Active Learning of Switched Nonlinear Dynamical Systems

Hadi Dayekh, Nicolas Basset, Thao Dang

Active Learning of Switched Nonlinear Dynamical Systems

Hadi Dayekh[1], Nicolas Basset[1], and Thao Dang[1]

***Abstract*— Most hybrid system identification methods rely on passive learning techniques, limiting the accuracy of the learned model to the data at hand. We present an active learning approach to identify state-dependent switched nonlinear dynamical systems with polynomial ODEs. Counterexample trajectories indicating a divergence between the system under learning and a learned hypothesis model are provided by an approximate equivalence query. Segmentation is applied on the true trajectories of the counterexamples before treating each segment. We provide a way to incrementally update the learned continuous dynamics to accommodate each segment if needed, without any assumption on the number of modes, before updating the mode regions. Our method uses multivariate polynomial regression for finding the continuous dynamics and multinomial logistic regression for the mode regions. We illustrate our approach and its effectiveness on multiple examples, including a parametric one with 20 modes.**

I. Introduction

Hybrid system identification is mostly done in passive learning ways. These passive learning techniques consider given data at once to infer a model, and they are appropriate when a lot of data is available; however unnecessary data can be overused in the identification algorithm, which could lead to complexity issues. On the other hand, even when available data is abundant, the data might still be insufficient to derive a good model of the system under learning. This problem of quality of data can be remedied by active learning which is more adaptive to the need of the identification algorithm, since it allows correcting the learned model by acquiring new data, guided by instances of mismatch between the learned model and the system under learning. Consequently, with less but more informative data, active learning can provide a more accurate model with less computational complexity.

Passive learning of hybrid systems is closely related to hybrid systems identification, initially investigated in control theory. Significant results were achieved for Piecewise affine AutoRegressive eXogenous (PWARX) or Switched affine AutoRegressive eXogenous (SARX) models (see [1, 2] for surveys and references therein). PWARX can be seen as a particular class of linear hybrid automata where switching dynamics are deterministic. A major challenge of this identification problem includes the inference of the dynamics of submodels, as well as the partition of the state and input spaces. The existing major approaches to address this are mainly based on algebraic formulation, clustering, and optimization. The algebraic approach [3, 4] specifies the conditions for each data point to belong to one submodel by a homogeneous polynomial equation, and then the switching dynamics can be deduced. The clustering approach uses features to assign submodels to data points [5]. This can be done iteratively in order to minimize the number of submodels while satisfying a desired matching error bound [6]. This optimization-based approach can be extended to systems with nonlinear dynamics by expressing the dynamics of submodels in kernel expansion form and solving the associated relaxed optimization problem using kernel support vector regression. A dynamic programming formulation is used in [7] to solve the problem of segmentation of an ARX trajectory with a minimum number of mode changes. In [8] for switched linear systems non-smooth programming is exploited in an algorithm that is linear in the size of data, rather than exponential as in a direct mixed-integer linear programming approach [9]. For linear complementarity systems, a new efficient optimization procedure using violation-based loss leads to a significant efficiency improvement [10]. Besides, machine learning techniques are used, such as in [11] which employs symbolic regression to infer a symbolic discrete dynamical model with continuous mappings. In [12], a Bayesian approach is used to identify PWARX models.

Recent work addresses hybrid models with more general switching dynamics (guards, invariants, and resets) and with continuous dynamics in continuous time, such as [13, 14, 15] for affine hybrid automata, [16] for switched nonlinear systems and [17, 18] for nonlinear hybrid automata.

While passive learning techniques for the identification of hybrid models have become more and more well-established, active learning is still less investigated. Active learning has been intensively studied in the context of automata learning since [19] and recently applied for timed automata learning [20, 21, 22, 23, 24]. In [25], an active identification problem is defined as how to generate the input stimuli to a linear dynamical system in order to accelerate its parameter estimation. Concerning hybrid systems, the work [26] has some active learning flavor where model synthesis is considered to expand the current model so as to capture new experimental data. In [13], an inferred linear hybrid automaton can be refined by modifying the input stimuli. In [27] the active learning feature is more emphasized since automata learning techniques are applied to learn a discretized hybrid system, and then model-based testing is used to generate data suffi-

[1]Université Grenoble Alpes, CNRS, Grenoble INP, VERIMAG, Grenoble, France
firstname.lastname@univ-grenoble-alpes.fr

* This work is partially supported by the joint French-Japanese ANR-JST project CyPhAI, the ANR project Maveriq, and the Auvergne-Rhône-Alpes Region Project DetAI

cient for creating a neural network behavioral model. Note however that in this work the active learning process is done over a discretized state space of the real system, while in our approach it is done over the hybrid state space to directly discover a hybrid model. Our previous work [28] proposes a framework for active learning of mode partition (that is, state-space partition for submodels) with the assumption that the continuous modes are already correctly identified in a previous step. In this work, we do not need this assumption and propose an algorithm for learning both continuous dynamics and mode partition in an active manner. This new algorithm also allows us to demonstrate the advantage of active learning in terms of data utilization.

The paper is organized as follows: section II provides the problem statement as well as introduces the necessary techniques required for our algorithm, namely segmentation and multivariate polynomial regression. Subsequently, section III goes over the main steps of our active learning approach and explains our equivalence query approximation. That being done, section IV and section V detail our algorithm in updating the continuous dynamics of a given hypothesis to treat counterexamples, as well as how we find the mode regions, with section IV being the main contribution of this paper. We illustrate our approach on several examples in section VI. Finally, we provide a conclusion and talk about future work and perspectives in section VII.

II. Preliminaries

In this section, we provide the reader with a formal definition of the problem we aim to solve, as well as certain concepts and techniques necessary to our algorithm.

A. Problem Statement

The problem we tackle is identifying *switched nonlinear dynamical systems* (SNDS) whose dynamics are governed by n continuous differential equations, as follows:

$$\dot{\boldsymbol{x}} = \boldsymbol{f}_j(\boldsymbol{x}) \;\text{ if } \boldsymbol{x} \in \mathcal{X}_j,\; j = 1, \ldots, n$$

where $\boldsymbol{x} \in \mathcal{X}_1 \cup \mathcal{X}_2 \cup \ldots \cup \mathcal{X}_n = \mathcal{X} \subseteq \mathbb{R}^D$ and $\mathcal{X}_i \cap \mathcal{X}_j = \varnothing,\; i \neq j$. Furthermore, each $\boldsymbol{f}_j(\boldsymbol{x})$ is polynomial in the components of $\boldsymbol{x}$ with a known maximum degree, and each $\mathcal{X}_j$, called *mode region*, is defined by a conjunction of polynomial inequalities, also with a known maximum degree.

