

CODE: A global approach to ODE dynamics learning

Nils Wildt

*Institute for Modelling Hydraulic and Environmental Systems
University of Stuttgart*

nils.wildt@iws.uni-stuttgart.de

Daniel M. Tartakovsky

*Department of Energy Science and Engineering
Stanford University*

tartakovsky@stanford.edu

Sergey Oladyshkin

*Institute for Modelling Hydraulic and Environmental Systems
University of Stuttgart*

sergey.oladyshkin@uni-stuttgart.de

Wolfgang Nowak

*Institute for Modelling Hydraulic and Environmental Systems
Cluster of Excellence SimTech
University of Stuttgart*

wolfgang.nowak@iws.uni-stuttgart.de

Abstract

Ordinary differential equations (ODEs) are a conventional way to describe the observed dynamics of physical systems. Scientists typically hypothesize about dynamical behavior, propose a mathematical model, and compare its predictions to data. However, modern computing and algorithmic advances now enable purely data-driven learning of governing dynamics directly from observations.

In data-driven settings, one learns the ODE’s right-hand side (RHS). Dense measurements are often assumed, yet high temporal resolution is typically both cumbersome and expensive. Consequently, one usually has only sparsely sampled data. In this work we introduce ChaosODE (CODE), a Polynomial Chaos ODE Expansion in which we use an arbitrary Polynomial Chaos Expansion (aPCE) for the ODE’s right-hand side, resulting in a global orthonormal polynomial representation of dynamics. We evaluate the performance of CODE in several experiments on the Lotka-Volterra system, across varying noise levels, initial conditions, and predictions far into the future, even on previously unseen initial conditions. CODE exhibits remarkable extrapolation capabilities even when evaluated under novel initial conditions and shows advantages compared to well-examined methods using neural networks (NeuralODE) or kernel approximators (KernelODE) as the RHS representer. We observe that the high flexibility of NeuralODE and KernelODE degrades extrapolation capabilities under scarce data and measurement noise.

Finally, we provide practical guidelines for robust optimization of dynamics-learning problems and illustrate them in the accompanying code.

1 Introduction

Scientists strive to understand the mechanisms of the world. In mathematical modeling, mechanisms are often formulated as ordinary differential equations (ODE) dynamics. Let us assume a model $\frac{d}{dt}x(t) = f(x(t))$. The state $x(t) \in \mathbb{R}^n$ evolves in time $t \in [t_0, t_{end}]$ according to a continuous dynamic $f : \mathbb{R}^n \mapsto \mathbb{R}^n$. Very often, the concrete dynamic $f(x(t), \theta)$ also depends on several parameters θ that alter the dynamical behavior. To calibrate the model, numerous techniques have been developed in recent years to determine the function that best fits the observed data. Typically, scientists propose models that depend on various parameters. In the

simplest case, the parameters have physical meanings and can be determined through measurements. If the parameters are either not observable by measurements or have no physical measurable meaning, they need to be found by inversion and optimization techniques. The model is then fitted on the basis of a performance metric *e.g.* minimizing the pointwise distance between the model output and real-world observations (*e.g.* Tarantola, 2005; van de Schoot et al., 2021). If the model is not suitable, one iterates model selection and improvement (Höge et al., 2018).

Relaxing the assumption of a fixed parametric model, we may search over a function space for the RHS. In doing so, we can use optimization to discover the dynamics in a very general way. However, we assume that we only have state measurements at discrete points in time and no inherent derivative information of the states themselves. Hence we cannot just solve a simple regression problem for the RHS, but need to optimize in a highly nonlinear setting in which each evaluation of the model involves the time integration of the current estimated dynamics. The choice of the RHS function space (ansatz) governs the approximation power and the training complexity. In the work on NeuralODE, modeling of arbitrary learnable operators was widely seen for the first time (Chen et al., 2019). This idea was later formally generalized for arbitrary right-hand sites and termed Universal Differential Equations (UDE) by Rackauckas et al. (2020). Within this general UDE setting, the RHS representation is itself a key modeling decision. In succession to the idea of NeuralODEs, over the past few years, a thorough comprehension of the dynamics involved in learning Ordinary Differential Equations (ODEs) via adequate datasets has been developed. This enhanced understanding has increasingly shown its practical applications in a multitude of fields, particularly within geosciences (Shen et al., 2023) and hydrology (Höge et al., 2022). In parallel, other RHS representations were investigated, specifically the use of Gaussian Processes, which are another known nonlinear approximation method in machine learning applications (Kanagawa et al., 2018; Hegde et al., 2022). As these approaches only rely on the mean path, we refer to these methods with KernelODE, highlighting the underlying theory of kernel methods (Santin & Haasdonk, 2021). Both neural network- and kernel-based frameworks typically require dense training datasets containing hundreds of data points. However, many practical applications, such as contaminant transport of TCE (Wüst et al., 1999) and PFAS (Croll et al., 2022), commonly have far fewer observations. In such sparse settings, highly flexible models risk poor extrapolation without strong inductive bias.

While NeuralODEs have gained widespread use in scientific applications, the simplicity often portrayed in studies does not align with the difficult challenges practitioners encounter when trying to effectively optimize dynamics using limited or noisy data. A notable issue is that tests on out-of-distribution, previously unseen initial states are rarely conducted, leaving the ability to extrapolate largely unexplored.

We address this challenge of learning from limited data by introducing arbitrary Polynomial Chaos ODE Expansion (CODE), a new global and orthonormal polynomial RHS approximation. This approach naturally represents polynomial dynamics (common across physics, chemistry, and biology) and can approximate arbitrary functions. In CODE, we find a multinomial expansion to represent the dynamics of differential equations. This is realized using the arbitrary Polynomial Chaos (aPC) Expansion as described in (Oladyshkin & Nowak, 2012), an approach built on the generalized Polynomial Chaos expansion (Xiu & Karniadakis, 2002). In contrast to a full polynomial basis (Fronk & Petzold, 2023), our data-driven basis is orthogonal with respect to the data distribution. In our case, this is the distribution of the states that we know over time. The ChaosODE results in relatively few coefficients that determine the behavior of the RHS while enabling the calculation of dynamics across the entire allowable domain, since polynomials are global functions. Previous work on improving UDEs in time-only settings projected the NeuralODE solution onto an orthogonal basis (De Brouwer & Krishnan, 2023), while PCE collocation learning was applied for physics infusion by (Sharma et al., 2024), although their approach focused on general (collocation) PDE solving and surrogate modeling rather than time-only applications.

Overall, we hypothesize that problem-tailored RHS spaces improve learning. Therefore, we aim to answer the following unresolved questions:

1. How do current UDE learning techniques perform when training data is both scarce and noisy?
2. What extrapolation and generalization capabilities do KernelODE and NeuralODE exhibit, and how does CODE's global approximation structure compare in these aspects?

-
3. Does the orthonormal structure of the dynamics representation facilitate learning in highly non-linear systems?

We first present CODE, followed by the other methods already established in Section 2. Then we illustrate the training procedure followed by the results and a discussion (Section 3) to be concluded in Section 4.

2 Methodology

This section provides a complete explanation of the general RHS learning problem, introduces the CODE framework and comparison methods, and provides training and implementation specifics.

2.1 From data to an ODE

Consider a system whose states we observe over time. The observed data are denoted as $\mathcal{D} = \{(t_i, \hat{\mathbf{x}}_i)\}$ for $i = 1, \dots, N$, where N represents the number of distinct measurements taken at arbitrary times and sorted in ascending order. Measurements typically contain noise ε , such that $\hat{\mathbf{x}}_i = \mathbf{x}_i + \varepsilon_i$, where $\mathbf{x}_i = \mathbf{x}(t_i)$ represents the true unknown value without noise.

Since the time intervals $\Delta t = t_i - t_{i-1}$ can be arbitrary, we cannot directly approximate the derivative using $\frac{d}{dt}x_i \approx \lim_{\Delta t \rightarrow 0} \frac{x_i - x_{i-1}}{\Delta t}$. Instead, we assume that the state evolution follows a differential equation. Specifically, we consider an autonomous ordinary differential equation (ODE) of the form

$$\frac{d}{dt}\mathbf{x}(t) = \dot{\mathbf{x}}(t) = f(\mathbf{x}(t)), \quad (1)$$

where $\mathbf{x}(t) \in \mathbb{R}^n$ denotes the state of the system at time $t \in \mathbb{R}$ and $f(\mathbf{x}(t))$ is the vector field. When combined with an initial state (t_0, \mathbf{x}_0) , this formulation defines an initial value problem (IVP) (Bronstein et al., 1989).

Such IVPs are solved with

$$\mathbf{x}(t) = \int_{t_0}^t f(\mathbf{x}(\tau)) d\tau + \mathbf{x}_0. \quad (2)$$

The ODE is determined by the dynamics $f(\cdot)$ and an initial condition. Ultimately, the goal is therefore to find an approximation of the flow field $\tilde{f}_{\theta}(\cdot) \approx f(\cdot)$ and an estimation of $\mathbf{x}(t_0) \approx \tilde{\mathbf{x}}(t_0) = \tilde{\mathbf{x}}_0$. We assume that $\tilde{f}(\cdot)$ is parameterized by θ and, in general, we can determine the dynamics and initial conditions by finding

$$\theta = \begin{pmatrix} \theta' \\ \mathbf{x}_0 \end{pmatrix},$$

in the general case of unknown x_0 . In the following, we call $\tilde{f}_{\theta}(\cdot)$ the representation of the RHS. The optimal parameters are obtained by solving the optimization problem:

$$\theta^* = \arg \min_{\theta} \ell_d, \quad (3)$$

where $\tilde{\mathbf{x}}_i(\theta)$ denotes the prediction of the model at the measurement point i as a function of the parameters θ and the distance loss between the observed data at several times and the prediction at the same times, for example, $\ell_d = \sum_{i=1}^N \|\mathbf{x}_i - \tilde{\mathbf{x}}_i\|_d$. Next, we elaborate on different approaches to model $\tilde{f}(\cdot)$.

Choice of a basis for the vector field The choice of representation space fundamentally determines the quality and generalization of the approximation. To approximate functions with specific properties, the hypothesis class must be able to express those properties. Consider the practical implication: if we seek to approximate physically valid right-hand sides (RHS) of differential equations, then the chosen representation space must be capable of encoding physical constraints and behaviors. The representation space acts as a constraint on the solution space and it cannot approximate what the chosen framework cannot express. This principle motivates our comparative analysis of three distinct RHS representation approaches.

-
- NeuralODE (see Section 2.2): Neural networks encode the dynamics, leveraging universal approximation capabilities.
 - KernelODE (see Section 2.3): Gaussian process mean trajectories represent dynamics, provide smoothness guarantees, and localized basis function.
 - ChaosODE (see Section 2.4): Polynomial basis representation offers analytical tractability and global approximation.

Each representation space imposes different structural assumptions and constraints, directly influencing both the quality and characteristics of the resulting approximation. Thus, the representation is a fundamental modeling decision that determines the space of achievable solutions, not merely a technical implementation detail.

Next, we summarize the existing approaches and introduce a novel ChaosODE approach.

2.2 NeuralODE

Neural Ordinary Differential Equations (NeuralODE) leverage neural networks to learn dynamic representations (Chen et al., 2019). Building on the foundational work in neural networks (Hopfield, 1982; Rumelhart et al., 1986), the core concept is elegantly simple: a neural network serves as the regressor for the RHS side of an ODE.

