Volume **X**, Number **0X**, XX **200X** pp. **X–XX**

LEARNING DYNAMICAL SYSTEMS USING LOCAL STABILITY PRIORS

Arash Mehrjou*

Empirical Inference Department of Max Planck Institute for Intelligent Systems
Tübingen, Germany, and
Department of Computer Science, ETH Zürich
Zürich, Switzerland
amehrjou@tuebingen.mpg.de

Andrea Iannelli

Automatic Control Laboratory, ETH Zürich
Zürich, Switzerland
iannelli@control.ee.ethz.ch

Bernhard Schölkopf

Empirical Inference Department of Max Planck Institute for Intelligent Systems
Tübingen, Germany
Department of Computer Science, ETH Zürich
Zürich, Switzerland
bs@tuebingen.mpg.de

Abstract. A computational approach to simultaneously learn the vector field of a dynamical system with a locally asymptotically stable equilibrium and its region of attraction from the system's trajectories is proposed. The nonlinear identification leverages the local stability information as a prior on the system, effectively endowing the estimate with this important structural property. In addition, the knowledge of the region of attraction can be used to design experiments by informing the selection of initial conditions from which trajectories are generated and by enabling the use of a Lyapunov function of the system as a regularization term. Simulation results show that the proposed method allows efficient sampling and provides an accurate estimate of the dynamics in an inner approximation of its region of attraction.

1. **Introduction.** Learning ordinary differential equations (ODE) of a dynamical system given observed trajectories is the main goal of system identification [18] and the branches of machine learning that are concerned with dynamical systems [7]. To achieve a satisfactory accuracy, the importance of exploiting prior knowledge of the system, especially in a nonlinear context, is recognized [30]. A distinctive property of nonlinear systems is that stability is no more a feature associated with the whole system but with each of its attractors [15]. Moreover, stability is in the general case a local concept, which might only hold in regions surrounding the attractor. This is the case, for example, for the region of attraction (RoA) of equilibria [6] and region

2020 *Mathematics Subject Classification.* Primary: 93D05, 93B30; Secondary: 62J02.

Key words and phrases. Region of Attraction, Identification with priors, Lyapunov theory, Regularization, Nonlinear regression.

* Corresponding author: amehrjou@tuebingen.mpg.de.

of contraction of limit cycles [10]. We consider the former in this work. Given its importance in describing the qualitative properties of a system, local stability is a key structural constraint to encode in a nonlinear system identification algorithm.

In the realm of machine learning, learning ODEs is closely related to training recurrent models. Hence, a stable vector field is potentially a more desirable recurrent learning machine that is robust against vanishing or exploding learning signals. Incorporating *global* stability information has been proposed to obtain more stable training dynamics. The notion of contracting stability is used in [26] to avoid vanishing and exploding gradients in Recurrent Neural Networks (RNNs). The stability is enforced by projecting the model parameters onto a contracting stable model via solving a convex optimization problem. A less conservative parameter set is proposed in [27] by taking into account the structural information of RNNs that are formed by linear functions followed by nonlinear activations. An alternative approach is to restrict the search space for the target ODE to a set of stable systems [33]. However, this approach is limited to vector fields that only have a single globally stable attractor, which is generally not the case.

Our work focuses on jointly learning the system dynamics and its region of attraction. Broadly speaking, the proposed methodology is articulated around two phases that take place in parallel and are entirely data-driven. In one phase, given trajectories of the system, the stability information is distilled into a Lyapunov function that is estimated by a neural network. The other phase uses the best estimate of the Lyapunov function together with a possibly different set of system trajectories to learn the dynamics.[1].

Related works: Learning ODEs, Lyapunov functions, and region of attractions have been studied separately in the literature. Learning ODEs in Reproducing Kernel Hilbert Space (RKHS), initially developed in the machine learning community, has been later widely adopted in other fields, e.g. system identification [23] and transfer operators learning [19].

Even though the stability information can be enforced in linear systems by learning the impulse response in an L^1 RKHS induced by an integrable kernel [3], it is not straightforward for nonlinear systems to impose the Lyapunov notion of stability by the choice of a kernel. In addition to RKHS methods, neural networks have also been employed as function approximators to learn ODEs [29]. Recently, a new class of neural networks has been proposed where the time steps are modeled as the layers of the network. Hence, continuous dynamics correspond to a network with infinitely many layers. The trained neural network then corresponds to the learned ODE [5].

Computing a Lyapunov function for an asymptotically stable equilibrium is traditionally carried out using knowledge of the vector field f. For example, [9] uses Radial Basis Functions to estimate the Lyapunov function. In [13] a compositional approach based on deep neural networks has been proposed and we refer the reader to the comprehensive review [11] for model-based approaches to the computation of Lyapunov functions. Closer to our work is [12], where system trajectories are first used to approximate the ODE and then to estimate the Lyapunov function. The main drawback of this approach is that any inaccuracy in estimating the ODE propagates to the estimation of the Lyapunov function. As discussed later, our coupled ODE&RoA learning method does not suffer from this one-directional flow

[1]Supporting information and code: https://sites.google.com/view/learnstableode

of information that may cause accumulation of error in the computation of a Lyapunov function of the system. While most approaches in the RoA literature are model-based [6, 35, 14], recent works have also considered purely data-driven methods. A sampling strategy is proposed in [20] to estimate the RoA in real-time. A probabilistic method is used in [1] to safely sample and learn the RoA for systems with uncertainty. The RoA estimation module of our algorithm is inspired by [28] where a neural model was proposed to learn the Lyapunov function and gradually change it such that its level sets become closer to the region of attraction of the system.

Contribution: We propose an iterative learning strategy whereby the ODE and RoA of a dynamical system are learned from observed trajectories. We use multilayer perceptrons (MLP) as universal function approximators to learn a sequence of Lyapunov functions whose level sets get closer to the true RoA throughout learning. The ODE is learned using general function approximators, namely RKHS and Neural Nets. Learning the RoA and the ODE are interlaced and inform each other through an iterative algorithm that is shown to be more efficient than learning them separately. The advantage of the Lyapunov function for learning the ODE is twofold. First, it can be used to frugally sample initial conditions for the (numerical or real) experiments. Second, it can be used to regularize the ODE learning process towards the vector fields for which the level sets of the current Lyapunov function are inner estimates of the true RoA. The approach proposed here differs from prior works in the literature as follows. First, the estimation of the Lyapunov function is purely data-driven, hence a potential error in the ODE estimation is not propagated to the estimated Lyapunov function. Second, the direction of information is from the Lyapunov function to the ODE unlike [12] and other model-based approaches where an initially learned ODE is used to estimate the Lyapunov function. Third, the iterative nature of the method actively guides experimentation and determines from where new trajectories should be initiated, rather than relying on a set of passively collected trajectories. In this sense, the proposed method can be considered as an example of active learning. The newly developed framework is finally supported by extensive simulations carried out on the Van der Pol oscillator benchmark. A pictorial representation of the outcome of the algorithm is presented in Figure 1.

2. Co-learning RoA and ODE.

2.1. Problem statement.

Consider an autonomous nonlinear system of the form:

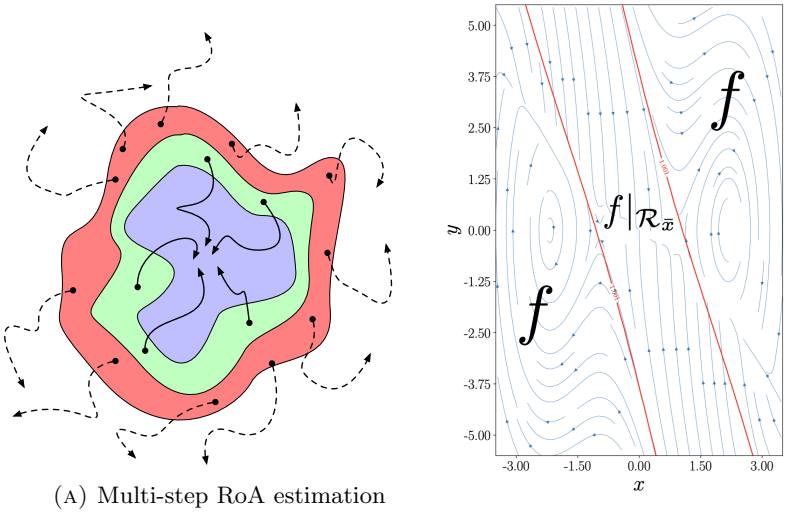
$$\dot{x} = f(x), \quad x(0) = x_0, \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}^n$ is the vector field. The vector $\bar{x} \in \mathbb{R}^n$ is an equilibrium point of (1) if $f(\bar{x}) = 0$. We consider the case where $\bar{x}$ is an *asymptotically* stable equilibrium. Let $\phi(t, x_0)$ denote the flow associated with (1)W, i.e. the solution of (1) at time t with initial condition x_0. The region of attraction (RoA) associated with $\bar{x}$ is defined as [15]

$$\mathcal{R}_{\bar{x}} := \big\{x_0 \in \mathbb{R}^n : \lim_{t\to\infty} \phi(t, x_0) = \bar{x}\big\}. \tag{2}$$

That is, $\mathcal{R}_{\bar{x}}$ is the set of all initial states that eventually converge to $\bar{x}$.

We aim to build a data-driven algorithm that jointly estimates two objects: **1**) The vector field restricted to the RoA denoted by $f_{\bar{x}} := f|_{\mathcal{R}_{\bar{x}}}$. **2**) A Lyapunov function V whose level sets approximate $\mathcal{R}_{\bar{x}}$.



(A) Multi-step RoA estimation

(B) Restricted vector field to the RoA

FIGURE 1. (a) One step of the iterative RoA estimation algorithm. The boundary of the RoA is learned as a boundary function of a classifier. The blue area is the current estimate of the RoA. The green area contains initial states that converge to the estimated RoA and eventually the equilibrium. The red area contains the initial states that diverge. Each step of the algorithm learns the boundary between stable and unstable regions. (b) For the vector field f, $f|_{\mathcal{R}_{\bar{x}}}$ denotes its restriction to the RoA, which is the objective of the ODE learning phase.

In a continuous-time setting, ODE estimation can be seen as a regression problem from x_t to $y_t = \dot{x}_t$. We assume the sensors only measure the states, but not their time derivative. Hence, $\dot{x}_t$ must be estimated from state measurements. We also consider the realistic scenario where the sensors corrupt the state measurement with independent Gaussian noise, i.e. the algorithm has access to $\tilde{x}_t = x_t + \epsilon_t$,

where ϵ_t is identically and independently sampled from a normal distribution $\mathcal{N}(0, \sigma^2)$.