We assume we have access to the system under learning (SUL) as a black box, that is, given an initial condition $\boldsymbol{x}_0$, a time step t_s, and a time range T, we can execute the SUL to generate a trajectory of the system under such conditions.

B. Segmentation and Estimating Derivatives

In our algorithm, the first step in treating any trajectory generated by the system under learning will be segmentation. Segmentation is a process aiming at detecting the change points that occur in a trajectory indicating that the system passes from one mode to another.

To perform segmentation, we use a similar method to [16], which compares estimated forward and backward derivatives at every point of the trajectory. In the event where the relative difference of the forward and backward derivatives is greater than a certain threshold, we deduce that there is a change point in the neighborhood of that point. The relative difference between two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, coined in [16] and used in [17, 28] is a normalized difference defined as:

$$rd(\boldsymbol{x}, \boldsymbol{y}) = \frac{||\boldsymbol{x} - \boldsymbol{y}||}{||\boldsymbol{x}|| + ||\boldsymbol{y}||} \tag{1}$$

In the next treatment, points in a small neighborhood of the change point which are used to estimate the forward and backward derivatives are then dropped. This is performed due to the uncertainty of where the change point really is, and in order to ensure that we are left with segments such that each is assumed to follow one mode.

A segment is represented by an index s. We define $\sigma : \mathcal{X} \rightharpoonup \mathbb{N}$ to be the segment partial function, mapping points in the state space $\mathcal{X}$ to the segment to which they belong. At each point of the learning algorithm, the matrix of training data is the *domain* of the partial function σ, denoted by $\mathbf{dom}(\sigma)$, where each row represents a training data point. Given a segment s, we represent by $\sigma^{-1}(s)$ the matrix of all points that belong to segment s.

Now that the segmentation is done, the next step is to estimate the derivatives of points in each segment. When doing segmentation, as mentioned earlier, discarding points too close to change points ensures that the derivative estimation at points in one mode does not use points in a different mode that have a different derivative function. Given points belonging to a certain segment, we use the five-point-stencil method [29], which uses forward and backward data along the trajectory, to estimate the derivatives.

We denote by $\mathbf{der}(\boldsymbol{X})$ the matrix of derivatives of each data point $\boldsymbol{x}$ in a matrix $\boldsymbol{X}$.

C. Identifying Dynamics of a Single Mode

Consider a matrix of data points $\boldsymbol{X}$ with its corresponding estimated derivatives $\mathbf{der}(\boldsymbol{X})$. In the event where all data points in $\boldsymbol{X}$ belong to a certain mode j, we use multivariate polynomial regression to estimate a function $\hat{\boldsymbol{f}}$ that governs the dynamics of this mode.

This approach consists of extracting polynomial features of each data point in $\boldsymbol{X}$ and then training a linear regression model on the transformed data. Given a vector $\boldsymbol{x} \in \boldsymbol{X}$ of dimension D, as well as a polynomial degree γ, one could extract $Q = \left(\binom{D+\gamma}{\gamma} - 1\right)$ monomials from $\boldsymbol{x}$ of degree less than or equal to γ, excluding 1. We denote by $\boldsymbol{\Psi}(\boldsymbol{x})$ the vector of polynomial features extracted from $\boldsymbol{x}$ and by $\boldsymbol{\Psi}(\boldsymbol{X})$ the matrix of extracted features of all row vectors $\boldsymbol{x}$ in $\boldsymbol{X}$, the matrix of all data points. As an example,

$$\boldsymbol{\Psi}((x_1, x_2)) = ({x_1}^2, {x_2}^2, x_1 x_2, x_1, x_2)$$

is the vector of all monomials up to degree 2 of the vector (x_1, x_2). For affine systems, $\boldsymbol{\Psi}(\boldsymbol{X}) = \boldsymbol{X}$ and $Q = D$.

Now, the d^{th} component d of $\hat{\boldsymbol{f}}(\boldsymbol{x})$ can be written as a scalar product of the polynomial features $\boldsymbol{\Psi}(\boldsymbol{x})$ with a vector $\omega^{(d)}$ of Q coefficients, in addition to a *bias* term $\beta^{(d)}$. That

is, $\hat{\boldsymbol{f}}^{(d)} = \boldsymbol{\Psi}(x)^T \cdot \boldsymbol{\omega}^{(d)} + \beta^{(d)}$. This reduces the problem of finding the function $\hat{\boldsymbol{f}}$ to fitting D linear regression models on each component of the vector function $\hat{\boldsymbol{f}}$, each with Q coefficients and a bias term.

III. Overview of the Approach

A. Active Learning and Equivalence Queries

In active learning terminology, given a hypothesis $\mathfrak{H}$ of the learned system, an equivalence query [19] is a check posed to an oracle to determine whether this hypothesis is equivalent to the system under learning. In the case where it is not, the oracle returns a counterexample illustrating this non-equivalence. The principle idea of active learning is to use such counterexamples to ameliorate the hypothesis until the equivalence query returns no counterexamples.

In our setting, as we do not have an oracle to decide whether a given hypothesis system is equivalent to the SUL, we approximate an equivalence query by generating trajectories from both systems and comparing them. Given a hypothesis $\mathfrak{H}$, the system under learning $\mathfrak{S}$, some user-defined number of trajectories $ntrajs$, a time step, a time range, as well as certain bounds limiting the state space of interest, we estimate an equivalence query by executing $\mathfrak{S}$ and $\mathfrak{H}$ on $ntrajs$ randomly chosen initial conditions within the defined bounds, with the same time step and time range. We then compare each pair of trajectories by computing the average relative difference (RD) of each pair of points (see Equation (1)). If this difference is less than a certain threshold tol, we consider that the hypothesis trajectory is matching with the one generated by $\mathfrak{S}$. Otherwise, this pair is considered to be a counterexample, and hence one needs to use the SUL trajectory to improve the learned model. We also allow for a user-defined number $ncexs$ of counterexamples to be returned, allowing to stop the equivalence query check once we have $ncexs$ counterexamples. Once the equivalence query returns no counterexamples on a certain hypothesis $\mathfrak{H}$, this hypothesis is returned as the final learned model.