Formally, the learned function is defined as $\tilde{f} : \mathbb{R}^n \mapsto \mathbb{R}^n$, where n denotes the number of states. The NeuralODE framework has inspired researchers in diverse disciplines to integrate ODEs with artificial neural networks as universal function approximators (Rackauckas et al., 2020).

The selected network architecture should capture the essential features of the expected dynamics. According to the universal approximation theorem, it is theoretically possible to model arbitrary dynamics using this approach (Hornik, 1991). However, the choice of network architecture and hyperparameters significantly influences the performance of the model. The many small choices that finally determine the dynamics in a NeuralODE setting often require timely fine-tuning and can easily induce researchers' bias towards certain dynamical behavior.

In general, NeuralODEs offer remarkable flexibility in modeling dynamical systems. Although this approach can arbitrarily finely interpolate within the training data domain, extrapolation remains arbitrary, as we lack understanding of the effective ansatz function generated by the network combination that represents the dynamics. This expansive solution space, while powerful for interpolation tasks, requires careful consideration when predicting system behavior beyond the observed regime. Consequently, extrapolation beyond the training data can be unreliable, as neural-network parameterizations often generalize poorly outside the data support.

2.3 Kernel ODE

The second approach employs kernel methods to approximate the RHS of an ODE. Like neural networks, kernels provide a regression framework for learning dynamics from data. The key distinction lies in their construction: Kernel methods build the approximation using basis functions centered at selected data points, which makes the method inherently local.

A *kernel* is a function $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ that measures the similarity between inputs. For all $x, x' \in \mathcal{X}$, the kernel satisfies:

$$k(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}} \quad (4)$$

where $\phi : \mathbb{R}^d \mapsto \mathcal{H}$ maps the input to a Hilbert space \mathcal{H} (Hofmann et al., 2008; Santin & Haasdonk, 2021; Schaback & Wendland, 2006). This mapping enables nonlinear regression without explicitly computing high-dimensional features.

The dynamics approximation uses a kernel expansion:

$$\tilde{f}_{1,\mathcal{K}}^{\theta^{\mathcal{K}}}(x) = \sum_{i=1}^n c_i^1 k(x, x_i) \quad (5)$$

Here, $c_i^1 \in \mathbb{R}$ are coefficients and $\{x_i\}_{i=1}^n$ are collocation points (also called inducing or pilot points ([Schaback & Wendland, 2006](#); [Titsias, 2009](#); [Doherty et al., 2011](#))). These collocation points define where the kernel basis functions are centered. The approximation quality depends directly on their number and placement, as the method interpolates between these points using the kernel-weighted combination.

In the ideal case where we know the true dynamics f at collocation points, the target vector is $\theta^{\mathcal{K}} = [\theta_1, \theta_2, \dots, \theta_n]^{\top}$ with $\theta_i = f(x_i)$. The coefficients are then computed through:

$$c = (K + \lambda I)^{-1} \theta^{\mathcal{K}} \quad (6)$$

The kernel matrix K has entries $K_{ij} = k(x_i, x_j)$ and is positive definite by construction. The regularization parameter $\lambda > 0$ prevents overfitting and ensures numerical stability, with I denoting the identity matrix.

In practice, we do not have access to the true dynamics at collocation points. Therefore, the unknown collocation values $\theta^{\mathcal{K}}$ are considered to be trainable parameters. During training, these collocation values $\theta^{\mathcal{K}}$ should be adjusted so that the resulting ODE solution matches the observed data. Given $\theta^{\mathcal{K}}$, coefficients follow directly from the linear system $c = (K + \lambda I)^{-1} \theta^{\mathcal{K}}$. The optimization thus focuses on finding collocation values that produce dynamics consistent with measurements, while the kernel framework handles interpolation between collocation points.

For systems with multiple state variables ($n > 1$), we construct separate kernel regressors for each dimension. Each regressor maps $\mathbb{R}^n \mapsto \mathbb{R}$. For a two-dimensional system ($n = 2$), the complete approximation becomes:

$$\tilde{f}_{\mathcal{K}}^{\theta^{\mathcal{K}}}(x) := \begin{pmatrix} \tilde{f}_{1,\mathcal{K}}^{\theta^1}(x) \\ \tilde{f}_{2,\mathcal{K}}^{\theta^2}(x) \end{pmatrix} \quad (7)$$

The data-dependent nature of kernel methods affects the approximation quality across the state space. Near collocation points, the approximation typically achieves high accuracy. Between collocation points, the kernel provides smooth interpolation. Outside the region covered by pilot points, the extrapolation behavior depends entirely on the chosen kernel function. Kernel-specific hyperparameters (such as length scales) and the regularization parameter λ require tuning through cross-validation or similar methods.

While some implementations frame this approach within a Gaussian Process (GP) framework ([Heinonen et al., 2018](#)), the deterministic kernel regression presented here is equivalent to using the mean prediction of a GP ([Kanagawa et al., 2018](#)). Probabilistic interpretation adds uncertainty quantification, but does not change the fundamental regression mechanism. As a local method, extrapolation behavior is governed by the kernel choice and its length scale(s), which then corresponds to a notion of smoothness in the dynamic space.

2.4 Chaos ODE

In this work, we introduce Chaos ODE that employs data-driven orthonormal polynomial chaos basis functions ([Oladyshkin & Nowak, 2012](#)) to approximate the RHS of an ODE. Unlike its traditional role in uncertainty quantification, we re-purpose aPC as a data-driven global and orthonormal basis over the state space for learning dynamics.

The dynamics $f(\cdot)$ are approximated through an expansion in orthogonal polynomials:

$$f(x) = \sum_{p=0}^{\infty} c_p \Phi_p(x), \quad (8)$$

where $x \in \mathbb{R}^n$ denotes the state variable and $\{\Phi_p\}_{p \in \mathbb{N}_0}$ forms an orthonormal polynomial basis. These basis functions satisfy the orthogonality condition with respect to a weight function $w(x)$ induced by a probability measure \mathcal{P} :

$$\langle \Phi_p, \Phi_q \rangle_{\mathcal{P}} := \int_{\mathbb{R}^n} \Phi_p(x) \Phi_q(x) w(x) dx = \int_{\mathbb{R}^n} \Phi_p(x) \Phi_q(x) d\mathcal{P}(x) = \delta_{pq}, \quad \forall p, q \in \mathbb{N}_0, \quad (9)$$

where δ_{pq} denotes the Kronecker delta. The expansion coefficients are obtained via projection:

$$c_p = \langle f, \Phi_p \rangle_{\mathcal{P}}, \quad p \in \mathbb{N}_0. \quad (10)$$

The probability measure \mathcal{P} defines the weight function for orthonormality. In classical Wiener polynomial chaos expansion, the orthogonal family is fixed by the input distribution, *i.e.* Hermite polynomials for Gaussian distribution. Generalized polynomial chaos extends this mapping across the Askey scheme to cover additional standard distributions (*e.g.* Legendre for uniform, etc.: (Xiu & Karniadakis, 2002)). In contrast, we employ arbitrary polynomial chaos (Oladyshkin & Nowak, 2012), constructing data-driven orthonormal polynomials from empirical moments of \mathcal{P} . For practical computation, we truncate the expansion at polynomial degree N_{\max} :

$$\tilde{f}_C^{\theta^C}(x) \approx \sum_{p=0}^{N_{\max}} \theta_p^C \Phi_p(x), \quad (11)$$

where θ_p^C are the parameters to be learned. The truncation degree N_{\max} balances the approximation accuracy against the computational cost. For n -dimensional systems, multivariate polynomials are constructed through tensor products, with the total number of basis functions growing combinatorially with dimension and polynomial degree.

This approach provides a global approximation with spectral convergence rates for smooth dynamics. Unlike local methods such as kernels, polynomial chaos expansions use global basis functions spanning the entire state space. Although there are standard polynomial families for common distributions (Sullivan, 2015), we construct data-driven orthonormal polynomials using arbitrarily chaotic polynomial chaos. This approach adapts the basis to the empirical distribution of the state variables.

We construct each polynomial Φ_d of degree d as:

$$\Phi_d(x) = \frac{1}{\sqrt{h_d}} \sum_{i=0}^d c_i x^i, \quad (12)$$

where h_d is a normalization constant and c_0, \dots, c_d are coefficients to be determined. To find these coefficients, we enforce orthogonality through a moment-matching approach (Oladyshkin & Nowak, 2012).

The orthogonality constraint equation 9 requires that $\langle \Phi_i, \Phi_j \rangle_{\mathcal{P}} = \delta_{ij}$. Combined with the auxiliary condition $c_d = 1$ (fixing the leading coefficient), this yields a linear system. Using the k -th raw moment $\mu_k = \int_{\mathbb{R}} x^k d\mathcal{P}(x)$, we obtain:

$$\begin{bmatrix} \mu_0 & \mu_1 & \cdots & \mu_{d-1} \\ \mu_1 & \mu_2 & \cdots & \mu_d \\ \vdots & \vdots & \ddots & \vdots \\ \mu_{d-1} & \mu_d & \cdots & \mu_{2d-2} \\ 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{d-1} \\ c_d \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}. \quad (13)$$

This Hankel matrix system determines the coefficients for the d -th polynomial using moments of up to order $2d - 1$. When the distribution \mathcal{P} is unknown, we estimate the moments from the data:

$$\tilde{\mu}_k = \frac{1}{N} \sum_{j=1}^N x_j^k, \quad (14)$$

where $\{x_j\}_{j=1}^N$ are the observed state values. By the law of large numbers, $\tilde{\mu}_k \rightarrow \mu_k$ as $N \rightarrow \infty$.

To ensure numerical stability and maintain orthonormality, we normalize each polynomial after construction:

$$\Phi_d(x) = \frac{\Phi_d(x)}{\|\Phi_d\|_{L^2(\mathcal{P})}}, \quad (15)$$

where the norm is estimated using Monte Carlo integration:

$$\|\Phi_d\|_{L^2(\mathcal{P})}^2 \approx \frac{1}{N} \sum_{j=1}^N [\Phi_d(x_j)]^2. \quad (16)$$

This normalization prevents numerical instabilities that arise when polynomials of increasing degree have vastly different magnitudes.

For multivariate systems with $\mathbf{x} \in \mathbb{R}^n$, we construct tensor product polynomials:

$$\Phi_{\alpha}(\mathbf{x}) = \prod_{i=1}^n \Phi_{\alpha_i}(x_i), \quad (17)$$

where $\alpha = (\alpha_1, \dots, \alpha_n)$ is a multi-index with total degree $|\alpha| = \sum_{i=1}^n \alpha_i$. Restricting to polynomials with $|\alpha| \leq N_{\max}$ yields

$$M = \binom{n + N_{\max}}{n} = \frac{(n + N_{\max})!}{n! N_{\max}!} \quad (18)$$

basis functions. This combinatorial growth with dimension n limits the method to moderate-dimensional problems.

Overall we now have three different RHS approximators that can be trained, with an overview in Table 1

Table 1: Overview of different RHS approximation approaches.