Let $\tilde{x}_{\bar{t}}$ denote a set of state measurements obtained by sampling the flow function $\phi(t, x_0)$ at times $\bar{t} = (t_1 < t_2 < \ldots < t_r)$ indexed by $\bar{r} := (1, 2, \ldots, r)$. Hence $\tilde{x}_{\bar{t}} = (\tilde{x}_{t_1}, \tilde{x}_{t_2}, \ldots, \tilde{x}_{t_r})$ where $\tilde{x}_{t_k} = \phi(t_k; x_0) + \epsilon_{t_k}$. Let S be the sampling operator defined as $S_{\bar{t}}\phi(\cdot; x_0) := (\phi(t_1, x_0), \phi(t_2, x_0), \ldots, \phi(t_r, x_0))$. The noisy state measurements $\tilde{x}_{\bar{t}}$ starting from different initial conditions are the main source of information at disposal to learn the ODE. Letting the system evolve from a specified initial condition is the experiment with which this information is extracted. In the adopted notation a parenthesized superscript refers to a trajectory and an unparenthesized superscript refers to a particular state dimension. Therefore, $x_{\bar{t}}^{(j)} \in \mathbb{R}^{n \times r}$ is the j^{th} trajectory starting from $x_0^{(j)} \in \mathbb{R}^n$ and $x_{\bar{t}}^{(j),s} \in \mathbb{R}^{1 \times r}$ is the trajectory of its s^{th} state dimension.

The total number of available trajectories is limited by a defined budget, which has to do with e.g. the cost of experiments. In addition to the quantity, the locations of the initial conditions might also be constrained by physical limitations as the system has to be safely operated. Because the system is nonlinear, trajectories of

equal length are not equally informative about the dynamics, hence the choice of initial states plays the role of *experiment design*. This is indeed a key enabler in our work, and it has not been fully exploited in the literature. See [16] where the rich spectrum of nonlinear responses associated with different initial conditions is leveraged for identifying the Koopman operator.

Given an equilibrium $\bar{x}$, we recall that the objective is to learn $f_{\bar{x}}$ instead of f, which can be a much more challenging task and, in many engineering applications, not necessary, as the system will be operated around one of its stable attractors. Hence, the region of attraction $\mathcal{R}_{\bar{x}}$ is a privileged subset of the state space to sample the trajectories from. Specifically, two reasons are emphasized: **1)** Frugality (i.e., estimating $f_{\bar{x}}$ using the least number of trajectories). This is encouraged by the fact that trajectories inside $\mathcal{R}_{\bar{x}}$ will never leave it and are most informative about $f_{\bar{x}}$. **2)** Safety (i.e., generated trajectories will not diverge). This property is guaranteed by the definition of the RoA of an equilibrium.

When $\mathcal{R}_{\bar{x}}$ is not given (that is the case in most practical situations, especially when f is not known), it also needs to be estimated from the trajectories of the system. Standard approaches to compute inner (i.e. guaranteed) estimates $\hat{\mathcal{R}}_{\bar{x}}$ of the true RoA $\mathcal{R}_{\bar{x}}$ are model-based, i.e. they require the knowledge of f. While this is not the case here, a well-known fact from the literature often used in model-based approach will be leveraged here.

Lemma 1. [15] *Let $\mathcal{D}$ be an open subset of $\mathbb{R}^n$ and $\bar{x} \in \mathcal{D}$. Suppose there exists a 1-time continuously differentiable function $V : \mathbb{R}^n \to \mathbb{R}$ such that:*

$$\begin{aligned} &V(\bar{x}) = 0 \quad \text{and} \quad V(x) > 0, && \forall x \in \mathcal{D}\backslash\bar{x}, \\ &\langle \nabla V(x), f(x) \rangle < 0, && \forall x \in \mathcal{D}\backslash\bar{x}, \\ &\Omega_{V,\gamma} := \{x \in \mathbb{R}^n : V(x) \leq \gamma\}, && \Omega_{V,\gamma} \subseteq \mathcal{D}, \end{aligned} \tag{3}$$

and $\Omega_{V,\gamma}$ is bounded. Then, $\hat{\mathcal{R}}_{\bar{x}} := \Omega_{V,\gamma} \subseteq \mathcal{R}_{\bar{x}}$.

Notice that $\langle \nabla V(x), f(x) \rangle$ represents the Lie derivative of V, that is the derivative of V along the trajectories of f. When f is known, a common approach to compute $\hat{\mathcal{R}}_{\bar{x}}$ is via Sums of Squares (SOS) optimization [21, 6], whereby one finds polynomial functions that satisfy set containment conditions (3)). Depending on the chosen degree of the polynomial Lyapunov function, the estimates in SOS-based approaches might be quite conservative, i.e., the true RoA $\mathcal{R}_{\bar{x}}$ is much larger than the estimated one $\hat{\mathcal{R}}_{\bar{x}}$.

To overcome this issue, in Section 2.2 we propose to leverage the universal approximation property of deep neural networks to compute Lyapunov functions whose largest contractive level set $\Omega_{V,\gamma^\star}$ approximates the shape of $\mathcal{R}_{\bar{x}}$.

Section 2.3 details the twofold role of the estimated RoA: guiding experimentation to produce most suitable trajectories to learn $f_{\bar{x}}$ and also encoding an appropriate local stability prior that facilitates learning $f_{\bar{x}}$. Finally, in Section 2.4 the coupled algorithm to jointly learn $\mathcal{R}_{\bar{x}}$ and $f_{\bar{x}}$ is presented. As the estimated RoA gets closer to the true one, more informative trajectories are sampled which yield a better estimate of the vector field. Moreover, training is also regularized by the estimated Lyapunov function.

2.2. Estimating RoA from trajectories.

Let $V(\cdot;\theta) : \mathbb{R}^n \to \mathbb{R}_+$ be a candidate Lyapunov function for system (1) parameterized by θ. As required by the definition of the Lyapunov function, $V(\cdot;\theta)$ has to be *positive definite* in some neighbourhood

Algorithm 1: The coupled algorithm to learn RoA and ODE concurrently

input :

- f: The oracle that generates trajectories from the system
- $\mathcal{V}_{\text{init}}$: Initial inner estimate of the RoA
- T: length of the trajectories

output:

- $V(\cdot;\theta), c$: The Lyapunov function V and its level value c that estimates the true ROA
- $f(\cdot;\psi)$: The estimate of the dynamics function

init $V(\cdot;\theta)$ to the initial inner estimate of ROA
init ψ by a zero-mean Gaussian distribution with standard deviation of 0.1
set the interpolant kernel function $k(\cdot,\cdot)$
set the growth coefficient α
set the growth threshold $\mathcal{T}_g$
set Interpolants $\leftarrow$ [ ]
Set $\mathcal{G} \leftarrow \mathcal{S}_{\alpha c} \backslash \mathcal{S}_c$
while vol$(\mathcal{G}) > \mathcal{T}_g$ **do**
  $\{x^{(j)}\}_{j=1}^J \leftarrow$ Sample J initial points from $\mathcal{G}$
  $\{x_{[T]}^{(j)}\}_{j=1}^J \leftarrow$ Trajectories starting from $\{x^{(j)}\}_{j=1}^J$
  $\Phi \leftarrow$ compute interpolants from $\{x_{[T]}^{(j)}\}_{j=1}^J$ and kernel k by Algorithm 2
  Interpolants $\leftarrow$ Interpolants $\cup \{\hat{x}^{(j)}(\cdot;\Phi^{(j)})\}_{j=1}^J$
  $\psi \leftarrow$ Update ODE using the interpolants and $V(\cdot;\theta)$ by solving Equation (13)
  $\theta, c \leftarrow$ Grow RoA using the gap $\mathcal{G}$ and the current $V(\cdot;\theta)$ by the method explained in Section 2.2
  $\mathcal{G} \leftarrow \mathcal{S}_{\alpha c}(V(\cdot;\theta)) \backslash \mathcal{S}_c(V(\cdot;\theta))$
end

around the equilibrium point. The function $V : U \subseteq \mathbb{R}^n \to \mathbb{R}$ is said to be positive definite in the neighbourhood U if **(1)** $V(0) = 0$ and **(2)** $V(x) > 0$ for all $x \in U$ such that $x \neq 0$

To ensure $V(\cdot;\theta)$ is positive definite, instead of parameterizing it directly, it is modelled as the inner product of a feature extractor $v(\cdot;\theta)$ with itself, i.e., $V(\cdot;\theta) = v^\mathsf{T}(\cdot;\theta)v(\cdot;\theta)$ where $v(\cdot;\theta) : \mathbb{R}^n \to \mathbb{R}^n$ is a multilayer perceptron. We will denote by $\mathcal{S}_c(V) := \{x \in \mathbb{R}^n : V(x;\theta) < c\}$ the sublevel set (interior of a level set) of V with level value c. The goal is to find θ and c such that $\mathcal{S}_c(V(\cdot;\theta))$ is a good approximation of the true RoA $\mathcal{R}_{\bar{x}}$. The parameter θ characterizes the shape of a level set while c determines its size. Recalling the definition of the RoA in (2) and the result of Lemma 1, the following multi-step supervised learning approach is considered.

1. The Lyapunov function is initialized to a quadratic function with the loss function
$$\theta_0^* = \underset{\theta}{\operatorname{argmin}}\, \mathbb{E}_{B_{r_0}}[\|V(x;\theta) - x^\mathsf{T}Qx\|^2] \tag{4}$$
where $\mathbb{E}_{B_{r_0}}$ represents the empirical expectation on samples that are uniformly drawn from the unit ball $B_{r_0}(\bar{x})$ centered at $\bar{x}$ with radius r_0. Matrix

Q characterizes the target quadratic function that is set to I in the absence of further information. The reason behind this choice for the initial Lyapunov function is the fact that when the equilibirum point is asymptotically stable as assumed here, one can always choose Q as the positive definite matrix which is the unique solution of the Lyapunov equation $A^TQ + QA = -I$, where A represents the Jacobian of the nonlinear vector field linearized at the equilibrium.. After pre-training the Lyapunov network $V(\cdot;\theta)$ to approximate I, a line search over c while checking the Lyapunov decrease condition of Lemma 1 is performed to find the largest level value such that $\mathcal{S}_c(V(\cdot;\theta_0^*))$ remains inside $\mathcal{R}_{\bar{x}}$. The decrease condition is checked on the vertices of a rectangular grid that sample the state space with regular spacing. (See Appendix A.1 for further details about the grid).

