup_{S_i \in S} S_i} \leq error_{max}$ **then**
12 **return** $\{\{\bigcup_{S_i \in S} S_i\}\}$;
13 $\mathcal{S}set \leftarrow \emptyset$;
14 **foreach** $\mathcal{S} \in PruningSearch(\{S_1, \cdots, S_{k-1}\}, Pr, nmode, error_{max})$ **do**
15 **foreach** $\mathcal{S}_j \in \mathcal{S}$ **do**
16 $\mathcal{S}' \leftarrow \mathcal{S}$;
17 assume $\mathcal{S}_j \cup S_k = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$;
18 $A \leftarrow \left[\Phi(\boldsymbol{x}_1)^T, \ldots, \Phi(\boldsymbol{x}_n)^T\right]^T$;
19 $B \leftarrow \left[\bar{f}(\boldsymbol{x}_1)^T, \ldots, \bar{f}(\boldsymbol{x}_n)^T\right]^T$;
20 $F \leftarrow LinearRegression(A, B)$;
21 $error_{\mathcal{S}_j \cup S_k} \leftarrow \|A \cdot F - B\|^2_{Frob}$;
22 **if** $\forall pr \in Pruned.\ pr \nsubseteq \mathcal{S}_j \cup S_k$ *and* $error_{\mathcal{S}_j \cup S_k} \leq error_{max}$ **then**
23 $\mathcal{S}set \leftarrow \mathcal{S}set \cup \{(\mathcal{S}' \setminus \mathcal{S}_j) \cup \{\mathcal{S}_j \cup S_k\}\}$;
24 **foreach** $\mathcal{S} \in PruningSearch(\{S_1, \cdots, S_{k-1}\}, Pruned, nmode - 1, error_{max})$ **do**
25 $\mathcal{S}set \leftarrow \mathcal{S}set \cup \{\mathcal{S} \cup \{S_k\}\}$;
26 **return** $\mathcal{S}set$;

tion 4.4). Hence, the input to the algorithm is a collection S of sets of points. First, we consider all pairs of sets in the collection. For each pair of sets $S_i = \{\boldsymbol{x}_{i1}, \ldots, \boldsymbol{x}_{im}\}$ and $S_j = \{\boldsymbol{x}_{j1}, \ldots, \boldsymbol{x}_{jm}\}$, we perform linear regression using their combined data. Precisely, form matrix $A = \left[\Phi(\boldsymbol{x}_{i1})^T, \ldots, \Phi(\boldsymbol{x}_{im})^T, \Phi(\boldsymbol{x}_{j1})^T, \ldots, \Phi(\boldsymbol{x}_{jn})^T\right]^T$ containing the values of monomials of position, and $B = \left[\bar{f}(\boldsymbol{x}_{i1})^T, \ldots, \bar{f}(\boldsymbol{x}_{im})^T, \bar{f}(\boldsymbol{x}_{j1})^T, \ldots, \bar{f}(\boldsymbol{x}_{jn})^T\right]^T$ containing the estimated derivatives. Then linear regression is performed to find a new matrix of coefficients F, and compute the sum of squared error $error_{sos}$ between the predicted values $A \cdot F$ and original values B. If $error_{sos}$ is greater than $error_{max} = (|S_1| + |S_2| + \cdots + |S_k|) \cdot \delta$ which is the maximal sum of squared error tolerated in every cluster, we can conclude that the two segments should not occur in the same cluster. Here δ represents the absolute error tolerance for each point (i.e., Mean Squared Error tolerance).

Next, we enumerate the clustering combinations that remain after the above pruning, by calling the procedure *PruningSearch* on line 10, to be described later. After *PruningSearch*, we obtain a set of clustering combinations $\mathcal{S}set$. For each combination $\mathcal{S}$, we compute the total sum of squared error $error_{\mathcal{S}}$ over all linear

regressions. Finally, the algorithm returns the combination with the least $error_S$ and the corresponding coefficient matrices.

Now we describe the detail of *PruningSearch*. The main idea is to enumerate all combinations, avoiding each pair in the set of pruned pairs *Pruned*, and also pruning the search when a cluster has error above $error_{max}$. The algorithm proceeds by iteratively adding each of the k segments in one of the *nmode* classes. If there is only one mode remaining, then the remaining segments must form a class. Otherwise, suppose the previous $k-1$ segments has been classified into *nmode* models, we can add the k'th segment into each class, or make the k'th segment form a new class. In the former case, we check that by extending a class with the k'th segment, we do not encounter any pruned pairs, nor do the sum of squared error exceed $error_{max}$. If that is the case, this branch of the search is pruned.

Theorem 1. If there exists a clustering combination such that the total sum of squared error is bounded by $error_{max}$, then the clustering combination returned by Algorithm 2 is the one with the smallest total sum of squared error.

Proof. From the setting of $error_{max}$, if any partial cluster has total sum of squared error greater than $error_{max}$, then any cluster combination containing it must also have total sum of squared error greater than $error_{max}$. By assumption, this cannot be a combination with least total sum of squared error, and hence can be safely pruned from the search. □

Corollary 1.1. If the trajectory has been correctly segmented then for all points in the input of Algorithm 2, the sum of squared error between the original and predict derivatives is bounded as follows:

$$\sum_{\boldsymbol{x}(t_j)\in\bigcup_{S_i\in S} S_i} |\tilde{f}(\boldsymbol{x}(t_j)) - f(\boldsymbol{x}(t_j))|^2 \leq \frac{1}{18}h^9 \cdot (\sum_{l=1}^{nmode} T_l \cdot |D_l|^2) + O(h^{10}) \tag{19}$$

where T_l represents the time stayed in mode l and D_l represents the maximum of $|\nabla^6_{\boldsymbol{x}}\boldsymbol{x}(t)|$ with $t \in T_l$.

Proof. Follows from Theorem 1 and Equation (15). □

The previous two algorithms depend on the performance of the segmentation process. If some change-points escape detection, the subsequent clustering or merging algorithm will not produce good results. Hence, we next describe a method that does not rely on a prior segmentation.