Name	Representation	Trainable parameters	Hyperparameters
NeuralODE	$\tilde{f}_{\mathcal{N}}^{\theta^{\mathcal{N}}}(x) := \mathcal{N}\mathcal{N}^{\theta^{\mathcal{N}}}(x)$	Weights and biases	Network architecture, activation functions
ChaosODE	$\tilde{f}_{\mathcal{C}}^{\theta^{\mathcal{C}}}(x) := \begin{pmatrix} \varphi_1^N(x)\theta_1^{\mathcal{C}} \\ \varphi_2^N(x)\theta_2^{\mathcal{C}} \end{pmatrix}$	Expansion coefficients	Maximal polynomial degree
KernelODE	$\tilde{f}_{\mathcal{K}}^{\theta^{\mathcal{K}}}(x) := \begin{pmatrix} \tilde{f}_{1,\mathcal{K}}^{\theta^1}(x) \\ \tilde{f}_{2,\mathcal{K}}^{\theta^2}(x) \end{pmatrix}$	Pilot values at pilot points	Number & location of pilot points, Kernel function

2.5 Training technicalities and guidelines

Training high-dimensional surrogates with neural networks is well-established. Gradient-based optimizers such as Adam (Kingma & Ba, 2014) with properly tuned learning rates consistently find good optima. Beyond achieving low training losses, these models must also generalize to unseen data.

In the ODE context, generalization takes two distinct forms: predicting state estimates for unseen initial conditions (\mathbf{x}_0) and for unseen time points. The first requires an ansatz space that correctly extrapolates functional behavior beyond the training domain. The second (temporal) type relies on accurate integration over longer horizons once the RHS is learned. In all three setups, the RHS can theoretically be easily

determined if we could evaluate $\mathbf{f}(x(t))$ at any time. However, as we assume there are only a few observations, an optimization determines the interpolation and extrapolation. Optimizing through ODE solutions presents three key challenges in our setting:

1. Parameter sensitivity: Small parameter changes can produce large trajectory variations. This sensitivity requires the optimizer to accept poor intermediate solutions while traversing unfavorable regions of the loss landscape.
2. Solver divergence: When the ODE solver diverges, it produces neither solutions nor gradients, preventing informed parameter updates.
3. RHS-dependent dynamics: Different right-hand side formulations create distinct parameter-trajectory dependencies, complicating the development of a unified optimization strategy.

In the following paragraphs, we describe how we obtain stable training for all three RHS choices. To make the three approaches comparable, we succeeded in identifying one optimization approach for all scenarios.

Multiple- vs. single shooting Single-shooting methods solve the entire ODE system from x_0 over the full time interval $t \in [t_0, t_{end}]$ using a single set of parameter coefficients. This approach faces a fundamental stability bottleneck: if the current parameter estimates of the RHS cause numerical instabilities (non-finite values) at any point during integration, the entire solution trajectory becomes invalid, preventing objective function evaluation and halting optimization progress. Although present for all three RHS choices, this is especially severe in the CODE setting, as polynomials grow unboundedly outside their stable regions, rapidly producing overflow errors that terminate integration.

This creates a circular dependency problem. Successful parameter updates during training require stable numerical integration to compute gradients and objective values. However, achieving numerical stability often requires parameter values that are already close to the optimal solution. Without good initial parameter guesses, the optimization becomes trapped in overly conservative parameter regions that produce stable but overly smooth, mean-like solutions instead of capturing the dynamic behavior present in the observed data.

Although global optimization methods can potentially escape these conservative regions, they face the same stability constraints. Even when exploration successfully identifies parameter regions that yield stable trajectories, subsequent exploitation phases must navigate carefully to avoid reentering unstable regimes. In practice, this makes it extremely difficult for single-shooting approaches to converge to solutions that accurately capture non-trivial dynamics without requiring exceptional initial parameter estimates.

Multiple-shooting methods for ODEs (Turan & Jäschke, 2021) prevent solver divergence by partitioning the integration interval into smaller subintervals, avoiding non-finite values during integration. Although multiple shooting requires careful initialization of each segment's initial values and enforcement of continuity constraints, we employ a hybrid approach that bypasses this complexity. Multiple shooting generates initial approximations that, though segmented and non-smooth, provide robust starting points. A subsequent single-shooting refinement step produces smooth and accurate solutions through gradient-based optimization. This two-stage strategy combines numerical stability with high fit accuracy, as our tests confirm. Sequential optimization approaches, which increase the number of data segments, consistently underperformed compared to this direct multiple-shooting method in our preliminary tests.

Discretize then Optimize vs. Optimize then Discretize We adopt a discretize-then-optimize approach (Onken & Ruthotto, 2020; Sapienza et al., 2024) for robust optimization. This method discretizes the ODE using a fourth-order Runge-Kutta (RK4) scheme, then applies automatic differentiation through the discretized operations. This contrasts with optimize-then-discretize (adjoint) methods, which derive continuous gradients before numerical discretization. Our choice follows recent findings that discretize-then-optimize provides comparable accuracy, along with a simpler implementation and faster computation for our problem class, which our experiments confirm.

Initial RHS Parameter Estimation Despite the application of global optimization techniques, accurate initial parameter estimates for the respective RHS representative remain crucial for convergence efficiency. We exploit the regularity typically present in time-series data by approximating the solution with a kernel regression surrogate. Since kernel regression is closed under linear operations, we can compute time derivatives at arbitrary points throughout the dataset. This provides a known surrogate $\tilde{f}_{\text{kernel}}(\cdot)$ that can be directly evaluated, thereby transforming the problem into a simple regression scenario. Using these derivative estimates to conduct preliminary training of all RHS models, we obtain initial coefficients that avert immediate divergence during the subsequent optimization phase.

Optimization Pipeline From these initial parameter values, we perform a sequence of optimization steps, each balancing numerical stability, exploration, and exploitation. First, we use a Particle Swarm Optimization (PSO) algorithm ([Eberhart & Kennedy, 1995](#)) in a single-shot setting to move from arbitrary parameter initializations to a stable solution that does not diverge. This is possible, as the initial parameter estimation allows for plausible solutions. The PSO is well suited here, as initial infeasible parameter configurations will simply drop out, leaving us with those that produce stable solutions. In addition, it is fast, allowing for high exploration capabilities. From then on, we employed the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) ([Hansen & Ostermeier, 2001](#)) in a multiple shooting setup. CMA-ES is a gradient-free evolutionary strategy. It provides multistart optimization capabilities while adapting the search distribution toward more favorable solutions. This allows exploration within the feasible parameter space, minimizing the summed losses across all segments. It is again quite robust to intermediate infeasible solutions, as they would again just be dropped in favor of newly drawn coefficient combinations. These two methods conclude the numerically stable and exploratory parts. After CMA-ES converged, we used a Quasi-Newton method ([Nocedal & Wright, 2006](#)) in the multiple-shooting setup. This exploits the local gradients as well as an approximation of the Hessian and, in most cases, converges to locally approximated parameter sets. Finally, we smooth the segmented multiple-shooting solution via a single-shooting Quasi-Newton refinement. All optimization was performed using the Manopt.jl and Optim.jl julia libraries ([Bergmann, 2022](#); [Mogensen & Riseth, 2018](#)).

3 Results and discussion

Traditional model calibration optimizes a limited number of physical parameters to minimize data loss while assuming a fixed functional form. Recent advances in machine learning enable a more powerful approach; we can simultaneously learn the functional dependencies during optimization. We learn a plausible dynamic within a space of possible dynamical systems. Neural Ordinary Differential Equations (NeuralODEs) have driven recent progress in this field. Their high flexibility enables them to approximate arbitrary dynamics. However, this flexibility comes at a cost: unclear extrapolation capabilities beyond the training domain.

We demonstrate how selecting a suitable basis can substantially improve extrapolation to new initial conditions. To this end, we evaluate the recently introduced CODE and related methods on the Lotka–Volterra benchmark ([Lotka, 1925](#); [Volterra, 1926](#)) for learning dynamics. We compare ChaosODE, NeuralODE, and KernelODE across the four scenarios S1–S4 described in Table 2.

Table 2: Experimental scenarios for polynomial chaos expansion modeling

Scenario ID	Scenario Type	Description
S1	Perfect information baseline	Optimization from globally optimal initial coefficients on 144 training observations to evaluate how local optimization affects extrapolation capabilities without noise.
S2	Data effect	Systematic increase in training data size to identify minimum data requirements for reliable model performance without noise.
S3	Noise effect	Analyze the impact of data noise on the training robustness.
S4	ChaosODE advantage	Investigate ChaosODE, test orthonormal- vs. standard- basis.

3.1 Performance Metrics

Given that the state magnitudes in our benchmark problem are of comparable scale, we employ the mean squared error (MSE) as our evaluation metric:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (x_i - \hat{x}_i)^2 \quad (19)$$

where x_i represents the simulation result, and \hat{x}_i denotes the corresponding observed data.

Table 3: Performance evaluation setup for dynamic system modeling

Comparison ID	Comparison Type	Description	Purpose
<i>ex-it</i>	In training time	Evaluation using original training dataset	Measures in-distribution training performance
<i>ex-oot</i>	Out of training time	Testing with lengthened time intervals but same x_0 as in <i>ex-it</i>	Assesses model's ability to extrapolate dynamics over longer durations
<i>ex-ood</i>	Out of distribution	Testing with new, different initial conditions for prolonged time	Evaluates model's generalization to unseen starting points

3.2 Benchmark problem

The Lotka-Volterra predator-prey model is a frequently studied problem in data-based ODE learning problems as in the NeuralODE study ([Chen et al., 2019](#)):

$$\frac{dx}{dt} = \alpha x - \beta xy \quad (20)$$

$$\frac{dy}{dt} = \gamma xy - \delta y. \quad (21)$$

For our benchmark, we replicate the conditions from ([Chen et al., 2019](#)) with $\alpha = 1.5$, $\beta = 1$, $\gamma = 1$, and $\delta = 3$ and initial conditions $x_0 = (1.0, 1.0)^T$. From a numerical solution of the (forward) ODE problem, we

sample N equidistant time points t_i at which we save the two states. These tuples (t_i, x_i, y_i) are then the training data where we assume that we know (t_0, x_0, y_0) . Some points and the true solution and the training data over time are shown in Figure 1, and the true solution in the state space is shown in Figure 2. The training data points are shown as red dots, and the true solution is shown as a black line. The training data points are not equally spaced in the state space, but equally spaced in time.

We evaluate model performance using three distinct setups to assess different aspects of the learned dynamics. The first setup (*ex-it*) evaluates the in-distribution performance by assessing the model on the original training time interval while using the same initial conditions employed during training. The second setup (*ex-oot*) examines temporal extrapolation capabilities by extending the prediction horizon to twice the training duration while maintaining the same initial conditions. The third setup (*ex-ood*) tests generalization to unseen initial conditions by combining the extended time horizon with different starting points not present in the training data. Table 3 provides an overview of these evaluation strategies. Specifically, we use the following settings:

- *ex-it*: $t \in (0.0, 7.0)$ and $u_0 = (1.0, 1.0)^T$
- *ex-oot*: $t \in (0.0, 14.0)$ and $u_0 = (1.0, 1.0)^T$
- *ex-ood*: $t \in (0.0, 14.0)$ and $u_0^{ood} = (0.5, 0.5)^T$.