2. The sublevel set $\mathcal{S}_c(V)$ is expanded by multiplying c with $\alpha > 1$ to obtain the gap region $\mathcal{G}_\alpha(V,c) := \mathcal{S}_{\alpha c}(V) \backslash \mathcal{S}_c(V)$ (for the sake of a lighter notation, $V(\cdot;\theta)$ will be written as V throughout the text when possible).
3. The system is experimented by drawing J initial states inside $\mathcal{G}_\alpha(V,c)$ that run for a specified number of time steps. Each initial state is given the label $+1$ (stable) if its trajectory enters $\mathcal{S}_c(V)$. Otherwise, it is given the label -1 (unstable). Let $\mathcal{O} : \mathcal{G} \to \{+1,-1\}$ be the oracle function that simulates the system from an initial condition chosen from the gap region $\mathcal{G}$ and assigns the label as mentioned. Ultimately, the experiments will produce a dataset consisting of J pairs $\{(\tilde{x}_{\bar{t}}^{(j)}, \mathcal{O}(x_0^{(j)}))\}_{j=1}^{(J)}$.
4. The parameters θ of the Lyapunov function $V(\cdot;\theta)$ are learned such that the initial states $x_0 \in \mathcal{G}_\alpha(V,c)$ that correspond to $l(x_0) = +1$ ($l(x_0) = -1$) falls inside (outside) $\mathcal{S}_c(V)$. As both θ and c affect the shape of the sublevel set $S_c(V(\cdot;\theta))$, we keep c fixed to 1.0 and only train θ. Minimizing the following loss function serves this purpose as a supervised learning task,

$$\mathcal{L}_\theta = \sum_{x_0 \in \mathcal{G}} \ell(V(x_0;\theta), l(x_0)) + \lambda_{\text{RoA}} 1_{[\mathcal{O}(x_0^{(j)})=+1]} \times \sum_{j=1}^{J} \sum_{k=1}^{r} \left[V\left(x_{t_{k+1}}^{(j)};\theta\right) - V\left(x_{t_k}^{(j)};\theta\right) \right] \tag{5}$$

with $\ell : \mathbb{R} \times \{+1,-1\} \to \mathbb{R}$ defined as

$$\ell(V(x;\theta), \mathcal{O}(x)) = \mathcal{O}(x)(V(x;\theta) - c). \tag{6}$$

With the loss function (5), training the Lyapunov function can be seen as training an energy-based model where, in Figure 1, the energy of the points located in the red (unstable) area increases while the energy of the points located in the green area (stable) decreases. Iterations of Stochastic Gradient Descent (SGD) are run on (5) until a pre-defined tolerance on the loss decrease is met. The solution will be the updated Lyapunov function and consequently its level sets. Here, we provide the intuition behind each term of the loss function of Equation (5). The first term $\ell(V(x_0;\theta), l(x_0))$ is the classifier loss that differentiates between the stable and unstable states (inside and outside the RoA) to push the Lyapunov function in the direction where its level set gives a better estimate of the RoA. The second term is non-zero

only for the states that are inside the current estimate of the RoA. Minimizing $V(x^{(j)}_{t_{k+1}};\theta) - V(x^{(j)}_{t_k})$ over the observed trajectories makes the Lyapunov function decrease over the system's trajectory. This provides an extra learning signal for training the Lyapunov function so that it meets the second condition of lemma 1.

5. Ideally, the value of c must remain fixed and the change of θ must be sufficient to update the Lyapunov function. However, due to practical reasons such as the limited capacity of the neural network and the local minima in optimization, we check the value of c after each iteration to make sure $\mathcal{S}_c(V)$ remains fully within $\mathcal{R}_{\bar{x}}$. To this end, we update the value of c by a line search so that every $x \in \mathcal{S}_c(V(\cdot;\theta))$ satisfies $\langle \nabla_x V(x;\theta), f(x)\rangle < 0$ in accordance with the second condition of Lemma 1. In practice, we don't have access to f. Hence, the empirical version of the decrease condition is tested for the states along the observed trajectories, i.e., $V(x_{t_{k+1}}) - V(x_{t_k}) < 0$. Hence, c is set to a value such that the decrease condition is satisfied for all pairs $(x_{t_k}, x_{t_{k+1}})$ chosen from the observed trajectories for which $x_{t_k}, x_{t_{k+1}} \in \mathcal{S}_c(V(\cdot;\theta))$
6. Go back to step 2 until every point from the produced gap $\mathcal{G}_\alpha(V, c)$ is unstable. Instability is verified when the trajectories do not enter $S_c(V)$ after a specified number of time steps.

2.3. **Learning ODE from trajectories.** In this section, we explain our method to learn the vector field from the observed noisy trajectories. This section is presented as a stand-alone module. The way the ODE learning module is incorporated in the overall coupled algorithm with the RoA estimation module is explained in Section 2.4. We employ a 2-stage approach to learn the ODE. The first step (Section 2.3.1) consists of an RKHS-based interpolation to smooth out the noisy observed trajectories. The second step (Section 2.3.2) learns the ODE function as a regression problem from states to their time derivatives, which are now computed using interpolated smooth trajectories.

It is well known that learning a complex nonlinear function from scarce data is prone to overfitting. If prior information on the functional form of the unknown function is available, a wise practice is to restrict the hypothesis space to the set of functions that is more likely to contain the target function.

2.3.1. *Learning interpolants.* Sensor measurements typically give the noisy states $\tilde{x}_t$ at discrete times. This causes two important issues. First, the states might be only available on irregular (non-equidistant) and possibly sparse time intervals. Second, the noise in the state measurements will be magnified when the time derivative $\dot{\tilde{x}}_t$ is computed by methods such as finite-difference. To overcome both problems, we propose to learn an interpolating function for every trajectory as a function of time.

We describe the interpolation for one trajectory, but the method is the same for every other trajectory. Let $\bar{t} = (t_1 < t_2 < \ldots < t_r)$

be a sequence of r irregular sampling times. Let $\phi(t, x_0)$ be the flow function (trajectory) starting from x_0. Assume $S_{\bar{t}} : (\mathbb{R}_+ \to \mathbb{R}^n) \to \mathbb{R}^n \times \mathbb{R}^r$ is a sampling operator that samples the input function at times $\bar{t}$.

We use the extension of the Shannon sampling theorem to learn the interpolating functions using regularized least squares [8]. We briefly review the preliminary concepts here as they will also be needed when reviewing the RKHS method for learning the vector field.

Let $k : \mathbb{R}_+ \times \mathbb{R}_+ \to \mathbb{R}$ be a continuous symmetric map and $\mathcal{H}_{k,\bar{t}} := \{\sum_{t\in\bar{t}} a_t k(\cdot, t) : t \in \mathbb{R}$ and $|\{a_{t'} : t' \in \mathbb{R}, a_{t'} \neq 0\}| \leq \infty\}$ where $|\cdot|$ is used to denote cardinality of the set. That is, $\mathcal{H}_{k,\bar{t}}$ is a Hilbert space defined as a finite linear combination of $\{k_t : t \in \bar{t}\}$, where only a finite number of a_t are nonzero. An inner product on this space is defined as $\langle k(\cdot, t_1), k(\cdot, t_2)\rangle = k(t_1, t_2)$. If k is a Mercer kernel, H_k corresponds to a reproducing kernel Hilbert space and $\mathcal{H}_{k,\bar{t}}$ is the closed subspace generated by $\{k(\cdot, t), t \in \bar{t}\}$. For the function $\phi(\cdot; x_0) : \mathbb{R}_+ \to \mathbb{R}, \phi(\cdot; x_0) \in \mathcal{H}_{k,\bar{t}}$, the goal is to find $\hat{\phi}(\cdot; x_0) \in \mathcal{H}_{k,\bar{t}}$ from samples $S_{\bar{t}}\phi(\cdot; x_0)$ contaminated with noise such that $\hat{\phi}(\cdot; x_0)$ is a good approximation to $\phi(;x_0)$. Let $k_{\bar{t},\bar{t}}$ denote a matrix defined as $(k_{\bar{t},\bar{t}})_{i,j} = k(t_i, t_j)$.

Assumption 2.1. $k_{\bar{t},\bar{t}}$ is well-defined, bounded, and positive with bounded $k^{-1}_{\bar{t},\bar{t}}$.

This assumption is satisfied for kernels such as polynomial and Gaussian [24], given that all sampled times are distinct in $\bar{t} = (t_1 < t_2 < \ldots < t_r)$.

While $\phi(\cdot; x_0)$ is the function to be reconstructed, we only observe noisy samples from it at times $\bar{t} = (t_1, \ldots, t_r)$ as

$$\tilde{x}_k = \phi(t_k; x_0) + \epsilon_{t_k}, \quad \epsilon_{t_k} \sim \mathcal{N}_{0,\sigma}. \tag{7}$$

Every dimension of the observed trajectories is interpolated separately as a function of time. To control the contribution of each measurement to the learned interpolation, we weight the observations by a scalar weighting function $w : \bar{t} \to \mathbb{R}_+$. In the absence of any prior about the weighting function, *Voronoi tessellation* is a commonly adopted option [31].

Definition 2.2. Let $X \subset \mathbb{R}^n$ be compact. The Voronoi tessellation for a set of distinct points $\mathbf{x} := \{x_i\}_{i=1}^m \in X^m$ is the collection of pairwise disjoint open sets $\mathcal{V}_i(\mathbf{x}), i = 1, \ldots, m$ defined by

$$\mathcal{V}_i(\mathbf{x}) = \{z \in X | \|x_i - z\|_{\mathbb{R}^n} < \|x_j - z\|_{\mathbb{R}^n}, \ \textit{if } i \neq j\} \tag{8}$$

In $\mathbb{R}^n$, the weights are chosen as $w_i = \mu(\mathcal{V}_i(\mathbf{x}))$ where μ is the strictly positive Borel measure on $\mathbb{R}^n$. In $\mathbb{R}$, the measure μ will simply be the length of the intervals. That is, for the sampling times $\bar{t} \subset \mathbb{R}^r_+$, the tesselation takes a simple form as $\mathcal{V}_0(\bar{t}) = [0, t_1]$, $\mathcal{V}_r(\bar{t}) = [t_r - t_{r-1}]$, and $\mathcal{V}_i(\bar{t}) = [\frac{t_{i+1}-t_i}{2} - \frac{t_i - t_{i-1}}{2}]$ for $i = 2, \ldots, r-1$. For every trajectory, the interpolant of the s^{th} state represented by $\hat{\phi}^{(j),s}(\cdot; x_0)$ is obtained by solving the following regularized optimization problem

$$\hat{\phi}^{(j),s} = \underset{\hat{\phi}}{\operatorname{argmin}} \sum_{t\in\bar{t}} w_t(\hat{\phi}(t) - \tilde{x}_t^{(j),s})^2 + \gamma_{\text{traj}}\|\hat{\phi}\|_k \tag{9}$$

that has closed-form solution due to the following theorem.