B. Our Active Learning Algorithm

We will go over the main steps of our active learning algorithm before explaining the details of each step. In order to create a first hypothesis, we need some initial trajectories of the system under learning to begin with. These trajectories are treated as if they are counterexamples, since we begin with an empty learned model. Furthermore, our algorithm treats them the exact same way as consequent counterexamples returned by equivalence queries.

Now, in order to create a hypothesis, we need to find both the continuous dynamics and the mode regions. Throughout the learning algorithm, let $\mathfrak{F} = \{\hat{\boldsymbol{f}}_1, \ldots, \hat{\boldsymbol{f}}_n\}$ be the currently learned continuous dynamics of the system, where n is the current number of discovered modes, which could be updated along the learning algorithm. Furthermore, let $\mathfrak{R} = \{\hat{\mathcal{X}}_1, \ldots, \hat{\mathcal{X}}_n\}$ be the current mode regions.

Given trajectories of the counterexamples generated by the system under learning $\mathfrak{S}$, as mentioned in subsection II-B, the first step is to perform segmentation and estimate the derivatives. This returns the segment partial function σ, which implicitly contains all training data mapped to segment identifiers.

In order to find the dynamics, we need to know which mode the data we treat belongs to. Since we assume each segment of every trajectory belongs to one mode (see subsection II-B), it suffices to find the mode of each segment. Let $\lambda : \mathbb{N} \rightharpoonup \{1, \ldots, n\}$ be the mode partial function, mapping each segment $s \in \mathbf{range}(\sigma)$ to a mode, if known. We denote by $(\lambda \circ \sigma)^{-1}(j)$ the matrix of all points whose mode is found to be j.

Our high-level main algorithm is summarized in Algorithm 1. After segmenting the counterexample trajectories, line 7 incrementally updates the current continuous dynamics $\mathfrak{F}$ to accommodate the counterexamples, as well as finds the modes of each segment in these trajectories to update the mode partial function λ. Having found the mode of each data point, line 8 runs a logistic regression classifier to find new mode regions. The details of these two steps are explained in the next two sections.

Given the updated $\mathfrak{F}$ and $\mathfrak{R}$, we create an executable hypothesis from both, before posing a new equivalence query. The algorithm terminates when the equivalence query returns no counterexamples, in which case the current hypothesis $\mathfrak{H}$ is returned as the final result.

Algorithm 1 The main algorithm.

Input: $\mathfrak{S}$ the system under learning.
Output: $\mathfrak{H}$ the learned system.

```
1:  procedure ActiveLearning(𝔖)
2:      FirstTrajs ← first executed trajectories of 𝔖.
3:      CExs ← FirstTrajs ▷ we start with an empty hypothesis.
4:      σ ← ∅; λ ← ∅; 𝔉 ← ∅
5:      repeat
6:          σ ← SegmentAndDerivate(CExs, σ) ▷ segments CExs and adds it to σ
7:          λ, 𝔉 ← UpdateContinuous(σ, λ, 𝔉)
8:          ℜ ← FindRegions(σ, λ)
9:          𝔥 ← (𝔉, ℜ)                ▷ executable hypothesis
10:         CExs ← EquivalenceQuery(𝔖, 𝔥)
11:     until CExs = ∅
12:     return 𝔥
```

IV. Updating Continuous Dynamics

After segmenting and estimating the derivatives at each point in each segment of the counterexamples, we start treating the segments one by one to incrementally learn the continuous dynamics. Figure 1 summarizes the algorithm of updating the continuous dynamics of the learned model to fit a given set of counterexamples, and Algorithm 2 shows the detailed procedure. For each segment s of the counterexample trajectories, we have to perform two checks. The *first check* aims to identify whether we need to update the continuous dynamics to fit this segment, or whether an

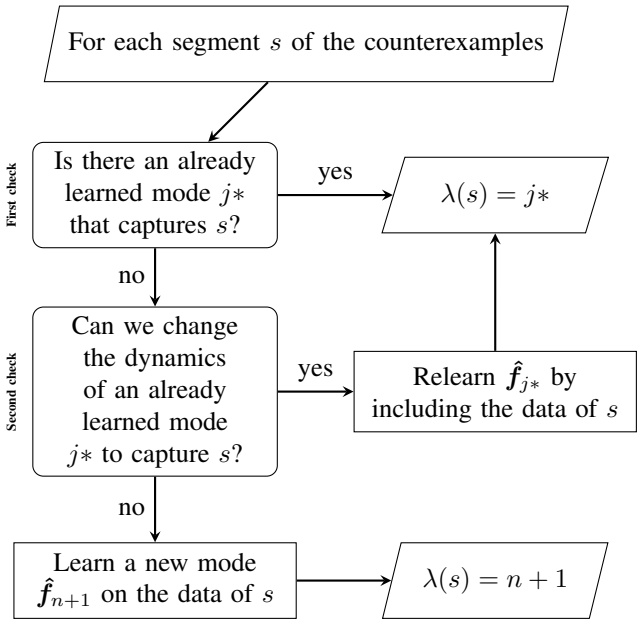


Fig. 1: A flowchart of our algorithm on updating the continuous dynamics of the learned model, given a segmented set of counterexample trajectories.

already learned mode can capture it. In case a modification of the dynamics is needed, the *second check* aims to find out whether we need to create a new mode to fit this segment, or whether we can include it in the training data of an already learned mode.

A. First check: Is there an already learned mode that can capture this segment?

This check tells us whether the current learned continuous dynamics are sufficiently good to capture segment s and hence need no update. The idea is to calculate the derivatives, predicted by the learned model, at points in this segment for each mode by computing $\hat{\boldsymbol{f}}_j(\sigma^{-1}(s))$ for all $j \in \{1, \ldots, n\}$. The next step is to compare each of these predicted derivatives to the derivatives estimated from the SUL trajectory data points, that is $\mathbf{der}(\sigma^{-1}(s))$. If there is a mode $j*$ that minimizes the average relative difference between the predicted derivatives $\hat{\boldsymbol{f}}_{j*}(\sigma^{-1}(s))$ and the estimated derivatives $\mathbf{der}(\sigma^{-1}(s))$ under a certain tolerance, we conclude that the mode of s is $j*$, and hence we do not need to update the continuous dynamics for s and can pass on to the next segment. This first check is summarized in lines 6-10 of Algorithm 2.