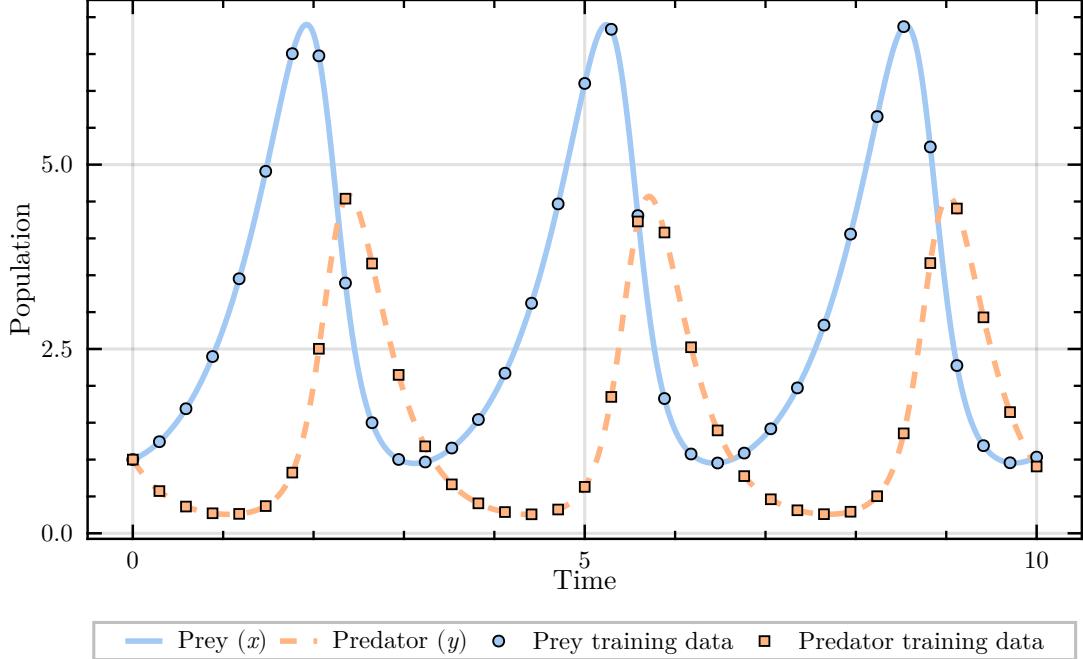


Figure 1: Numerical Lotka-Volterra ODE solution and $N = 35$ training data points over time.

3.3 Scenario results

The scenarios presented below progress from a very general setup to more specialized aspects of ODE learning. Each of the scenarios is evaluated on the three comparison setups, as mentioned before.

S1: Perfect information – local overfitting In this scenario, we investigate how local training data can distort globally accurate approximations when the RHS representation is local. We start by assuming access

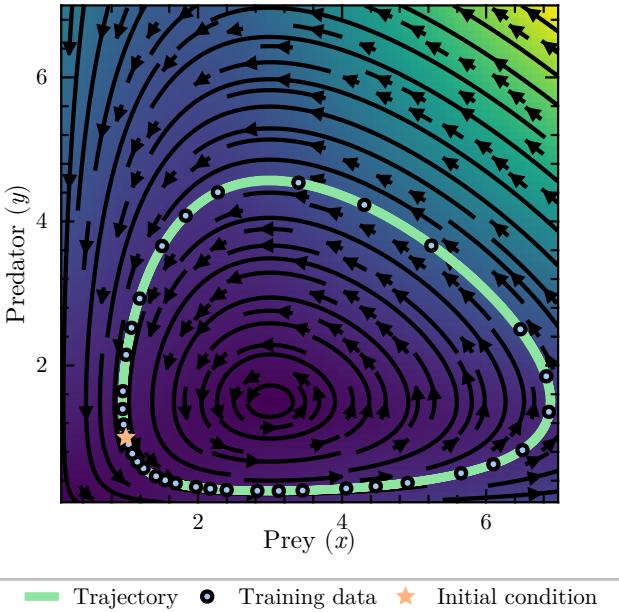


Figure 2: Numerical Lotka-Volterra ODE solution and $N = 35$ training data points in the state space

to the true RHS function throughout the entire state space, which allows us to pretrain the RHS globally optimal coefficients. This global pretraining establishes an initially accurate approximation of the dynamics everywhere.

After establishing these ideally pretrained models, we subsequently fine-tune them using only a single trajectory starting from $x_0 = (1.0, 1.0)^T$ with dense observations (144 data points per state variable). This setup reveals a critical trade-off: while the models improve their accuracy along the observed trajectory, their performance deteriorates in regions of the state space that are not covered by the training data. This phenomenon demonstrates how local data can "carry away" initially accurate global approximations, creating a bias toward the observed trajectory at the expense of generalization elsewhere.

The results reveal substantial differences among the three modeling approaches. CODE achieves high precision with MSE of 1.23×10^{-6} for in-training (ex-it) and maintains this accuracy for temporal extrapolation (ex-oot) with 2.83×10^{-6} . Most notably, it retains strong generalization to unseen initial conditions with an ex-ood MSE of only 0.0119. This superior performance stems from the polynomial chaos basis being ideally matched to the Lotka-Volterra dynamics (quadratic in the states). Since the true RHS consists of polynomial terms (αx , βxy , γxy , and δy), the polynomial basis functions can represent these dynamics exactly. While the ex-ood MSE should theoretically be zero given this exact representation, two factors contribute to the observed error: numerical integration introduces accumulating floating-point errors over the extended time horizon, and the optimization may converge to a near-optimal local minimum that provides an excellent least-squares fit on the training trajectory while deviating slightly from the true global coefficients.

In contrast, NeuralODE and KernelODE exhibit the local overfitting more prominently. Both achieve reasonable training accuracy: KernelODE reaches a MSE of 7.85×10^{-6} and NeuralODE achieves a MSE of 2.78×10^{-3} on the training trajectory. However, their performance degrades dramatically when tested on new initial conditions. NeuralODE's ex-ood MSE increases to 15.84, while KernelODE's MSE rises to 3.59. These large errors indicate that local fine-tuning has compromised their global approximation capabilities.

The degradation occurs because both models adapt too flexibly to local data. Neural networks use nonlinear activation functions while kernel models employ localized basis functions, both allowing excessive adaptation to the training trajectory. During fine-tuning, gradient updates concentrate on parameters affecting observed data

points, while parameters governing unobserved regions drift from their optimal values. Though KernelODE’s structured basis functions provide better resilience than NeuralODE, both exhibit substantial generalization errors compared to CODE, highlighting that RHS selection is a critical modeling decision.

Figure 3 visualizes these differences in the phase space. The training data appears as green points, with the true trajectories shown in orange and the learned model predictions in orange for two different initial conditions. The NeuralODE solution collapses immediately from the outer periodic orbit into the inner one, producing a trajectory visible only along the training data. KernelODE maintains a more stable extrapolation, though its trajectory slowly spirals inward rather than forming a periodic orbit, deviating moderately from the expected black trajectory. CODE produces distinct orbits that closely match the true dynamics, exhibiting only minor inward spiraling. These visualizations directly correspond to the quantitative errors: NeuralODE’s immediate divergence reflects its ex-ood MSE of 15.84, KernelODE’s gradual drift aligns with its MSE of 3.59, and CODE’s near-perfect orbits confirm its minimal MSE of 0.0119.

This scenario demonstrates a fundamental challenge in ODE learning: accuracy on training data does not guarantee generalization. Even with perfect initial approximations, local fine-tuning can destroy global accuracy when the chosen basis functions do not match the underlying and required dynamics. Table 4 summarizes these performance metrics, emphasizing that successful extrapolation requires an appropriate model structure.

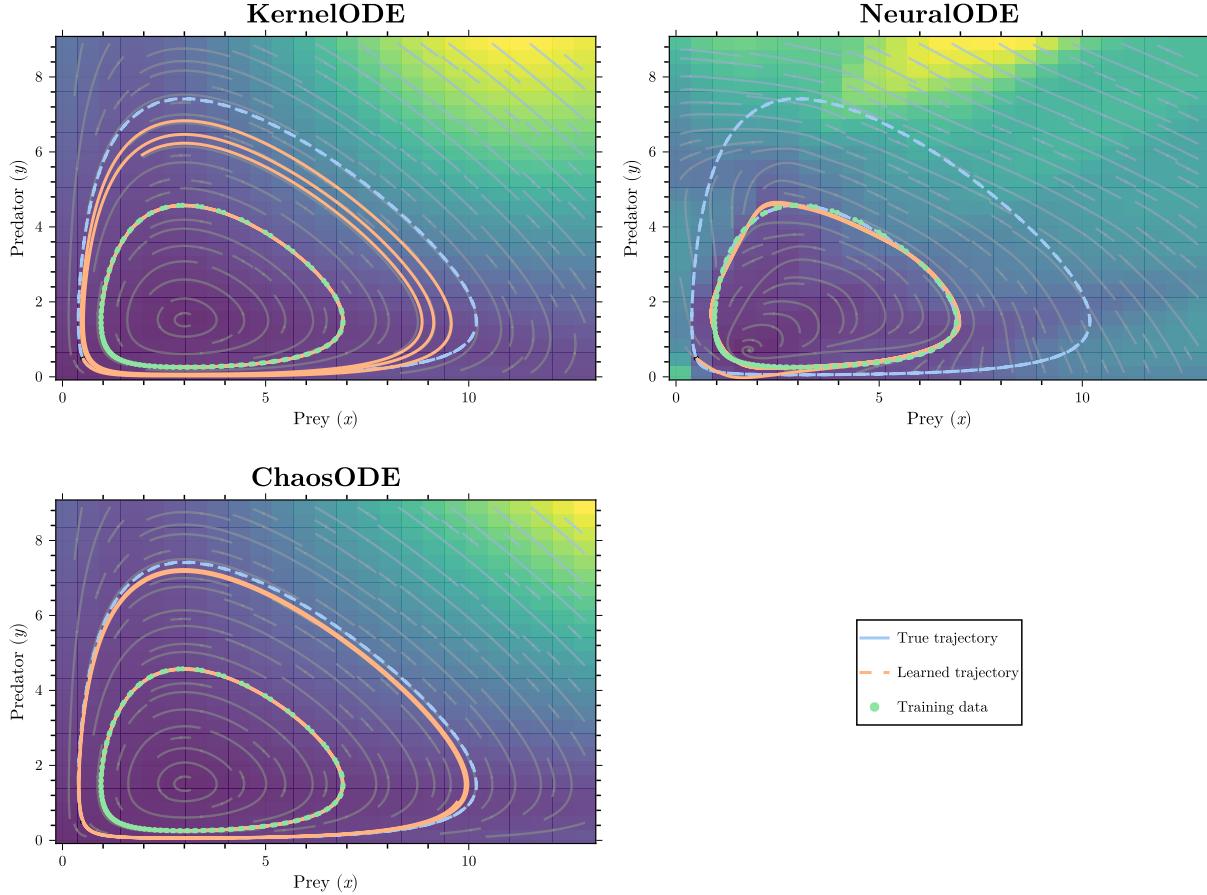


Figure 3: Phase space visualization of three different modeling approaches for the Lotka-Volterra system in S1. The heatmap shows vector field magnitude, streamlines represent flow direction, and colored trajectories show model predictions from two initial conditions, resulting in two separate (orange) periodic orbits. Training data points are marked with crosses.

Ansatz	ex-it	ex-oot	ex-ood
NeuralODE	$2.78 \cdot 10^{-3}$	$2.77 \cdot 10^{-3}$	15.84
KernelODE	$7.85 \cdot 10^{-6}$	$7.93 \cdot 10^{-6}$	3.59
ChaosODE	$1.23 \cdot 10^{-6}$	$2.83 \cdot 10^{-6}$	0.0119

Table 4: Performance comparison in S1. In-training mean squared error (ex-it), out-of-training mean squared error (ex-oot), and out-of-distribution mean squared error (ex-ood) for different ODE models.

S2: Data effect – more is not better? In this scenario, we examine how effectively the various RHS choices can learn the generating dynamics of the Lotka–Volterra system. We increased the number of training data points from 10 to 500. For each training point size, we run 10 independent optimizations with different random initializations to account for starting dependent training outcomes.