4.4. Extending identification of piecewise affine models

After projecting the positions from $\mathbb{R}^n$ to $\mathbb{R}^{\binom{n+d}{d}}$ using the mapping Φ, the relationship between position and derivative in each mode can be described by an affine function (see Equation (11)). Hence, the overall relation between position and derivative in all modes can be described by a piecewise affine function. This means we can apply any technique for identification of piecewise affine models to our problem. More specifically, we will apply a variant of the technique described in [AS14]. This gives a method for identification of SNDSs without prior segmentation, instead classifying the points and estimating the coefficients at the same time.

In [AS14], an absolute error tolerance for each point needs to be given before the identification process, but there is no description on how to determine such an error tolerance. Hence, we first modify the original algorithm to use a relative error tolerance. The second modification is that we extend a point to a small piece along the given trajectory (in other words, along time), while the original method extends a point to nearby points according to their positions. The benefit is that the points in a small continuous time interval have a bigger chance to be in the same mode than the points nearby. Third, the points on the boundary between modes are likely to have poorly estimated derivatives. If these points are taken into account, they may have a large effect on the resulting model. Hence, we selectively discard the points that do not fit well, and consider them only afterwards. Finally, we use Algorithm 2 to reduce the number of classes and simplify the resulting model. Details of our variant of the procedure we call InferByPWA is shown in Algorithm 4, and explained below.

Let S be the union of points in the trajectories of Y. We maintain a collection S_L containing the currently found classes of points.

We begin by choosing a point y from S. First, we attempt to extend y into a sequence *seq*. Starting

Algorithm 4: InferByPWA

```
// INFERBYPWA: Extension of identification of piecewise affine models
```

Input: the set of trajectories Y, ϵ is the error tolerance, k is the number of classes.
Output: list of classes and coefficient matrices.

$S_L \leftarrow \emptyset$;
$Drop \leftarrow \emptyset$;
$S \leftarrow$ set of points in Y;
while $\bigcup_{Y_i \in Y_L} Y_i \neq S$ **do**
  choose a point $y \in S \backslash \bigcup_{Y_i \in Y_L} Y_i$;
  $Y_y \leftarrow$ trajectory containing y;
  $seq \leftarrow Extending(Y_y, y, \epsilon)$;
  if $|seq| < M$ **then**
    add y to $Drop$;
    continue;
  $L \leftarrow seq$;
  while $\top$ **do**
    $F \leftarrow LinearRegression(A_L, B_L)$;
    $L' \leftarrow \emptyset$;
    for $\widetilde{y} \in Y \backslash \bigcup_{Y_i \in Y_L} Y_i$ **do**
      if $d(\Phi(\widetilde{y}) \cdot F, \bar{f}(\widetilde{y})) < \epsilon$ **then**
        add $\widetilde{y}$ to L';
    if $|L| < |L'|$ **then**
      $L \leftarrow L'$;
    else
      break;
  add L to S_L;
$\{S_1, \cdots, S_k\}, \{F_1, \cdots, F_k\} \leftarrow InferByMerge(S_L, \epsilon, k)$; // Algorithm 2 to reduce number of classes.
return $\{S_1, \cdots, S_k\}, \{F_1, \cdots, F_k\}$;

from the sequence only containing y, we try to extend it to the left and to the right, along the trajectory containing y. This is described separately as procedure *Extending* (Algorithm 5). First, we try to extend the sequence to the left, at each step recompute the linear regression using points of the new sequence and test whether the error is within tolerance ϵ. Extension stops when the error grows larger than ϵ, or the length reaches a given limit *lim*. Next, the sequence is extended to the right in the same way. If at the end of the extension process, the number of points in the sequence is less than the step number M, we put the point y into a separate set and restart since this indicates that the point is likely to be near a switch between modes.

Next, we attempt to extend the sequence *seq* into a larger set by alternatingly search for unclassified points that lie close to the linear regression F over $\Phi(\boldsymbol{x})$ and $\bar{f}(\boldsymbol{x})$, and recomputing the linear regression using the extended set of points, until the size of the set can no longer be enlarged. Finally, we add the extended set of points into S_L, and begin the next iteration as long as there exist uncovered points.

After all points have been handled, we obtain a collection of classes S_L. This collection of classes may still break a mode into several classes. Hence, we again perform the Algorithm 2 to reduce the number of classes and simplify the resulting model.

4.5. Boundary determination using SVM

After preliminarily classifying the points on the trajectory into different classes, and estimating the coefficients of ODEs for each class, we determine the boundary between classes using SVM with polynomial kernel functions. This assumes that the boundaries are given by polynomial inequalities of known maximum degree.

Algorithm 5: Extending

```
// Extending a segment
```

Input: a trajectory $y = (\boldsymbol{x}(t_1), \ldots, \boldsymbol{x}(t_n))$, point $\boldsymbol{x}(t_i)$ with $1 \le i \le n$, error tolerance ϵ, and maximum number of iterations lim.

Output: a segment containing $\boldsymbol{x}(t_i)$

$l \leftarrow i, h \leftarrow i$;
while $h - l < lim$ **do**
  $F \leftarrow LinearRegression(A_{[l-1:h]}, B_{[l-1:h]})$;
  if $\forall r.\ d(A_{[l-1:h]}[r] \cdot F, B_{[l-1:h]}[r]) < \epsilon$ **then**
    $l \leftarrow l - 1$;
  else
    break;
while $h - l < lim$ **do**
  $F \leftarrow LinearRegression(A_{[l:h+1]}, B_{[l:h+1]})$;
  if $\forall r.\ d(A_{[l:h+1]}[r] \cdot F, B_{[l:h+1]}[r]) < \epsilon$ **then**
    $h \leftarrow h + 1$;
  else
    break;
return $(\boldsymbol{x}(t_l), \ldots, \boldsymbol{x}(t_h))$;

Note that during the classification reported in the above sections, we dropped some points which are likely to be near the change points. There is a great possibility that these points are significant support vectors as they lie near the boundary between two modes. Hence, we would prefer to find out which modes these points belong to. We again compute the approximate derivatives using BDF in both forward and backward directions (using LMM_f and LMM_b), then test if either estimated derivative lies close to one of the modes according to its estimated coefficients.