B. Second check: Can we modify an already learned mode to capture this segment?

The second check aims to decide whether we could change the dynamics of an already discovered mode to include the points of s. We do this by trying to fit a regression model that combines s with the data of a previously learned mode. Concretely, for each mode j, we fit a multivariate polynomial regression on the data of this mode $((\lambda \circ \sigma)^{-1} j, \mathbf{der}((\lambda \circ \sigma)^{-1} j)$ concatenated with the data of segment s, in order to learn a new function $\tilde{\boldsymbol{f}}_j$. In a similar fashion to the first check, we compare the predicted derivatives of data in segment s to its estimated derivatives to see whether this new function is a good fit for segment s. On top of that, we check whether this new function still fits points belonging to mode j. This last check is important since if we want to update an already learned mode to capture s, we have to make sure that this update still capture well the data already assigned to this mode.

In the event where there is a mode $j*$ that, when fitting a new function that learns its data augmented with the data of the segment, can capture both the data of segment s as well as its original data up to a certain tolerance, we conclude that segment s belongs to mode $j*$, and that the dynamics of this mode are updated to this new function $\tilde{\boldsymbol{f}}_j$. Lines 12-23 summarize this second check.

Algorithm 2 Updating continuous dynamics.

Input: σ the partial function mapping data points to their corresponding segments, λ the partial function mapping segments to their corresponding modes, $\mathfrak{F}$ the current continuous dynamics.

Output: updated λ containing modes of previously unidentified segments, as well as the potentially updated continuous dynamics $\mathfrak{F}$.

1: **procedure** UPDATECONTINUOUS(σ, λ, $\mathfrak{F}$)
2: $\{\hat{\boldsymbol{f}}_1, \ldots, \hat{\boldsymbol{f}}_n\} \leftarrow \mathfrak{F}$ ▷ *this defines all $\hat{\boldsymbol{f}}_j$ as well as n*
3: $\mathcal{S} \leftarrow \mathbf{range}(\sigma) \setminus \mathbf{dom}(\lambda)$ ▷ *all segments with no assigned modes*
4: **for all** $s \in \mathcal{S}$ **do**
5: $\boldsymbol{X}_s \leftarrow \sigma^{-1}(s)$; $\dot{\boldsymbol{X}}_s \leftarrow \mathbf{der}(\sigma^{-1}(s))$
6: $RDs \leftarrow ()$
7: **for all** $j \in \{1, \ldots, n\}$ **do**
8: $RDs \leftarrow RDs \oplus \mathrm{RD}(\hat{\boldsymbol{f}}_j(\boldsymbol{X}_s), \dot{\boldsymbol{X}}_s)$ ▷ *$\oplus$ is a concatenation operator*
9: **if** $\min RDs \leq tol$ **then** ▷ *first check*
10: $\lambda(s) \leftarrow \arg\min RDs$
11: **else**
12: $RDs \leftarrow ()$
13: **for all** $j \in \{1, \ldots, n\}$ **do**
14: $\mathfrak{X}_j \leftarrow (\lambda \circ \sigma)^{-1}(j)$; $\dot{\mathfrak{X}}_j \leftarrow \mathbf{der}(\mathfrak{X}_j)$
15: $\boldsymbol{X}_{\text{aug}} \leftarrow \mathfrak{X}_j \oplus \boldsymbol{X}_s$; $\dot{\boldsymbol{X}}_{\text{aug}} \leftarrow \dot{\mathfrak{X}}_j \oplus \dot{\boldsymbol{X}}_s$
16: $\tilde{\boldsymbol{f}}_j \leftarrow \text{POLYREGRESSION}(\boldsymbol{X}_{\text{aug}}, \dot{\boldsymbol{X}}_{\text{aug}})$
17: $\dot{\boldsymbol{X}}_{\text{pred}} \leftarrow \tilde{\boldsymbol{f}}_j(\boldsymbol{X}_s)$; $\dot{\mathfrak{X}}_{\text{pred}} \leftarrow \tilde{\boldsymbol{f}}_j(\mathfrak{X}_j)$
18: $RD_{\text{seg}} \leftarrow \mathrm{RD}(\dot{\boldsymbol{X}}_{\text{pred}}, \dot{\boldsymbol{X}}_s)$
19: $RD_{\mathfrak{X}} \leftarrow \mathrm{RD}(\dot{\mathfrak{X}}_{\text{pred}}, \dot{\mathfrak{X}}_j)$
20: $RDs \leftarrow \max\{RD_{\text{seg}}, RD_{\mathfrak{X}}\}$
21: **if** $\min RDs \leq tol$ **then** ▷ *second check*
22: $\lambda(s) \leftarrow \arg\min RDs$
23: $\hat{\boldsymbol{f}}_{\lambda(s)} \leftarrow \tilde{\boldsymbol{f}}_{\lambda(s)}$
24: **else** ▷ *creating a new mode*
25: $\hat{\boldsymbol{f}}_{n+1} \leftarrow \text{POLYREGRESSION}(\boldsymbol{X}_s, \dot{\boldsymbol{X}}_s)$
26: $\lambda(s) \leftarrow n + 1$
27: $n \leftarrow n + 1$
28: **return** λ, $\{\hat{\boldsymbol{f}}_1, \ldots, \hat{\boldsymbol{f}}_n\}$

Finally, in case there is no existing mode j that can capture the segment s, we conclude that this segment belongs to a new mode. In this event, we increase the number of modes by one and train a multivariate polynomial regression on the data of s as a first guess for the dynamics of this new mode, as shown in lines 24-27.

Before continuing, we remark that in [17, 18] mode grouping is done using dynamic time warping (DTW) distance between trajectories. The DTW distance might be misleading when trajectories from two different continuous dynamics may be similar in shape in some subsets of the state space. Our decision for mode updates is similar to the one used in [16] for merging segments if they fit well in a regression model; however, we do this merge incrementally, avoiding retraining the regression when unnecessary.