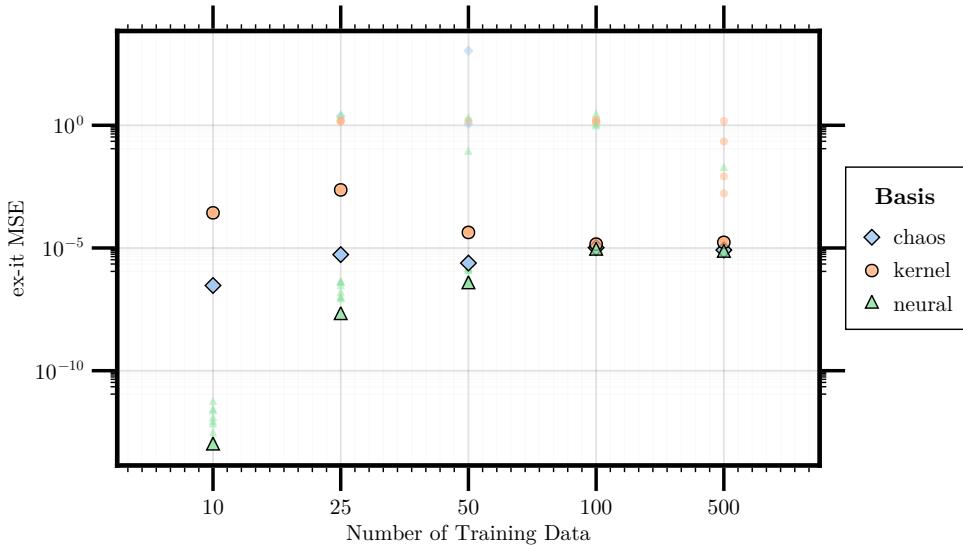


Figure 4: Convergence of the MSE *ex-it* error as training data size increases from 10 to 100 samples. Training data contains no noise ($\sigma = 0.0$). Stars indicate the best-performing model for each combination of training data size and basis function.

First, we analyze the behavior in the *ex-it* setup, where we test on the training data points. The observed error represents the training error and demonstrates how well the model can fit the given data points without requiring generalization. In Figure 4, we observe that NeuralODE particularly benefits from its locality and high flexibility. The model can accurately fit individual data points, even with limited training data. The other two methods do not show significant improvement as the data volume increases. However, with insufficient data points, we may violate the Nyquist sampling theorem (Nyquist, 1928) for oscillatory behavior, potentially yielding solutions that pass through data points at multiples of the true frequency.

Generally, we would expect that methods with stronger structural assumptions require fewer data points to achieve good generalization performance. We do see that more data points lead to less spread among all three RHS basis choices. The low flexibility of KernelODE and its mismatch to Lotka–Volterra dynamics make it the weakest in *ex-it*.

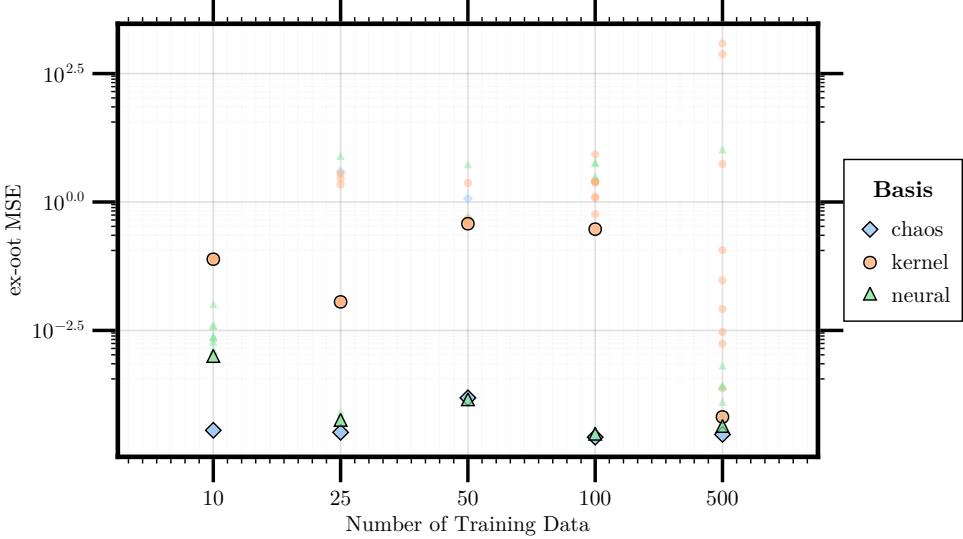


Figure 5: Convergence of the MSE *ex-oot* error as training data size increases from 10 to 100 samples. Training data contains no noise ($\sigma = 0.0$). Stars indicate the best-performing model for each combination of training data size and basis function.

For tests in the *ex-oot* setup (see Fig. 5), we first observe that we are no longer within the training dataset, and the average MSE range becomes higher by several magnitudes. However, as the ChaosODE RHS is ideally able to represent the true dynamics, it succeeds in learning the most accurate representation within our comparison. In particular, a higher data density improves NeuralODE *ex-oot* performance, aligning with established scaling laws (Kaplan et al., 2020). In these experiments, the best KernelODE solution performs worse than the best NeuralODE solution for all numbers of training data.

Interpolation in dense data regimes is well established and can be achieved using many machine learning tools. However, generalization remains one of the biggest challenges (Rohlfs, 2025). In the *ex-ood* setup, the model is evaluated on trajectories from new initial conditions not seen during training. Without additional assumptions, no model has theoretical guarantees for generalizing to these cases, particularly when predicting system behavior from initial conditions requires extrapolation beyond the training data range.

The assumptions that explicitly or implicitly inform the modeling approach differ across methods: for NeuralODE, these include all hyperparameter choices such as activation functions and the number of layers; for KernelODE, they primarily involve the kernel choice and kernel hyperparameters; for CODE, our key assumption is the highest polynomial degree. The three basis function examples clearly demonstrate that different basis choices lead to different extrapolation properties. In this noise-free scenario, we therefore demonstrate that choosing a basis function well-suited to approximate the observed dynamics enables successful extrapolation beyond the training domain.

As shown in Figure 6 for the *ex-ood* case, CODE solutions achieve the best extrapolation performance across all training data sizes. Since the training data contains no noise, we attribute the increased error with larger datasets to the transition from multiple-shooting to single-shooting optimization. With more data points, the multiple-shooting approach introduces higher continuity errors between segments that are not adequately resolved during the final optimization step. In general, we observe that selecting an appropriate basis function

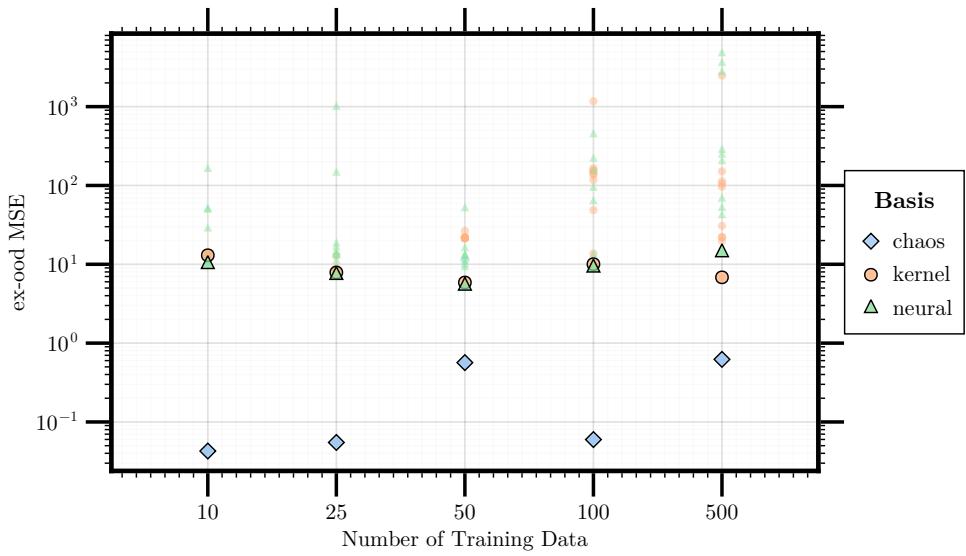


Figure 6: MSE *ex-ood* error as a function of training data size. Training data contains no added noise ($\sigma = 0.0$). Stars indicate the best-performing model for each combination of training data size and basis function.

at the beginning of the learning process — in our case, a polynomial basis—is essential for extrapolation that goes beyond random guessing.

S3: Noise makes training more difficult. Measured data typically contains noise, including measurement errors represented by the random component ε . In the S3 experiments, we model this noise as $\varepsilon \sim \mathcal{N}(0, \sigma)$ with varying noise levels σ . Figures 7 and 8 show training point errors across different noise levels for datasets with 10 and 500 data points, respectively. As noise increases, the minimum achieved MSE rises across all methods, as to be expected. The 500-data-point case reveals increased variance, particularly for KernelODE, though no divergent values or optimization failures occur within the displayed range. To assess optimization

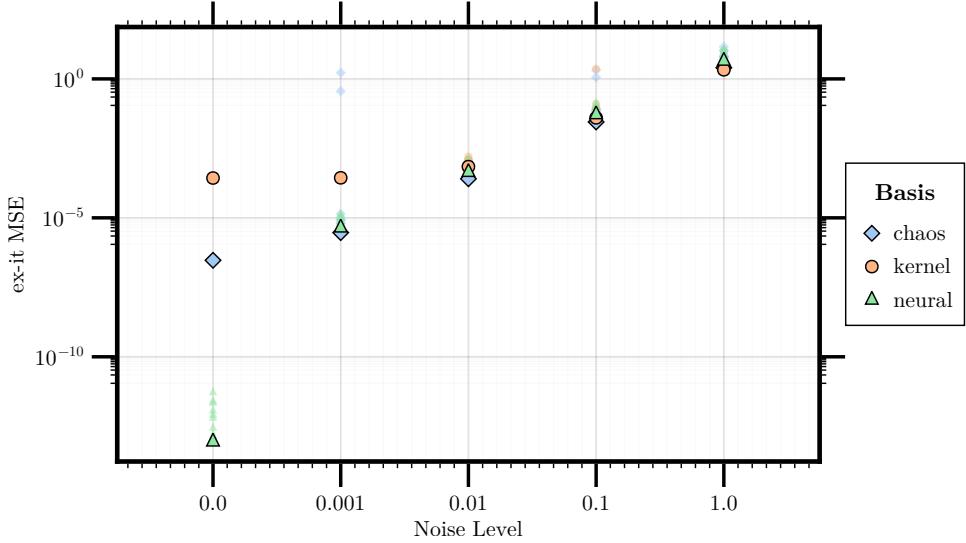


Figure 7: Comparison of MSE on training data (*ex-it*) across different noise levels and methods, trained on 10 data points.

robustness comprehensively in S3, we analyze success rates across different conditions. We define training failure as achieving $\text{MSE} > 10.0$, evaluating all combinations of training point quantities and ten random seeds (providing different initial values and noise realizations). Figure 9 presents these results. The *ex-it* (in-training time) performance remains consistently high, with success rates between 95% and 100% across all conditions. This demonstrates robust training stability despite noise variations, different realizations, and varying data quantities. The *ex-oott* (out-of-training time) case naturally yields higher MSEs, resulting in lower success rates. CODE achieves the highest success rates, particularly at low noise levels ($\sigma = 0.0, 0.001, 0.01$). This superior performance results from the perfect fit of the basis for the underlying polynomial dynamics. However, high noise ($\sigma = 1.0$) causes a notable drop in CODE success rates. Noise creates competing effects on the training dynamics. The MSE loss function should theoretically center solutions within the empirical distribution by accounting for symmetric noise. However, increased noise generates multiple parameter configurations with identical loss values. This condition makes the optimization problem less well-posed and can cause oscillations between training points, especially as we have not fine-tuned the methods to the higher noise, *e.g.* tuning the regularization, bandwidth for KernelODE or optimizer metaparameters. In contrast, NeuralODE and KernelODE show slight stability improvements with additional noise. This improvement potentially results from regularization effects that are not specifically optimized for the noise levels or data quantities tested. The results for the *ex-oott* and *ex-oott* are provided in the appendix Section A.