Theorem 2.3 ([31], Theorem 2)**.** *Assume the flow function is component-wise realizable, i.e., $\phi^s(\cdot, x_0) \in \mathcal{H}_{k,\bar{t}}$ for $s = 1, \ldots, n$. Fix a regularization parameter $\lambda_{traj} > 0$, and define the diagonal weight matrix $D_w = \text{diag}(w_1, w_2, \ldots, w_r) \in \mathbb{R}^{r\times r}$. The approximation of the s^{th} component $\hat{\phi}^s$ to $\phi^s(\cdot, x_0)$ of Equation (7) belongs to $\mathcal{H}_{k,\bar{t}}$ and is represented by*

$$\hat{\phi}^s(\cdot; \alpha) = \sum_{k=1}^{r} a_k k(\cdot, t_k) \tag{10}$$

where the coefficients $\alpha := \{a_i\}_{i=1}^r$ are obtained using the following matrix equation

$$(k_{\bar{t},\bar{t}} D_w k_{\bar{t},\bar{t}} + \gamma_{traj} k_{\bar{t},\bar{t}})\alpha = k_{\bar{t},\bar{t}} D_w \tilde{x}^s_{\bar{t}}. \tag{11}$$

Algorithm 2: Interpolation

input :
- $\{\tilde{x}^{(j)}\}_{j=1}^{J}$: J noisy state trajectories form J different initial points
- kernel function $k(\cdot,\cdot)$

output: Φ consisting of $\hat{\phi}^{(j),s}$ for all s and j

```
for j = 1, ..., J do
    for s = 1, ..., d do
        initialize α in φ̂(·; α) = Σ_{i=1}^{m} α_i k(t, t_i)
        φ̂^{(j),s} ← argmin_{φ̂} the objective function of Equation (9)
    end
end
```

Notice that $\tilde{x}_{\bar{t}}^{s} \in \mathbb{R}^r$ represents the noisy state measurements of the s^{th} state at times $\bar{t}$.

The pseudocode for learning the interpolants for the observed trajectories is presented in Algorithm 2.

The choice of the kernel — The choice of the kernel function plays a crucial role in the quality of the interpolation of trajectories. Because the kernel k determines the members of its associated Hilbert space $\mathcal{H}_{k,\bar{t}}$, a reasonable kernel is the one that ensures $\phi^s(\cdot, x_0) \in \mathcal{H}_{k,\bar{t}}$. It is not straightforward to choose such a kernel in the absence of further information on the shape of trajectories. However, the kernel can be chosen from the class of the so-called *universal kernels* with the property that their associated RKHS is dense in the set of continuous functions $C(Z)$ for every compact set $Z \subset \mathbb{R}^n$ (See Definition 4.52 of [32]).

The Gaussian RBF kernel defined as $k(x,y) = \exp(-\gamma^{-2}\|x-y\|_2^2), \gamma > 0$ is an example of universal kernel and it will be used here by choosing its bandwidth γ with Median heuristic [4]. The chosen bandwidth by this heuristic was about 0.07. Because the input space is assumed to be compact, we make the reasonable technical assumption that the length of every trajectory is bounded by a $\bar{T} \in \mathbb{R}_+$.

Putting together learned interpolants for every dimension of the state vector gives $\hat{\phi} = [\hat{\phi}^1, \phi^2, \ldots, \hat{\phi}^n]$, that is the interpolated trajectory with $\hat{\phi} \in \mathcal{H}_{k,\bar{t}}^n$ and $\hat{\phi}(t) \in \mathbb{R}^n$ for $t \in \mathbb{R}_+$. A similar procedure as above is repeated for every trajectory. Let $j \in J$ be the index that iterates over all the produced trajectories. For the initial state $x_0^{(j)}$, the states are sampled at times $\bar{t}^{(j)}$ to produce the noisy sequence of measurements $\tilde{x}_{\bar{t}^{(j)}}^{(j)}$. Hence, the interpolation phase gives a set of $J \times n$ functions $\{\hat{\phi}^s(\cdot; x_0^{(j)})\}$ where $s = 1, \ldots, n$ is the dimension index of the state vector and $j = 1, \ldots, J$ refers to the j^{th} trajectory started at the initial state $x_0^{(j)}$. Notice that the sampling patterns $t^{(i)}$ and $t^{(j)}$ can be different for $i \neq j$. In addition to addressing the problem of differentiating noisy trajectories by approximating them in an RKHS of smooth functions, it is worth observing that even though state trajectories are only available at discrete times, $\hat{\phi}^{(j)}(t; x_0^{(j)})$ can be evaluated at any time $0 \leq t < \bar{T}$ and be oversampled on demand.

2.3.2. *Learning the vector field.* We use a method known in the literature as *gradient matching* to fit $\hat{f}(\cdot;\psi)$ parameterised by ψ to the time derivative of the interpolants $\hat{\phi}(\cdot;\alpha)$ fitted to the observed trajectories by (9). Since the observed trajectories

are interpolated by functions of a smooth RKHS, computing time derivative can be done analytically as $d\hat{\phi}(s;\alpha)/ds = \sum_{k=1}^{r} a_k \partial k(s,t_k)/\partial s$. Hence, the vector field is learned by solving the optimization problem $\psi^*_{\text{base}} = \text{argmin}_\psi \mathcal{L}_{\text{base}}$ where

$$\mathcal{L}_{\text{base}} = \sum_{j=1}^{J} \sum_{t\in\bar{t}^{(j)}} \| \frac{d\hat{\phi}^{(j)}(t;\alpha^{(j)})}{dt} - \hat{f}(\hat{\phi}^{(j)}(t;\alpha^{(j)});\psi)\|^2 \tag{12}$$

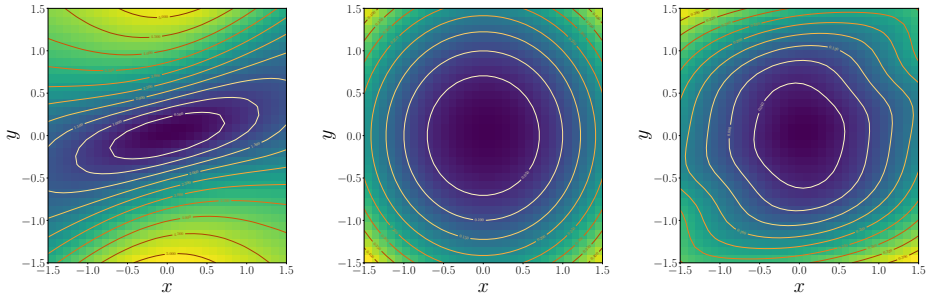
with respect to ψ that parameterises the model of the dynamics. Recall that $\phi^{(j)} = [\phi^{(j),1}, \phi^{(j),2}, \ldots, \phi^{(j),n}]$ where every $\phi^{(j),s}$ is the fitted interpolant (10) to the s^{th} dimension of the j^{th} trajectory. Moreover, the parameters of the interpolants of every state dimension are encapsulated as $\alpha^{(j)} = [\alpha^{(j),1}, \alpha^{(j),2}, \ldots, \alpha^{(j),n}]$. Notice that unlike Equation (9), here we use uniform weighting instead of Voronoi tessellation, which is costly to compute in higher dimensions. The uniform weighting is a valid choice here as the length of the experimented trajectories is chosen not to be too long to ensure the trajectories won't get too close to each other near the equilibrium and provide redundant information about the ODE. In fact, as explained in Item 2 of Appendix A.6, the length of the experimented trajectory is chosen roughly as the time it takes for a trajectory initiated from the surrounding gap to enter the previous estimate of the RoA (e.g. see Figure 4).

Observe that (12) is a typical regression problem for which the temporal order does not matter as long as the interpolated state vector is matched to its time derivative at each point in time. Results obtained by solving this regression problem using both kernel methods and neural networks will be presented to emphasize that the proposed way to incorporate the stability information in the ODE estimation does not depend on the estimation method employed to learn the ODE.

2.4. **A coupled ODE-RoA algorithm.** The information of RoA is leveraged to learn the ODE in two ways: experiment design and regularization. The algorithm proposed in Section 2.2 is a multi-step method that gradually improves the estimate of the RoA by learning a Lyapunov function. We interlace the steps of the ODE learning algorithm between the steps of the RoA estimation to incorporate the stability information when solving (12). The iterative process continues until the RoA cannot be improved further. The estimated ODE will then be an approximation of the true dynamics within the RoA of the equilibrium. The pseudocode of the coupled algorithm is presented in Algorithm 1. Notice that the algorithm continues until the volume of the produced gap is less than a pre-defined threshold $\mathcal{T}_g$.

In the following, two ways in which the stability information contributes to learning the ODE are illustrated.

2.4.1. *Experiment design.* In the experiment design phase, the RoA information is used to choose the initial states from which the trajectories are produced. Since the objective is to estimate $f_{\bar{x}}$ (i.e. the restriction of the vector field to the RoA of $\bar{x}$) and to make efficient use of the experimentation budget, it is not beneficial to choose initial states whose trajectories do not contribute to the accuracy of the estimated ODE within the RoA. Assume $\mathcal{S}_c(V(\cdot;\theta))$ is the current estimate of the RoA. New initial states are chosen randomly from the gap region $\mathcal{G} = \mathcal{S}_{\alpha c}(V)\backslash\mathcal{S}_c(V)$. The estimate $\hat{f}$ of the ODE is then updated with the trajectories that start from newly chosen initial states. To prevent the algorithm from forgetting the vector field in inner areas of the RoA, a mixture of the previously collected trajectories (30%) and new ones (70%) are used to optimize the objective function (12). While in principle



(A) Random network (B) Quadratic (C) Pre-trained

FIGURE 2. Pre-training the randomly initialized neural network with a quadratic function. The background color shows the values of the Lyapunov function evaluated over a fine grid of points. Lighter colors correspond to larger values. The contours show the level sets.