Applying SVM with more than two classes is a well-studied problem, and we choose the following known approach: first learn the boundary between one class and the other classes, obtaining the first region G_1, and then among the remaining classes, select one again learning the next region $G_1^C \cap G_2$, repeating this process until all classes are separated. We refer to [BGPV05] for more discussion of the approaches. The main difference is that we use SVM with polynomial kernel function. From Theorem 1, we have the following corollary.

Corollary 1.2. In InferByMerge, the result of Equation (19) continues to hold after the SVM step, if the labelled points from different modes are classified by SVM with 100 percent accuracy.

4.6. Complexity of the methods

Most parts of our methods have polynomial complexity in terms of the number of input points n. In particular, segmentation has complexity $O(n)$. Each iteration of InferByMerge has polynomial complexity in n, as linear regression takes polynomial time (it is reduced to computing matrix multiplication and inverse). However, the whole merging process potentially needs to search over all subsets of classes, which is exponential in the number of classes after segmentation. In order to accelerate the search process, we use a pruning method, which is effective in most of the practical situations. InferByPWA calls Algorithm 2 when reducing the number of modes, while other parts, e.g. linear regression and the extending process, all have polynomial complexity in the number of input points. In Section 5, we will see that both methods can handle inputs of up to thousands of points within a reasonable time.

5. Experiments

In this section, we describe five SNDS models and the evaluation of the above proposed methods on them. We designed our own examples since we are currently not aware of any standard benchmarks consisting of switched nonlinear dynamical systems.

Before reporting the results of the experiments, we describe the evaluation method which we apply to measure the "closeness" between the inferred system and the original system. We compute the actual and predicted derivative along multiple trajectories, including both the input trajectory and additional trajectories from other initial points. The relative difference between actual and predicted derivatives are computed according to Equation (18). Then the results are averaged yielding an overall score d_{avg} between 0 and 1, where 0 and 1 represent perfect and worst predictions respectively.

Now we introduce the arrangement of our experiments. We first present our five examples, distinguished by their dimension, degree of the polynomials in the ODE and the boundary, and the number of modes, intended to cover the different aspects of the problem. We utilized our two methods to infer models of the examples based on a group of fixed parameters, i.e., timestep size t_{step}, number of initial points N_{init}, and the simulation time length I.

Then, for evaluating robustness, we modify the parameters for each example and compare the performance of different methods on the variants. The intuition is that fewer initial points are intended to be more difficult, as it means less data provided to the algorithm. Likewise, a larger time step is intended to be more difficult, as it means both less data and the derivative estimates from LMM are likely to be less accurate. The absolute error tolerance δ is a tunable parameter in the method InferByMerge.

We also compared our methods with the baseline clustering method DBSCAN, identification of ARX models using the system identification toolbox in Matlab, and standard optimization functions in SciPy, as well as with state-of-the-art method in [AS14] for piecewise affine models. Some of the results are shown below.

We implemented[2] both proposed methods using Python 3.7 with libraries SciPy, scikit-learn and LIBSVM. All of the evaluations have been carried out on an 1.8GHz Intel Core-i7 8550U processor with 8GB RAM running 64-bit Windows.

Each concrete case of experiments is run as follows:

1. *Training data generation:* For each initial point, solve the ODE numerically. Here we use SciPy's `solve_ivp` function. In particular, we use the event mechanism of the function to allow simulation of a switched system. This yields a number of discrete trajectories given by lists of time-position pairs.
2. *Inferring SNDS:* Feed the data to each of the three algorithms described in Section 4, and obtain the inferred SNDS.
3. *Evaluation:* Compare the inferred SNDS with the real SNDS by evaluating the predicted derivative along each of the original trajectories. Then compare the predicted and actual derivative using Equation (18), and average the obtained measure along the trajectories. This gives us a single number between 0 and 1 that indicates the quality of the inferred model. Also, for each of the additional points, use the original model to find trajectories starting at those points, and compute the measure also along these trajectories. This tests whether the inferred model also provides good predictions along other possible trajectories of the system.

In the following, all figures are rounded to 5 digits after the decimal point.

5.1. The Isolette example

This example shows the case in Fig. 1. It can be represented as an SNDS as follows.

If $x_1 \leq a_3$ then

$$\begin{cases} \dot{x_1} = a_1 \cdot (x_1 - x_2) \\ \dot{x_2} = 1 \end{cases}$$

else

$$\begin{cases} \dot{x_1} = a_2 \cdot (x_1 - x_2) \\ \dot{x_2} = -1 \end{cases}$$

[2] The implementation is available at https://github.com/Leslieaj/InferDynamic.

The system given in Section 1 corresponds to $a_3 = 98.5$ and $a_1 = a_2 = -0.026$. We choose two initial points $N_{init} = \{(99.5, 80), (97.5, 100)\}$, the timestep size $t_{step} = 0.1$, the simulation time length $I = 50$, the error tolerance $\epsilon = 0.01$, and apply InferByPWA (Algorithm 4).

The original coefficient matrices thus are as follows.

$$F_1 = \begin{bmatrix} -0.026 & 0 \\ 0.026 & 0 \\ 0 & 1 \end{bmatrix}, \quad F_2 = \begin{bmatrix} -0.026 & 0 \\ 0.026 & 0 \\ 0 & -1 \end{bmatrix}$$

The inferred coefficient matrices are

$$\bar{F}_1 = \begin{bmatrix} -2.60000 \times 10^{-2} & -8.00269 \times 10^{-14} \\ 2.60000 \times 10^{-2} & -2.49814 \times 10^{-14} \\ -3.54935 \times 10^{-13} & 1.00000 \end{bmatrix}, \quad \bar{F}_2 = \begin{bmatrix} -2.60006 \times 10^{-2} & -4.15116 \times 10^{-5} \\ 2.60007 \times 10^{-2} & 5.45885 \times 10^{-6} \\ 4.86236 \times 10^{-5} & -9.96412 \times 10^{-1} \end{bmatrix}$$

The boundary is

$$x_1 - 0.00034x_2 - 98.46672 = 0.$$

The average relative distance $d_{avg} = 6.55855 \times 10^{-6}$. Here, we also present the simulation of original and inferred models on the given initial points in Fig. 2.