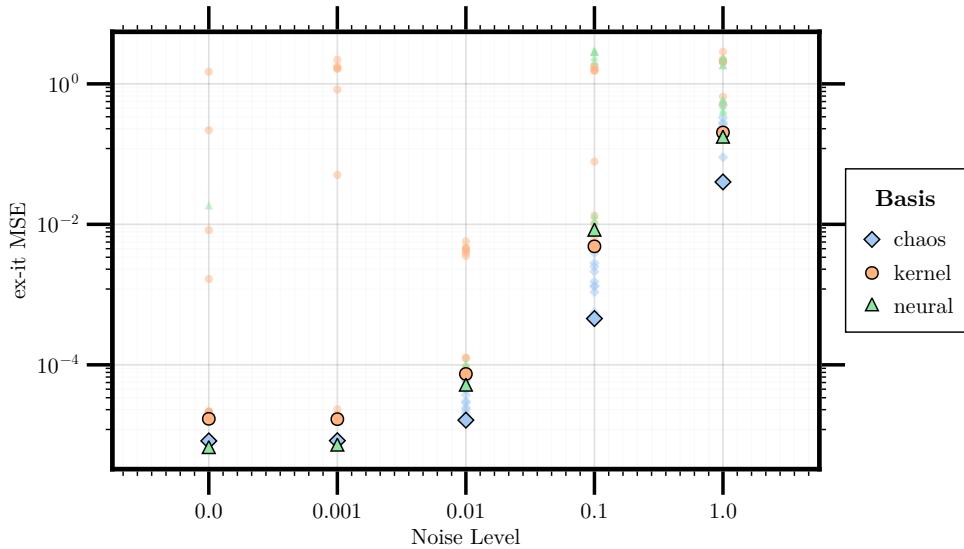


Figure 8: Comparison of MSE on training data (*ex-it*) across different noise levels and methods, trained on 500 data points.

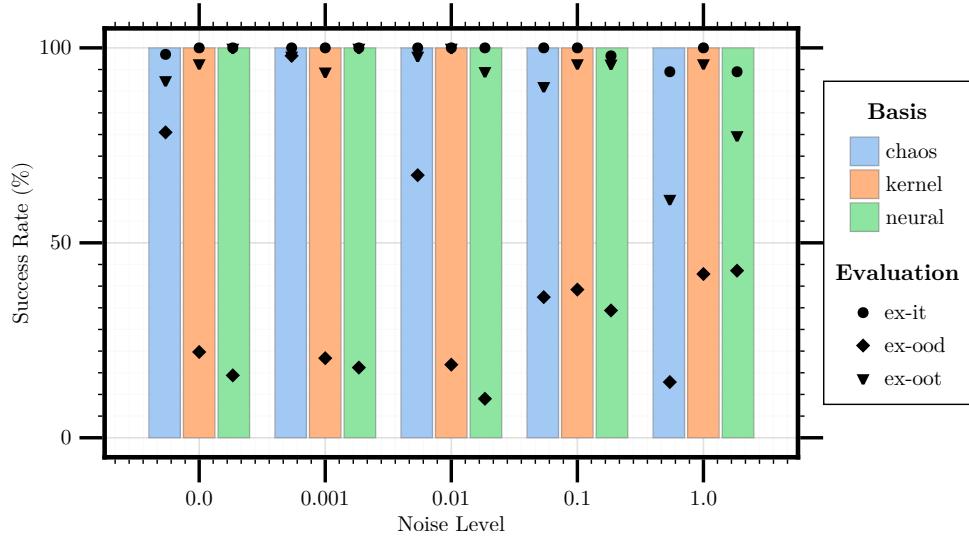


Figure 9: Success rates (percentage of solutions with $MSE < 10.0$) across different noise levels and training data sizes.

S4: Orthonormal vs. Non-orthonormal Basis Section S4 examines how basis construction affects CODE performance in all evaluation metrics. We compare results using a data-dependent orthonormal basis against a non-orthonormal polynomial basis (such as the Vandermonde basis) under identical training conditions. Again, 10 runs were performed for each configuration.

The orthonormal basis creates a well-conditioned optimization problem by ensuring that the basis functions have unit norm and are mutually orthogonal. This conditioning avoids the numerical difficulties associated with the Vandermonde basis, where the basis functions can have vastly different scales and high correlation ([Gautschi \(1983\)](#)). The resulting system behaves similarly to what would be achieved by preconditioning the non-orthonormal case, but achieves this conditioning directly through basis construction. Similar results were reported in other regression-like scenarios ([Li et al. \(2011\)](#)).

The orthonormal basis consistently reduces errors across all evaluation scenarios. Figure 10 shows substantial improvements in out-of-distribution performance, particularly in low-data regimes. Training performance exhibits similar patterns (Figure 11), with the orthonormal basis achieving lower MSE values in noise and data conditions. These improvements come with reduced computational cost; Figure 13 demonstrates faster convergence times, especially in data-limited scenarios.

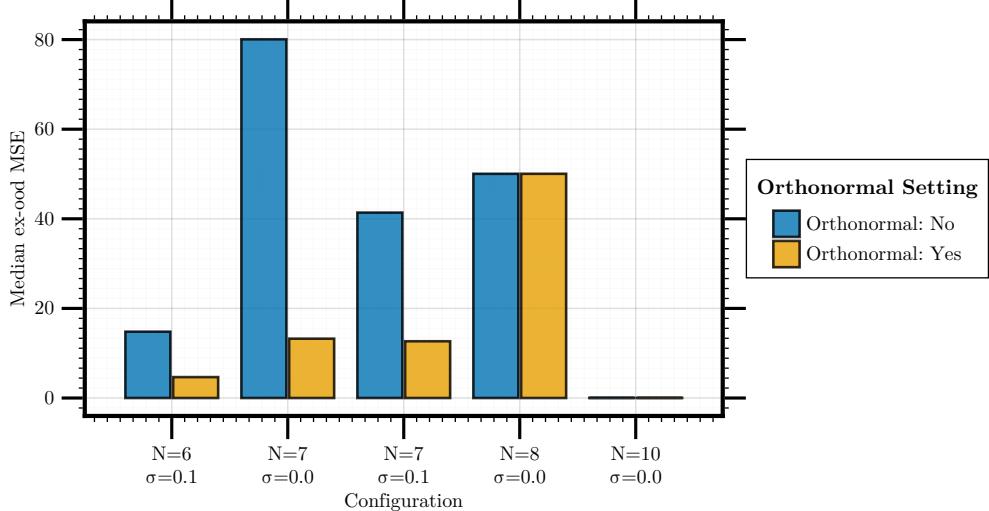


Figure 10: Out-of-distribution performance (*ex-ood*) comparing orthonormal versus non-orthonormal basis construction across noise and data conditions. The orthonormal basis shows consistent improvements, particularly in low-data regimes.

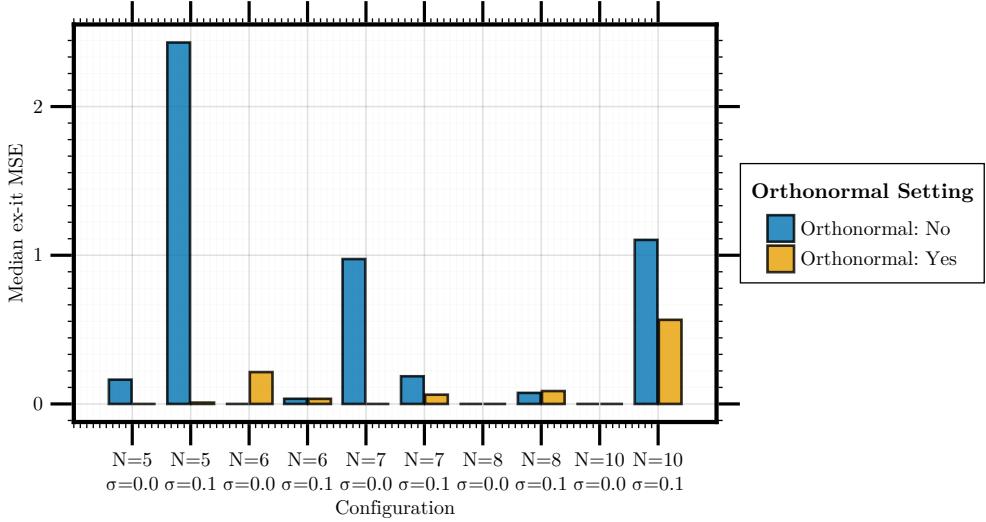


Figure 11: Training performance (*ex-it*) comparing orthonormal versus non-orthonormal basis construction across different noise levels and data quantities. Orthonormal basis construction reduces training error, with largest improvements observed in low-data conditions.

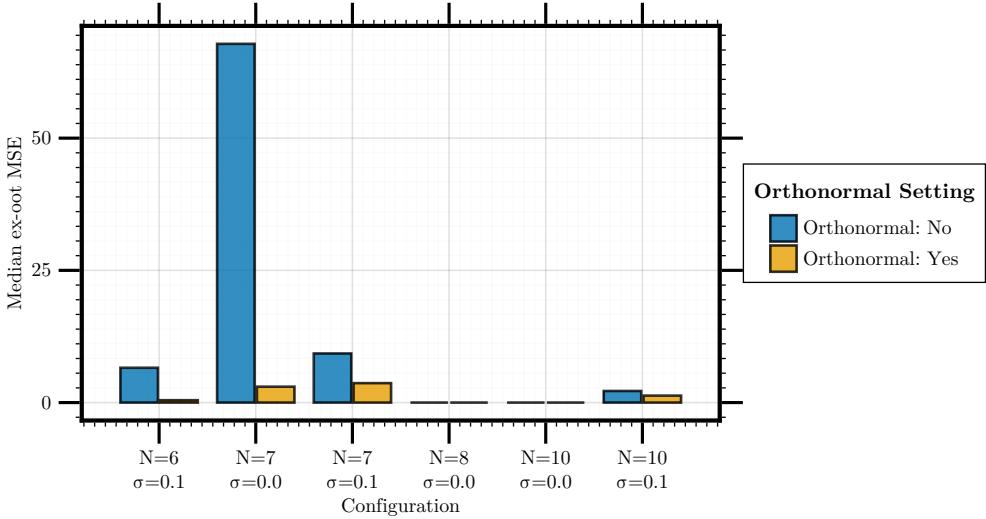


Figure 12: Training performance (*ex-oot*) comparing orthonormal versus non-orthonormal basis construction across different noise levels and data quantities. Orthonormal basis construction reduces training error, with largest improvements observed in low-data conditions.