V. Finding Mode Regions

After finding the modes of each segment of the counterexamples, as well as updating the continuous dynamics, the next step in the algorithm is to find the mode regions $\mathfrak{R}$. Since during the segmentation process, we drop points near the decision boundaries of the mode regions, we observe that augmenting the training data with finely spaced points generated between extremities of consecutive segments provides a better result for the region finding described below. For these additionally generated points, we re-apply segmentation to get a more precise location of the change point.

Given this cumulative data, finding the regions can be phrased as a multi-class classification problem for which we use multinomial logistic regression. In order to deal with polynomial boundaries, we extract polynomial features from the training data similarly to what we did in subsection II-C. Let $\boldsymbol{\Psi}_c(\boldsymbol{x})$ be the vector of monomials of components of $\boldsymbol{x}$ up to a certain degree κ. To handle the linear case, we take $\kappa = 1$ and hence $\boldsymbol{\Psi}_c(\boldsymbol{x}) = \boldsymbol{x}$.

A. Two-mode Systems ($n = 2$)

For two modes, the logistic regression model provides coefficients of the decision boundary that separates the two-mode regions: $\mathcal{B}(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{\Psi}_c(\boldsymbol{x}) + b = 0$ We then deduce the two estimated mode regions:

$$\hat{\mathcal{X}}_1 = \{\boldsymbol{x} \mid \mathcal{B}(\boldsymbol{x}) \leq 0\}\,,\ \hat{\mathcal{X}}_2 = \{\boldsymbol{x} \mid \mathcal{B}(\boldsymbol{x}) \geq 0\}$$

B. Multi-mode Systems ($n > 2$)

In case of more than two modes, the multinomial logistic regression model returns a set of linear coefficients $(\boldsymbol{w}_j, b_j)$ per mode j. The mode of a new point $\boldsymbol{x}$ is deduced as $\arg\max_j\ \boldsymbol{w}_j^T \boldsymbol{\Psi}_c(\boldsymbol{x}) + b_j$.

The regions that construct the state space partition for the modes are deduced as:

$$\hat{\mathcal{X}}_j = \left\{ \boldsymbol{x} \mid \bigwedge_{k \neq j} (\boldsymbol{w}_j - \boldsymbol{w}_k)^T \boldsymbol{\Psi}_c(\boldsymbol{x}) + (b_j - b_k) \geq 0 \right\}$$

Furthermore, the decision boundary between two modes j and k is $(\boldsymbol{w}_j - \boldsymbol{w}_k)^T \boldsymbol{\Psi}_c(\boldsymbol{x}) + (b_j - b_k) = 0$.

One could use linear programming techniques to get rid of redundant or useless hyperplanes that may occur [6].

VI. Results and Discussion

In this section, we illustrate our algorithm on three different examples. Besides showing the effectiveness of our learning approach, each example aims to highlight the importance of a specific step in our training algorithm. We run equivalence queries with 100 trajectories each, and fixed time range and time step for each example. The tolerance for the average relative difference between a SUL trajectory and a learned system trajectory is set to be 0.1.

A. Polynomial Example

We illustrate each step of our algorithm, especially focusing on the role of the second check (see subsection IV-B), on a three-mode example with polynomial continuous dynamics and linear boundaries, which is taken from [16]. For ease of comparison with the learned system, we name the three modes of the system under learning A, B, and C. The system is described by the following dynamics:

If $-x_2 \geq 0$

$$(A)\ \begin{cases} \dot{x}_1 = -7 \\ \dot{x}_2 = -x_1 \end{cases}$$

else if $-x_1 \geq 0$

$$(B)\ \begin{cases} \dot{x}_1 = 0.5{x_1}^2 + 0.5x_2 \\ \dot{x}_2 = -9x_1 + 3 \end{cases}$$

else

$$(C)\ \begin{cases} \dot{x}_1 = 5 \\ \dot{x}_2 = -0.1x_1 - 10 \end{cases}$$

The regions corresponding to these three modes are thus:

$$\begin{aligned} \mathcal{X}_A &= -x_2 \geq 0 \\ \mathcal{X}_B &= \quad x_2 > 0\ \wedge -x_1 \geq 0 \\ \mathcal{X}_C &= \quad x_2 > 0\ \wedge\ \ x_1 > 0 \end{aligned}$$

In order to create a first hypothesis, we consider an initial set of trajectories used for training. These trajectories are shown in Figure 2a. As mentioned in Algorithm 1, the first step in treating any trajectory is to do segmentation. Figure 2b shows the same trajectories after segmentation. The spaces that appear between consecutive segments of the same trajectory are due to the fact that we discard points near change points to ensure that points in one segment belong to a single mode (see subsection II-B). We add numbers to the figure in order to represent the identifiers of each segment.

The next step is to identify and update the learned continuous dynamics by treating each segment at a time. As we start with an empty hypothesis, the data in segment 1 is used to learn the dynamics of a first mode. Subsequent segments are treated as explained in section IV.

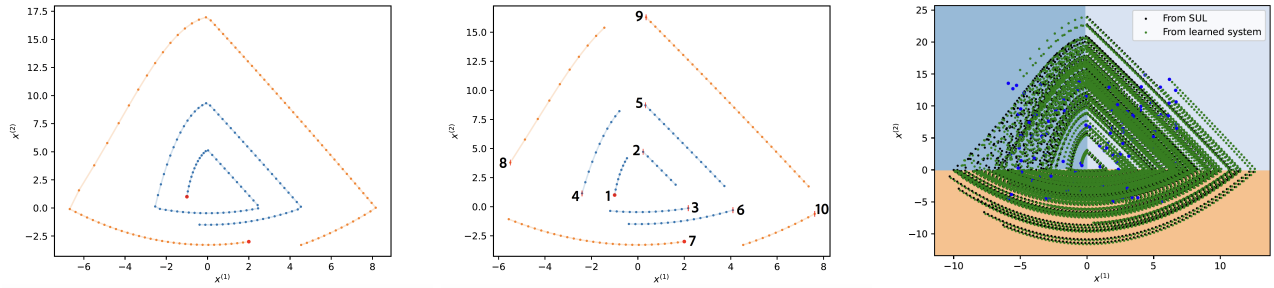
After treating segments 1 through 3, the learning algorithm finds three modes already, whose continuous dynamics, with coefficients rounded to the nearest fourth decimal place, are:

Mode 1:

$$\begin{cases} \dot{x}_1 = 0.03x_1 + 0.492x_2 - 0.002{x_1}^2 + 0.04x_1x_2 - 0.021{x_2}^2 + 0.6 \\ \dot{x}_2 = -8.968x_1 - 0.008x_2 - 0.536{x_1}^2 + 0.042x_1x_2 - 0.023{x_2}^2 + 3.641 \end{cases}$$

Mode 2:

$$\begin{cases} \dot{x}_1 = +5.0 \\ \dot{x}_2 = -0.02x_1 + 0.04x_2 - 0.001x_1x_2 - 10.203 \end{cases}$$



(a) (b) (c)

Fig. 2: (a-b) Two initial training trajectories of the polynomial example, one in blue and one in orange, before and after segmentation. The red dots represent the starting point of each trajectory, the red vertical bars represent the starting points of segments, and the numbers are segment identifiers. (c) The final equivalence query of the polynomial example.