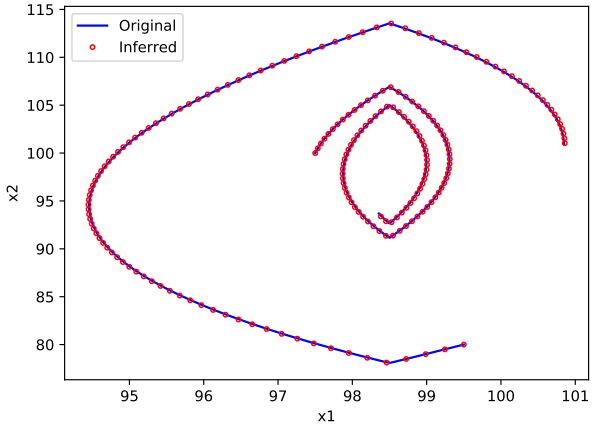


Fig. 2: The simulation result for the Isolette example. The trajectory of the inferred model closely follows that of the original model, showing that our method performs well on this example.

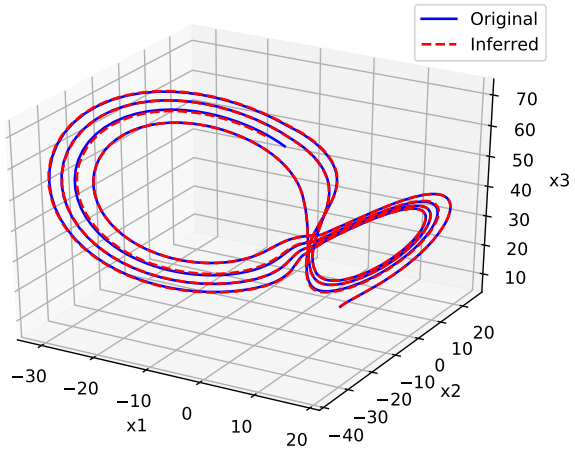


Fig. 3: The simulation result for switched Lorenz attractor. The divergence of blue and red lines are expected as the system is *chaotic*, but it can be seen that the red lines lie on roughly the same attractor as the blue lines.

5.2. A switched version of Lorenz attractor

The second case is a switched version of Lorenz attractor which has three dimensions, with ODE given by polynomials of degree two, and two modes. It represents a case where the system is *chaotic*. The corresponding SNDS is as follows.

If $x_1 + x_2 \geq 0$ then

$$\begin{cases} \dot{x_1} = \sigma_1(x_2 - x_1) \\ \dot{x_2} = x_1(\rho_1 - x_3) - x_2 \\ \dot{x_3} = x_1 x_2 - \beta_1 x_3 \end{cases}$$

else

$$\begin{cases} \dot{x_1} = \sigma_2(x_2 - x_1) \\ \dot{x_2} = x_1(\rho_2 - x_3) - x_2 \\ \dot{x_3} = x_1 x_2 - \beta_2 x_3 \end{cases}$$

One standard choice of coefficients is $\sigma_1 = 10, \beta_1 = 8/3, \rho_1 = 28, \sigma_2 = 28, \beta_2 = 4, \rho_2 = 46.92$. We choose two initial points $N_{init} = \{(5, 5, 5), (2, 2, 2)\}$, timestep size $t_{step} = 0.004$, simulation time length $I = 5$, absolute error tolerance $\delta = 0.2$, and apply InferByMerge (Algorithm 2).

Since the number of coefficients are too large ($3 \times \binom{3+2}{2} = 30$), we omit the four coefficient matrices and only report the average relative distance $d_{avg} = 8.86362 \times 10^{-5}$. The boundary is

$$x_1 + 0.98804x_2 + 0.00086x_3 - 0.01007 = 0$$

and the simulation result for this case is shown in Fig. 3.

In this case, there are 2502 points in total, with 1761 points belonging to the first mode and the rest belonging to the second one. With respect to Equation (19), $D_1 = 0.30153 \times 10^{21}$, $D_2 = 0.33927 \times 10^{24}$, and the sum of squared error is 0.34628, it is easy to calculate that

$$0.34628 < \frac{1}{18} \times 0.004^9 \times (1761 \times 0.004 \times D_1 + 741 \times 0.004 \times D_2) = 14.67599$$

corresponding to the result of Equation (19).

5.3. An example with degree three

In this example, back in two dimensions, the ODE is given by polynomials of degree three, and the boundary separating the two modes has degree two. The SDNS is given as follows.

If $x_2 \geq dx_1^2$, then

$$\begin{cases} \dot{x_1} = a_1 x_1^2 + b_1 x_2^3 \\ \dot{x_2} = c_1 x_1 \end{cases}$$

else

$$\begin{cases} \dot{x_1} = a_2 x_1^2 + b_2 x_2^3 \\ \dot{x_2} = c_2 x_1 \end{cases}$$

One standard choice of coefficients is $a_1 = 0.1, a_2 = 0.04, a_3 = -0.9, b_1 = -0.1, b_2 = 0.06, b_3 = -0.7$ and $d = -0.2$. The settings of parameters and experimental results are given in Table 1.

5.4. An example with four dimensions

This example tests the performance of the algorithm over larger dimensions. The SNDS has dimension four, degree two and two modes as follows.

If $x_3 \geq 0$ then

$$\begin{cases} \dot{x_1} = -sx_2 \\ \dot{x_2} = x_1 - r_1 \\ \dot{x_3} = x_4 \\ \dot{x_4} = -p(x_3 + r_2)^2 \end{cases}$$

else

$$\begin{cases} \dot{x_1} = -sx_2 \\ \dot{x_2} = x_1 - r_1 \\ \dot{x_3} = x_4 \\ \dot{x_4} = -p(x_1 - x_3)^2 \end{cases}$$

One choice of coefficients is $s = 1, r_1 = 5, r_2 = 5$ and $p = 0.1$. We also wrapped settings and results in Table 1.

5.5. An example with more than two modes

The last example tests the case with more than two modes. This example is in three dimensions, with ODE given by polynomials of degree two, and three modes.