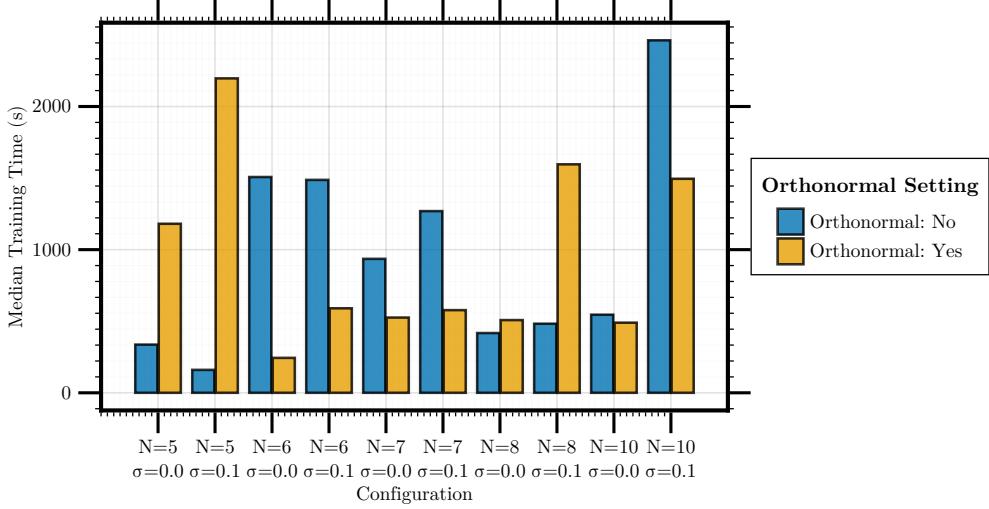


Figure 13: Median training time comparing orthonormal versus non-orthonormal basis construction across different noise levels and data quantities. Orthonormal basis construction often reduces computational time in the low data regime.

4 Conclusion

We introduce the novel ChaosODE (CODE) approach, an orthonormal polynomial chaos-based approximation method to learn arbitrary ODE dynamics from time-series state observations. We demonstrated that selecting problem-tailored ansatz spaces for the right-hand side (RHS) provides substantial benefits in training robustness across varying noise levels and data availability. Through a systematic comparison with two established approaches, NeuralODE and KernelODE, we addressed three foundational questions about structure-aware universal differential equation (UDE) learning:

- **RQ1 (Scarce and noisy data):** How do current UDE learning techniques perform when training data is both scarce and noisy? We found CODE’s superior performance under challenging data conditions, showing robust learning capabilities even with limited and noisy observations.
- **RQ2 (Extrapolation and generalization):** What extrapolation and generalization capabilities do KernelODE and NeuralODE exhibit, and how does CODE’s global approximation structure compare? We showed that CODE’s global orthonormal polynomial approximation structure enables effective extrapolation to unseen initial state values, providing clear advantages over local approximation methods.
- **RQ3 (Orthonormal structure benefits):** Does the orthonormal structure of the dynamics representation facilitate learning in highly non-linear systems? We developed a robust optimization pipeline that addresses numerical instability issues in ODE learning and demonstrated that the orthonormal basis yields more effective learning compared to a non-orthonormal polynomial basis.

This research challenges the common practice of using NeuralODE as a default for autonomous continuous-time UDE learning, advocating instead for architectural decisions informed by problem structure. We highlight that the selection of the RHS basis should align with the system dynamics, as it implicitly introduces inductive

bias. Our results indicate that carefully choosing the approximation space improves learning efficiency and model performance in ODE tasks.

Acknowledgments

We thank the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) for supporting this work by funding – EXC2075 – 390740016 under Germany’s Excellence Strategy and the Collaborative Research Centre SFB 1313, Project Number 327154368. We acknowledge the support by the Stuttgart Center for Simulation Science (SimTech).

Data Availability Statement

Code (in julia) to reproduce all figures and results will be publicly available upon acceptance at <https://github.com/NilsWildt/CODE-Global-ODE-learning>. All findings can be reproduced by running the provided scripts as described in the repository documentation.

References

- Ronny Bergmann. Manopt.jl: Optimization on manifolds in Julia. *Journal of Open Source Software*, 7(70):3866, 2022. doi: 10.21105/joss.03866.
- Mathieu Besançon, Theodore Papamarkou, David Anthoff, Alex Arslan, Simon Byrnes, Dahua Lin, and John Pearson. Distributions.jl: Definition and modeling of probability distributions in the julastats ecosystem. *Journal of Open Source Software*, 6(57):2822, 2021. doi: 10.21105/joss.02822.
- Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017. doi: 10.1137/141000671.
- Milan Bouchet-Valat and Bogumił Kamiński. Dataframes.jl: Flexible and fast tabular data in julia. *Journal of Statistical Software*, 107(4):1–32, 2023. doi: 10.18637/jss.v107.i04.
- Ilja N Bronstein, Konstantin A Semendjajew, Gerhard Musiol, and Heiner Mühlig. Taschenbuch der mathematik. 24. Auflage, Thun u. Frankfurt/Main: Harri Deutsch, 1989.
- Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural Ordinary Differential Equations, December 2019.
- Henry C Croll, Steven Chow, Nadezda Ojeda, Kellogg Schwab, Carsten Prasse, Ryan Capelle, Jamie Klamerus, Joan Oppenheimer, and Joseph G Jacangelo. Adaptation of selected models for describing competitive per-and polyfluoroalkyl substances breakthrough curves in groundwater treated by granular activated carbon. *Journal of Hazardous Materials*, 433:128804, 2022.
- Guillaume Dalle and Adrian Hill. Differentiationinterface.jl. arXiv preprint arXiv:2505.05542, 2025.
- Simon Danisch and Julius Krumbiegel. Makie.jl: Flexible high-performance data visualization for Julia. *Journal of Open Source Software*, 6(65):3349, 2021. doi: 10.21105/joss.03349.
- George Datseris, Jonas Isensee, Sebastian Pech, and Tamás Gál. Drwatson: the perfect sidekick for your scientific inquiries. *Journal of Open Source Software*, 5(54):2673, 2020. doi: 10.21105/joss.02673.
- Edward De Brouwer and Rahul G Krishnan. Anamnesic neural differential equations with orthogonal polynomial projections. *arXiv preprint arXiv:2303.01841*, 2023.
- Jonnie Diegelman and Christopher Rackauckas. Componentarrays.jl: Arrays with named components, 2021.
- Vaibhav Kumar Dixit and Christopher Rackauckas. Optimization.jl: A unified optimization package. *arXiv preprint arXiv:2307.15200*, 2023.
- John E Doherty, Michael N Fienen, and Randall J Hunt. Approaches to highly parameterized inversion: Pilot-point theory, guidelines, and research directions. *Scientific investigations report*, 2010-5168, 2011.

-
- Russell Eberhart and James Kennedy. Particle swarm optimization. In *Proceedings of the IEEE international conference on neural networks*, volume 4, pp. 1942–1948. Perth, Australia, 1995.
- Colby Fronk and Linda Petzold. Interpretable polynomial neural ordinary differential equations. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 33(4):043101, 2023. ISSN 1054-1500, 1089-7682. doi: 10.1063/5.0130803.
- Walter Gautschi. The condition of vandermonde-like matrices involving orthogonal polynomials. *Linear Algebra and its Applications*, 52-53:293–300, 1983. ISSN 0024-3795. doi: [https://doi.org/10.1016/0024-3795\(83\)80020-2](https://doi.org/10.1016/0024-3795(83)80020-2).
- Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.
- Pashupati Hegde, Çağatay Yıldız, Harri Lähdesmäki, Samuel Kaski, and Markus Heinonen. Variational multiple shooting for bayesian odes with gaussian processes. In *Uncertainty in Artificial Intelligence*, pp. 790–799. PMLR, 2022.
- Markus Heinonen, Cagatay Yildiz, Henrik Mannerström, Jukka Intosalmi, and Harri Lähdesmäki. Learning unknown ode models with gaussian processes. In *International Conference on Machine Learning*, pp. 1959–1968. PMLR, 2018.
- Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. Kernel methods in machine learning. *The Annals of Statistics*, 36(3):1171–1220, 2008. doi: 10.1214/009053607000000677.
- Marvin Höge, Thomas Wöhling, and Wolfgang Nowak. A primer for model selection: The decisive role of model complexity. *Water Resources Research*, 54(3):1688–1715, 2018.
- Marvin Höge, Andreas Scheidegger, Marco Baity-Jesi, Carlo Albert, and Fabrizio Fenicia. Improving hydrologic models for predictions and process understanding using neural odes. *Hydrology and Earth System Sciences*, 26(19):5085–5102, October 2022. doi: 10.5194/hess-26-5085-2022.
- John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982.
- Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.
- Michael Innes. Don’t unroll adjoint: Differentiating ssa-form programs. *arXiv preprint arXiv:1810.07951*, 2018.
- Jonas Isensee, Tim Holy, et al. Jld2.jl: Julia data format, 2018.
- Motonobu Kanagawa, Philipp Hennig, Dino Sejdinovic, and Bharath K Sriperumbudur. Gaussian processes and kernel methods: A review on connections and equivalences, 2018.
- Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.
- Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- Yiou Li, Mihai Anitescu, Oleg Roderick, and Fred Hickernell. Orthogonal bases for polynomial regression with derivative information in uncertainty quantification. *Visualization of Mechanical Processes: An International Online Journal*, 1(4), 2011.
- A.J. Lotka. *Elements of physical biology*. Williams & Wilkins, 1925.
- Patrick K. Mogensen and Asbjørn N. Riseth. Optim: A mathematical optimization package for julia. *Journal of Open Source Software*, 3(24):615, 2018. doi: 10.21105/joss.00615.

-
- Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 2006.
- H. Nyquist. Certain topics in telegraph transmission theory. *Transactions of the American Institute of Electrical Engineers*, 47(2):617–644, 1928. doi: 10.1109/T-AIEE.1928.5055024.
- Sergey Oladyshkin and Wolfgang Nowak. Data-driven uncertainty quantification using the arbitrary polynomial chaos expansion. *Reliability Engineering & System Safety*, 106:179–190, 2012.
- Derek Onken and Lars Ruthotto. Discretize-optimize vs. optimize-discretize for time-series regression and continuous normalizing flows. *arXiv preprint arXiv:2005.13420*, 2020.
- Avik Pal and Christopher Rackauckas. Lux.jl: Explicit parameterization of deep neural networks in julia. *arXiv preprint arXiv:2309.16088*, 2023.
- Christopher Rackauckas and Qing Nie. Differentialequations.jl – a performant and feature-rich ecosystem for solving differential equations in julia. *Journal of Open Research Software*, 5(1):15, 2017. doi: 10.5334/jors.151.
- Christopher Rackauckas, Yingbo Ma, Julius Martensen, Collin Warner, Kirill Zubov, Rohit Supekar, Dominic Skinner, Ali Ramadhan, and Alan Edelman. Universal differential equations for scientific machine learning. *arXiv preprint arXiv:2001.04385*, 2020.
- Jarrett Revels, Miles Lubin, and Theodore Papamarkou. Forward-mode automatic differentiation in Julia. *arXiv:1607.07892 [cs.MS]*, 2016.
- Chris Rohlfs. Generalization in neural networks: A broad survey. *Neurocomputing*, 611:128701, 2025. ISSN 0925-2312. doi: 10.1016/j.neucom.2024.128701.
- David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.
- Gabriele Santin and Bernard Haasdonk. Kernel methods for surrogate modeling. In Peter Benner, Stefano Grivet-Talocia, Alfio Quarteroni, Gianluigi Rozza, Wil Schilders, and Luís Miguel Silveira (eds.), *Model Order Reduction*, volume 2. De Gruyter, 2021.
- Facundo Sapienza, Jordi Bolíbar, Frank Schäfer, Brian Groenke, Avik Pal, Victor Boussange, Patrick Heimbach, Giles Hooker, Fernando Pérez, Per-Olof Persson, et al. Differentiable programming for differential equations: A review. *arXiv preprint arXiv:2406.09699*, 2024.
- Robert Schaback and Holger Wendland. Kernel techniques: from machine learning to meshless methods. *Acta numerica*, 15:543–639, 2006.
- Himanshu Sharma, Lukáš Novák, and Michael Shields. Physics-constrained polynomial chaos expansion for scientific machine learning and uncertainty quantification. *Computer Methods in Applied Mechanics and Engineering*, 431:117314, 2024.
- Chaopeng Shen, Alison P Appling, Pierre Gentine, Toshiyuki Bandai, Hoshin Gupta, Alexandre Tartakovsky, Marco Baity-Jesi, Fabrizio Fenicia, Daniel Kifer, Li Li, et al. Differentiable modelling to unify machine learning and physical models for geosciences. *Nature Reviews Earth & Environment*, 4(8):552–567, 2023.
- Timothy John Sullivan. *Introduction to uncertainty quantification*, volume 63. Springer, 2015.
- Albert Tarantola. *Inverse problem theory and methods for model parameter estimation*. SIAM, 2005.
- Michalis Titsias. Variational learning of inducing variables in sparse gaussian processes. In David van Dyk and Max Welling (eds.), *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pp. 567–574, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 16–18 Apr 2009. PMLR.
- Evren Mert Turan and Johannes Jäschke. Multiple shooting for training neural differential equations on time series. *IEEE Control Systems Letters*, 6:1897–1902, 2021.