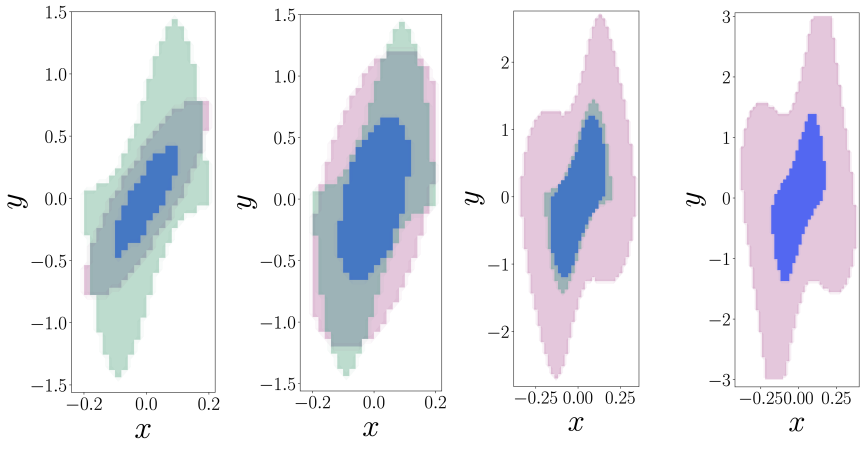
one could use a similar mixture of new and old trajectories to learn the Lyapunov function (see point 4 of Section 2.2), it was found empirically that did not bring any concrete advantage. A more in-depth analysis of this aspect is left for future research.

2.4.2. *Regularization.* In the regularization phase, the Lyapunov function whose level set is the current estimate of the RoA is employed to incentivize learning a stable vector field. We incorporate this structural constraint in the loss function with a Lagrange multiplier. Therefore, (12) is augmented as

$$\mathcal{L}^{\psi}_{\text{reg}} = \mathcal{L}^{\psi}_{\text{base}} + \lambda_{\text{ODE}} \sum_{x \in S_c(V)} \langle \nabla_x V(x;\theta), \hat{f}(x;\psi) \rangle \tag{13}$$

which is minimized with respect to ψ for a fixed θ corresponding to the current step of the Lyapunov / RoA estimation algorithm of Section 2.2. This term encourages the dynamics $\hat{f}$ to be compatible with the negativity condition of the current estimate of the Lyapunov function prescribed by Lemma 1 while the first term is a simple regression loss that concerns how well the model fits to the time deriative of the trajectories. Recall that the input to this algorithm are interpolants that are already fitted to the observed noisy and irregularly sampled trajectories in Section 2.3.1.

The proposed algorithm has a few hyperparameters consisting of the hyperparameters of each of its modules (i.e., RoA estimation and ODE learning) and also the hyperparameters that are introduced when these modules interact in the coupled algorithm. A detailed discussion on how to choose each of these hyperparameters is given in Appendix A.6. Here we provide a brief rationale on their choices. The hyperparameters that affect the function class of the regression tasks (e.g. the bandwidth of the kernel in RKHS regression) are chosen by the median heuristics and cross-validation. The hyperparameters that determine the relative weight of the constituent terms in a loss function are generally chosen in such a way that all terms become of the same order of magnitude. The hyperparameters that determine the number of experimented trajectories and their length are determined by the experimentation budget.



(A) Step 1 (B) Step 4 (C) Step 7 (D) Step 10

FIGURE 3. (Van der Pol Oscillator) The growth stages of the RoA estimation algorithm of Section 2.3. Color codes are as follows. Green: True RoA. Blue: Estimated RoA (the largest sublevel set of the learned Lyapunov function that satisfies the decrease condition (3)). Pink: The gap $\mathcal{G} = \mathcal{S}_{\alpha c}(V(\cdot;\theta))\backslash\mathcal{S}_c(V(\cdot;\theta))$ from which the inital states of the trajectories are picked.

3. **Experiments.** In this section, the proposed framework for co-learning RoA and ODE is demonstrated on two well-known test cases in the nonlinear systems literature [35, 34, 6, 14], i.e., the Van der Pol oscillator and the inverted pendulum. Further details on the numerical experiments can be found in the Appendix.

3.1. **Van der Pol Oscillator.** Van der Pol oscillator is a dynamical system defined as:

$$\begin{aligned} \dot{x} &= -y \\ \dot{y} &= x + \gamma(x^2 - 1)y \end{aligned} \tag{14}$$

where γ is the damping parameter. When $\gamma > 0$, the system has an unstable limit cycle around the equilibrium and the true RoA of the asymptotically stable equilibrium (at the origin) is the area encircled by the limit cycle (the green region in Figure 3a).

We first show the performance of the RoA estimation algorithm. Recall that the Lyapunov function is parameterized as $V(\cdot;\theta) = v^{\mathsf{T}}(\cdot;\theta)v(\cdot;\theta)$, where $v(\cdot;\theta)$ is an MLP. Because the weights of the network are initialized at random, the initial shape of the level sets of $V(\cdot;\theta_{\text{init}})$ can be far from the shape of the true RoA, which negatively affects the learning algorithm. We observed that the initial weights that result in functions with level sets too far from the shape of the RoA can eventually give a highly conservative estimate of the stability region. To improve the stability of the learning algorithm, we first pre-train the network with the loss function (4). Figure 2(a) shows the level sets of the randomly initialized network. After training with the loss function (4) for $Q = I$ and $r_0 = 0.1$, the level sets of the pre-trained network (Figure 2(c)) become closer to those of the target quadratic function (Figure 2(b)).

We consistently observed that pre-training increases significantly the stability of the algorithm and results in a less conservative final estimate of the RoA.

After pre-training, the RoA estimation algorithm of Section 2.2 is applied with the growth parameter $\alpha = 3.0$. Four steps of the iterative algorithm are shown in Figure 3. The true RoA (green region) is computed by brute force simulation of trajectories starting from the points in a grid that covers the denoted rectangle in the state space. As can be seen, as the computation of the ROA progresses, $\mathcal{S}_c(V(\cdot;\theta))$ gets closer to $\mathcal{R}_{\bar{x}}$ in shape and size and is always contained within it.

The iterations of the coupled algorithm proposed in Section 2.4 consist of expanding the RoA and sampling initial conditions near its boundary to generate trajectories for learning the ODE. It can be seen in Figure 4 that as the estimated RoA (red boundary) grows, the trajectories that are sampled near the boundary become more informative about farther areas from the equilibrium point while are guaranteed to remain stable because they all start from within the inner estimate of the RoA. This gives a systematic way to do experimentation with the system to accurately and efficiently identify it within the RoA.

All trajectories are contaminated with Gaussian measurement noise with a standard deviation 0.05 according to the noise model (7). The trajectories are collected during the growth stages and used for learning the ODE with the loss function (12) with more emphasis on the newer trajectories. This is done to make sure the learned ODE becomes accurate in the newly expanded region around the current RoA while preserving the knowledge it has acquired in the previous steps. The inclusion of the Lyapunov and RoA information in ODE estimation is modular in our proposed algorithm and any regression method that is used to learn ODEs by gradient matching (12) can make use of the estimated Lyapunov function in the same spirit as done here. To emphasize this modularity, we show the progressively improving estimate of the ODE using the multi-step proposed algorithm when the ODE learning phase is carried out by kernel methods (Figure 5) or neural networks (Figure 6). It is apparent that the learned vector field matches the true one inside the RoA. The area where two vector fields are in strong agreement (in the sense of Euclidean distance) grows, while the area with the mismatch between them shrinks (better agreement corresponds to the lighter shade.) This confirms the benefit of the proposed co-learning algorithm to put emphasis on learning the unknown dynamics in the region of the state space that is of practical interest.

As mentioned in Section 2.4, the benefits of the proposed coupling between the Lyapunov and ODE estimation steps are twofold: guiding the experimentation using the RoA information (see Section 2.4.1) and using the Lyapunov information to enforce stability of the vector field (see Section 2.4.2). The first benefit was shown in the progressive training of the vector field in Figures 4 to 6. To illustrate the second benefit, we show that when only very few trajectories are available, the Lyapunov information is highly useful to preserve the structural information of the vector field (that is, the stability properties of the equilibrium of interest.) We run the same experiment reported in Figures 4 to 6 but this time with much less experimentation budget. Specifically, we use 30% of the trajectories and the length of each trajectory is halved. These modifications make it harder for the learning algorithm to preserve the stability information as can be seen in the rightmost and middle plots of Figure 7 where the loss function (12) is used for learning the vector field. The leftmost plot in Figure 7, however, shows that regularizing the base gradient matching loss function with the Lyapunov regularizer is successful in

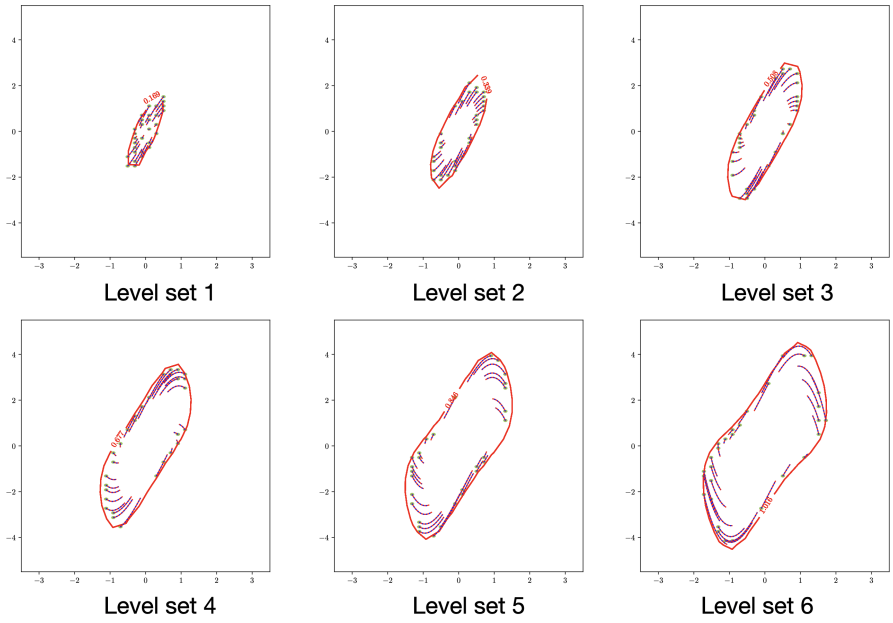


FIGURE 4. Sampled trajectories from inside the estimated RoA of each growth stage.