If $-x_2 \geq 0$ then

$$\begin{cases} \dot{x_1} = c_1 \\ \dot{x_2} = c_2 x_1 \end{cases}$$

else if $-x_1 \geq 0$ then

$$\begin{cases} \dot{x_1} = a_1 x_1^2 + a_2 x_2 \\ \dot{x_2} = a_3 x_1 + a_4 \end{cases}$$

else

$$\begin{cases} \dot{x_1} = b_1 \\ \dot{x_2} = b_2 x_1 + b_3 \end{cases}$$

One choice of coefficients is $a_1 = 0.5, a_2 = 0.5, a_3 = -9, a_4 = 3, b_1 = 5, b_2 = -0.1, b_3 = -10, c_1 = -7, c_2 = -1$. Given three initial points $N_{init} = \{(-1, 1), (1, 4), (2, -3)\}$, timestep size $t_{step} = 0.01$, simulation time length $I = 5$, error tolerance $\epsilon = 0.01$, and applying InferByPWA, the average relative distance is $d_{avg} = 3.71359 \times 10^{-5}$. The simulation result for this case is shown in Fig. 4. The inferred boundaries are

$$0.00457x_1 + x_2 - 0.02923 = 0 \quad \text{and} \quad x_1 + 0.00346x_2 - 0.03707 = 0.$$

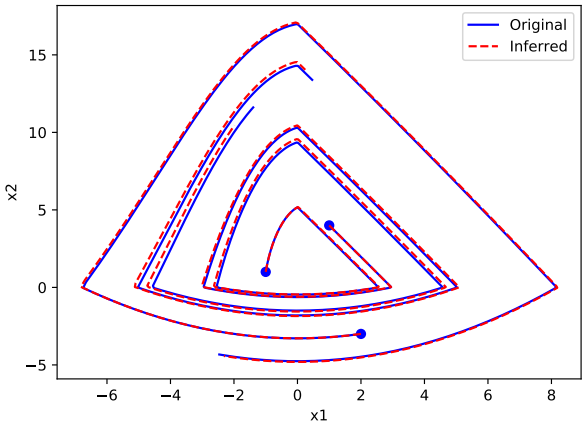


Fig. 4. The simulation result for Example 5.5.

Table 1. Experimental results on variants.

VID	EID	$\lvert N_{init}\rvert$	t_{step}	I	δ	InferByMerge		InferByPWA		DBSCAN		Mosaic		
						d^1_{avg}	t^1_{infer}	d^2_{avg}	t^2_{infer}	d^D_{avg}	t^D_{infer}	d^A_{avg}	t^A_{infer}	N^A
1	A	2	0.100	50	0.010	0.00047	0.6	0.00047	0.4	0.00096	0.2	0.08582	9.4	11
2	A	3	0.100	50	0.010	0.00001	1.5	0.00001	0.8	0.00080	0.3	0.07176	8.8	15
3	A	2	0.200	50	0.010	0.00001	0.3	0.00096	0.3	0.00100	0.1	0.14168	4.1	12
4	A	3	0.200	50	0.010	0.00001	0.7	0.00002	0.4	0.00001	0.2	0.11809	5.4	13
5	B	2	0.010	5	0.200	0.00139	1.4	0.00147	1.0	0.13315	2.6	0.14948	295.9	207
6	B	1	0.002	15	0.200	0.00005	12.8	0.00005	4.8	0.00005	1.8	0.09949	1717.8	740
7	B	2	0.002	10	0.400	0.00010	25.0	0.00006	6.2	0.00010	2.2	0.03255	2757.1	1112
8	B	5	0.004	5	0.400	0.00006	17.9	0.00011	3.5	0.00005	1.3	0.07927	478.1	320
9	C	5	0.010	10	0.001	0.00017	24.4	0.00008	2.6	0.00047	1.2	0.04365	46.8	60
10	C	3	0.010	10	0.001	0.00129	4.3	0.00119	1.6	0.00129	0.6	0.22206	18.2	31
11	C	3	0.050	10	0.010	0.00036	4.1	0.00149	0.5	0.00036	0.2	0.19723	15.3	31
12	C	2	0.010	15	0.010	0.00444	6.4	0.00537	1.6	0.00442	0.5	0.29037	16.6	17
13	D	4	0.010	5	0.010	0.00020	3.7	0.00055	1.8	0.00016	0.5	0.03707	21.9	40
14	D	2	0.010	5	0.010	0.00150	1.0	0.00025	1.1	0.00150	0.3	0.33632	5.7	14
15	D	4	0.002	5	0.010	0.00031	19.6	0.00031	7.6	0.00031	2.4	0.09396	37.5	35
16	D	3	0.010	10	0.005	0.00111	5.6	0.00098	3.9	0.00096	0.7	0.01599	51.7	59
17	E	3	0.010	5	0.010	0.00033	2.9	0.00033	6.5	0.27233	0.4	0.04612	54.7	58
18	E	2	0.010	10	0.010	0.00020	5.1	0.00060	8.0	0.32967	0.7	0.08562	70.5	72
19	E	5	0.020	5	0.010	0.00097	4.3	0.00095	3.5	0.00101	0.8	0.04348	79.5	79
20	E	5	0.010	5	0.010	0.00090	9.8	0.00089	10.9	0.00089	2.1	0.01340	92.4	79

VID: the variant ID.
EID: the example ID indicates which example the variant belongs to.
$\lvert N_{init}\rvert$: the number of initial points (two additional points are used for testing in each case).
t_{step}: time step size.
I: interval of simulation.
δ: the absolute error tolerance for each point (only in InferByMerge).
d^1_{avg}, d^2_{avg}, d^D_{avg}, d^A_{avg}: average relative distances using InferByMerge, InferByPWA,the DBSCAN clustering method and Mosaic [AS14], respectively.
t^1_{infer}, t^2_{infer}, t^D_{infer}, t^A_{infer}: wall-clock inference time in seconds using InferByMerge, InferByPWA, the DBSCAN clustering method, and Mosaic, respectively.
N^A: the number of learned piecewise linear modes in Mosaic