Rens van de Schoot, Sarah Depaoli, Ruth King, Bianca Kramer, Kaspar Märtens, Mahlet G. Tadesse, Marina Vannucci, Andrew Gelman, Duco Veen, Joukje Willemsen, and Christopher Yau. Bayesian statistics and modelling. *Nature Reviews Methods Primers*, 1(1):1–26, January 2021. ISSN 2662-8449. doi: 10.1038/s43586-020-00001-2.

Vito Volterra. Variazioni e fluttuazioni del numero d'individui in specie animali conviventi. *Memorie della R. Accademia Nazionale dei Lincei, Serie VI*, 2:31–113, 1926.

Wolfgang F Wüst, Ralf Köber, Oliver Schlicker, and Andreas Dahmke. Combined zero-and first-order kinetic model of the degradation of tce and cis-dce with commercial iron. *Environmental science & technology*, 33(23):4304–4309, 1999.

Dongbin Xiu and George Em Karniadakis. The wiener–askey polynomial chaos for stochastic differential equations. *SIAM journal on scientific computing*, 24(2):619–644, 2002.

A Supplementary material S3: Further analysis of the noise cases

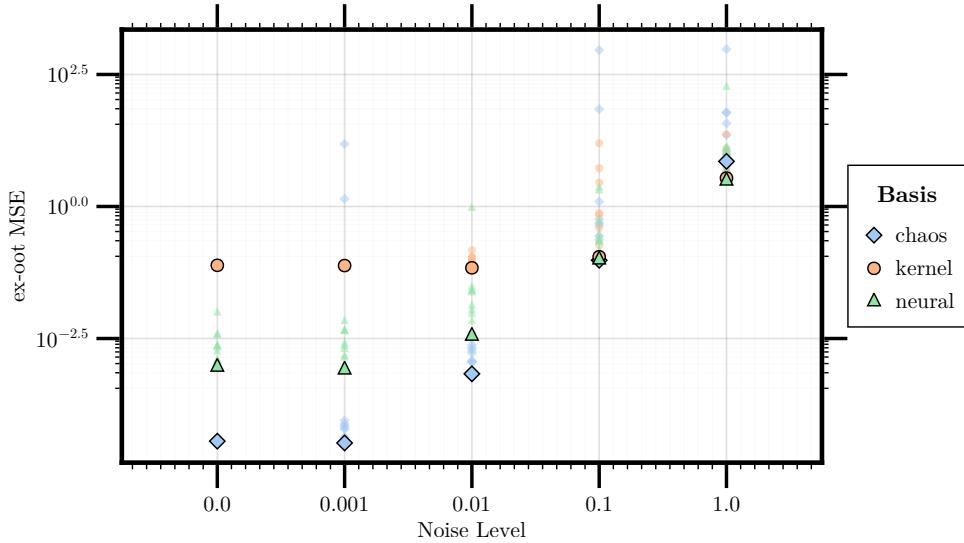


Figure 14: Comparison of extrapolation MSE (*ex-oot*) across different noise levels and methods, trained on 10 data points.

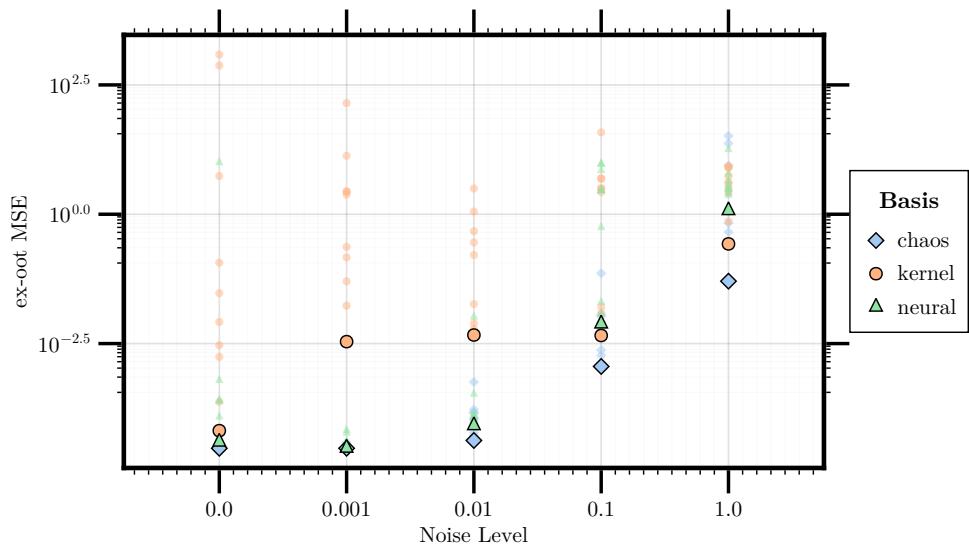


Figure 15: Comparison of extrapolation MSE (*ex-oot*) across different noise levels and methods, trained on 500 data points.

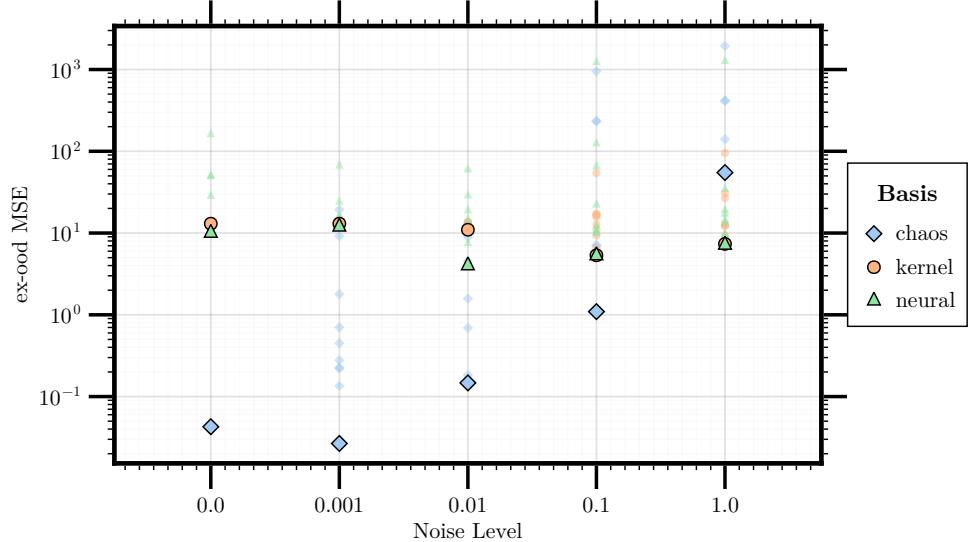


Figure 16: Comparison of out-of-distribution MSE (*ex-ood*) across different noise levels and methods, trained on 10 data points.

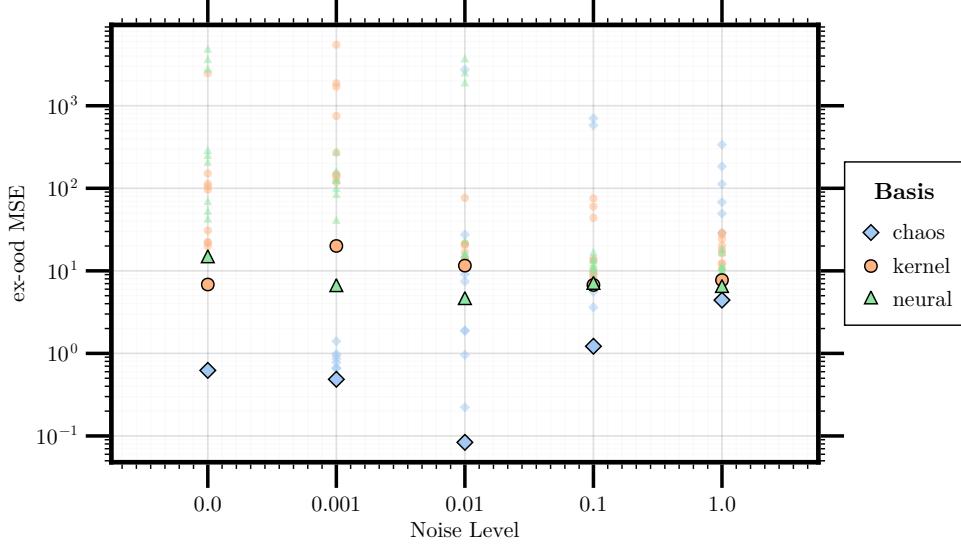


Figure 17: Comparison of out-of-distribution MSE (*ex-ood*) across different noise levels and methods, trained on 500 data points.

B Used Packages

For the implementation we rely on Julia (Bezanson et al., 2017) using the following packages (among others):

- AlgebraOfGraphics.jl for grammar-based data visualization
- ComponentArrays.jl for structured array handling (Diegeman & Rackauckas, 2021)
- DifferentiationInterface.jl as an interface to automatic differentiation backends (Dalle & Hill, 2025)
- DataFrames.jl and DataFramesMeta.jl for tabular data handling (Bouchet-Valat & Kamiński, 2023)
- DifferentialEquations.jl ecosystem (OrdinaryDiffEq, OrdinaryDiffEqTsit5, SciMLBase) for solving differential equations (Rackauckas & Nie, 2017)
- Distributions.jl for probability distributions (Besançon et al., 2021)
- DrWatson.jl for scientific project management (Datseris et al., 2020)
- ForwardDiff.jl for forward-mode automatic differentiation (Revels et al., 2016)
- JLD2.jl for data serialization (Isensee et al., 2018)
- Lux.jl for neural network implementations (Pal & Rackauckas, 2023)
- Makie.jl and CairoMakie.jl for plotting and visualization (Danisch & Krumbiegel, 2021)
- Optimization.jl for optimization algorithms (Dixit & Rackauckas, 2023)
- Zygote.jl for automatic differentiation (Innes, 2018)