Mode 3:

$$\begin{cases} \dot{x}_1 = -0.025x_1 + 0.001x_2 + 0.01{x_1}^2 - 0.054x_1x_2 + 0.059{x_2}^2 - 7.015 \\ \dot{x}_2 = -1.001x_1 + 0.001{x_1}^2 - 0.002x_1x_2 + 0.007{x_2}^2 - 0.002 \end{cases}$$

Proceeding, the learning algorithm keeps on treating subsequent segments one at a time. Segment 4 fails the first check, as none of the already learned equations captures it. However, when trying to combine its data with that of an already learned mode, it passes the second check, updating the equations of mode 1. Similarly, segments 5 and 6 fail the first check but pass the second one, with segment 5 updating mode 2, and segment 6 updating mode 3. After treating the first six segments, and thus the first trajectory, the learned continuous dynamics are:

Mode 1:

$$\begin{cases} \dot{x}_1 = 0.5x_2 + 0.5{x_1}^2 \\ \dot{x}_2 = -9.0x_1 + 3.0 \end{cases}$$

Mode 2:

$$\begin{cases} \dot{x}_1 = +5.0 \\ \dot{x}_2 = -0.1x_1 - 10.0 \end{cases}$$

Mode 3:

$$\begin{cases} \dot{x}_1 = -0.003x_1 + 0.036x_2 - 0.005x_1x_2 + 0.018{x_2}^2 - 6.989 \\ \dot{x}_2 = -1.0x_1 \end{cases}$$

Moving to the second trajectory, segment 7 also fails the first check, but passes the second concluding that the equations of mode 3 must be updated to:

$$\begin{cases} \dot{x}_1 = -7.001 \\ \dot{x}_2 = -1.0x_1 \end{cases}$$

Segments 8, 9, and 10 subsequently pass the first check, as the now well-learned dynamics of modes 1, 2 and 3 capture them respectively, and hence no update in the continuous dynamics is needed.

After learning the continuous dynamics to fit to the initial trajectories, we train a logistic regression classifier with finely spaced points between segment extremities in order to create a first hypothesis. We run equivalence query estimations with $ncexs$ set to 5, that is, if we find 5 counterexamples, the equivalence query returns them as counterexamples without fully testing 100 trajectories. For this example, three equivalence queries return counterexamples. At each time, the SUL trajectories of the counterexamples are treated to see whether we need to update the continuous dynamics first or the mode regions. In this run, all counterexample trajectories passed the first check on each segment, and hence the continuous dynamics of the first hypothesis were not changed. The counterexample data was used, however, to retrain a better logistic regression model.

The fourth equivalence query returned no counterexamples (Figure 2c), and the average relative difference of compared trajectories of this final equivalence query is 0.0113.

Denote by $\mathcal{B}_{i,j}(\boldsymbol{x}) = 0$ the found classifier boundary between modes i and j, and $\mathcal{B}_{j,i}(\boldsymbol{x}) = -\mathcal{B}_{i,j}(\boldsymbol{x})$. We normalize each boundary by dividing with the coefficient with maximum absolute value, to get:

$$\begin{aligned} \mathcal{B}_{1,2}(\boldsymbol{x}) &= -x_1 - 0.005x_2 + 0.027 \\ \mathcal{B}_{1,3}(\boldsymbol{x}) &= -0.017x_1 + x_2 - 0.060 \\ \mathcal{B}_{2,3}(\boldsymbol{x}) &= \quad 0.013x_1 + x_2 - 0.061 \end{aligned}$$

Consequently, the mode regions of the learned model are derived as in subsection V-B and given by:

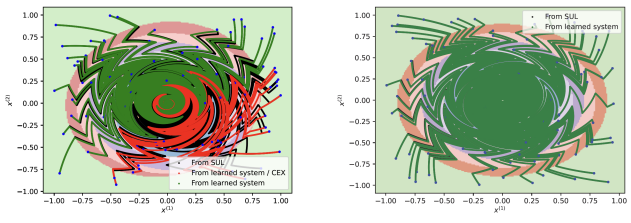
$$\begin{aligned} \hat{\mathcal{X}}_1 &= \mathcal{B}_{1,2}(\boldsymbol{x}) \geq 0 \wedge \mathcal{B}_{1,3}(\boldsymbol{x}) \geq 0 \\ \hat{\mathcal{X}}_2 &= \mathcal{B}_{2,1}(\boldsymbol{x}) \geq 0 \wedge \mathcal{B}_{2,3}(\boldsymbol{x}) \geq 0 \\ \hat{\mathcal{X}}_3 &= \mathcal{B}_{3,1}(\boldsymbol{x}) \geq 0 \wedge \mathcal{B}_{3,2}(\boldsymbol{x}) \geq 0 \end{aligned}$$

Modes 1, 2, and 3 of the learned model correspond respectively to modes B, C, and A of the SUL.

B. Parametric Affine Example

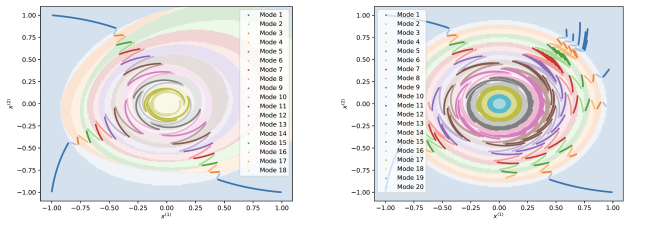
In [28], we construct a system that is parametric in the number of modes, designed to push the boundaries of the learning algorithm by increasing the number of modes in the system. The continuous dynamics of each mode in this example are affine[1], whereas the mode boundaries are described by circles in the state space, all sharing the same origin $O = (0, 0)$.