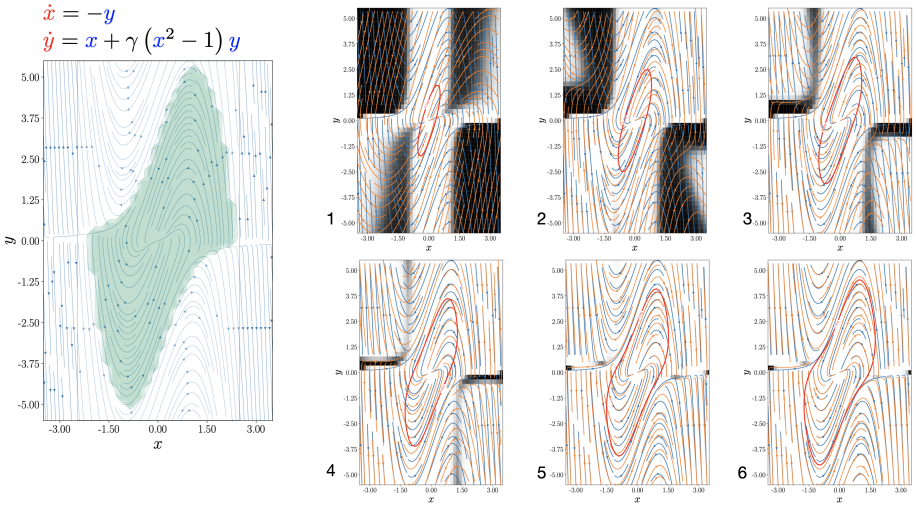


FIGURE 5. (Van der Pol Oscillator) Left: True RoA. Right: The progressively learned ODE over the steps of the coupled algorithm. The darker background shows larger mismatch between the learned and true vector fields.

preserving the stability of the learned vector field even for scarce and short observed trajectories. Notice that this experiment concerns the effect of the reduction of the experimentation budget on learning the ODE in the presence and absence of

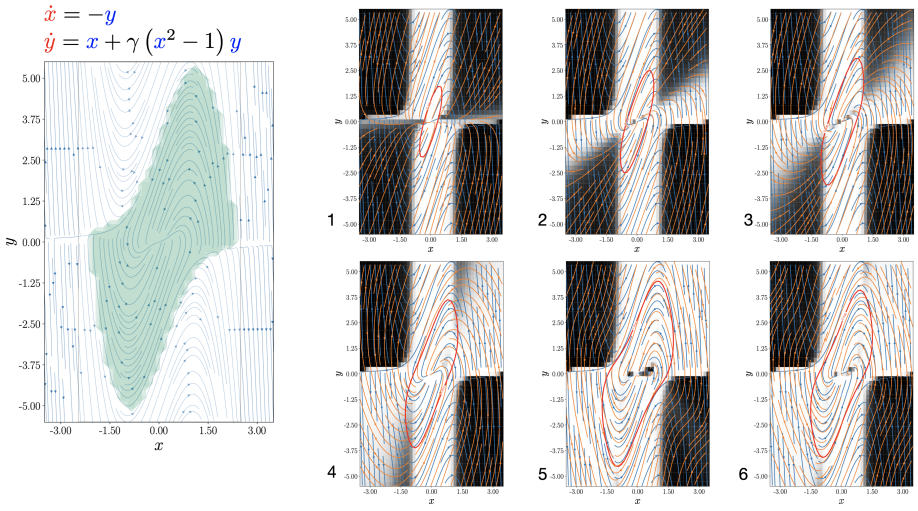


FIGURE 6. (Van der Pol Oscillator) The progressively learned ODE using neural networks.

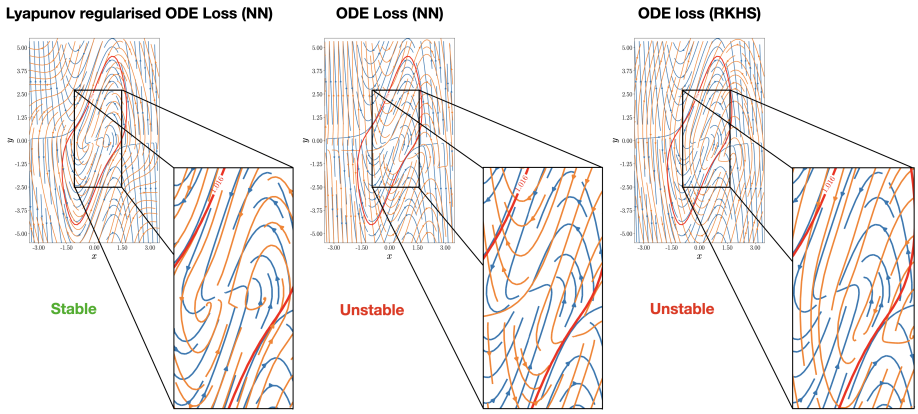


FIGURE 7. (Van der Pol Oscillator) Learned ODEs with and without the Lyapunov regularization term in the loss function.

the Lyapunov regularizer. In order to isolate this effect on learning the ODE from learning the Lyapunov function, we used the learned Lyapunov function of Figure 6.

To compare quantitatively, we fix the total number of trajectories to 100 collected over 10 growth steps of the algorithm. In the first scenario (called blind sampling) the samples are taken uniformly from a rectangle around the equilibrium. The second scenario uses the RoA estimation to guide sampling at each growth step similar to Figure 4 and optimizes (12) to learn the ODE. The third scenario is the same as the second one except that the estimated Lyapunov function is used to regularize the ODE learning as (13). We compute the MSE error between the true and learned vector fields restricted to the true RoA. The MSE for each scenario normalized by the largest error (first scenario) becomes 1 ± 0.37, 0.67 ± 0.21, 0.33 ± 0.09 for the first, second and third scenario respectively. As expected, because the

accuracy of the vector field only within the RoA is of practical interest, RoA-guided sampling is a more efficient way of using the available experimentation budget. Moreover, the ODE is learned with higher accuracy if the estimated Lyapunov function is used as a regularizer.

3.2. **Inverted Pendulum.** The proposed algorithm is here applied on the inverted pendulum. It is a 2-dimensional dynamical system defined as

$$\begin{aligned} \dot{x} &= y \\ \dot{y} &= \frac{g}{l\sin(x)} + \frac{u}{(ml^2)} \end{aligned} \tag{15}$$

where l is the length of the pendulum, m is the pendulum mass and g is the gravity constant. We use $l = 0.5$, $m = 0.25$, and $g = 9.81$ in this experiment. The control signal u is the force applied to the pendulum, which is typically defined by a controller designed in order to keep the pendulum upright. Here, we control system (15) [2, 28] with a state-feedback LQR controller which is saturated by limiting the absolute value of the control input signal with a threshold of 0.7 [17]. Because of the saturation, the system will have a smaller RoA compared to the unsaturated closed-loop scenario, and thus the algorithm developed in this paper can be used to estimate the safe set of initial conditions converging to the equilibrium. The level sets of the Lyapunov function of every growth stage are used to sample the initial state for experimentation as can be seen in Figure 8. Notice that the trajectories are sampled from around the RoA. Hence, as the estimated RoA grows, they become more informative about the ODE at greater distances from the equilibrium. Moreover, the trajectories are stable as they start from within the RoA. The experimental condition is similar to the one described in Section 3.1. The experimented trajectories of each growth stage are used to train an ODE model implemented as a neural network with the same architecture as the one used to estimate the ODE of Section 3.1 with the same training hyperparameters (See Appendix A.4). The results of this multi-stage learning algorithm are presented in Figure 9. On the left, the RoA of the system is overlayed on the stream plot of the system dynamics. On the right, the numbered figures show the order of the progressive ODE learning algorithm. The network has a better estimate of the underlying dynamics in the regions shown with lighter colors. As can be seen, the lighter regions get closer to the equilibrium as the training progresses and more trajectories from inside the estimated RoA become available for training.

To verify the effect of Lyapunov regularization, we repeat the scenarios used at the end of Section 3.1 for quantitative comparison. The total number of trajectories is fixed to 100 collected over 10 growth steps of the algorithm. In the first scenario (blind sampling), the samples are taken uniformly from a rectangle around the equilibrium. In the second scenario the RoA estimation is used to guide sampling at each growth step similar to Figure 8 and optimizes (12) to learn the ODE. The third scenario uses the Lyapunov estimation both for guiding the experimentation and also for regularizing the estimation of the underlying ODE as (13). We compute the MSE error between the true and learned vector fields restricted to the true RoA. The MSE for each scenario normalized by the largest error (first scenario) becomes 1 ± 0.53, 0.72 ± 0.42, 0.59 ± 0.16 for the first, second and third scenario respectively. This verifies again that RoA-guided sampling makes more efficient use of the experimental budget. Furthermore, the information on the Lyapunov function is a useful prior that provides a better estimate of the underlying ODE.

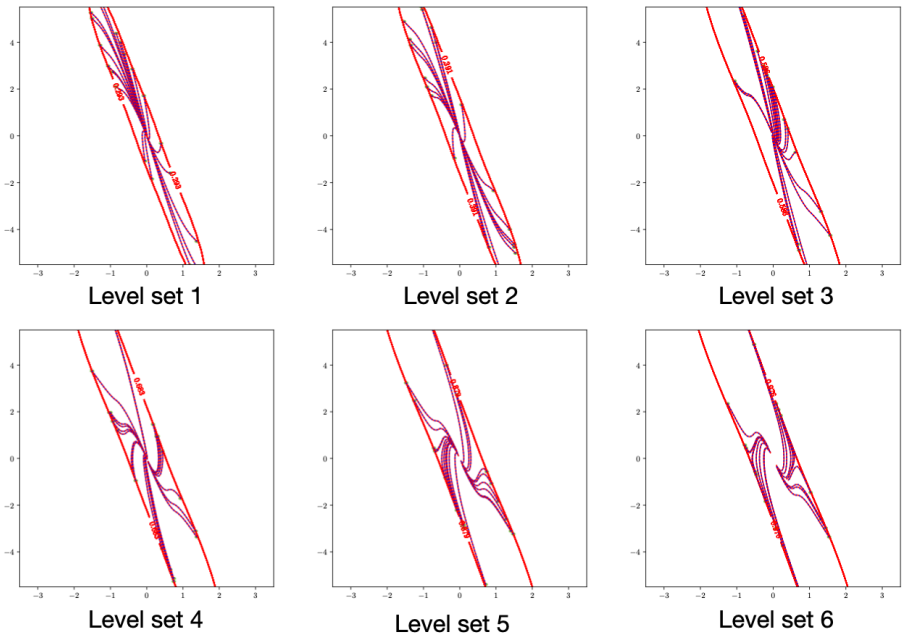


FIGURE 8. (Inverted Pendulum) Sampled trajectories from inside the estimated RoA of each growth stage.