5.6. Comparison and Discussion

In order to evaluate the robustness and compare the performance of the methods, we modify the parameters, i.e, initial points N_{init}, timestep size t_{step}, and the simulation time length I for the five examples and form a collection of variants. For all variants, the relative error tolerance $\epsilon = 0.01$ and the choices of the absolute error tolerance δ used in InferByMerge have also been given. The new SNDS models and

the results of the experiments are shown in Table 1. In this table, d^1_{avg}, d^2_{avg}, d^D_{avg}, and d^A_{avg} represent the average relative distances using InferByMerge (Algorithm 2), InferByPWA (Algorithm 4), the DBSCAN clustering method, and Mosaic [AS14], respectively. t^1_{infer}, t^2_{infer}, t^D_{infer}, and t^A_{infer} represent the wall-clock inference time in seconds using InferByMerge, InferByPWA, the DBSCAN clustering method, and Mosaic, respectively. From the table, it can be seen that DBSCAN performs well on some cases, but is not sufficiently robust, for example performing poorly on the cases with three modes and two of the cases for the Lorenz attractor. This is to be expected from the discussion of its weaknesses at the end of Section 4.2. InferByMerge and InferByPWA are robust across all of our cases, and each performs better on a different set of cases. InferByPWA extends identification methods of piecewise affine models, and uses the idea of merging modes in InferByMerge at the end, hence it performs well on almost all of the cases. In experiment A, the dynamics of both modes are linear and we can see that even if the sampling stepsize is large, the model learned is close to the original. While in the other experiments, we need a smaller stepsize to get a more accurate approximation of derivatives. After checking the relative differences between derivatives at different points, it can be seen that most of the differences are due to those near the switch points.

Below, we compare with three other representative methods. Both Mosaic and standard optimization methods only deal with the problem of fitting the relationship between positions and derivatives, so we perform the LMM stage first and then apply these two methods.

5.6.1. Comparison with Mosaic

First, we compare with the tool Mosaic from the paper [AS14]. Our InferByPWA is a modification from the method reported in that paper. However, only learning of piecewise linear models are considered there, and nonlinear models are approximated by a large number of piecewise linear modes. The results from Mosaic are given in Table 1. Here column d^A_{avg} is the average relative error, t^A_{infer} is time spent in each case, and N^A is the number of (piecewise linear) modes. It can be seen that across all cases, the relative error is much larger than that obtained using our InferByMerge and InferByPWA, the running time is longer, and the number of modes (an indicator of the complexity of the model) is larger. This demonstrates that by considering nonlinearity with polynomial equations directly, our method gives a significant improvement over learning of purely piecewise linear models.

5.6.2. Comparison with ARX models

For comparison with inference methods of ARX models, we performed experiments using the System Identification Toolbox in Matlab. We applied Matlab's `arx` and `nlarx` functions (for linear and nonlinear ARX models) and chose different parameters to find a (linear or nonlinear) ARX model fitting the trajectories. The results show a big difference between the predicted and the original trajectories. Also, the predicted trajectories are no longer continuous and difficult to interpret.

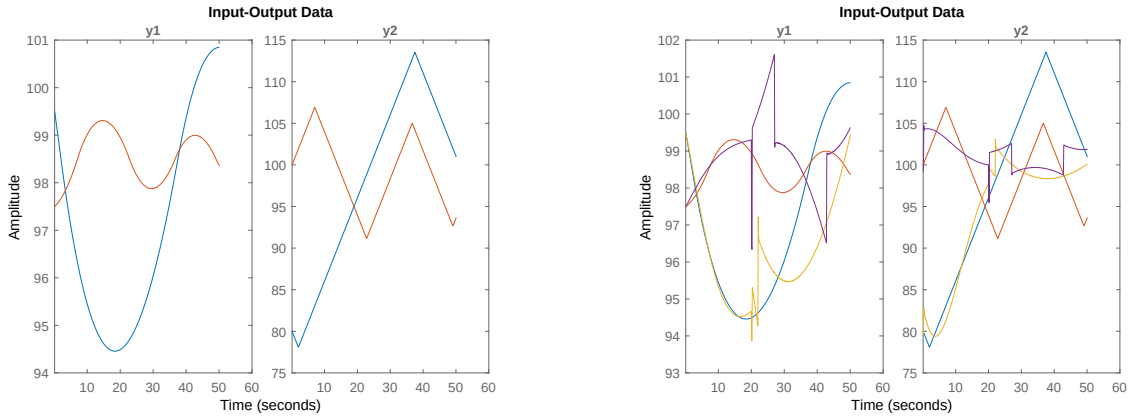
For example, the result of inferring linear ARX models on the Isolette example is shown in Fig. 5. In this figure, part (a) shows the given trajectories. In part (b), it is superimposed with the predicted trajectory from the inferred linear ARX model. As can be seen, the predicted trajectory follows the given trajectory initially, but diverges around the first mode switch. This indicates that ARX models as provided in Matlab are unable to handle mode switches. The result of using the nonlinear ARX model is even worse, hence we omit it here.

The reason may be that the multiple modes in SNDS are difficult to explain using a single ARX model. This shows the necessity of using segmentation and clustering (as in this work) to learn an interpretable system.

5.6.3. Comparison with standard optimization methods

The problem in this paper (after the LMM stage) can be stated as an optimization problem, minimizing the sum (or average) of the relative difference between predicted derivative and the derivative computed by the LMM method. The objective function is as follows.

$$\sum_{\boldsymbol{x}(t_j)\in\bigcup_{S_i\in S} S_i} \min_{0\le m\le N} d(\Phi(\boldsymbol{x}(t_j))\cdot[F_1^m,\cdots,F_i^m,\cdots,F_n^m],\bar{f}(\boldsymbol{x}(t_j))) \tag{20}$$



(a) The trajectories provided to ARX model method (b) The predicted trajectory by using the inferred ARX model

Fig. 5. Experimental results on the Isolette example using ARX models.

Table 2. Experimental results using standard optimization method "Nelder-Mead" in SciPy.