[1]The continuous dynamics are chosen to be affine in order to easily vary the qualitative behavior of the resulting systems.



(a) (b)

Fig. 3: A test equivalence query of a passively learned 10-mode parametric example with 15 random trajectories vs. the final equivalence query of an actively learned 10-mode parametric example.



(a) For the first hypothesis. (b) For the final hypothesis.

Fig. 4: The training data used to create the first and last hypotheses of the parametric example with 20 modes, along with the found modes of each point.

Given a point $\boldsymbol{x}$, define $d^2(\boldsymbol{x})$ to be the squared distance from the origin O. The mode regions are defined as follows:

$$\mathcal{X}_1 = \left\{ \boldsymbol{x} \mid d^2(\boldsymbol{x}) \geq \left(\frac{n-1}{n} \right)^2 \right\}$$

$$\mathcal{X}_j = \left\{ \boldsymbol{x} \mid \left(\frac{n-j}{n} \right)^2 \leq d^2(\boldsymbol{x}) < \left(\frac{n-j+1}{n} \right)^2 , \ 1<j\leq n \right\}$$

Furthermore, the continuous dynamics in a mode j are:

$$\begin{cases} \dot{x}_1 = -2.5x_1 + \alpha_j x_2 \\ \dot{x}_2 = -\alpha_j x_1 - 2.5x_2 \end{cases}$$

where $\alpha_j = 2j$ if j is odd, otherwise $\alpha_j = -2j$, leading to alternating spiraling towards the center between clockwise and anti-clockwise.

Our previous optimization based approach in [28] was able to learn the parametric example with 5 modes, but no more. Our current approach can correctly learn the example even with 20 modes. We illustrate our algorithm on the parametric example with 10 modes and 20 modes.

1) $n = 10$: In this section, we show the interest of active learning in the choice of useful trajectories for learning. The idea is to actively learn the parametric example with 10 modes, starting from one trajectory to create a first hypothesis, and then stopping the equivalence queries at the first counterexample. This allows us to count exactly how many trajectories were needed in our approach to identify the dynamics of the system.

We run our algorithm, and the ninth equivalence query returns no counterexample (shown in Figure 3b), meaning that we treated 9 trajectories in total to correctly learn the model. The average relative difference between the trajectories of the SUL and those of the learned model in the final equivalence query is 0.0019.

In order to show the interest of active learning, we run our algorithm passively on 15 random trajectories to get a hypothesis. This hypothesis is tested with an equivalence query shown in Figure 3a, which returns 41 counterexamples out of 100 trajectories. We conclude that treating carefully chosen trajectories (nine counterexamples in the case of active learning) provides a model that outperforms a passively learned model with even higher number of initial trajectories. We precise that with 15 trajectories, our algorithm correctly identifies the continuous dynamics, as they are affine. However, as we have a high number of modes and nonlinear boundaries, passive learning with 15 trajectories failed at capturing the mode regions.

2) $n = 20$: We run our algorithm on the parametric example with 20 modes. As we have a large number of modes, we choose to process more counterexamples at a time to avoid retraining the logistic regression model many times. For this, we choose to return an equivalence query on the first 10 counterexamples.

To create an initial hypothesis, we start with three initial trajectories. These trajectories as well as their found modes are shown in Figure 4a. In these trajectories, only 18 modes out of 20 are present. However, as counterexamples from equivalence queries are provided, more modes are discovered and learned, showing the interest of our incremental approach in identifying the continuous dynamics.

For this example, the seventh equivalence query returns no counterexamples, after learning exactly 20 modes and their mode regions. The average relative difference of the final equivalence query is 0.0166. Figure 4b shows the cumulative data (initial trajectories and all subsequent counterexamples) used to learn the model.

VII. Conclusion and Future Work

We present an active learning approach to identifying switched nonlinear dynamical systems with polynomial dynamics. Our approach assumes access to the system under learning as an executable black box, given initial conditions. Our algorithm uses segmentation to detect change points in the trajectories, then it is based on multivariate polynomial regression for fitting the continuous dynamics and logistic regression for the mode regions. We make no assumptions on the number of modes in the system under learning. While our method can be used in a passive setting, we leverage it to the scope of active learning since our continuous dynamics learning is incremental by design, meaning that it can accommodate new data without changing the entire found dynamics whenever possible. By construction, and given assumptions about the accuracy of the multivariate regression fit, we ensure that the average relative distance

between the estimated and predicted derivatives of points in each mode is bounded by a certain tolerance.

We are cognizant that our methodology is not without certain limitations. While our approach works well on many experiments with different characteristics (which are not presented in this paper), we acknowledge that the choice of tolerance for the first and second checks in our algorithm is an important one. On one hand, since there are bounds to the accuracy of multivariate polynomial regression and derivative estimations, choosing a very small tolerance could lead to an overestimation of the number of modes. On the other hand, choosing a very large tolerance could lead to segments belonging to modes with different but similar dynamics to be classified together. A more exhaustive study on the choice of this hyperparameter is needed.

Currently, we retrain the logistic regression classifier from scratch after each equivalence query. We plan to design an incremental classifier that would not need to relearn regions with already well-classified data. To the best of our knowledge, while incremental classifiers exist, they do not give a clear definition of the found mode regions in terms of polytopes in the polynomial space. We think that having a clear definition of the regions is crucial to the explainability of the learned model as well as certain verifications that could be used on it.

In addition to the aforementioned visions, our future work includes extending the learning algorithm to more general forms of hybrid systems, such as hybrid automata with resets and jumps, as well as allowing for input variables.

REFERENCES

[1] S. Paoletti, A. L. Juloski, G. Ferrari-Trecate, and R. Vidal, "Identification of Hybrid Systems A Tutorial," *European Journal of Control*, vol. 13, no. 2-3, pp. 242–260, Jan. 2007.

[2] A. L. Juloski, W. P. M. H. Heemels, G. Ferrari-Trecate, R. Vidal, S. Paoletti, and J. H. G. Niessen, "Comparison of Four Procedures for the Identification of Hybrid Systems," in *Hybrid Systems: Computation and Control*. Springer Berlin Heidelberg, 2005, vol. 3414, pp. 354–369.