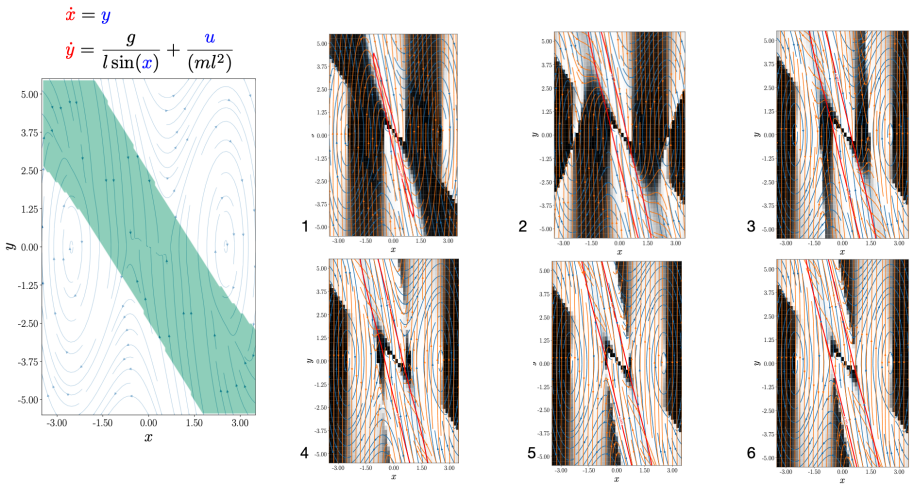


FIGURE 9. (Inverted Pendulum) The progressively learned ODE using neural networks.

4. **Conclusion and Future Work.** The paper proposes a method to co-learn the region of attraction and vector field of a dynamical system from the observed trajectories. The steps of the algorithm are designed such that knowledge of the estimated Lyapunov function can be used to better sample the experimental trajectories and to regularize the ODE learning. It is shown that by making efficient use of the distilled knowledge in the Lyapunov function, the ODE can be estimated

more accurately and with the less experimental cost within the RoA of the system. The proposed algorithm can be seen as a collaborative game between two agents one of which learns the dynamics and the other one assesses the stability of the learned dynamics. Theoretical understanding of the asymptotic and transient conditions of this game is a question to pursue in future works.

An interesting future direction of this work is to make the interaction between the Lyapunov estimator $\hat{V}$ and ODE estimator $\hat{f}$ bi-directional. In this work, the learned Lyapunov function is used to regularize the loss function for learning the ODE, but one can think of using the learned ODE to contribute to learning the Lyapunov function as well, and this could provide better generalization properties to the learned function $\hat{V}$.

Appendix A. **.** The appendix presents implementation details and parameter choices for the numerical experiments.

A.1. **Grid simulation.** To be able to handle a continuous state space, we discretize the state space and construct a grid in a rectangular neighborhood around the equilibrium. The limits of the rectangle are chosen by guessing the maximum and minimum values of each state inside the region of attraction. In applications, one can specify this based on the subset of the state-space of interest. Each axis of the rectangle is then divided into a specified number of intervals (100 intervals in our experiments). More divisions give a closer approximation to the continuous regime at the price of a significant increase in the computational cost. By building this grid, the continuous state space is replaced by a set of finitely many points on the grid. These finite sets of points are then used for calculations and visualizations. For example, to compute the true RoA of the equilibrium of a 2-dimensional system, the trajectories that start from every point of the grid are simulated for a specified number of time steps. If their distance to the equilibrium falls below a specified threshold, they are considered stable and the initial points of the stable trajectories belong to the true RoA. Similarly, to find a level value c that ensures $S_c(V)$ lives within the true RoA, the points of the grid that fall inside $S_c(V)$ are checked to satisfy Lyapunov decrease condition (3).

A.2. **RoA Learning.**
The radius r_0: The radius of the ball in which the Lyapunov function is initialized to match the quadratic function. The value of this radius is not critical because the level value c is always chosen, after every update in θ, such that $S_c(V(\cdot;\theta_0^*))$ lives within the RoA, by checking the Lyapunov decrease condition for its interior points. After normalizing the state values by their assumed maximum value (so that the state values remain in $[0,1]$) we set the radius of this initial set to 0.1.
Discretization time interval Δt: To simulate the evolution of trajectories in time, we used $x_{\text{next}} = f(x_{\text{current}}) * \Delta t + x_{\text{current}}$ with $\Delta t = 0.01$.
Length of the trajectory to verify instability: If a trajectory starting from the gap $\mathcal{G}_\alpha(V,c) := \mathcal{S}_{\alpha c}(V) \backslash \mathcal{S}_c(V)$ does not enter $\mathcal{S}_c(V)$ after 50 steps (that is equivalent to $50 \times \Delta t = 0.5$ time units for $\Delta t = 0.01$), the trajectory is labelled as unstable.
Growth parameter α: The growth parameter determines the size of the gap $\mathcal{G}_\alpha(V,c) := \mathcal{S}_{\alpha c}(V) \backslash \mathcal{S}_c(V)$ ($V(\cdot;\theta)$ produced at every step from which the initial states are chosen to produce experiments. We set it to 3.0 in our experiments.

Lyapunov multilayer perceptron: The Lyapunov function is parameterised as $V(\cdot;\theta) = v(\cdot;\theta)v(\cdot;\theta)$. We implement $v(\cdot;\theta) : \mathbb{R}^2 \to \mathbb{R}$ as a 3−layer MLP (dense layers) with dimensions $[16, 32, 64]$ and nonlinearities $[\texttt{tanh}, \texttt{tanhh}, \texttt{Linear}]$.
Relative weight λ_{RoA}: According to Equation (5) in Section 2.2, this hyperparameter determines the relative weight between updating the Lyapunov function so that its level sets gets closer to the RoA and ~~also~~ making sure the decrease condition of Equation (3) will not be violated inside the estimated RoA. As a convention and inspired by the common terminology of the machine learning community, by "hyperparameter", we refer to high-level or structural parameters such as the relative weights of constituent terms in a loss function, the kernel bandwidth in a kernel-based regression algorithm or the number of hidden layers in a neural network. On the other hand, by "parameters", we refer to lower-level parameters such as the wights $\{a_{k=1}^r\}$ in an RKHS expansion or the weights and bias parameters in a neural network.
Training details: We used 10 growth stages for the RoA learning algorithm presented in Section 2.2. We used Stochastic Gradient Descent (SGD) optimization method with learning rate 0.001 for 100 step for the optimization problem Equation (5). As mentioned in the main text, there are two sets of parameters that affect the chosen level set of $V(\cdot;\theta)$ as the approximation to $\mathcal{R}_{\bar{x}}$ (because $\mathcal{S}_c(V(\cdot;\theta)) \approx \mathcal{R}_{\bar{x}}$). One set is the weights of the employed neural network (θ) and the other parameter is the level value c. We keep the level value fixed ($c = 1$) so that the level set is determined only by the weights of the neural network. After each growth stage, we check if the decrease condition is satisfied for every point within $\mathcal{S}_c(V(\cdot;\theta))$ and decreases c if necessary. This extra caution is needed to make sure the chosen level set never crosses the boundary of the true RoA.

A.3. **Trajectory interpolation.**
The choice of the kernel for trajectory interpolants: We used a gaussian kernel $k(x, y) = \exp(-\gamma^{-2}\|x - y\|_2^2), \gamma > 0$ for trajectory interpolation and chose its bandwidth γ using median heuristics.
Weighting function: We used the Voronoi tessellation (Theorem 2.2) to weight the observed samples from the trajectories.
Measurement noise: Observed trajectories are contaminated with additional noise as $\tilde{x}_t = \phi(t; x_0)_t + \epsilon_t$ with $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$. We set $\sigma = 0.05$.
γ_{traj}: The regularization weight controls the smoothness of the interpolation functions. It should be chosen in response to the level of the measurement noise. This weight together with the bandwidth of the kernel controls the fit of the interpolant to observed data versus its smoothness. We chose $\gamma_{\text{traj}} = 0.2$ in our experiments for the Gaussian measurement noise with standard deviation $\sigma = 0.05$.

A.4. **Coupled algorithm.** To show that the incorporation of the Lyapunov stability information in ODE learning is independent of the algorithm used to learn the ODE, both RKHS and neural networks are tested for the regression problem Equation (12). Here are the hyperparameters for each.
The RKHS method to learn the ODE by Lyapunov regularization: The theory behind the ODE learning using RKHS is the same as what is described in Section 2.3.1 for learning the interpolants with the following exception. The closed-form solution Equation (11) is no longer possible in the presence of the regularization term Equation (13), especially because V is implemented as a neural network. However, we can still solve Equation (13) thanks to automatic differentiation and

stochastic solvers. Hence, we treat both RKHS and neural network methods similarly when solving Equation (13) by using SGD optimization on the weights $\psi = w$ of the neural network or the coefficients $\psi = \alpha$ of the RKHS method.
The choice of the kernel for the RKHS method: We used a Gaussian kernel $k(x, y) = \exp(-\gamma^{-2}\|x - y\|_2^2), \gamma > 0$ for ODE learning and chose its bandwidth by median heuristic. The threshold $\mathcal{T}_g$ in Algorithm 1 that determines the minimum volume of the produced gap at each growth stage is set to 10 percent of the volume of the initial sublevel set that is obtained after pre-training the Lyapunov network. That is, if the produced gap $\mathcal{G} = \mathcal{S}_{\alpha c}(V(\cdot;\theta))\backslash\mathcal{S}_c(V(\cdot;\theta))$ is smaller than a portion of the initial estimate of RoA by pretraining, it is taken as a sign that the estimate of the RoA is fairly accurate and there is no much room for improvement. Notice that the volume of a set is approximated by the number of grid vertices that fall within that set.
Network architecture for the neural network method: We implement the vector field function $\hat{f}(\cdot;\psi) : \mathbb{R}^n \to \mathbb{R}^n$ as a 3−layer MLP (dense layers) with dimensions $[8, 8, 16]$ and nonlinearities $[\texttt{tanh}, \texttt{tanhh}, \texttt{Linear}]$.
Relative weight λ_{ODE}: The relative weight of the Lyapunov regularization term in the loss function Equation (13) is set to $\lambda_{\text{ODE}} = 0.1$
Training details: When optimizing the regularized loss function Equation (13), both RKHS and neural network models are treated by SGD optimizer of PyTorch. After some manual hyperparameter tuning we set $\texttt{learning\_rate} = 0.001$ with $\texttt{batch\_size} = 8$.
Computational power: We ran the experiments on a desktop computer with CPU configuration: 3,2 GHz Quad-Core Intel Core i5.
Average runtime of the experiments: The multi-step coupled algorithm took less than 30 minutes (wall-clock-time) on the machine with the above-mentioned configuration.