VID	#1		#2		#3		#4		#5	
	d^O_{avg}	t^O_{infer}	d^O_{avg}	t^O_{infer}	d^O_{avg}	t^O_{infer}	d^O_{avg}	t^O_{infer}	d^O_{avg}	t^O_{infer}
1	0.59704	98.5	0.46404	285.4	0.42152	110.2	0.42111	245.8	0.45285	112.6
2	0.55577	138.4	0.42927	435.5	0.45377	210.0	0.38807	525.4	0.55437	130.0
3	0.41466	67.8	0.45715	56.2	0.57919	52.9	0.43419	78.8	0.57795	54.8
4	0.56721	82.9	0.58181	100.1	0.56825	118.6	0.47693	82.7	0.44028	72.0
5-10	**TO**		**TO**		**TO**		**TO**		**TO**	
11	**TO**		0.74582	981.6	0.71463	961.8	**TO**		0.45468	991.9
12-20	**TO**		**TO**		**TO**		**TO**		**TO**	

#1 - #5: Indexes represent 5 trials from different initial points respectively.
TO: Timeout ($>$ 20 mins).

where S is the set of trajectories, d is the relative difference defined in Section 4, N is the number of modes, F_i^m represents the coefficients of the i-th component of the polynomial specifying the dynamics in mode m, and $\bar{f}(\boldsymbol{x}(t_j))$ is our estimate of the derivative at point $\boldsymbol{x}(t_j)$ according to the trajectories. The independent variables of this objective function is all of the coefficients F_i^m.

We applied standard optimization methods in `scipy.optimize`. However, since the objective function is not convex, and moreover not differentiable everywhere, the methods do not perform well in our case studies. Here we report the results using method "Nelder-Mead" and randomly choosing 5 initial points for the optimization. The experimental results on the same 20 examples are shown in Table 2. In this table, #1 - #5 represent 5 trials from different random initial points. We also set 1200 seconds as the timeout. t^O_{infer} and d^O_{avg} represent the running time and the average relative difference in the derivative, respectively. The experimental results indicate that the optimization method easily falls into a local minimum and does not performs well in all our examples.

6. Conclusion

In this paper, we presented two methods for the identification of switched nonlinear dynamical systems from trajectory data, and proposed a way to evaluate an inferred model by comparison to the original model. Using this evaluation, we tested the two methods for robustness and compared them on five classes of examples. The experimental results show that these two methods are robust across a wide range of situations, and each performs better on different cases.

The techniques introduced in this paper can form a basis on which to extend to identification of more general classes of systems, including nonlinear hybrid systems with internal state and non-determinstic be-

havior, systems switching modes depending on time and systems with higher-order ODEs. Another direction is to extend the method to handle perturbations in the input data, for example coming from noise in the measurement process. Finally, as the stability and convergence of Linear Multistep Methods for single ODEs are analyzed in [KD19], we would like to analyze stability and convergence properties for our methods in the switched case.

References

[ACZ+20] Jie An, Mingshuai Chen, Bohua Zhan, Naijun Zhan, and Miaomiao Zhang. Learning one-clock timed automata. In *Proceedings of 26th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 444–462. Springer, 2020.

[AL89] Ivan Auger and Charles Lawrence. Algorithms for the optimal identification of segment neighborhoods. *Bulletin of Mathematical Biology*, 51(1):39–54, 1989.

[Ang87] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987.

[AS14] Rajeev Alur and Nimit Singhania. Precise piecewise affine models from input-output data. In *Proceedings of the 14th International Conference on Embedded Software*, pages 3(1)–3(10), 2014.

[AWZ+ss] Jie An, Lingtai Wang, Bohua Zhan, Naijun Zhan, and Miaomiao Zhang. Learning real-time automata. *SCIENCE CHINA Information Sciences*, 2021, In press.

[BDG+20] Ezio Bartocci, Jyotirmoy Deshmukh, Felix Gigler, Cristinel Mateis, Dejan Ničković, and Xin Qin. Mining shape expressions from positive examples. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):3809–3820, 2020.

[BG08] John Charles Butcher and Nicolette Goodwin. *Numerical methods for ordinary differential equations*, volume 2. Wiley Online Library, 2008.

[BGPV05] Alberto Bemporad, Andrea Garulli, Simone Paoletti, and Antonio Vicino. A bounded-error approach to piecewise affine system identification. *IEEE Transactions on Automatic Control*, 50(10):1567–1580, 2005.

[BHKL09] Benedikt Bollig, Peter Habermehl, Carsten Kern, and Martin Leucker. Angluin-style learning of NFA. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 1004–1009, 2009.

[BIB11] Roberto Baptista, Joao Yoshiyuki Ishihara, and Geovany Araujo Borges. Split and merge algorithm for identification of piecewise affine systems. In *Proceedings of the 2011 American Control Conference*, pages 2018–2023, 2011.

[BPK16] Steven Brunton, Joshua Proctor, and Jose Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.

[BVVB05] José Borges, Vincent Verdult, Michel Verhaegen, and Miguel Ayala Botto. A switching detection method based on projected subspace classification. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 344–349, 2005.

[DD17] Samuel Drews and Loris DAntoni. Learning symbolic automata. In *Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 173–189. Springer, 2017.

[Eyk74] Pieter Eykhoff. *System Identification and State Estimation*. Wiley, 1974.

[FCC+08] Azadeh Farzan, Yu-Fang Chen, Edmund M Clarke, Yih-Kuen Tsay, and Bow-Yaw Wang. Extending automated compositional verification to the full class of omega-regular languages. In *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 2–17. Springer, 2008.

[FMLM03] Giancarlo Ferrari-Trecate, Marco Muselli, Diego Liberati, and Manfred Morari. A clustering technique for the identification of piecewise affine systems. *Automatica*, 39(2):205–217, 2003.

[GAG+00] Ary L. Goldberger, Luis A. N. Amaral, Leon Glass, Jeffrey M. Hausdorff, Plamen Ch. Ivanov, Roger G. Mark, Joseph E. Mietus, George B. Moody, Chung-Kang Peng, and H. Eugene Stanley. Physiobank, physiotoolkit, and physionet: Components of a new research resource for complex physiologic signals. *Circulation*, 101(23):e215–e220, 2000.