[3] R. Vidal, S. Soatto, Yi Ma, and S. Sastry, "An algebraic geometric approach to the identification of a class of linear hybrid systems," in *42nd IEEE Int. Conf. on Decision and Control (IEEE Cat. No.03CH37475)*, vol. 1. IEEE, 2003, pp. 167–172.

[4] N. Ozay, C. Lagoa, and M. Sznaier, "Robust identification of switched affine systems via moments-based convex optimization," in *Proceedings of the 48h IEEE Conf. on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conf.*, 2009, pp. 4686–4691.

[5] G. Ferrari-Trecate, M. Muselli, D. Liberati, and M. Morari, "A clustering technique for the identification of piecewise affine systems," in *Hybrid Systems: Computation and Control*. Springer Berlin Heidelberg, 2001, pp. 218–231.

[6] A. Bemporad, A. Garulli, S. Paoletti, and A. Vicino, "A bounded-error approach to piecewise affine system identification," *IEEE Transactions on Automatic Control*, vol. 50, no. 10, pp. 1567–1580, Oct. 2005.

[7] N. Ozay, "An exact and efficient algorithm for segmentation of arx models," *2016 American Control Conf. (ACC)*, pp. 38–41, 2016.

[8] G. Berger, M. Narasimhamurthy, K. Watanabe, M. Lahijanian, and S. Sankaranarayanan, "An algorithm for learning switched linear dynamics from data," in *Advances in Neural Information Processing Systems*, vol. 35. Curran Associates, Inc., 2022, pp. 30 419–30 431.

[9] A. Bemporad, J. Roll, and L. Ljung, "Identification of hybrid systems via mixed-integer programming," in *Proceedings of the 40th IEEE Conf. on Decision and Control (Cat. No.01CH37228)*, vol. 1, 2001, pp. 786–792 vol.1.

[10] W. Jin, A. Aydinoglu, M. Halm, and M. Posa, "Learning linear complementarity systems," *CoRR*, vol. abs/2112.13284, 2021.

[11] D. L. Ly and H. Lipson, "Learning symbolic representations of hybrid dynamical systems," *Journal of Machine Learning Research*, vol. 13, no. 115, pp. 3585–3618, 2012.

[12] A. Juloski, S. Weiland, and W. Heemels, "A bayesian approach to identification of hybrid systems," *IEEE Transactions on Automatic Control*, vol. 50, no. 10, pp. 1520–1533, 2005.

[13] I. Lamrani, A. Banerjee, and S. K. S. Gupta, "Hymn: Mining linear hybrid automata from input output traces of cyber-physical systems," in *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, 2018, pp. 264–269.

[14] M. G. Soto, T. A. Henzinger, and C. Schilling, "Synthesis of hybrid automata with affine dynamics from time-series data," in *Proceedings of the 24th Int. Conf. on Hybrid Systems: Computation and Control*, May 2021, p. 1–11.

[15] X. Yang, O. A. Beg, M. Kenigsberg, and T. T. Johnson, "A framework for identification and validation of affine hybrid automata from input-output traces," *ACM Transactions on Cyber-Physical Systems*, vol. 6, no. 2, p. 1–24, Apr 2022.

[16] X. Jin, J. An, B. Zhan, N. Zhan, and M. Zhang, "Inferring Switched Nonlinear Dynamical Systems," *Formal Aspects of Computing*, vol. 33, no. 3, pp. 385–406, Jun. 2021.

[17] A. Gurung, M. Waga, and K. Suenaga, *Learning Nonlinear Hybrid Automata from Input–Output Time-Series Data*, ser. Lecture Notes in Computer Science. Springer Nature Switzerland, 2023, vol. 14215, p. 33–52.

[18] I. Saberi, F. Faghih, and F. S. Bavil, "A passive online technique for learning hybrid automata from input/output traces," *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 1, pp. 9:1–9:24, Oct 2022.

[19] D. Angluin, "Learning regular sets from queries and counterexamples," *Information and Computation*, vol. 75, no. 2, p. 87–106, Nov. 1987.

[20] B. Aichernig, A. Pferscher, and M. Tappler, "From passive to active: Learning timed automata efficiently," in *NASA Formal Methods - 12th Int. Symposium, NFM 2020, Proceedings*, ser. Lecture Notes in Computer Science, vol. 12229. Springer, Aug. 2020, pp. 1–19.

[21] L. Henry, T. Jéron, and N. Markey, "Active learning of timed automata with unobservable resets," in *Formal Modeling and Analysis of Timed Systems*. Springer Int. Publishing, 2020, pp. 144–160.

[22] R. Xu, J. An, and B. Zhan, "Active learning of one-clock timed automata using constraint solving," in *Automated Technology for Verification and Analysis*. Springer Int. Publishing, 2022, pp. 249–265.

[23] O. Sankur, "Timed automata verification and synthesis via finite automata learning," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science, vol. 13994. Springer, 2023, pp. 329–349.

[24] M. Waga, "Active learning of deterministic timed automata with myhill-nerode style characterization," in *Computer Aided Verification - 35th Int. Conf., CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part I*, ser. Lecture Notes in Computer Science, vol. 13964. Springer, 2023, pp. 3–26.

[25] A. J. Wagenmaker and K. G. Jamieson, "Active learning for identification of linear dynamical systems," in *Annual Conf. Computational Learning Theory*, 2020.

[26] M. García Soto, T. A. Henzinger, C. Schilling, and L. Zeleznik, "Membership-based synthesis of linear hybrid automata," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science. Springer Int. Publishing, 2019, p. 297–314.

[27] B. K. Aichernig, R. Bloem, M. Ebrahimi, M. Horn, F. Pernkopf, W. Roth, A. Rupp, M. Tappler, and M. Tranninger, "Learning a behavior model of hybrid systems through combining model-based testing and machine learning," in *Testing Software and Systems*, C. Gaston, N. Kosmatov, and P. Le Gall, Eds. Cham: Springer International Publishing, 2019, pp. 3–21.

[28] H. Dayekh, N. Basset, and T. Dang, "Hybrid system identification through optimization and active learning," 2024, to appear in ADHS'24.

[29] T. Sauer, *Numerical Analysis*. Pearson Education, 2012.