A.5. **Implementation technologies.** The algorithm is implemented as a package in Python. We used Scikit-learn[22] for kernel-based estimation method and PyTorch [25] for neural-network modules.

A.6. **How to choose the hyperparameters.** The chosen values of the hyperparameters for the presented experiments are provided in Appendices A.1 to A.4. In this section we discuss the intuition behind each of these hyperparameters extensively that guide the way to choose them. The hyperparameters can be divided into three sets. We discuss each of these sets of hyperparameters separately in the following:

1. The hyperparameters that affect the trajectory interpolations: The interpolation is the technique used to turn the time-sampled trajectories into a continuous function. This can be seen as a regression task whose hyperparameters are chosen by K-fold cross-validation (K=5 in this paper). We can also employ prior knowledge (if exists) about the system dynamics and sampling frequency to choose the prior weight in the loss function. If the system dynamics is not too fluctuating or the sampling frequency is high, the data can speak for itself and we can reduce the weight of the prior γ_{traj} .
2. The hyperparameters that affect the ODE learning. This hyper-parameter trades off the information provided by the trajectories and the information provided by the Lyapunov function for learning the ODE. It is important to notice that the Lyapunov function is also trained in our work. Hence, the value

of λ_{ODE} reflects how much we trust the learned Lyapunov function at any stage of training. As a general rule of thumb in machine learning algorithms, we choose λ_{ODE} such that both terms in eq (13) are of the same order of magnitude. However, we observed that changes by one order of magnitude (multiplied by 10 and divided by 10) do not cause a significant change in the results. The other effective hyper-parameter here is the length of the trajectory when we can choose it ourselves. It is possible to think about a real-world scenario where the length of the trajectories is constrained by the system under study due to safety or energy regulations. In case we are free to choose the length of the experimented trajectories, we can make sure the trajectories do not exist in the inner estimate of the RoA provided by the current estimate of the Lyapunov function. This is unlikely because we only sample from inside the RoA, but because the Lyapunov function itself is learnable, the RoA estimate might be inaccurate and the initial states chosen too close to the RoA boundary may happen to fall outside of the true RoA of the system. With this safety measure in mind, the length of the trajectory is a proxy of the information it provides about the underlying ODE. However, this information does not increase linearly with the length of the trajectories especially when they are stable. The reason is that the trajectories starting within the RoA get closer to each other near the equilibrium. Hence the information to provide becomes more redundant as they get closer to the equilibrium. Therefore as a rule of thumb, the experimentation trajectory length is set to a length that is about the time needed for that trajectory to enter the previous estimate of the RoA. Because the objective function of the ODE learning part only includes two adjacent time steps, smaller trajectories that cover the whole RoA altogether will be as informative as longer trajectories. To explain more clearly, consider three-level sets associated with level values $C > B > A$. The above argument states that having a long trajectory from level set C to level set A does not provide more information than having a trajectory from level set C to B and another trajectory from level set B to C.

3. The hyperparameters that affect the Lyapunov learning. The growth parameters α: This hyper-parameter determines the size of the gap around the current RoA from which the trajectories are initiated to later be used to improve the estimate of the RoA and the Lyapunov function. The gap should not be too thin or too small. If it is too thin, not many vertices of the grid (the grid we used to discretize the space) will fall within it and it's less likely we can get a fair balance of stable and unstable trajectories to train the classifier that determines the boundary of the RoA. The same issue occurs when the samples from the gap are highly biased towards unstable samples which again leads to a skewed dataset to train the RoA boundary classifier. Therefore, a rule of thumb is to choose α in a way that the resultant dataset for binary classification is not too biased towards positive (stable) or negative (unstable) trajectories. This can be done by starting from a small α in the initial phases and checking the positive to negative ratio of the labels. Then, it increases until the resultant dataset is almost balanced. The Lyapunov function regularization parameter λ_{RoA}: This hyperparameter determines the relative weight of the satisfaction of the Lyapunov decrease condition within the RoA and the accuracy of its level set in separation unstable and unstable trajectories. The second term in Equation (5) has to be fulfilled for every

point within the RoA. This may imply over-emphasizing it in the objective function by increasing λ_{RoA}. However, this may lead to overshadowing the first term (boundary classification term) by the second term. We observed a highly large value of λ_{RoA} results in a more conservative estimate of the RoA. To avoid this problem, we use a common approach in machine learning algorithms, namely in the presence of multiple terms in the objective function the relative weights are tuned in such a way that all terms have the same order of magnitude.

REFERENCES

[1] F. Berkenkamp, R. Moriconi, A. P. Schoellig and A. Krause, Safe learning of regions of attraction for uncertain, nonlinear systems with Gaussian processes, in *2016 IEEE 55th Conference on Decision and Control (CDC)*, 2016.

[2] O. Boubaker, The inverted pendulum benchmark in nonlinear control theory: a survey, *International Journal of Advanced Robotic Systems*, **10** (2013), 233.

[3] C. Carmeli, E. De Vito and A. Toigo, Vector valued reproducing kernel Hilbert spaces of integrable functions and Mercer theorem, *Analysis and Applications*, **4** (2006), 377–408.

[4] S. Chen, Optimal bandwidth selection for Kernel density functionals estimation, *Journal of Probability and Statistics*, **2015** (2015), 1–21.

[5] T. Q. Chen, Y. Rubanova, J. Bettencourt and D. K. Duvenaud, Neural ordinary differential equations, in *Advances in neural information processing systems*, 2018, 6571–6583.

[6] G. Chesi, *Domain of Attraction: Analysis and Control via SOS Programming*, Springer, 2011.

[7] A. Chiuso and G. Pillonetto, System identification: A machine learning perspective, *Annual Review of Control, Robotics, and Autonomous Systems*, **2** (2019), 281–304.

[8] T. Evgeniou, M. Pontil and T. Poggio, Regularization networks and support vector machines, *Advances in computational mathematics*, **13** (2000), 1–50.

[9] P. Giesl, Construction of a global Lyapunov function using radial basis functions with a single operator, *Discrete & Continuous Dynamical Systems - B*, **7** (2007), 101–124.

[10] P. Giesl and S. Hafstein, Construction of a CPA contraction metric for periodic orbits using semidefinite optimization, *Nonlinear Analysis: Theory, Methods and Applications*, **86** (2013), 114 – 134.

[11] P. Giesl and S. Hafstein, Review on computational methods for Lyapunov functions, *Discrete & Continuous Dynamical Systems - B*, **20** (2015), 2291–2331.

[12] P. Giesl, B. Hamzi, M. Rasmussen and K. Webster, Approximation of Lyapunov functions from noisy data, *Journal of Computational Dynamics*, **7** (2020), 57–81.

[13] L. Grüne, Computing Lyapunov functions using deep neural networks, *Journal of Computational Dynamics*, **8** (2021), 131–152.

[14] A. Iannelli, P. Seiler and A. Marcos, Region of attraction analysis with integral quadratic constraints, *Automatica*, **109** (2019), 1–10.

[15] H. K. Khalil, *Nonlinear systems*, Prentice Hall, 1996.

[16] M. Korda and I. Mezić, Optimal construction of Koopman eigenfunctions for prediction and control, *IEEE Transactions on Automatic Control*, **65** (2020), 5114–5129.

[17] F. L. Lewis, D. Vrabie and V. L. Syrmos, *Optimal Control*, John Wiley & Sons, 2012.

[18] L. Ljung, *System Identification: Theory for the User*, Prentice Hall PTR, 1999.

[19] I. G. K. Matthew O. Williams Clarence W. Rowley, A kernel-based method for data-driven Koopman spectral analysis, *Journal of Computational Dynamics*, **2** (2015), 247–265.

[20] E. Najafi, R. Babuška and G. A. Lopes, A fast sampling method for estimating the domain of attraction, *Nonlinear Dynamics*, **86** (2016), 823–834.

[21] P. A. Parrilo, Semidefinite programming relaxations for semialgebraic problems, *Mathematical Programming*, **96** (2003), 293–320.

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, Scikit-learn: Machine learning in Python, *Journal of Machine Learning Research*, **12** (2011), 2825–2830.

[23] G. Pillonetto, F. Dinuzzo, T. Chen, G. De Nicolao and L. Ljung, Kernel methods in system identification, machine learning and function estimation: A survey, *Automatica*, **50** (2014), 657–682.

[24] T. Poggio and C. R. Shelton, On the mathematical foundations of learning, *American Mathematical Society*, **39** (2002), 1–49.
[25] A. Poyton, M. S. Varziri, K. B. McAuley, P. J. McLellan and J. O. Ramsay, Parameter estimation in continuous-time dynamic models using principal differential analysis, *Computers & chemical engineering*, **30** (2006), 698–708.
[26] M. Revay and I. Manchester, Contracting implicit recurrent neural networks: Stable models with improved trainability, in *Learning for Dynamics and Control*, PMLR, 2020, 393–403.
[27] M. Revay, R. Wang and I. R. Manchester, A convex parameterization of robust recurrent neural networks, *IEEE Control Systems Letters*, **5** (2021), 1363–1368.
[28] S. M. Richards, F. Berkenkamp and A. Krause, The Lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems, in *Conference on Robot Learning*, PMLR, 2018, 466–476.
[29] Y. Rubanova, T. Q. Chen and D. K. Duvenaud, Latent ordinary differential equations for irregularly-sampled time series, in *Advances in Neural Information Processing Systems*, 2019, 5321–5331.
[30] J. Schoukens and L. Ljung, Nonlinear system identification: A user-oriented road map, *IEEE Control Systems Magazine*, **39** (2019), 28–99.
[31] S. Smale and D.-X. Zhou, Shannon sampling and function reconstruction from point values, *Bulletin of the American Mathematical Society*, **41** (2004), 279–305.
[32] I. Steinwart and A. Christmann, *Support Vector Machines*, Springer Science & Business Media, 2008.
[33] M. M. Tobenkin, I. R. Manchester and A. Megretski, Convex parameterizations and fidelity bounds for nonlinear identification and reduced-order modelling, *IEEE Transactions on Automatic Control*, **62** (2017), 3679–3686.
[34] U. Topcu, A. Packard and P. Seiler, Local stability analysis using simulations and sum-of-squares programming, *Automatica*, **44** (2008), 2669 – 2675.
[35] G. Valmorbida and J. Anderson, Region of attraction estimation using invariant sets and rational Lyapunov functions, *Automatica*, **75** (2017), 37–45.

E-mail address: amehrjou@tuebingen.mpg.de
E-mail address: iannelli@control.ee.ethz.ch
E-mail address: bs@tuebingen.mpg.de