[GPV12] Andrea Garulli, Simone Paoletti, and Antonio Vicino. A survey on switched and piecewise affine system identification. *IFAC Proceedings Volumes*, 45(16):344 – 355, 2012.

[Hen96] Thomas A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*, pages 278–292. IEEE Computer Society, 1996.

[HV05] Yasmin Hashambhoy and René Vidal. Recursive identification of switched ARX models with unknown number of models and unknown orders. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 6115–6121. IEEE, 2005.

[KD19] Rachael Keller and Qiang Du. Discovery of dynamics using linear multistep methods, 2019.

[LB08] Fabien Lauer and Gérard Bloch. Switched and piecewise nonlinear hybrid system identification. In *Proceedings of the 11th International Workshop on Hybrid Systems: Computation and Control*, pages 330–343. Springer, 2008.

[LBG18] Imane Lamrani, Ayan Banerjee, and Sandeep KS Gupta. Hymn: mining linear hybrid automata from input output traces of cyber-physical systems. In *Proceedings of the 2018 IEEE Industrial Cyber-Physical Systems*, pages 264–269. IEEE, 2018.

[LBV10] Fabien Lauer, Gérard Bloch, and René Vidal. Nonlinear hybrid system identification with kernel models. In *Proceedings of the 49th IEEE Conference on Decision and Control*, pages 696–701. IEEE, 2010.

[LCZL17] Yong Li, Yu-Fang Chen, Lijun Zhang, and Depeng Liu. A novel learning algorithm for büchi automata based on

family of dfas and classification trees. In *Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 208–226, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg.

[Lju99] Lennart Ljung. *System Identification: Theory for the User*. Prentice Hall, Upper Saddle River, New Jersey, 1999.

[Lju15] Lennart Ljung. System identification: An overview. In *Encyclopedia of Systems and Control*. Springer, 2015.

[Lor63] Edward N. Lorenz. Deterministic nonperiodic flow. *Journal of Atmospheric Sciences*, 20(2):130–148, 1963.

[MKY11] Abdullah Mueen, Eamonn Keogh, and Neal Young. Logical-shapelets: an expressive primitive for time series classification. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1154–1162, 2011.

[MM14] Oded Maler and Irini-Eleftheria Mens. Learning regular languages over large alphabets. In *Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 485–499. Springer, 2014.

[Moo58] Edward F. Moore. Gedanken-experiments on sequential machines. *Annals of Mathematics*, 34:129–153, 1958.

[MRBF15] Ramy Medhat, Sethu Ramesh, Borzoo Bonakdarpour, and Sebastian Fischmeister. A framework for mining hybrid automata from input/output traces. In *Proceedings of the 2015 International Conference on Embedded Software*, pages 177–186. IEEE, 2015.

[NQF+19] Dejan Ničković, Xin Qin, Thomas Ferrère, Cristinel Mateis, and Jyotirmoy Deshmukh. Shape expressions for specifying and extracting signal features. In *Proceedings of the 19th International Conference on Runtime Verification*, pages 292–309. Springer, 2019.

[NSV+12] Oliver Niggemann, Benno Stein, Asmir Vodencarevic, Alexander Maier, and Hans Kleine Büning. Learning behavior models for hybrid timed systems. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*. AAAI Press, 2012.

[OG92] Jose Oncina and Pedro Garcia. Identifying regular languages in polynomial time. In *Advances in Structural and Syntactic Pattern Recognition, Volumn 5 of Series in Machine Perception and Artificial Intelligence*, pages 99–108. World Scientific, 1992.

[OLB10] Henrik Ohlsson, Lennart Ljung, and Stephen Boyd. Segmentation of arx-models using sum-of-norms regularization. *Automatica*, 46(6):1107–1111, 2010.

[Oza16] Necmiye Ozay. An exact and efficient algorithm for segmentation of ARX models. In *Proceedings of the 2016 American Control Conference*, pages 38–41. IEEE, 2016.

[Pen55] Roger Penrose. A generalized inverse for matrices. *Mathematical Proceedings of the Cambridge Philosophical Society*, 51(3):406–413, 1955.

[PJFV07] Simone Paoletti, Aleksandar Lj. Juloski, Giancarlo Ferrari-Trecate, and René Vidal. Identification of hybrid systems a tutorial. *European journal of control*, 13(2-3):242–260, 2007.

[POTN03] Hui Peng, Tohru Ozaki, Yukihiro Toyoda, and Kazushi Nakano. Nonlinear system modelling using the rbf neural network-based regressive model. *IFAC Proceedings Volumes*, 36(16):333 – 338, 2003. Proceedings of the 13th IFAC Symposium on System Identification.

[SG09] Muzammil Shahbaz and Roland Groz. Inferring mealy machines. In *Proceedings of the 16th International Symposium on Formal Methods*, pages 207–222. Springer, 2009.

[SHSZ19] Miriam García Soto, Thomas A Henzinger, Christian Schilling, and Luka Zeleznik. Membership-based synthesis of linear hybrid automata. In *Proceedings of the 31st International Conference on Computer Aided Verification*, pages 297–314. Springer, 2019.

[TAB+19] Martin Tappler, Bernhard K. Aichernig, Giovanni Bacci, Maria Eichlseder, and Kim G. Larsen. L^*-based learning of Markov decision processes. In *Proceedings of the 23rd International Symposium on Formal Methods*, pages 651–669, 2019.

[Vaa17] Frits Vaandrager. Model learning. *Communications of the ACM*, 60(2):86–95, 2017.

[VdWW11] Sicco Verwer, Mathijs de Weerdt, and Cees Witteveen. The efficiency of identifying timed automata and the power of clocks. *Information and Computation*, 209(3):606–625, 2011.

[VdWW12] Sicco Verwer, Mathijs de Weerdt, and Cees Witteveen. Efficiently identifying deterministic real-time automata from labeled data. *Machine Learning*, 86(3):295–333, 2012.

[XPTP20] Wenquan Xu, Hui Peng, Xiaoying Tian, and Xiaoyan Peng. DBN based SD-ARX model for nonlinear time series prediction and analysis. *Applied Intelligence*, 50(12):4586–4601, 2020.